Enforcement di sicurezza del sistema

Ticket System

Dicembre 2020

Castaldo	Alessandro	M63000946
Allocca	Roberto	M63001045
Cesarano	Carmine	M63000948

Sommario

1. Specifiche del sistema software

1.1 Introduzione

Nell'ambito di questo elaborato si è deciso di operare l'enforcement di sicurezza dell'applicazione '**Ticket System'**, precedentemente sviluppata dal nostro stesso team, con approccio prototipale, nell'ambito dell'elaborato finale per il corso di *Progettazione e Sviluppo dei Sistemi Software*.

L'applicazione è accessibile tramite un client web browser agli *operatori*, ai *clienti*, e al *manager* e permette ai primi di gestire tutto il ciclo di vita di un ticket di assistenza e all'ultimo di gestire gli utenti del sistema.

1.2 Attori del sistema

Gli utenti, come già precedentemente previsto, sono suddivisi per ruolo:

- **Manager**: può effettuare il login, aggiungere nuovi utenti al sistema e gestirli. In particolare può visualizzare la lista degli utenti registrati, abilitare o disabilitare degli stessi, monitorare i login degli utenti.

- **Cliente**: può effettuare il login, sottomettere un nuovo ticket al sistema, visualizzare quelli da lui precedentemente inseriti e ricevere una notifica al cambio di stato di uno di essi.

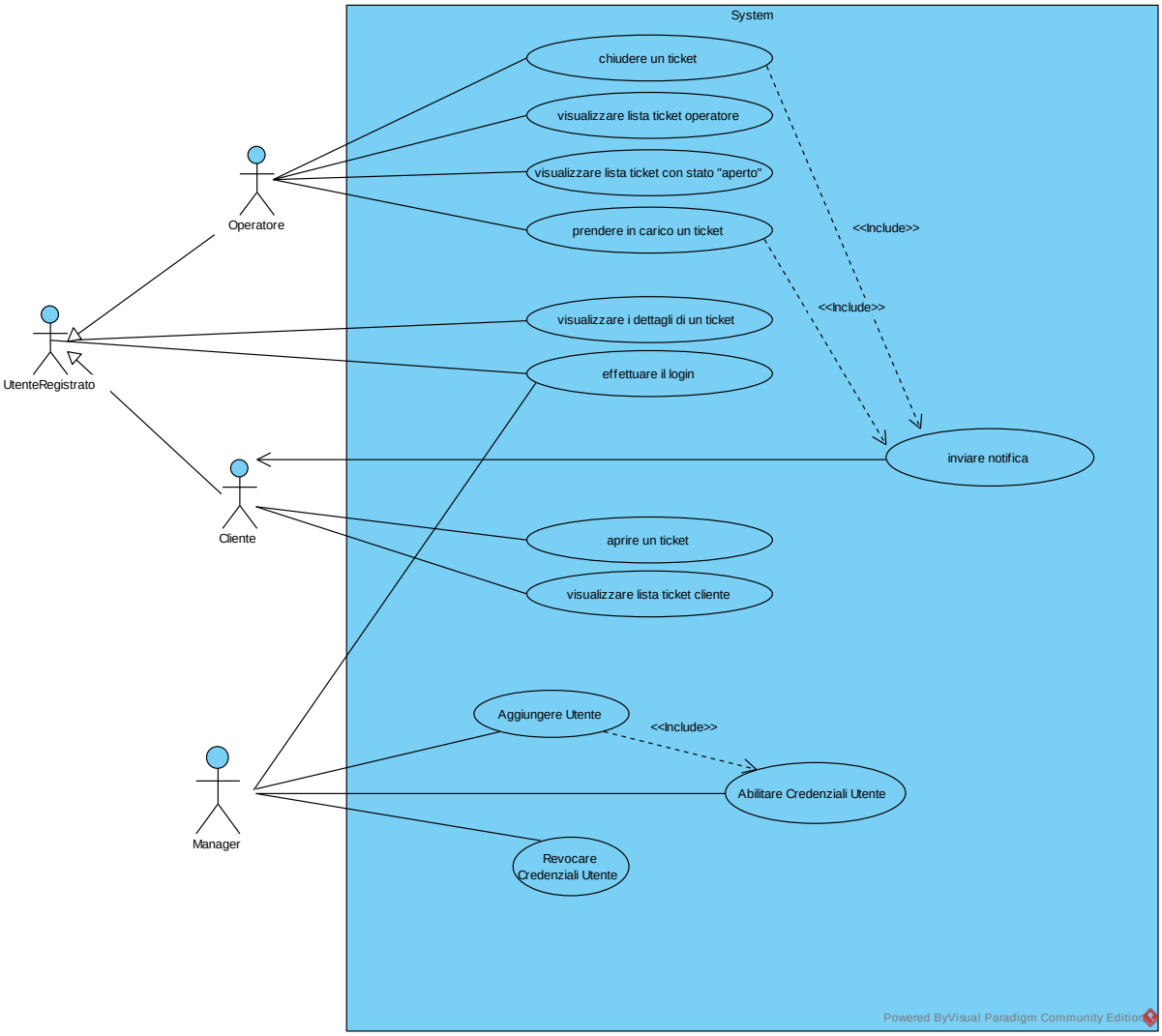
- **Operatori**: può effettuare il login, prendere in carico un ticket ed eventualmente chiuderlo, oltre che visualizzare la lista di tutti i ticket attualmente aperti e quelli da lui presi in carico.

- **Anonimo:** utente non autenticato al sistema, che può visualizzare la sola interfaccia di login.

1.3 Requisti funzionali

Di seguito sono riportati i casi d'uso del sistema pre-esistenti, oltre che quelli relativi al manager, colorati in azzurro, che sono stati aggiunti in questa sede:

Caso d'uso	Attore	Descrizione
Apertura ticket	Cliente	Gestione apertura del ticket
Visualizzazione dettagli	Cliente, Operatore,Manager	Gestione visualizzazione dei dettagli ticket
Presa in carico	Operatore	Gestione presa in carico del ticket e creazione della notifica
Chiusura	Operatore	Gestione chiusura del ticket e creazione della notifica
Visualizzazione lista cliente	Cliente	Visualizzazione lista ticket aperti dal cliente
Visualizzazione lista ticket aperti	Operatore, Manager	Visualizzazione lista di tutti i ticket nello stato aperto
Visualizzazione lista operatore	Operatore	Visualizzazione lista di tutti i ticket presi in carico dall'operatore
Login	Cliente, Manager	Login per clienti e operatori
Aggiunta Utente	Manager	Il manager ha la facoltà di aggiungere un utente
Revoca Utente	Manager	Il manager ha la facoltà di revocare un utente laddove lo ritenga necessario
Abilitazione Utente	Manager	Il manager ha la facoltà di abilitare un utente laddove sia stato disabilitato


System
chiudere un ticket
visualizzare lista ticket operatore
visualizzare lista ticket con stato "aperto"
prendere in carico un ticket
<<Include>>
<<Include>>
visualizzare i dettagli di un ticket
effettuare il login
inviare notifica
aprire un ticket
visualizzare lista ticket cliente
Aggiungere Utente
<<Include>>
Abilitare Credenziali Utente
Revocare Credenziali Utente
Operatore
UtenteRegistrato
Cliente
Manager
Powered ByVisual Paradigm Community Edition

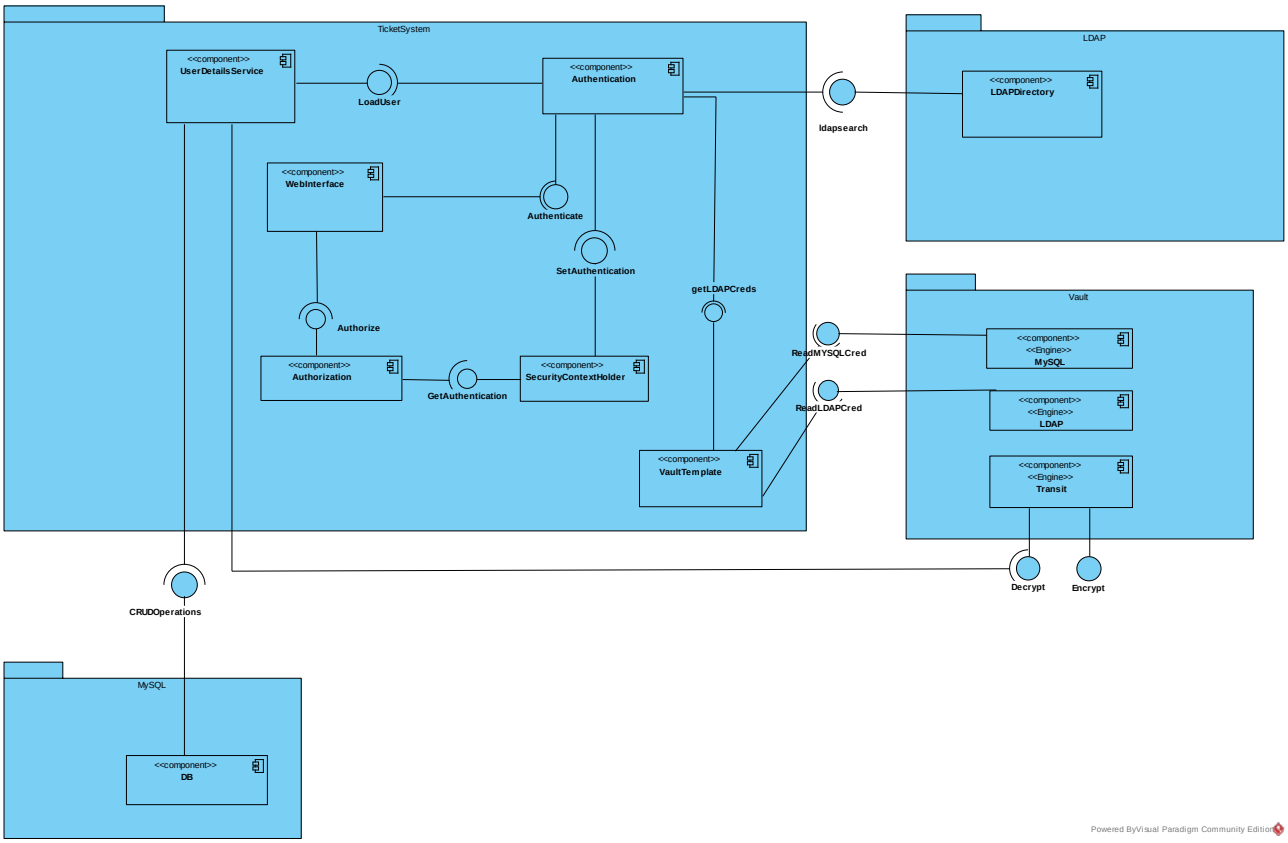
1.4 Requisiti di sicurezza

Di seguito sono riportati i requisiti di sicurezza, funzionali e non, che vengono introdotti nell'ambito dell'enforcement, e che non erano stati previsti nelle fasi di progettazione e sviluppo dell'applicazione.

Autenticazione	L'autenticazione degli utenti al sistema deve avvenire mediante le funzionalità offerte da Spring Security, in comunicazione con un server OpenLDAP, che memorizza all'interno della directory, in maniera sicura, le credenziali degli utenti.
Autorizzazione	L'autorizzazione dei diversi tipi di utenti alle diverse risorse del sistema è operata utilizzando un modello di tipo *RBAC Server-Pull*. A seguito di autenticazione, il web server utilizza la stessa directory LDAP utilizzata per l'autenticazione, come '*Role Server'*, per recuperare le informazioni relative il ruolo dell'utente.
Comunicazione sicura	Tutte le comunicazioni tra client e application server, e tra application server e servizi di back-end, quali MySQL, Vault e il server LDAP, devono garantire confidenzialità ed integrità, oltre che autenticazione. Entrambe le parti di ogni comunicazione necessitano inoltre di essere mutuamente autenticate, salvo per quella stabilita tra web client e application server, la quale non richiede mutua autenticazione, ma soltanto autenticazione del server.
Storage e transit sicuro	Tutti i dati sensibili degli utenti e quelli relativi ai ticket devono essere trasferiti, tra i diversi servizi, e memorizzati all'interno del DB in maniera sicura, mediante l'uso della crittografia.
Account Management System	Deve essere predisposto un meccanismo che permetta ad un manager di sistema di gestire gli account degli utenti registrati. In particolare deve poter visualizzare, aggiungere e revocare utenti, oltre che monitorare i login degli stessi. Inoltre, un account deve essere automaticamente disabilitato dopo aver effettuato tre tentativi di login con credenziali errate.
Session Management	La sessione di autenticazione deve essere gestita opportunamente per evitare problematiche di furto di sessione.
Deployment	I servizi che costituiscono l'architettura devono essere dispiegati su docker container.
Livello di sicurezza NIST	Il sistema, al termine della fase di enforcement, deve assestarsi almeno ad un livello di sicurezza LOW, facendo riferimento allo standard di sicurezza NIST 800-53.

1.5 Component Diagram

Nel seguente diagramma sono mostrate le principali interfacce offerte dai componenti dell'architettura, con particolare riferimento alle funzionalità di autenticazione, autorizzazione e transito sicuro dei dati.

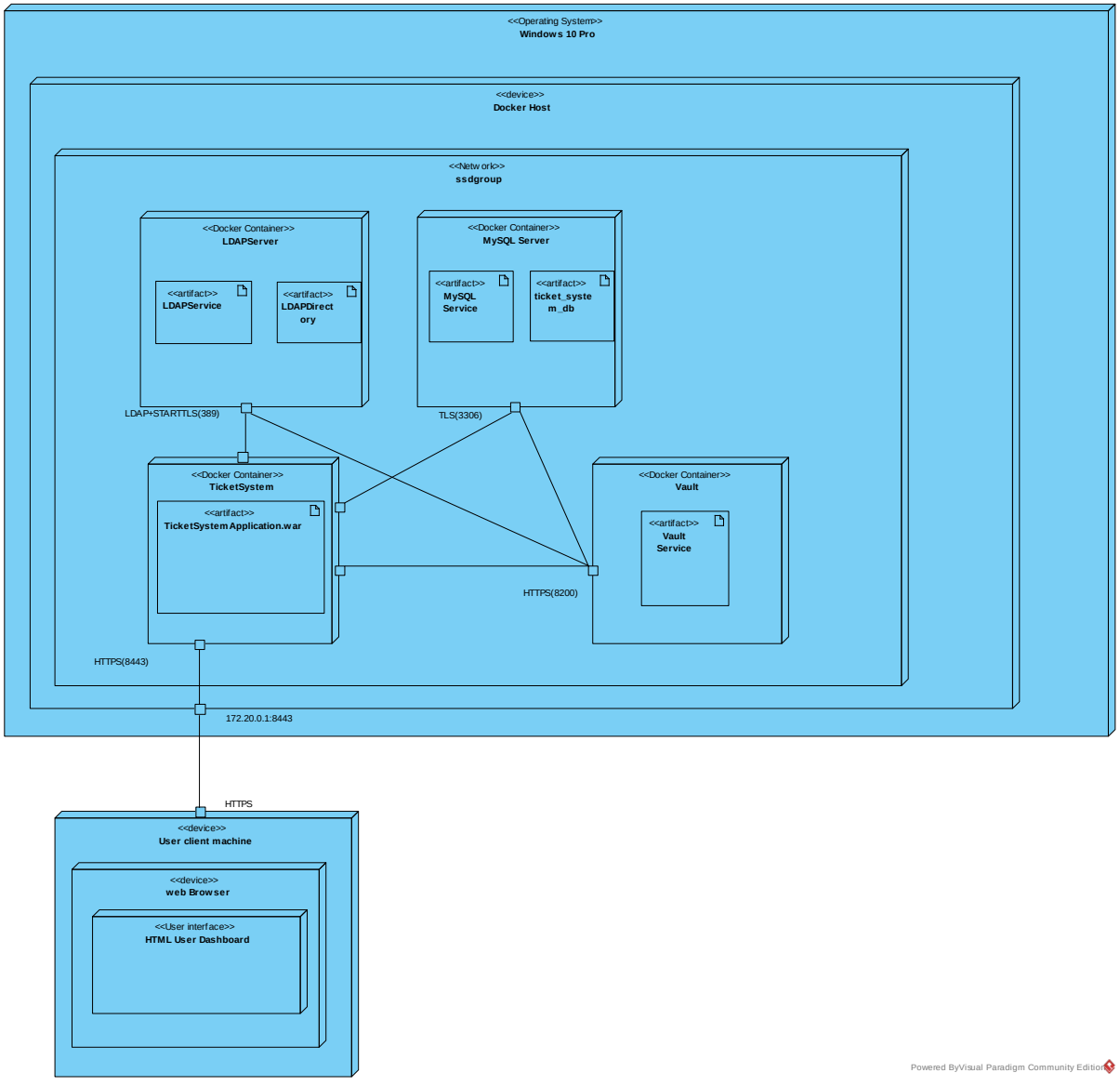


1.6 Deployment diagram

Per quanto riguarda il deployment del sistema, tutti i servizi che fanno parte dell'architettura vengono distribuiti tramite la piattaforma Docker, in dei '***docker container***'. Questo fornisce diversi vantaggi dal punto di vista della sicurezza:

- **Modularità**: i diversi servizi sono dispiegati su nodi diversi. In questo modo è più semplice trovare e risolvere eventuali vulnerabilità, analizzando il servizio specifico, senza dover analizzare l'intera applicazione;

- **Isolamento:** ogni container gode di un isolamento logico dei processi, ma anche di rete visto che conosce soltanto il proprio stack di rete e non può ottenere l'accesso alle risorse degli altri container se queste non sono esplicitamente esposte tramite delle porte pubbliche.
- **Superficie di attacco più piccola**: se tutti i conteiner sono dispiegati su un unico host, dal punto di vista fisico, è necessario mettere in sicurezza l'unico host server, che ospita il docker daemon;
- **Facilità nell'update**: con i container è possibile applicare velocemente patch che risolvano eventuali vulnerabilità su un singolo servizio dell'architettura;
- **Environment parity**: la dockerizzazione permette una automatica esportazione della configurazione dell'ambiente tra il contesto di sviluppo e quello di produzione. Dal punto di vista della sicurezza è importante per rendere indipendenti le configurazioni dall'ambiente host.



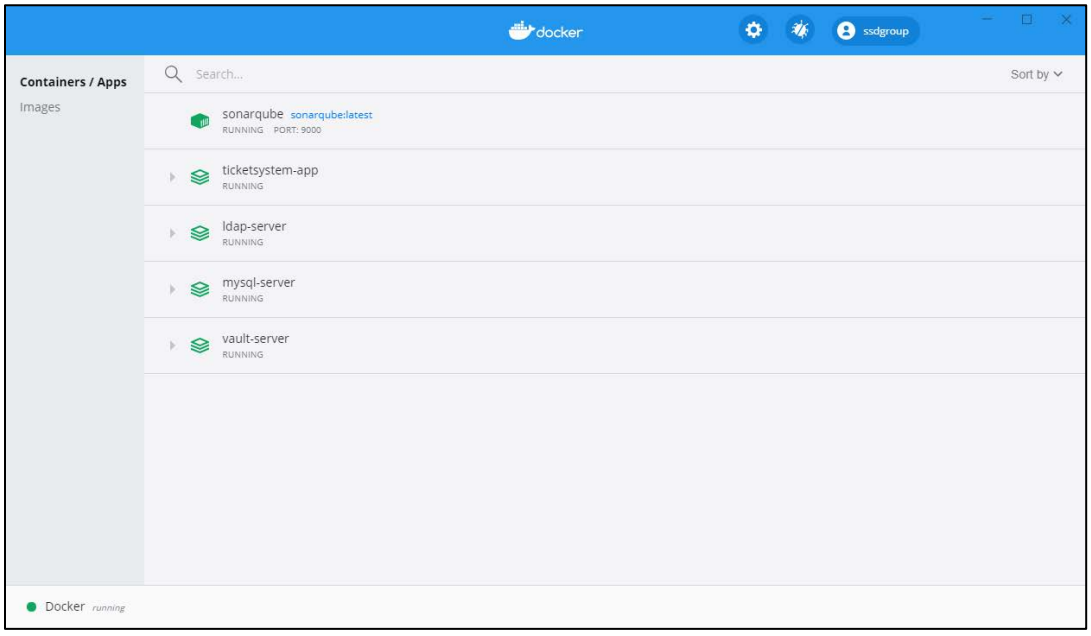
Come mostrato dal diagramma di deployment, l'architettura complessiva consta dei seguenti servizi:

- **Application Server**: esegue l'applicazione Spring Boot;
- **MySQL Server**: utilizzato per la persistenza dei dati;
- **LDAP Server**: utilizzato a supporto dei meccanismi di autenticazione e autorizzazione;
- **Vault Server**: utilizzato come servizio di gestione sicura dell'accesso ai diversi servizi da parte dell'application server, oltre che come '*Encryption as a Service*'.

