Authentication and Authorization Testing

Aim:

To familiarize authentication and authorization using burpsuite

Description:

Authentication is the process of verifying who someone is, whereas authorization is the process of verifying what specific applications, files, and data a user has access to. The situation is like that of an airline that needs to determine which people can come on board. The first step is to confirm the identity of a passenger to make sure they are who they say they are. Once a passenger's identity has been determined, the second step is verifying any special services the passenger has access to, whether it's flying first-class or visiting the VIP lounge.

Authentication is used to verify that users really are who they represent themselves to be. Once this has been confirmed, authorization is then used to grant the user permission to access different levels of information and perform specific functions, depending on the rules established for different types of users.

While user identity has historically been validated using the combination of a username and password, today's authentication methods commonly rely upon three classes of information:

- What you know: Most commonly, this is a password. But it can also be an answer to a security question or a one-time pin that grants user access to just one session or transaction.
- What you possess: This could be a mobile device or app, a security token, or digital ID card.
- What you are: This is biometric data such as a fingerprint, retinal scan, or facial recognition.

Oftentimes, these types of information are combined using multiple layers of authentication. For example, a user may be asked to provide a username and password to complete an online purchase. Once that's confirmed, a one-time pin may be sent to the user's mobile phone as a second layer of security. Combining multiple authentication methods with consistent authentication protocols, organizations can ensure security as well as compatibility between systems.

Once a user is authenticated, authorization controls are then applied to ensure users can access the data they need and perform specific functions such as adding or deleting information—

based on the permissions granted by the organization. These permissions can be assigned at the application, operating system, or infrastructure levels. Two common authorization techniques include:

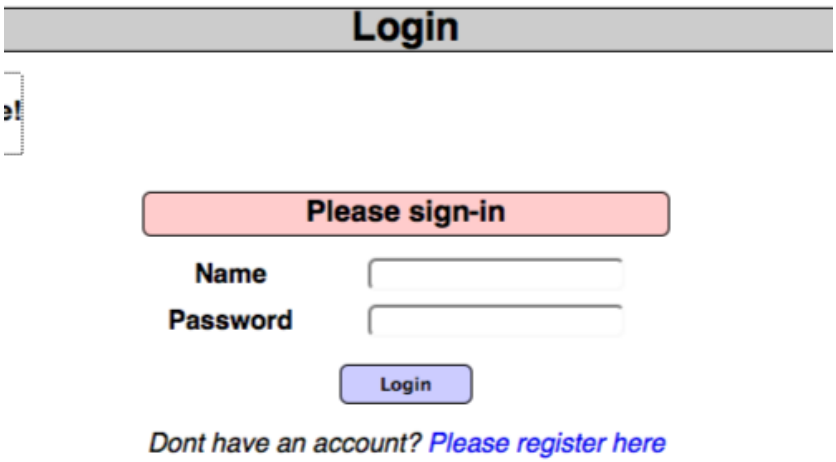
- Role-based access controls (RBAC): This authorization method gives users access to information based on their role within the organization. For example, all employees within a company may be able to view, but not modify, their personal information such as pay, vacation time, and 401K data. Yet human resources (HR) managers may be given access to all employees' HR information with the ability to add, delete, and change this data. By assigning permissions according to each person's role, organizations can ensure every user is productive, while limiting access to sensitive information.
- Attribute-based access control (ABAC): ABAC grants users permissions on a more granular level than RBAC using a series of specific attributes. This may include user attributes such as the user's name, role, organization, ID, and security clearance. It may include environmental attributes such as the time of access, location of the data, and current organizational threat levels. And it may include resource attributes such as the resource owner, file name, and level of data sensitivity. ABAC is a more complex authorization process than RBAC designed to further limit access. For example, rather than allowing all HR managers in an organization to change employees' HR data, access can be limited to certain geographical locations or hours of the day to maintain tight security limits.

Procedure:

1. Testing for Default Credentials

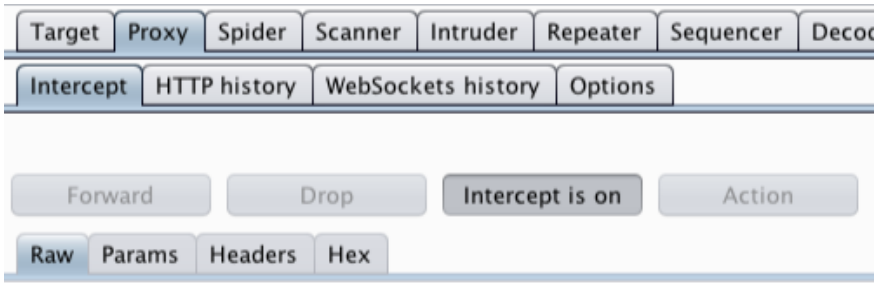
Authentication lies at the heart of an application's protection against unauthorized access. If an attacker is able to break an application's authentication function then they may be able to own the entire application.

The following tutorial demonstrates a technique to bypass authentication using a simulated login page from the "Mutillidae" training tool. The version of "Mutillidae" we are using is taken from OWASP's Broken Web Application Project. Find out how to download, install and use this project.

