**Test Suite (automatepractice.com, login functionality tests)**

*Test Case 1 (Invalid Email Format)*

**Preconditions:**

The login page for automated.com is opened in the browser.

The login form is empty.

**Steps and Expected results:**

*Step 1* => Login form submitted (email with an invalid format and a valid password provided)

*Expected result 1* => Access denied, an error message pops up informing the user that he provided an email with invalid formatting.

**“Tear Down”**

Clear all the form fields

*Test Case 2 (Invalid credentials – invalid email)*

**Preconditions:**

The login page for automated.com is opened in the browser.

**Steps and Expected results:**

*Step 1* => Login form submitted (invalid email and valid password provided)

*Expected result 1* => Access denied, an error message pops up informing the user that he provided invalid credentials (not giving away any further info so it can’t be used by a potential attacker).

**“Tear Down”**

Clear all the form fields

*Test Case 3 (Invalid credentials – invalid password)*

**Preconditions:**

The login page for automated.com is opened in the browser.

**Steps and Expected results:**

*Step 1* => Login form submitted (valid email and invalid password provided)

*Expected result 1* => Access denied, an error message pops up informing the user that he provided invalid credentials (not giving away any further info so it can’t be used by a potential attacker).

**“Tear Down”**

Clear all the form fields

*Test Case 4 (Valid credentials)*

**Preconditions:**

The login page for automated.com is opened in the browser.

**Steps and Expected results:**

*Step 1* => Login form submitted (valid email and valid password provided)

*Expected result 1* => User logged in successfully and redirected to his dashboard.

*Test Case 5 (Server Side Validation and Password Hashing (Simple SQL injection attack))*

**Preconditions:**

Kali Linux OS launched on a virtual machine;

“Burp” proxy configured;

Browser configured to use a proxy (same port and host as your “Burp” proxy);

Simulating scenario – a malicious user somehow got hold of the name of your user accounts table in the DB (for demonstration purposes it will be called “accounts”)

**Steps and Expected results:**

*Step1 =>* Submit the login form with some password and email;

*Expected result =>* Form submitted successfully, POST request is caught by your Burp proxy, request in pending status.

*Step2 =>*  Substitute the original payload in Burp on the pending request – change password’s text to: *Select \* from accounts where email = ‘<an existing valid email>’ and password = ‘123456’ and 1=1 #’.*

*Expected result =>* Payload changed successfully

*Step3 =>*  Forward the request with the new payload via Burp.

*Expected result =>* Correct server side validation was triggered, incoming password was hashed and compared to the password hash in db, SQL injection attack failed, access denied.