Assignment 3: Web Security

Group: Oblig3WebSecurity 3

Members: Joachim Leiros, Thor Aasheim, Fred Christiansen, Andreas Seljeset

Part 1 – identifying vulnerabilities

Vulnerability #1: Broken Authentication: Session management

Description: While performing a search, the username is passed as a parameter and users are able to change the username in the URL, thus performing the search for the username they appended the URL.

Possible consequences: Users can apply data for other users and access information only certain users have access to.

File(s): SearchResultServlet.java

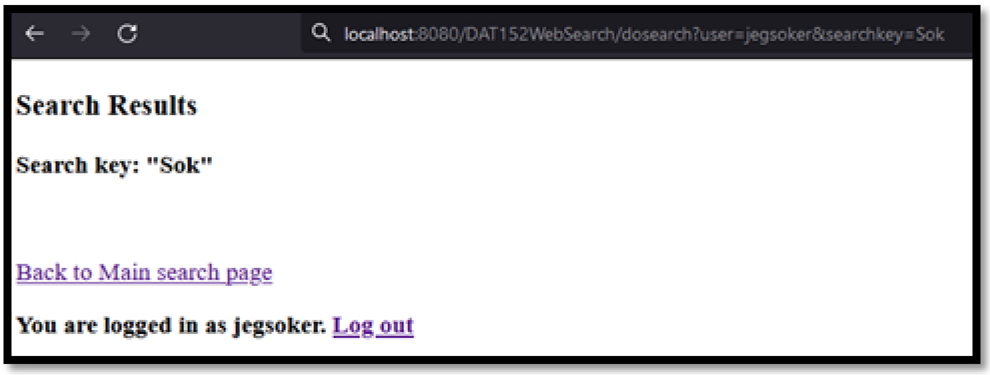
Code: doGet method of SearchResultServlet.java

Payload: Editing the URL from:

User actually performing the search: Localhost … / dosearch?user=user1&searchkey=mittsok

User being spoofed to perform the search: Localhost … / dosearch?user=admin&searchkey=mittsok

Technique: Manual, this was discovered while exploring the application and one member noticed the username was passed when searching. We tried changing the username and discovered that the search was still performed. We explored the code and saw that the username was passed when performing a search, but no authentication on the user performing the search is the same user of the session.


localhost:8080/DAT152WebSearch/dosearch?user=jegsoker&searchkey=Sok

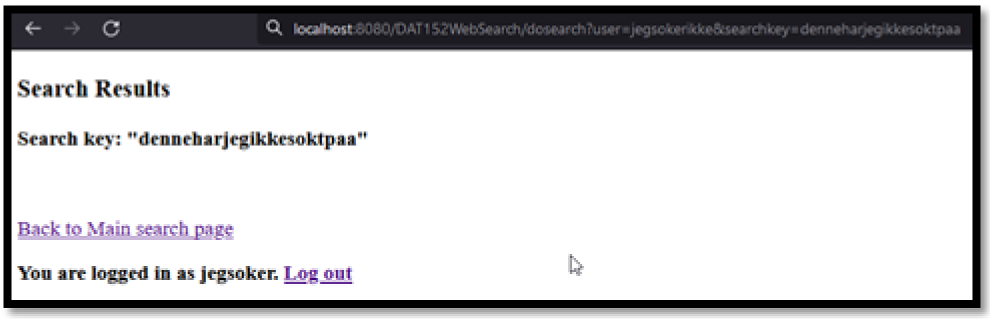
Search Results

Search key: "Sok"

Back to Main search page

You are logged in as jegsoker. Log out

The search is performed for the other user while we are logged in as “jegsoker”. We append “jegsokerikke”s username and search parameter in the URL and it is passed to the server.

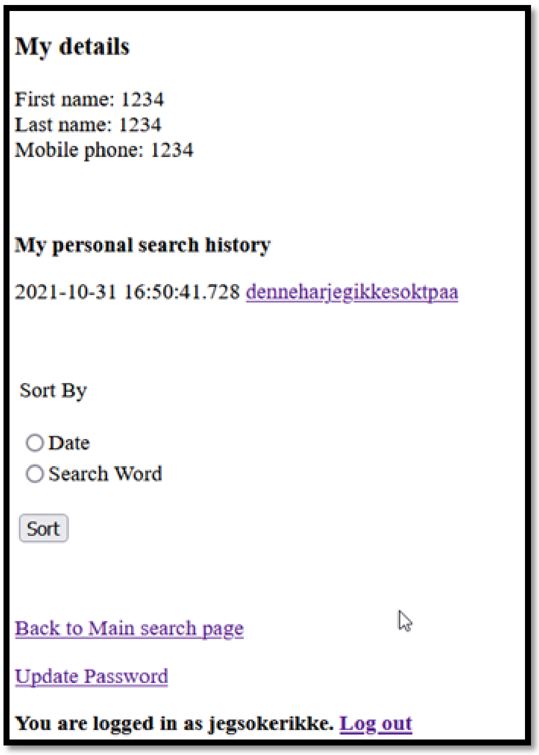

localhost:8080/DAT152WebSearch/dosearch?user=jegsokerikke&searchkey=denneharjegikkesoktpaa

Search Results

Search key: "denneharjegikkesoktpaa"

Back to Main search page

You are logged in as jegsoker. Log out


My details

First name: 1234
Last name: 1234
Mobile phone: 1234

My personal search history

2021-10-31 16:50:41.728 denneharjegikkesoktpaa

Sort By

○ Date
○ Search Word

Sort

Back to Main search page

Update Password

You are logged in as jegsokerikke. Log out

Here, the personal search history shows that jegsokerikke has searched for “denneharjegikkesoktpaa”. But in reality it was jegsoker that performed the search.

Vulnerability #2: SQL injection (SQLi)

Description: Poor SQL query handling

Possible consequences: SQLi vulnerability can result in an attacker gaining access to sensitive data, and may even add data or alter existing data. In one of the worst cases, an attacker can simply register themselves as an admin or log in as admin without approval from existing administrators. Attackers can modify an SQL query to return additional results.

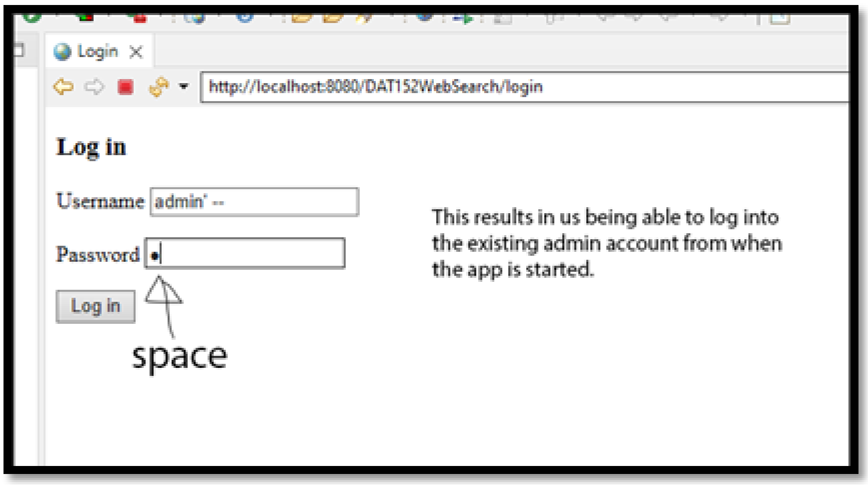
File(s): AppUserDAO.java, SearchItemDAO.java

Code: in AppUserDao.java, getAuthenticatedUser, getUserClientID, clientIDExist, saveUser, updateUserPassword, updateUserRole are affected from SQLi vulnerability.

SearchItemDAO; getSearchHistoryForUser, and getSearchHistoryForUser (sortkey), getSearchItemList contain SQL vulnerability.

All functions mentioned above used a simple SQL query to retrieve or alter the database. Every function had essentially the same vulnerability, where one could alter the query through user-input. Examples shown below.

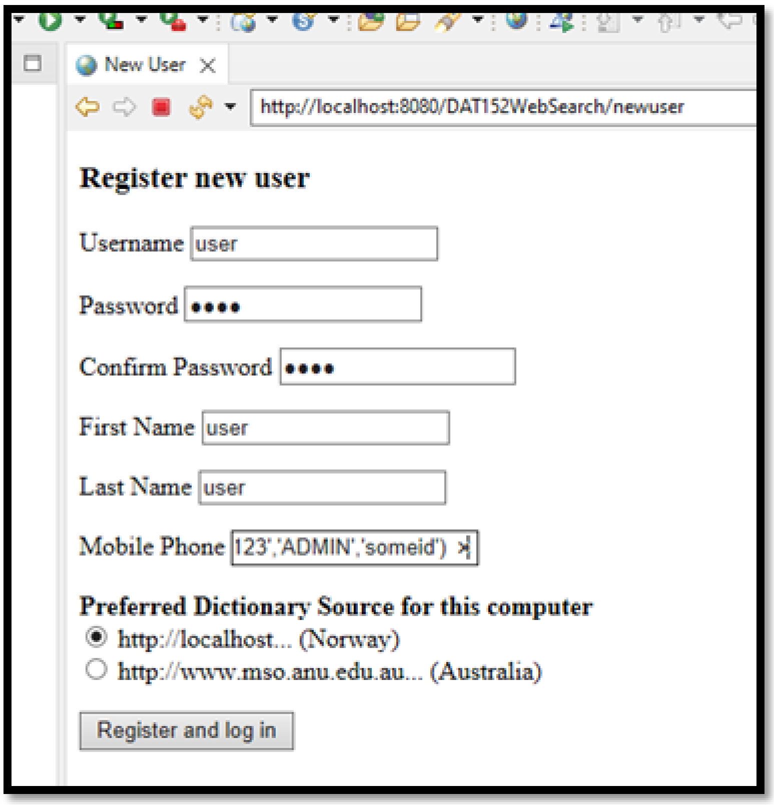
Payload: "admin' --" and "123','ADMIN','someid') --"


Login
http://localhost:8080/DAT152WebSearch/login
Log in
Username admin' --
Password •
Log in
space
This results in us being able to log into the existing admin account from when the app is started.