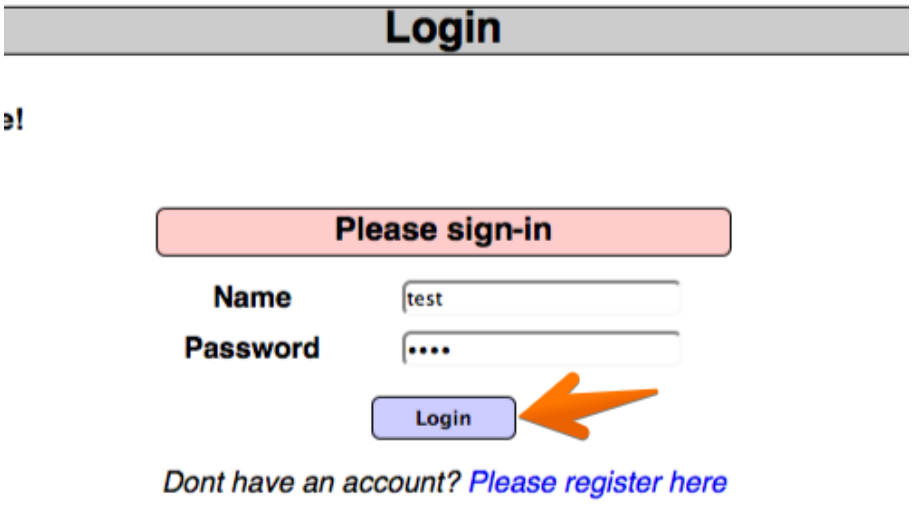


First, ensure that Burp is correctly configured with your browser.

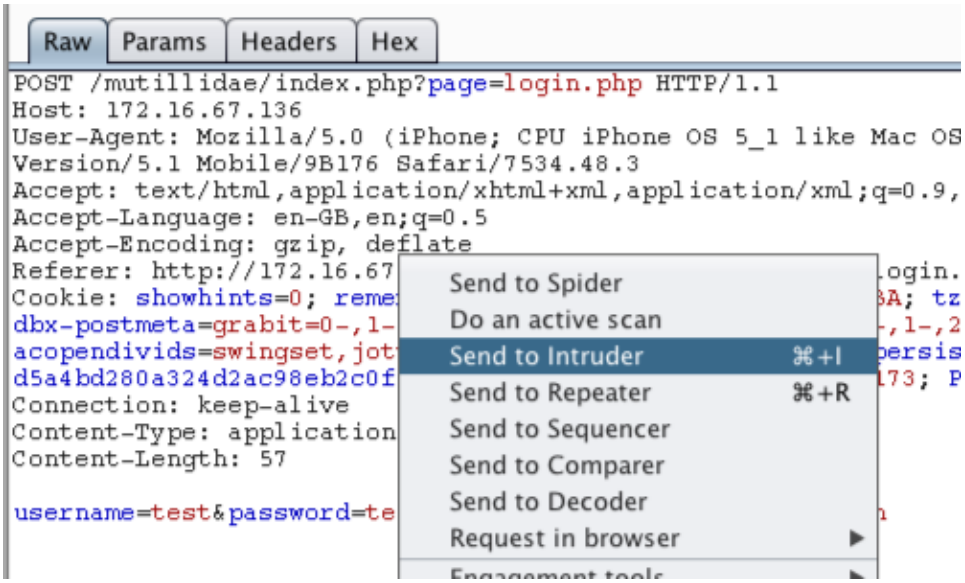
In the Burp Proxy tab, ensure "Intercept is off" and visit the login page of the application you are testing in your browser.



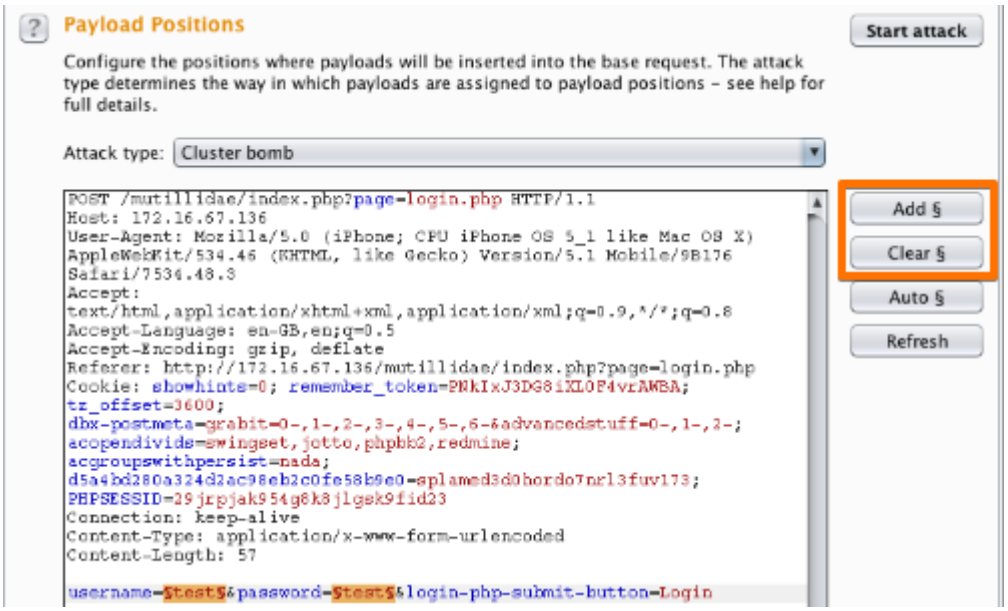
Return to Burp.In the Proxy "Intercept" tab, ensure "Intercept is on".



