# LDAP - Data Exfiltration & Blind Exploitation

Now that we have discussed how to bypass authentication using LDAP injection in the previous section, we will focus on data exfiltration in this section. Even if the result of the LDAP search is not displayed to us, it is still possible to exfiltrate data with a methodology similar to the one used for blind SQL injections.

Just like with XPath injection, there is no sleep function in LDAP so we need an indicator by the web application that informs us whether the query returns any results or not. This leaks a bit of information to us, which allows us to exfiltrate data without it being displayed.

## Data Exfiltration

In a scenario where the web application displays results to us, data exfiltration is simple since we can inject a wildcard to match all entries with the specified attribute. Consider a search filter like the following, where we can supply the username in the uid attribute:

Code: ldap

(&(uid=admin)(objectClass=account))

If we simply supply a wildcard character as a username, the web application will display details about all accounts since the search filter becomes the following:

Code: ldap

(&(uid=\*)(objectClass=account))

We can achieve the same effect if we can inject a payload into an or clause of a search filter. Consider a search filter like this:

Code: ldap

(|(objectClass=organization)(objectClass=device))

This search filter matches all organization and device entries. If our payload is injected into the second objectClass attribute, we can force the backend to leak all entries by injecting a wildcard such that the search filter looks like this:

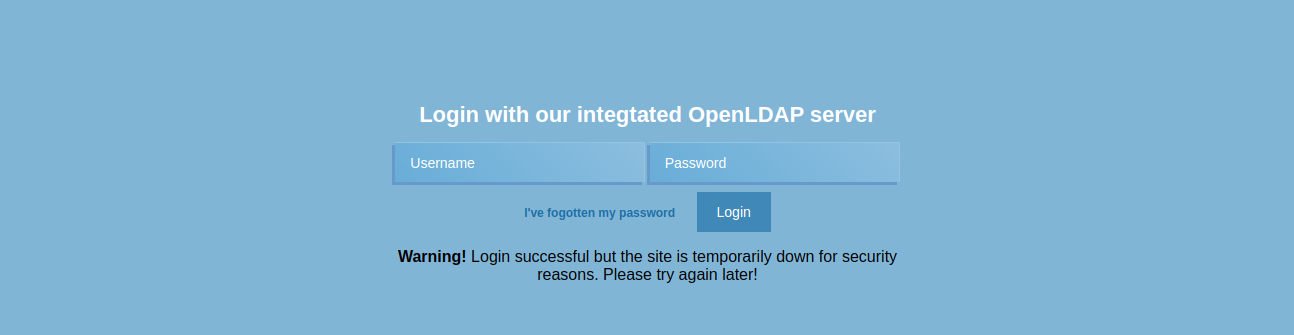
Code: ldap

(|(objectClass=organization)(objectClass=\*))

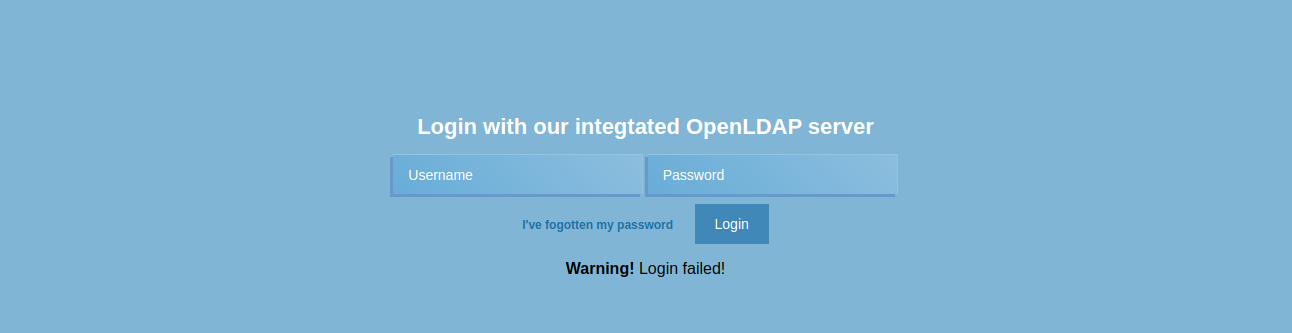
Thus, data exfiltration is straightforward provided the web application displays the results of the search filter to us.