Admin' --

After clicking "log in" with the payload above results the user having access to the admin account.

Another vulnerability below:



New User

http://localhost:8080/DAT152WebSearch/newuser

Register new user

Username	user
Password	••••
Confirm Password	••••
First Name	user
Last Name	user
Mobile Phone	123','ADMIN','someid') --

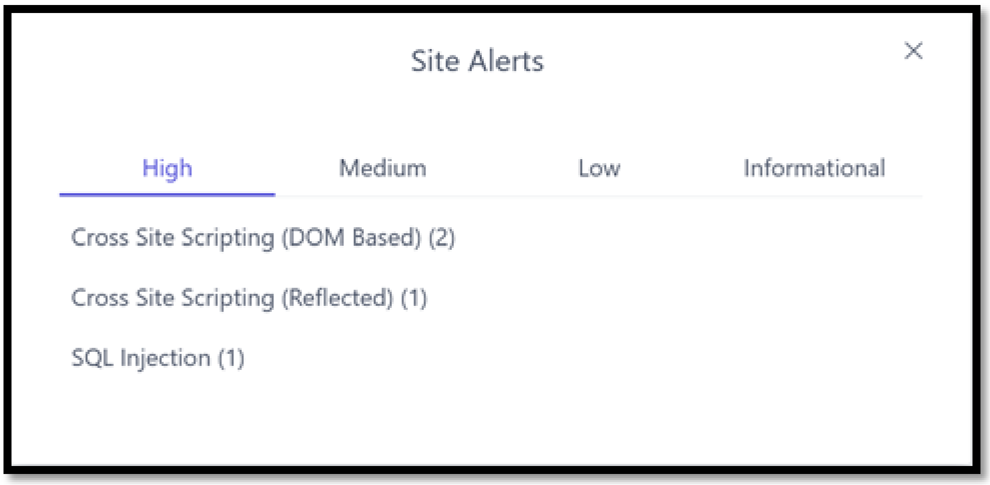
Preferred Dictionary Source for this computer

[✓ http://localhost... (Norway)]
http://www.mso.anu.edu.au... (Australia)

123','ADMIN','someid') --

By inputting the payload above as the "mobile number" you can create an admin account, by giving yourself said role. This allows you to see user information, and change roles of other users/admins.

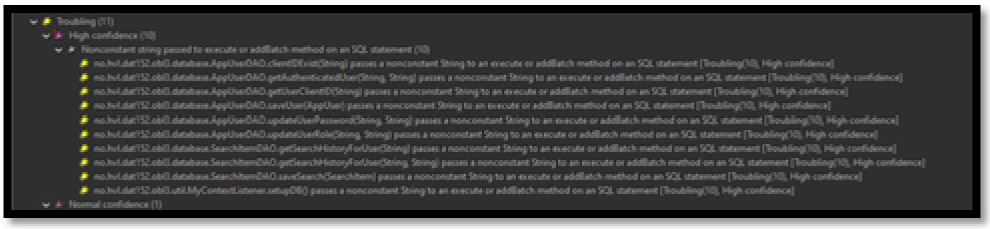
Analysis Technique: We used OWASP ZAP and FindBugs for Eclipse. We received an alert that the site could be faulty in this regard. We then tried to alter the SQL-query from login and create a user page, in which we were successful in gaining access to an existing admin account, and also creating an admin account without proper authorization.


Site Alerts

Cross Site Scripting (DOM Based) (2)

Cross Site Scripting (Reflected) (1)

SQL Injection (1)



```
Troubling (11)
  High confidence (10)
    Nonconstant string passed to execute or addBatch method on an SQL statement (10)
      no.hvl.dat152.obl3.database.AppUserDAO.clientIDExist(String) passes a nonconstant String to an execute or addBatch method on an SQL statement [Troubling(10), High confidence]
      no.hvl.dat152.obl3.database.AppUserDAO.getAuthenticatedUser(String, String) passes a nonconstant String to an execute or addBatch method on an SQL statement [Troubling(10), High confidence]
      no.hvl.dat152.obl3.database.AppUserDAO.getUserClientID(String) passes a nonconstant String to an execute or addBatch method on an SQL statement [Troubling(10), High confidence]
      no.hvl.dat152.obl3.database.AppUserDAO.saveUser(AppUser) passes a nonconstant String to an execute or addBatch method on an SQL statement [Troubling(10), High confidence]
      no.hvl.dat152.obl3.database.AppUserDAO.updateUserPassword(String, String) passes a nonconstant String to an execute or addBatch method on an SQL statement [Troubling(10), High confidence]
      no.hvl.dat152.obl3.database.AppUserDAO.updateUserRole(String, String) passes a nonconstant String to an execute or addBatch method on an SQL statement [Troubling(10), High confidence]
      no.hvl.dat152.obl3.database.SearchItemDAO.getSearchHistoryForUser(String) passes a nonconstant String to an execute or addBatch method on an SQL statement [Troubling(10), High confidence]
      no.hvl.dat152.obl3.database.SearchItemDAO.getSearchHistoryForUser(String, String) passes a nonconstant String to an execute or addBatch method on an SQL statement [Troubling(10), High confidence]
      no.hvl.dat152.obl3.database.SearchItemDAO.saveSearch(SearchItem) passes a nonconstant String to an execute or addBatch method on an SQL statement [Troubling(10), High confidence]
      no.hvl.dat152.obl3.util.MyContextListener.setupDB() passes a nonconstant String to an execute or addBatch method on an SQL statement [Troubling(10), High confidence]
  Normal confidence (1)
```

Vulnerability #3: Cross-Site Scripting (XSS) Reflected and Stored

Description: It is possible to use XSS on the search bar in Dat152WebSearch, since it doesn't sanitise its inputs, making it possible to run script and code directly into the website, it is also possible to store scripts in the search history with the same method.

Possible consequences: It is possible to block out admins as long as the script is in the top five searches, this with <script>location.reload();</script>. Since it is possible to run all kinds of scripts it would also be possible to run files from the search results, making it possible to download malicious software on the admins computers, the possibilities are almost endless.

File(s): SearchResultServlet.java

Code:

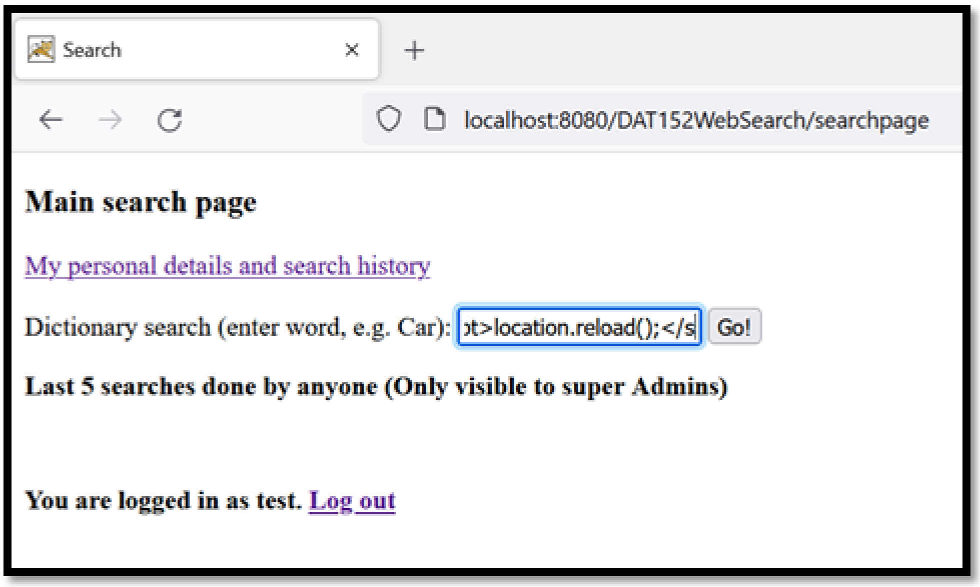
```
String searchkey =
Validator.validString(request.getParameter("searchkey"));
```

gets it code directly from the search bar, it doesn't check the content of the string. To fix this we decided to use the Jsoup Html Parser

```
searchkey = Jsoup.clean(searchkey, Safelist.basic());
```

which checks and cleans html code from string negating XSS attack on the searchbar and then printing the searches as normal code.