In your browser enter some arbitrary details in to the login page and submit the request.



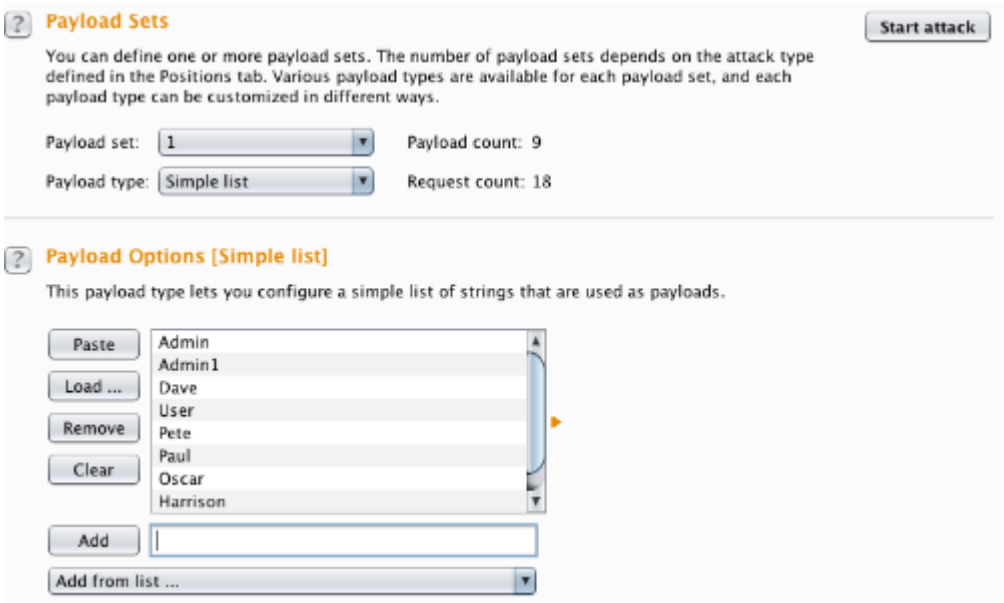
The captured request can be viewed in the Proxy "Intercept" tab.Right click on the request to bring up the context menu.Then click "Send to Intruder".



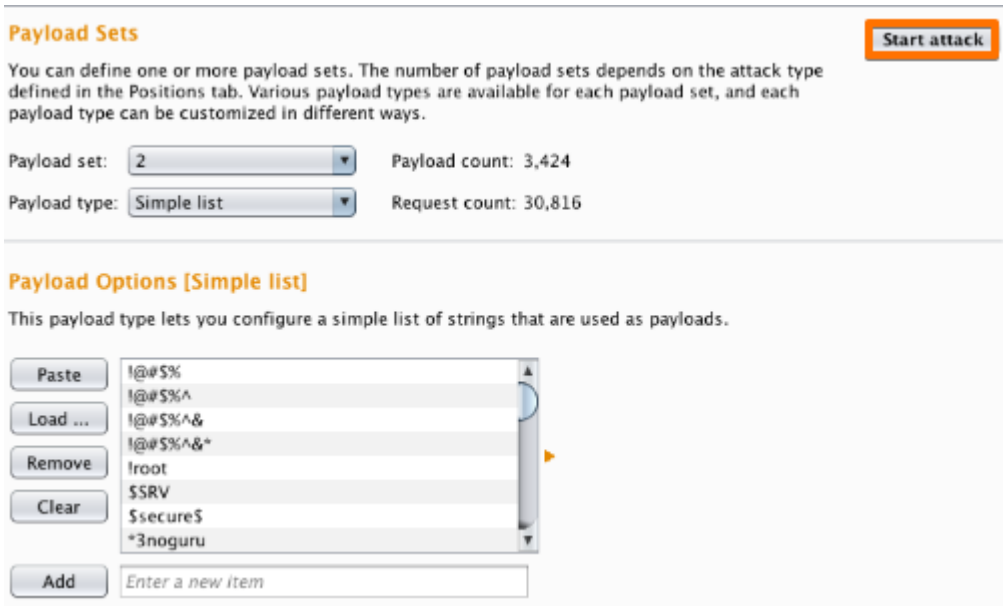
Go to the Intruder "Positions" tab.

Add the "username" and "password" parameter values as payload positions by highlighting them and using the "Add" button.

Change the attack to "Cluster bomb" using the "Attack type" drop down menu.



