

Project Topic:

GO GAME

Authors:

Sneha Wattamwar (G01209653)
Devika Walavalkar (G01211392)

Course - SWE 681 001 / ISA 681 001 (Fall 2020)
(Taught by Professor Dr. David A. Wheeler)

Dated December 9, 2020

Table of Contents:

1. INTRODUCTION
 - 1.1. Architecture
 - 1.2. Components
2. DESIGN AND ARCHITECTURE
3. INSTALLATIONS AND OPERATING INSTRUCTIONS
4. GAME RULES
5. ASSURANCE CASES
 - 5.1. SSL/TLS Implementation
 - 5.2. SQL Injection
 - 5.3. Cross-Site Request Forgery (CSRF)
 - 5.4. Sensitive data exposure – Password Storage
 - 5.5. Sensitive data exposure – Failed Login
 - 5.6. Session Management
 - 5.7. Click Jacking
 - 5.8. Cross Site Scripting
 - 5.9. Input Validation
 - 5.10. Bandit
 - 5.11. OWASP ZAP
6. CONCLUSION
7. REFERENCES

1. INTRODUCTION

Go is one of the oldest board games originating from ancient China. It is a strategy game played among two players by taking turns to place white and black stones on the board. The objective of the game is to surround more territory than the other player. This is possible by capturing your opponent's stones by completely surrounding them.

The game starts with an empty board. Thereafter, each player alternately makes the move. Player can place a stone on unoccupied positions. Once a stone is placed you cannot move it unless it is captured and taken off the board. Go is usually played on various grid sizes like 9*9, 13*13 and 19*19. The game has been implemented for a 9*9 grid.

We have covered security requirements expected from the project using knowledge gained during the course ISA681. The game is played over a secure TCP/IP connection between two players and maximized its security by minimizing the attack surface and implementing several other secure software practices like input validation using regular expressions, protection against SQL injection and use of salted hashes for storing passwords/sending data over secure connection.

2. DESIGN AND ARCHITECTURE

This section includes the description of the major components that are a part of the game's functionality. It gives an abstract idea of the implementation structure.

1.1 Architecture

Our game includes a client-server architecture. We have used the Flask Python framework, Jinja 2 template engine and SQLAlchemy as the Object Relational Mapper for interacting with SQL databases. Frontend of this application uses technologies - HTML, CSS, jQuery and AJAX to send RESTful requests to the Flask backend. The database used is sqlite3.

The primary reason for choosing the Flask Framework is the availability of the Flask-Security extension to quickly add common security mechanisms to our application. It even provides a built-in module Flask-WTF which allows integration with WTForms, provides global CSRF protection and secures form using CSRF token. Further, Flask-Login provides user session management, handles the common tasks of logging in, logging out, and remembering your user's sessions over extended periods of time. Since the focus of this project is more on the security aspect, we decided to go ahead with the Flask framework and leverage its authentication and authorization security features along with protection against various attacks.

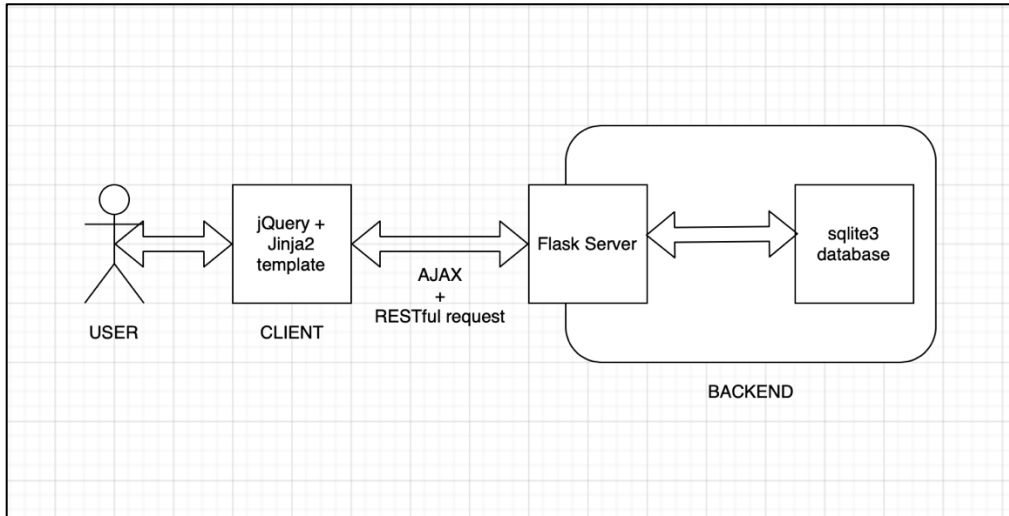


Figure 1. Client-Server Architecture

1.2 Components

- app folder
 1. routes.py

This file includes all the methods which are mapped to a specific route. Some of the methods are `create_game`, `join_game`, `update_board` and `stop_game`. They have a route () decorator that includes the URL and HTTP methods.
 2. piece.py

This file includes the Piece class which contains two types – `LightPiece` and `DarkPiece`. These types represent the two players in the game.
 3. model.py

This file includes data model class like `User`, `Game` and `GameMove`. SQLAlchemy maps these classes to the database tables.
 4. gameboard.py

This file includes the `GameBoard` class with its methods. These methods initialize the gameboard, check for valid moves and the board bounds.
 5. forms.py

This file imports Flask-WTF extension. It allows us to define classes for `LoginForm` and `SignUpForm`.
 6. init.py

This file includes all import statements of the flask libraries along with the initialization of their objects. The libraries that are imported here are the SQLAlchemy, LoginManager, CSRFProtect and Bcrypt.

