Vulnerabilities in password-based login

In this section, we'll look more closely at some of the most common vulnerabilities that occur in password-based login mechanisms. We'll also suggest ways that these can potentially be exploited. There are even some interactive labs so that you can try and exploit these vulnerabilities yourself.

For websites that adopt a password-based login process, users either register for an account themselves or they are assigned an account by an administrator. This account is associated with a unique username and a secret password, which the user enters in a login form to authenticate themselves.

In this scenario, the mere fact that they know the secret password is taken as sufficient proof of the user's identity. Consequently, the security of the website would be compromised if an attacker is able to either obtain or guess the login credentials of another user.

This can be achieved in a variety of ways, as we'll explore below.

Brute-force attacks

A brute-force attack is when an attacker uses a system of trial and error in an attempt to guess valid user credentials. These attacks are typically automated using wordlists of usernames and passwords. Automating this process, especially using dedicated tools, potentially enables an attacker to make vast numbers of login attempts at high speed.

Brute-forcing is not always just a case of making completely random guesses at usernames and passwords. By also using basic logic or publicly available knowledge, attackers can fine-tune brute-force attacks to make much more educated guesses. This considerably increases the efficiency of such attacks. Websites that rely on password-based login as their sole method of authenticating users can be highly vulnerable if they do not implement sufficient brute-force protection.

Brute-forcing usernames

SIGN UP LOGIN

business logins in the format `firstname.lastname@somecompany.com`. However, even if there is no obvious pattern, sometimes even high-privileged accounts are created using predictable usernames, such as `admin` or `administrator`.

During auditing, check whether the website discloses potential usernames publicly. For example, are you able to access user profiles without logging in? Even if the actual content of the profiles is hidden, the name used in the profile is sometimes the same as the login username. You should also check HTTP responses to see if any email addresses are disclosed. Occasionally, responses contain emails addresses of high-privileged users like administrators and IT support.

Brute-forcing passwords

Passwords can similarly be brute-forced, with the difficulty varying based on the strength of the password. Many websites adopt some form of password policy, which forces users to create high-entropy passwords that are, theoretically at least, harder to crack using brute-force alone. This typically involves enforcing passwords with:

- A minimum number of characters
- A mixture of lower and uppercase letters
- At least one special character

However, while high-entropy passwords are difficult for computers alone to crack, we can use a basic knowledge of human behavior to exploit the vulnerabilities that users unwittingly introduce to this system. Rather than creating a strong password with a random combination of characters, users often take a password that they can remember and try to crowbar it into fitting the password policy. For example, if `mypassword` is not allowed, users may try something like `Mypassword1!` or `Myp433w0rd` instead.

In cases where the policy requires users to change their passwords on a regular basis, it is also common for users to just make minor, predictable changes to their preferred password. For example, `Mypassword1!` becomes `Mypassword1?` or `Mypassword2!`.

This knowledge of likely credentials and predictable patterns means that brute-force attacks can often be much more sophisticated, and therefore effective, than simply iterating through every possible combination of characters.

Username enumeration

Username enumeration is when an attacker is able to observe changes in the website's behavior in order to identify whether a given username is valid.

Username enumeration typically occurs either on the login page, for example, when you enter a valid username but an incorrect password, or on registration forms when you enter a username that is already taken. This greatly reduces the time and effort required to brute-force a login because the attacker is able to quickly generate a shortlist of valid usernames.

While attempting to brute-force a login page, you should pay particular attention to any differences in:

- **Status codes**: During a brute-force attack, the returned HTTP status code is likely to be the same for the vast majority of guesses because most of them will be wrong. If a guess returns a different status code, this is a strong indication that the username was correct. It is best practice for websites to always return the same status code regardless of the outcome, but this practice is not always followed.
- **Error messages**: Sometimes the returned error message is different depending on whether both the username AND password are incorrect or only the password was incorrect. It is best practice for websites to use identical, generic messages in both cases, but small typing errors sometimes creep in. Just one character out of place makes the two messages distinct, even in cases where the character is not visible on the rendered page.
- **Response times**: If most of the requests were handled with a similar response time, any that deviate from this suggest that something different was happening behind the scenes. This is another indication that the guessed username might be correct. For example, a website might only check whether the password is correct if the username is valid. This extra step might cause a slight increase in the response time. This may be subtle, but an attacker can make this delay more obvious by entering an excessively long password that the website takes noticeably longer to handle.

LAB APPRENTICE
Username enumeration via different responses →

LAB PRACTITIONER
Username enumeration via subtly different responses →

LAB PRACTITIONER
Username enumeration via response timing →

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.

LAB PRACTITIONER
Broken brute-force protection, IP block →

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.

LAB PRACTITIONER
Username enumeration via account lock →

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.

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.

LAB EXPERT
Broken brute-force protection, multiple credentials per request →

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, 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.