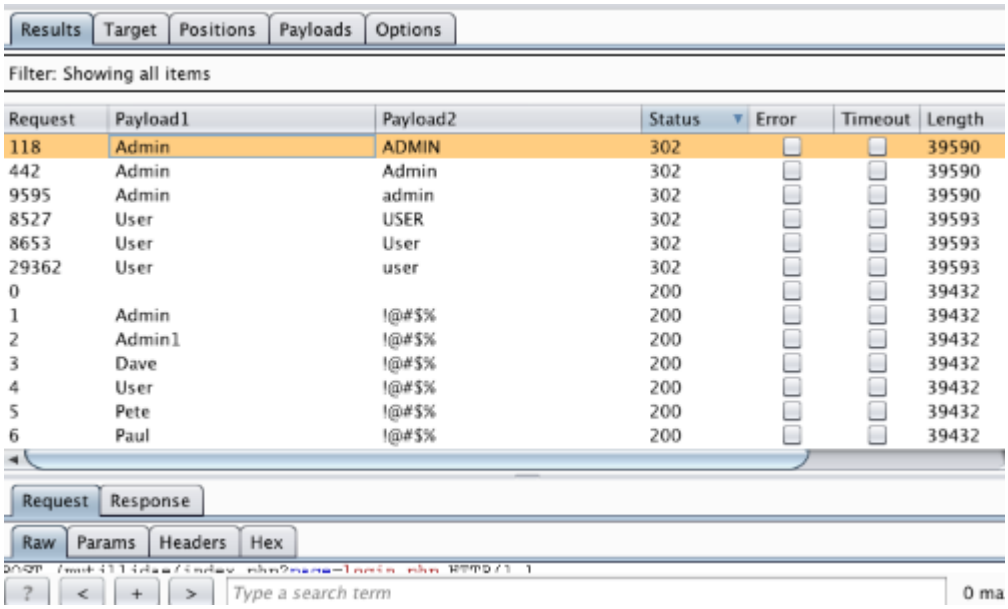
Go to the "Payloads" tab.
In the "Payload sets" settings, ensure "Payload set" is "1" and "Payload type" is set to "Simple list".
In the "Payload settings" field enter some possible usernames. You can do this manually or use a custom or pre-set payload list.



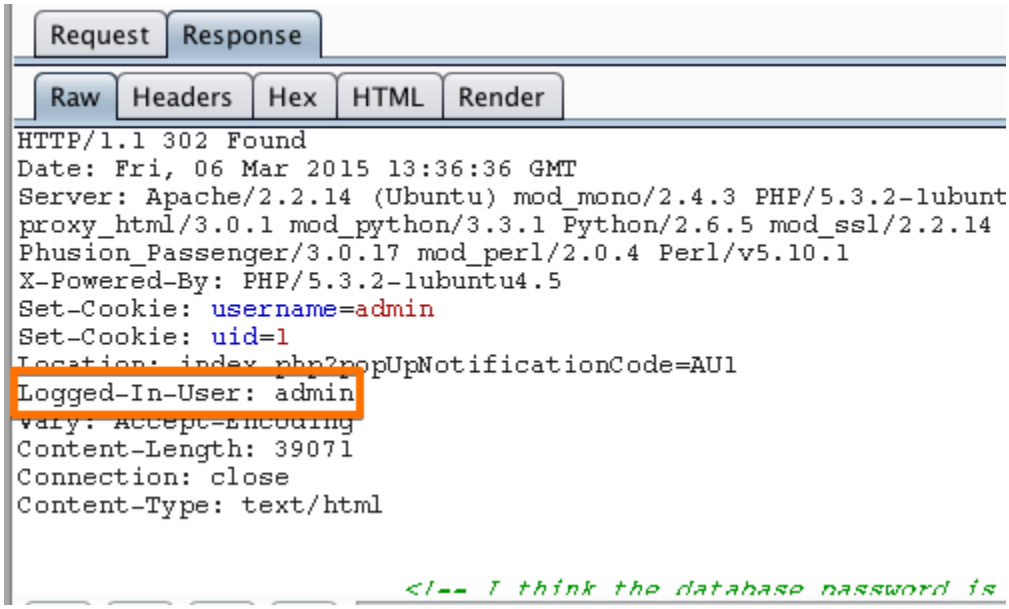
Next, in the "Payload Sets" options, change "Payload" set to "2".

In the "Payload settings" field enter some possible passwords. You can do this manually or using a custom or pre-set list.

