Team Members:
Tyler Dionne (tdionne2021@my.fit.edu), Kendall Kelly (kelly2021@my.fit.edu)

Project Advisor:
Sneha Sudhakaran, ssudhakaran@fit.edu

Project Title:
FIT Automated Transfer Credit Evaluation

Client:
Sneha Sudhakaran

Website:
https://tylerdionne.github.io/ATCE-FIT/index.html

Milestone 6 Progress Evaluation

1. Progress of Current Milestone:

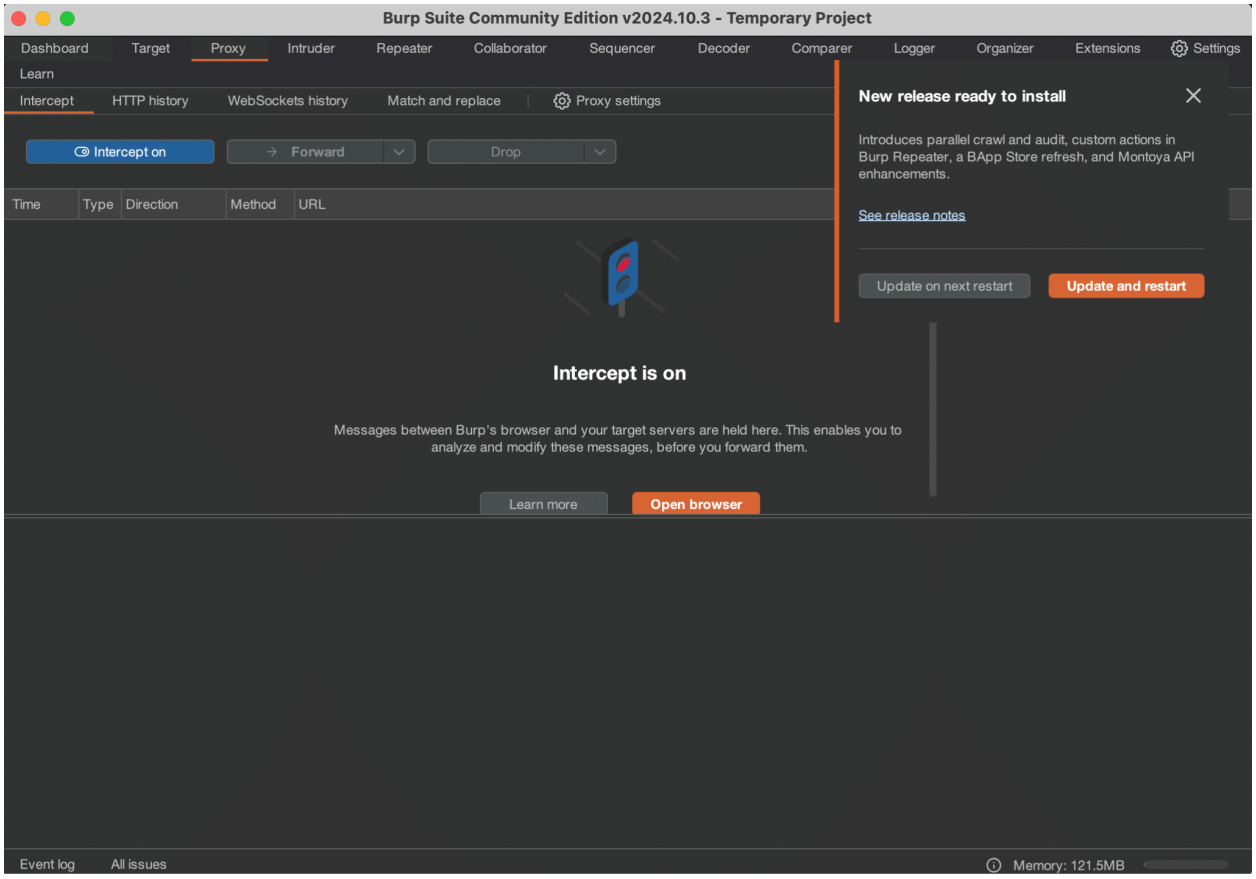
Task	Completion %	Tyler	Kendall	To Do
Create test cases for the fuzzing of the /login page	100%	100%	100%	N/A
Install BurpSuite	100%	100%	0%	N/A
Learn BurpSuite (how to open the site in the burpsuite chromium browser use the intercept tool, use the repeater tool to edit and send requests)	100%	100%	100%	N/A
Use burp suite to fuzz the /login page with all of the test cases and analyze the output searching for bugs. Testing both the username input field and the password input field with the test cases.	100%	100%	100%	N/A
Document findings professionally the same as	100%	100%	100%	N/A

a professional fuzzing environment				
Install ZAP	100%	100%	0%	N/A
Learn how to use ZAP (load in target address and then run the Spider auto analyzer tool to generate a report)	100%	0%	100%	N/A
Analyze the ZAP report taking into account the findings under the “Alerts” tab analyzing each entry and the severity	100%	100%	100%	N/A
Overview of general web application security improvements that can be made in the main Flask python file to create an overall secure application that resists common exploits	100%	100%	100%	N/A