- templates folder
 1. base.html
This file includes the header links of the pages like Home, Logout, Login
 2. login.html
This file renders the login page with login fields which allows existing user to login into the application
 3. signup.html
This file renders the signup page which allows new users to register with credentials such as username and password.
 4. gameboard.html
This file renders the board of the GO game
 5. games.html
This file renders all the games that a user can join
 6. index.html
This file includes the option to create a game, join an existing game, see the global and individual stats, see the logs of completed games.
 7. game.html
Lists the games which a user can join
 8. game_moves
List the moves played by the players in a completed game
- static folder
 1. javascripts
This folder includes the main.js file which binds actions of the board with the board html file. It even routes the URL as per the action by the player
 2. css
This folder contains all the css files required to render board and other pages

3. INSTALLATION AND OPERATING INSTRUCTIONS

1. Install python
brew install python (For Mac)

2. Install venv package
python3 -m venv venv
3. Create a virtual env
virtualenv venv
4. Activate the venv in order to tell the system to use it
source venv/bin/activate
5. Install flask in the virtual env
pip install flask
6. Install the remaining dependencies
pip install -r requirements.txt
7. Execute the flask application
flask run --cert=cert.pem --key=key.pem
8. Open browser and enter the URL
https://127.0.0.1:5000/

4. GAME RULES

- The game starts with an empty 9*9 board grid
- Each player alternately makes a move by placing a stone on a vacant position on the board
- Each player can either make a move or they can pass their turn
- If a player makes two successive passes, then the game ends and the player who occupies the most territory wins the game
- The player's territory comprises of the count of positions he has surrounded
- A stone can be removed only when the opponent player occupies all adjacent intersections.
- Following is a screenshot for the Audit logs after the game is completed.
- We can view all the moves played during the game and the final score along with the user.

Game: [Home Logout](#)

Moves for game103

Turn	Player Name	(X,Y)	Color	Action	Time
	testuser	(None, None)	None	Created	2020-12-10 02:20:24.578816
	testuser	(None, None)	None	StartGame	2020-12-10 02:20:24.578816
	testuser	(1, 0)	dark	Moves	2020-12-10 02:20:24.578816
	demo-999	(4, 1)	light	Moves	2020-12-10 02:20:24.578816
	demo-999	(4, 1)	light	Moves	2020-12-10 02:20:24.578816
	testuser	(5, 0)	dark	Moves	2020-12-10 02:20:24.578816
	testuser	(5, 0)	dark	Moves	2020-12-10 02:20:24.578816
	demo-999	(5, 2)	light	Moves	2020-12-10 02:20:24.578816
	demo-999	(5, 2)	light	Moves	2020-12-10 02:20:24.578816
	testuser	(6, 3)	dark	Moves	2020-12-10 02:20:24.578816
	testuser	(6, 3)	dark	Moves	2020-12-10 02:20:24.578816
	testuser	(None, None)	None	Gameover	2020-12-10 02:20:24.578816

Winner : testuser

Final Score

testuser : 3

demo-999 : 2

5. ASSURANCE CASES

In this section we will be providing a list of assurance cases along with respective claims and evidence. We will be stating arguments concerned to various security weaknesses and justify our defense against vulnerabilities arising from the same. It covers various security requirements imposed in this project and provides screenshots to ensure that they have been successfully satisfied. We will see the implemented protections against a list of common vulnerabilities which will answer the prime question ‘Why you believe it’s secure?’

5.1 SSL/TLS Implementation

5.1.1 CLAIM

SSL/TLS refers to a protocol for securing communications between a client and a server. It provides privacy and data integrity for building secure web apps over the internet [CLOUDFLARE2020].

As it encrypts data sent over the Internet it ensures that eavesdroppers and hackers are unable to see what you transmit which is particularly useful for private and sensitive information such as passwords, credit card numbers, and personal correspondence. [INTERNETSOC]

The most common way to know if your browser is connected via TLS is to check if the URL in your address starts with “https” and has a padlock showing that your connection is secure. Most browsers today support TLS by default. We have enabled TLS connection in our application to securely transfer information between the client and server.