Ognuno di questi docker container si interfaccia con una bridge network software, virtualizzata da Docker localmente, attraverso la quale tutti i servizi possono comunicare tra di loro, come se fossero sulla stessa rete locale. Ognuno dei servizi collegati alla rete deve esporre un porto sul quale restare in ascolto:

Port	Container
80	Spring Application (redirect su 8443)
8443	Spring Application
389	LDAP Server
3306	MySQL Server
8200	Vault Server

Le porte di cui sopra sono private e non esposte pubblicamente, eccetto che non sia stata prevista una esplicita mappatura, come nel caso delle porte 80 e 8443, che devono essere accessibili dalla rete esterna attraverso l'IP '*172.20.0.1*' che funge da gateway. Di seguito sono mostrati i container in esecuzione:



2. Security Enforcement

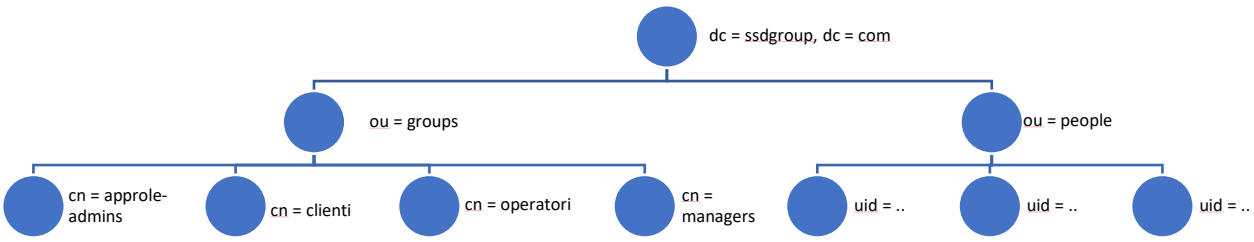
In questa sezione vengono analizzate tutte le strategie adottate per operare l'enforcement di sicurezza e garantire, quindi, i requisiti di sicurezza menzionati in precedenza.

2.1 Autenticazione e autorizzazione delle risorse Web

2.1.1 OpenLDAP

A supporto dei meccanismi di autenticazione e autorizzazione viene utilizzato OpenLDAP, un' implementazione open source del protocollo **Lightweight Directory Access Protocol**, utilizzato per la gestione e l'interrogazione di *directory*.

Le directory, in particolare, sono insiemi di dati organizzati in un albero secondo una struttura gerarchica, come quella che segue:



Le entry di una directory sono definite come oggetti ottenuti per composizione di oggetti più semplici che includono via via il loro predecessore, con tutti gli attributi ad esso associati. Ogni nodo dell'albero definisce una entry della directory, ed è identificato da un proprio *Distinguished Name* (DN), composto, ad esempio, come segue:

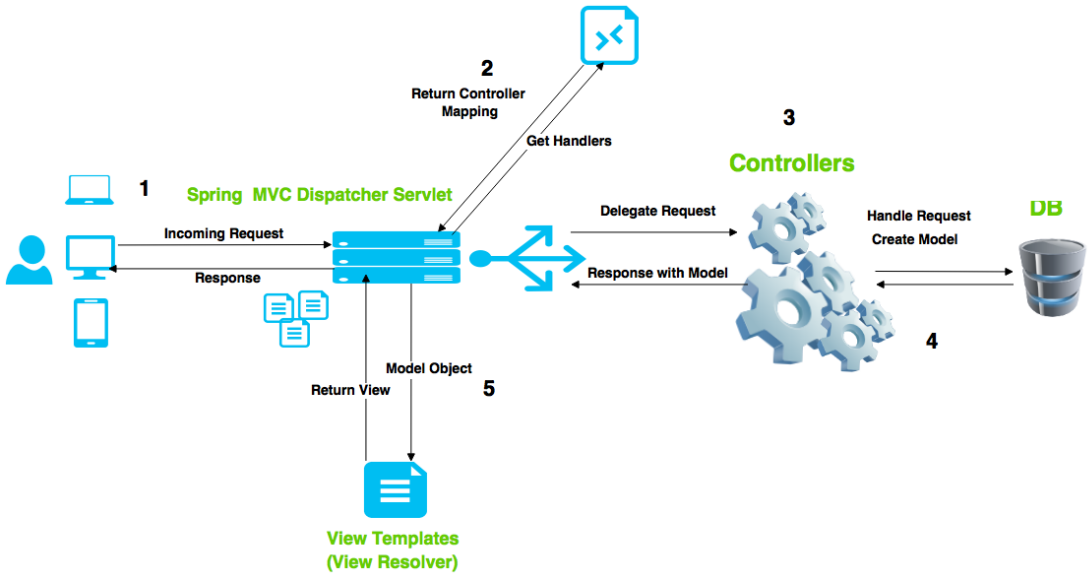
dn = *[cn = clienti, ou = groups, dc = ssdgroup,dc = com]*

Se da un lato, con una operazione di '*ldapsearch*' è possibile leggere le credenziali di un utente dalla directory, e quindi implementare **meccanismi di autenticazione**, dall'altro, costruendo un opportuno albero che organizza gli utenti in rami diversi a seconda del loro ruolo, è possibile poi, tramite logiche applicative, implementare un opportuno **meccanismo di autorizzazione** basato sui ruoli.

2.1.2 Overview Spring Framework

Per sviluppare lo strato di back-end della web-application è stato utilizzato *Spring*, un framework open source per lo sviluppo di applicazioni su piattaforma Java.

Il framework Spring permette lo sviluppo di applicazioni che seguono un pattern architetturale MVC. L'implementazione dell'MVC nelle applicazioni Web Spring prevede per lo strato Controller l'utilizzo di un pattern denominato "**Front Controller**".



Tale pattern prevede un unico ingresso per le chiamate HTTP rappresentato da una servlet, nella fattispecie chiamato **Dispatcher Servlet**. Il suo compito, dunque, è quello di prendere un URI in entrata e trovare il preciso gestore di tale richiesta (metodi sulle classi Controller) tramite l'**Handler Mapping**, un componente specializzato nell'interpretare le richieste e decidere a quale controller devono essere inoltrate. Il controller, dopo avere eseguito la sua logica applicativa, ne restituisce i risultati al Dispatcher, che attraverso l'oggetto **View Resolver** fornisce la specifica View al client.

2.1.3 Gestione delle risorse del sistema

Tutte le risorse messe a disposizione dal sistema ai suoi utenti, dunque, sono mappate e raggiungibili solo attraverso specifiche richieste HTTP, i cui URL sono strutturati opportunamente, per cui sarà necessario in seguito agire su queste per implementare il **meccanismo di autorizzazione**.

Di seguito sono riportate tutte le richieste HTTP messe a disposizione dal sistema:

Richiesta	**URL**	**Ruolo autorizzato**	**Descrizione**
GET	/dashboard/**	ROLE_CLIENTI ROLE_OPERATORI	Carica l'oggetto Account e la lista delle notifiche e reindirizza alla home page dell'utente loggato.
GET	/ticket/send	ROLE_CLIENTI	Reindirizza alla pagina di "inserimento nuovo ticket"
POST	/ticket/send	ROLE_CLIENTI	Provvede alla creazione e allo storage del nuovo oggetto Ticket.
GET	/ticket/history_aperti	ROLE_OPERATORI	Carica la lista di tutti i ticket aperti presenti nel sistema e reindirizza alla relativa pagina di visualizzazione.
GET	/ticket/history_cliente	ROLE_CLIENTI	Carica la lista dei ticket aperti dal cliente loggato e reindirizza alla relativa pagina di visualizzazione.
GET	/ticket/history_operatore	ROLE_OPERATORI	Carica la lista dei ticket presi in carico dall'operatore loggato e reindirizza alla relativa pagina di visualizzazione.
GET	/ticket/details/*	ROLE_CLIENTI ROLE_OPERATORI	Carica l'oggetto ticket avente l'id passato nella richiesta e reindirizza alla relativa pagina di dettaglio.
POST	ticket/details/aggiorna_stato/*	ROLE_OPERATORI	Permette il cambio di stato del ticket e il relativo salvataggio oltre che la creazione e il salvataggio della notifica e il suo invio al cliente.

GET	/login-panel/index	PERMIT ALL	Reindirizza al pannello di login.
GET	/login-panel/login?error	PERMIT ALL	Reindirizza al pannello di login con un messaggio di errore nel caso di credenziali errate.
POST	/login-panel/login?logout	ROLE_CLIENTI ROLE_OPERATORI	Reindirizza al pannello di login con un opportuno messaggio a seguito del logout.
GET	/login-panel/accessDenied	ROLE_CLIENTI ROLE_OPERATORI	Reindirizza ad una pagina di errore nel caso in cui l'utente loggato non abbia i permessi per effettuare una determinata richiesta.
GET	/login-panel/welcome	ROLE_CLIENTI ROLE_OPERATORI	Reindirizza alla dashboard a seguito del login.
GET	/resources/public/**	PERMIT ALL	Recupera le risorse statiche per caricare il template grafico.
GET	/resources/private/**	DENY ALL	Recupera le risorse private (keystore, configuration file).
POST	/user/queue/notify	ROLE_OPERATORI	Invia una notifica sulla coda della socket '/ws' all'atto del cambiamento di stato di un ticket.
GET	/ws/**	ROLE_OPERATORI ROLE_CLIENTI	Permette di collegarsi alla socket '/ws'.
GET	/user/list	ROLE_MANAGER	Carica la lista di tutti gli utenti registrati al sistema.
	/user/profile/*	ROLE_MANAGER	Carica il dettaglio di un profilo utente.
GET	/user/add	ROLE_MANAGER	Reindirizza alla pagina di "aggiunta di un utente".
POST	/user/add	ROLE_MANAGER	Provvede all'inserimento di nuovo Utente.
POST	/user/cambiaStato/*	ROLE_MANAGER	Permette di abilitare o disabilitare un utente.
	other requests	DENY ALL	

2.1.4 Overview di Spring Security

Ad alto livello, gli step base che devono essere effettuati per il **processo di autenticazione e autorizzazione**, sono i seguenti:

1. Autenticare le credenziali fornite dal client con il supporto della directory LDAP;
2. Associare delle *authorities* all'utente autenticato, basate sulle informazioni riguardanti il ruolo, recuperate dalla directory LDAP;
3. Attuare l'access control garantendo o meno l'accesso degli utenti alle risorse richieste, sulla base delle policy costruite come 'risorsa-ruolo-permessi'.

Per effettuare autenticazione e autorizzazione vengono sfruttati ampiamente i meccanismi messi a disposizione dal framework Spring, in particolare con il modulo Spring Security. Il cuore del modello di autenticazione di Spring è il <*SecurityContexHolder*>**:**

Questo oggetto contiene il <*SecurityContext*> che a sua volta include un oggetto di tipo <*Authentication*>. Quest'ultimo rappresenta l'oggetto nel quale Spring conserva le informazioni dell'utente autenticato attualmente in sessione. Quindi in ogni punto della logica di business si può far riferimento ad esso per recuperare, in particolare le seguenti informazioni:

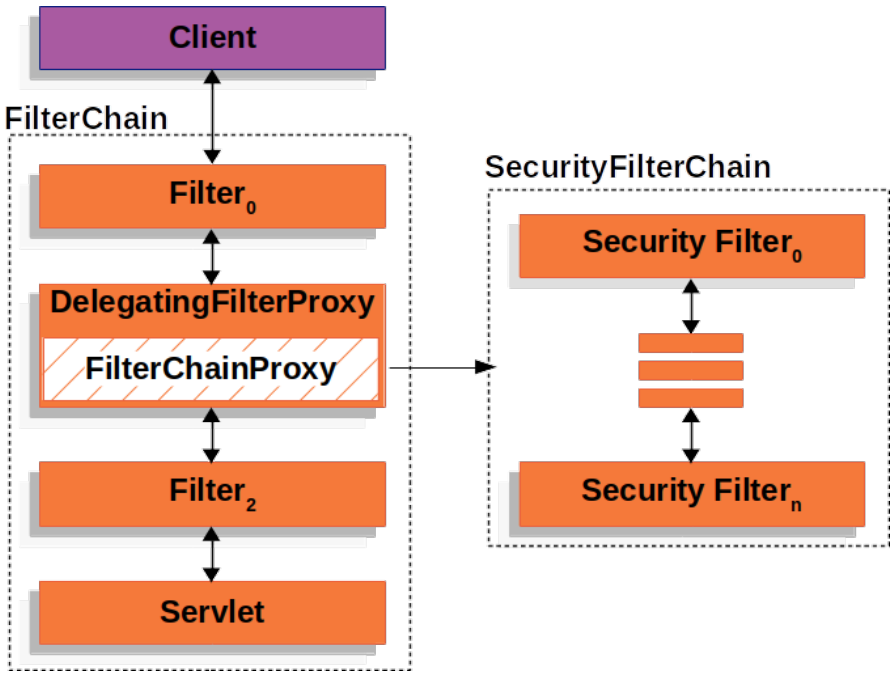
- **Principal**: identifica l'utente e ne mantiene le informazioni principali;
- **Credentials:** nel caso specifico corrispondono ad una password
- **Authorities:** un oggetto che rappresenta il ruolo dell'utente autenticato e sul quale è possibile definire opportune policy di accesso alle risorse.

Di default, Spring Security, dopo la fase di autenticazione, provvede al reset dell'oggetto <*Credentials*> all'interno di un <*Authentication*> per evitare che le credenziali utente restino fruibili durante la sessione HTTP.

2.1.5 Autenticazione con Spring Security e LDAP

Spring Security, nel livello web, è basato sui **Servlet Filter**. Quando un client sottopone una richiesta HTTP viene deciso, in base al percorso URI, quali filtri e quali servlet, tra quelli disponibili. Una servlet può gestire un'unica richiesta, ma una richiesta può passare attraverso una catena ordinata di filtri.

Di default, Spring Security vede un singolo filtro, ma ne possono essere aggiunti altri custom. In particolare il singolo Filter può essere implementato da un <*FilterProxy*> che contiene tutta la logica di sicurezza e quindi permette, in primo luogo, di intercettare un certo di tipo di richieste e poi di smistare le stesse ai diversi filtri della propria catena interna.



Questo, a livello di codifica, viene effettuato, ad esempio, con la seguente chiamata:

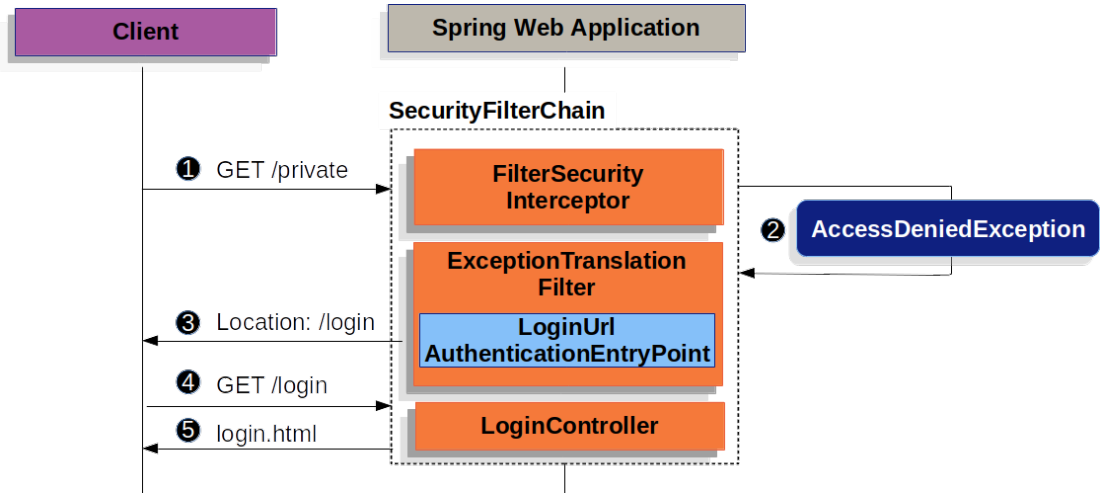
```
<mvcMatchers("/dashboard").authenticated()>
```

la quale consente di intercettare le richieste HTTP con l'URI "/dashboard" ed applicare il filtro che verifica l'autenticazione dell'utente. Quindi, come mostrato dall'esempio, tutti i filtri messi a disposizione dal framework possono essere di supporto all'implementazione di meccanismi **meccanismi di autenticazione** e **autorizzazione**.

I filtri più importanti da menzionare, inserti di default nella **filter chain**, sono i seguenti:

- **IntegrationFilter**: responsabile di recuperare un'eventuale autenticazione precedentemente salvata nella sessione HTTP.
- **FilterSecurityInterceptor:** esamina la richiesta e determina se l'utente dispone dei privilegi necessari per accedere alla risorsa.
- **ExceptionTranslationFilter:** traduce le eccezioni per gestirle ed effettuare redirect.
- **AbstractAuthenticationProcessingFilter**: determina se una richiesta ricevuta da un client HTTP è una richiesta di autenticazione.

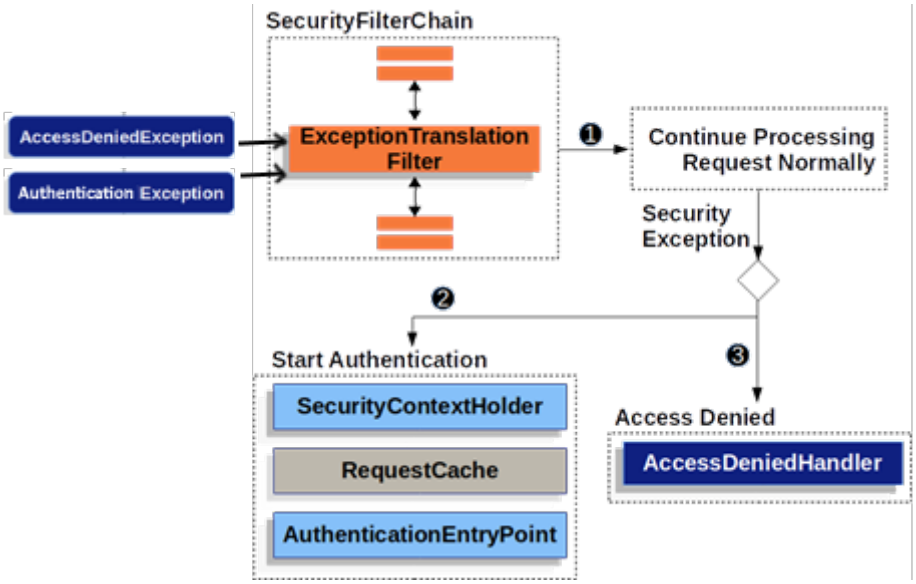
Per illustrare il funzionamento dei filtri menzionati, nell'ambito dell'autenticazione e dell'autorizzazione, di seguito è mostrato il flusso innescato da una richiesta per una risorsa da parte di un utente non autenticato.