Payload:



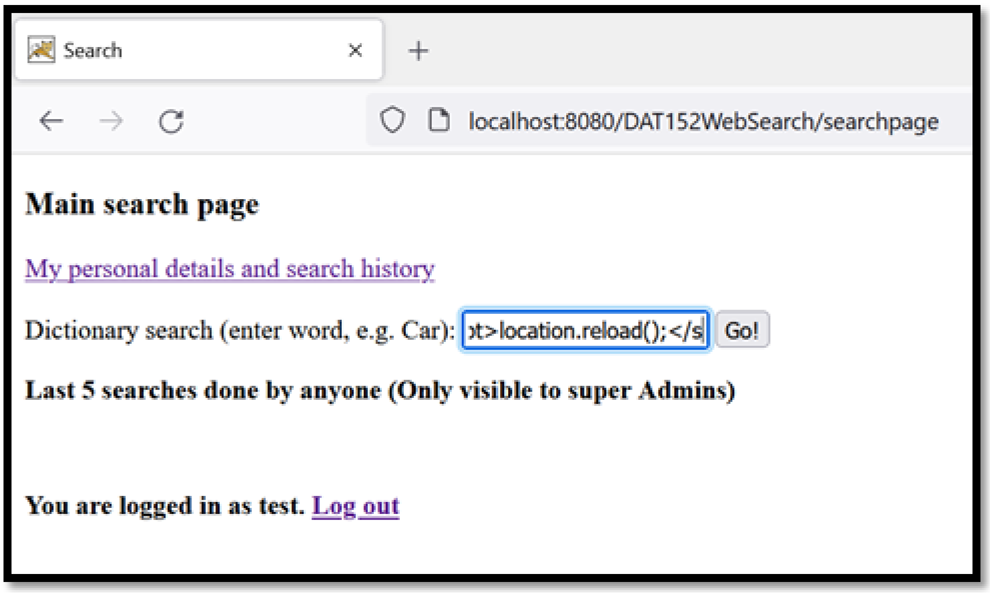
Main search page

My personal details and search history

Dictionary search (enter word, e.g. Car): pt>location.reload();</s Go!

Last 5 searches done by anyone (Only visible to super Admins)

You are logged in as test. Log out



Search

localhost:8080/DAT152WebSearch/searchpage

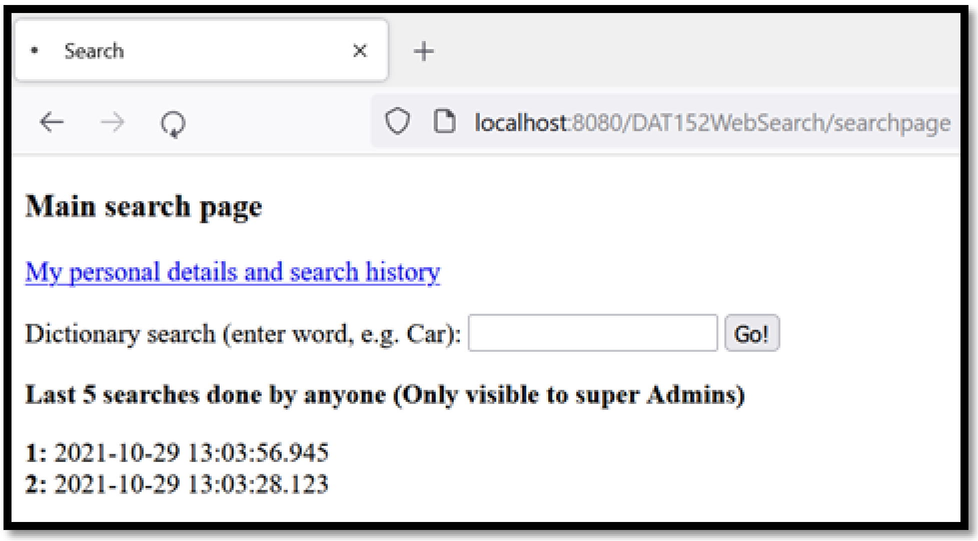
Main search page

My personal details and search history

Dictionary search (enter word, e.g. Car): ›t>location.reload();</s Go!

Last 5 searches done by anyone (Only visible to super Admins)

You are logged in as test. Log out



Search

localhost:8080/DAT152WebSearch/searchpage

Main search page

My personal details and search history

Dictionary search (enter word, e.g. Car): Go!

Last 5 searches done by anyone (Only visible to super Admins)

1: 2021-10-29 13:03:56.945
2: 2021-10-29 13:03:28.123

Analysis Technique: Used OWASP/manual, it said there was a high possibility for cross-site scripting, it pointed to searchpage.jsp, I found that searchkey would be the spot, since it was collecting from the searchbar, then I searched around until I found the searchkey parameter in the SearchResultServlet and saw that there was no check for html code from the search bar, since java doesn't check for html code by itself.

Vulnerability #4: Cross-Site Request Forgery (CSRF)

Description: In the WebSearch project there is no CSRF token generated when you log in. Websites normally use CSRF token to make sure it is the same user requesting a GET or POST method that is logged inn. The tokens help preventing unauthorized users make changes or request that they are not allowed to.

Possible consequences: An unauthorized user can request a role change from a normal user to an admin, this by trying to get a logged to click/hyperlink, usually via social engineering.

In this application a potential attack is a user trying to get a logged in administrator to click the link *http://localhost:8080/DAT152WebSearch/updaterole?username=test&role=ADMIN*. This link will requests a role change to admin for the user test. Since there is no authentication checking if the logged in user is the same user performing the request, the request will pass.

File(s): updaterole.jsp, updateRoleServlet.java

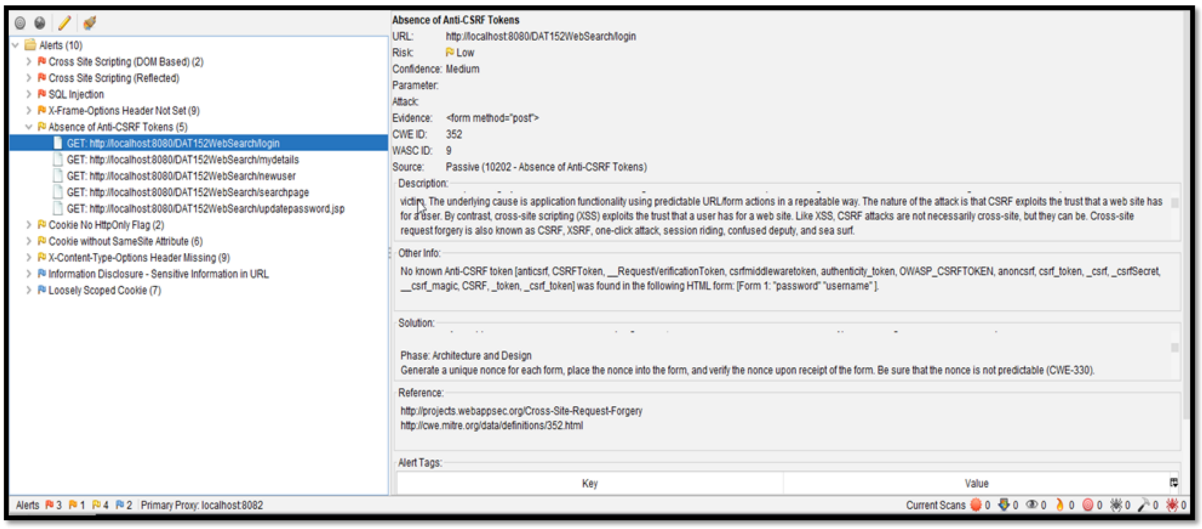
Code: Missing CSRF token in JSP

Missing authentication of user in updateRoleServlet doPost (Line 25)

Payload:

<a href=*http://localhost:8080/DAT152WebSearch/updaterole?username=test&role=ADMIN*>

Analysis Technique: OWASP was used and alerted us that there was no CSRF token on the web application.



```
Alerts (10)
  Cross Site Scripting (DOM Based) (2)
  Cross Site Scripting (Reflected)
  SQL Injection
  X-Frame-Options Header Not Set (9)
  Absence of Anti-CSRF Tokens (5)
    GET: http://localhost:8080/DAT152WebSearch/login
    GET: http://localhost:8080/DAT152WebSearch/mydetails
    GET: http://localhost:8080/DAT152WebSearch/newuser
    GET: http://localhost:8080/DAT152WebSearch/searchpage
    GET: http://localhost:8080/DAT152WebSearch/updatepassword.jsp
  Cookie No HttpOnly Flag (2)
  Cookie without SameSite Attribute (6)
  X-Content-Type-Options Header Missing (9)
  Information Disclosure - Sensitive Information in URL
  Loosely Scoped Cookie (7)

Absence of Anti-CSRF Tokens
URL:        http://localhost:8080/DAT152WebSearch/login
Risk:       Low
Confidence: Medium
Parameter:
Attack:
Evidence:   <form method="post">
CWE ID:     352
WASC ID:    9
Source:     Passive (10202 - Absence of Anti-CSRF Tokens)
Description:
victim. The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a web site has
for a user. By contrast, cross-site scripting (XSS) exploits the trust that a user has for a web site. Like XSS, CSRF attacks are not necessarily cross-site, but they can be. Cross-site
request forgery is also known as CSRF, XSRF, one-click attack, session riding, confused deputy, and sea surf.
Other Info:
No known Anti-CSRF token [anticsrf, CSRFToken, __RequestVerificationToken, csrfmiddlewaretoken, authenticity_token, OWASP_CSRFTOKEN, anoncsrf, csrf_token, _csrf, _csrfSecret,
__csrf_magic, CSRF, _token, _csrf_token] was found in the following HTML form: [Form 1: "password" "username" ].
Solution:

Phase: Architecture and Design
Generate a unique nonce for each form, place the nonce into the form, and verify the nonce upon receipt of the form. Be sure that the nonce is not predictable (CWE-330).
Reference:
http://projects.webappsec.org/Cross-Site-Request-Forgery
http://cwe.mitre.org/data/definitions/352.html
Alert Tags:
Key    Value

Alerts 3 1 4 2   Primary Proxy: localhost:8082        Current Scans 0 0 0 0 0 0 0 0
```