5.1.2 EVIDENCE

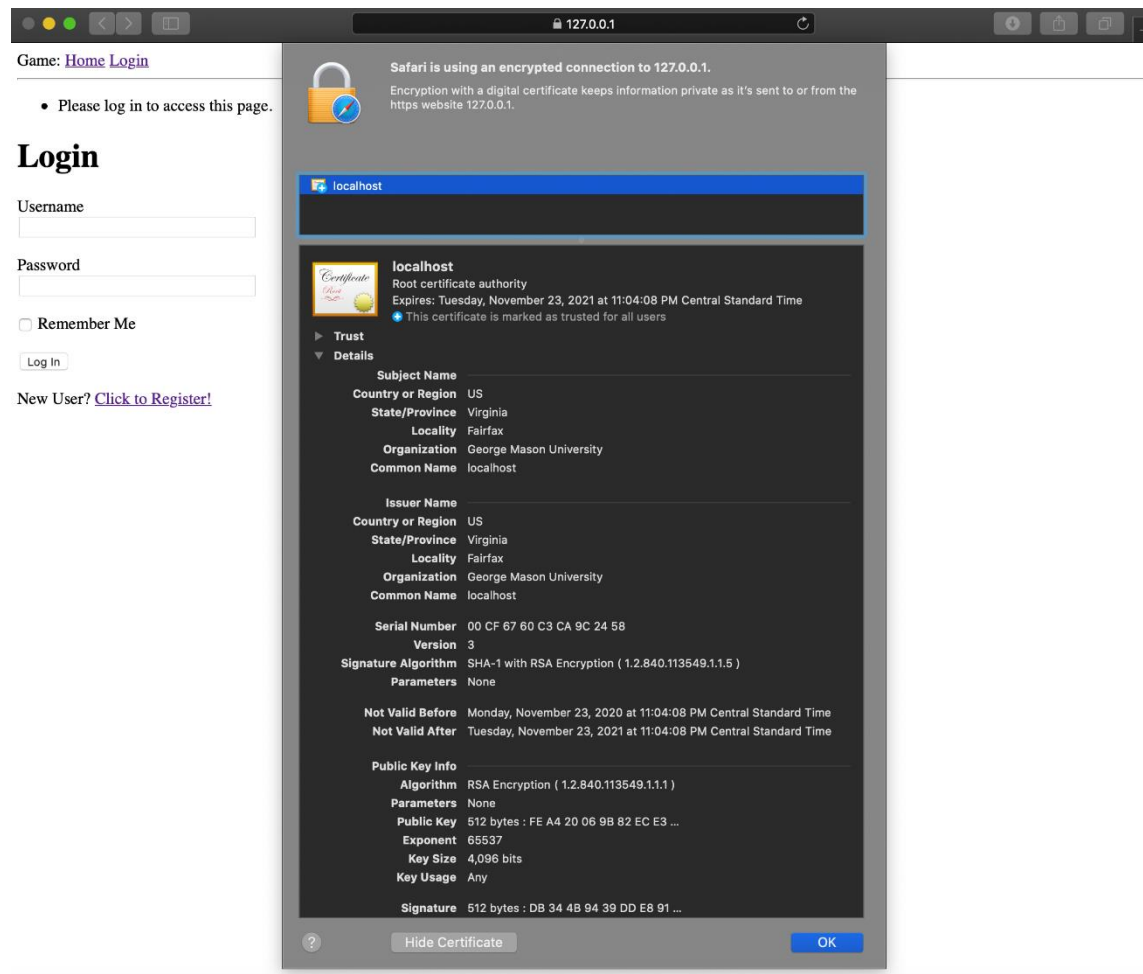
In order to implement encryption and security through the TLS protocol we have setup a secure and encrypted Flask server.

We created a self-signed certificate with its corresponding private key using the openssl command.

When the client establishes a connection with the server and requests an encrypted connection, the server responds with its SSL Certificate. This certificate acts as identification for the server, as it includes the server name and domain. This even ensures that the information provided by the server is correct, the certificate is cryptographically signed by a certificate authority, or CA. As the client knows and trusts the CA, it can confirm that the certificate signature indeed comes from this entity, and with this the client can be certain that the server it is connected to is legitimate. [GRINBERG2017]

- We created a self-signed certificate cert.pem and its corresponding private key key.pem, with a validity period of 365 days.
- As this certificate was created locally, the Certificate Authority is able to trust it. Thus, we are able to set up a secure HTTPS connection on our local machine.
- The flask command makes use of the certificate and private key for execution of the code
flask run --cert=cert.pem --key=key.pem

- The certificate uses RSA algorithm with key size of 4096 bits.
- The certificate even shows details like the Country, State and Time of creation.



5.2 SQL Injection

5.2.1 CLAIM

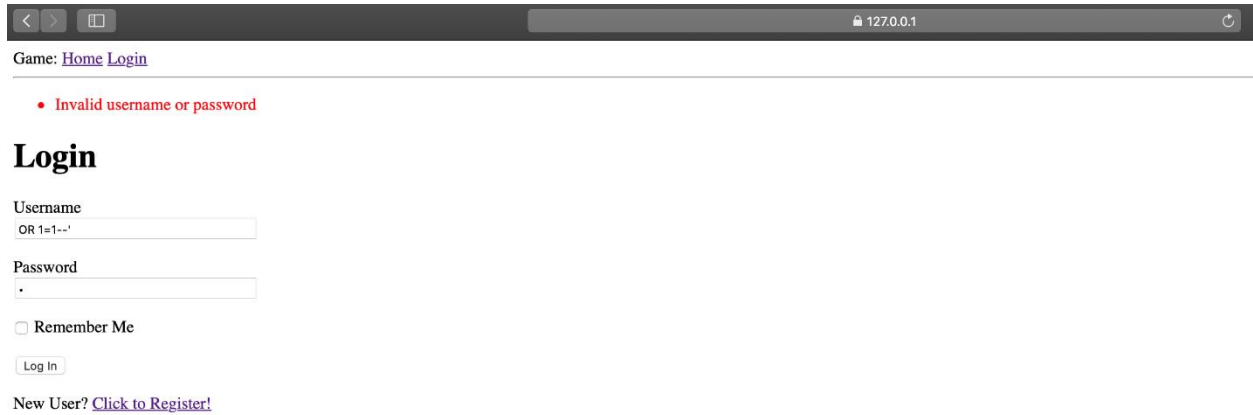
SQL Injection attack consists of insertion or “injection” of a SQL query via the input data from the client to the application. [SQLOWASP2020]

It can attack users by injecting SQL commands via user input form and have them executed on the server. These SQL commands perform harmful actions like reading sensitive data, modifying the data stored in a database, and performing unauthorized administrative tasks in the database server. We have implemented mechanisms for securing our application from such attacks. [Bogoev2020]

5.2.2 EVIDENCE

Flask framework provides an extension Flask-SQLAlchemy, that is used as an Object Relational Mapper. This feature, under the hood, escapes any parameters and/or special

characters that would be interpreted as part of valid SQL commands. The queries are constructed using query parametrization. By default, SQL Alchemy quotes special characters such as semicolons or apostrophes, which prevents SQL injection-style attacks on our application.