2. Discussion of Each Completed Task:

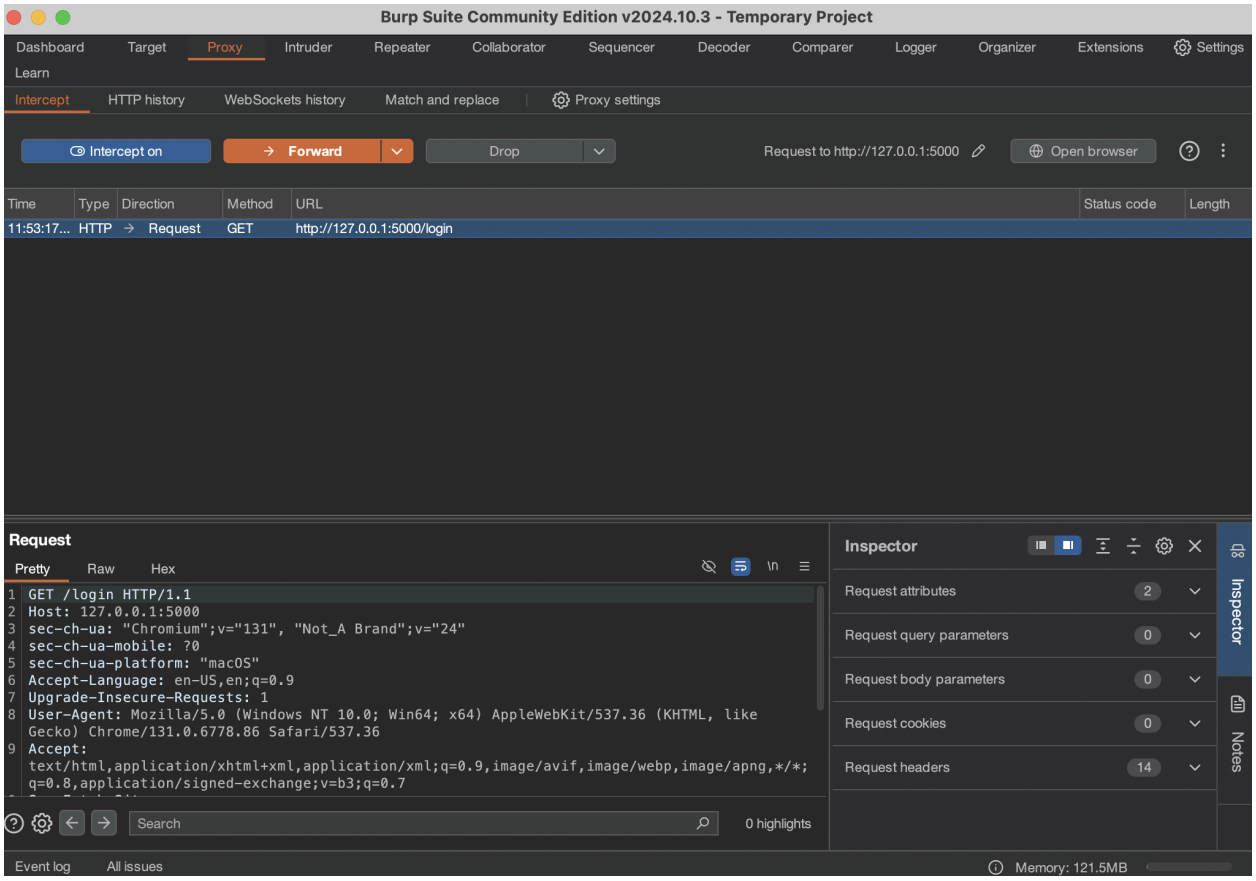
Local Pentesting and Vulnerability Testing
1.) Test Login and Registration with BurpSuite
Launch Flask app
$ python3 app.py

Open Burp Suite → Proxy → Options → ensure intercept is on.



