Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

Progettazione e implementazione di un meccanismo per associare access token OAuth al certificato X.509 di un client

Relatore:
Prof. Ozalp Babaoglu

Candidato:
Ivan De Simone

Correlatore:
Prof. Francesco Giacomini

II Sessione
Anno Accademico 2024/2025

Alla mia famiglia,
che ha sempre creduto in me.

Introduzione

Esistono diversi sistemi per garantire l'accesso sicuro a risorse protette all'interno di un'infrastruttura distribuita. Tra questi, uno dei più diffusi è basato sull'utilizzo di access token OAuth, spesso sotto forma di JSON Web Token (JWT), che agiscono come credenziali temporanee, consentendo a un soggetto autenticato di accedere a determinate risorse senza dover ripetere il processo di autenticazione ad ogni richiesta.

In una tipica architettura client-server, in cui il processo di autenticazione e autorizzazione si fonda sull'utilizzo di access token, garantire l'autenticità e la validità di questi ultimi è cruciale per la sicurezza. Se un attaccante riuscisse ad entrare in possesso di un access token in corso di validità, potrebbe ottenere l'accesso a risorse protette del server, impersonando un utente legittimo.

Per mitigare questo tipo di attacco, sono state proposte diverse soluzioni volte a legare in modo univoco l'access token all'entità a cui è stato originariamente rilasciato. Tra gli approcci più significativi vi sono quelli descritti nel RFC 8705 (OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens) e nel RFC 9449 (OAuth 2.0 Demonstrating Proof of Possession). Il primo sfrutta il protocollo mutual TLS per vincolare il token al certificato del client, mentre il secondo introduce un meccanismo analogo basato su firme digitali a livello applicativo.

Questa tesi si concentra sull'implementazione e sulla valutazione del primo di questi due approcci, in quanto all'apparenza meno intrusivo, dimostrando la fattibilità tecnica dell'integrazione del RFC 8705 all'interno di un prodotto software esistente.

La motivazione principale di questo lavoro nasce dall'esigenza di prevenire accessi non autorizzati a risorse dovuti al furto di access token in sistemi di calcolo distribuiti su larga scala, quale è ad esempio la Worldwide LHC Computing Grid, utilizzata per la gestione dei dati prodotti dagli esperimenti di Fisica delle Alte Energie.

Indice

Introduzione **i**

1 Contesto tecnologico **1**

1.1 INDIGO IAM 1

1.1.1 IAM Login Service 3

1.1.2 Dashboard 3

1.2 Tecnologie coinvolte 3

1.2.1 NGINX 4

1.2.2 OAuth 2.0 4

1.2.3 Certificati X.509 5

1.2.4 Mutual TLS 6

2 Modifiche al sistema **7**

2.1 Implementazioni effettuate 7

2.1.1 IAM Login Service 7

2.1.2 Dashboard 9

2.1.3 NGINX 10

2.2 Impatto delle modifiche 10

3 Test e validazione **13**

3.1 Ambiente di validazione 13

3.2 Test effettuati 15

Conclusioni e sviluppi futuri **19**

A Modifiche ai sorgenti **21**

A.1 IAM Login Service 21

A.2 Dashboard 29

Bibliografia **37**

Elenco delle figure

1.1 Diagramma dell'architettura attuale 2
1.2 Diagramma di sequenza OAuth 2.0 5

2.1 Access token senza hash del certificato 8
2.2 Access token con hash del certificato 9
2.3 `git diff --stat` su IAM Login Service 11
2.4 `git diff --stat` sulla dashboard 12

3.1 Schema dell'ambiente di validazione 14

Capitolo 1

Contesto tecnologico

In questo capitolo vengono introdotte le tecnologie coinvolte nell'implementazione del RFC 8705 (OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens) [1].

1.1 INDIGO IAM

INDIGO IAM [2] è un servizio di Identity and Access Management sviluppato dal CNAF dell'Istituto Nazionale di Fisica Nucleare (INFN) e mantenuto dalla comunità Worldwide LHC Computing Grid (WLCG) [3]. Nato con l'obiettivo di offrire un'infrastruttura di autenticazione e autorizzazione flessibile e conforme agli standard, è progettato per soddisfare le esigenze delle comunità scientifiche europee. In un contesto distribuito, INDIGO IAM fornisce i meccanismi necessari per l'autenticazione degli utenti, la certificazione dei diritti di accesso alle risorse e la gestione centralizzata dei privilegi.

Per la delega dell'autorizzazione, INDIGO IAM implementa il protocollo OAuth 2.0 [4], il quale definisce tre elementi fondamentali:

- Authorization Server (AS), responsabile del rilascio e della validazione degli access token;
- Resource Server (RS), che espone le risorse protette e ne consente l'accesso in presenza di un access token valido;

- Client, che richiede e utilizza gli access token per accedere alle risorse protette per conto dell'utente.

L'integrazione del RFC 8705 introduce un meccanismo in cui, al momento dell'emissione dell'access token, viene incluso al suo interno l'hash del certificato del client. In fase di utilizzo del token da parte del client, tale hash è utilizzato per validare l'accesso alle risorse protette. Al fine di limitare l'impatto della sperimentazione a software sotto il controllo del CNAF, per tutti e tre i ruoli OAuth 2.0 sono stati individuati componenti che fanno parte di INDIGO IAM, ovvero il token issuer (AS), le API esposte (RS) e la dashboard web (client).

L'architettura complessiva del sistema è schematizzata in Figura 1.1.

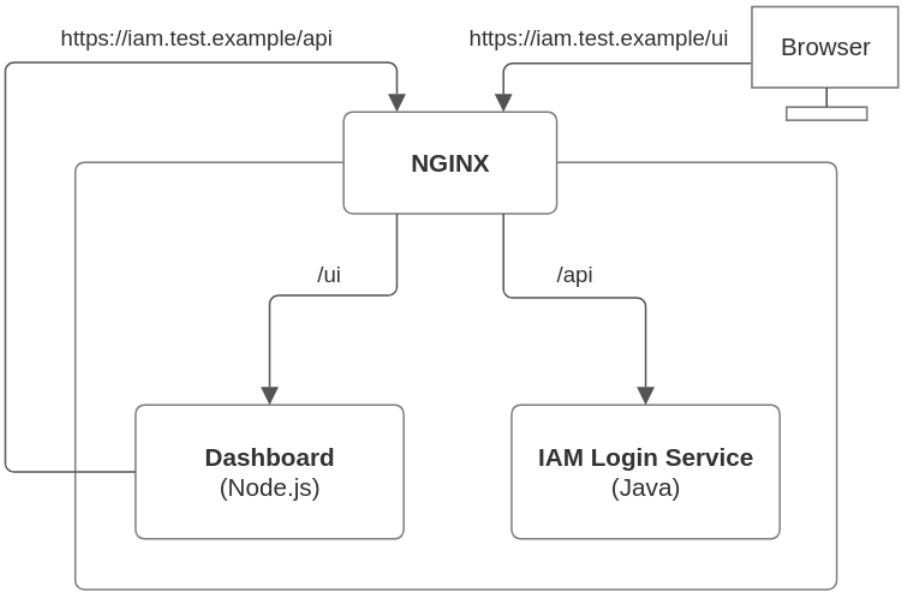

Figura 1.1: Diagramma dell'architettura attuale. Gli utenti, tramite il browser, ed i client, interagiscono con NGINX, il quale direziona le richieste verso la dashboard (`/ui`) oppure verso IAM Login Service (`/api`).

