IRRIGATION SURPLUS -
A SECURE WEB APPLICATION

WEB SECURITY COURSE WORK

AHMED MAHMOUD | SALEM SALEM | MOHAMED BERY | OMAR KANDIL
CU2000512 | CU1900953 | CU1901011 | CU1900527

Chapter 1: Introduction

For this course work we are asked to develop a secure web application. The web application of our choice is platform that can help farmers control their water irrigation using data from satellites. As a team, we had some previous experience in web development since we all were enrolled in a web development course. In the previous course we used PHP as the back end (server-side language) but in this project we are planning to use Python and Django. Some of our expected learning outcomes are learn how to use the Django framework, learn basic/advances web security features, Learn the basics of penetration testing (test our website), and learn how to work in a team using git version control.

1.1 Software Development Life Cycle

While creating the web application and the report, the team followed the software development life cycle shown in figure 1. Using this technique helped us organize and distribute the tasks easily among the team members while creating a structured pattern to follow.

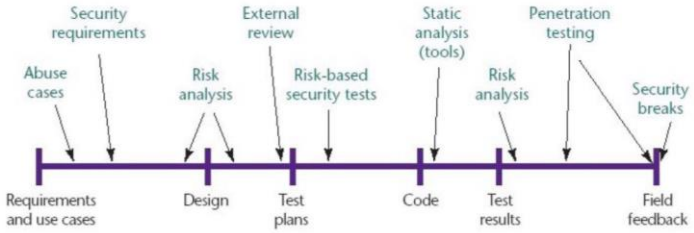


Figure 1 software development life cycle

1.2 The Idea

Maa, the Arabic word for water, is an essential part of our life. Our planet faces many obstacles in a rapidly evolving world that produces a great amount of waste. It is reported that 70% of humans' water consumption comes from the agriculture sector, and of that value 40% is lost during the process. Therefore, we set out to try and alleviate the problem which could potentially save around 30% of the Earth's water.

Our solution aims to solve the problem by giving the agricultural industry a platform that can be used to track the surplus of water for their plots of land. Decreasing the surplus used would greatly help humans conserve water that might be difficult to come by in areas of the world that suffer from droughts.

Our platform combines the information from satellite imagery with open-source data (total of 5 data sources) to produce a water surplus that the user should aim to reduce. Two types of satellite imagery are used to calculate the surplus water. The Sentinel-2 satellite gives us access to the moisture index in the field being studied. The MODIS satellite gives us access to the potential evapotranspiration in the area. These two values being plugged in to the moisture index formula (by C. W. Thornthwaite) $mi = \frac{(S-D)}{PET} \times 100$, would result in getting the surplus water used by the user.

Table of Contents

Chapter 1: Introduction 2
1.1 Software Development Life Cycle 2
1.2 The Idea 2
Chapter 2: Security Requirements & Abuse Cases 5
Chapter 3: Design 6
3.1 Front-end Design 6
3.2 Back-end Design 7
Chapter 4: Test Plans and Risk Analysis 8
4.1 Risks 8
4.2 Test Plans 8
Chapter 5: Code Implementation 9
5.1 Web Application code 9
5.2 The Data Aggregation Algorithm 11
Chapter 6: Implemented Security Features 12
6.1 XSS 12
6.2 Cross-Site Request Forgery 13
6.3 Clickjacking 13
6.4 Injection Attacks 13
6.5 Authentication, Cookies, Session, and password Management 14
6.7 Access Control 14
6.8 File Upload Vulnerabilities 15
6.9 Application-Layer Denial-of-Service Attacks 15
6.10 Web Server Configuration Security 15
Chapter 7: Test Results & Penetration Testing Report 16
Test Results 16
Penetration Testing Report 17
Fixes Done After the Pen-testing Report 21
Chapter 8: Scalability and Maintenance 22
Chapter 9: Conclusion 22

Table of Figures

Figure 1 software development life cycle 2
Figure 2 Color pallet and fonts 6
Figure 3 Logo 6
Figure 4 Adobe XD samples 6
Figure 5 Back-end tech stack 7
Figure 6 layout block 10
Figure 7 URL patterns 10
Figure 8 Views.py 10
Figure 9 modles.py 10
Figure 10 Environment data 11
Figure 11 All fields from login request are safe from SQLi 17
Figure 12-The script tag did not execute, and the user was met with an invalid credentials message 18
Figure 13-OS commands were not executed 19
Figure 14-Server log 20
Figure 15 Removing session ID from request 20
Figure 16-Example of request showing session ID 21

Chapter 2: Security Requirements & Abuse Cases

During the planning process we must take into consideration the globally recognized security threat to develop a more secure code. According to OWASP, the Open Web Application Security Project, the top ten security threats in 2021 are

1- Broken Access Control
2- Cryptographic Failures
3- Injection
4- Insecure Design
5- Security Misconfiguration
6- Vulnerable & Outdated Components
7- Identification & Authentication Failures
8- Software and Data Integrity Failures
9- Security Logging and Monitoring Failures
10- Server-Side Request Forgery

During the development phase we are required to implement solutions to these threats to have a secure environment for our users. To understand how these vulnerabilities are used by malicious attacker we must look through abuse cases for each threat only then we can use the optimum methods patch them. The table below shows the different vulnerabilities and the abuse cases related to them.