Vulnerability #5: SSO and weakness in JWT authentication token

Description: SSO and weakness in JWT authentication token-DAT152WebBlogApp

File(s):

BlogWebApp: blog.jsp, WebSearch: Token.java

Code: authorizationCodeRequest in Token.java, blog.jsp (Line 35)

Possible consequences:

What is the id_token(authenticationtoken)used in the DAT152WebBlogApp?

4A4676A752957EF59915B85F62BD0181 (not persistent)

Where is the id_token(authenticationtoken) stored?

The id_token is stored as a cookie in your browser.

What are the vulnerabilities that you think exist in this id_token?

The token is unencrypted, if it is intercepted it can possibly be used to login as the user it was intercepted from. The token is not properly verified which means an expired token can be used and it could be altered to provide admin privileges.

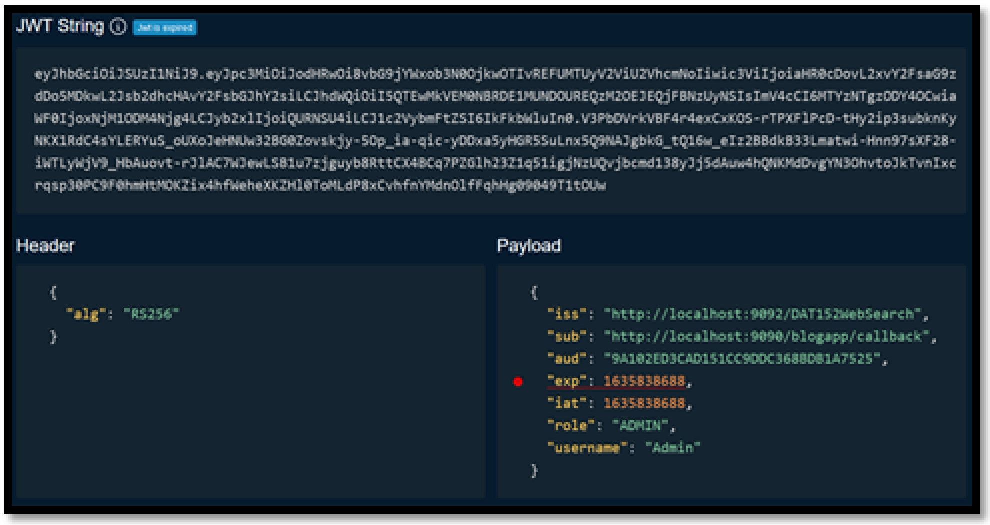
What security decisions are being made using this id_token?

We assume we are on a network where interception of the id_token is not a concern.

Can a user elevate his privilege in this id_token?

We used https://token.dev/ to easily decode and alter the JWT. We took a standard JWT and were able to remove admin privileges as well as re-implement admin privileges by editing the payload, we were also able to edit the expiration date of the token.

Original JWT:

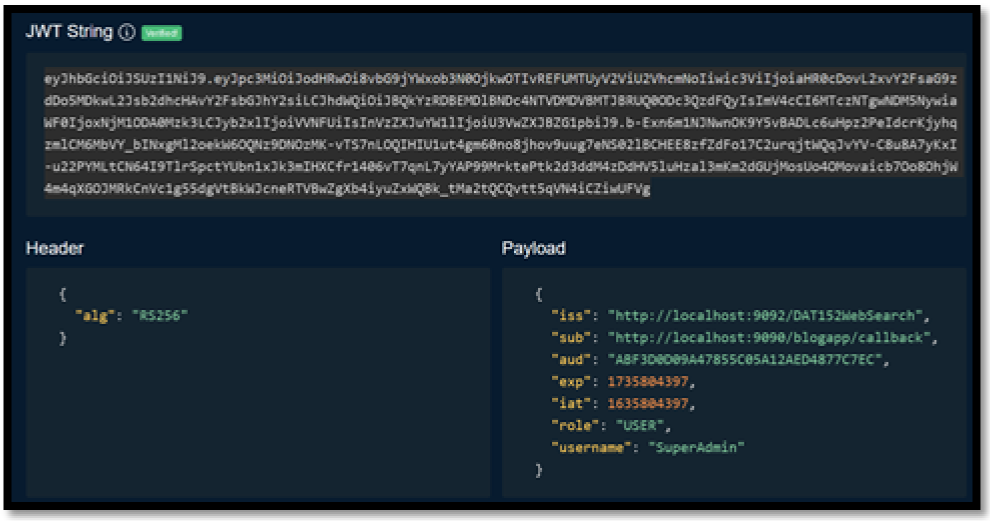


```
eyJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJodHRwOi8vbG9jYWxob3N0OjkwOTIvREFUMTUyV2ViU2VhcmNoIiwic3ViIjoiaHR0cDovL2xvY2FsaG9z
dDo5MDkwL2Jsb2dhcHAvY2FsbGJhY2siLCJhdWQiOiI5QTEwMkVEM0NBRDE1MUNDOUREQzM2OEJEQjFBNzUyNSIsImV4cCI6MTYzNTgzODY4OCwia
WF0IjoxNjM1ODM4Njg4LCJyb2xlIjoiQURNSU4iLCJ1c2VybmFtZSI6IkFkbWluIn0.V3PbDVrkVBF4r4exCxKOS-rTPXFlPcD-tHy2ip3subknKy
NKX1RdC4sYLERYuS_oUXoJeHNUw32BG0Zovskjy-5Op_ia-qic-yDDxa5yHGR5SuLnx5Q9NAJgbkG_tQ16w_eIz2BBdkB33Lmatwi-Hnn97sXF28-
iWTLyWjV9_HbAuovt-rJlAC7WJewLS81u7zjguyb8RttCX4BCq7PZGlh23Z1q51igjNzUQvjbcmd138yJj5dAuw4hQNKMdDvgYN3OhvtoJkTvnIxc
rqsp30PC9F0hmHtMOKZix4hfWeheXKZHl0ToMLdP8xCvhfnYMdnOlfFqhHg09049T1tOUw
```

```
{
  "alg": "RS256"
}
```

```
{
  "iss": "http://localhost:9092/DAT152WebSearch",
  "sub": "http://localhost:9090/blogapp/callback",
  "aud": "9A102ED3CAD151CC9DDC368BDB1A7525",
  "exp": 1635838688,
  "iat": 1635838688,
  "role": "ADMIN",
  "username": "Admin"
}
```

Altered JWT:

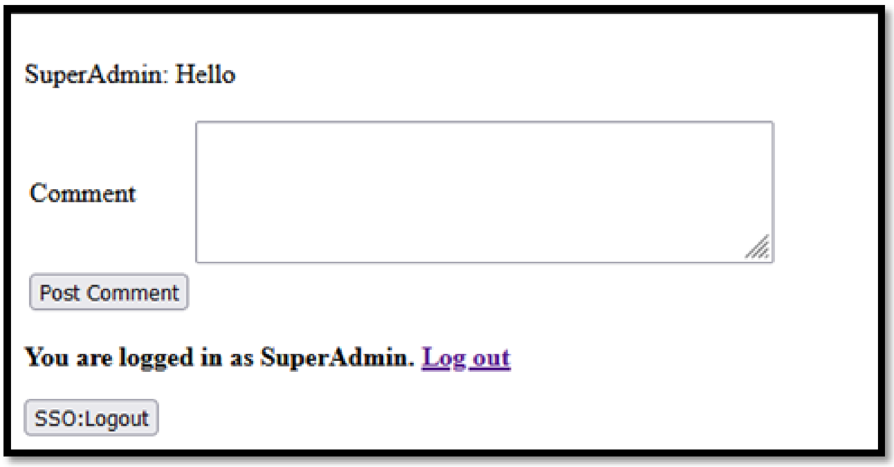


```
eyJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJodHRwOi8vbG9jYWxob3N0OjkwOTIvREFUMTUyV2ViU2VhcmNoIiwic3ViIjoiaHR0cDovL2xvY2FsaG9z
dDo5MDkwL2Jsb2dhcHAvY2FsbGJhY2siLCJhdWQiOiJBQkYzRDBEMDlBNDc4NTVDMDVBMTJBRUQ0ODc3QzdFQyIsImV4cCI6MTczNTgwNDM5Nywia
WF0IjoxNjM1ODA0Mzk3LCJyb2xlIjoiVVNFUiIsInVzZXJuYW1lIjoiU3VwZXJBZG1pbiJ9.b-Exn6m1NJNwnOK9Y5vBADLc6uHpz2PeIdcrKjyhq
zmlCM6MbVY_bINxgMl2oekW6OQNz9DNOzMK-vTS7nLOQIHIU1ut4gm60no8jhov9uug7eNS02l8CHEE8zfZdFo17C2urqjtWQqJvYV-C8u8A7yKxI
-u22PYMLtCN64I9TlrSpctYUbn1xJk3mIHXCfr1406vT7qnL7yYAP99MrktePtk2d3ddM4zDdHV5luHzal3mKm2dGUjMosUo4OMovaicb7Oo8OhjW
4m4qXGOJMRkCnVc1g55dgVtBkWJcneRTVBwZgXb4iyuZxWQBk_tMa2tQCQvtt5qVN4iCZiwUFVg
```

```
{
  "alg": "RS256"
}
```

```
{
  "iss": "http://localhost:9092/DAT152WebSearch",
  "sub": "http://localhost:9090/blogapp/callback",
  "aud": "ABF3D0D09A47855C05A12AED4877C7EC",
  "exp": 1735804397,
  "iat": 1635804397,
  "role": "USER",
  "username": "SuperAdmin"
}
```

Result of altered JWT:



SuperAdmin: Hello

Comment

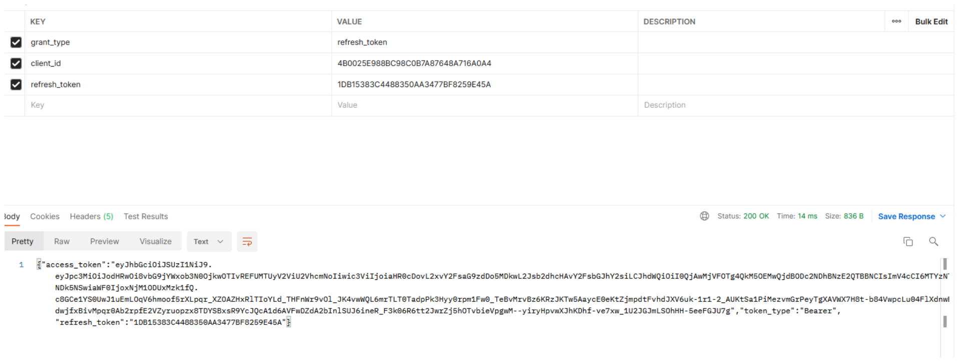
You are logged in as SuperAdmin. Log out

Token endpoint

Can a user use her/his refresh_token to request for access_token for another user whose client_id is known?

We were able to access user b's acces_token with user a's refresh_token, but we were not able to access userinfo for user b.

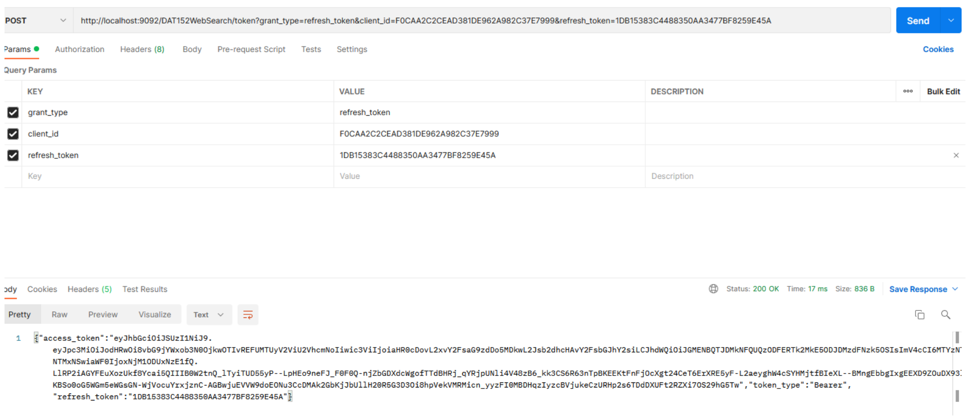
Here user A gains access to user B's access token with user A's refresh token.



	KEY	VALUE	DESCRIPTION
✓	grant_type	refresh_token	
✓	client_id	4B0025E988BC98C0B7A87648A716A0A4	
✓	refresh_token	1DB15383C4488350AA3477BF8259E45A	

Status: 200 OK Time: 14 ms Size: 836 B

```
{"access_token":"eyJhbGciOiJSUzI1NiJ9.
 eyJpc3MiOiJodHRwOi8vbG9jYWxob3N0OjkwOTIvREFUMTUyV2ViU2VhcmNoIiwic3ViIjoiaHR0cDovL2xvY2FsaG9zdDo5MDkwL2Jsb2dhcHAvY2FsbGJhY2siLCJhdWQiOiI0QjAwMjVFOTg4QkM5OEMwQjdBODc2NDhBNzE2QTBBNCIsImV4cCI6MTYzN
 NDk5NSwiaWF0IjoxNjM1ODUxMzk1fQ.
 c8GCe1YS0UwJ1uEmLOqV6hmoof5rXLpqr_XZOAZHxRlTIoYLd_THFnWr9vOl_JK4vwWQL6mrTLT0TadpPk3Hyy0rpm1Fw0_TeBvMrvBz6KRzJKTw5AaycE0eKtZjmpdtFvhdJXV6uk-1r1-2_AUKtSa1PiMezvmGrPeyTgXAVWX7H8t-b84VwpcLu04FlXdnwl
 dwjfxBivMpqr0Ab2rpfE2VZyruopzx8TDYSBxsR9YcJQcA1d6AVFwDZdA2bInlSUJ6ineR_F3k06R6tt2JwrZj5hOTvbieVpgwM--yiryHpvwXJhKDhf-ve7xw_1U2JGJmLSOhHH-5eeFGJU7g","token_type":"Bearer",
 "refresh_token":"1DB15383C4488350AA3477BF8259E45A"}
```



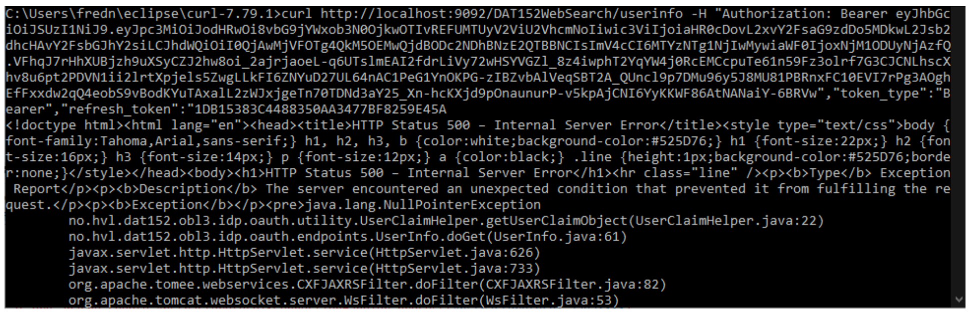
```
POST http://localhost:9092/DAT152WebSearch/token?grant_type=refresh_token&client_id=F0CAA2C2CEAD381DE962A982C37E7999&refresh_token=1DB15383C4488350AA3477BF8259E45A
```

Query Params

	KEY	VALUE	DESCRIPTION
✓	grant_type	refresh_token	
✓	client_id	F0CAA2C2CEAD381DE962A982C37E7999	
✓	refresh_token	1DB15383C4488350AA3477BF8259E45A	