1.1.1 IAM Login Service

IAM Login Service costituisce il componente centrale di INDIGO IAM, e implementa sia la funzionalità di Authorization Server che quella di Resource Server. In particolare, il servizio si occupa di gestire l'autenticazione degli utenti, emettere gli access token e validare le richieste provenienti dai client registrati.

Sviluppato in Java utilizzando il framework Spring Boot [5], IAM Login Service espone un insieme di API REST che permettono la gestione delle identità, dei gruppi e delle politiche di autorizzazione.

Nel contesto di questa tesi, IAM Login Service svolge sia il ruolo di token issuer, rilasciando l'access token dopo che l'utente si è autenticato e ha dato il consenso, sia quello di server delle risorse, esponendo le API e autorizzandone l'accesso in base al contenuto dell'access token.

1.1.2 Dashboard

La dashboard web è un client del servizio INDIGO IAM. Essa fornisce un'interfaccia grafica che consente agli utenti di interagire con il sistema e di accedere alle risorse messe a disposizione da INDIGO IAM. Tramite la dashboard è quindi possibile amministrare gli utenti, i gruppi ed i propri client registrati, rendendo più agevole la gestione complessiva del sistema.

La dashboard è un'applicazione web, sviluppata tramite il framework Next.js [6], che funge da client OAuth 2.0, agendo per conto dell'utente autenticato. Essa utilizza gli access token emessi da IAM Login Service per accedere alle risorse protette (API), allegandoli alle richieste effettuate verso il Resource Server.

Nell'ambito di questo elaborato, si è deciso di utilizzare la dashboard per comprendere l'impatto delle modifiche dovute all'adozione del RFC 8705 nei client esistenti.

1.2 Tecnologie coinvolte

L'implementazione del RFC 8705 si basa su tecnologie e protocolli ampiamente diffusi nel panorama informatico. Di seguito vengono descritti il software NGINX [7], il pro-

tocollo OAuth 2.0, lo standard X.509 [8] per i certificati digitali ed il protocollo mutual TLS [9].

1.2.1 NGINX

NGINX è un software open source sviluppato per agire come HTTP web server, reverse proxy, load balancer, proxy server per i protocolli TCP/UDP e mail proxy server.

Nel presente lavoro, viene impiegato come reverse proxy, esponendo un'unica porta di accesso al sistema e instradando successivamente il traffico in ingresso verso i servizi IAM Login Service e dashboard. Inoltre, NGINX espone il certificato X.509 server, consentendo di applicare TLS termination. Tale approccio permette di utilizzare il protocollo HTTP per la comunicazione tra il reverse proxy e i servizi interni, mantenendo un ambiente isolato e sicuro. L'adozione di NGINX consente di semplificare la configurazione dei singoli componenti applicativi, centralizzare la gestione delle connessioni e migliorare la sicurezza complessiva del sistema.

1.2.2 OAuth 2.0

OAuth 2.0 è un protocollo per la delega di autorizzazione, progettato per consentire a un'applicazione client di ottenere accesso limitato a risorse protette. Il funzionamento del protocollo si basa sull'emissione di un access token da parte di un Authorization Server, che l'applicazione client utilizza per accedere alle risorse fornite da un Resource Server. Questo meccanismo permette di effettuare il Single Sign-On (SSO), ovvero l'accesso a differenti servizi mediante un unico insieme di credenziali.

Nel caso di INDIGO IAM, il flusso utilizzato dalla dashboard per ottenere un access token segue il modello di "Authorization Code Grant". In Figura 1.2 è illustrato tale flusso:

1. la dashboard (client) direziona il browser verso la pagina di autenticazione di IAM Login Service, allegando alla richiesta l'indirizzo di ritorno;

2. l'utente si autentica con le proprie credenziali e fornisce il consenso;

3. il browser viene reindirizzato sulla dashboard, ottenendo un codice temporaneo;

4. la dashboard invia all'Authorization Server il codice ottenuto e riceve in cambio l'access token;

5. la dashboard include l'access token nelle richieste verso il Resource Server.

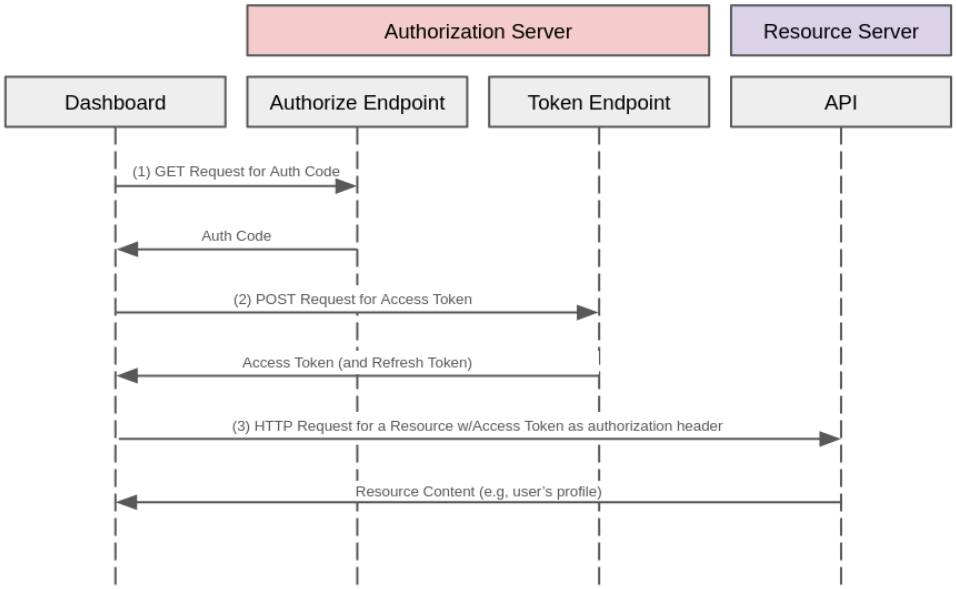

Figura 1.2: Flusso "Authorization Code Grant" di autenticazione e autorizzazione OAuth 2.0 (Copyright INFN)

1.2.3 Certificati X.509

I certificati digitali X.509 costituiscono uno degli standard più diffusi per l'associazione di una chiave pubblica a un soggetto, che può essere un server o un client. Ogni certificato è firmato da una Certificate Authority (CA) che ne garantisce l'autenticità e l'integrità. All'interno del protocollo Transport Layer Security (TLS), tali certificati vengono utilizzati per stabilire connessioni cifrate sicure, verificando l'identità delle parti coinvolte nella comunicazione.