Security Threat	Description	Abuse Case
Application Layer DDOS	Malicious behavior that targets layer 7 of the Network OSI Model which handles all the requests ex. Post, get , etc..	Http Flood against a server, the attacker uses multiple PCs to send HTTP requests to the server eventually causing it to collapse
Broken Authentication	With hundreds of millions of valid credentials and password combinations, attackers can easily access a variety of administrative accounts and tools. They can also perform brute force attacks and manage session tokens.	As an attacker, I have a variety of tools and methods that I can use against these administrative accounts. These include dictionary attacks and automated brute force.
Broken Access Control	Capturing access control is a core skill for attackers. They can perform this attack through manual means or through automation.	As an attacker, I can get around access controls by changing the URL, internal application state, or HTML page, or by utilizing a tailored API attack tool.
Cross-Site Scripting (XSS)	In the OWASP Top 10, XSS is the second most common problem, appearing in almost two-thirds of all apps.	As an attacker, I utilize reflected XSS when an application or API outputs HTML with unvalidated and unescaped user input. If my attack is successful, the attacker will be able to execute arbitrary HTML and JavaScript in my victim's browser. Typically, the user will be required to click on a malicious link that directs them to an attacker-controlled page, such as malicious watering hole websites, advertising, or something similar.
Injection	An injection vector can be almost any form of data source, including environment variables, public and private web services, and all types of users. When an attacker may deliver hostile data to an interpreter, this is known as an injection vulnerability.	As an attacker, I will inject SQL, NoSQL queries, OS commands, XML parsers, and SMTP headers into the User or API interfaces' input fields.
Security Misconfiguration	To get unauthorized access of the system, attackers will frequently try to exploit unpatched vulnerabilities or gain access to default accounts, unused pages, unprotected files and directories.	As an attacker, I scan for and exploit security flaws in any element of the application stack, as well as wrongly set permissions on cloud services.

Chapter 3: Design

3.1 Front-end Design

A well-designed website provides good user experience and helps the website visitors access and navigate the website with ease. To create a professional design that looks appealing, we had to go through a few steps shown below. As for the technologies used for the front end we used HTML, CSS, JavaScript, and Bootstrap.

Step	Description
Creating a logo	A good logo is crucial as it catches the eye, makes a good first impression, and serves as the cornerstone of your brand identity. We created a logo that represents our theme by using a drop of water as the main shape for the logo. Figure 3 shows the logo.
Choosing the color palette	We choose 4 main colors to serve as the main color pallet for the website (2 shades of green, 1 shade of black, and white). The 2 green colors represent the agricultural aspect of the project. Figure 2 shows the color pallet.
Choosing the fonts	Choosing the right fonts that match the logo and the color pallet are also important. Without a good font stack, the website will not look as appealing. Figure 2 shows the font stack.
Design the website in Adobe XD for reference	Adobe XD is a free design tool created by adobe in 2016. This tool helped us to create the outline (a template) of the website. Figure 4 shows samples of the Adobe XD design.

MA

Figure 3 Logo

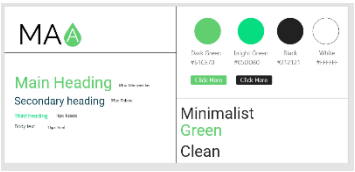


Figure 2 Color pallet and fonts

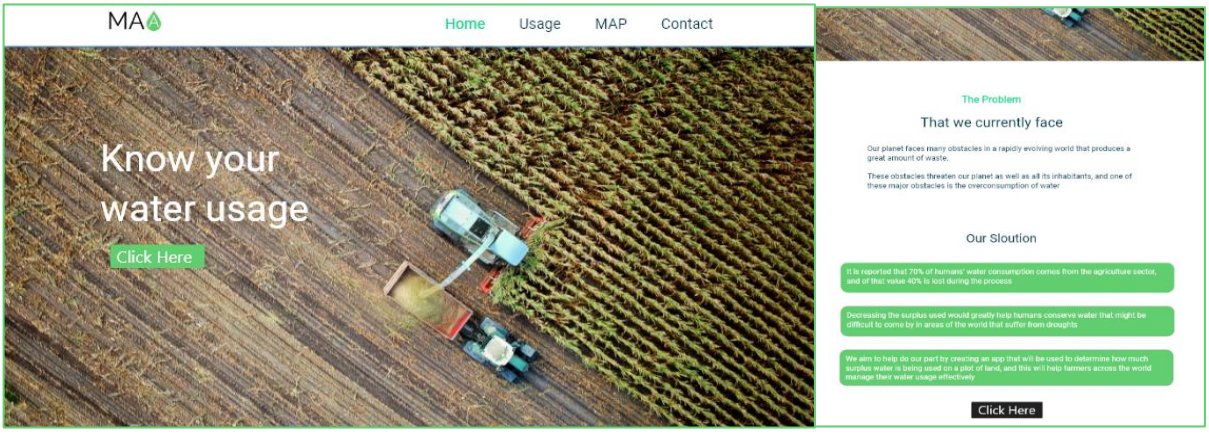


Figure 4 Adobe XD samples

3.2 Back-end Design

The back end of a web application refers to the parts of the code that allow it to run but are not visible to the user. The back end of a web application stores and accesses most data and running logic. A backend stack or tech stack refers to the theologies used in the back end. Figure 5 below shows the tech stack we plan to use along with why we will choose every technology.

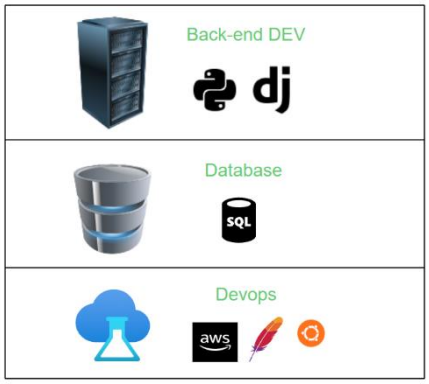


Figure 5 Back-end tech stack

Python & Django

As the main server-side programing language we want to use is Python. Python is a very popular programing language with many libraries these libraires like GDAL helped us achieve our goal with ease. For a web framework we used Django. Django is a high-level Python web framework for creating secure and maintainable websites efficiently. It is built by experienced developers that takes care of a lot of the hassle (a batteries included framework) so we can focus on developing our application instead of reinventing the wheel.