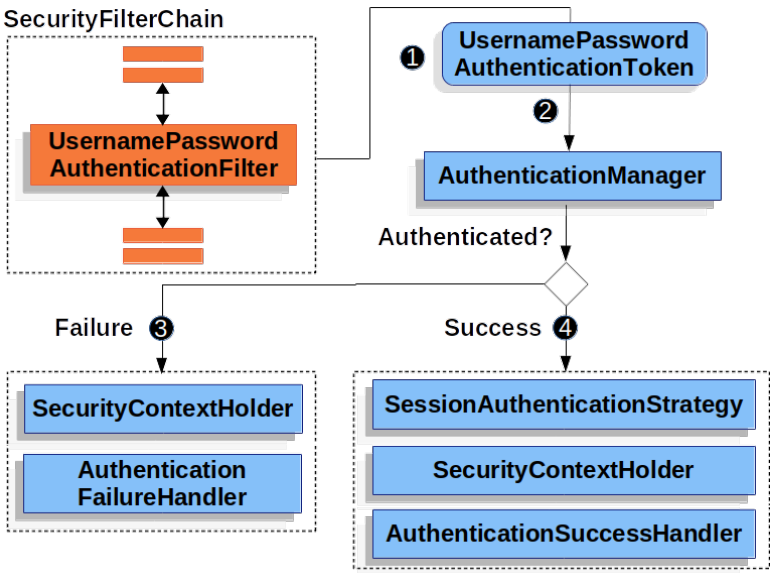
Una generica richiesta HTTP, inviata dal client, prima di essere gestita da una specifica servlet, deve essere intercettata a monte da un filtro '***FilterSecurityInterceptor'***. Se la richiesta richiedeva autenticazione prima di essere servita, questo filtro la contrassegna come *denied* alzando un' eccezione '*AccessDeniedException*'.

Il filtro **ExceptionTranslationFilter** permette la traduzione delle eccezioni '*AccessDeniedException*' e '*AuthenticationException'* in risposte HTTP. Quindi, per la prima delle due, fa partire il meccanismo di autenticazione, facendo il redirect all' <*AuthenticationEntryPoint*>, per la seconda invoca l'handler '*AccessDeniedHandler'*.

Dal momento che l'utente non è autenticato, l'*ExceptionTranslationFilter* avvia il processo di autenticazione e risponde al client con un redirect alla pagina di login come configurato nell' <*AuthenticationEntryPoint*>.

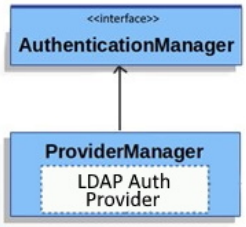


L'utente a questo punto, tramite la pagina di login, può sottomettere al sistema le credenziali mediante un' ulteriore richiesta HTTP. Nel caso specifico, nel quale si fa uso di username e password come credenziali, la richiesta HTTP di autenticazione viene intercettata dal filtro **UsernamePasswordAuthenticationFilter**, il quale estende *AbstractAuthenticationProcessingFilter*.



Questo filtro, a partire dalla *HttpServletRequest* di autenticazione, genera uno <*UsernamePasswordAuthenticationToken*>, una specializzazione della classe <*Authentication*>. Questo oggetto è passato successivamente ad uno <**AuthenticationManager**> per essere autenticato. Se l'autenticazione ha successo, l'oggetto <Authentication> viene settato nel <*SecurityContextHolder*> e l'<*AuthenticationSuccessHandler*> viene invocato.

Lo specifico flusso di autenticazione delle credenziali dipende dal tipo di **AuthenticationManager** utilizzato. Infatti, quest'ultimo rappresenta soltanto un'interfaccia che può essere implementata da un qualsiasi <*ProviderManager*> che, a sua volta, può contenere uno o più <*AuthenticationProvider*>. Nel caso specifico verrà utilizzato un <***LdapAuthenticationProvider***>.



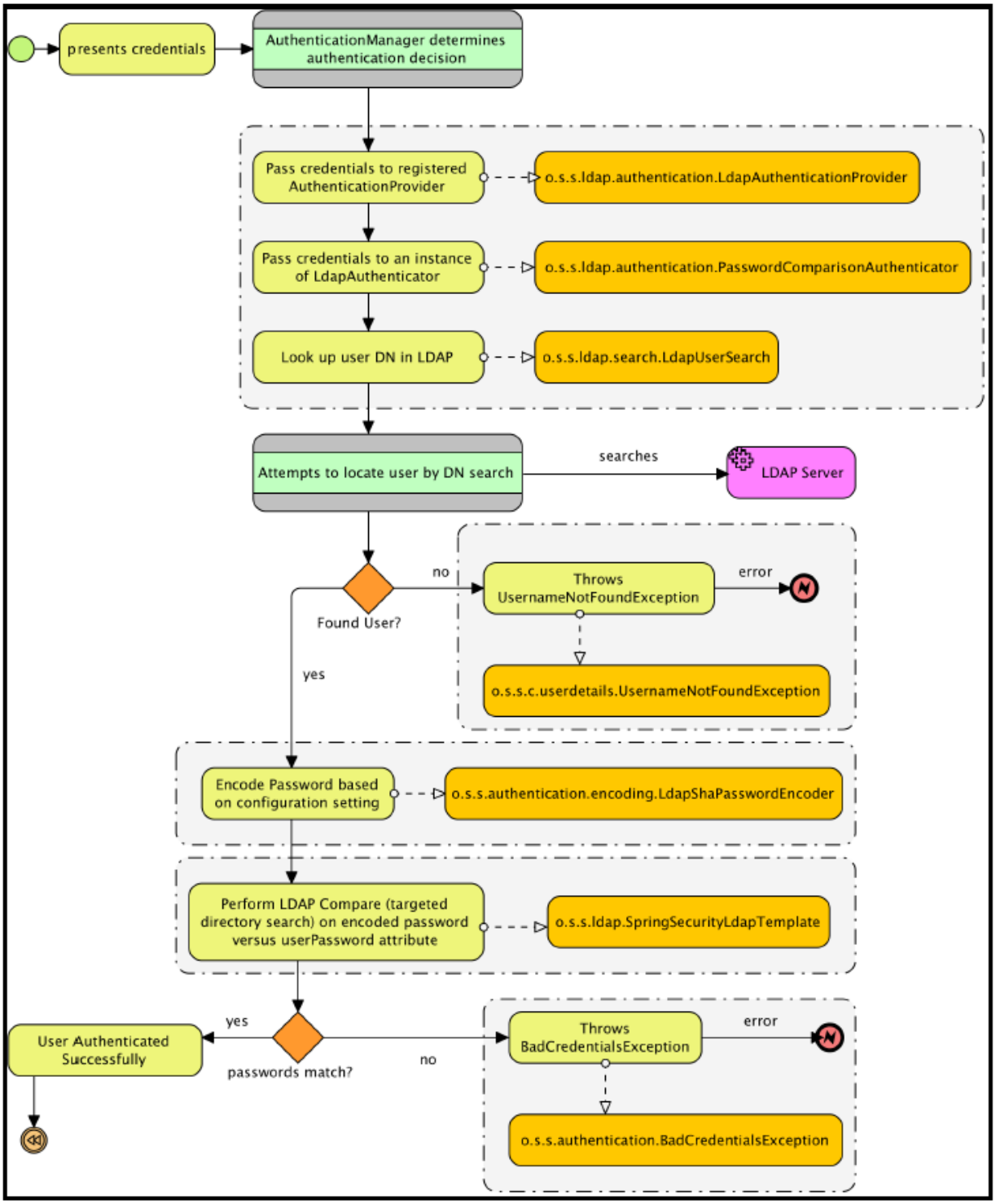
La responsabilità di configurarazione del sistema di autenticazione e delle operazioni di autenticazione stesse, sono demandate, a livello di codifica, ai seguenti due metodi:

```
@Override
public void init(AuthenticationManagerBuilder auth) throws Exception {

    auth.ldapAuthentication()
        .userDnPatterns(userDnPattern)
        .groupSearchBase("ou=groups")
        .contextSource(contextSource())
        .passwordCompare()
        .passwordEncoder(new BCryptPasswordEncoder())
        .passwordAttribute("userPassword");
}
```

```
@Bean
public LdapContextSource contextSource() {

    VaultResponse response = vaultTemplate.read("/openldap/static-cred/ssdgroup");

    LdapContextSource contextSource = new LdapContextSource();
    contextSource.setUrl(ldapUrl);
    contextSource.setBase(ldapBaseDn);
    contextSource.setUserDn(response.getData().get("dn").toString());
    contextSource.setPassword(response.getData().get("password").toString());
    contextSource.setAuthenticationStrategy(new DefaultTlsDirContextAuthenticationStrategy());

    return contextSource;
}
```

Prima di tutto, è necessaria una fase preliminare di autenticazione ('**bind**' su LDAP) dell'utente su LDAP che l'Application Server, utilizzerà per effettuare tutte le operazioni sulla directory. Le credenziali associate a questo utente vengono recuperate da Vault, facendo uso dei metodi esposti dal Vault Template (aspetto approfondito in seguito).

L'applicazione, dopo essersi autenticata ad LDAP può, a sua volta, tramite l'*<LDAPAuthenticationProvider>*, autenticare le credenziali degli end-user, interpellando la directory LDAP con operazioni di '**search**'. Di seguito il flusso dell'autenticazione nel dettaglio:

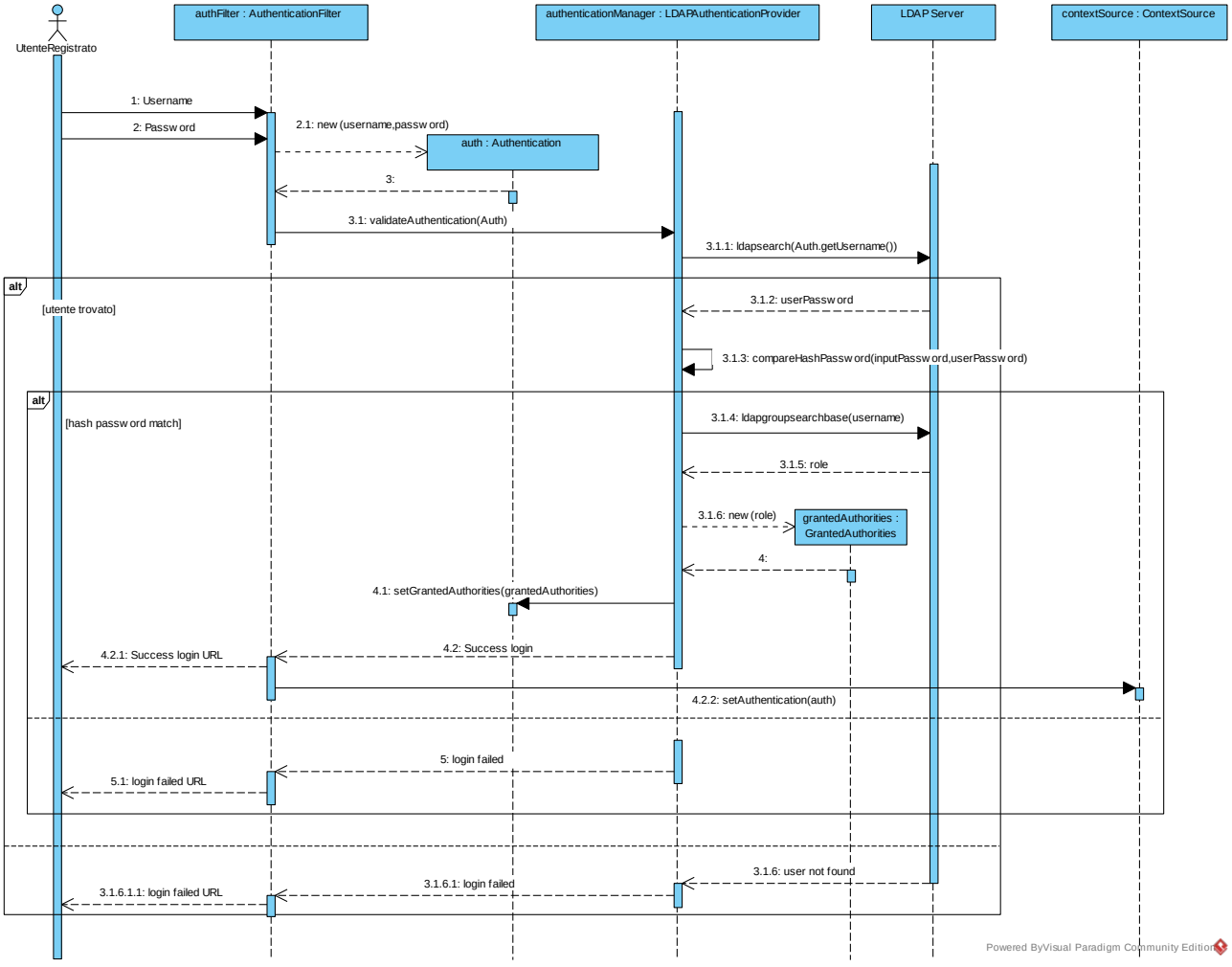


All'atto di una richiesta di autenticazione, con una operazione di '*LDAPsearch*', viene recuperata dalla directory LDAP l'entry il cui '*uid*' corrisponde all'username fornito dall'utente, se esiste.

Nel caso specifico è stato utilizzato un meccanismo di tipo **ServerPull-based** perchè si è scelto di salvare le password nel server LDAP in formato 'salted hash', tramite l'algoritmo **BCrypt**, e quindi non è possibile comparare direttamente le password sul server LDAP. Data questa scelta, la vera e propria autenticazione, quindi la comparazione della password che è contenuta nell'entry, con quella fornita dall'utente, deve essere fatta localmente sull'application server.

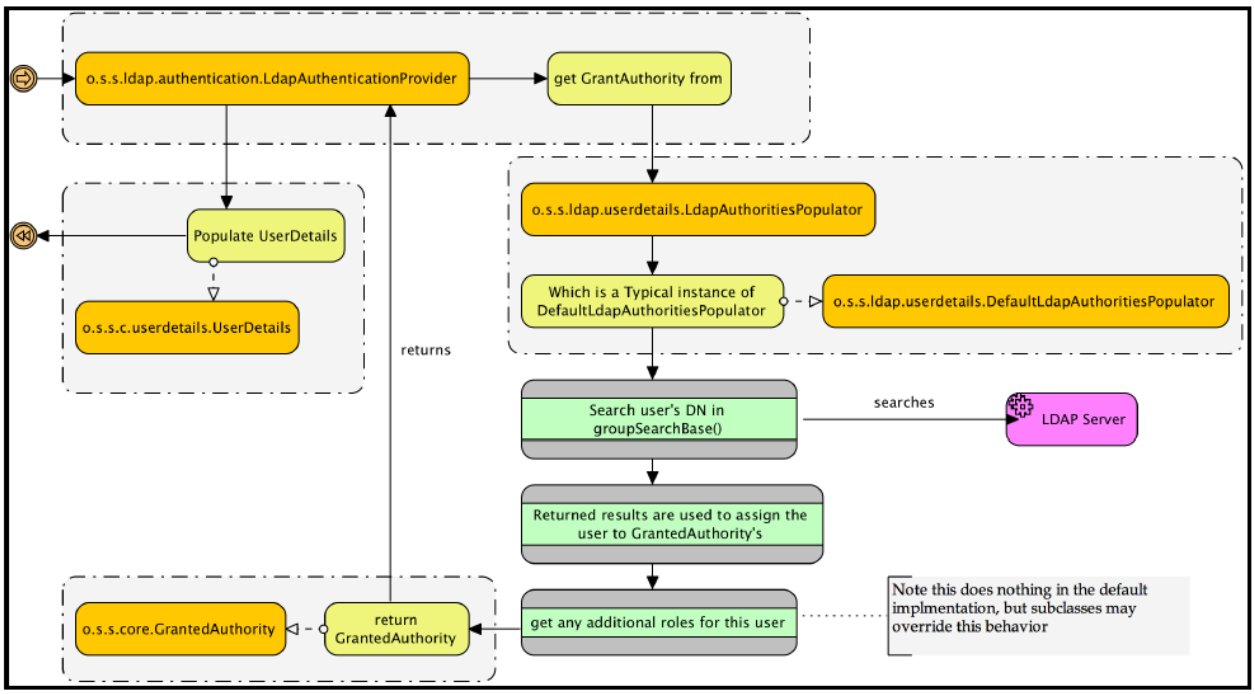
Se le password corrispondono l'utente viene autenticato correttamente, altrimenti viene generata una '*AuthenticationException*', ricadendo negli scenari visti in precedenza.

Di seguito è riportato il sequence diagram relativo allo scenario di autenticazione che presuppone, come pre-condizione, che l'application server abbia già richiesto a Vault le credenziali di autenticazione al server LDAP:



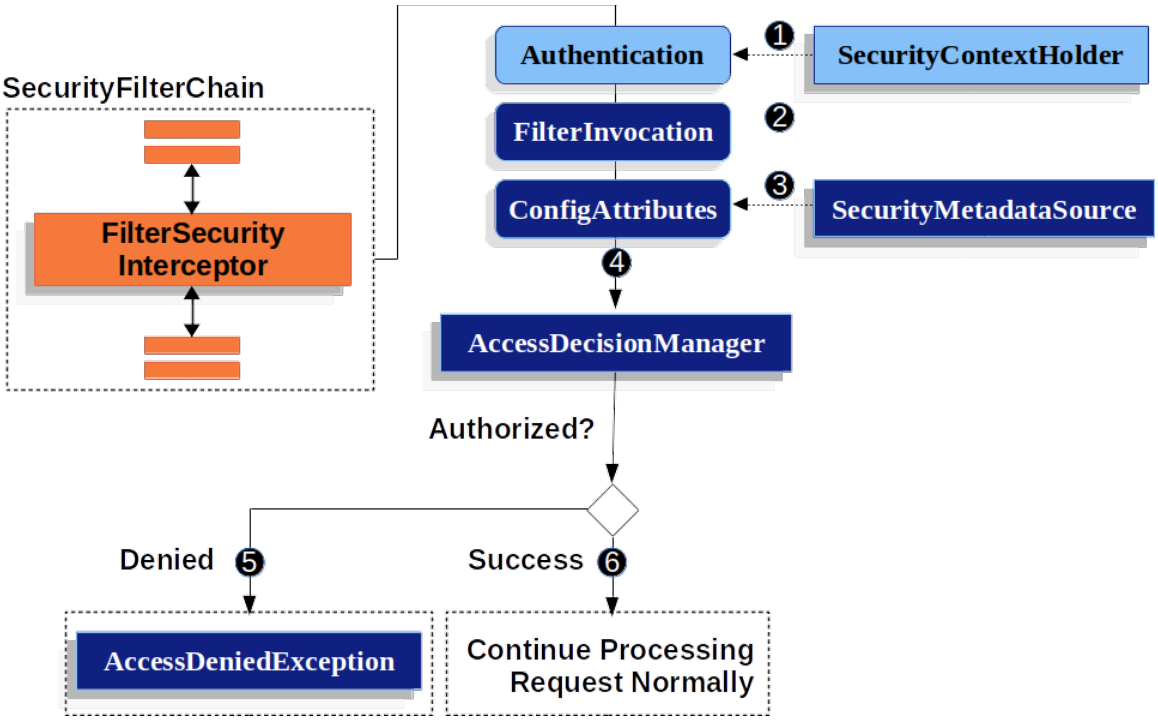
2.1.6 Autorizzazione con Spring Security e LDAP

Dopo che l'utente è stato autenticato con successo, l'informazione di appartenza ad un determinato gruppo (inteso come ruolo) è recuperata attraverso una nuova interrogazione al server LDAP, che permette la creazione di un oggetto <*GrantedAuthorities*>, che diviene parte dell'oggetto <*Authentication*>.

