# Creating API System Under Test

OWASP PurpleTeam

## Personal Details:

**Name:** Shane Gomes

**University:** Vellore Institute of Technology, Chennai.

**Email:** shane2907gomes@gmail.com

**Phone:** +91 - 7999564964

**Country of Residence:** INDIA

**Timezone:** Asia/Kolkata **(**UTC +05:30)

**Primary Language:** English

**About Me:**
I am currently a third year (junior) student pursuing B. Tech in Computer Science and Engineering at Vellore Institute of Technology, Chennai. I will be finishing my semester in early May, giving me plenty of time to prepare and work on my GSoC project.
If selected I would be able to give 40 hours a week to this project, barring exam season for the next semester. Although if required I am ready to burn the midnight oil so as to produce results that even exceed expectations. Cyber Security was; is; and forever will be the need of the hour, and I am all for it.

## Synopsis

Creating a vulnerable API *SUT* will help PurpleTeam test and confirm that their IaC is producing correct and consistent results. So what are we testing? Ideally, the API created would have multiple types of authentication techniques and then we can use PurpleTeam to perform regression testing on it. The outcome of this project is to make testing SUT faster and easier. This project is important to make sure that PurpleTeam is working as expected

## Technical Knowledge

I am a Computer Science student, so I am properly equipped with the required knowledge for completing this project, and more importantly I am a quick learner. I am well versed with the usage of Linux, while also grasping on the best practices while using git, and contributing to open source. I have already set-up everything that is required for this project, proving my experience of working with microservices in docker containers. I also have a reasonable amount of knowledge on how Terraform works and how it helps keep code *DRY*.

I am also a proficient coder, starting out with C & C++ (about 5 years ago). Moving onto Python (a 3-star hacker-rank coder), as well as Java, PHP, and SQL in my university assignments. I have a few side-projects under my belt, and built a website (this was a mock-up) for an architectural company, so I have experience using HTML, CSS, and emphasis on JavaScript.
In the National Cyber Olympiad I was placed at the top of my school and top 200 (out of thousands) in my region.

Github : https://github.com/shaneg07
Hacker-rank : https://www.hackerrank.com/shane_5
Mock-website design : https://shaneg07.github.io/mock-spacon-website/

## Project Details :

We will be using Zaproxy to test the vulnerabilities present in the created API.
The following are the general steps when configuring the application authentication with ZAP:

**Step 1. Define a context -** Include the target application inside the context. The unwanted URLs such as the logout page, password change functionality should be added to the exclude in context section.

**Step 2. Set the authentication mechanism**

**Step 3. Define your auth parameters -** Provide the settings on how to communicate to the authentication service of your application. The settings would include the login URL and payload format.

**Step 4. Set relevant logged in/out indicators -** ZAP needs to identify whether the application is authenticated or not.

**Step 5. Add a valid user and password -** Add a user account with valid credentials in ZAP. Multiple users can be created if the application exposes different functionality based on user roles.

The code for these steps (modules) can be found on the zaproxy website.

The authentication techniques that would ideally be implemented are as follows :

## Form Based Authentication -

```
def set_form_based_auth():
    login_url = 'http://localhost:8090/bodgeit/login.jsp'
    login_request_data =
'username={%username%}&password={%password%}'
    form_based_config = 'loginUrl=' + urllib.parse.quote(login_url) +
'&loginRequestData=' + urllib.parse.quote(login_request_data)
    zap.authentication.set_authentication_method(context_id,
'formBasedAuthentication', form_based_config)
    print('Configured form based authentication')
```

## Script Based Authentication -

The following are some of the scripting languages supported by ZAP.

JavaScript
Python
Ruby
Groovy
Zest
Snippet:

```
def set_script_based_auth():
    post_data = "username={%username%}&password={%password%}" +
"&Login=Login&user_token={%user_token%}"
    post_data_encoded = urllib.parse.quote(post_data)
    login_request_data =
"scriptName=auth-dvwa.js&Login_URL=http://localhost:3000/login.php&CS
RF_Field=user_token" \
                         "&POST_Data=" + post_data_encoded

    zap.authentication.set_authentication_method(context_id,
'scriptBasedAuthentication', login_request_data)
    print('Configured script based authentication')
```

//To upload the script, code as below:

```
def upload_script():
    script_name = 'auth-dvwa.js'
    script_type = 'authentication'
    script_engine = 'Oracle Nashorn'
    file_name = '/tmp/auth-dvwa.js'
    charset = 'UTF-8'
    zap.script.load(script_name, script_type, script_engine,
file_name, charset=charset)
```

## JSON Based Authentication

```
def set_json_based_auth():
    login_url = "http://localhost:3000/rest/user/login"
    login_request_data = 'email={%username%}&password={%password%}'

    json_based_config = 'loginUrl=' + urllib.parse.quote(login_url) +
'&loginRequestData=' + urllib.parse.quote(login_request_data)
    zap.authentication.set_authentication_method(context_id,
'jsonBasedAuthentication', json_based_config)
    print('Configured form based authentication')
```

```
//code to add the script
```

```
def add_script():
    script_name = 'jwtScript.js'
    script_type = 'httpsender'
    script_engine = 'Oracle Nashorn'
    file_name = '/tmp/jwtScript.js'
    zap.script.load(script_name, script_type, script_engine,
file_name)
```

## Timeline -

### Week 1-2

Understanding the relevant parts of purpleteam-iac-sut and trying to understand what the final product would look like.
This will involve a lot of strategizing and drafting of gameplans, while familiarizing myself with Zaproxy, Terraform, etc.

### Week 3-4

Researching upon building a new vulnerable API. This includes understanding the working of other APIs which are vulnerable, and supporting openAPI, GraphQL, SOAP, etc.

### Week 5-6

Start working on the different authentication techniques that will be used to test the vulnerable API, thereby allowing purpleteam to perform its testing. We will start with Form-Based authentication, and proceed to Script as well as JSON based methods.

## Week 7-9

We now work on containerising APIs using Docker, and using “*terragrunt apply”* to deploy it in the IaC, as well as *“terragrunt destroy”* to destroy it. The deployed API can now be used to perform tests. The deploying and testing phase will be important, since everything has to be optimized to achieve trivial spin up and bring down.

## Week 10-12

Take feedback from the community and iterate on the designs and improvise on use cases. Ensure code quality by adding more test cases and working with more videos. Work to make documents, blogs or videos to help increase the user base for this product (Subject to developer community approval).

## Week 13

Spare week in case of some work getting delayed, in case of any emergency or otherwise.

## Conclusion :

Thus, I hope this proposal gives you a small insight into what the work-plan is and how we are going to proceed. Most of what I have understood has been covered here. But at the same time I am ever-eager to learn more, anything that I might have missed here, I will learn and incorporate and deliver nothing less than perfection.

This is just the start of my open-source contributing journey, and I hope this journey never ends, as there are few things that are as rewarding as contributing to something bigger than yourself. But in the end, it is all about giving to the community and growing together.