The screenshot shows a web browser window with a dark header bar containing navigation icons and the address '127.0.0.1'. Below the header, the page content includes a link 'Game: [Home](#) [Login](#)'. A red error message 'Invalid username or password' is displayed. The main heading is 'Login'. Below it are two input fields: 'Username' with a hint 'OR 1=1--' and 'Password' with a masked character. There is a 'Remember Me' checkbox and a 'Log In' button. At the bottom, it says 'New User? [Click to Register!](#)'.

5.3 Cross-Site Request Forgery (CSRF)

5.3.1 CLAIM

Cross-Site Request Forgery (CSRF) is a type of attack that occurs when a malicious web site, email, blog, instant message, or program causes a user's web browser to perform an unwanted action on a trusted site when the user is authenticated. [CSRFOWASP] In this the attacker uses the user's authentication credentials to execute unwanted actions. In order to secure our application against CSRF we verify the `csrf_token()` against a hidden field.

5.3.2 EVIDENCE

We have protected our application by using the CSRF extension of Flask form. We prevent such attacks by using a CSRF token to validate the request origin. For unsafe requests with unwanted consequences, like an HTTP POST form submission, we send a valid CSRF token for the server to verify the source of the request. Thus, we have provided CSRF protection for our application.

- First, we enable CSRF protection globally for a Flask app by registering the CSRFProtect extension.

```

from flask_wtf.csrf import CSRFProtect
from flask_bcrypt import Bcrypt

app = Flask(__name__)
app.config.from_object(Config)
db = SQLAlchemy(app)
login = LoginManager(app)
bcrypt = Bcrypt(app)
csrf = CSRFProtect(app)

```

- Then we render a hidden input with the token in the Flask Form.

```

<form method="post">
    <input type="hidden" name="csrf_token" value="{{ csrf_token() }}" />
</form>

```

- Finally, add the X-CSRFToken header while sending an AJAX request

```

$.ajaxSetup({
    beforeSend: function(xhr) {
        xhr.setRequestHeader('X-CSRFToken', csrf_token);
    }
});

```

- This shows the csrf_token() value rendered on the game screen

```

</header>
<form action="/create_game" method="POST">
  <input type="hidden" name="csrf_token" value="ImIzMTY5YjJmZTFiZTIzNzg2YTFiZDM5OGNhODcwNTl1M2JhYWxkZDIi.X9Etfw.iCF7oLLasi3KV10YmtLbP6JTkuA">

```

This mechanism provides global protection against CSRF attacks to all the REST calls.

5.4 Sensitive data exposure – Password Storage

5.4.1 CLAIM

This security threat occurs when the web application doesn't adequately protect sensitive information like session tokens, passwords, banking information, or any other similar crucial data which, if leaked, can be concerning for the user. This situation can lead to situations like identity theft causing serious issues. In order to prevent this, we have stored passwords using salted hashing function, bcrypt.

5.4.2 EVIDENCE

Flask-Bcrypt is a Flask extension that provides bcrypt hashing utilities. We are using this to secure our application's sensitive data.

- In order to use this, we first simply import the class wrapper and pass the Flask app object.

```
from flask_wtf.csrf import CSRFProtect
from flask_bcrypt import Bcrypt

app = Flask(__name__)
app.config.from_object(Config)
db = SQLAlchemy(app)
login = LoginManager(app)
bcrypt = Bcrypt(app)
csrf = CSRFProtect(app)
```

- Then we can easily use its two primary hashing methods by way of the bcrypt object.

```
def set_password(self, password):
    self.password = bcrypt.generate_password_hash(password, 10).decode('utf-8')

def check_password(self, password):
    return bcrypt.check_password_hash(self.password, password)
```

- By default, the bcrypt algorithm has a maximum password length of 72 bytes and ignores any bytes beyond that. The value 10 in generate_password_hash() sets the number of log_rounds parameter in bcrypt.gensalt() which determines the complexity of the salt.
- The following image shows how password are stored in the database

```
sqlite> select * from user;
1|sneha|sneha@gmail.com|$2b$10$MNmfrp90PY0DhSkE5aLD90TbM8gMlg2vP7W2h/1TMoFMuQQzadCvu|0|0|0
2|snehaw|sneha3@gmail.com|$2b$10$4E4qZ3147GW6oe9SNbzkp05tnknLIv8hwwj/m7cs4nZxWMZAhhKkdK|0|0|0
3|devika|devika@gmail.com|$2b$10$NJS4RAemudpcfhJ81LzsFeMMXSbATw1BAbVpclBomkHGG6VDi6Cfm|0|0|0
4|devikaw|devikaw@gmail.com|$2b$10$0LHU/fmidniZdV2umPPyWuCXFph.YvnsznotGrwaMiv1cyrrJqtfu|0|0|0
5|testuser|testuser@gmail.com|$2b$10$KFrGBMxTU2FN8d6yZLarv.8L8czjV94fjCW9WQ8P5elKPuXZCp7MG|1|0|0
```

5.5 Sensitive data exposure – Failed Login

5.5.1 CLAIM

An application should always respond with a generic error message regardless of whether the user ID or password is incorrect. The reason being specific error messages can tell the attacker if an account exists or not. Thus, to avoid this we have implemented generic error messages for the login screen.