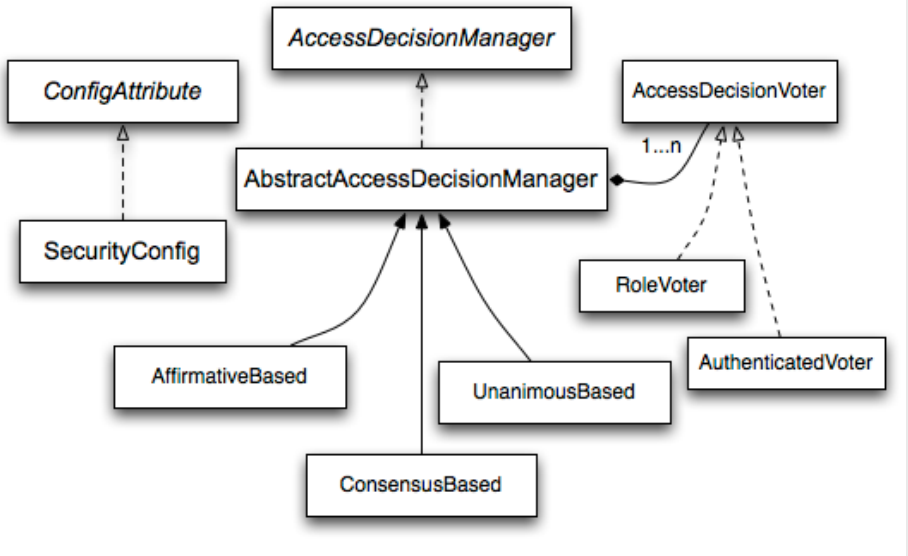


Per quanta riguarda il vero e proprio **meccanismo di autorizzazione**, Spring Security implementa nativamente un access control mechanism basato su Policy Enforcement Point (PEP) per operare l'enforcement dell'autorizzazione sulle richieste HTTP.

Il PEP nel framework è rappresentato da un filtro **FilterSecurityInterceptor**, che intercetta tutte le richieste HTTP che riguardano una risorsa protetta e ne controlla l'accesso, determinando se il client che l'ha inoltrata dispone dei privilegi necessari per accedere alla risorsa richiesta.



In particolare, il filtro ottiene l'oggetto <*Authentication*> dal <*ContextSource*>, oltre che un oggetto <ConfigAttributes>. Entrambi verranno passati all'**AccessDecisionManager** che provvede a leggere le <GrantedAuthorities> settate all'interno dell'oggetto <Authentication>.



Più nel dettaglio ad un AccessDecisionManager è associato un insieme di voter che possono essere di due tipi, <*RoleVoter*>, <*AuthenticatedVoter*>. Il primo si esprime sulla base della rappresentazione in String dell'oggetto *GrantedAuthority* (che deve contenere il prefisso ROLE_), mentre il secondo si esprime sullo status di autenticazione dell'utente.

Per ogni richiesta di autorizzazione, ognuno dei voter deve fornire uno tra i seguenti esiti:

- *ACCESS_ABSTAIN*
- *ACCESS_DENIED*
- *ACCESS_GRANTED*

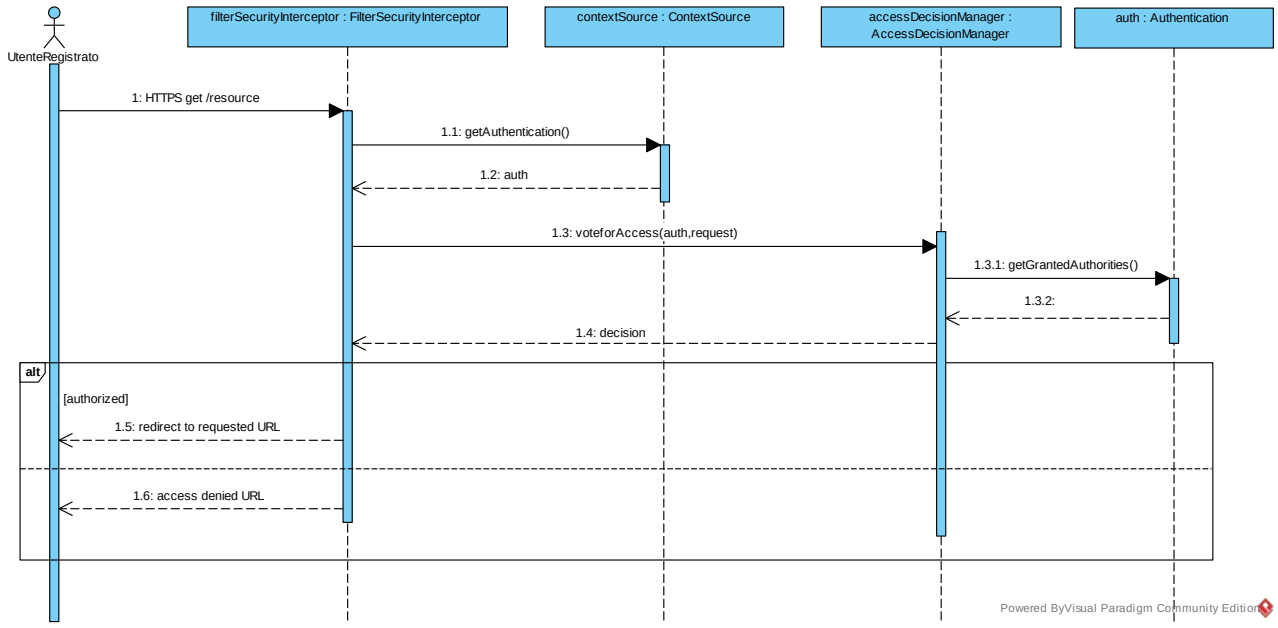
È possibile utilizzare diverse politiche di voto per ottenere la decisione finale, sfruttando le diverse implementazioni dell'*<AccessDecisionManger>*. Di default è utilizzata l'implementazione **'*AffirmativeBased*'**, che garantisce l'accesso alla risorsa richiesta se uno o più dei voter hanno fornito esito *'ACCESS_GRANTED'*.

Utilizzando la configurazione descritta, e quindi sfruttando i ruoli rappresentati dall'oggetto *<GrantedAuthority>*, il framework consente l'implementazione di un controllo accessi di tipo **Role-Based (RBAC)**.

A livello implementativo, tutto il meccanismo di autorizzazione si esplica nella configurazione di un **WebSecurityConfigurerAdapter**, come segue:

```
@EnableWebSecurity
@Configuration
public class SecurityConfiguration extends WebSecurityConfigurerAdapter{

    @Override
    protected void configure(HttpSecurity httpSecurity) throws Exception {

        httpSecurity
                    .authorizeRequests()

                    .mvcMatchers("/").permitAll()
                    .mvcMatchers("/login-panel/login").permitAll()
                    .mvcMatchers("/resources/public/**").permitAll()
                    .mvcMatchers("/resources/private/**").denyAll()
                    .mvcMatchers("/login-panel/accessDenied").authenticated()
                    .mvcMatchers("/login-panel/welcome").authenticated()
                    .mvcMatchers("/dashboard/**").authenticated()
                    .mvcMatchers("/ticket/details/*").authenticated()
                    .mvcMatchers("/user/list").access("hasAuthority('ROLE_MANAGER')")
                    .mvcMatchers("/user/profile/*").access("hasAuthority('ROLE_MANAGER')")
                    .mvcMatchers("/user/add").access("hasAuthority('ROLE_MANAGER')")
                    .mvcMatchers("/user/cambiaStato/*").access("hasAuthority('ROLE_MANAGER')")
                    .mvcMatchers("/ticket/send").access("hasAuthority('ROLE_CLIENTI')")
                    .mvcMatchers("/ticket/history_cliente").access("hasAuthority('ROLE_CLIENTI')")
                    .mvcMatchers("/ticket/history_operatore").access("hasAuthority('ROLE_OPERATORI')")
                    .mvcMatchers("/ticket/history_aperti").access("hasAuthority('ROLE_OPERATORI')")
                    .mvcMatchers("/ticket/aggiorna_stato/*").access("hasAuthority('ROLE_OPERATORI')")
                    .mvcMatchers("/user/queue/notify").access("hasAuthority('ROLE_OPERATORI')")
                    .mvcMatchers("/ws/**").authenticated()
                    .anyRequest().denyAll()
```

Di seguito, a titolo di esempio, è riportato il sequence diagram di uno scenario di autorizzazione di una richiesta su una risorsa effettuata da un utente già autenticato.



2.2 Autenticazione e Autorizzazione dei servizi di back-end

L'accesso ai diversi servizi offerti dal server MySQL e dal server LDAP, da parte dell'application server, è gestito in modo sicuro sfruttando i servizi esterni messi a disposizione da ***Hashicorp Vault***. In particolare, vengono sfruttate le funzionalità di ***dynamic credentials*** e ***static credentials*** per favorire una gestione centralizzata delle credenziali, in termini della loro manutenzione, rotazione o revoca, oltre che per evitare la problematica delle ***hardcoded credentials***, relativa a credenziali esposte in chiaro nei file di configurazione di Spring.

2.2.1 Installazione e configurazione Vault

A questo scopo, è necessario installare un Vault Server, dispiegato su un docker container, e configurarlo opportunamente, in modo che comunichi con l'application server.

All'atto dell'installazione di un Vault Server, l'amministratore di sistema riceve una **master key** e un **root token** necessari rispettivamente per l'avvio del server e l'accesso alle configurazioni dei servizi esposti.

Quindi l'amministratore deve farsi carico, in primo luogo, della configurazione dei seguenti servizi:

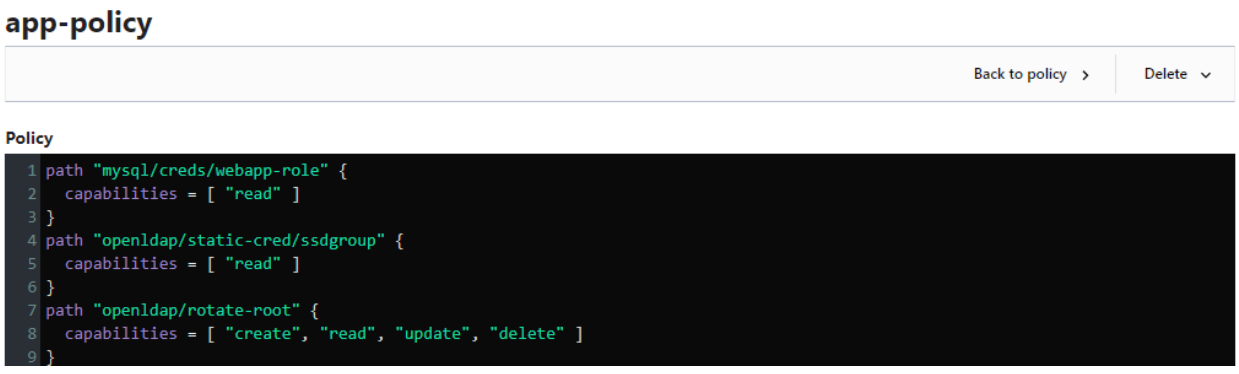
- **MySQL Secret Engine**: così come è stato configurato, permette la *generazione credenziali dinamiche* per l'accesso al MySQL Server, da parte dell'Application Server. Di fatto, quando il servizio è invocato, lo stesso aggiunge appositamente un utente tra gli users del DB, a cui sono associate tutti e soli i permessi relativi alle operazioni richieste dall'applicazione sul database.
  Per permettere ciò, a sua volta, Vault deve accedere al MySQL Server, utilizzando un utente del DBMS avente esclusivamente privilegi di *insert* e *delete* sulla tabella '*mysql.users*':

  ```
  CREATE USER 'vault_user'@'%' IDENTIFIED BY '{{password}}';
  GRANT insert,delete PRIVILEGES ON mysql.user TO 'vault_user'@'%' WITH GRANT OPTION;
  ```

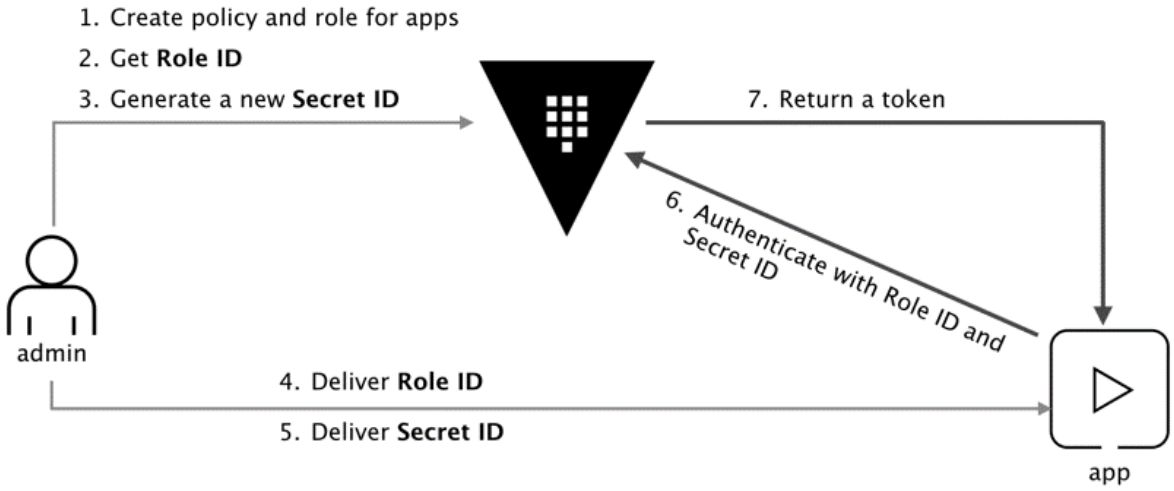
- **OpenLDAP Secret Engine**: così come è stato configurato, permette la *generazione* e la *rotazione di credenziali statiche* per l'accesso all'LDAP Server, da parte dell'Application Server. Per permettere ciò, a sua volta, Vault deve accedere all'LDAP Server, utilizzando un utente *'admin'* che abbia i permessi di lettura e scrittura delle credenziali sull'utente di LDAP associato all'applicazione Spring. Un parametro di configurazione importante è quello che permette l'automatica rotazione di queste credenziali, dopo un TTL di 12h.

2.2.2 Autenticazione e autorizzazione al servizio Vault

I servizi configurati precedentemente sono messi a disposizione da Vault all'esterno tramite delle chiamate API REST. Anche l'utilizzo di queste ultime deve essere opportunamente autentenicato ed autorizzato. Infatti l'amministratore oltre che configurare gli '*engine*' deve predisporre un profilo di autenticazione, associato univocamente all'applicazione Spring, e a cui sono associate opportune policy che gli permettono l'accesso a tutti e soli i servizi che deve invocare tramite API.



Al profilo di autenticazione configurato, vengono associati un *RoleID* e un *SecretID*, fornite all'applicazione Spring, che le permettono, all'atto del suo avvio, di autenticarsi a Vault.



Una volta autenticata, l'app può richiedere successivamente i servizi tramite le API esposte da Vault, utilizzando un token associato alla sessione di autenticazione.

2.2.3 Autenticazione e autorizzazione al servizio MySQL

Sfruttando il **MySQL Secret Engine**, configurato precedentemente, è possibile, per l'applicazione, usufruire del servizio di Vault *'**Dynamic Credential Creation**'*.

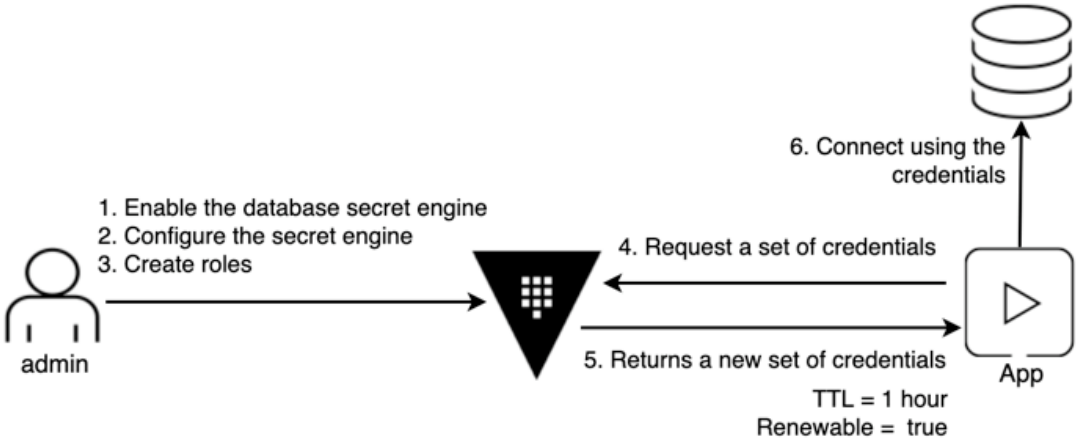
All'avvio dell'applicazione, con una chiamata API a Vault è possibile chiedere la generazione dinamica di credenziali per l'accesso al DB, e quindi la contestuale creazione dell'utente che l'applicazione può utilizzare, durante la sessione di connessione con il DB, per effettuare le operazioni sullo stesso.

Quindi la chiamata API "*read mysql/creds/webapp-role*" chiede a Vault di eseguire la seguente query sul DB:

> *CREATE USER '{{name}}'@'%' IDENTIFIED BY '{{password}}';GRANT insert, select, update ON ticket_system.* TO '{{name}}'@'%';*

Le credenziali generate sono caratterizzate da un TTL definito dall'amministratore, pari a 12h nel caso specifico, al termine del quale Vault automaticamente chiede il renew delle stesse.

Inoltre, un altro parametro che caratterizza le credenziali è il tempo *max_lease*, che rappresenta la durata complessiva massima delle credenziali, tenendo conto di tutti i rinnovi, al raggiungimento della quale l'utente creato nel database viene eliminato.



È necessario inoltre osservare che ogni singola chiamata della API '*read*' da parte dell'applicazione, che avviene ad ogni singolo riavvio del server, rinnova le credenziali precedentemente generate, anche se non ancora scadute.

2.2.4 Autenticazione e autorizzazione al servizio LDAP

Sfruttando l'**LDAP Secret Engine** configurato precedentemente è possibile, per l'applicazione, usufruire del servizio di lettura di *'**Static Credential**'*.

Le credenziali recuperate sono relative ad una LDAP entry, che rappresenta l'entità con la quale l'applicazione Spring può loggarsi ad LDAP e può consultare tutte le informazioni delle altre entry, per implementare i meccanismi di autenticazione e autorizzazione, nonché di revoca e riabilitazione di un utente della directory. Si è reso necessario, dunque, creare appositamente un utente che avesse privilegi di *write* sull'albero.

'***uid=approle, ou=people, dc=ssdgroup, dc=com'***

A tale scopo, il suddetto utente è stato inserito in un gruppo '*approle-admins'*, al quale sono stati assegnati, tramite una LDAP policy, i privilegi di *'write'* su tutto il sottoalbero.

Come è possibile notare dalla tabella che segue, i privilegi di *'write'* includono tutta una serie di altri privilegi di livello inferiore, quindi oltre a comprendere quelli necessari per l'aggiunta e rimozione di nuove entry, includono anche quelli di *'read'*, necessari per l'autenticazione degli end-user.