SQL Lite

For the database of choice, we will use SQLite. SQLite can handle HTTP queries with low to medium load. SQLite can compress files into smaller packages with less data. SQLite is a temporary dataset that is used within an application to process data. And since our application does not have a lot of data and SQLite is supported by Django out of the box it is the perfect choice.

AWS, Ubuntu, and Apache

AWS has a free tire package that will allow us to host the web application for free. The server operating system will be Ubuntu and the webserver will be Apache. On Linux platforms, Apache is the most widely used Web server. Client computers request Web pages, which are served by Web servers. Clients commonly use Web browser software like Firefox, Opera, Chromium, or Internet Explorer to request and display Web sites.

Chapter 4: Test Plans and Risk Analysis

4.1 Risks

Through our design, we can tell that there are multiple functionalities that will require user inputs, and all these fields that take user inputs will introduce risks to the webapp. The risks associated with text user inputs are as follows:

- Cross-site-scripting (XSS)
- SQL injection (SQLi)
- OS command injection

Another type of risk is the risk attributed to role-based functionalities, since there are different roles for different users then access control should be highlighted to ensure that no user oversteps their permitted actions on the webapp. Some of the risks associated with this aspect are

- Cross site request forgery (CSRF)
- Broken authentication
- Broken access control

4.2 Test Plans

Testing is crucial to ensure that the webapp is working as intended and is not causing any data leakage or unwanted actions. The best way to test the webapp would be to test the use cases as well as the abuse cases, this way the app is tested to make sure that when the user enters everything correctly, the desired outcome is reached. Testing the abuse cases ensures that if anything is entered incorrectly, it does not create logic errors or disrupt the data in the database in any way.

Test Case	Description
Opening map w/o login	Opening the map page without logging in should produce a prompt for the user to login
Log in w/o registration	The user is given that their login credentials are incorrect without describing which credentials are incorrect
Registration with incorrect inputs	Registration should not be complete without correct inputs
Registration with correct inputs	Registration should be complete, and user is redirected to the website
Login with valid registration	User is logged in and session is started
Opening map with login	Map page should be accessible to user, where they can choose inputs for search request
Submitting search without entering valid coordinates	User should be prompted to re-enter data
Submitting search with valid coordinates	User is redirected to results page with the results displayed and their search count reduced by 1

Logging out	User should not be able to access map page after logout
User fills in contact form with invalid data	User is prompted to re-enter data
User fills in contact form with illegal characters	Characters are escaped and viewed as plain text
User fills in contact form correctly	Message is viewable in admin page and user is told message has been sent

Chapter 5: Code Implementation

Prior to this section we did not write any code, but in this section, we will look at the final code and briefly explain every element of it.

5.1 Web Application code

Django structure

When creating a Django project, it automatically creates a few files and directories. Each file serves a certain purpose. The below table shows some of the files/directories and a brief description of its purpose.

File	Purpose
Settings.py	It holds the Django project's settings. The most significant file is setting.py, which is used to add all the apps and middleware applications. This is the Django project's primary configuration file.
Urls.py	The URL stands for universal resource locator, and it comprises all the endpoints that our website should have. It's utilized to provide you the URLs of resources (pictures, webpages, websites, and so on) that are available on the internet. Simply said, this file instructs Django that if a user enters this URL, they should be sent to that specific website or picture.
Models.py	Models.py is a Python module that provides web application models as classes. It is regarded as the most crucial part of the App's file structure. The database's structure is defined by models. It contains information about the actual design, data set relationships, and attribute limitations.
Views.py	When it comes to the Django app structure, views are equally crucial. Views are the user interface for interacting with a Django web application. It organizes all the views into classes.
Manage.py	This file is utilized in our projects as a command-line tool. We'll utilize this file to debug, deploy, and execute our web apps. The code for starting the server, make migrations or migrations, and a few other commands are all in this file, which we execute in the code editor.
Forms.py	A python file that includes all the forms used in the web application.
Static files	The static folder is a folder that includes all the static files like the images and the CSS.
Templates	Is a folder that includes all the HTML pages of the website.

Templates

The templates folder includes all the html pages of the website. There is a total of 8 HTML files (home, contact, map, results, results error ,usage, and the layout) the layout page is the main structure of the code it includes the headings, navigation bar, and the footer. All the other pages inherit from the layout page and change the main section. This is a feature called html blocks that is available in Django. Figure 6 shows an example of the blocks used.

```
    <!-- ======= Header ======= -->
    <header id="header" class="fixed-top">···
    </header>    <!-- End Header -->

    {% block body %}
    {% endblock %}

    <!-- ======= Footer ======= -->
    <footer id="footer">···
    </footer>
    <!-- End Footer -->
```

Figure 6 layout block

Urls.py

The urls.py includes all the routes available for our web application there are 7 main routes (home, contact, map, results, results error, and usage). Every route calls a certain function the views.py file figure 7 shows the URL patterns used.

```
urlpatterns = [

    path("home/", views.home,name="home"),
    path("usage/", views.usage, name="usage"),
    path("map/", views.map, name="map"),
    path("contact/", views.contact, name="contact"),

    path("login/", views.login, name="login"),
    path("registration/", views.registration, name="registration"),
]
```