I certificati X.509 possono essere rappresentati in diversi formati di codifica, tra cui i più comuni sono PEM e DER. Il formato Privacy Enhanced Mail (PEM) utilizza una

codifica Base64 [10], racchiusa tra intestazioni testuali, che lo rende al contempo leggibile e portabile. Il formato Distinguished Encoding Rules (DER), invece, rappresenta la versione binaria del certificato, più compatta rispetto alla controparte PEM.

Ai fini di questa tesi, i certificati sono stati generati da una Certificate Authority di test tramite la suite OpenSSL [11], con lo scopo di simulare un'infrastruttura di fiducia completa senza dipendere da CA pubbliche.

1.2.4 Mutual TLS

Il Transport Layer Security (TLS) è un protocollo crittografico progettato per garantire la riservatezza, l'integrità e l'autenticità delle comunicazioni tra due entità. Durante la fase di handshake, il server presenta al client un certificato digitale che consente di verificarne l'identità e di stabilire una connessione cifrata.

Mutual TLS (mTLS) estende questo meccanismo introducendo un'autenticazione reciproca tra client e server. Oltre a validare il certificato del server, il client presenta a sua volta un certificato digitale. In questo modo, le parti sono in grado di stabilire una connessione cifrata in cui l'identità di entrambe è verificata.

Capitolo 2

Modifiche al sistema

In questo capitolo sono descritte le modifiche dei componenti software di INDIGO IAM per l'implementazione del RFC 8705.

2.1 Implementazioni effettuate

L'applicazione del RFC 8705 ha richiesto interventi mirati sui principali componenti del sistema INDIGO IAM, al fine di introdurre il supporto per l'autenticazione tramite mutual TLS. Le modifiche hanno interessato IAM Login Service, la dashboard e la configurazione di NGINX.

2.1.1 IAM Login Service

Le modifiche apportate a IAM Login Service hanno riguardato:

1. l'estensione del meccanismo di creazione degli access token, al fine di inserire al loro interno un hash del certificato del client;

2. l'aggiunta di un passaggio di verifica in fase di validazione del token, con l'obiettivo di controllare tale hash.

Gli interventi hanno interessato rispettivamente le componenti Authorization Server e Resource Server.

Per quanto riguarda l'Authorization Server, è stato necessario modificare il processo di emissione degli access token in modo da includere al loro interno l'hash crittografico del certificato del client. In particolare, al momento della generazione del token, viene calcolato l'hash SHA-256 del certificato client in formato DER, il cui valore è poi inserito nel campo `x5t#S256` del claim `cnf` (confirmation claim). L'emissione del token conforme al RFC 8705 è consentita esclusivamente in presenza di un certificato valido presentato dal client. Qualora il certificato non sia disponibile, il servizio IAM Login Service rilascia un access token sprovvisto del claim `cnf`, e quindi del valore hash (si veda un esempio in Figura 2.1). Se invece il certificato è correttamente fornito e validato, viene inoltrato dal reverse proxy a IAM Login Service tramite header HTTP. A questo punto, il certificato viene convertito dal formato PEM al formato DER e successivamente sottoposto alla funzione di hash SHA-256. Il risultato, codificato in Base64, viene infine incluso nell'access token. In Figura 2.2 è mostrato un esempio di access token conforme al RFC 8705.

```
{
  "iss": "https://iam.test.example:8443",
  "sub": "73f16d93-2441-4a50-88ff-85360d78c6b5",
  "iat": 1760015939,
  "exp": 1760019539,
  "client_id": "30ce96a7-de5e-4282-bf11-16be13621e45",
  "scope": "openid profile iam:admin.read scim:read scim:write
    iam:admin.write email",
  "jti": "97de228e-7989-4f24-a2b9-84e448fe0606"
}
```

Figura 2.1: Esempio di access token rilasciato da IAM Login Service, senza l'hash del certificato del client

Per la parte relativa al Resource Server, è stato implementato un controllo di corrispondenza dell'hash nella fase di validazione dei token: quando un client richiede l'accesso a una risorsa protetta, il server estrae il certificato del client dagli header HTTP, lo converte in formato DER e ne calcola l'hash SHA-256, codificando il risultato in Base64. Il valore così ottenuto viene confrontato con quello presente nel campo `x5t#S256` del-

```
{
  "iss": "https://iam.test.example:8443",
  "sub": "73f16d93-2441-4a50-88ff-85360d78c6b5",
  "iat": 1760015939,
  "exp": 1760019539,
  "client_id": "30ce96a7-de5e-4282-bf11-16be13621e45",
  "scope": "openid profile iam:admin.read scim:read scim:write
    iam:admin.write email",
  "jti": "97de228e-7989-4f24-a2b9-84e448fe0606",
  "cnf": {
    "x5t#S256": "Qircx-blDMNYi5OhGMe5n23gCiUmy6i-kFJI2iUIe6U"
  }
}
```


Figura 2.2: Esempio di access token rilasciato da IAM Login Service, contenente l'hash del certificato del client

l'access token utilizzato per l'accesso alla risorsa. La richiesta viene autorizzata se e solo se i due valori coincidono, e la risorsa viene restituita al client. In caso di mancata corrispondenza o di errore nel processo di verifica, il server risponde con un codice `401 - Unauthorized`, negando l'accesso alla risorsa.

2.1.2 Dashboard

L'integrazione di mutual TLS all'interno della dashboard richiede l'utilizzo di un certificato client per l'instaurazione della connessione con il servizio IAM Login Service. Nel software originale, il supporto al protocollo OAuth 2.0 è fornito dalla libreria Auth.js [12], la quale non prevede nativamente la possibilità di effettuare connessioni mTLS. Per superare tale limitazione, sono state valutate due alternative: la modifica locale della libreria tramite un fork oppure l'implementazione manuale del flusso di autenticazione. Ai fini di questa tesi è stata scelta la seconda soluzione, privilegiando la semplicità e la rapidità dell'implementazione, con l'obiettivo di dimostrare la fattibilità dell'intervento.

Come descritto nella sezione 1.2.2, l'esecuzione del flusso OAuth 2.0 "Authorization Code Flow" prevede l'invio di richieste tramite il protocollo HTTP. Per implementare manualmente tale flusso è stata impiegata l'API "fetch" di Node.js [13], alla quale è stato affiancato un HTTP Agent [14] dedicato, responsabile della gestione della connessione mutual TLS e della presentazione del certificato client. La connessione mTLS è obbligatoria per la richiesta al Token Endpoint, da cui viene ottenuto l'access token, e per tutte le interazioni successive con il Resource Server nel caso di accesso a risorse protette (si veda la Figura 1.2).

Per garantire la compatibilità con il resto dell'applicazione, il flusso OAuth 2.0 manuale è stato implementato preservando l'interfaccia esposta dalla libreria Auth.js. È stato inoltre necessario fornire alla dashboard un certificato X.509 client valido, al fine di consentire la corretta instaurazione della connessione mTLS con il server.

2.1.3 NGINX

In quanto interfaccia primaria del sistema verso l'esterno e unico punto di collegamento tra la dashboard e IAM Login Service (si veda la Figura 1.1), anche la configurazione del reverse proxy NGINX richiede alcune modifiche per supportare l'integrazione del meccanismo di mutual TLS.