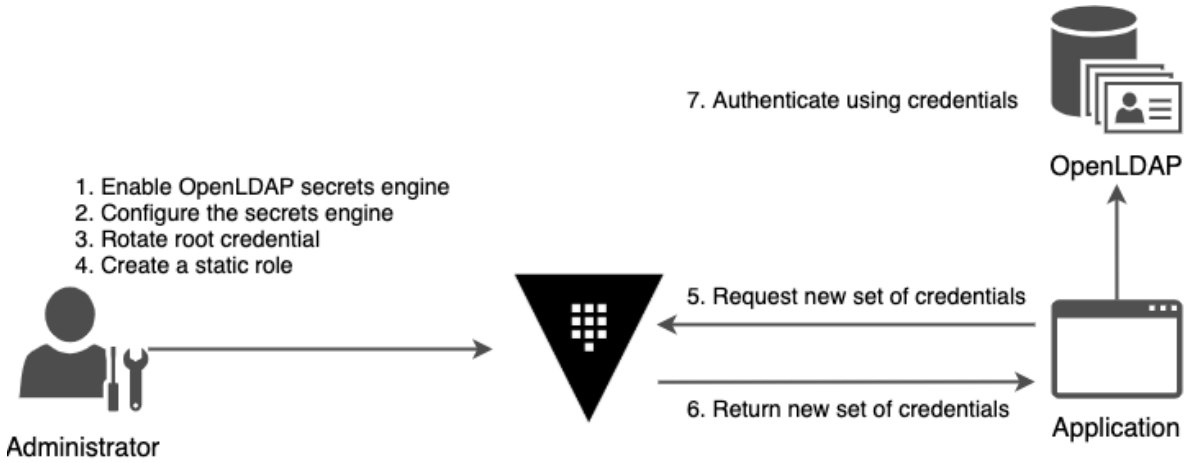
Table 6.4: Access Levels

Level	Privileges	Description
none =	0	no access
disclose =	d	needed for information disclosure on error
auth =	dx	needed to authenticate (bind)
compare =	cdx	needed to compare
search =	scdx	needed to apply search filters
read =	rscdx	needed to read search results
write =	wrscdx	needed to modify/rename
manage =	mwrscdx	needed to manage

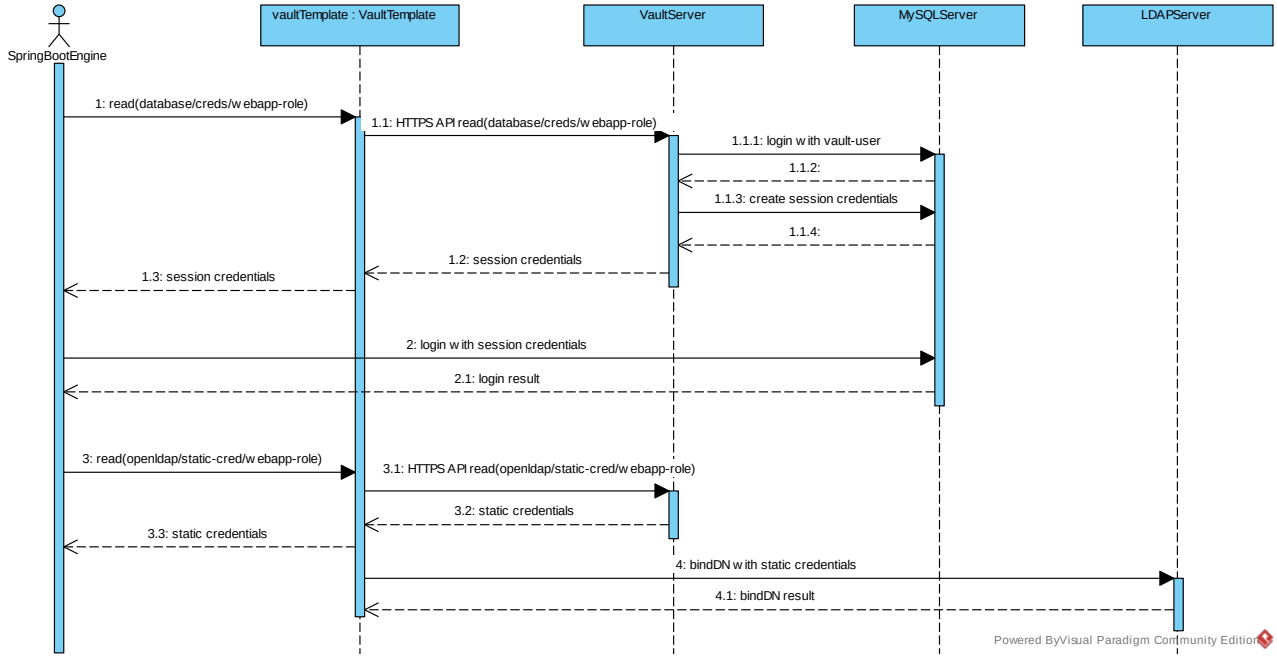
Di seguito è riportata la policy LDAP applicata al gruppo 'approle-admins':

```
dn: olcDatabase={1}mdb,cn=config
changetype: modify
delete: olcAccess
-
add: olcAccess
olcAccess: to attrs=userPassword,shadowLastChange by self write
                                                  by dn="cn=admin,dc=ssdgroup,dc=com" write
                                                  by group/groupOfUniqueNames/uniqueMember=cn=approle-admins,ou=groups,dc=ssdgroup,dc=com write
                                                  by anonymous auth
                                                  by * none
olcAccess: to * by group/groupOfUniqueNames/uniqueMember=cn=approle-admins,ou=groups,dc=ssdgroup,dc=com write
olcAccess: to * by self write by dn="cn=admin,dc=ssdgroup,dc=com" write
```

Quindi, all'occorrenza, quando l'applicazione necessita di consultare la directory LDAP, per stabilire la connessione con essa, le credenziali relative all'utente *'approle'* sono richieste a Vault con una chiamata API: '***read openldap/static-cred/ssdgroup'***.



Di seguito è riportato il sequence diagram relativo allo scenario di recupero, tramite Vault, delle credenziali per l'accesso sia ad LDAP e MySQL.



2.3 Transit e storage sicuro dei dati

Il **Transit Secret Engine** di Vault è un servizio che permette di implementare una '*Encryption as a Service*'. Ciò vuol dire che può essere utilizzato dall'applicazione come servizio esterno per crittare dati, sia in transito che per lo storage.

Nella pratica è molto utile perchè permette di evitare lo storage insicuro di chiavi di crittazione, che saranno, invece, conosciute soltanto da Vault che quindi, all'occorrenza, dovrà essere interpellato, con API specifiche, per la crittazione o la decrittazione dei dati.

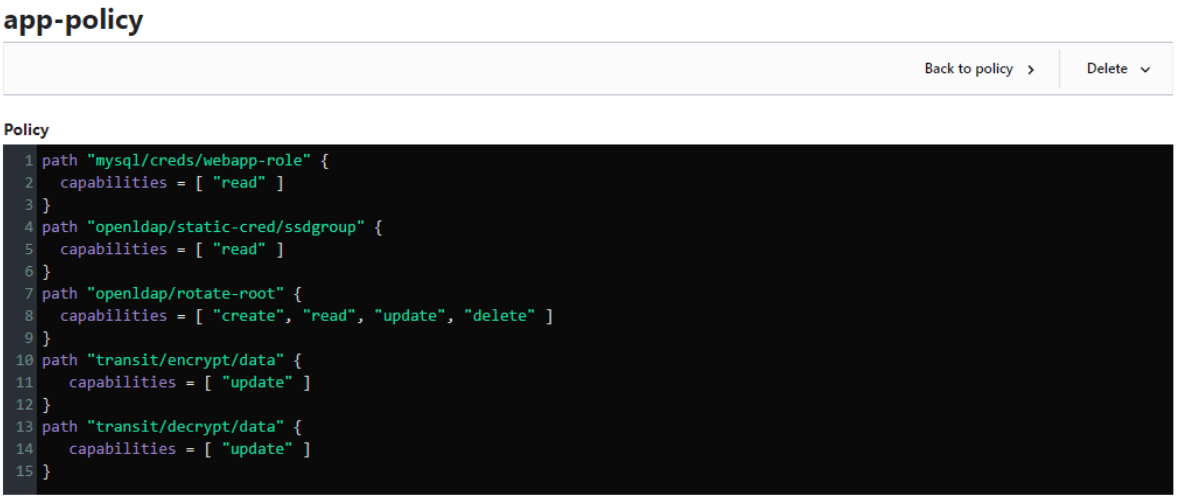
2.3.1 Configurazione Vault

Per configurare questo servizio l'amministratore deve abilitare il ***'Transit Secret Engine'***, oltre che creare delle chiavi crittografiche. In particolare sono state create due chiavi crittografiche, una per i dati dell'utente (*user_data_key*), l'altra per i dati del ticket (*ticket_data*_key).

Questa configurazione permette a Vault di esporre i servizi di encryption e decryption all'esterno, tramite le API:

- *transit/encrypt/<key_name>*
- *transit/decrypt/<key_name>*

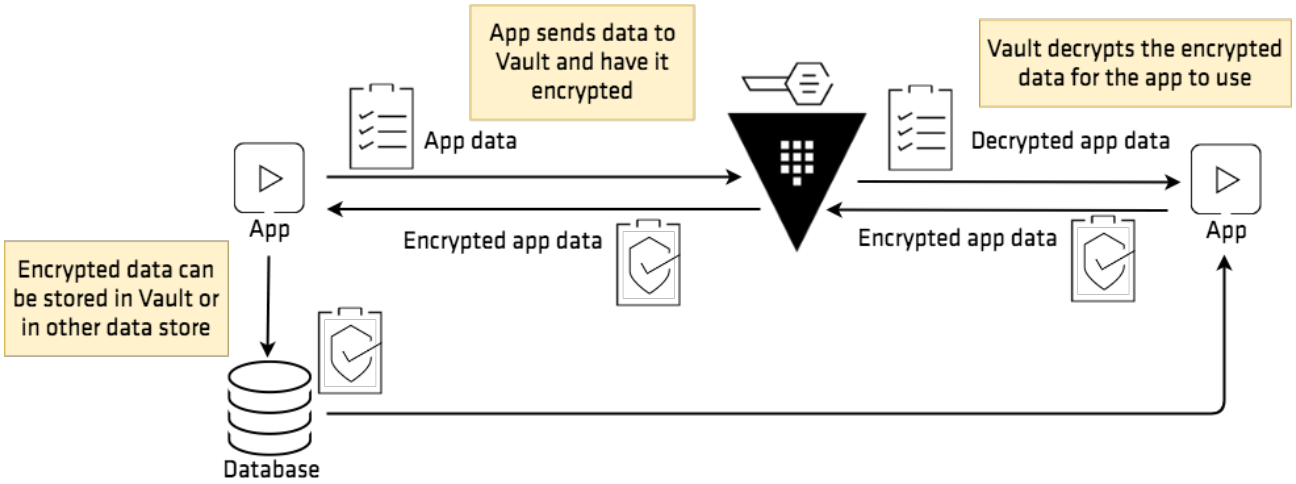
L'amministratore di sicurezza deve, quindi, estendere le policy relative al profilo di autenticazione utilizzato dall'applicazione, includendo i permessi di utilizzo delle API sopra citate.



È necessario notare che l'amministratore, periodicamente può ruotare le chiavi di encryption, creandone nuove versioni. Le vecchi chiavi resteranno comunque funzionanti, per decrittare eventuali dati crittati in passato.

2.3.2 Data Encryption & Decryption

Il servizio di '*Encryption as a Service*', come accennato, può essere utilizzato all'occorrenza, quando è necessario crittare o decrittare dati da salvare nel database o da leggere dal database.



In particolare, nell'applicazione sviluppata, tra tutti i dati conservati nel database, sono stati ritenuti dati sensibili le seguenti informazioni:

Tabella	**Colonna**
utenti	full_name
utenti	email
utenti	phone
ticket	title
ticket	description

Attraverso l'annotation ***'@Converter'*** del framework Spring è possibile marcare tutti gli attributi delle classi DAO che necessitano di una conversione, in lettura o in scrittura:

```
@Convert(converter = TransitConverterData.class)
@Column(name = "title", nullable = false, length = 250)
public String getTitle() {
    return this.title;
}
```

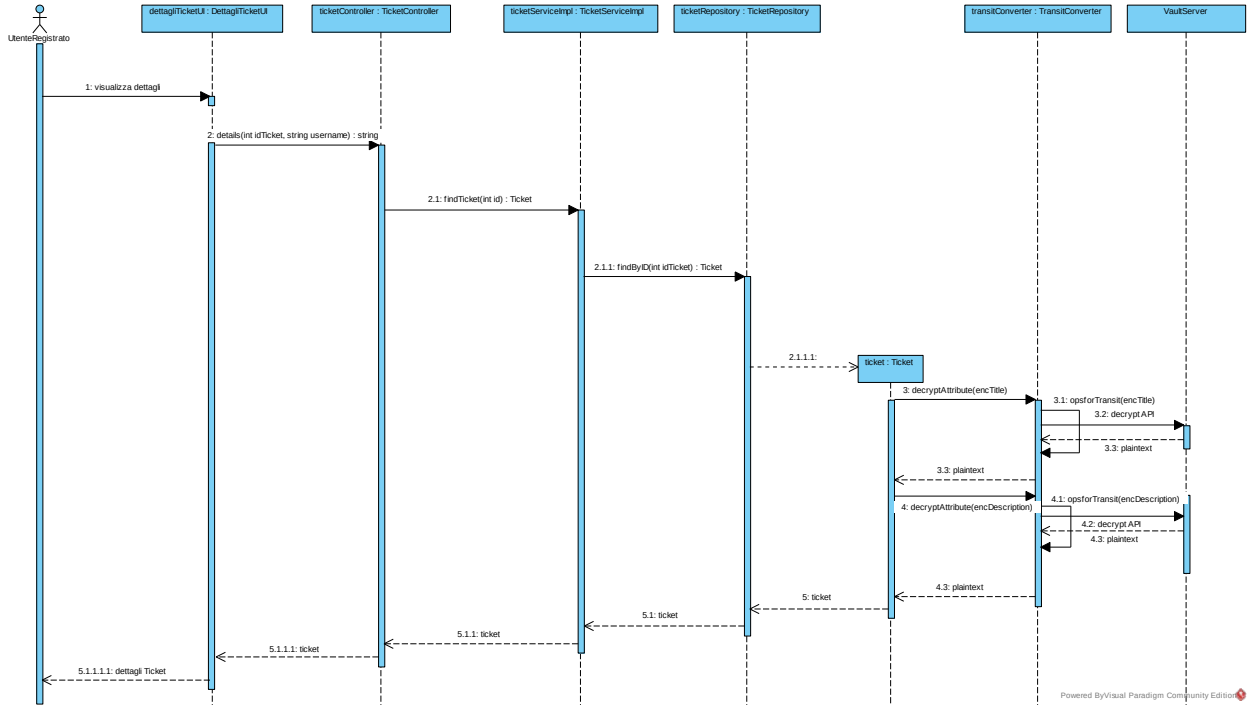
```
@Convert(converter = TransitConverterUser.class)
@Column(name = "full_name", nullable = false)
public String getFullName() {
    return this.fullName;
}
```

Come è possibile notare, con l'annotation '*@Converter*' è possibile specificare inolte la classe che si occuperà, all'occorrenza, dell'encryption e della decryption. In particolare, vengono utilizzate le classi <***TransitConverterUser***> e <***TransitConverterData***> per convertire i dati degli utenti e del ticket rispettivamente con le chiavi crittografiche ***user_data_key*** e ***ticket_data_key***.

A titolo di esempio, è riportata la classe Converter utilizzata per i dati dell'utente. Quest'ultima sfrutta la classe di framework Spring <***VaultOperations***> per utilizzare direttamente le API esposte da Vault relative all'encryption e la decryption.

```
public class TransitConverterUser implements AttributeConverter<String, String> {

        @Override
        public String convertToDatabaseColumn(String column) {
            VaultOperations vaultOps = BeanUtil.getBean(VaultOperations.class);
            Plaintext plaintext = Plaintext.of(column);
            String cipherText = vaultOps.opsForTransit().encrypt("user_data_key", plaintext).getCiphertext();
            return cipherText;
        }

        @Override
        public String convertToEntityAttribute(String column) {
            VaultOperations vaultOps = BeanUtil.getBean(VaultOperations.class);
            Ciphertext ciphertext = Ciphertext.of(column);
            String plaintext = vaultOps.opsForTransit().decrypt("user_data_key", ciphertext).asString();
            return plaintext;
        }

    }
```

Di seguito, infine, è riportato il sequence diagram relativo allo scenario di decrittazione dei dettagli del ticket:



2.4 Comunicazione sicura

Tutte le comunicazioni che avvengono tra i diversi servizi dell'architettura distribuiti sui docker, oltre che quelle che avvengono con i client, necessitano dei requisiti di **integrità** dei dati e **confidenzialità**, oltre che, per la maggior parte di esse, anche del requisito di **autenticazione**

Questi requisiti possono essere forniti a livello di trasporto, operando sulle comunicazioni TCP/IP, e quindi configurando il protocollo crittografico **Transport Layer Security** (TLS). Di seguito, è riportata la lista dei canali di comunicazione utilizzati nell'architettura:

Source	Destination	Protocol
Spring Application Server	Client	TLS
Client	Spring Application Server	TLS
Spring Application Server	Vault Server	TLS
Vault Server	Spring Application Server	TLS
Spring Application Server	LDAP Server	LDAP + StartTLS
LDAP Server	Spring Application Server	LDAP + StartTLS
Spring Application Server	MySQL Server	TLS
MySQL Server	Spring Application Server	TLS
Vault Server	LDAP Server	LDAP + StartTLS
LDAP Server	Vault Server	LDAP + StartTLS
Vault Server	MySQL Server	TLS
MySQL Server	Vault Server	TLS

È necessario osservare che il server Spring è stato configurato in modo da redirezionare tutte le richieste di tipo HTTP, dirette alla porta 80, verso la porta 8443 su HTTPS.