Figure 7 URL patterns

Views.py

The views file is the main element of our web application. It includes all the functions that work after a user goes to a certain route. There are 7 mains functions in the views file. figure 8 shows 2 of the functions responsible for the login and registration of users. It is also visible form the registration function that we can do certain tasks according to the type of the request. For example, if the request method is GET then render the registration form and if the method is POST then extract the fields from the form and validate them.

```
def login(request):
    return render(request, "old_templates/login.html")


def registration(request):
    if request.method == "POST":
        user = User()
        user.name = request.POST["Fullname"]
        user.username = request.POST["Username"].lower()
        user.email = request.POST["email"].lower()
        user.password = request.POST["password"]
        user.save()
        return render(request, "old_templates/reg_success.html")
    else:
        return render(request, "old_templates/registration.html")
```

Figure 8 Views.py

Models.py

The models in Django represents tables in the SQLite database for our project the database design has 2 tables one for the users this includes a full name, username, email, and a password and the other table for the number of searches available the relationship between the 2 tables is a one-to-one relationship. Figure 9 shows the models.py file.

```
class Searches(models.Model):
    user = models.OneToOneField(User,
                                on_delete=models.CASCADE,
                                null=True,)

    avsearches = models.IntegerField(null=True)
    def __str__(self):
        stringav = str(self.avsearches)
        return stringav
```

Figure 9 modles.py

5.2 The Data Aggregation Algorithm

The webapp utilizes 5 different data sources to calculate the water surplus for a given coordinate. To get a correct value for the water surplus, each data source is used to calculate certain values and each value is used as input for information to that will be displayed to the user.

Environment Data

The first two data sources are ones used to determine the soil type (dry, humid, and wet), region (desert, semi-arid, semi-humid, and humid), temperature (in ranges of 10 degrees Celsius), and weather condition (normal, sunny, windy, and rainy) of the coordinates. The first data source is several arrays that denote the soil type and region, which can be visualized in the figure 10.

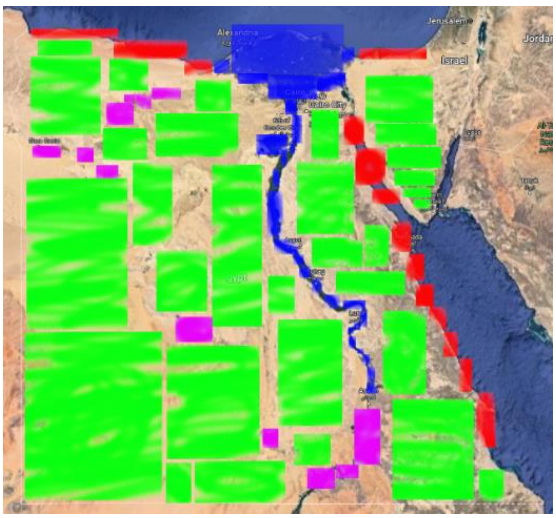


Figure 10 Environment data

Each color in the figure shows a certain region and soil type combination where green is dry soil and desert region, blue is humid soil and semi-humid region, red is humid soil and humid region, and purple is dry soil and semi-arid region. This information will be crucial when getting the water requirements for the crop and coordinates.

The next data source is a weather API that can be used to extract the weather conditions and temperature at the coordinates. An API call is made, and the returned data is parsed so that it fits one of the categories discussed earlier.

After getting the information from the weather API, we have all the data to match the coordinate attributes to the third data source which is the data set containing the water requirements for all combinations of crops, soil types, regions, temperature, and weather conditions. All the above will be displayed to the user in the results page.

Satellite Data

The water surplus is calculated using the formula for moisture index by C.W. Thornthwaite $S = \frac{mi \times PE}{100}$, and there are two values in that equation that need to be extracted to be able to get the surplus. The first value, mi, can be extracted from Sentinel-2 satellite imagery of bands B8A and B11; making this the fourth data source. The second value, PET, can be extracted from the MODIS satellite imagery, which is the fifth and final data source used to calculate the required surplus water value.

Chapter 6: Implemented Security Features

Since this is a web security course, an important section of our project is the implemented security features. We used the WEB SECURITY A Whitehat Perspective book as a guideline for this section of the project. The book covers 10 different chapters every chapter is a certain aspect of web security. The below elements represent the 10 chapters and how we went by issuing every one of them.

6.1 XSS

XSS attacks enable an attacker to inject client-side JavaScript code into other users' browsers. This is commonly accomplished by putting malicious scripts in a database, from which they may be accessed and presented to other users, or by convincing users to click a link, causing the attacker's JavaScript to be performed by the user's browser. XSS attacks, on the other hand, can come from any untrusted data source, such as cookies or web services, if the data is not sufficiently sanitized before being included in a page. To prevent XSS attacks in our application, we did the following

- Added front end validation

  Added validation with regex to remove any symbols using JavaScript and HTML

- Added back-end validation

  This is because front end validation is easy to mitigate

- Did not store any html in the database

  Doing so can enable an attacker to create a stored XSS attack

- Did not use the HTTP response method

  This can allow an attacker to perform a reflected XSS attack

- Enabled Auto escape Globally

  Is a built-in feature in Django that automatically escapes any special characters

6.2 Cross-Site Request Forgery

Cross-site request forgery (commonly known as CSRF) is a web security flaw that allows an attacker to trick users towards doing things they don't want to do. It allows an attacker to partially bypass the same origin policy, which is intended to keep websites from interacting with one another. To prevent CSRF attacks in our web server we did the following

- Used Django's CSRF protection in all the forms on the website

  Django prevents CSRF attacks by producing a CSRF token on the server, sending it to the client, and requiring the client to provide the token in the request header. The server will next check if the token provided by the client matches the one produced before; if it does not, the request will be denied. Due to the Same-Origin Policy's security, attackers will be unable to access this token.

