Brute Forcing Passwords and Username Enumeration:

See if you can enumerate usernames by entering different username values and observing the messages being sent back from the webpage, these messages are often different if you supply a valid username vs. and invalid one.

**Flawed brute-force protection**

It is highly likely that a brute-force attack will involve many failed guesses before the attacker successfully compromises an account. Logically, brute-force protection revolves around trying to make it as tricky as possible to automate the process and slow down the rate at which an attacker can attempt logins. The two most common ways of preventing brute-force attacks are:

* Locking the account that the remote user is trying to access if they make too many failed login attempts
* Blocking the remote user's IP address if they make too many login attempts in quick succession

Both approaches offer varying degrees of protection, but neither is invulnerable, especially if implemented using flawed logic.

For example, you might sometimes find that your IP is blocked if you fail to log in too many times. In some implementations, the counter for the number of failed attempts resets if the IP owner logs in successfully. This means an attacker would simply have to log in to their own account every few attempts to prevent this limit from ever being reached.

In this case, merely including your own login credentials at regular intervals throughout the wordlist is enough to render this defense virtually useless.

\*\*We can also try switching the username if it is in the request, every few attempts to possibly bypass the brute force protection

**Account locking**

One way in which websites try to prevent brute-forcing is to lock the account if certain suspicious criteria are met, usually a set number of failed login attempts. Just as with normal login errors, responses from the server indicating that an account is locked can also help an attacker to enumerate usernames.

Locking an account offers a certain amount of protection against targeted brute-forcing of a specific account. However, this approach fails to adequately prevent brute-force attacks in which the attacker is just trying to gain access to any random account they can.

For example, the following method can be used to work around this kind of protection:

1. Establish a list of candidate usernames that are likely to be valid. This could be through username enumeration or simply based on a list of common usernames.
2. Decide on a very small shortlist of passwords that you think at least one user is likely to have. Crucially, the number of passwords you select must not exceed the number of login attempts allowed. For example, if you have worked out that limit is 3 attempts, you need to pick a maximum of 3 password guesses.
3. Using a tool such as Burp Intruder, try each of the selected passwords with each of the candidate usernames. This way, you can attempt to brute-force every account without triggering the account lock. You only need a single user to use one of the three passwords in order to compromise an account.

Account locking also fails to protect against credential stuffing attacks. This involves using a massive dictionary of username:password pairs, composed of genuine login credentials stolen in data breaches. Credential stuffing relies on the fact that many people reuse the same username and password on multiple websites and, therefore, there is a chance that some of the compromised credentials in the dictionary are also valid on the target website. Account locking does not protect against credential stuffing because each username is only being attempted once. Credential stuffing is particularly dangerous because it can sometimes result in the attacker compromising many different accounts with just a single automated attack.

\*\*So basically, some websites will block usernames if a certain user of the site is making too many requests in a short time. This can easily be bypassed using credential stuffing which is user a large list of both usernames and password and the cluster bomb burp intruder setting. This will try all possible combos of usernames and passwords. As long as were consistently changing which username is tried we can bypass this.

**User rate limiting**

Another way websites try to prevent brute-force attacks is through user rate limiting. In this case, making too many login requests within a short period of time causes your IP address to be blocked. Typically, the IP can only be unblocked in one of the following ways:

* Automatically after a certain period of time has elapsed
* Manually by an administrator
* Manually by the user after successfully completing a CAPTCHA

User rate limiting is sometimes preferred to account locking due to being less prone to username enumeration and denial of service attacks. However, it is still not completely secure. As we saw an example of in an earlier lab, there are several ways an attacker can manipulate their apparent IP in order to bypass the block.

As the limit is based on the rate of HTTP requests sent from the user's IP address, it is sometimes also possible to bypass this defense if you can work out how to guess multiple passwords with a single request.

\*\* if we see passwords being entered in JSON format we can try adding multiple values to the password field in JSON format, this could possibly lead to us being able to access the users account simply by having a large array of passwords being submitted all in 1 request. IF we see JSON on a login form we should look carefully, example shown below:

1. With Burp running, investigate the login page. Notice that the POST /login request submits the login credentials in JSON format. Send this request to Burp Repeater.
2. In Burp Repeater, replace the single string value of the password with an array of strings containing all of the candidate passwords. For example:

"username" : "carlos",

"password" : [

"123456",

"password",

"qwerty"

...

1. Send the request. This will return a 302 response.
2. Right-click on this request and select **Show response in browser**. Copy the URL and load it in the browser. The page loads and you are logged in as carlos.

**HTTP basic authentication**

Although fairly old, its relative simplicity and ease of implementation means you might sometimes see HTTP basic authentication being used. In HTTP basic authentication, the client receives an authentication token from the server, which is constructed by concatenating the username and password, and encoding it in Base64. This token is stored and managed by the browser, which automatically adds it to the Authorization header of every subsequent request as follows:

Authorization: Basic base64(username:password)

For a number of reasons, this is generally not considered a secure authentication method. Firstly, it involves repeatedly sending the user's login credentials with every request. Unless the website also implements HSTS, user credentials are open to being captured in a man-in-the-middle attack.

In addition, implementations of HTTP basic authentication often don't support brute-force protection. As the token consists exclusively of static values, this can leave it vulnerable to being brute-forced.

HTTP basic authentication is also particularly vulnerable to session-related exploits, notably [CSRF](https://portswigger.net/web-security/csrf), against which it offers no protection on its own.

In some cases, exploiting vulnerable HTTP basic authentication might only grant an attacker access to a seemingly uninteresting page. However, in addition to providing a further attack surface, the credentials exposed in this way might be reused in other, more confidential contexts.

\*\* we can also try to add a X-Forwarded-For header to the request containing the client’s IP address or just a different IP each time to overcome IP based protections. This header is the standard header for identifying the originating IP address of a client connection to a web server. \*\* if we go to burp settings -> project -> sessions -> session handling rules -> select bypass WAF so its on (or add it by pressing add, invoke burp extension) this will apply the loopback address in the x-forwarded-for header for all requests. Of course this extension can be changed to add a lot more then just that or any other IP for the value.