L'intervento principale alla configurazione di NGINX ha riguardato l'abilitazione del meccanismo di autenticazione dei client e la definizione di un insieme di Certificate Authorities fidate, in modo da consentire la verifica dei certificati presentati dai client.

Poiché IAM Login Service necessita del certificato client per emettere l'access token e validare le richieste di accesso alle risorse, NGINX è stato configurato per inoltrare tale informazione all'applicazione. Come descritto in precedenza, questo avviene tramite l'inclusione di un apposito header HTTP nelle richieste direzionate verso tale destinazione, in particolare `X-SSL-Client-Cert`.

2.2 Impatto delle modifiche

Gli interventi descritti nei paragrafi precedenti sono stati progettati per estendere le funzionalità esistenti in INDIGO IAM, riducendo al minimo l'impatto sull'architettura e

sulla struttura del codice. Lato IAM Login Service, l'adozione delle specifiche introdotte dal RFC 8705 può essere realizzata con uno sforzo limitato, così come lato client, preservando la compatibilità con le implementazioni già conformi a OAuth 2.0. Il problema sussiste quando si utilizzano librerie che non supportano nativamente mutual TLS, ed è quindi necessario svolgere del lavoro consistente per integrare tali modifiche nelle librerie.

Dal punto di vista quantitativo, l'invasività delle modifiche può essere valutata attraverso l'analisi delle differenze nel codice sorgente, che permette di stimare il carico in termini di linee di codice effettivamente aggiunte o modificate. Osserviamo di seguito le informazioni fornite dal comando `git diff --stat`, prendendo come riferimento il commit di partenza e il commit che conclude l'implementazione, rispettivamente per IAM Login Service (Figura 2.3) e per la dashboard (Figura 2.4).

```
$ git diff 30790ac8 bccd985b --stat
 .../security/IamApiSecurityConfig.java       |   2 +
 .../filters/MtlsTokenBindingFilter.java      | 100 ++++++++++++++++++
 .../common/BaseAccessTokenBuilder.java       |  28 +++++
 .../it/infn/mw/iam/util/x509/X509Utils.java  |  17 +++
 4 files changed, 147 insertions(+)
```

Figura 2.3: Output del comando `git diff --stat` su IAM Login Service. Consultare l'appendice A.1 per l'elenco completo delle differenze introdotte nel codice sorgente.

Poiché INDIGO IAM è un progetto completamente open source, potrebbero esistere organizzazioni che adottano una diversa implementazione di IAM Login Service o della dashboard. In tali casi, l'integrazione delle specifiche previste dal RFC 8705 comporterebbe uno sforzo sostenibile, in quanto le modifiche richieste risultano circoscritte a componenti specifici. Per quanto riguarda il servizio IAM Login Service, l'intervento si concentra unicamente nei processi di emissione e validazione degli access token. Lato client, le modifiche si limitano all'aggiunta di un HTTP Agent per la presentazione del certificato durante le richieste.

```
$ git diff b8f595b 4a8ab45 --stat
 src/app/api/mtls/callback/route.ts | 16 ++++++
 src/app/mtls/page.tsx              | 36 ++++++++++++
 src/middleware.ts                  |  2 +-
 src/services/oauth-mtls.ts         | 84 ++++++++++++++++++++++++++++++
 src/services/users-mtls.ts         | 28 ++++++++++
 src/utils/fetch-mtls/index.ts      | 67 ++++++++++++++++++++++++
 6 files changed, 232 insertions(+), 1 deletion(-)
```

Figura 2.4: Output del comando `git diff --stat` sulla dashboard. Consultare l'appendice A.2 per l'elenco completo delle differenze introdotte nel codice sorgente.

Capitolo 3

Test e validazione

In questo capitolo vengono descritti i test effettuati per verificare la correttezza delle modifiche apportate.

3.1 Ambiente di validazione

Per la fase di test e validazione è stato predisposto un ambiente di esecuzione isolato basato su Docker Compose [15], all'interno del quale sono stati integrati i componenti software oggetto di modifica. Tale approccio ha permesso di ricreare una configurazione completa del sistema INDIGO IAM in modo controllato, garantendo al contempo semplicità di gestione e riproducibilità degli esperimenti.

Come è possibile vedere in Figura 3.1, l'ambiente di validazione comprende i seguenti servizi:

- IAM Login Service, contenente i sorgenti Java compilati ed eseguiti in modalità di sviluppo;
- Dashboard, contenente l'applicazione web eseguita in modalità di sviluppo;
- NGINX, configurato come reverse proxy;
- Database SQL, dedicato alla persistenza dei dati gestiti da IAM Login Service.

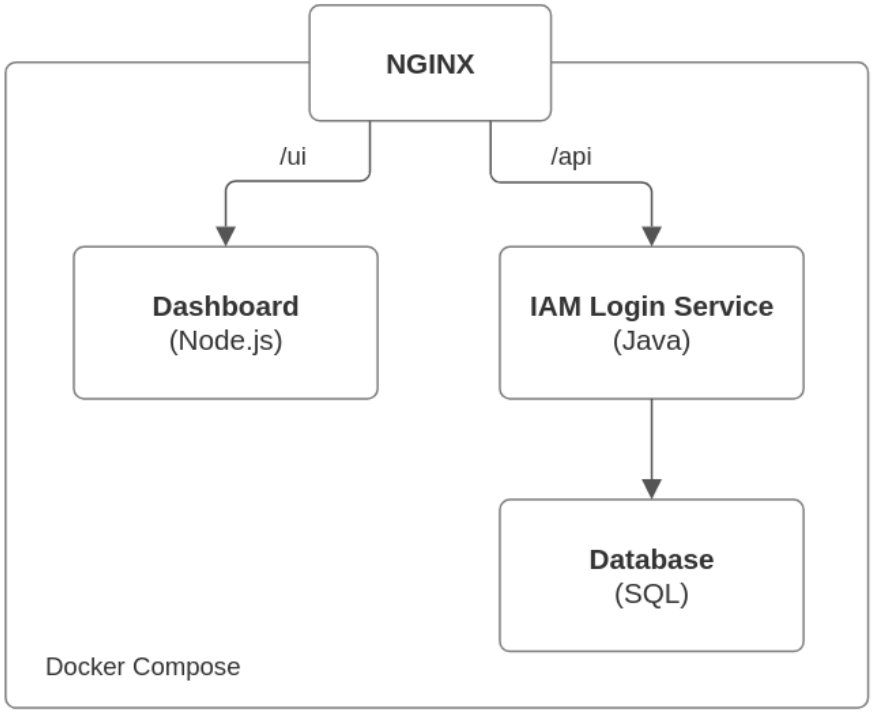

Figura 3.1: Schema dell'ambiente di validazione. Il reverse proxy NGINX consente l'interazione con i servizi IAM Login Service e dashboard. Il Database memorizza i dati di IAM Login Service.

L'immagine Docker [16] utilizzata per il deployment di INDIGO IAM in ambiente di sviluppo è stata generata a partire da un'immagine base Node.js, all'interno della quale è stato installato Java Temurin. Questa configurazione consente di avviare e arrestare rapidamente l'intero ambiente di test, semplificando la verifica funzionale delle modifiche apportate, e di simulare il networking all'interno di un contesto isolato, riducendo il rischio di interferenze con l'ambiente host.