6.3 Clickjacking

A clickjacking attack occurs when a malicious website frames another website. An unknowing user may be duped into doing undesired activities on the target site because of this attack. To prevent this in our web security we did the following

- Used Django's X-Frame-Options middleware

  The X-Frame-Options middleware in Django provides clickjacking protection by preventing a site from being rendered inside a frame in a supporting browser. On a per-view basis, you can disable the protection or customize the exact header value sent.

6.4 Injection Attacks

An injection attack is when malicious code is injected into the website and the attacker receives all the data from the database. The number one online application security risk in the OWASP Top 10 is this attack type, which is considered a major problem in web security. There are different types of injection attacks but for this section we focused on SQL injection and OS command injection. To prevent the 2 attacks, we did the following

- Used Django query sets

  Django's query sets are protected from SQL injection since their queries are constructed using query parameterization. A query's SQL code is defined separately from the query's parameters. Since parameters may be user-provided and therefore unsafe, they are escaped by the underlying database driver.

- Added front end and back-end validations

  The same as the XSS validation

- Enabled Auto escape Globally

  The same as the XSS validation

6.5 Authentication, Cookies, Session, and password Management

Session security

Subdomains inside a site can set cookies for the entire domain on the client. If cookies from subdomains not controlled by trusted users are allowed, session fixation is possible. An attacker may, for example, enter onto good.example.com and get a legitimate session for their account. Because a subdomain is allowed to put cookies on *.example.com, if the attacker has access over bad.example.com, they may use it to relay their session key to you. You'll be logged in as the attacker when you visit good.example.com, and you may mistakenly submit sensitive personal data (such as credit card information) into the attacker's account. If excellent, another possible attack would be. When example.com's SESSION COOKIE DOMAIN is set to "example.com," session cookies from that site are delivered to bad.example.com.

Cookies

Cookies are commonly used to hold session IDs or access tokens, which are the holy grail of attackers. Attackers can mimic users or elevate their privileges on your application once they've been exposed or hacked. To counter any cookies related attack we did the following

- Added expiry date and a max age

  To inures that the same cookie is not always valid. This is helpful in situations were if the cookie was stolen the attacker does not have limitless access to the compromised account

Password Management

To inure that all the users use a strong password we set the following requirements

- The password cannot be similar to the username, name, or email
- The password must be at least 8 characters
- The password cannot be a commonly used password
- The password cannot be only numerical
- The password saved in the database are hashed using the sha256 hashing algorithm with salts

6.7 Access Control

Users in a system are granted restricted access based on their organizational function. This functionality refines and secures the application's access. Django comes with authentication and authorization built in. These characteristics may be used to create a Role Based Access Control. User, Group, and Permission are all built-in models in Django user authentication. This provides the application's granular access control.

6.8 File Upload Vulnerabilities

For this certain chapter we did not implement any security features to address it because our web application does not have any file uploads. Therefore, these types of vulnerabilities are not applicable in our situation.

6.9 Application-Layer Denial-of-Service Attacks

To prevent DDOS attacks including http/https flooding attacks we used cloud flare. cloud flare is content delivery network (CDN). CDNs are used to make websites faster but with cloud flare they also add a application firewall (layer 7 firewall) this firewall can detect any DDOS attacks and prevent it. Also cloud flare have a free tire package that come with the application layer firewall out of the box.

6.10 Web Server Configuration Security

Another side of security is the server security. To ensure that the server itself is secure we did the following

- Removed unnecessary services
- Set permissions and privileges
- Created separate environments for development and production
- Added different permissions and privileges

Chapter 7: Test Results & Penetration Testing Report

Testing is a vital part of any web development project; it ensures that the project is up to standards when it comes to functionality and security level. The test cases were discussed previously in the report and included both abuse cases and use cases. There is also a penetration testing report conducted on the project to ensure that there are no security vulnerabilities that can lead to cyber-attacks.

Test Results

The same table used for the test plans will be used to display the results of the tests. The tests were conducted on the latest version of the app, and so any problems found during testing will need to be addressed in future updates.

Test Case	Result	Status
Opening map w/o login	User cannot access map without logging in	Success
Log in w/o registration	User cannot log in and is told to enter correct credentials	Success
Registration with incorrect inputs	Registration is not complete until the correct data is entered	Success
Registration with correct inputs	Registration is complete, and user is redirected to the website	Success
Login with valid registration	User is logged in and session is started	Success
Opening map with login	User can enter inputs and send a request	Success
Submitting search without entering valid coordinates	User is told to enter valid data	Success
Submitting search with valid coordinates	User is taken to results page and displayed values	Success
Logging out	User is prompted to login to view map again	Success
User fills in contact form with invalid data	User is prompted to re-enter data	Success
User fills in contact form with illegal characters	Characters are escaped and viewed as plain text	Success
User fills in contact form correctly	Message is viewable in admin page and user is told message has been sent	Success

All test cases were successful which means that since the beginning of the development phase the correct measures were taken to ensure that the project completes the required functionality while not breaking under invalid inputs.

Penetration Testing Report

The main body of the pen testing report will be a table of known vulnerabilities for webapps of this sort and the result of the testing done to ensure the webapp is safe. Then there will be examples for each type of vulnerability tested.