## Blind Exploitation

Now let's discuss the more advanced and realistic cases of blind exploitation of LDAP injection vulnerabilities. When starting the lab below, we can see a slightly modified version of the lab from the previous section. When we attempt to log in, the administrator seems to have been notified about the LDAP injection from the previous section, so the site is in maintenance mode:



However, the web application responds differently, if we provide invalid credentials:

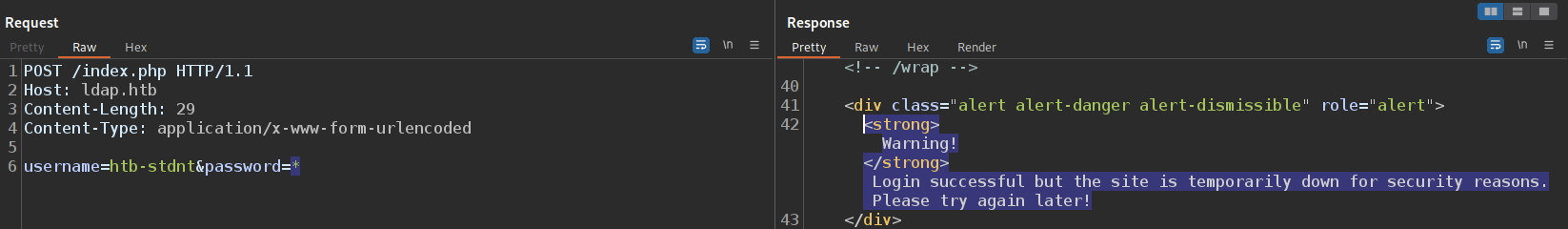


We can exploit this difference in the response to exfiltrate data from the directory server. Remember that the search filter used for authentication looks similar to the following:

Code: ldap

(&(uid=htb-stdnt)(password=p@ssw0rd))

We can verify that the web application is still vulnerable to LDAP injection by providing a wildcard as the password:

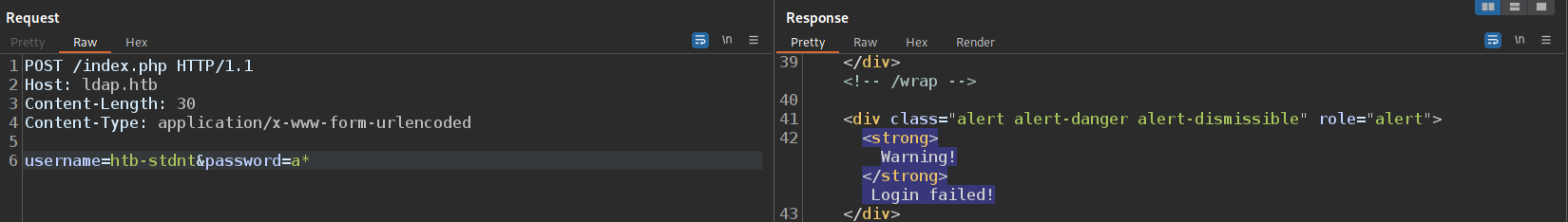


We can now brute-force the password character-by-character by injecting substring search filters. We can start with the first character by setting the password to a\*, resulting in the following search filter:

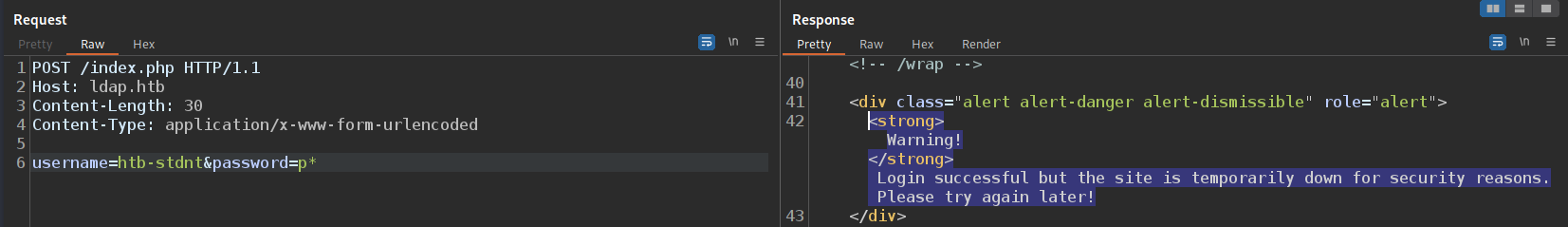
Code: ldap

(&(uid=htb-stdnt)(password=a\*))