3.2 Test effettuati

Al fine di verificare la correttezza e la robustezza dell'implementazione, sono stati eseguiti diversi test manuali, in aggiunta ai test automatizzati già presenti nel progetto. Gli scenari di prova sono stati progettati per coprire i principali casi d'uso previsti dal meccanismo di mutual TLS e per validare la corretta gestione dei certificati in fase di autenticazione (connessione a `/token`) e accesso alle risorse (connessione a `/api`).

Accesso alle risorse

Operazioni:

1. La dashboard si connette con IAM Login Service in mTLS, presentando un certificato valido.
2. La dashboard richiede una risorsa, stabilendo una connessione mTLS con lo stesso certificato.

Risultato:
`200 OK` - IAM Login Service autentica la richiesta e restituisce la risorsa richiesta.

Certificato non corrispondente

Operazioni:

1. La dashboard si connette con IAM Login Service in mTLS, presentando un certificato valido A.
2. La dashboard richiede una risorsa, stabilendo una connessione mTLS con un diverso certificato valido B.

Risultato:
`401 Unauthorized` - IAM Login Service rifiuta la richiesta a causa della mancata corrispondenza tra l'hash del certificato registrato nell'access token e quello presentato nella connessione mTLS.

Certificato scaduto durante l'autenticazione

Operazioni:

1. La dashboard tenta di connettersi con IAM Login Service in mTLS, presentando un certificato scaduto.

Risultato:
`495 SSL Certificate Error` - NGINX rifiuta la connessione a causa della non validità del certificato presentato in fase di autenticazione.

Certificato scaduto durante la richiesta di risorse

Operazioni:

1. La dashboard si connette con IAM Login Service in mTLS, presentando un certificato A valido.
2. La dashboard richiede una risorsa, stabilendo una connessione mTLS con lo stesso certificato A, nel frattempo scaduto.

Risultato:
`495 SSL Certificate Error` - NGINX rifiuta la connessione a causa della non validità del certificato in fase di richiesta delle risorse.

Certificato assente durante l'autenticazione

Operazioni:

1. La dashboard tenta di connettersi con IAM Login Service senza presentare alcun certificato.

Risultato:
`200 OK` - IAM Login Service emette un access token sprovvisto del claim `cnf`, e di conseguenza senza hash del certificato client.

Certificato assente durante la richiesta di risorse

Operazioni:

1. La dashboard si connette con IAM Login Service in mTLS, presentando un certificato valido.
2. La dashboard richiede una risorsa protetta senza presentare alcun certificato.

Risultato:
`401 Unauthorized` - IAM Login Service rifiuta la richiesta a causa dell'assenza del certificato necessario per la verifica del token.

Conclusioni e sviluppi futuri

In questa tesi è stata presentata l'implementazione e la validazione del meccanismo descritto nel RFC 8705 (OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens) all'interno del sistema INDIGO IAM. L'obiettivo principale era dimostrare la fattibilità tecnica dell'integrazione di mutual TLS in un'infrastruttura di autenticazione e autorizzazione esistente, valutandone l'impatto architetturale e sul codice sorgente.

La proof of concept ha mostrato che l'adozione delle specifiche sopracitate è possibile con modifiche limitate e non invasive, preservando la compatibilità con le funzionalità già conformi a OAuth 2.0. I test di validazione hanno confermato il corretto funzionamento del meccanismo di associazione tra access token e certificato X.509 client, nonché la capacità del sistema di rifiutare connessioni non conformi.

Poiché la sperimentazione è stata condotta in un ambiente controllato e con componenti opportunamente adattati, ulteriori sviluppi riguarderanno l'integrazione del protocollo mutual TLS nelle librerie pubbliche comunemente utilizzate, come ad esempio Auth.js, per favorirne l'adozione nei sistemi in produzione.

Un'evoluzione naturale del lavoro consiste inoltre nell'estendere l'implementazione ai flussi Token Exchange e Refresh Token, previsti dalle specifiche OAuth 2.0, al fine di supportare scenari più complessi e di lunga durata.

Appendice A

Modifiche ai sorgenti

In questa appendice sono riportate le differenze introdotte nel codice sorgente dei componenti IAM Login Service e dashboard, ottenute tramite l'esecuzione del comando `git diff`, prendendo come riferimento il commit di partenza e il commit che conclude l'implementazione.

A.1 IAM Login Service

Tutti i file elencati di seguito appartengono alla directory `iam/iam-login-service/src/main/java/it/infn/mw/iam`, che costituisce la struttura principale del codice sorgente di IAM Login Service.

`/config/security/IamApiSecurityConfig.java`

```
@@ -40,6 +40,7 @@ import org.springframework.security.web.context.
   ↪ SecurityContextPersistenceFilter

 import it.infn.mw.iam.api.proxy.ProxyCertificatesApiController;
 import it.infn.mw.iam.config.IamProperties;
+import it.infn.mw.iam.config.security.filters.MtlsTokenBindingFilter;
 import it.infn.mw.iam.config.security.IamWebSecurityConfig.
    ↪ UserLoginConfig;
 import it.infn.mw.iam.core.oauth.
    ↪ FormClientCredentialsAuthenticationFilter;
```

```
@@ -149,6 +150,7 @@ public class IamApiSecurityConfig {
              .accessDeniedHandler(new OAuth2AccessDeniedHandler())
          .and()
            .addFilterAfter(resourceFilter,
               ↪ SecurityContextPersistenceFilter.class)
+           .addFilterAfter(new MtlsTokenBindingFilter(),
   ↪ OAuth2AuthenticationProcessingFilter.class)
          .cors()
          .and()
          .sessionManagement()
```