Vulnerability	Tests conducted	Results
SQL Injection	Tested login request, map page request, and registration request for union/blind injection	No vulnerabilities found
XSS	Tested all input fields for XSS vulnerability	No vulnerabilities found
OS Command injection	Tested all input fields for OS Command injection vulnerability	No vulnerabilities found
Access control circumvention	Tested method-based access control circumvention techniques	No vulnerabilities found but error page is given
User ID leakage	Requests searched for user ID	No vulnerabilities found

Examples for SQL Injection:

SQLmap was used to test all three requests that could be vulnerable to SQLi and as seen in figure 1 there were no injectable fields. This was true for all requests tested.

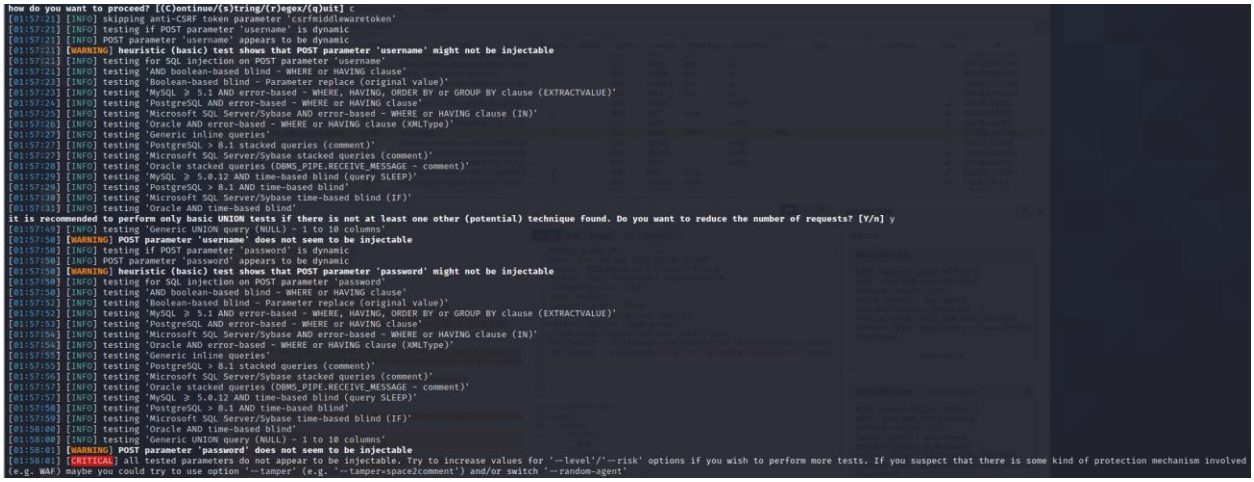


Figure 11 All fields from login request are safe from SQLi

Example for XSS:

Used <script>alert(1);</script> to make sure that input fields sanitize tag characters, this was the case with all input tags.