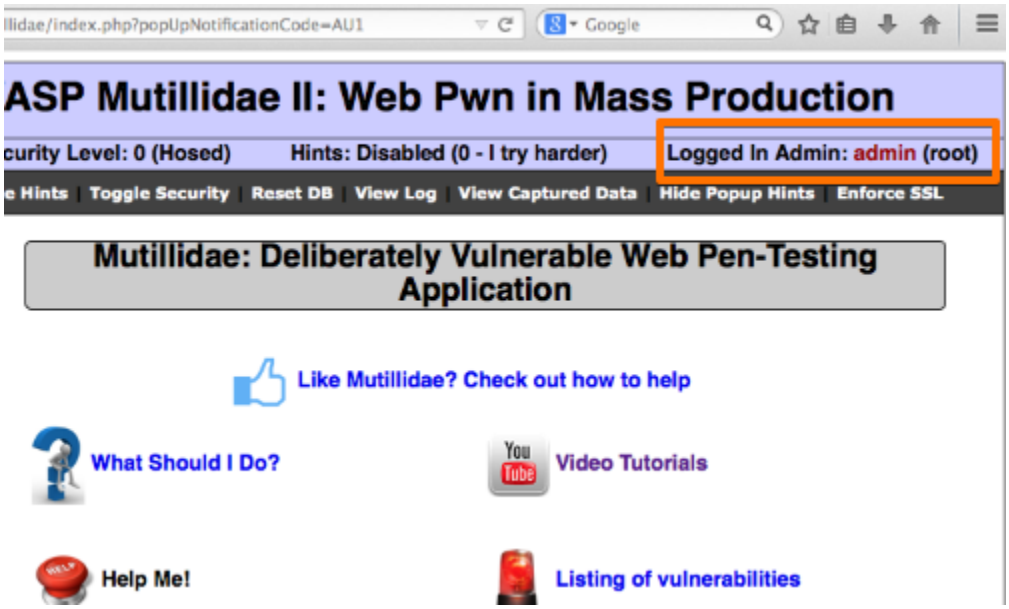
Click the "Start attack" button.



In the "Intruder attack" window you can sort the results using the column headers.In this example sort by "Length" and by "Status".

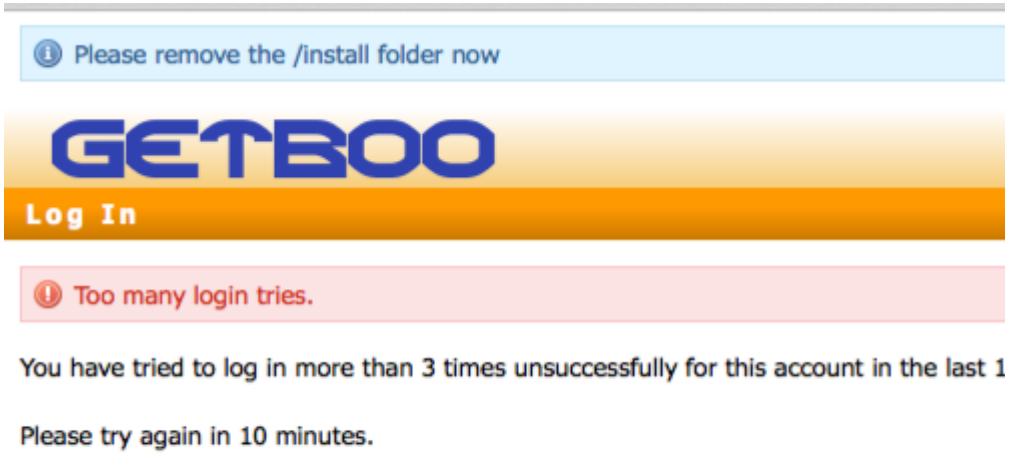


The table now provides us with some interesting results for further investigation.
By viewing the response in the attack window we can see that request 118 is logged in as "admin".



To confirm that the brute force attack has been successful, use the gathered information (username and password) on the web application's login page.

Account Lock Out



In some instances, brute forcing a login page may result in an application locking out the user account. This could be the due to a lock out policy based on a certain number of bad login attempts etc.Although designed to protect the account, such policies can often give rise to further vulnerabilities. A malicious user may be able to lock out multiple accounts, denying access to a system.In addition, a locked out account may cause variances in the behavior of the application, this behavior should be explored and potentially exploited.

2.Testing for Weak Lock out Mechanism

Account lockout mechanisms should be present within an application to mitigate brute-force login attacks. Typically, applications set a threshold between three to five attempts. Many applications lock for a period of time before a re-attempt is allowed.

Penetration testers must test all aspects of login protections, including challenge questions and response, if present.

Account lockout mechanisms are used to mitigate brute force attacks. Some of the attacks that can be defeated by using lockout mechanism:

- ✓ Login password or username guessing attack.
- ✓ Code guessing on any 2FA functionality or Security Questions.

Account lockout mechanisms require a balance between protecting accounts from unauthorized access and protecting users from being denied authorized access. Accounts are typically locked after 3 to 5 unsuccessful attempts and can only be unlocked after a predetermined period of time, via a self-service unlock mechanism, or intervention by an administrator.

Despite it being easy to conduct brute force attacks, the result of a successful attack is dangerous as the attacker will have full access on the user account and with it all the functionality and services they have access to.

3.Testing for Browser Cache Weaknesses

Browser caching is provided for improved performance and better end-user experience. However, when sensitive data is typed into a browser by the user, such data can also be cached in the browser history. This cached data is visible by examining the browser's cache or simply by pressing the browser's back button.

The goal of poisoning the cache is to make the clients load unexpected resources partially or controlled by the attacker.

The poisoned response will only be served to users who visit the affected page while the cache is poisoned. As a result, the impact can range from non-existent to massive depending on whether the page is popular or not.

4. Testing for Weak Password Policy

Testing for Weak password policy is part of Web Application penetration testing and it is covered in OWASP authentication testing section.

Please refer OWASP testing guide "**Testing for Weak password policy (OTG-AUTHN-007)**" for password policy testing. In this article, i will show you how to automate item number 4.