Sorgente A.1: Modifiche al file IamApiSecurityConfig.java

/config/security/filters/MtlsTokenBindingFilter.java

```
@@ -0,0 +1,100 @@
+/**
+ * Copyright (c) Istituto Nazionale di Fisica Nucleare (INFN).
   ↪ 2016-2021
+ *
+ * Licensed under the Apache License, Version 2.0 (the "License");
+ * you may not use this file except in compliance with the License.
+ * You may obtain a copy of the License at
+ *
+ *     http://www.apache.org/licenses/LICENSE-2.0
+ *
+ * Unless required by applicable law or agreed to in writing, software
+ * distributed under the License is distributed on an "AS IS" BASIS,
+ * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   ↪ implied.
+ * See the License for the specific language governing permissions and
+ * limitations under the License.
+ */
+package it.infn.mw.iam.config.security.filters;
+
+import static it.infn.mw.iam.core.oauth.profile.common.
   ↪ BaseAccessTokenBuilder.CERT_HASH_FIELD_NAME;
```

```
+import static it.infn.mw.iam.core.oauth.profile.common.
   ↪ BaseAccessTokenBuilder.CLIENT_CERT_HEADER;
+import static it.infn.mw.iam.core.oauth.profile.common.
   ↪ BaseAccessTokenBuilder.CNF_CLAIM_NAME;
+import static it.infn.mw.iam.util.x509.X509Utils.
   ↪ getCertificateThumbprint;
+
+import java.io.IOException;
+import java.text.ParseException;
+import java.util.Map;
+import java.util.Optional;
+
+import javax.servlet.FilterChain;
+import javax.servlet.ServletException;
+import javax.servlet.http.HttpServletRequest;
+import javax.servlet.http.HttpServletResponse;
+
+import org.springframework.security.authentication.
   ↪ InsufficientAuthenticationException;
+import org.springframework.web.filter.OncePerRequestFilter;
+
+import com.nimbusds.jwt.JWTClaimsSet;
+import com.nimbusds.jwt.SignedJWT;
+
+public class MtlsTokenBindingFilter extends OncePerRequestFilter {
+
+  private static final String AUTH_HEADER = "Authorization";
+  private static final String CLIENT_HEADER = "X-Client";
+
+  @Override
+  protected void doFilterInternal(HttpServletRequest request,
   ↪ HttpServletResponse response,
+      FilterChain chain) throws ServletException, IOException {
+
+    try {
+      String client = request.getHeader(CLIENT_HEADER);
+
+      if (client == null || !client.equals("dashboard-mtls")) {
```

```
+        chain.doFilter(request, response);
+        return;
+      }
+
+      String presentedCert = request.getHeader(CLIENT_CERT_HEADER);
+
+      if (presentedCert == null || presentedCert.isBlank()) {
+        throw new InsufficientAuthenticationException("Missing mTLS
    ↪ certificate");
+      }
+
+      Optional<String> presentedThumbprint = getCertificateThumbprint(
    ↪ presentedCert);
+
+      if (presentedThumbprint.isEmpty()) {
+        throw new InsufficientAuthenticationException("Missing mTLS
    ↪ certificate thumbprint");
+      }
+
+      String auth = request.getHeader(AUTH_HEADER);
+
+      if (auth == null || !auth.startsWith("Bearer ")) {
+        throw new InsufficientAuthenticationException("Missing or
    ↪ invalid Authorization header");
+      }
+
+      String tokenValue = auth.substring("Bearer ".length()).trim();
+
+      try {
+        SignedJWT jwt = SignedJWT.parse(tokenValue);
+        JWTClaimsSet claims = jwt.getJWTClaimsSet();
+
+        Map<String, Object> cnf = claims.getJSONObjectClaim(
    ↪ CNF_CLAIM_NAME);
+
+        if (cnf != null && cnf.containsKey(CERT_HASH_FIELD_NAME)) {
+          String expectedThumbprint = cnf.get(CERT_HASH_FIELD_NAME).
    ↪ toString();
```

```
+          if (!expectedThumbprint.equals(presentedThumbprint.get())) {
+            throw new InsufficientAuthenticationException("mTLS
   ↪ certificate thumbprint mismatch");
+          }
+        } else {
+          throw new InsufficientAuthenticationException("Missing
   ↪ certificate thumbprint in token");
+        }
+
+      } catch (ParseException e) {
+        throw new InsufficientAuthenticationException("Invalid access
   ↪ token format");
+      }
+
+      chain.doFilter(request, response);
+    } catch (InsufficientAuthenticationException e) {
+      response.sendError(HttpServletResponse.SC_UNAUTHORIZED, e.
   ↪ getMessage());
+    }
+  }
+}
```

Sorgente A.2: Modifiche al file MtlsTokenBindingFilter.java

/core/oauth/profile/common/BaseAccessTokenBuilder.java

```
@@ -18,14 +18,18 @@ package it.infn.mw.iam.core.oauth.profile.common;
 import static com.google.common.base.Strings.isNullOrEmpty;
 import static it.infn.mw.iam.core.oauth.IamOAuth2RequestFactory.
    ↪ AUD_KEY;
 import static it.infn.mw.iam.core.oauth.granters.
    ↪ TokenExchangeTokenGranter.TOKEN_EXCHANGE_GRANT_TYPE;
+import static it.infn.mw.iam.util.x509.X509Utils.
   ↪ getCertificateThumbprint;
 import static java.util.Objects.isNull;

 import java.text.ParseException;
 import java.time.Instant;
```

```
 import java.util.Date;
 import java.util.Map;
+import java.util.Optional;
 import java.util.UUID;

+import javax.servlet.http.HttpServletRequest;
+
 import org.mitre.oauth2.model.OAuth2AccessTokenEntity;
 import org.mitre.oauth2.model.SavedUserAuthentication;
 import org.mitre.openid.connect.model.UserInfo;
@@ -34,6 +38,8 @@ import org.slf4j.LoggerFactory;
 import org.springframework.security.oauth2.common.exceptions.
    ↪ InvalidRequestException;
 import org.springframework.security.oauth2.provider.
    ↪ OAuth2Authentication;
 import org.springframework.security.oauth2.provider.OAuth2Request;
+import org.springframework.web.context.request.RequestContextHolder;
+import org.springframework.web.context.request.
   ↪ ServletRequestAttributes;

 import com.google.common.base.Splitter;
 import com.google.common.collect.Maps;
@@ -56,6 +62,9 @@ public abstract class BaseAccessTokenBuilder
   ↪ implements JWTAccessTokenBuilder {
   public static final String SCOPE_CLAIM_NAME = "scope";
   public static final String ACT_CLAIM_NAME = "act";
   public static final String CLIENT_ID_CLAIM_NAME = "client_id";
+  public static final String CNF_CLAIM_NAME = "cnf";
+  public static final String CERT_HASH_FIELD_NAME = "x5t#S256";
+  public static final String CLIENT_CERT_HEADER = "X-SSL-Client-cert";
   public static final String SPACE = " ";

   public static final String SUBJECT_TOKEN = "subject_token";
@@ -159,6 +168,12 @@ public abstract class BaseAccessTokenBuilder
   ↪ implements JWTAccessTokenBuilder {

     builder.claim(CLIENT_ID_CLAIM_NAME, token.getClient().getClientId
        ↪ ());
```

```
+    Optional<String> clientCertificateHash =
   ↪ getClientCertificateThumbprint();
+    if (clientCertificateHash.isPresent()) {
+      Map<String, Object> cnfValue = Map.of(CERT_HASH_FIELD_NAME,
   ↪ clientCertificateHash.get());
+      builder.claim(CNF_CLAIM_NAME, cnfValue);
+    }
+
     String audience = null;

     if (hasAudienceRequest(authentication)) {
@@ -193,4 +208,17 @@ public abstract class BaseAccessTokenBuilder
   ↪ implements JWTAccessTokenBuilder {
       builder.claim("acr", savedAuth.getAdditionalInfo().get("acr"));
     }
   }
+
+  private Optional<String> getClientCertificateThumbprint() {
+    HttpServletRequest request =
+        ((ServletRequestAttributes) RequestContextHolder.
   ↪ currentRequestAttributes()).getRequest();
+
+    String clientCert = request.getHeader(CLIENT_CERT_HEADER);
+
+    if (clientCert == null || clientCert.isBlank()) {
+      return Optional.empty();
+    }
+
+    return getCertificateThumbprint(clientCert);
+  }
 }
```