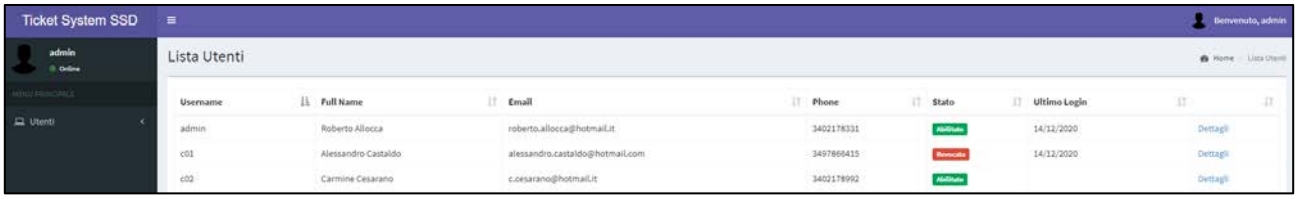
```
private Connector getHttpConnector() {
    Connector connector = new Connector(TomcatServletWebServerFactory.DEFAULT_PROTOCOL);
    connector.setScheme("http");
    connector.setPort(80);
    connector.setSecure(false);
    connector.setRedirectPort(8443);
    return connector;
}
```

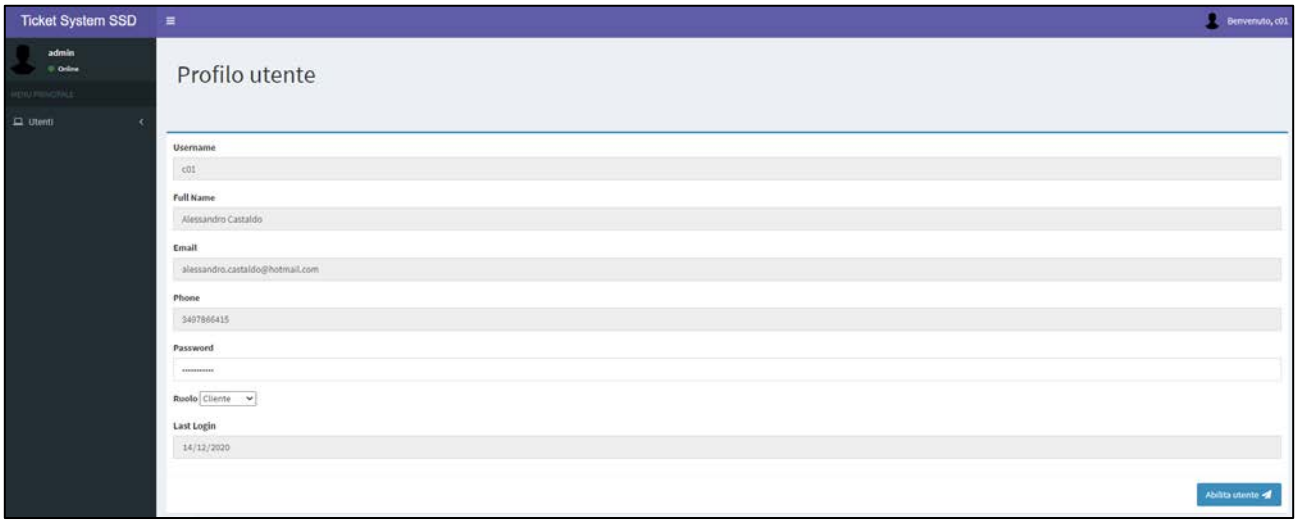
2.5 Account Management System

Per soddisfare il requisito relativo all'Account Management è stato, prima di tutto, creato un account '**Manager'**, inserendo la relativa entry in LDAP e creando un opportuno gruppo nel sottoalbero della directory.
È importante creare un gruppo che rappresenti il '**ruolo Manager'** perché, a partire da questo, è possibile implementare le logiche applicative che riguardano l'autorizzazione alle risorse che il manager deve utilizzare, in particolare:

- Lista degli account registrati
- Monitoring dell'ultimo login di ogni account
- Revoca delle credenziali relative ad un account
- Riabilitazione delle credenziali relative ad un account

È opportuno osservare che non è stata prevista la funzionalità di rimozione completa dell'account. Infatti, all'atto della revoca delle credenziali, viene operata soltato un rimozione dell'entry relativa all'account da disabilitare, dalla directory LDAP, non consentendo, di fatto, all'utente, di autenticarsi. Non vengono, dunque, perse le informazioni relazionate all'utente, in modo che quest'ultimo abbia la possibilità di riaccreditarsi al sistema, trovando lo stato del proprio profilo inalterato, dopo la riabilitazione da parte del manager.





In fase di aggiunta di un utente, un meccanismo di input validation vincola la scelta della password e fa sì che venga rispettato determinato pattern. In particolare una valida password deve contenere:

- Minimo 8 caratteri;
- Almeno un carattere speciale stampabile;
- Almeno un carattere maiuscolo;
- Almeno un carattere minuscolo;
- Almeno un numero;
- Assenza di caratteri non stampabili;
- Non contenere lo spazio.

Ciò rende un attacco *brute-force* molto meno efficace, perchè richiederebbe un tempo molto lungo. Allo scopo di debellare definitivamente questa minaccia, viene realizzato un ulteriore enforcement, implementando un meccanismo che disabiliti automaticamente le credenziali di un utente, qualora venissero effettuati **tre tentativi consecutivi** di accesso falliti.

```
@Component
public class AuthenticationListener implements ApplicationListener <AbstractAuthenticationEvent>
{

    @Autowired
    private UtenteService utenteService;

    @Override
    public void onApplicationEvent(AbstractAuthenticationEvent appEvent)
    {
      if (appEvent instanceof AuthenticationSuccessEvent) {
          AuthenticationSuccessEvent event = (AuthenticationSuccessEvent) appEvent;
          String username = ((UserDetails)event.getAuthentication().getPrincipal()).getUsername();

          Utente utente = utenteService.findByUsername(username.toString());
          utente.setAttempts(0);
          utenteService.save(utente);
      }

      if (appEvent instanceof AuthenticationFailureBadCredentialsEvent) {
          AuthenticationFailureBadCredentialsEvent event = (AuthenticationFailureBadCredentialsEvent) appEvent;
          String username = event.getAuthentication().getPrincipal().toString();

          Utente utente = utenteService.findByUsername(username);

          if (utente != null) {
              System.out.println("Bad Credentials: password errata");

              utente.setAttempts(utente.getAttempts() + 1);
              if (utente.getAttempts() > 2) {
                  utente.setStatus(0);
                  utenteService.removeLDAP(utente);
              }

              utenteService.save(utente);
          }
          else {
              System.out.println("Bad Credentials: utente non esistente");
          }

      }
    }
}
```

2.6 Session Management

L'ultimo enforcement riguarda la gestione delle sessioni di autenticazione. Attraverso le funzioni di framework esposte in Spring Security è possibile controllare esattamente quando le sessioni devono essere create e come Spring interagirà con esse.

In particolare, la sessione viene creata solo se richiesto, a seguito di autenticazione:

```
httpSecurity
            .sessionManagement()
            .sessionCreationPolicy(SessionCreationPolicy.IF_REQUIRED)
            .invalidSessionUrl("/login-panel/login?expired");
```

In questa fase, sono stati implementati alcuni meccanismi di sicurezza atti a mitigare la minaccia del **session hijacking**. Quest'ultimo si riferisce allo sfruttamento, da parte di un attaccante, di una normale sessione di lavoro per ottenere, tramite i cookies, un accesso non autorizzato alle informazioni o ai servizi di un sistema. Ciò può essere fatto con diversi tipi di attacchi:

- **Session fixation**
- **Session side jacking (MITM)**
- **Cross-site scripting**

Spring Security offre nativamente delle protezioni a questa minaccia, fornendo delle direttive per configurare opportunamente la sessione.

In particolare, si è scelto di:

- Invalidare ed eliminare i cookies di sessione all'atto del logout.
- Invalidare la sessione dopo un certo periodo di inattività, effettuando automaticamente il logout.
- Consentire un'unica sessione di autenticazione alla volta
- Settare i cookies come non accessibili da script lato client, ma accessibili solo tramite HTTP (flag *HTTPOnly*).

Di seguito, sono riportati i controlli implementati:

```
    httpSecurity
                .logout()
                .logoutUrl("/process-logout")
                .logoutSuccessUrl("/login-panel/login?logout")
                .deleteCookies("JSESSIONID")
                .and()
                .exceptionHandling().accessDeniedPage("/login-panel/accessDenied");

    httpSecurity
                .sessionManagement()
                .sessionCreationPolicy(SessionCreationPolicy.IF_REQUIRED)
                .invalidSessionUrl("/login-panel/login?expired")
                .maximumSessions(1)
                .maxSessionsPreventsLogin(true);
}

@Configuration
public class MyHttpSessionListener implements HttpSessionListener {
    @Override
    public void sessionCreated(HttpSessionEvent event) {
        event.getSession().setMaxInactiveInterval(600);                // 10 minutes
    }
}

@Bean
public HttpSessionEventPublisher httpSessionEventPublisher() {
    return new HttpSessionEventPublisher();
}
```

È opportuno osservare che è necessario implementare un Publisher che notifica tutti gli eventi di sessione, ed un Listener, in ascolto su di essi.

2.7 Input Validation

Oltre che la password, anche tutti i campi informativi che possono essere inseriti all'interno delle varie dashboard devono essere opportunamente validati al fine di evitare attacchi di tipo Injection sia sul server SQL che sul server LDAP.

In particolare per ognuno degli attributi che deve essere controllato, vengono utilizzate delle *@Annotation*, proprie di Spring Security, per controllare le diverse proprietà, ad esempio:

- `@Min, @Max, @NotNull, @Pattern(regexp = "^[a-zA-Z\\u0020]*$")`

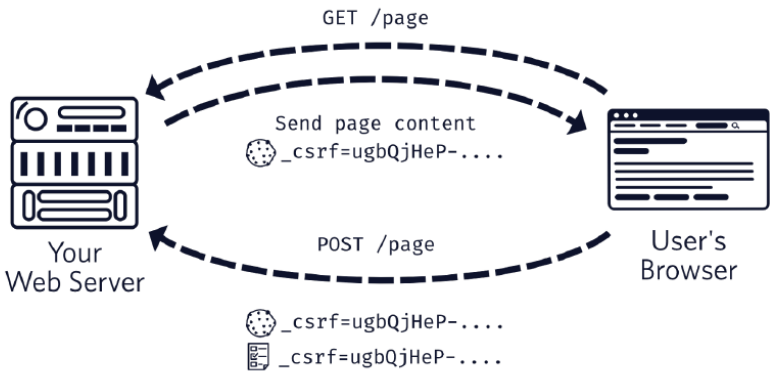
2.8 Cross-site Request Forgery

Il Cross-site request forgery (CSRF) è una minaccia che può essere sfruttata quando i siti web dinamici non risolvono la vulnerabilità per la quale accettano richieste da un client senza meccanismi per controllare se la richiesta sia stata inviata intenzionalmente oppure no. In particolare si ritiene non intenzionale una richiesta che non parta dagli elementi grafici delle views.

Le contromisure adottate per mitigare questa tipologia di attacco comprendono:

- Utilizzare richieste HTTP POST per tutte le risorse critiche del sistema.
- Proteggere le richieste HTTP POST con il meccanismo di sicurezza nativamente offerto da Spring Security.

Il meccanismo di sicurezza di Spring è basato sui token e funziona nel seguente modo: per ogni richiesta GET relativa ad una pagina che contiene una POST, il server assegna al client un token pseudo-casuale sufficientemente lungo. Lo stesso token dovrà essere inserito nell'header della richiesta HTTP POST e confrontato dal server. La richiesta verrà servita solo se il token ricevuto corrisponde a quello generato dal server.



2.9 Cross-site Scripting

Il Cross-site Scripting è un'attacco che permette ad un attaccante di inserire o eseguire codice malevolo lato client al fine di raccogliere dati, manipolarli, reindirizzare a risorse riservate, ecc. Questa tipologia di attacco sfrutta la vulnerabilità di un sito web dinamico che impiega un insufficiente controllo dell'input nei form.

Con il framework Spring, questo può essere mitigato operando una opportuna **Input Validation**, e nei form HTML, con il tag '***htmlEscape'*** che permette di fare l'escape di caratteri speciali utilizzati comunemente per eseguire script, che verranno invece considerati con il loro significato letterale:

```
<form:input path="someFormField" htmlEscape="true" />
```

2.10 Configuration File Encryption

Per l'applicazione del protocollo TLS sulle comunicazioni tra i diversi servizi dell'architettura, come accennato, è necessario utilizzare dei certificati, opportunamente mantenuti all'interno di keystore. In particolare:

- **Keystore_app**: keystore contenente coppia di chiave e certificato necessari per la comunicazione TLS tra i client browser e l'Application Server.
- **Keystore_vault**: keystore contenente coppia di chiave e certificato necessari per la comunicazione TLS tra l'Application Server e Vault.
- **Keystore_ldap**: keystore contenente coppia di chiave e certificato necessari per la comunicazione TLS tra l'Application Server ed LDAP Server.
- **Keystore_mysql**: keystore contenente coppia di chiave e certificato necessari per la comunicazione TLS tra l'Application Server e MySQL Server.

Tutti i keystore sono opportunamete crittati, e le password per l'accesso sono mantenute in chiaro all'interno dei file di configurazione dell'applicazione Spring. Oltre a queste, nei file di configurazione sono presenti le credenziali che l'Application Server utilizza per accedere ai servizi di Vault (secretID, roleID).

La presenza in chiaro, nei file di configurazione, di queste informazioni sensibili rappresenta un forte vulnerabilità per il sistema, dunque si è provveduto a crittarle mediante '*PBEWithMD5AndDES*', una funzione crittografica che effettua un encryption password-based, utilizzando MD5 e DES.

Mediante l'uso della libreria *JASYPT* (Java Simplified Encryption), queste informazioni sono decrittate solo all'occorrenza, sfruttando una secret key, passata all'interprete JAVA con il comando di esecuzione dell'applicazione:

```
FROM openjdk:8u171-jdk-alpine
ARG JAR_FILE=target/*.war
COPY ${JAR_FILE} ticket-system.war
ENTRYPOINT ["java","-jar","-Djasypt.encryptor.password=secretkey","/ticket-system.war"]
```

Di seguito un esempio di file di configurazione con le password crittate:

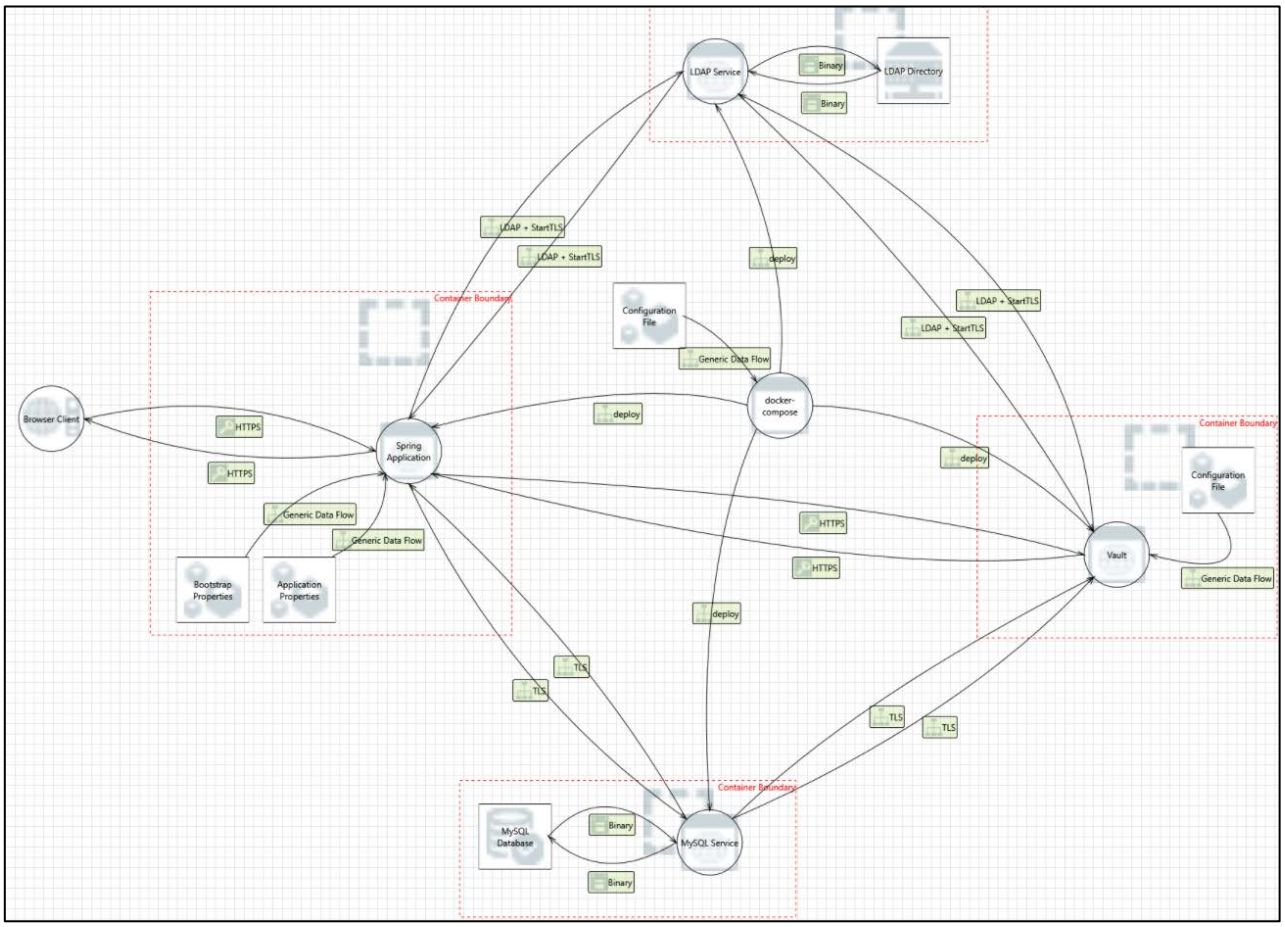
```
spring.cloud.vault.app-role.app-role-path=approle
spring.cloud.vault.app-role.role-id=ENC(JAGW8p6JRXLaVipDo0xqB8u7LEX4OiWw0JLftYK/63NUONHRUq5ZxumLcUR+WMbb)
spring.cloud.vault.app-role.secret-id=ENC(QhdovtDNkz4cv4y8ui0M1zBZmvhryi7qDLguXSmfHUWln5Dkygaqqp9Trei0KhX2)
spring.cloud.vault.authentication=APPROLE

# Keystore contains public and private key to allow secure TLS communication APP<->Vault
spring.cloud.vault.ssl.trust-store=classpath:private/keystore_vault.p12
spring.cloud.vault.ssl.trust-store-password=ENC(Qk4w8hvxdyk/Z3nlToHuEtBg9ePtXRVM)
```

3. Analisi di sicurezza

Per la fase di analisi di sicurezza è stato adottato un approccio di tipo **threat-oriented**.

3.1 Threat Modeling

In questa fase, è stata utilizzata la piattaforma '*Microsoft Threat Modeling Tool*', per operare un'identificazione delle *threat sources* e dei *threat events* caratteristici della specifica architettura progettata.



La modellazione dell'architettura, comprensiva della configurazione di tutti i componenti e i connettori, è stata fatta tenendo presente tutte le contromisure già adottate nella fase di Security Enforcement.

Pertanto, l'analisi condotta dal tool ha elencato, per ciascuna categoria, tutti e soli i threats per i quali non è stata adottata alcuna contromisura, oppure per i quali non è stato possibile specificare quelle adottate, per via delle limitazioni del tool in termini di configurazione.

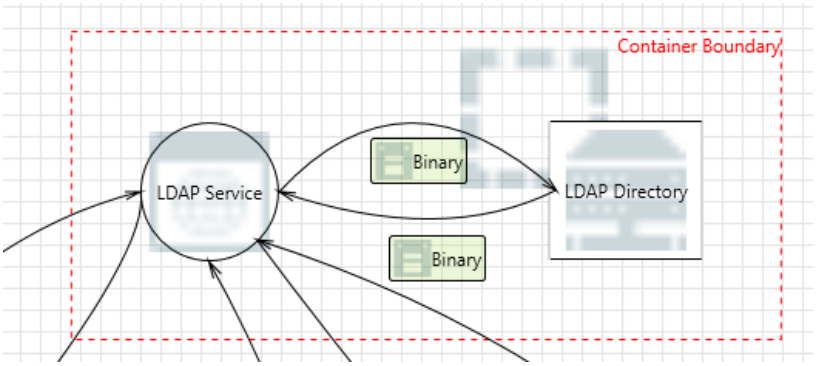
In molti casi, infatti, dato il threat rilevato, era già stata predisposta una mitigazione; un esempio è costituito dall'isolamento fornito dalla dockerizzazione, che non è portato in conto dalla threat analysis operata dal software, perchè non configurabile.
I threats individuati dal tool sono categorizzati secondo la classificazione di Microsoft **STRIDE**. In allegato il report consultabile.

3.1.1 Spoofing

Lo spoofing è l'atto, da parte di un attaccante, di falsificare l'identità in una comunicazione, minando il requisito di **autenticazione**. Nel contesto dell'architettura in esame, sono stati rilevati differenti threats che riguardano questa categoria:

Threat	Asset	Mitigato	Note
Spoofing	Browser Client -> Spring Application	si	L'autenticazione del Client, in questa comunicazione non è un requisito richiesto.
Spoofing	Spring Application <–> File Application Properties	si	Isolamento del Docker
Spoofing	Vault <–> Configuration File	si	Isolamento del Docker
Spoofing	MySQL Service <–> MySQL Database	si	Isolamento del Docker
Spoofing	LDAP Service <–> LDAP Directory	si	Isolamento del Docker

Tutti i threats di tipo spoofing che riguardano interazioni '*binary*', interne ai docker container, di fatto, sono mitigate per la proprietà di isolamento logico degli stessi.