Status: 200 OK Time: 17 ms Size: 836 B

```
{"access_token":"eyJhbGciOiJSUzI1NiJ9.
 eyJpc3MiOiJodHRwOi8vbG9jYWxob3N0OjkwOTIvREFUMTUyV2ViU2VhcmNoIiwic3ViIjoiaHR0cDovL2xvY2FsaG9zdDo5MDkwL2Jsb2dhcHAvY2FsbGJhY2siLCJhdWQiOiJGMENBQTJDMkNFQUQzODFERTk2MkE5ODJDMzdFNzk5OSIsImV4cCI6MTYzN
 NTMxNSwiaWF0IjoxNjM1ODUxNzE1fQ.
 LlRP2iAGYFEuXozUkf8Ycai5QIIIB0W2tnQ_lTyiTUD55yP--LpHEo9neFJ_F0F0Q-njZbGDXdcWgofTTdBHRj_qYRjpUNli4V48zB6_kk3CS6R63nTpBKEEKtFnFjOcXgt24CeT6ErXRE5yF-L2aeyghW4cSYHMjtfBIeXL--BMngEbbgIxgEEXD9ZOuDX93
 KBSo0oG5WGm5eWGsGN-WjVocuYrxjznC-AGBwjuEVVW9doEONu3CcDMAk2GbKjJbUllH20R5G3D3Oi8hpVekVMRMicn_yyzFI0MBDHqzIyzcBVjukeCzURHp2s6TDdDXUFt2RZXi7OS29hG5Tw","token_type":"Bearer",
 "refresh_token":"1DB15383C4488350AA3477BF8259E45A"}
```



```
C:\Users\fredn\eclipse\curl-7.79.1>curl http://localhost:9092/DAT152WebSearch/userinfo -H "Authorization: Bearer eyJhbGc
iOiJSUzI1NiJ9.eyJpc3MiOiJodHRwOi8vbG9jYWxob3N0OjkwOTIvREFUMTUyV2ViU2VhcmNoIiwic3ViIjoiaHR0cDovL2xvY2FsaG9zdDo5MDkwL2Jsb2
dhcHAvY2FsbGJhY2siLCJhdWQiOiI0QjAwMjVFOTg4QkM5OEMwQjdBODc2NDhBNzE2QTBBNCIsImV4cCI6MTYzNTg1NjIwMywiaWF0IjoxNjM1ODUyNjAzfQ
.VFhqJ7rHhXUBjzh9uXSyCZJ2hw8oi_2ajrjaoeL-q6UTslmEAI2fdrLiVy72wHSYVGZl_8z4iwphT2YqYW4j0RcEMCcpuTe61n59Fz3olrf7G3CJCNLhscX
hv8u6pt2PDVN1ii2lrtXpjels5ZwgLLkFI6ZNYuD27UL64nAC1PeG1YnOKPG-zIBZvbAlVeqSBT2A_QUncl9p7DMu96y5J8MU81PBRnxFC10EVI7rPg3AOgh
EfFxxdw2qQ4eobS9vBodKYuTAxalL2zWJxjgeTn70TDNd3aY25_Xn-hcKXjd9pOnaunurP-v5kpAjCNI6YyKKWF86AtNANaiY-6BRVw","token_type":"B
earer","refresh_token":"1DB15383C4488350AA3477BF8259E45A
<!doctype html><html lang="en"><head><title>HTTP Status 500 – Internal Server Error</title><style type="text/css">body {
font-family:Tahoma,Arial,sans-serif;} h1, h2, h3, b {color:white;background-color:#525D76;} h1 {font-size:22px;} h2 {fon
t-size:16px;} h3 {font-size:14px;} p {font-size:12px;} a {color:black;} .line {height:1px;background-color:#525D76;borde
r:none;}</style></head><body><h1>HTTP Status 500 – Internal Server Error</h1><hr class="line" /><p><b>Type</b> Exception
 Report</p><p><b>Description</b> The server encountered an unexpected condition that prevented it from fulfilling the re
quest.</p><p><b>Exception</b></p><pre>java.lang.NullPointerException
        no.hvl.dat152.obl3.idp.oauth.utility.UserClaimHelper.getUserClaimObject(UserClaimHelper.java:22)
        no.hvl.dat152.obl3.idp.oauth.endpoints.UserInfo.doGet(UserInfo.java:61)
        javax.servlet.http.HttpServlet.service(HttpServlet.java:626)
        javax.servlet.http.HttpServlet.service(HttpServlet.java:733)
        org.apache.tomee.webservices.CXFJAXRSFilter.doFilter(CXFJAXRSFilter.java:82)
        org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:53)
```

Userinfo endpoint

If yes in above, can the obtained access_token be used to request for claims for the client?

Attempting to request user claims with the obtained access_token results in a null pointer exception.

Part 2 – Mitigating vulnerabilities

Vulnerability #1: Broken Authentication: Session management

Description: Broken authentication: Session management. Users was able perform a search on behalf of other users.

Part of code: SearchResultServlet.java / RequestHelper.java

Mitigation/control code:

In order to authenticate that the username performing the search is the same as the one requesting the search, we add a helper method in the RequestHelper.java (Line 32).

```
public static String getLoggedInUsername(HttpServletRequest
request) {
      return ((AppUser)
request.getSession().getAttribute("user")).getUsername();
      }
```

Here we request the username from the database of users.

Then in the SearchResultServlet.java (Line 47)

```
if (!RequestHelper.getLoggedInUsername(request).equals(user)) {
                response.sendRedirect("searchpage");
                return;
}
```

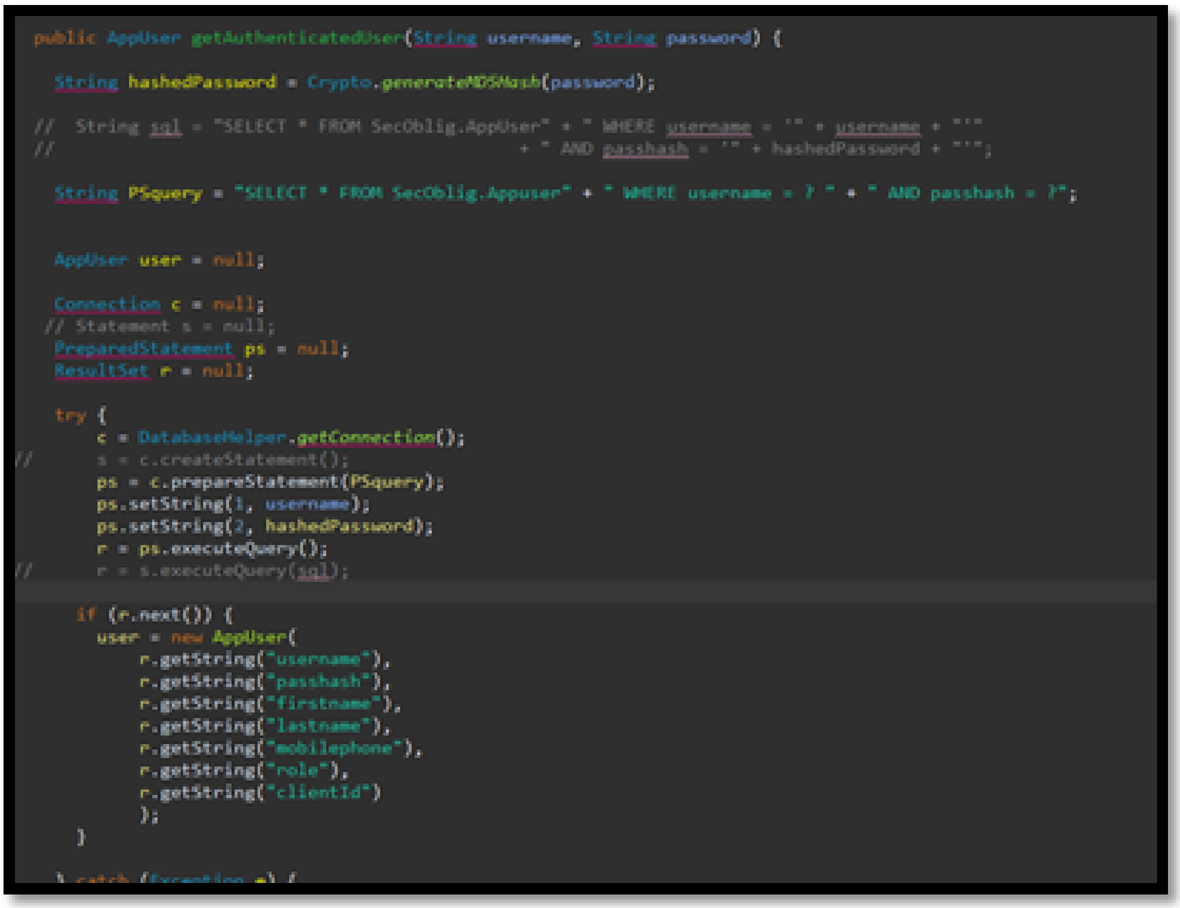
We use the helper method from RequestHelper to check that the username of the requester is the same as the username of the user.

Vulnerability #2: SQL injection (SQLi)

Description: SQL query can be altered through user input, which could result in unauthorized access to data for attacker, or access to different users' accounts.

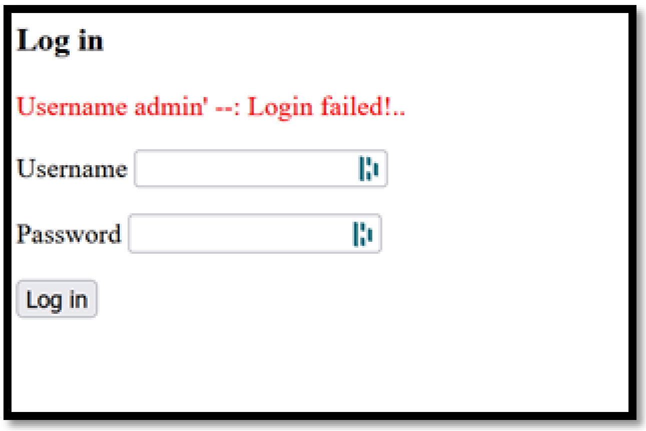
Part of code: The code we've fixed is in **AppUserDAO** and **SearchItemDAO**. Each and every SQL query has been swapped with a prepared statement. This is essentially a pre-compiled template where the actual variables are written as "?". The variables from user-input are sent later.