1. What characters are permitted and forbidden for use within a password? Is the user required to use characters from different character sets such as lower and uppercase letters, digits and special symbols?

2. How often can a user change their password? How quickly can a user change their password after a previous change? Users may bypass password history requirements by changing their password 5 times in a row so that after the last password change they have configured their initial password again.

3. When must a user change their password? After 90 days? After account lockout due to excessive log on attempts?

4. How often can a user reuse a password? Does the application maintain a history of the user's previous used 8 passwords?

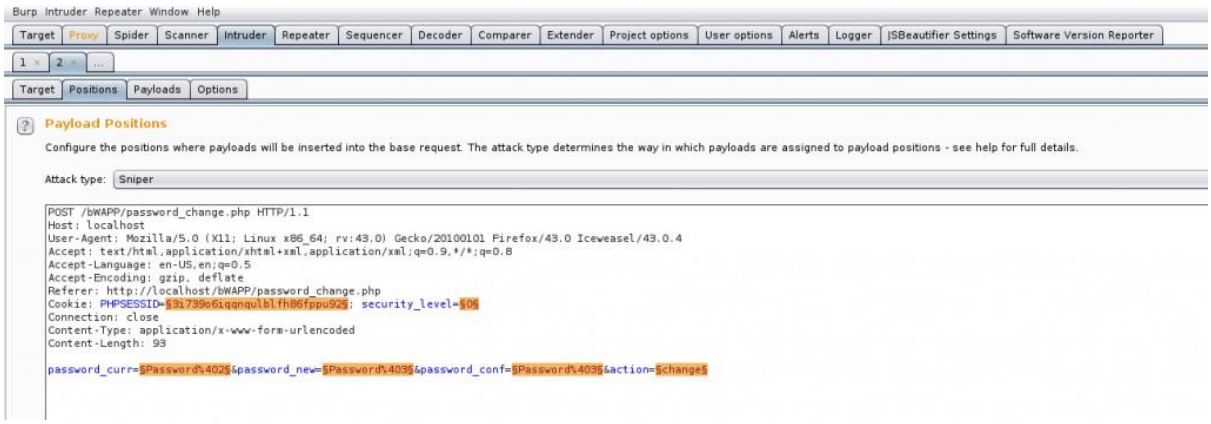
5. How different must the next password be from the last password?

6. Is the user prevented from using his username or other account information (such as first or last name) in the password?

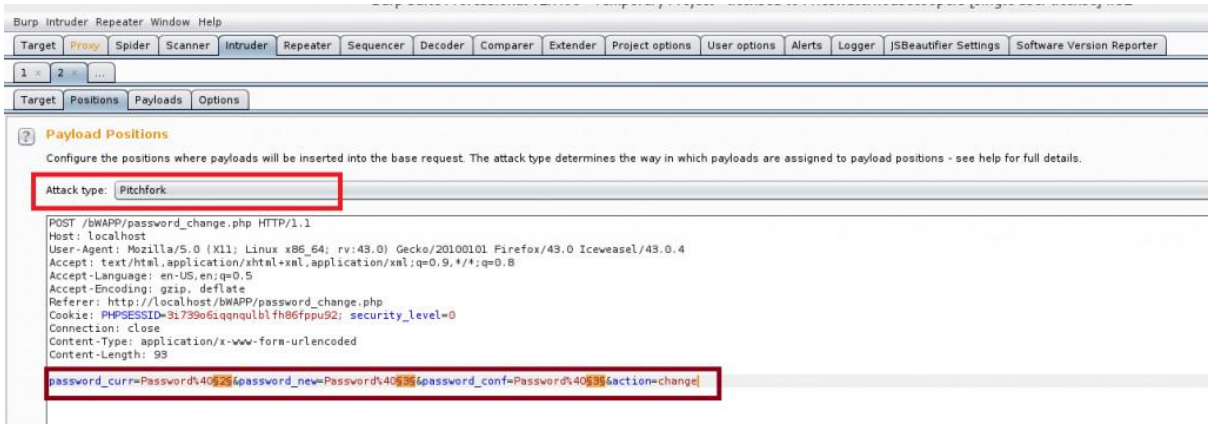
Login into application using existing credentials.