3.1.2 Tampering

Il tampering è l'atto di manipolare, distruggere o modificare deliberatamente dei dati, sia '*in transit*' che '*at rest*', attraverso canali non autorizzati, minando alla loro **integrità**.

Di seguito la lista dei threats, relativi a questa categoria, che sono stati individuati nella nostra architettura:

Threat	Asset	Mitigato	Contromisura
Process Memory Tampered	• Spring Application • MySQL • LDAP	si	Isolamento del docker
Replay Attacks	Tutte le connessioni	in parte	Solo a livello trasporto con TLS. Nessuna contromisura a livello applicazione.
Ip Fragmentation Attack	Tutte le connessioni	no	
Cross Site Scripting (Spring Application)	Spring Application	si	• Input Validation • HTML Escape
Session Hijacking	Browser Cookies	si	• La sessione viene distrutta al logout e dopo dieci minuti di inattività. • Non sono consentite sessioni contemporanee.

3.1.3 Repudiation

La minaccia di ripudio è associata a entità che possono negare di aver fatto un'azione senza che qualcuno possa provare il contrario.

Threat	Asset	Mitigato	Contromisura
Potential Data Repudiation	Tutti i servizi dispiegati	no	Dovrebbe essere implementato un meccanismo di logging delle azioni operate sul sistema.

3.1.4 Information Disclosure

La minaccia di divulgazione delle informazioni riguarda la rivelazione delle stesse ad individui che non dovrebbero averne accesso.

Threat	Asset	Mitigato	Contromisura
Authorization Bypass	MySQL Database	sì	il MySQL Database è accessibile direttamente solo dal MySQL Service, perchè isolato nel docker.
Authorization Bypass	LDAP Database	sì	LDAP Directory è accessibile direttamente solo dal LDAP Service, perchè isolato nel docker.
Information Disclosure	• Application Properties Spring • Bootstrap Properties Spring	si	I file di configurazione sono stati crittati con la libreria JASYPT

3.1.5 Denial of Service

La minaccia di Denial of service può tradursi nell'impossibilità da parte di utenti legittimi di raggiungere un determinato servizio web.

Threat	Asset	Mitigato	Contromisura
Data Flow TLS Interruption (SYN Flood)	Tutte le connessioni TLS	no	
Service Interruption	• LDAP Service • MySQL Service • Spring Application • Vault Service	no	

3.1.6 Elevation of Privilege

In questo tipo di minaccia, un utente non privilegiato potrebbe ottenere accesso privilegiato, potendo assumere permessi sufficienti a compromettere o distruggere l'intero sistema.

Threat	Asset	Mitigato	Contromisura
Cross Site Request Forgery	Spring Application	si	CSRF Protection
Privilege Escalation	Spring Application	si	E' stato progettato ed implementato un opportuno meccanismo di autorizzazione
Privilege Escalation	MySQL Server	si	Risolto dalla versione in esercizio di MySQL (5.7.32)

3.2 Vulnerability Analysis

Come previsto dall'approccio threat-oriented, dopo aver identificato le threat sources e i threat events sono state identificate tutte e sole le vulnerabilità che ricadono nel contesto dei threats emersi.

3.2.1 Vulnerabilità MTMT non mitigate

Di seguito sono analizzate tutte e sole le vulnerabilità legate ai threat analizzati dal tool di Microsoft che risultano essere, nella versione definitiva del sistema, non mitigate o mitigate in parte.

- Mancanza di una gestione a livello applicazione per l'invalidazione dei token di POST. Questa vulnerabilità permette ad un utente, legittimamente autenticato al sistema, di effettuare un **replay attack**, inoltrando un duplicato di un pacchetto precedentemente inviato al server. A livello trasporto, con TLS, questa minaccia è mitigata, evitando che lo stesso attacco venga eseguito da un utente non autenticato, impossibilitato a sniffare il traffico di rete.
- Mancanza di un controllo a livello rete relativo alle dimensioni dei pacchetti ricevuti, prima di essere accettati. Ciò espone il sistema alla minaccia dell'**IP Fragmentation**, con conseguente minaccia all'availability.
- Mancanza di un controllo per evitare gli attacchi **DoS**, a cui TLS, come anche TCP, è esposto. In particolare, manca la progettazione di meccanismi che permettano ai singoli servizi di gestire eventuali **SYN Flood** malevoli (come load balancers o firewalls).
- Mancanza, nel caso di assenza di servizio di un container Docker, di un meccanismo che garantisca la *business continuity* come, ad esempio, *Kubernetes*.

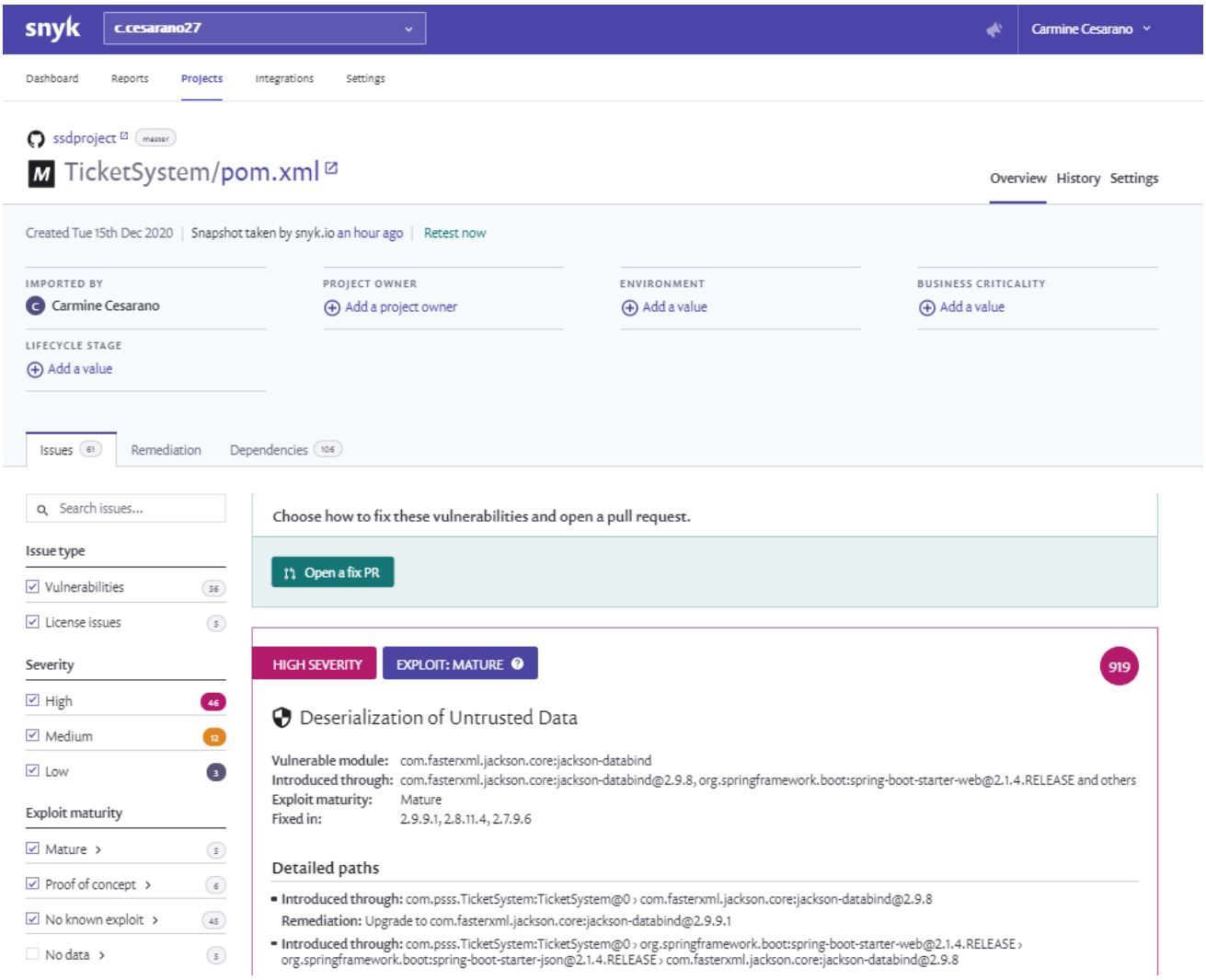
3.2.2 Vulnerabilità nelle tecnologie

Di seguito, per tutte le tecnologie utilizzate nel sistema in esame, sono state analizzate le vulnerabilità ad oggi note, tenendo conto delle versioni adottate. Inizialmente, sono state analizzate tutte le dipendenze incluse ed utilizzate nell'applicazione Spring:

Dipendenza	Versione	Esposta a	Fixed with
Spring Boot Starter Web	2.1.4.RELEASE	• Deserialization of Untrusted Data CVE-2019-12086 • XML External Entity Injection CVE-2020-25649 • Improper Input Validation CVE-2020-5421 • Reflected File Download CVE-2020-5398 • Cross-site Scripting CVE-2019-10219	2.1.17.RELEASE
Spring Boot Starter Data JPA	2.1.4.RELEASE	• SQL Injection CVE-2020-25638	2.1.7.RELEASE
Spring Boot Starter Security	2.1.4.RELEASE	nessuna nota	
Spring Boot Devtools	2.1.4.RELEASE	nessuna nota	
Spring Security Taglibs	5.1.5.RELEASE	nessuna nota	
Spring Boot Starter Test	2.1.4.RELEASE	nessuna nota	
Spring Boot Starter Websocket	2.1.4.RELEASE	nessuna nota	
Spring Messaging	5.1.6.RELEASE	nessuna nota	
Spring Boot Starter Integration	2.1.4.RELEASE	nessuna nota	
Spring LDAP Core	2.3.2.RELEASE	nessuna nota	
Spring Security LDAP	5.1.5.RELEASE	nessuna nota	
Unboundid LDAPSDK	4.0.10	nessuna nota	
Spring Cloud Starter Vault Config	2.1.2.RELEASE	• Information Exposure CVE-2020-26939	not fixed
Spring Cloud Vault Databases	2.1.2.RELEASE	nessuna nota	
Tomcat Embed Jasper	9.0.17	• Remote Code Execution CVE-2019-0232 • Denial of Service CVE-2019-10072 • Privilege Escalation CVE-2019-12418 • Session Fixation CVE-2019-17563	9.0.35
MySQL Connector Java	8.0.15	nessuna nota	
Tiles JSP	3.0.8	• Arbitrary Code Execution CVE-2014-0114	not fixed
Javax Servlet JSP JSTL	1.2.1	• XML External Entity Injection (XEE)	not fixed

Le vulnerabilità esposte, relative alle dipendenze, sono state ricercate utilizzando il framework **Synk.** Quest'ultimo, fornisce l'elenco dei threats a cui l'applicazione Spring è esposta, analizzando il file delle dipendenze *'pom.xml'* e confrontando le versioni degli artefatti utilizzati, con il database CVE delle vulnerabilità note, offerto dal MITRE.

Di seguito, è riportata la dashboard che permette di ispezionare il dettaglio per ognuno dei threat riscontrati, inclusa la versione dell'artefatto nel quale la vulnerabilità è risolta, e il grado di maturità dell'exploit (*nessun exploit*, *exploit diffuso*, *exploit conosciuto ma di difficile applicabilità*).



La maggior parte delle vulnerabilità riscontrate sono state fixate, aggiornando opportunamente le release delle dipendenze.

Successivamente, invece, con approccio manuale e consultando sempre il CVE del MITRE, sono state analizzate le vulnerabilità note relative a tutti gli altri servizi utilizzati nell'architettura in esame:

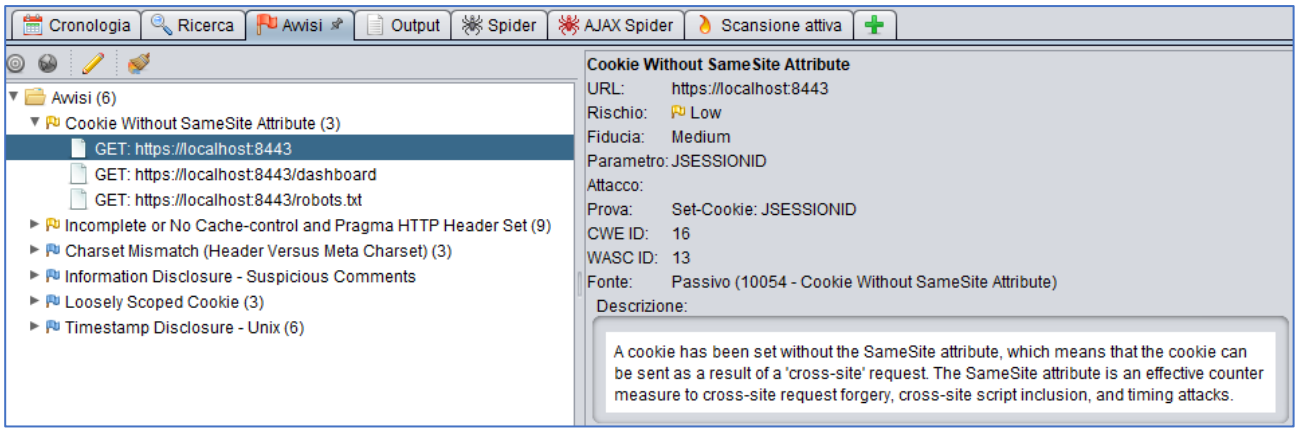
Service	Versione	Vulnerabilità note
MySQL Community Server	5.7.32	nessuna
OpenJDK	12	nessuna
slapd (openLDAP host)	2.4.50	CVE-2020-25692: un attaccante non autenticato potrebbe causare un DoS al processo slapd
Vault Server	1.6.0	nessuna
Windows 10 Pro	1909	nessuna
Docker Desktop Community	3.0.0	nessuna

3.3 Penetration testing

In questa fase, è stato utilizzato lo strumento **OWASP Zed Attack Proxy (ZAP)**, che permette di effettuare un penetration testing sull'applicazione in esecuzione e, dunque, di rilevare eventuali ulteriori vulnerabilità, non ancora emerse nelle fasi precedenti.

In particolare, eseguendo una **sessione di attacco automatica**, il tool prova a trovare, e contestualmente sfruttare tutte le eventuali vulnerabilità esposte dall'applicazione, per garantirsi l'acccesso alle risorse.

In particolare, a seguito di una sessione automatica sono state riscontrate due vulnerabilità di rischio **'Low'.**

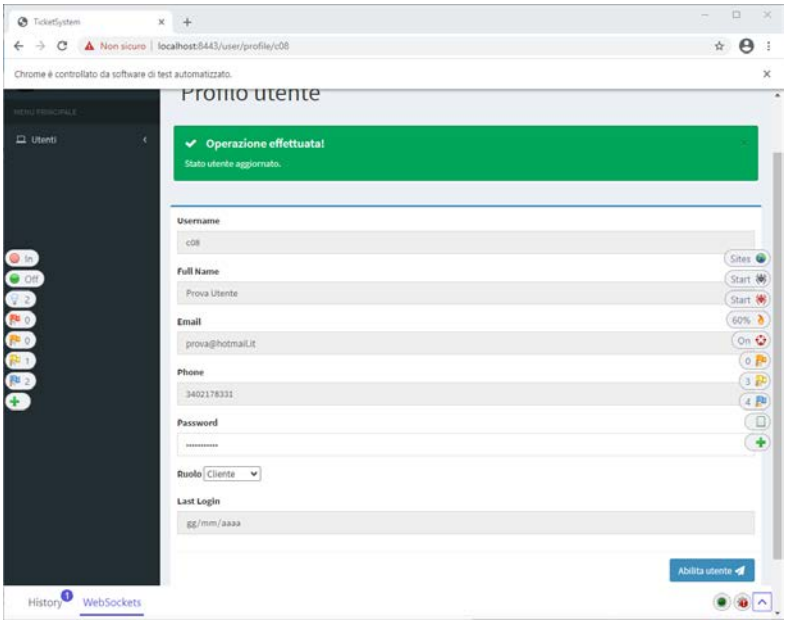


Tali vulnerabilità possono essere esposte come segue:

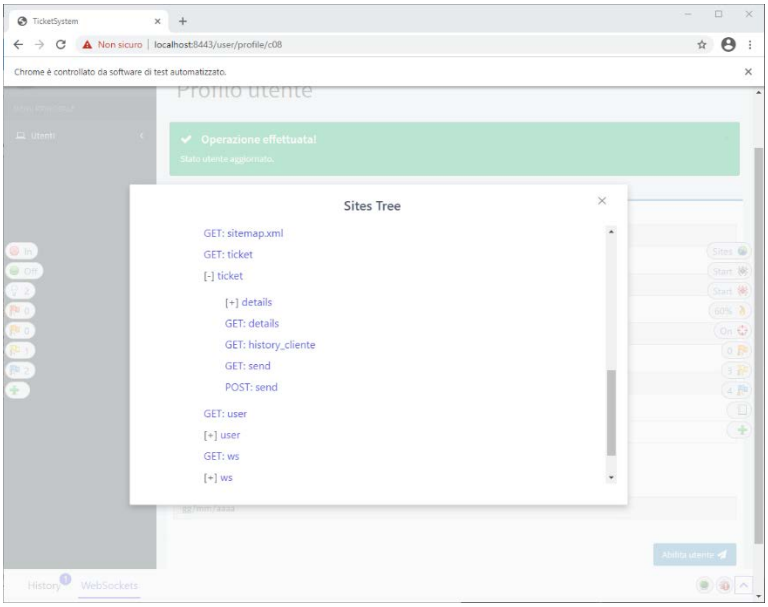
- **Cookie Without SameSite Attribute:** questo attributo può essere utilizzato per controllare se e come i cookies vengono inviati nelle richieste cross-site. Impostando questo attributo, viene fornito un ulteriore livello di difesa relativo agli attacchi CSRF che potrebbe mitigare eventuali difetti nella difesa basata su token (esposta precedentemente). Settando questo attributo a *'Strict'* si fornisce il massimo livello di sicurezza, e in questo modo il browser non includerà il cookie di sessione in nessuna risposta a richieste provenienti da pagine esterne al nostro sito, e bisognerà dunque ri-autenticarsi ogni volta.

- **Incomplete or No Cache-control and Pragma HTTP Header Set**: questi attributi permettono al server di indicare al browser client di non memorizzare in cache eventuali dati sensibili contenuti nelle risposte.

Utilizzando, invece, una **sessione di attacco manuale**, è possibile operare un analisi più approfondita. Questa permette, dopo aver attivato il tool di scansione, di effettuare ancora degli attacchi di tipo semi-automatizzato, ma che vengono testati dal tool durante una sessione di lavoro.



Dopo aver effettuato il login con i diversi tipi di utenti previsti dall'applicazione, navigando tra le risorse, ed utilizzando le funzionalità esposte, è possibile da un lato costruire la sitemap del sito, che definisce tutte le risorse raggiungibili dagli utenti, dall'altro, per ognuna di esse, provare una serie di attacchi, automaticamente, che permettono di trovare altre eventuali vulnerabilità non scoperte con l'analisi automatica.



In particolare, a seguito di questa sessione, non vengono riscontrate ulteriori vulnerabilità rispetto a quelle identificate con l'analisi automatica, e di cui si riporta il report in allegato.

4. Assessment NIST

In questa sezione finale, vengono analizzati tutti gli aspetti di sicurezza, così come presentati dallo standard **'NIST 800-53 rev4'**, per comprendere quanto la progettazione e la relativa implementazione del sistema presentato sia conforme a tali linee guida, con particolare riferimento ai livelli di sicurezza proposti. Riguardo il sistema in esame, data la bassa criticità del contesto in cui opera, si è scelto di far riferimento alla baseline ***LOW***, anche se, in alcune delle aree analizzate sono stati implementati controlli di livello ***MODERATE***.