Sorgente A.3: Modifiche al file BaseAccessTokenBuilder.java

/util/x509/X509Utils.java

```
@@ -16,10 +16,13 @@
```

```
 package it.infn.mw.iam.util.x509;

 import java.io.ByteArrayInputStream;
+import java.security.MessageDigest;
+import java.security.NoSuchAlgorithmException;
 import java.security.cert.CertificateException;
 import java.security.cert.CertificateFactory;
 import java.security.cert.X509Certificate;
 import java.util.Base64;
+import java.util.Optional;

 import eu.emi.security.authn.x509.impl.X500NameUtils;
 import it.infn.mw.iam.api.scim.exception.ScimValidationException;
@@ -68,4 +71,18 @@ public class X509Utils {
     return getCertificateSubject(getX509CertificateFromString(
        ↪ certValueAsString));
   }

+  public static Optional<String> getCertificateThumbprint(String cert)
   ↪  {
+    String sanitized = cert.replace("-----BEGIN CERTIFICATE-----", "")
+      .replace("-----END CERTIFICATE-----", "")
+      .replaceAll("\\s", "");
+    try {
+      byte[] der = Base64.getDecoder().decode(sanitized);
+      byte[] sha256 = MessageDigest.getInstance("SHA-256").digest(der)
   ↪ ;
+      String hash = Base64.getUrlEncoder().withoutPadding().
   ↪ encodeToString(sha256);
+      return Optional.of(hash);
+    } catch (NoSuchAlgorithmException e) {
+      return Optional.empty();
+    }
+  }
+
 }
```

Sorgente A.4: Modifiche al file X509Utils.java

A.2 Dashboard

Tutti i file elencati di seguito appartengono alla directory `iam-dashboard/src`, che contiene il codice sorgente della dashboard.

`/app/api/mtls/callback/route.ts`

```
@@ -0,0 +1,16 @@
+// SPDX-FileCopyrightText: 2025 Istituto Nazionale di Fisica Nucleare
+//
+// SPDX-License-Identifier: EUPL-1.2
+
+import { retrieveToken } from "@/services/oauth-mtls";
+import { NextResponse } from "next/server";
+
+export async function GET(request: Request) {
+  const url = new URL(request.url);
+  const code = url.searchParams.get("code");
+  if (!code) {
+    throw new Error("cannot get authorization code");
+  }
+  await retrieveToken(code);
+  return NextResponse.redirect("https://iam.test.example:8443/ui/mtls
   ↪ ");
+}
```

Sorgente A.5: Modifiche al file route.ts

`/app/mtls/page.tsx`

```
@@ -0,0 +1,36 @@
+// SPDX-FileCopyrightText: 2025 Istituto Nazionale di Fisica Nucleare
+//
+// SPDX-License-Identifier: EUPL-1.2
+
+import { auth, decodeJwtPayload, login, logout } from "@/services/
   ↪ oauth-mtls";
```

```
+import { getUsersPageMtls } from "@/services/users-mtls";
+
+export default async function MutualTLS() {
+  const session = await auth();
+  const loggedIn = !!session;
+  const decodedAccessToken = loggedIn ? await decodeJwtPayload(session
   ↪ ) : undefined;
+  const username = decodedAccessToken?.name;
+  const welcomeMessage = username ? `Hello ${username}!` : "Welcome";
+
+  const users = loggedIn ? await getUsersPageMtls(10) : undefined
+
+  return (
+    <div className="font-sans">
+      <main className="w-fit mx-auto flex flex-col justify-between p
   ↪ -32 text-center">
+        <h1>{welcomeMessage}</h1>
+        <form action={loggedIn ? logout : login} className="mx-auto">
+          <button className="border border-gray-300 rounded px-6 py-2
   ↪ my-8 cursor-pointer" type="submit">
+            {loggedIn ? "Logout" : "Login"}
+          </button>
+        </form>
+        {loggedIn && <p>{decodedAccessToken.client_id}</p>}
+        {users && <>
+          <h2 className="my-4">First 10 users</h2>
+          {users.Resources.map(u =>
+            <p key={u.id}>{u.displayName}</p>
+          )}
+        </>}
+      </main>
+    </div>
+  );
+}
```

Sorgente A.6: Modifiche al file page.tsx

/middleware.ts

```
@@ -10,7 +10,7 @@ import { settings } from "@/config";
 const { BASE_PATH } = settings;

 export const config = {
-  matcher: ["/((?!api|_next/static|_next/image|favicon.ico|signin|
   ↪ signout).*)"],
+  matcher: ["/((?!api|_next/static|_next/image|favicon.ico|signin|
   ↪ signout|mtls).*)"],
 };

 export default auth(async req => {
```

Sorgente A.7: Modifiche al file middleware.ts

/services/oauth-mtls.ts

```
@@ -0,0 +1,84 @@
+// SPDX-FileCopyrightText: 2025 Istituto Nazionale di Fisica Nucleare
+//
+// SPDX-License-Identifier: EUPL-1.2
+
+"use server";
+
+import { cookies } from "next/headers";
+import { redirect } from "next/navigation";
+import { mtlsFetch } from "@/utils/fetch-mtls";
+
+const IAM_AUTHORITY_URL = process.env.IAM_AUTHORITY_URL as string;
+const IAM_CLIENT_ID = process.env.IAM_CLIENT_ID as string;
+const IAM_CLIENT_SECRET = process.env.IAM_CLIENT_SECRET as string;
+const IAM_SCOPES = process.env.IAM_SCOPES as string;
+const IAM_REDIRECT_URI = process.env.IAM_REDIRECT_URI as string;
+
+export async function decodeJwtPayload(token: string) {
+  return JSON.parse(atob(token.split(".")[1]));
+}
```

```
+
+export async function auth() {
+  const cookiesStore = await cookies();
+  const accessToken = cookiesStore.get("access_token");
+  if (!accessToken) {
+    return;
+  }
+  return accessToken.value;
+}
+
+export async function login() {
+  const { authorization_endpoint } = await getOpenidConfiguration();
+  if (!authorization_endpoint) {
+    throw new Error("authorization_endpoint not found");
+  }
+
+  redirect(
+    authorization_endpoint +
+      `?client_id=${IAM_CLIENT_ID}` +
+      `&response_type=code` +
+      `&redirect_uri=${IAM_REDIRECT_URI}` +
+      `&scope=${IAM_SCOPES}`
+  );
+}
+
+export async function logout() {
+  const cookiesStore = await cookies();
+  cookiesStore.delete("access_token");
+  redirect("/mtls");
+}
+
+export async function retrieveToken(code: string) {
+  const { token_endpoint } = await getOpenidConfiguration();
+  if (!token_endpoint) {
+    throw new Error("token_endpoint not found");
+  }
+
+  const authorization = Buffer.from(
```

```
+    `${IAM_CLIENT_ID}:${IAM_CLIENT_SECRET}`
+  ).toString("base64");
+
+  const formData = new URLSearchParams();
+  formData.append("code", code);
+  formData.append("grant_type", "authorization_code");
+  formData.append("client_id", IAM_CLIENT_ID);
+  formData.append("redirect_uri", IAM_REDIRECT_URI);
+
+  const response = await mtlsFetch(token_endpoint, {
+    method: "POST",
+    body: formData.toString(),
+    headers: {
+      authorization: `Basic ${authorization}`,
+      "content-type": "application/x-www-form-urlencoded",
+    }
+  });
+  const json: any = await response.json();
+  const cookiesStore = await cookies();
+  cookiesStore.set("access_token", json["access_token"]);
+}
+
+async function getOpenidConfiguration() {
+  const wellKnownEndpoint = `${IAM_AUTHORITY_URL}/.well-known/openid-
   ↪ configuration`;
+  const response = await fetch(wellKnownEndpoint);
+  return response.json();
+}
```