5.5.2 EVIDENCE

In our application we show generic error messages to the user on the login screen.

Game: [Home Login](#)

• Invalid username or password

Login

Username
testuser

Password

☐ Remember Me

New User? [Click to Register!](#)

5.6 Session Management

5.6.1 CLAIM

Session management is used to facilitate secure interactions between a user and some service or application and applies to a sequence of requests and responses associated with that user. This weakness can allow an attacker to either capture or bypass the authentication methods that are used by a web application [Velasco2020]. We have implemented measures to prevent this attack in our application.

5.6.2 EVIDENCE

Flask-Session is an extension for Flask that adds support for Server-side session to our application.

- Firstly, we set the secret key to use the session. The session object of the flask package is used to set and get session data.
- The session object works like a dictionary, but it can also keep track of modifications. When we use sessions, the data is stored in the browser as a cookie.
- The flask.session.permanent flag and the app.permanent_session_lifetime flag allow Flask to know that you want the session to expire.

```
@app.before_request
def before_request():
    session.permanent = True
    app.permanent_session_lifetime = timedelta(minutes=2)
    session.modified = True
    g.user = current_user
```

```
Request
POST /home HTTP/1.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Origin: https://127.0.0.1:5000
Cookie: session=eJw8KtwzAMRK_lap2FKFo_X6CHKAKDokgkuJMuK0ogty9Alp0xefgyHIPs-pG_SLdLB9PM-1jmH4wS_-mZN7pJl01T3-OHlv2PKET2qW-mfPr1Br3TfrFLHs7ZGZXahYDKNDKCAFWTkxWjfnpJE4eE2gGH1YBDFUqwaLYsGEUoxRjdEqmXOgbySCNoE
c0AAQ21msjXYm4dX8s8EiFZ4dQw10XsMPmXL2k3ust8_Oc7uR1P18G3Juu-NT7sMcCEXpwaFHMdEBKWIS6MuWp8fCxFDdeb1A4RdW_4X9EtIA.W1IK2PH763KedXqBZ6TaS9cm, remember_token=5je91a911801a0a0582e2d7ed7ab6d4570b4a17747449c6
f98c657dab795883bce0091722e681eb94f8ee5657616efa0a2d8b1c53fd228721dc9495a8ecb301d92
Accept-Encoding: gzip, deflate, br
Host: 127.0.0.1:5000
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_5) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/13.1.1 Safari/605.1.15
Accept-Language: en-us
Referer: https://127.0.0.1:5000/
Connection: keep-alive
```

5.7 Click Jacking

5.7.1 CLAIM

The X-Frame-Options HTTP response header field indicates a policy on whether the browser should render the transmitted resource within a <frame> or within <iframe>. Servers can declare this policy in the header of their HTTP responses to prevent clickjacking attacks and ensuring that their content is not embedded into other pages or frames. [Ross2013]. We have implemented measures in the following way to prevent this attack in our application

5.7.2 EVIDENCE

Flask Prevents external sites from embedding your site in an iframe. It sets a response header that prevents a class of attacks where clicks in the outer frame can be translated invisibly to clicks on your page's elements.

- If the value of X-Frame-Options is DENY, then the receiving content with this header will not display this content in any frame.

```
@app.after_request
def apply_caching(response):
    response.headers["X-Frame-Options"] = "DENY"
    response.headers['X-XSS-Protection'] = '1; mode=block'
    return response
```

Response

```
HTTP/1.1 200 OK
Set-Cookie: session=.eJw9jsFKBTEMrf-laxdN0ybt-5khSRMUcZTOvJX47xYEd5fL5dzznY5Yfr2mx72e_pKOt5keCdBHA1JwRQvrhrnU0YPFKGaHCC6QA6Bp
Fp7B2TzlXTozlx361DplWog75g6VECBzHiYdq2o2KhVs4zJfLVaDlqQuTUKS1vky9eHnH7e_2rPy9efX9sDu1Yc9-e7n7tQBBpawkg9IHcU0lml7zvObTiqiMEs
6ecXHkFGFA.X9EttA.eqrVMDmGmOD4Og1KO1WaJUaYUcl; Expires=Wed, 09-Dec-2020 20:06:04 GMT; HttpOnly; Path=/
Content-Type: text/html; charset=utf-8
Date: Wed, 09 Dec 2020 20:04:04 GMT
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block
Content-Length: 3075
Vary: Cookie
Server: Werkzeug/1.0.1 Python/3.6.4
```

5.8 Cross Site Scripting

5.8.1 CLAIM

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites [Smirnov2018]. Following are the measures to prevent this attack in our application

5.8.2 EVIDENCE

The browser will try to prevent reflected XSS attacks by not loading the page if the request contains something that looks like JavaScript and the response contains the same data. Flask configures Jinja2 to automatically escape all values unless explicitly told otherwise. This should rule out all XSS problems caused in templates.

Above two figures show the settings for X-XSS-Protection.

The header X-XSS-Protection: 1; mode=block enables the XSS Filter. Thus, when a browser detects this it will prevent rendering of the pages.