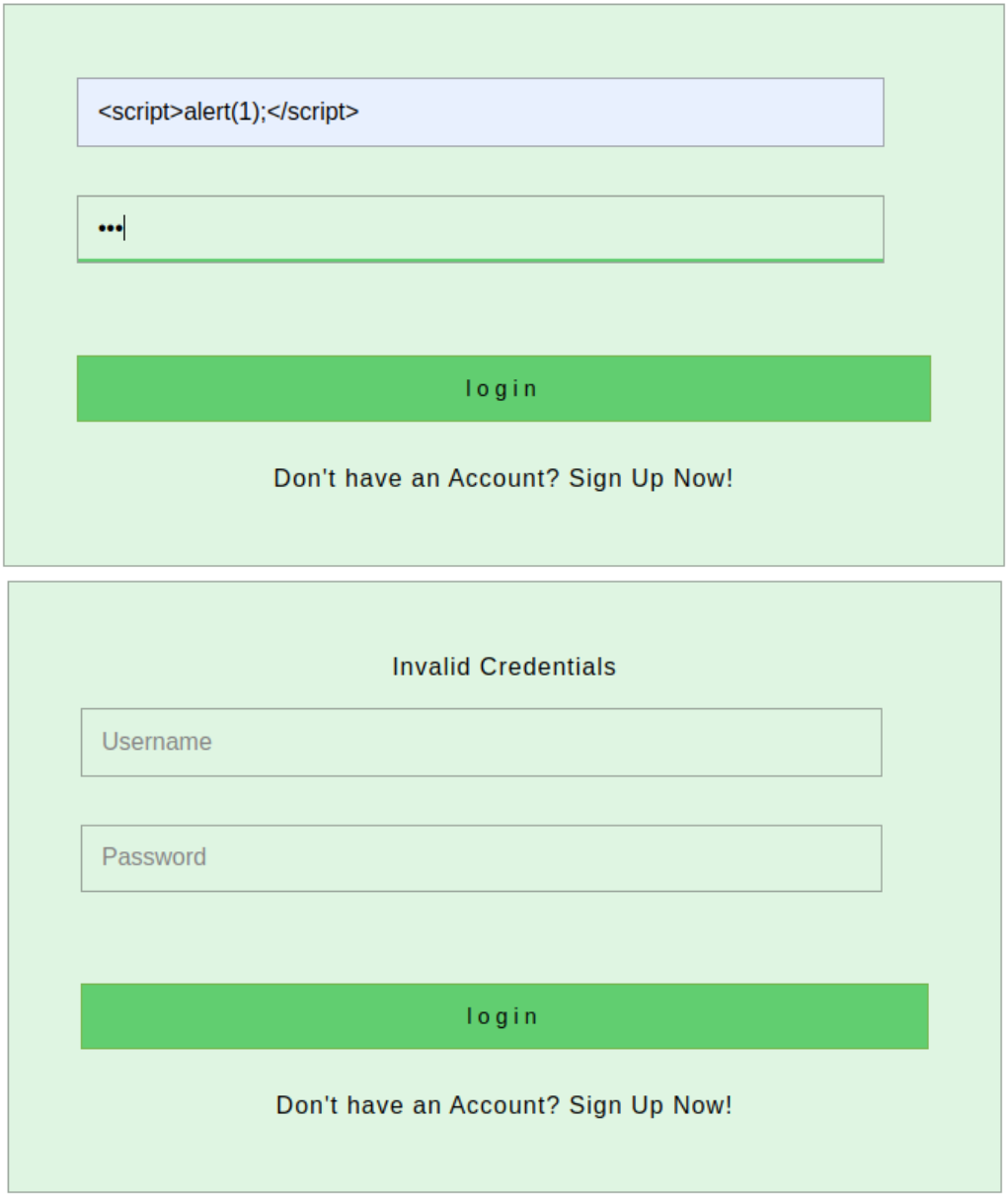


Figure 12-The script tag did not execute, and the user was met with an invalid credentials message

Examples for OS Command Injection

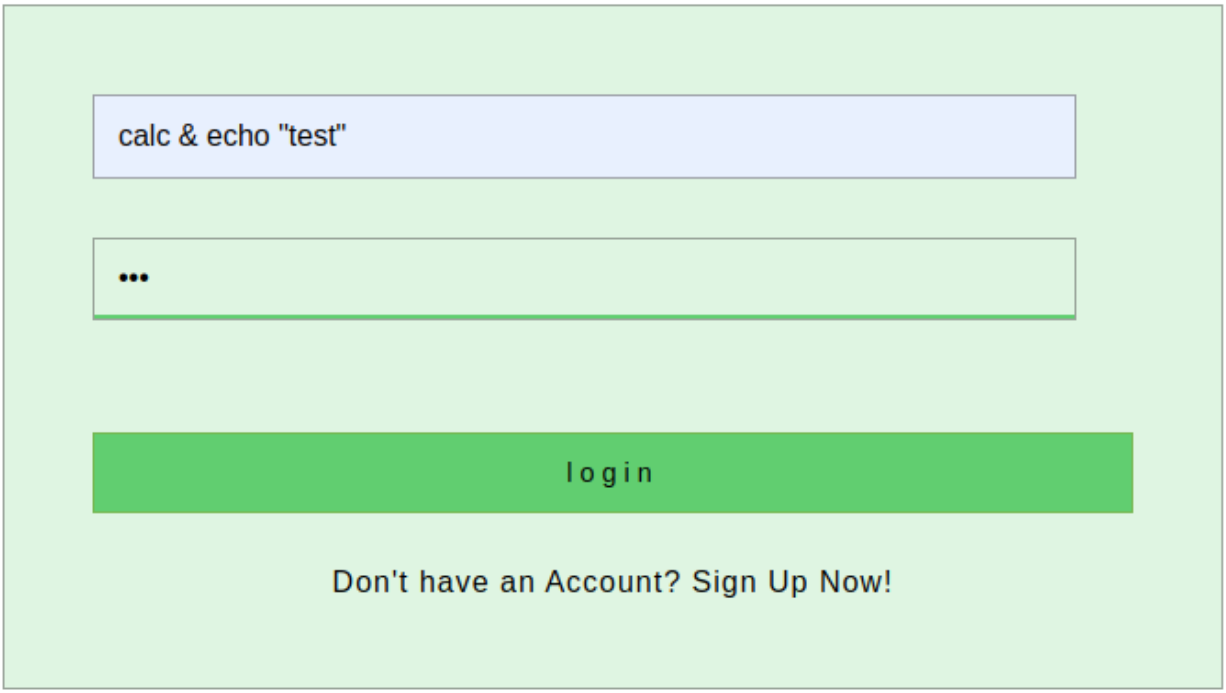


Figure 13-OS commands were not executed

Same as the XSS example, the commands tested in the login did not execute, this can further be confirmed by looking at the server log in figure 14:

```
[08/May/2022 08:09:51] "GET /water/registration/ HTTP/1.1" 200 8985
[08/May/2022 08:09:51] "POST /water/registration/ HTTP/1.1" 200 9210
[08/May/2022 08:09:51] "GET /water/registration/ HTTP/1.1" 200 8985
[08/May/2022 08:09:51] "POST /water/registration/ HTTP/1.1" 200 9210
[08/May/2022 08:09:51] "GET /water/registration/ HTTP/1.1" 200 8985
[08/May/2022 08:09:51] "POST /water/registration/ HTTP/1.1" 200 9210
[08/May/2022 08:16:21] "GET /water/logout/ HTTP/1.1" 200 11666
[08/May/2022 08:16:22] "GET /water/login/ HTTP/1.1" 200 7116
[08/May/2022 08:16:43] "POST /water/login/ HTTP/1.1" 200 7174
[08/May/2022 08:20:10] "GET /water/login/ HTTP/1.1" 200 7116
[08/May/2022 08:21:28] "POST /water/login/ HTTP/1.1" 200 7174
[08/May/2022 08:25:39] "POST /water/login/ HTTP/1.1" 200 7174
[08/May/2022 08:25:58] "GET /water/login/ HTTP/1.1" 200 7116
[08/May/2022 08:26:50] "POST /water/login/ HTTP/1.1" 200 7174
```

Figure 14-Server log

The figure does not show that the calculator was launched, nor did it echo “test” and so there is no risk of OS command injection.