Capture the request in burp suite proxy

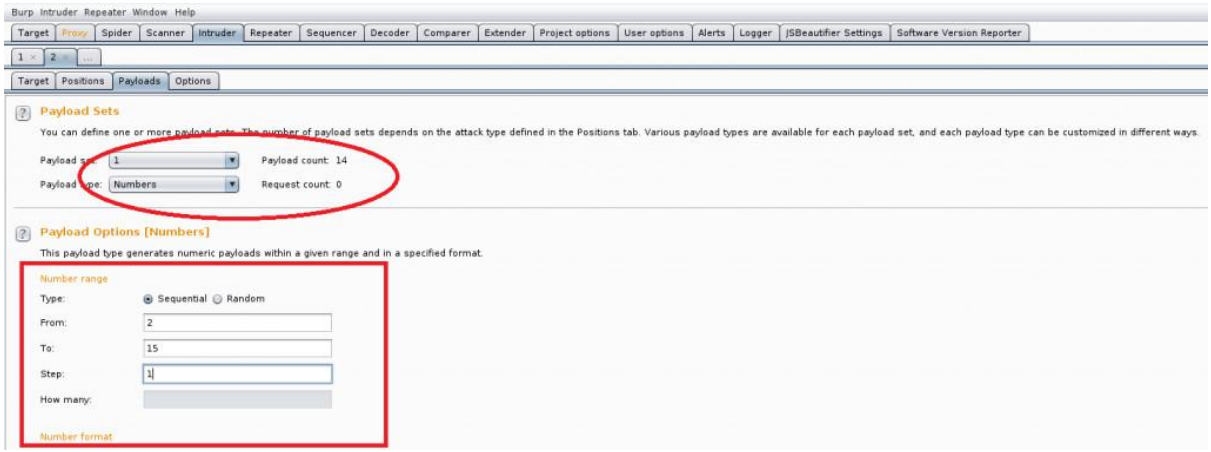
Send the request to intruder



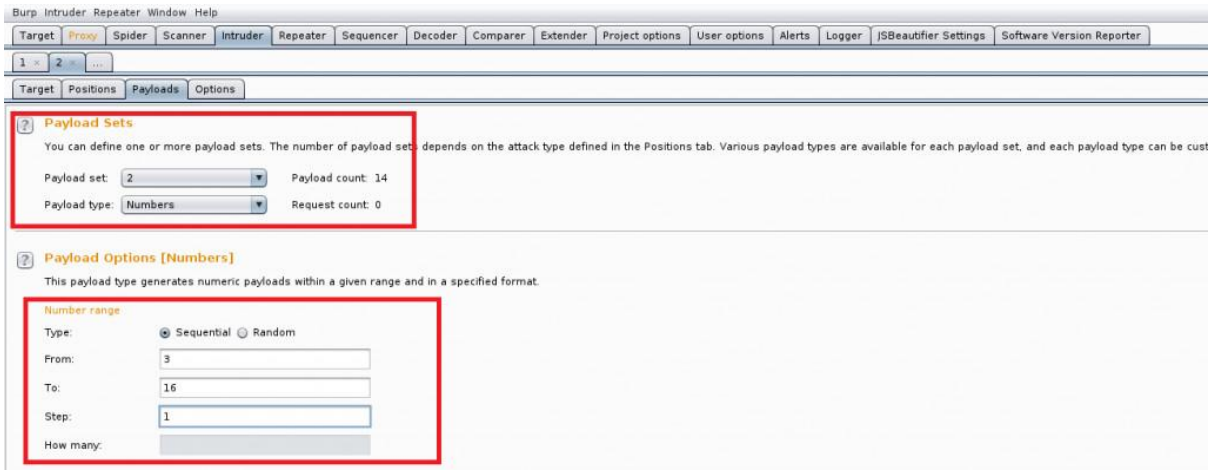
I will be using sequential number with Password@1 as my passwords, from intruder select last character of password and select attack type as pitchfork.



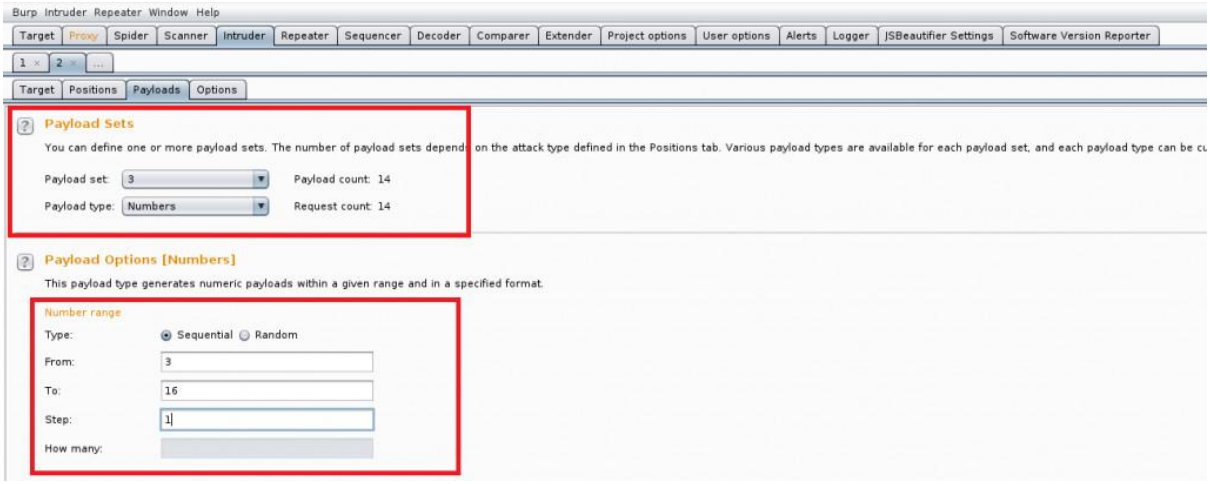
From payload tab select payload type as numbers and fill number range as shown in fig.