5.9 Input Validation

5.9.1 CLAIM

‘Input validation is the process of testing input received by the application for compliance against a standard defined within the application.’ [Clarke2009] Input Validation significantly contribute towards security against XSS, SQL Injection and other attacks, reducing their impact if implemented properly. One should always implement server-side input validation as the client-side validation cannot be trusted. An attacker can bypass client-side input validation by manipulating the client-side code. Always use Whitelists, not Blacklists, for validations. White list validation involves defining exactly what is authorized, and by definition, everything else is not authorized. [InputOWASP] Our application is secured against such vulnerability.

5.9.2 EVIDENCE

We have implemented server-side input validation using regex for patterns in the Whitelists which secures the application against this vulnerability.

Referring to the following screenshots, it displays the input validation error messages generated by the server during input validation. The inputs in our application are validated during signing up and creating a game.

Sign up input validation:

Game: [Home Login](#)

• Username must contain a length of at least 5 characters and a maximum of 8 characters. Username must contain at least one special character like - or _. Username must not start or end with these special characters and they can't be one after

Sign Up

Username
use

Email
sneha96@gmail.com

Password

Confirm Password

[Register](#)

Game: [Home Login](#)

• Password must contain a length of at least 8 characters and a maximum of 15 characters. Password must contain at least one lowercase character Password must contain at least one uppercase character Password must contain at least one digit [0-9]. Password must contain at least one special character like ! @ # & \$ %

Sign Up

Username
user[]

Email
sneha96@gmail.com

Password

Confirm Password

[Register](#)

Code including regex patterns for sign up input validation:

```
@app.route('/signup', methods=('GET', 'POST'))
def signup():
    if current_user.is_authenticated:
        return redirect(url_for('index'))
    form = SignUpForm()
    if form.validate_on_submit():
        username_re_match = re.fullmatch("^(?!.*[_]{2,})(?=[^_]+[^\s-]{5,8}$", form.username.data)
        password_re_match = re.fullmatch("^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%&]) [A-Za-z\d@$!%&]{8,15}$", form.password.data)
        if not bool(username_re_match):
            flash("Username must contain a length of at least 5 characters and a maximum of 8 characters.")
            flash("Username must contain at least one special character like - or _.")
            flash("Username must not start or end with these special characters and they can't be one after", 'error')
            return redirect(url_for('signup'))
        if not bool(password_re_match):
            flash("Password must contain a length of at least 8 characters and a maximum of 15 characters.")
            flash("Password must contain at least one lowercase character")
            flash("Password must contain at least one uppercase character")
```

Create game input validation:

Game: [Home](#) [Logout](#)

- GameName must contain a length of at least 5 characters and a maximum of 10 characters. Password must contain at least 8 characters and a maximum of 16 characters.

Hi, testuser!

Your Stats:

Wins	Losses	Ties
1	0	0

Global Stats:

User	Wins	Losses	Draws
sneha	0	0	0
snehaw	0	0	0
devika	0	0	0
devikaw	0	0	0
testuser	1	0	0

Join a pre-existing Game:

Create a Game:

Game Name

Show Completed Games:

Code including regex pattern for create game input validation:

```
@app.route("/create_game", methods=['POST'])
def create_game():
    gameName = request.form.get('gamename')
    gameName_re_match = re.match("^[A-Za-z0-9]{5,10}$", gameName)
    if not bool(gameName_re_match):
        flash("GameName must contain a length of at least 5 characters and a maximum of 10 characters."
              "Password must contain at least 8 characters and a maximum of 16 characters.", 'error')
        return redirect(url_for('index'))
    user = User.query.filter_by(username=current_user.username).first or 404()
```

5.10 Bandit

Bandit is a tool designed to find common security issues in Python code. To find the issues, bandit processes each file, builds an AST from it, and runs appropriate plugins against the AST nodes. Once Bandit has finished scanning all the files it generates a report. [Pypi2020]

- We first installed bandit on our system using the command
pip install bandit
- Then we executed it using
bandit -r path/to/your/code
- The bandit command output

```
snehawattamwar@Snehas-MBP Flask-Login % bandit -r App4/
[main] INFO profile include tests: None
[main] INFO profile exclude tests: None
[main] INFO cli include tests: None
[main] INFO cli exclude tests: None
[main] INFO running on Python 3.6.4
Run started:2020-12-09 12:46:48.238901

Test results:
>> Issue: [B201:flask_debug_true] A Flask app appears to be run with debug=True, which exposes the Werkzeug debug
ger and allows the execution of arbitrary code.
Severity: High Confidence: Medium
Location: App4/app/__init__.py:21
More info: https://bandit.readthedocs.io/en/latest/plugins/b201_flask_debug_true.html
20 if __name__ == "__main__":
21     app.run(ssl_context=('cert.pem', 'key.pem'), debug=True)

-----

Code scanned:
Total lines of code: 462
Total lines skipped (#nosec): 0

Run metrics:
Total issues (by severity):
Undefined: 0.0
Low: 0.0
Medium: 0.0
High: 1.0
Total issues (by confidence):
Undefined: 0.0
Low: 0.0
Medium: 1.0
High: 0.0
Files skipped (0):
```

In reference to the output, we found an issue with High severity and Medium confidence. The issue description states: Running Flask applications in debug mode results in the Werkzeug debugger being enabled. This includes a feature that allows arbitrary code execution.