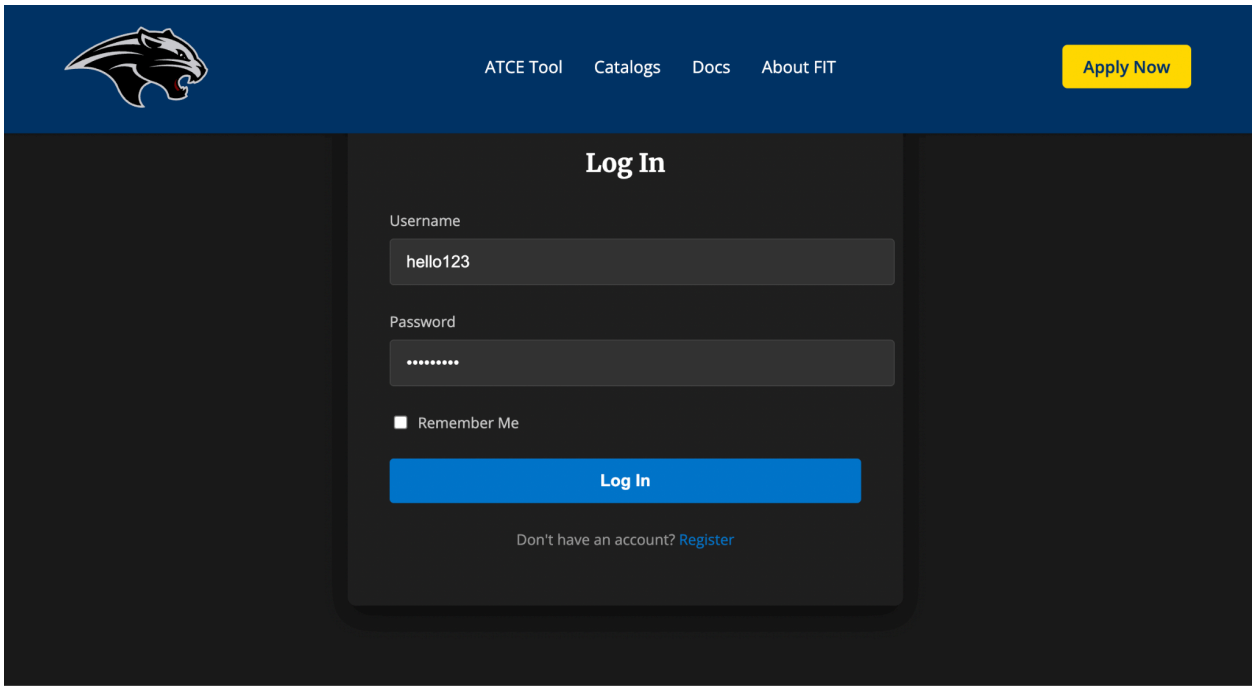
In the burpsuite chromium browser go to:

http://127.0.0.1:5000/login

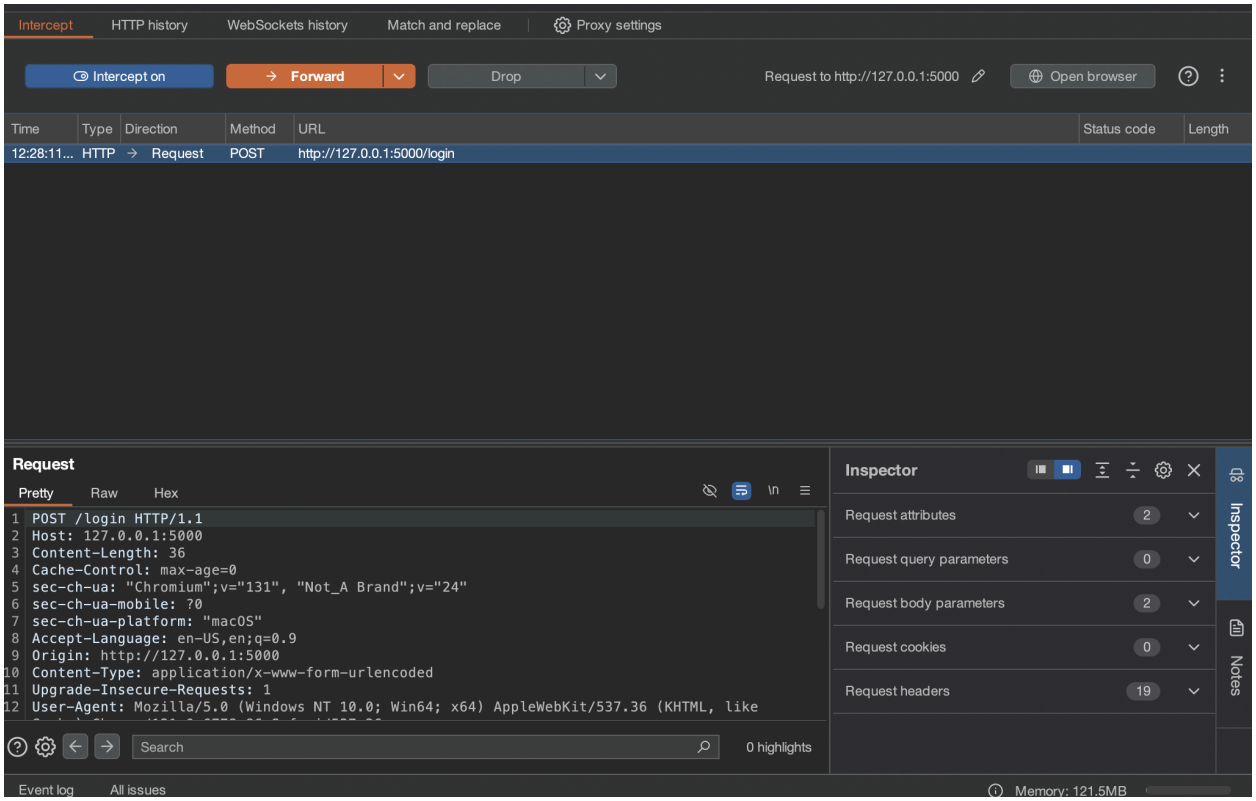


Once loaded in, will have to hit forward to send through the GET request.