Mitigation/control code: Prepared statement code snippet. This is the fix we've applied to all the other functions.



```
public AppUser getAuthenticatedUser(String username, String password) {

  String hashedPassword = Crypto.generateMD5Hash(password);

//  String sql = "SELECT * FROM SecOblig.AppUser" + " WHERE username = '" + username + "'"
//                                                 + " AND passhash = '" + hashedPassword + "'";

  String PSquery = "SELECT * FROM SecOblig.Appuser" + " WHERE username = ? " + " AND passhash = ?";


  AppUser user = null;

  Connection c = null;
 // Statement s = null;
  PreparedStatement ps = null;
  ResultSet r = null;

  try {
      c = DatabaseHelper.getConnection();
//      s = c.createStatement();
      ps = c.prepareStatement(PSquery);
      ps.setString(1, username);
      ps.setString(2, hashedPassword);
      r = ps.executeQuery();
//      r = s.executeQuery(sql);

    if (r.next()) {
      user = new AppUser(
          r.getString("username"),
          r.getString("passhash"),
          r.getString("firstname"),
          r.getString("lastname"),
          r.getString("mobilephone"),
          r.getString("role"),
          r.getString("clientId")
          );
    }

  } catch (Exception e) {
```

As a result of using the prepared statement, you can no longer log in with the "admin' --" input as shown below.


Log in

Username admin' --: Login failed!..

Username

Password

Log in

Vulnerability #3: Cross-site scripting(XSS)

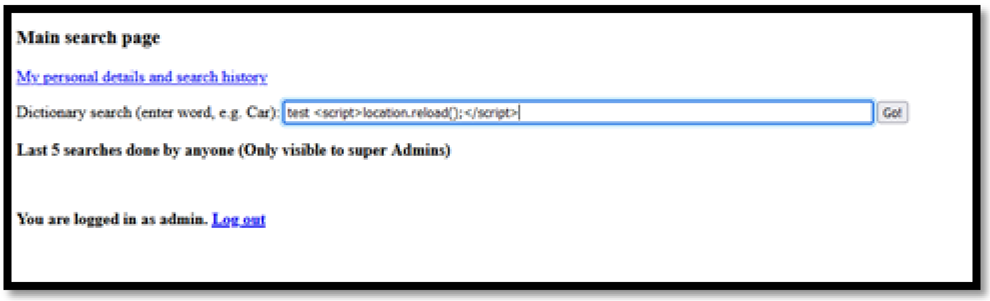
Description: It is possible to use XSS on the search bar in Dat152WebSearch, since it doesn't sanitise its inputs, making it possible to run script and code directly into the website, it is also possible to store scripts in the search history with the same method

Part of code: The code part "String searchkey = Validator.validString(request.getParameter("searchkey"));" gets it code directly from the search bar, it doesn't check the content of the string.

Mitigation/control code: To fix this we decided to use the Jsoup Html Parser where we clean the searchkey before we use it

```
 searchkey = Jsoup.clean(searchkey, Safelist.basic());
```

this checks and cleans html code from strings negating XSS attacks on the search bar and then printing the searches as normal code.


Main search page

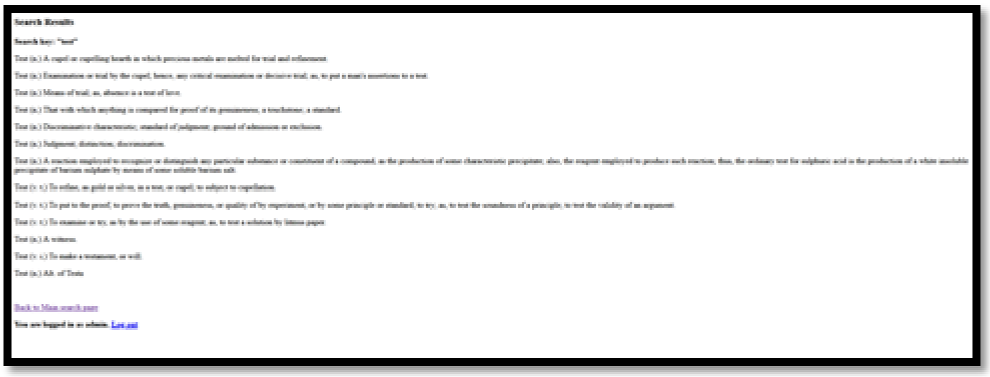
My personal details and search history

Dictionary search (enter word, e.g. Car): test <script>location.reload();</script>

Go!

Last 5 searches done by anyone (Only visible to super Admins)

You are logged in as admin. Log out



Search Results

Search key: "test"

Test (n.) A cupel or cupelling hearth in which precious metals are melted for trial and refinement.

Test (n.) Examination or trial by the cupel; hence, any critical examination or decisive trial; as, to put a man's assertions to a test.

Test (n.) Means of trial; as, absence is a test of love.

Test (n.) That with which anything is compared for proof of its genuineness; a touchstone; a standard.

Test (n.) Discriminative characteristic; standard of judgment; ground of admission or exclusion.

Test (n.) Judgment; distinction; discrimination.

Test (n.) A reaction employed to recognize or distinguish any particular substance or constituent of a compound, as the production of some characteristic precipitate; also, the reagent employed to produce such reaction; thus, the ordinary test for sulphuric acid is the production of a white insoluble precipitate of barium sulphate by means of some soluble barium salt.

Test (v. t.) To refine, as gold or silver, in a test, or cupel; to subject to cupellation.

Test (v. t.) To put to the proof; to prove the truth, genuineness, or quality of by experiment, or by some principle or standard; to try; as, to test the soundness of a principle; to test the validity of an argument.

Test (v. t.) To examine or try, as by the use of some reagent; as, to test a solution by litmus paper.

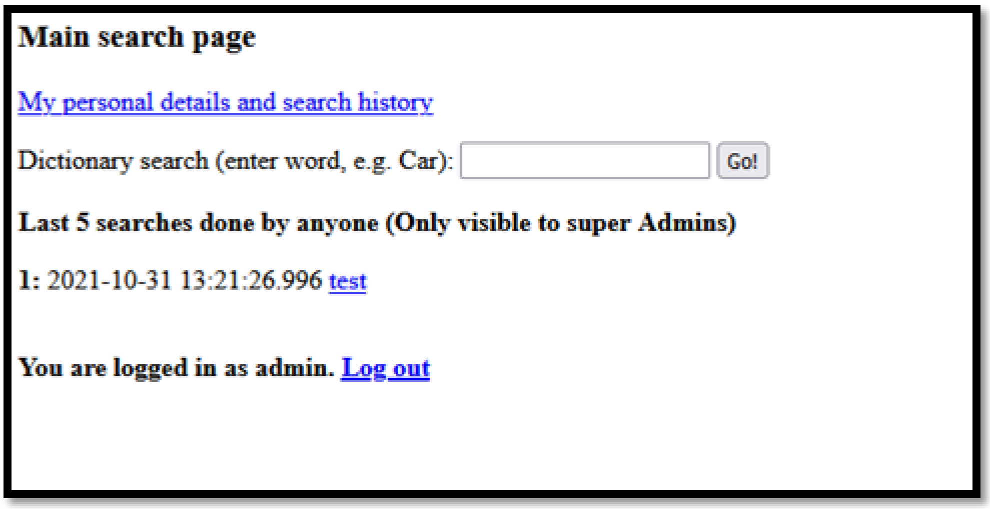
Test (n.) A witness.

Test (v. i.) To make a testament, or will.

Test (n.) Alt. of Testa

Back to Main search page

You are logged in as admin. Log out



Main search page

My personal details and search history

Dictionary search (enter word, e.g. Car):

Last 5 searches done by anyone (Only visible to super Admins)

1: 2021-10-31 13:21:26.996 test

You are logged in as admin. Log out

Vulnerability #4: Cross-site request forgery(CSRF)

Description: To mitigate the error and avoid CSRF, we have implemented an "anticsrf" token. When the user logs in the server generates a unique token which is stored in the user's session. In the JSP's we have added

<input type="hidden" name="anticsrf" value="${anticsrf}">

The token is gathered from the session and checked by the server that the token of the user requesting a post is the same that of the user performing the request.