E' possibile consultare l'elenco riassuntivo dei controlli, implementati e non, al seguente *link*.

4.1 Access Control [AC]

4.1.1 [AC-1] Access Control Policy and Procedures

Riguarda, in generale, la definizione di politiche e procedure per l'efficace implementazione di un meccanismo per il controllo degli accessi. Ciò è stato ampiamente coperto, dato che rappresenta uno dei principali obiettivi del lavoro di enforcement operato.

4.1.2 [AC-2] Account Management

Riguarda l'implementazione di un sistema di account management, che preveda una figura avente privilegi amministrativi, che abbia la responsabilità di gestione degli utenti registrati al sistema. In particolare, il sistema deve permettere all'amministratore (*manager*) di identificare gli utenti autorizzati, specificarne i privilegi di accesso, in funzione delle tipologie di account previste, monitorare gli accessi, revocare credenziali o riabilitarle, oltre che aggiungere nuovi account ed abilitarli al sistema.

Uno degli enhancement implementati, relativi a questa sezione, è il controllo che garantisce il logout da una sessione di un utente dopo un certo periodo di inattività (10 minuti).

[***TO MODERATE***] Tutti gli enhancement che prevedono gestione automatica degli account (ad esempio, disabilitazione automatica degli utenti inattivi) o dinamica (ad esempio cambio del ruolo a tempo di esecuzione) non sono stati implementati.

4.1.3 [AC-3] Access Enforcement

Riguarda, in generale, la definizione di politiche di autorizzazione per l'accesso logico, da parte di utenti o processi, alle informazioni e alle risorse del sistema. In particolare sono state utilizzate politiche basate su ruolo, definito staticamente all'interno della directory LDAP, e per quanto riguarda i meccanismi di autorizzazione alle risorse, sono stati sfruttati ampiamente quelli messi a disposizione dal framework Spring Security.

4.1.4 [AC-4] Information Flow Enforcement

[***TO MODERATE***] Non sono stati applicati particolari controlli sul flusso dell'informazione, come ad esempio limitazioni sul traffico che proviene da reti esterne verso quelle dell'organizzazione, tramite, ad esempio, tunnel crittografici, firewall, packet inspection mechanism, per restringere il traffico solo a quello che rispetta determinate policy interne.

4.1.5 [AC-5] Separation of Duties

La separation of duties è un approccio che permette di gestire i conflitti di interesse, e quindi di evitare eventuali attività malevole colluse. Include, ad esempio, la divisione delle funzioni di mission e di supporto al sistema tra diversi individui, contraddistinti da diversi ruoli, oltre che, la divisione di funzioni critiche per il sistema con diversi individui. Ciò è stato ampiamente rispettato con l'assegnazione oculata delle funzioni di business ai diversi ruoli previsti per il sistema, oltre che con la divisione della responsabilità di accesso alle funzioni di sicurezza tra diversi amministratori (ad esempio Vault necessita di tre chiavi per essere sbloccato).

4.1.6 [AC-6] Least Privilege

Riguarda l'adozione del principio del privilegio minimo, che consente ai diversi tipi di utenti l'autorizzazione ristretta alle sole funzioni necessarie a svolgere i propri task. È stato ampiamente rispettato, implementando anche diversi enhancement previsti per tale sezione, come l'applicazione dello stesso principio anche alle funzioni di sicurezza, oltre che l'assenza di account privilegiati di tipo 'superuser'.

[***TO MODERATE***] Un possibile enhancement, previsto dal livello moderate, sarebbe quello di prevedere l'auditing relativo all'utilizzo delle funzioni privilegiate, in modo da monitorare l'accesso alle funzionalità più critiche del sistema.

4.1.7 [AC-7] Unsuccessful Logon Attempts

Riguarda l'applicazione di un limite di tentativi di accesso non validi consecutivi da parte di un utente. Questo è stato soddisfatto implementando un meccanismo che blocca l'account in modo indefinito, impedendogli l'autenticazione, fin quando lo stesso non viene riabilitato dall'amministratore (*manager*), se l'utente ha superato il limite massimo di tre tentativi di accesso non validi.

4.1.8 [AC-8] System Use Notification

Riguarda l'utilizzo di sistemi di notifica, che sfruttano warning banner o pop-up, mostrati agli utenti prima che essi effettuino il login al sistema, e che forniscono agli stessi informazioni che riguardano la privacy dei dati sensibili.

4.1.9 [AC-11] Session Lock

[***TO MODERATE***] Riguarda l'implementazione di meccanismi che bloccano la sessione di un utente dopo un certo periodo di inattività, e che prevedono di ristabilire l'accesso solo previo login. Nel caso specifico non viene implementato, perchè dopo un periodo di inattività, la sessione viene distrutta completamente, effettuando il logout automatico, e non bloccandola.

4.1.10 [AC-12] Session Termination

Riguarda la terminazione delle sessioni avviata dall'utente, dopo specifici trigger definiti dall'organizzazione. In particolare questo controllo è stato realizzato implementando un meccanismo che termini la sessione dell'utente dopo che egli ha effettuato il logout, oltre che dopo un periodo di inattività di 10 minuti. In quest'ambito è stato implementato anche l'enhancement che prevede che sia mostrato un messaggio all'utente, dopo che la sessione è terminata.

4.1.11 [AC-14] Permitted actions without identification or authentication

Riguarda la definizione precisa delle funzionalità o le azioni che possono essere eseguite da entità senza che esse siano identificate o autenticate. In particolare, un utente anonimo può solo raggiungere la pagina di login, per effettuare l'accesso al sistema, oltre che raggiungere le risorse web pubbliche per permettere il caricamento del template della pagina web.

4.1.12 [AC-17] Remote Access

Riguarda la definizione di controlli che veicolano l'accesso di utenti al sistema da remoto, ossia attraverso la rete Internet. In particolare sono stati limitati gli access point alla rete interna, permettendo di restringere la superficie di attacco e quindi esponendo unicamente la porta 80 e 8443 per la comunicazione all'applicazione Spring. Inoltre tutte le comunicazioni con host remoti sono crittate con il protocollo di encryption (TLS) per garantire confidenzialità ed integrità.

[*TO MODERATE*] Un enhancement previsto per il livello moderate, non implementato, è quello che prevede il monitoring automatico degli accessi da remoto non autorizzati.

4.1.13 [AC-18, AC-19, AC-20, AC-21, AC-22]

Tutte le sezioni citate non riguardano funzionalità offerte dal sistema in esame.

4.2 Identification and Authentication [IA]

4.2.1 [IA-1] Identification and Authentication Policy and Procedures

Riguarda, in generale, la definizione di policies e procedure per l'effettiva implementazione dei security controls relativi all'identificazione e autenticazione.

4.2.2 [IA-2] Identification and Authentication

Riguarda l'implementazione di meccanismi atti a identificare univocamente ed autenticare le entità che interagiscono con il sistema (macchine e utenti). In particolare, vengono ampiente sfruttati i meccanismi messi a disposizione dal framework Spring Security per l'autenticazione end-user, e meccanismi offerti da Vault Server, a supporto dell'autenticazione dell'Application Server ai diversi servizi di back-end.

[TO LOW] Per attestarci almeno ad un livello LOW dovrebbe essere implementato un meccanismo di autenticazione multi-fattore. Non sono stati implementati tutti gli altri enhancement collegati, che ricadono nella baseline MODERATE, come ad esempio il controllo che riguarda la recapitazione dei fattori di autenticazione su dispositivi diversificati.

4.2.3 [IA-3] Device Identification and Authentication

[TO MODERATE] Riguarda l'implementazione di meccanismi di autenticazione a livello di device, ad esempio con MAC Address o IP, che non sono stati implementati.

4.2.4 [IA-4] Identifier Management

Riguarda l'applicazione e l'utilizzo di identificativi univoci, che prevengano anche il riuso, per identificare univocamente utenti, gruppi o ruoli, o in generale le entità che interagiscono con il sistema, come anche i servizi di back-end. Ciò è soddisfatto, ad esempio, rendendo univoci gli username degli utenti.

4.2.5 [IA-5] Authenticator Management

Riguarda, in generale, la gestione degli *authenticators*, intesi come qualsiasi oggetto atto ad autenticare un entità al sistema, quali ad esempio password, certificati o token. La gestione, nel caso specifico, è operata in modo oculato, ad esempio, memorizzando le password in modo crittato, mantenendo i certificati in *keystore* opportunamente protetti, le cui password di accesso sono, a loro volta, crittate nei files di configurazione, o ancora gestendo dinamicamente i token di accesso ai servizi di back-end, tramite il servizio, secure by design, offerto da Vault.

[TO MODERATE] Modalità più complesse di autenticazione dell'end-user come, ad esempio, quella PKI-based, con l'utilizzo dei certificati, e di identificazione come, ad esempio, la verifica fisica dell'identità di un soggetto, non sono state previste.

4.2.6 [IA-6] Authenticator Feedback

Prevede l'implementazione di meccanismi che permettano di evitare un feedback diretto conseguente all'inserimento delle credenziali nelle interfacce utente, come ad esempio l'oscuramento delle password all'atto dell'inserimento tramite 'asterischi'.

4.2.7 [IA-7, IA-8]

Le sezioni citate non riguardano funzionalità offerte dal sistema in esame.

4.2.8 [IA-9] Service Identification and Authentication

Riguarda, in architetture orientate ai servizi, meccanismi a supporto dell'identificazione e l'autenticazione dei servizi stessi. Ciò è soddisfatto tramite l'implementazione di meccanismi che permettano la mutua autenticazione tra i servizi di back-end tramite certificati trusted, che vengono trasmessi, ricevuti e validati.

4.3 System and Communications [SC]

4.3.1 [SC-1] System and Communication Protection Policy and Procedures

Riguarda, in generale, la definizione di policies e procedure per la protezione dei canali di comunicazione, tenendo conto della distribuzione dell'architettura e dei rischi rilevati, in funzione del contesto operativo.

4.3.2 [SC-2] Application Partitioning

Riguarda, dal punto di vista implementativo e di deploy, la separazione delle funzionalità utente, che rispecchiano i casi d'uso implementati dal sistema, dalle funzionalità di system management, quali ad esempio le interfacce di configurazione dei diversi servizi security enforcement, o dei servizi di back-end a supporto dell'application server. Ciò è ampiamente soddisfatto, dato l'isolmento dell'application server, che offre le funzionalità utente, da tutti gli altri servizi di back-end e di system management (LDAP, MySQL, Vault).

4.3.3 [SC-4] Information in Shared Resources

[TO MODERATE] Dovrebbero essere implementati meccanismi espliciti per prevenire trasferimenti non autorizzati tramite le risorse condivise del sistema, sia da parte di utenti che di processi.

4.3.4 [SC-5] Denial of Service Protection

[TO LOW] Per soddisfare la baseline LOW, dovrebbe essere applicato questo controllo, che prevede l'implementazione meccanismi atti a proteggere il sistema da attacchi che minano alla disponibilità dei servizi offerti dal sistema stesso.

4.3.5 [SC-7] Boundary Protection

Prevede la progettazione di un'architettura del sistema complessivo, con ben definiti confini esterni e una ben definita struttura interna della rete. Ciò è stato soddisfatto, adottando un approccio alla progettazione che include, ad esempio, la definizione delle porte esposte al pubblico, attraverso le quali il sistema può connettersi ad Internet e comunicare con l'esterno, e la definizione di una subnet privata, non accessibile al pubblico, attraverso la quale i servizi che costituiscono il sistema comunicano tra loro.

[TO MODERATE] Alcuni enhancement per coprire la baseline moderate prevedono, ad esempio, la limitazione del numero di connessioni provenienti dalla rete esterna, oppure di utilizzare politiche di connessione *deny all by default*, che negano l'accesso alla rete a tutti, e permettano invece *by exception*, cioè prevedendo delle eccezioni a tutti e soli i soggetti autorizzati.

4.3.6 [SC-8] Trasmission Confidentiality and Integrity

Prevede l'implementazione di meccanismi atti a proteggere la confidenzialità e l'integrità delle informazioni trasmesse. Ciò è ampiamente soddisfatto mediante l'applicazione e l'utilizzo di protocolli di crittografia a livello trasporto (TLS), che garantiscono entrambi i requisiti.

4.3.7 [SC-10] Network Disconnect

[TO MODERATE] Dovrebbero essere previsti meccanismi che permettano la terminazione delle connessioni instaurate tra i nodi, al termine di una sessione di comunicazione o dopo un certo periodo di inattività.

4.3.8 [SC-12] Cryptographic Key Establishment and Management

Questo controllo prevede che siano definite policies e procedure opportune per il management delle chiavi crittografiche, durante tutto il loro ciclo di vita, che comprende la loro generazione, distribuzione, storage e accesso. Ciò include anche la predisposizione di opportuni *keystore* protetti, per conservare le coppie crittografiche in maniera sicura e integra.

4.3.9 [SC-13] Cryptographic Protection

Riguarda l'implementazione di meccanismi opportuni per la crittografia di dati sensibili o informazioni che devono essere accedute in maniera sicura. Ad esempio, viene sfruttato l'algoritmo RSA per la generazione di coppie crittografiche (chiave, certificato), per l'autenticazione dei servizi di back-end, l'algoritmo MD5 e DES per la crittazione delle password nei file di configurazione, e la suite di Vault Encryption (AES-256) per la crittazione dei dati sensibili nel DB.

4.3.10 [SC-15] Collaborative Computing Devices

La sezione citata non riguarda funzionalità offerte dal sistema in esame.

4.3.11 [SC-17] Public Key Infrastructure Certificate

Riguarda l'utilizzo di infrastruttura a chiave pubblica, e quindi la generazione di certificati *trusted*, firmati da una Certification Authority riconosciuta. Questo tipo di controllo è stato simulato, emettendo dei certificati che sono stati firmati da una CA trusted, generata localmente, avente un certificato autofirmato. In produzione, dovrebbero essere rilasciati dei certificati con validità universalmente riconosciuta.

4.3.12 [SC-18] Mobile Code

[TO MODERATE] Dovrebbero essere effettuati controlli rigorosi relativi a codice, contenuto nei componenti di progetto, che potrebbe potenzialmente causare danni al sistema se utilizzato in modo malevolo. Questo comprende, ad esempio, codice JavaScript o AJAX, incluso tra le librerie del sistema in esame (template grafico), ma non certificato come *trusted*.

4.3.13 [SC-19] VOIP

La sezione citata non riguarda funzionalità offerte dal sistema in esame.

4.3.14 [SC-20, SC-21, SC-22] Secure Name/Address Resolution Service

[TO LOW] Il sistema dovrebbe prevedere meccanismi di autenticazione e verifica dell'integrità, quando viene utilizzato un sistema di risoluzione dei nomi o degli indirizzi (DNS). Nel caso specifico, non è stato previsto un meccanismo di risoluzione, dato che allo stato attuale si è reputato sufficiente raggiungere il sistema tramite l'IP dell'Application Server.

4.3.15 [SC-23] Session Authenticity

Prevede dei meccanismi atti a proteggere l'autenticità della sessione di comunicazione, garantendo confidenzialità per entrambi gli end-point. Ciò è assicurato implementando mutua autenticazione a livello trasporto, mediante l'utilizzo di certificati, per tutte le connessioni tra i servizi di back-end del sistema. Per quanto riguarda, invece, la comunicazione tra i client e l'application server, non si è ritenuto opportuno implementare lo stesso meccanismo, data l'impossibilità di identificare a priori tutti i client che vogliano instaurare una comunicazione con l'application server. Tuttavia, per mitigare la minaccia di attacchi di tipo MITM e Session Hijacking, sono stati implementati opportuni controlli applicativi, come descritto in precedenza.

4.3.16 [SC-28] Protection of Information *'at rest'*

Questo controllo definisce la necessità di implementare opportuni meccanismi atti a proteggere la confidenzialità e l'integrità dei dati *'at rest'*. Ciò è ampiamente soddisfatto mediante l'uso del servizio di *transit e storage*, offerto da Vault.

4.3.17 [SC-39] Process Isolation

Richiede la separazione dei contesti di esecuzione per ogni processo nel sistema complessivo. Ciò è stato ampiamente soddisfatto, mediante l'utilizzo dei container Docker, ciascuno dei quali possiede il proprio spazio isolato di esecuzione, ed esegue in maniera indipendente ciascuno dei servizi che compongono il sistema.

4.4 System and Information Integrity [SI]

Data la scarsa pertinenza dei controlli relativi a questa classe, in relazione al contesto didattico del progetto, non si è ritenuto mandatorio assestarsi completamente alla baseline LOW per questa categoria.

4.4.1 [SI-1] System and Information Integrity Policy and Procedures

Riguarda, in generale, la definizione di policies e procedure per la salvaguardia dell'integrità delle informazioni del sistema.

4.4.2 [SI-2] Flaw Remediation

Tale controllo prescrive di identificare e riportare eventuali difetti del sistema, intesi come potenziali vulnerabilità, tentando di mitigarli, mediante l'applicazione di *aggiornamenti*, *patches*, *service packs* ai componenti software che lo compongono. Ciò è stato realizzato in fase di vulnerability scanning nella quale, dopo aver identificato le vulnerabilità relative ai threats emersi e alle tecnologie utilizzate, mediante un'analisi semi-automatizzata, ci si è impegnati per risolverle, sia implementando *patches* di sicurezza, che aggiornando i componenti tecnologici a versioni più recenti. In tal modo, gran parte delle vulnerabilità relative alle tecnologie sono state risolte.

4.4.3 [SI-3, SI-4, SI-5]

[TO LOW] Non sono stati previsti controlli per l'identificazione e l'eventuale eradicazione di codice malevolo, per il monitoraggio di eventuali attacchi o accessi non autorizzati al sistema, nè notifiche di alcun tipo a tale scopo.

4.4.4 [SI-7] Software, Firmware and Information Integrity

[TO MODERATE] Non sono stati implementati tools atti alla verifica dell'integrità del sistema, e quindi che rilevano eventuali modifiche non autorizzate del software, firmware e, in generale, dell'informazione.

4.4.5 [SI-10] Informaton Input Validation

Questo controllo prevede che gli input forniti dall'utente al sistema informativo debbano essere preventivamente validati per evitare attacchi di tipo *injection* e *cross-site scripting*. Tale requisito risulta completamente soddisfatto, come discusso in precedenza.

4.4.6 [SI-11] Error Handling

Questo controllo prevede la gestione dei messaggi d'errore forniti all'utente, senza rivelazione di alcuna informazione che possa essere sfruttata da un potenziale attaccante per un exploit. Tale requisito è stato garantito nel sistema in esame, data la genericità dei messaggi d'errore proposti all'utente per gestire le eccezioni.

4.4.7 [SI-12] Information Handling and Retention

[TO LOW] Tale controllo, che appare incentrato su politiche di tipo giuridico e organizzativo, si considera non pertinente ai fini dell'elaborato

4.4.8 [SI-16] Memory Protection

Questo controllo prevede di realizzare meccanismi espliciti per la gestione di eventuali attacchi aventi lo scopo di eseguire codice in regioni protette della memoria, esterne a quella allocata per il processo. Tale meccanismo risulta automaticamente fornito, data la remota possibilità di realizzare attacchi alla memoria in ambiente Java.

5. Riferimenti

- “Identification and Implementation of Authentication and Authorization Patterns in the Spring Security Framework” – Aleksander Dikanski, Roland Steinegger, Sebastian Abeck
- Spring Security Reference
- Spring Vault Reference
- Vault Project Documentation
- Spring Security – Third Edition – Mick Knutson, Robert Winch, Peter Mularien
- Common Vulnerabilities and Exposure (CVE) – MITRE