Sorgente A.8: Modifiche al file oauth-mtls.ts

/services/users-mtls.ts

```
@@ -0,0 +1,28 @@
+// SPDX-FileCopyrightText: 2025 Istituto Nazionale di Fisica Nucleare
+//
+// SPDX-License-Identifier: EUPL-1.2
+
```

```
+"use server";
+
+import { settings } from "@/config";
+import { Paginated } from "@/models/pagination";
+import { User } from "@/models/scim";
+import { mtlsGetItem } from "@/utils/fetch-mtls";
+
+const { BASE_URL } = settings;
+
+export async function getUsersPageMtls(
+  count: number,
+  startIndex: number = 1,
+  filter?: string
+) {
+  let url = `${BASE_URL}/iam/account/search?count=${count}&startIndex=
   ↪ ${startIndex}`;
+  if (filter) {
+    url += `&filter=${filter}`;
+  }
+  try {
+    return await mtlsGetItem<Paginated<User>>(url);
+  } catch (err) {
+    console.error(err)
+  }
+}
```

Sorgente A.9: Modifiche al file users-mtls.ts

/utils/fetch-mtls/index.ts

```
@@ -0,0 +1,67 @@
+// SPDX-FileCopyrightText: 2025 Istituto Nazionale di Fisica Nucleare
+//
+// SPDX-License-Identifier: EUPL-1.2
+
+import fetch, { RequestInit } from "node-fetch";
+import { readFileSync } from "fs";
+import https from "https";
```

```
+import { auth } from "@/services/oauth-mtls";
+import { notFound } from "next/navigation";
+
+let agent: https.Agent | undefined;
+
+try {
+  const clientCert = readFileSync(`${process.env.CLIENT_CERT_PATH}`);
+  const clientKey = readFileSync(`${process.env.CLIENT_KEY_PATH}`);
+  const rootCA = readFileSync(`${process.env.CA_PATH}`);
+
+  agent = new https.Agent({
+    cert: clientCert,
+    key: clientKey,
+    ca: rootCA
+  });
+} catch (error) {
+  console.error("Failed to create mTLS agent:", error);
+}
+
+export async function mtlsFetch(endpoint: string | URL, init?:
   ↪ RequestInit) {
+  const options: RequestInit = init ?? {};
+  let { headers } = options;
+  options.headers = {
+    ...headers,
+    "X-Client": "dashboard-mtls"
+  };
+  options.agent = agent;
+  return fetch(endpoint, options);
+}
+
+async function mtlsAuthFetch(endpoint: string | URL, init?:
   ↪ RequestInit) {
+  const accessToken = await auth();
+  if (!accessToken) {
+    throw Error("Session not ready");
+  }
+
```

```
+  const options: RequestInit = init ?? {};
+  let { headers } = options;
+  options.headers = {
+    ...headers,
+    authorization: `Bearer ${accessToken}`
+  };
+  return mtlsFetch(endpoint, options);
+}
+
+export async function mtlsGetItem<T>(endpoint: string | URL): Promise<
   ↪ T> {
+  const response = await mtlsAuthFetch(endpoint);
+  if (response.ok) {
+    return response.json() as Promise<T>;
+  } else {
+    const status = response.status;
+    if (status === 404) {
+      notFound();
+    } else {
+      throw Error(
+        `mtlsGetItem from ${endpoint} failed with status ${status}`
+      );
+    }
+  }
+};
```

Sorgente A.10: Modifiche al file index.ts

Bibliografia

[1] B. Campbell, J. Bradley, N. Sakimura, T. Lodderstedt, *OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens*, RFC 8705, Internet Engineering Task Force (IETF), febbraio 2020.
Disponibile su: `https://datatracker.ietf.org/doc/html/rfc8705`

[2] *INDIGO IAM*, Istituto Nazionale di Fisica Nucleare (INFN).
Disponibile su: `https://indigo-iam.github.io/`

[3] *Worldwide LHC Computing Grid*, CERN.
Disponibile su: `https://home.cern/science/computing/grid`

[4] D. Hardt, Ed., *The OAuth 2.0 Authorization Framework*, RFC 6749, Internet Engineering Task Force (IETF), ottobre 2012.
Disponibile su: `https://datatracker.ietf.org/doc/html/rfc6749`

[5] *Spring Boot*, VMware Tanzu.
Disponibile su: `https://spring.io/projects/spring-boot`

[6] *Next.js*, Vercel.
Disponibile su: `https://nextjs.org/`

[7] *NGINX*, F5 Inc..
Disponibile su: `https://nginx.org/`

[8] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, W. Polk, *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*,

RFC 5280, Internet Engineering Task Force (IETF), maggio 2008.
Disponibile su: `https://datatracker.ietf.org/doc/html/rfc5280`

[9] E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.3*, RFC 8446, Internet Engineering Task Force (IETF), agosto 2018.
Disponibile su: `https://datatracker.ietf.org/doc/html/rfc8446`

[10] S. Josefsson, *The Base16, Base32, and Base64 Data Encodings*, RFC 4648, Internet Engineering Task Force (IETF), ottobre 2006.
Disponibile su: `https://datatracker.ietf.org/doc/html/rfc4648`

[11] *OpenSSL*, OpenSSL.
Disponibile su: `https://www.openssl.org`

[12] *Auth.js*, Better Auth Inc..
Disponibile su: `https://authjs.dev/`

[13] *Node.js*, OpenJS Foundation.
Disponibile su: `https://nodejs.org/en`

[14] *HTTP Agent*, Node.js.
Disponibile su: `https://nodejs.org/api/https.html#class-httpsagent`

[15] *Docker Compose*, Docker Inc..
Disponibile su: `https://docs.docker.com/compose/`

[16] *Docker*, Docker Inc..
Disponibile su: `https://www.docker.com/`

[17] D. Fett, B. Campbell, J. Bradley, T. Lodderstedt, M. Jones, and D. Waite *OAuth 2.0 Demonstrating Proof of Possession (DPoP)*, RFC 9449, Internet Engineering Task Force (IETF), settembre 2023.
Disponibile su: `https://datatracker.ietf.org/doc/html/rfc9449`

Ringraziamenti

Vorrei ringraziare il Professor Babaoglu e il Professor Giacomini per avermi consentito di lavorare ad un progetto così rilevante, trasmettendomi fiducia e serenità.

Un grazie a Jacopo, per la pazienza avuta e per tutto il tempo che mi hai dedicato, senza di te questo progetto di tesi sarebbe molto lontano dall'essere concluso.

...