Enter dummy login credentials and click "Log in."

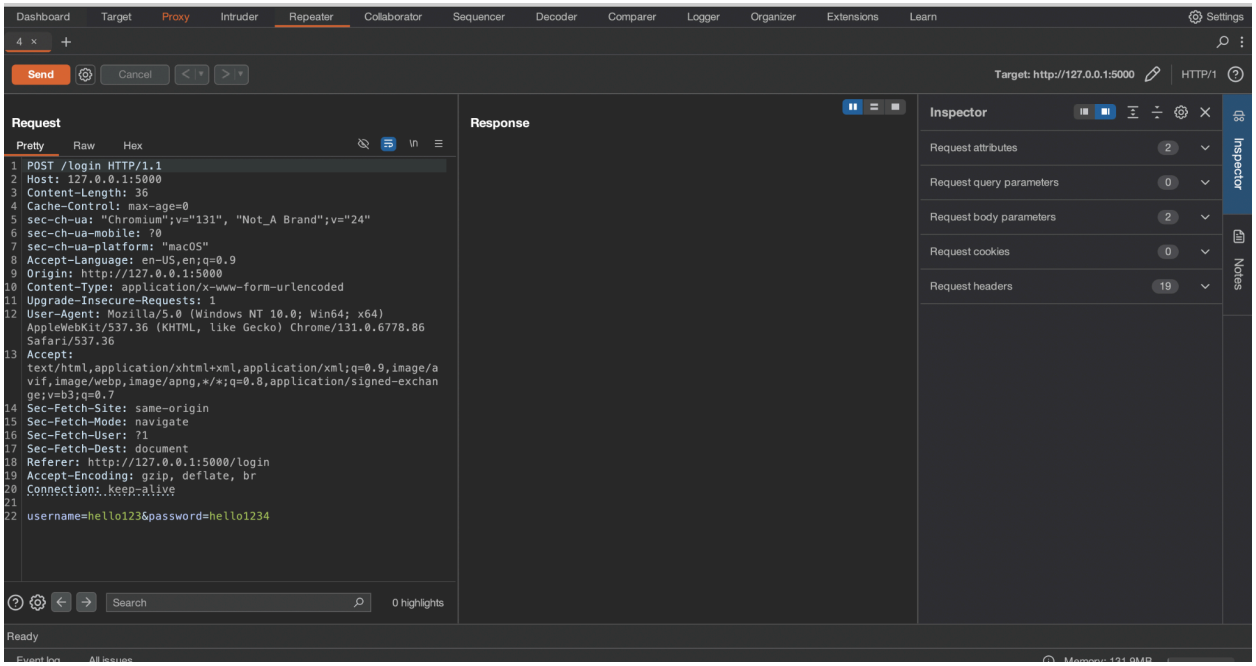


In Burp, we see the POST request intercepted.



Forward the request and go to HTTP history → Right Click → Send to Repeater

Then in the Repeater try to change the username, password field, remove the CSRF token and see how the app responds to incorrect data.



In order to properly fuzz login inputs the test cases should demonstrate testing of 1.) Input validation 2.) Error handling 3.) Security misconfigurations 4.) Edge cases

Test Cases for Fuzzing

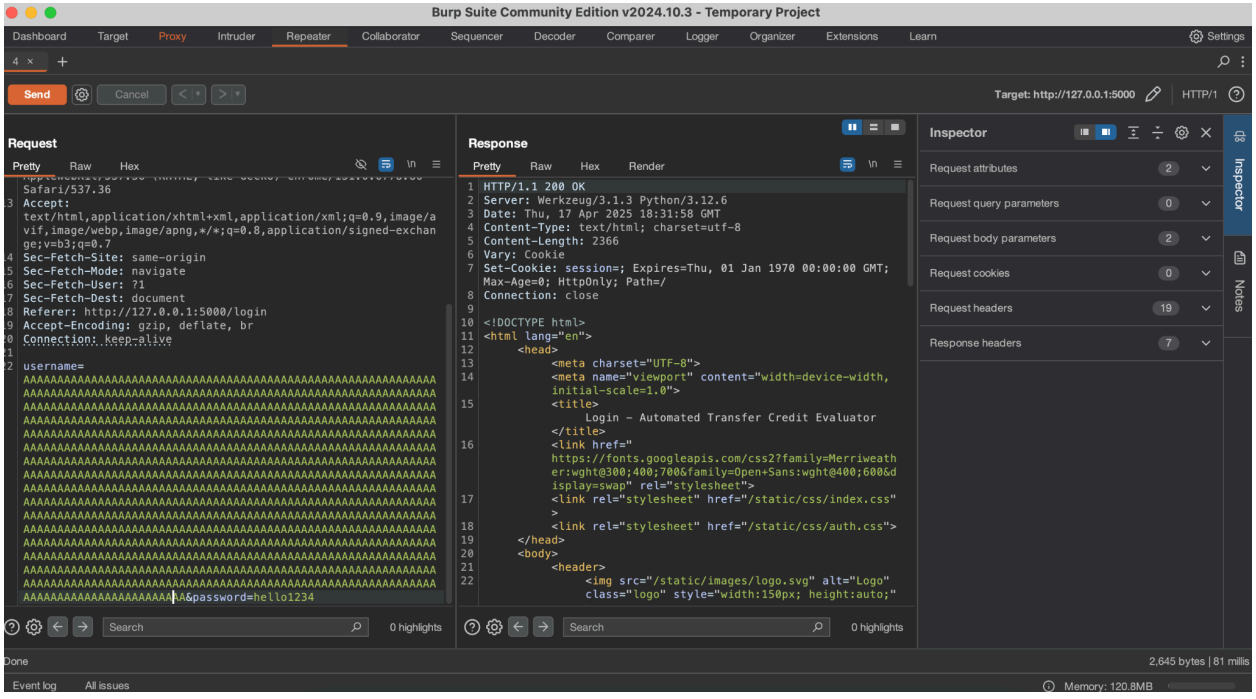
Purpose	**Input**
Normal test	admin, user1
Long input	a . . . a (1000+ chars)
Special characters	$$$$$$, <>!@#$%^&*()
SQL Injection	' OR 1=1--, admin'--
NoSQL Injection	{"$ne": null}
XSS Injection	<script>alert(1)</script>
Unicode	ユーザー
Whitespace	\tadmin\t
Path traversal	../../etc/passwd
Null byte	admin%00
Quotes/brackets	`"'{}[]``
Empty	“”
Command injection	&& /bin/sh\0;