Example for Method-Based Access Control Circumvention

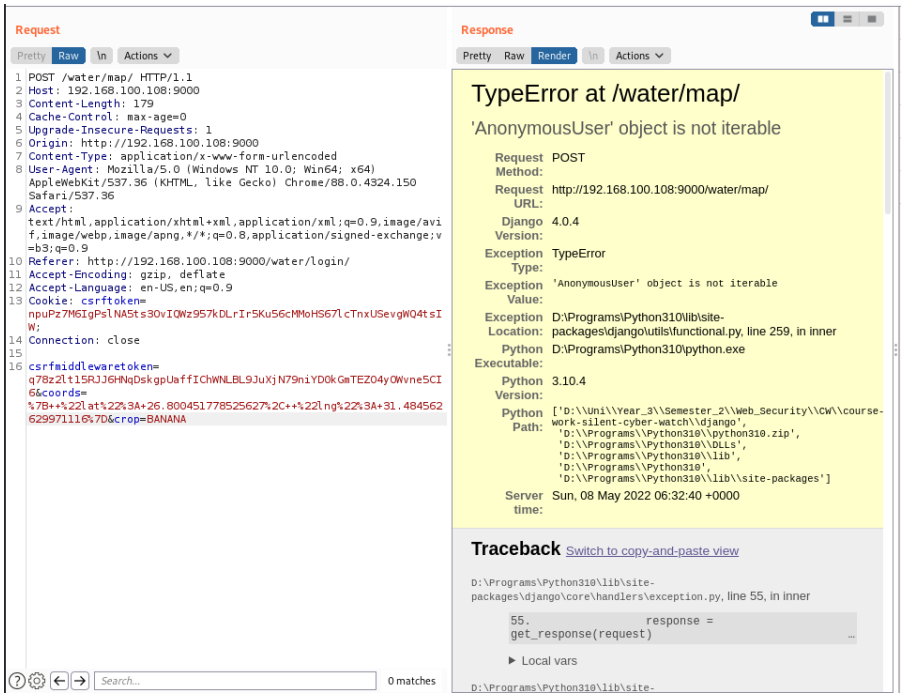


Figure 15 Removing session ID from request

Removing the session ID from the request lead to the object being requested not being the same as when a logged in user sends the same request and so an error page shows up. This will not lead to any data leakage or access to pages that the user should not have access to, however the error page should be changed to redirect to the login page so that the user can access the data properly.

Examples for User ID leakage

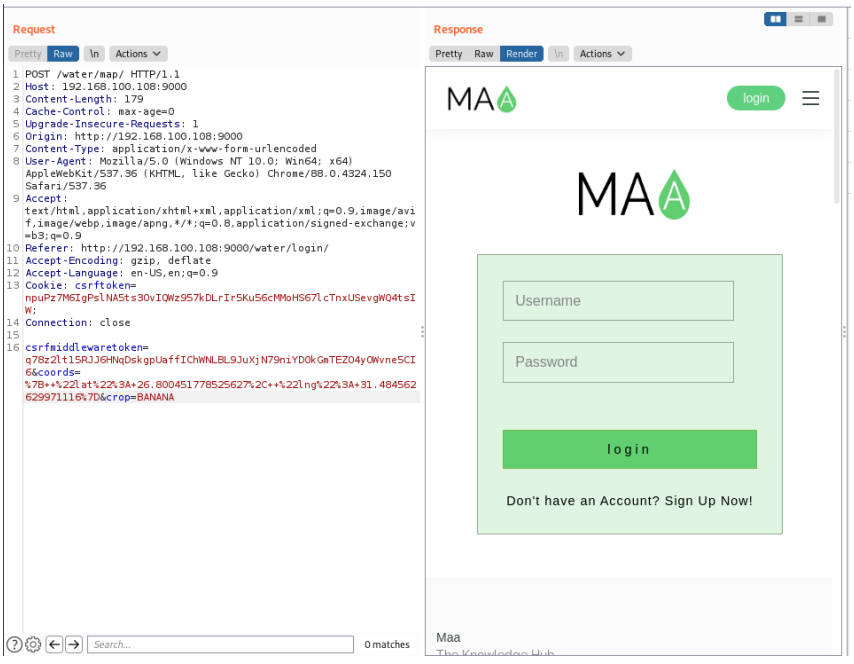
```
POST /water/map/ HTTP/1.1
Host: 192.168.100.108:9000
Content-Length: 179
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://192.168.100.108:9000
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.150 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=0.9
Referer: http://192.168.100.108:9000/water/login/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: csrftoken=npuPz7M6IgPslNA5ts3OvIQWz957kDLrIr5Ku56cMMoHS67lcTnxUSevgWQ4tsIW; sessionid=cbzhgfrvoc71pn1gdyxe7glp2d8ojrnf
Connection: close

csrfmiddlewaretoken=q78z2lt15RJJ6HNqDskgpUaffIChWNLBL9JuXjN79niYD0kGmTEZO4yOWvne5CI6&coords=%7B++%22lat%22%3A+26.8004517785256 27%2C++%22lng%22%3A+31.484562629971116%7D&crop=BANANA
```

Figure 16-Example of request showing session ID

As seen in figure 16, the requests done throughout the app use session IDs that are not iterable and so an attacker would not be able to find and exploit a user ID to bypass the access control methods set in place.

Fixes Done After the Pen-testing Report

Method-based access control circumvention



The map page was fixed so that when the user is not authenticated, they would be sent to login screen.

Chapter 8: Scalability and Maintenance

Scalability is all about being able to handle expansion. A successful web application must be able to scale up and down in a fast and efficient manner. A scalable web application can accommodate an increase in users and load without causing issues for the end users. To make sure that our web application is scalable we plan on doing the following

- Use cloud storage
- Use load balancers
- Use caching effectively

Chapter 9: Conclusion

In summary the team was able to build a fully functional web application with a great front-end and back-end design. The team also was able to implement many security features to make sure the end-user has the best experience. The entire project was created with the software development life cycle in mind.