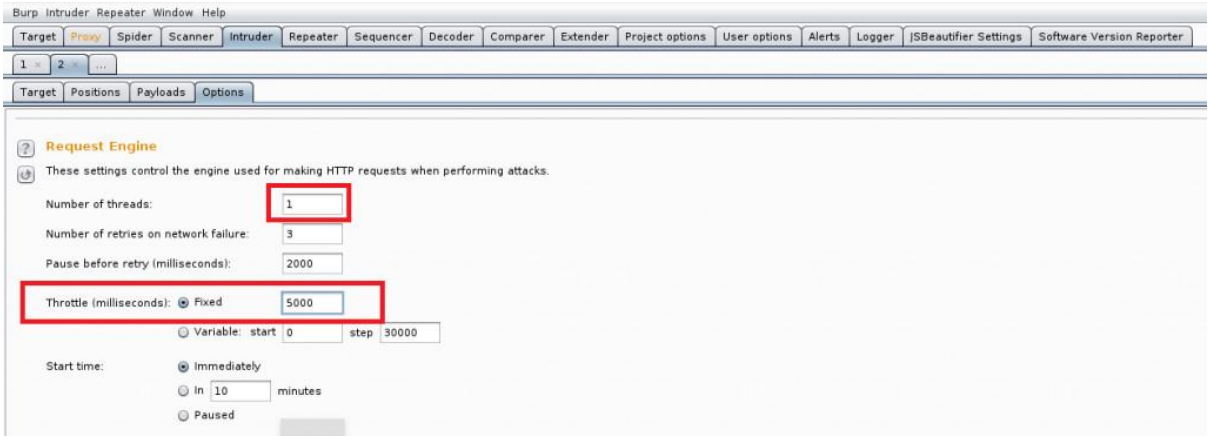
Similarly select payload for second position, note the difference from and to number range.



Select payload for third position



This part is very important, from intruder options tab select number of thread as one and throttle it for 5 sec(5000 millisec). Without this configuration there will be a chaos and password will be changed in random orders and logic will break.



Now you can see intruder result with password payload in results

Attack Save Columns

Results | Target | Positions | Payloads | Options

Filter: Showing all items

Request	Payload1	Payload2	Payload3	Status	Error	Timeout	Length	Comment
0				200			13948	
1	2	3	3	200			13950	
2	3	4	4	200			13948	
3	4	5	5	200			13948	
4	5	6	6	200			13948	
5	6	7	7	200			13948	
6	7	8	8	200			13948	
7	8	9	9	200			13948	
8	9	10	10	200			13948	
9	10	11	11	200			13948	
10	11	12	12	200			13948	
11	12	13	13	200			13948	
12	13	14	14	200			13948	
13	14	15	15	200			13948	
14	15	16	16	200			13948	

Request | Response

Raw | Params | Headers | Hex

```
POST /bWAPP/password_change.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:43.0) Gecko/20100101 Firefox/43.0 Iceweasel/43.0.4
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost/bWAPP/password_change.php
Cookie: PHPSESSID=3i739o6iqqnqulblfh86fppu92; security_level=0
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 93

password_curr=Password%402&password_new=Password%403&password_conf=Password%403&action=change
```

And password changed in response.

Attack Save Columns

Results | Target | Positions | Payloads | Options

Filter: Showing all items

Request	Payload1	Payload2	Payload3	Status	Error	Timeout	Length	C
0				200			13948	
1	2	3	3	200			13950	
2	3	4	4	200			13948	
3	4	5	5	200			13948	
4	5	6	6	200			13948	
5	6	7	7	200			13948	
6	7	8	8	200			13948	
7	8	9	9	200			13948	
8	9	10	10	200			13948	
9	10	11	11	200			13948	
10	11	12	12	200			13948	
11	12	13	13	200			13948	
12	13	14	14	200			13948	
13	14	15	15	200			13948	
14	15	16	16	200			13948	

Request | Response

Raw | Headers | Hex | HTML | Render

```
        <input type="password" id="password_curr" name="password_curr"></p>

        <p><label for="password_new">New password:</label><br />
        <input type="password" id="password_new" name="password_new"></p>

        <p><label for="password_conf">Re-type new password:</label><br />
        <input type="password" id="password_conf" name="password_conf"></p>

        <button type="submit" name="action" value="change">Change</button>

    </form>

    </br >
    <font color="green">The password has been changed!</font>
</di
```

5. Testing for Weak Password Change or Reset Functionalities

For any application that requires the user to authenticate with a password, there must be a mechanism by which the user can regain access to their account if they forget their password. Although this can sometimes be a manual process that involves contacting the owner of the website or a support team, users are frequently allowed to carry out a self-service password reset, and to regain access to their account by providing some other evidence of their identity.

As this functionality provides a direct route to compromise the user's account, it is crucial that it is implemented securely. The first step is to gather information about what mechanisms are available to allow the user to reset their password on the application. If there are multiple interfaces on the same site (such as a web interface, mobile application, and API) then these should all be reviewed, in case they provide different functionality.

Once this has been established, determine what information is required in order for a user to initiate a password reset. This can be the username or email address (both of which may be obtained from public information), but it could also be an internally-generated user ID.