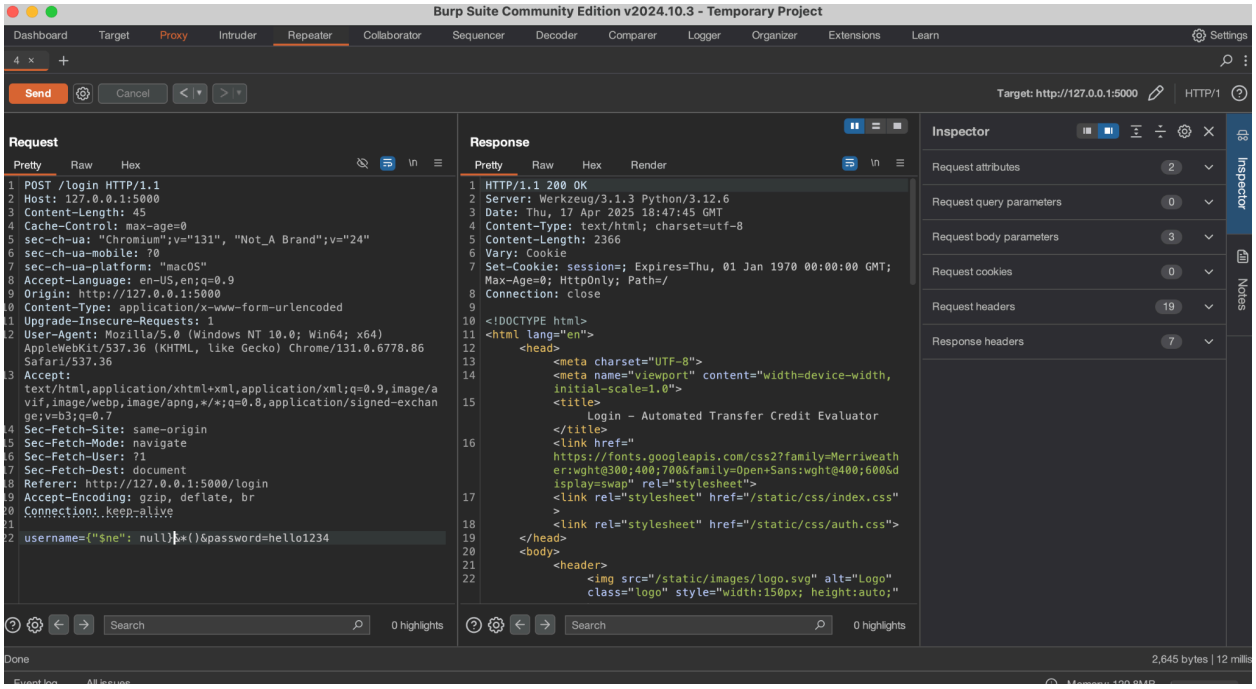
The goal of this is to see how the app deals with incorrect/strange data.

Burp Suite /login Fuzzing Results		
Input #	Input	Result/Output
01 (User + Pass)	admin	Normal (Login unsuccessful. Please check username and password.)
02 (User + Pass)	aaa … aaa (1000 a’s)	Normal (Login unsuccessful. Please check username and password.)
03 (User + Pass)	$$$$$$	Normal (Login unsuccessful. Please check username and password.)
04 (User + Pass)	<>!@#$%^&*()	Normal (Login unsuccessful. Please check username and password.)
05 (User + Pass)	' OR 1=1--, admin'--	Normal (Login unsuccessful. Please check username and password.)