Upon visiting the debugger documentation page and learned something new, we found the following:

Danger:

The debugger allows the execution of arbitrary code which makes it a major security risk. **The debugger must never be used on production machines. We cannot stress this enough. Do not enable the debugger in production.**

‘A publicly exposed debugger will subject the machine to remote code execution. Once attackers can execute arbitrary Python code on the server, she can directly leak all the sensitive data stored on the server.’[Li2018]

Fix - we removed the debug=True flag (Default value = False) and executed the bandit command again. Thus, we were able to fix the errors detected by the bandit.

```
snehawattamwar@Snehas-MBP Flask-Login % bandit -r App4/
[main] INFO profile include tests: None
[main] INFO profile exclude tests: None
[main] INFO cli include tests: None
[main] INFO cli exclude tests: None
[main] INFO running on Python 3.6.4
Run started:2020-12-09 20:34:44.011619

Test results:
No issues identified.

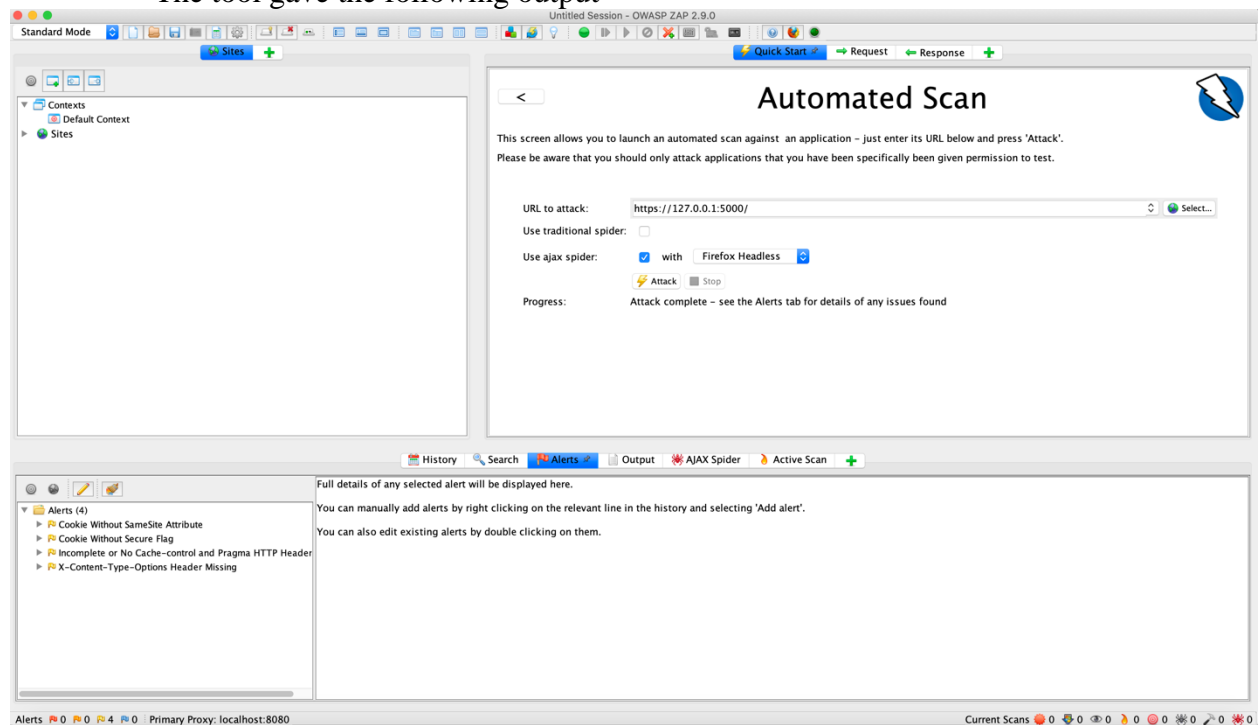
Code scanned:
Total lines of code: 462
Total lines skipped (#nosec): 0

Run metrics:
Total issues (by severity):
Undefined: 0.0
Low: 0.0
Medium: 0.0
High: 0.0
Total issues (by confidence):
Undefined: 0.0
Low: 0.0
Medium: 0.0
High: 0.0
Files skipped (0):
```

5.11 OWASP ZAP

The OWASP Zed Attack Proxy (ZAP) is one of the world's most popular web application security testing tools. It is made available for free as an open-source project and is contributed to and maintained by OWASP. This tool can be used during web application development by web developers or by experienced security experts during penetration tests to assess web applications for vulnerabilities. [Obbayi2018]

- We downloaded and installed the tool on our system.
- Then we launched the tool in Standard Mode.
- We entered the URL of our application
- The tool gave the following output



- The output of the tool shows that it found 4 vulnerabilities in our application

1. Cookie Without Same Site Attribute

2. Cookie Without Secure Flag

In order to fix these two errors, we used the Flask configuration options to set these parameters on the Session Cookie.

```
SESSION_COOKIE_SECURE=True,  
SESSION_COOKIE_HTTPONLY=True,  
SESSION_COOKIE_SAMESITE='Strict'
```

- The SECURE flag limits cookies to HTTPS traffic only
- HTTPOnly protects the contents of cookies from being read with JavaScript
- SameSite with 'Strict' prevents sending cookies with all external requests.

3. Incomplete or No Cache-Control and Pragma HTTP headers

In order to fix this, we need to ensure that the cache-control HTTP header is set with no-cache, no-store, must-revalidate; and that the pragma HTTP header is set with no-cache.