This search filter will return data if the user's password starts with an a, otherwise, it does not. We can observe the web application to determine whether the login was successful or not. In our case, it is unsuccessful:



We now have to loop through the entire alphabet (including digits and special characters) to determine the first character of the password. In our case, it is p:

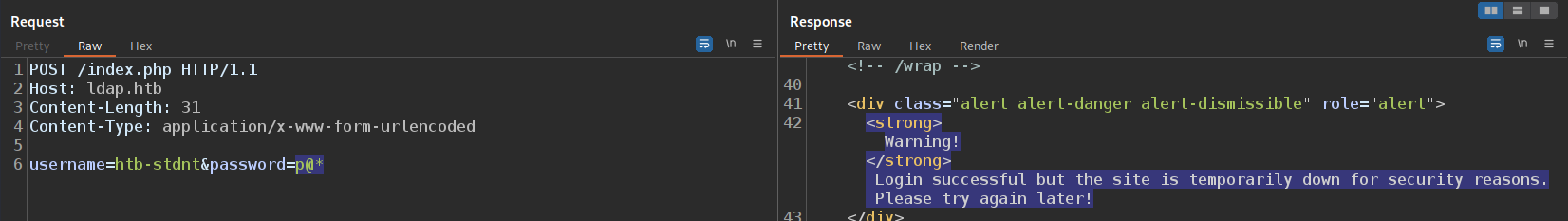


Now we can brute-force the second character, by altering the search filter to look like this:

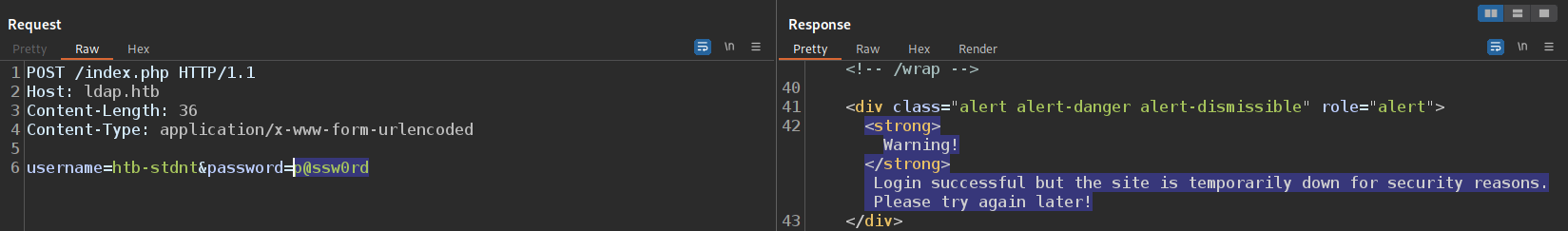
Code: ldap

(&(uid=htb-stdnt)(password=p@\*))

In our case, the second character is an @:



Now we have to repeat this process iteratively until no more characters are found, meaning we exfiltrated the entire password:

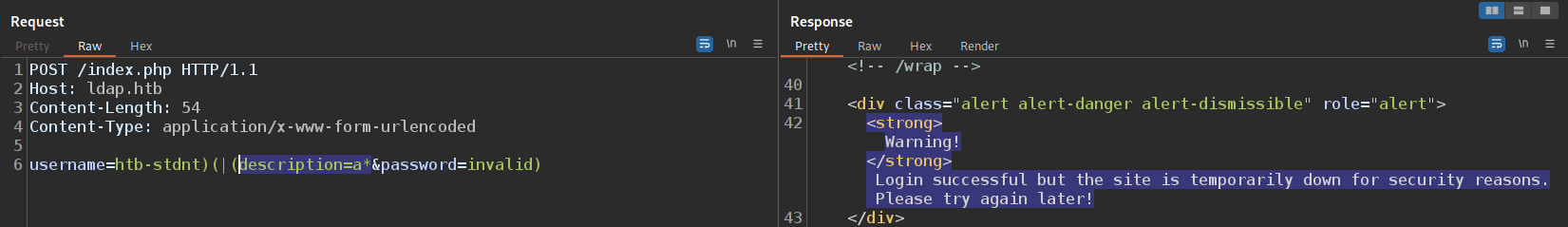


Furthermore, it is also possible to exfiltrate data from different attributes. As an example, we will target the description attribute of our user. If we submit a username of htb-stdnt)(|(description=\* and a password of invalid), the resulting search filter looks like this:

Code: ldap

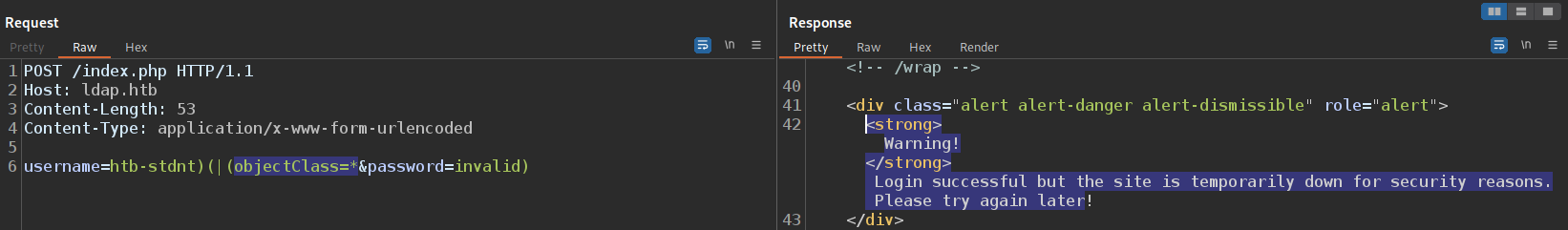
(&(uid=htb-stdnt)(|(description=\*)(password=invalid)))

Since the provided password is incorrect, our injected or clause only returns true if the condition for the description attribute is true. This now allows us to apply the same methodology as discussed above to brute-force the description attribute character-by-character:

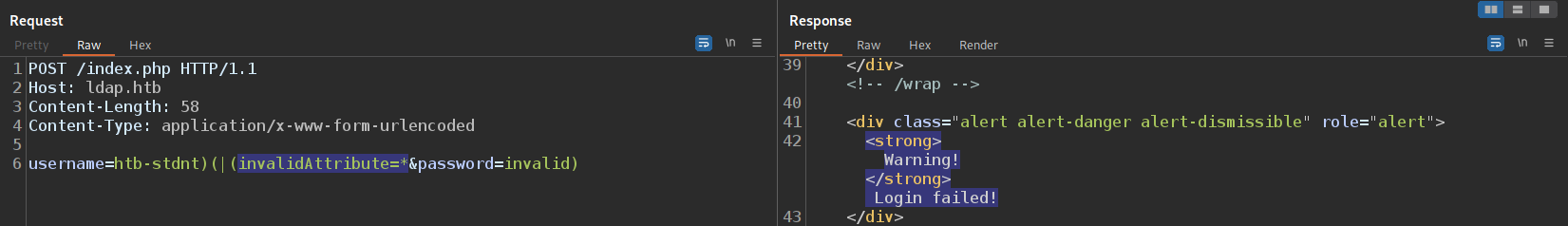


**Note:** Most LDAP attributes are case-insensitive. So if we need the correct casing, for instance for passwords, we might have to brute-force it.

We can target and exfiltrate any attribute of our specified user. Furthermore, we can even determine which attributes exist in our entry by specifying a wildcard. For valid attributes, the web application responds positively:



However, the web application responds negatively for invalid attributes:



**Note:** Similar to the exploitation of blind XPath injection, it is recommended to write a script to exfiltrate the data for us.