06 (User + Pass)	{"$ne": null}	Normal (Login unsuccessful. Please check username and password.)
07 (User + Pass)	<script>alert(1)</script>	Normal (Login unsuccessful. Please check username and password.)
08 (User + Pass)	ユーザー	Normal (Login unsuccessful. Please check username and password.)
09 (User + Pass)	\tadmin\t	Normal (Login unsuccessful. Please check username and password.)
10 (User + Pass)	../../etc/passwd	Normal (Login unsuccessful. Please check username and password.)
11 (User + Pass)	admin%00	Normal (Login unsuccessful. Please check username and password.)
12 (User + Pass)	“”	“Please fill out this field.” displayed properly
13 (User + Pass)	&& /bin/sh\0;	Normal (Login unsuccessful. Please check username and password.)

Some images from testing





The final results from the burpsuite driven fuzzing of the /login page show the application behaves normally when given incorrect, strange, and purposefully malicious data.

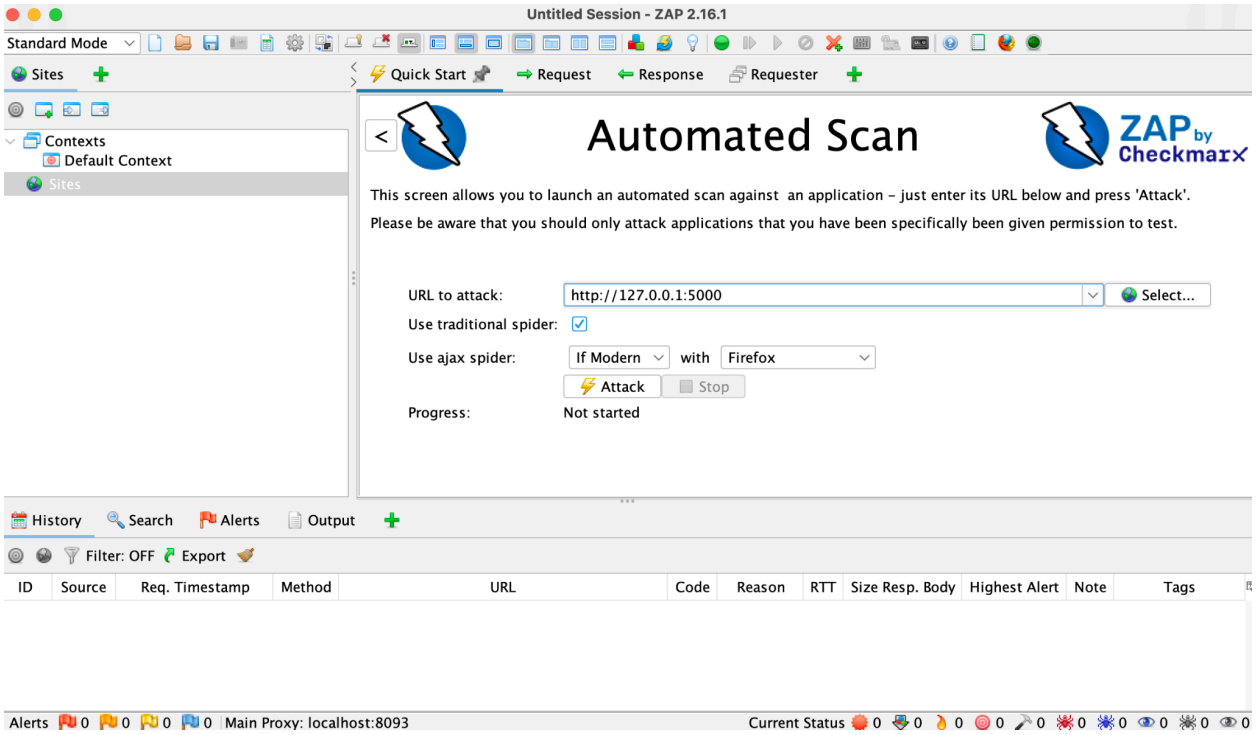
2.) Test for Common Web Vulnerabilities with OWASP ZAP