Part of code:

For each form under: Login.jsp, Mydetails.jsp, Newuser.jsp, Searchpage.jsp, Updatepassword.jsp, Roleupdate.jsp

UpdateRoleServlet.java (Line 40)

LoginServlet.java (Line 46)

Validator.java (Line 10)

AppUserDAO.java (Line 238)

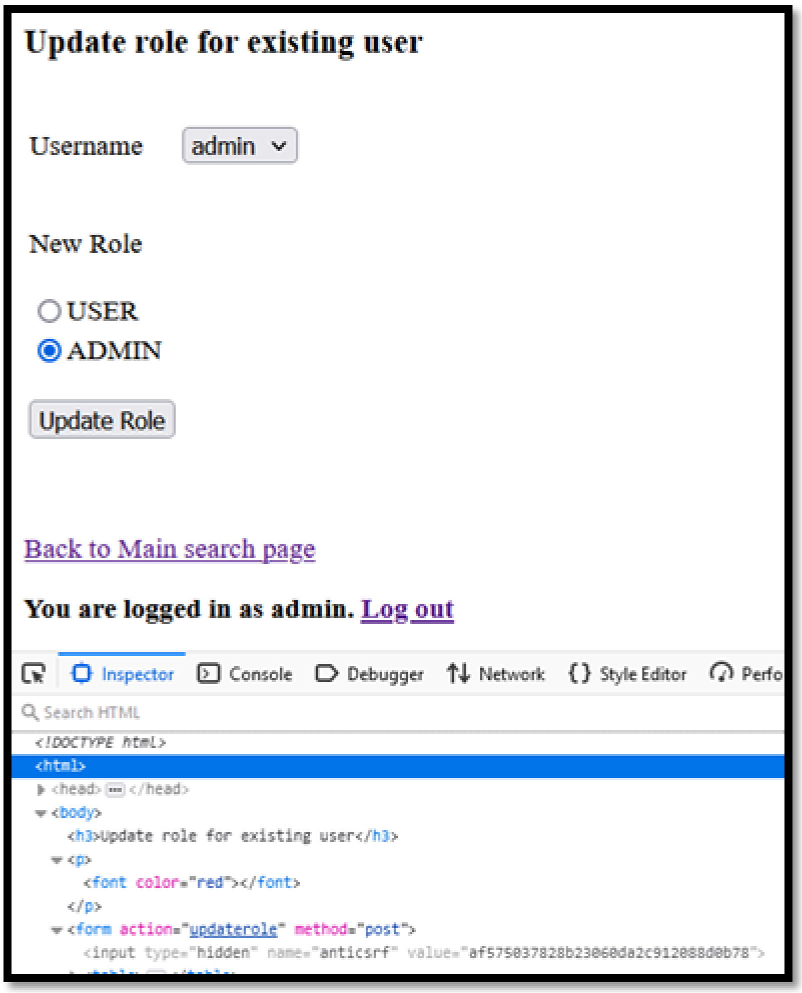
Mitigation/control code:

UpdateRoleServlet.java:

```
if (Validator.csrfValidity(request)) {
            request.getSession().invalidate();
            request.setAttribute("message", "CSRF tokens did not
match, no hackiehackie!");

request.getRequestDispatcher("login.jsp").forward(request,
response);

            System.out.println("CSRF Tokensdid NOT Match!");
      } else {
            System.out.println("CSRF Token DOES Match!");
  }
```

Validator.java: Validating if token is valid or not

```
public static boolean csrfValidity (HttpServletRequest r) {
     String a = "";
     try {
          a = (String) r.getSession().getAttribute("anticsrf");
     } catch (NullPointerException e) {
          return true;
          }
     String b = r.getParameter("anticsrf");
     return !a.equals(b);
     }
```



Update role for existing user

Username admin

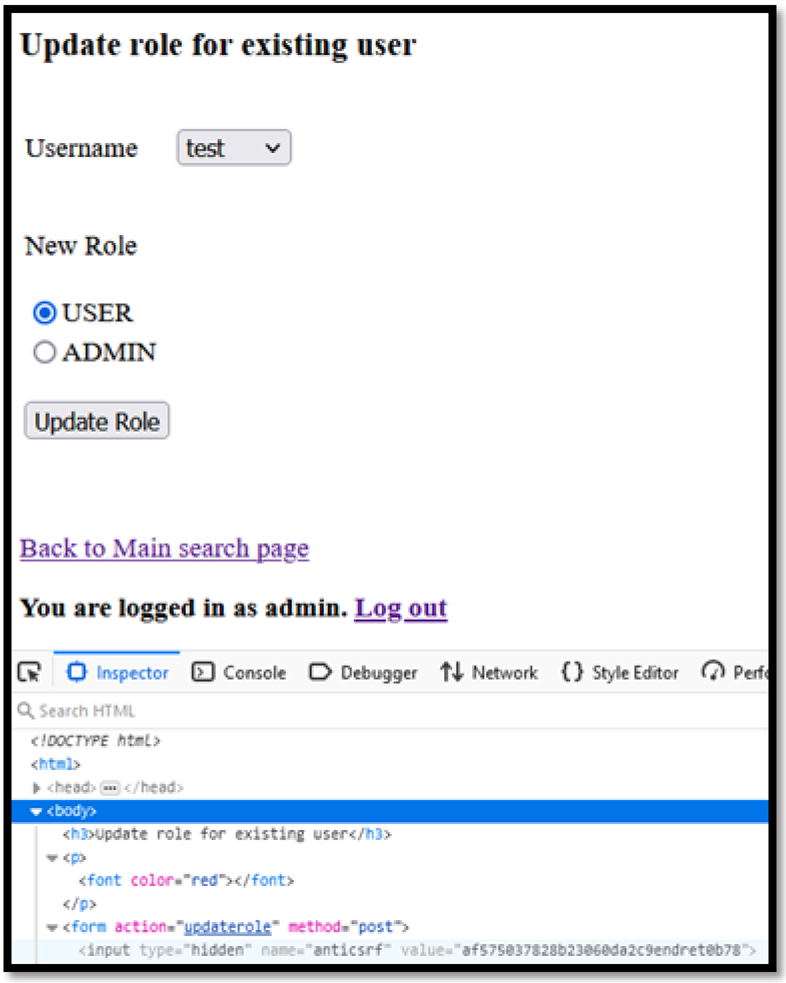
New Role

USER
[✓ ADMIN]

Back to Main search page

You are logged in as admin. Log out

```
<!DOCTYPE html>
<html>
 <head>…</head>
 <body>
   <h3>Update role for existing user</h3>
   <p>
     <font color="red"></font>
   </p>
   <form action="updaterole" method="post">
     <input type="hidden" name="anticsrf" value="af575037828b23060da2c912088d0b78">
```


Update role for existing user

Username test

New Role

USER
ADMIN

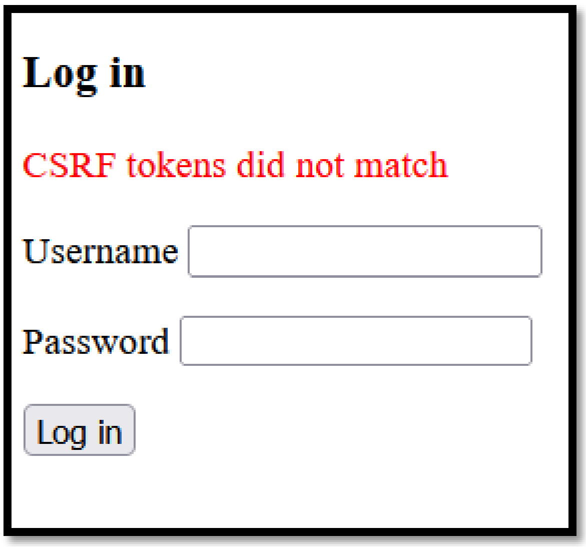
Update Role

Back to Main search page

You are logged in as admin. Log out

```
<!DOCTYPE html>
<html>
 <head> ··· </head>
 <body>
    <h3>Update role for existing user</h3>
  <p>
      <font color="red"></font>
    </p>
  <form action="updaterole" method="post">
      <input type="hidden" name="anticsrf" value="af5750378 28b23060da2c9endret0b78">
```

We manually edit the value of the anticsrf token to check if the server will accept a request where the tokens do not match.


Log in

CSRF tokens did not match

Username

Password

Log in

Vulnerability #5: SSO and weakness in JWT authentication token

Description: SSO and weakness in JWT authentication token

Can a user elevate his privilege in this id_token?

We used https://token.dev/ to easily decode and alter the JWT. We took a standard JWT and were able to remove admin privileges as well as re-implement admin privileges by editing the payload, we were also able to edit the expiration date of the token.

Mitigation:

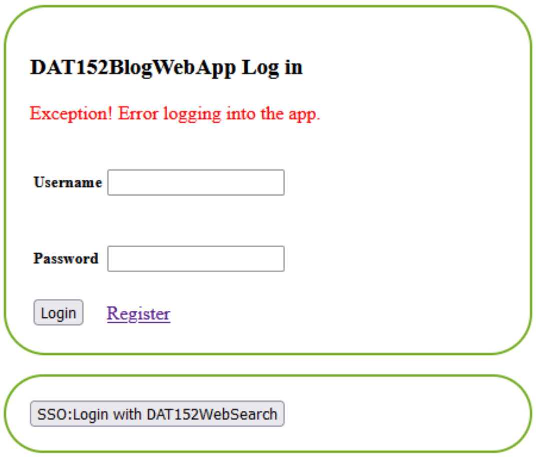
In Token.java **(Line 114)** we added a expiration time for the JWT

```
Date current =(new Date());
jwt.setIat(current);
Date expire = new Date();
expire.setTime(expire.getTime() + TimeUnit.MINUTES.toMillis(5));
jwt.setExp (expire);
```

In the blog.jsp **(Line 43)** we added a call to the verifyJWTAsymmetric method in the JWTHandler.java file to check if the JWT was valid or had been altered.

```
if (RequestHelper.isLoggedInSSO(id_token, pubkeypath)) { code }
```

The result of attempting to use an altered JWT to log in:


DAT152BlogWebApp Log in

Exception! Error logging into the app.

Username

Password