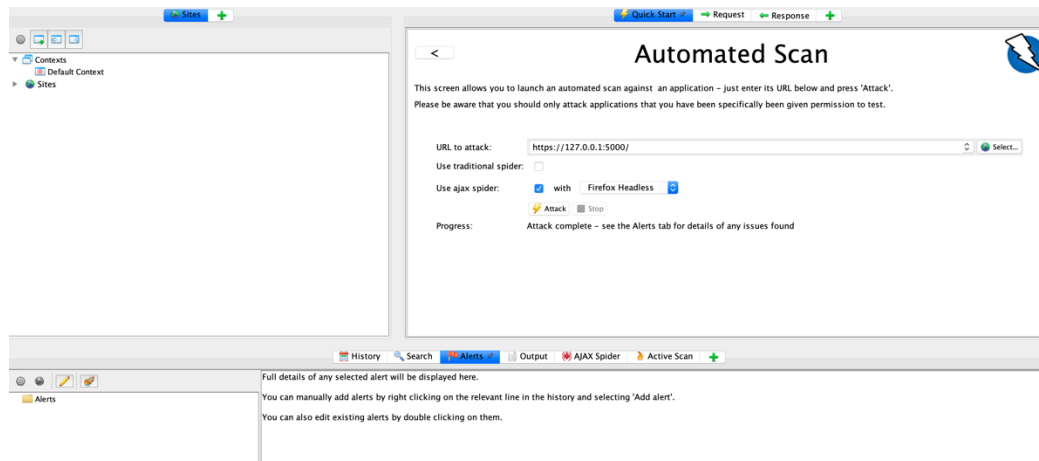
The no-cache directive means that a browser may cache a response, but it must first submit a validation request to an origin server. The no-store directive means that a browser may not cache a response, but it must first submit a validation request to an origin server. The must-revalidate directive ensures that a resource is revalidated each time before using it.

4. X-Content-Type-Options

In order to fix this, we need to ensure that the response header contains X-Content-Type-Options. So, we added this header to our response data with a value as 'nosniff'. This forces the browser to honor the response content type instead of trying to detect it, which can be abused to generate a cross-site scripting (XSS) attack.

```
@app.after_request  
def apply_caching(response):  
    response.headers["X-Frame-Options"] = "DENY"  
    response.headers['X-XSS-Protection'] = '1; mode=block'  
    response.headers['X-Content-Type-Options'] = 'nosniff'  
    response.headers["Cache-Control"] = "no-cache, no-store, must-revalidate"  
    response.headers["Pragma"] = "no-cache"  
    return response
```

- Finally, after fixing these errors we launched another 'Attack' action on our application. This time it gave 0 alerts. Thus, we successfully fixed all the errors and issues, making our application completely secure by verifying it against automated tests as well.



6. CONCLUSION

We have successfully implemented a secure game of GO that can be played between multiple players. The above list of assurance cases is amongst the Top 10 vulnerabilities. We have presented evidence for every assurance case along with screenshots. The application was also tested against automatic attacks by tools such as Bandit and Owasp for weaknesses. The detected weaknesses were analyzed, fixed, and verified again by re-running the tool.

7. REFERENCES

- [CLOUDFLARE2020] CloudFlare inc. What is TLS (Transport Layer Security)? (2020)
<https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/>
- [INTERNETSOC] Internet Society TLS Basics <https://www.internetsociety.org/deploy360/tls/basics/>
- [GRINBERG2017] Miguel Grinberg, Running Your Flask Application Over HTTPS, June 3 2017
<https://blog.miguelgrinberg.com/post/running-your-flask-application-over-https>
- [SQLOWASP2020] OWASP, SQL Injection, kingthorin 2020, https://owasp.org/www-community/attacks/SQL_Injection
- [Bogoev2020] Damyan Bogoev, Flask Series: Security, Dec 13, 2020
<https://damyanon.net/post/flask-series-security/>

- [CSRFOWASP] CheatSheetSeries, Cross-Site Request Forgery Prevention Cheat Sheet, 2020 https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html
- [Velasco2020] Roberto Velasco, Hdiv Security, Broken Authentication and Session Management OWASP Top 10 2013 - A2, 10/11/2020 <https://hdivsecurity.com/owasp-broken-authentication-and-session-management>
- [Ross2013] D. Ross, T. Gondrom, HTTP Header Field X-Frame-Options draft-ietf-websec-x-frame-options-02, 25th February 2013 <https://tools.ietf.org/html/draft-ietf-websec-x-frame-options-02>
- [Smirnov2018] Alexey Smirnov, Securing Flask web applications, July 11, 2018, <https://smirnov-am.github.io/securing-flask-web-applications/>
- [Pypi2020] Python Software Foundation, Bandit 1.6.3, 2020, <https://pypi.org/project/bandit/>
- [Obbayi2018] Lester Obbayi, Introduction to OWASP ZAP for Web Application Security Assessments, March 30, 2018 <https://resources.infosecinstitute.com/topic/introduction-owasp-zap-web-application-security-assessments/>
- [Clarke2009] Justin Clarke, 2009, Code-Level Defenses, <https://www.sciencedirect.com/topics/computer-science/input-validation#:~:text=Input%20validation%20is%20the%20process,business%20logic%20to%20validate%20input.>
- [InputOWASP] CheatSheetSeries, Input Validation Cheat Sheet, 2020, https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html
- [Li2018] Vickie Li, Hacking Flask Applications, Feb 2018 <https://medium.com/swlh/hacking-flask-applications-939eae4bffd>