Run ZAP

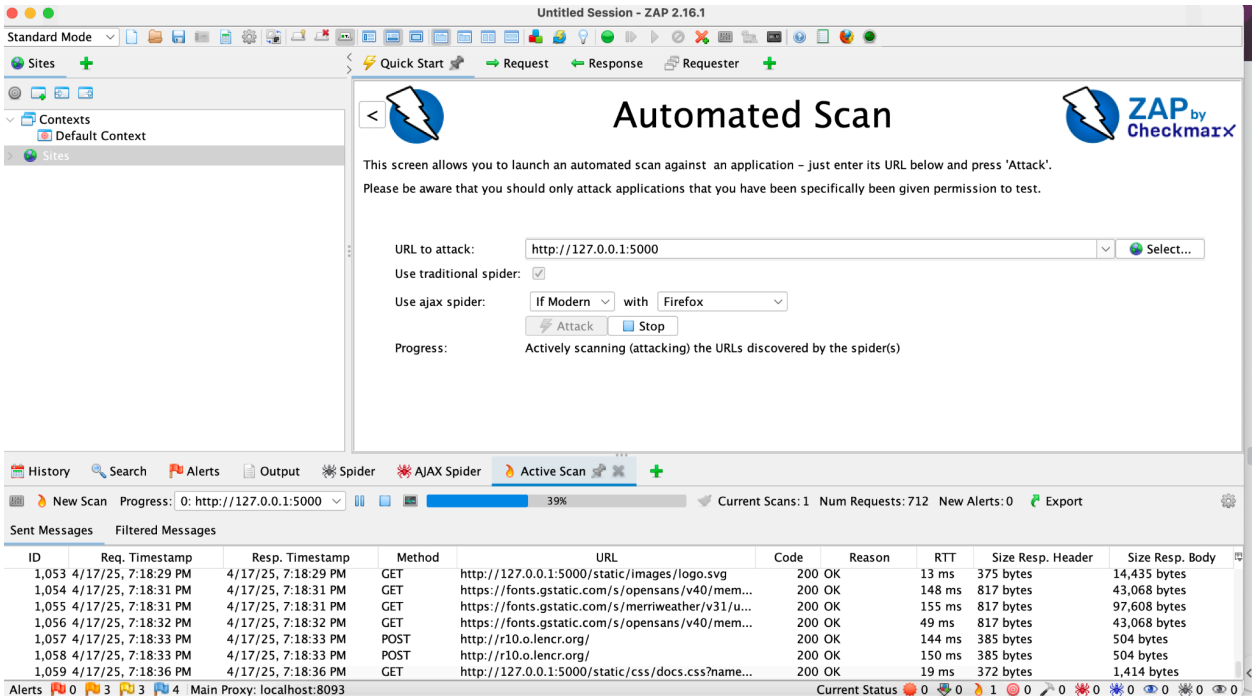
Start a new session

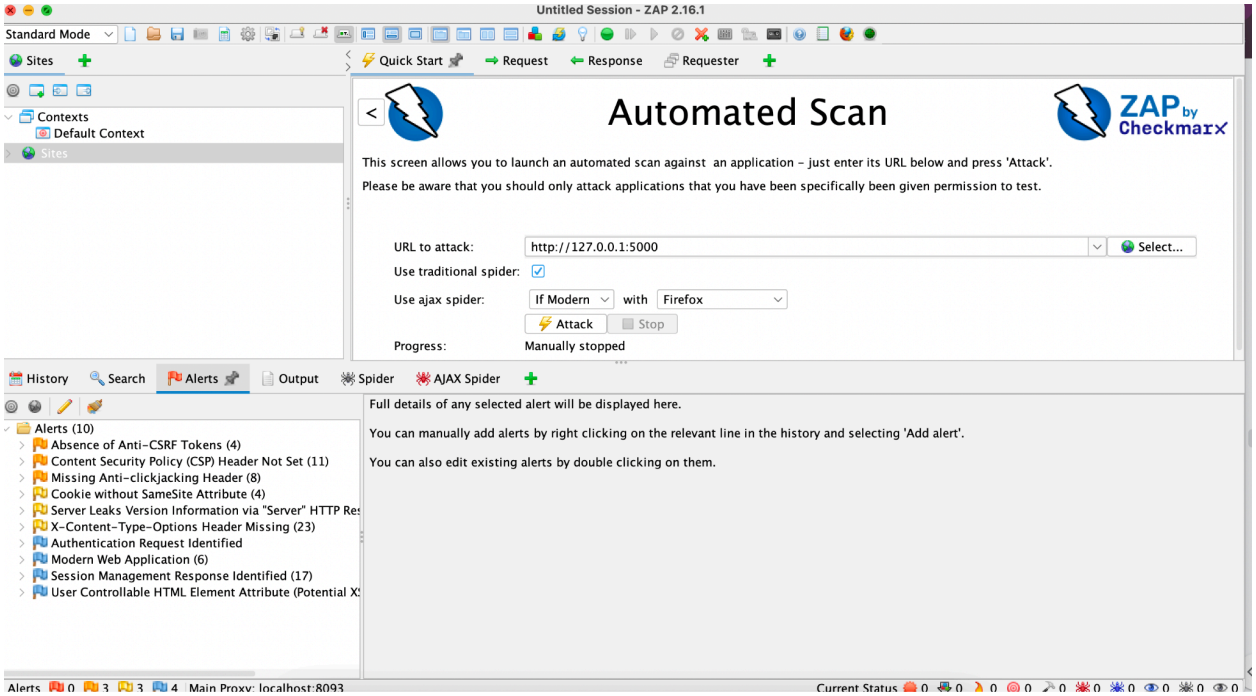
Set Flask web app as the target

http://127.0.0.1:5000



During scanning





Once the scan completes we see a summary of the issues found by ZAP.

We see several areas of our web application that have issues. The color of the flag signifies the severity of the issue if present. From this report we see that the “Absence of Anti-CSRF Tokens” is placed at the top of the list signifying that this is the most severe issue found.

CSRF stands for Cross-Site Request Forgery and basically is a type of attack where a malicious website tricks a user's browser into making an unwanted request to another site where that user is authenticated. An example is you are logged into your bank at bank.com and while you're logged in you go to a malicious website that has a hidden form that essentially sends a post request to bank.com and since you are logged in your browser includes the session cookie in the request that is in the malicious form.

3.) General web application security improvements

Input validation

```
def sanitize_input(user_input):
    # take out special characters and limit input length
    return ''.join(char for char in user_input if char.isalnum() or char.isspace())[:50]
```

Input validation is important in any web application that accepts user input. This is due to the fact that many security risks appear as soon as an application allows a user to provide input. This shows the reason why user input needs to be treated carefully and sanitized/validated to be sure the input will not cause any unexpected behavior in the application. The key idea is that a function strips out dangerous characters like ;, ', <, >, and & that can be used to inject

malicious code or commands. This security improvement protects against injection attacks such as SQL Injection, Command Injection, and Cross-site Scripting (XSS).

CSRF Protection

```
from flask_wtf.csrf import CSRFProtect
…
csrf = CSRFProtect(app)
app.config['SECRET_KEY'] = 'your-secret-key'
```

CSRF protection makes sure that only requests with a valid CSRF token which are included in legit form submissions are accepted by the server. This works by Flask-WTF inserting hidden tokens into forms and the server checks that token upon submission.

Rate Limiting

```
from flask_limiter import Limiter
from flask_limiter.util import get_remote_address

limiter = Limiter(
    app,
    key_func=get_remote_address,
    default_limits=["200 per day", "50 per hour"]
)
```

This protects against brute force attacks and denial of service by limiting how many times a client can send a request. It stops attackers from spamming endpoints like login forms or resource intensive APIs. It works by using the get_remote_address() function to track the user's IP address and then the limiter enforces a rule like no more than 50 requests per hour.

Security Header

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

These headers help prevent attacks like content sniffing, reflected xss in old browsers and clickjacking by using these headers to tell the browser to behave more securely. These headers ultimately help reduce the risk of common client side exploits.

X-Content-Type-Options: nosniff tells browsers not to guess the content type and stops it from executing a script disguised as a file.

X-Frame-Options: Deny blocks the site from being embedded in a iframe which stops clickjacking which is when users get tricked into clicking buttons.X-XSS-Protection: 1; mode=block enables XSS protection in older browsers and if XSS gets detected the page wont load.

3. Team Member Contribution of Milestone 6:
Tyler Dionne - Create test cases for the fuzzing of the /login page, Install BurpSuite, Overview of general web application security improvements that can be made in the main Flask python file to create an overall secure application that resists common exploits, Analyze the ZAP report taking into account the findings under the “Alerts” tab analyzing each entry and the severity, Install ZAP, Document findings professionally the same as a professional fuzzing environment

Kendall Kelly - Create test cases to fuzz the /login page. Become more familiar with BurpSuite and how to use its features. Use BurpSuite to fuzz the /login page with all of the test cases and find any bugs. Test both username and password input fields using the test cases, then document findings. Become more familiar with ZAP and how to use it. After getting the ZAP report, look at the “Alerts” tab and analyze each entry. Create an overview of the security improvements that could be made to make a more secure application.

4. Lessons Learned

From this project many lessons have been learned about web applications, the Flask web framework, the python programming language, html/css/javascript and how they work together to create websites with dynamic content/forms/tools, how to pentest/fuzz a web application with a user form such as a login page, how to create a full stack web application that allows the user to make an account and log in, how to create dynamically displayed content in html pages, how to use jinja2 templating to turn a website from a static website hosted on github into a standalone full stack web application, how to run Flask web applications locally, how to work with a database and integrate it within a main Flask file, how to use burpsuite to intercept requests with interceptor tool and send requests with the repeater tool, how to use OWASP ZAP to perform auto scans that generate reports that can be used to find vulnerabilities/weak spots in a website, how to create a website with a protected tool that can only be accessed by members with an account, how to use javascript inside of html to import libraries and read data from a pdf file.

Overall the lesson learned is that web application development and web application security are complex subjects with endless material, tools and methods used to create full stack web applications. Web application development is not a straightforward task and it is not something that has a clear path from start to finish. With different applications come different use cases and needs whether it be security or the way the UI is designed and therefore web application development is never straightforward.

5. Date(s) of meeting(s) with Client during the current milestone:

- Once a week every two weeks

6. Client feedback on the current milestone:
    - See Faculty Advisor Feedback below

7. Date(s) of meeting(s) with Faculty Advisor during the current milestone:
    - Once a week every two weeks

8. Faculty Advisor feedback on each task for the current Milestone:
Faculty Advisor Signature: _____ Date: _____

Evaluation by Faculty Advisor

Faculty Advisor: detach and return this page to Dr. Chan (HC 209) or email the scores to pkc@cs.fit.edu

Score (0-10) for each member: circle a score (or circle two adjacent scores for .25 or write down a real number between 0 and 10)

Tyler Dionne	0	1	2	3	4	5	5.5	6	6.5	7	7.5	8	8.5	9	9.5	10
Kendall Kelly	0	1	2	3	4	5	5.5	6	6.5	7	7.5	8	8.5	9	9.5	10

Faculty Advisor Signature: _____ Date: _____