HTTP Basic Auth

For the simplest cases, you can use HTTP Basic Auth.

In HTTP Basic Auth, the application expects a header that contains a username and a password.

If it doesn't receive it, it returns an HTTP 401 "Unauthorized" error.

And returns a header `WWW-Authenticate` with a value of `Basic`, and an optional `realm` parameter.

That tells the browser to show the integrated prompt for a username and password.

Then, when you type that username and password, the browser sends them in the header automatically.

Simple HTTP Basic Auth

- Import `HTTPBasic` and `HTTPBasicCredentials`.
- Create a "`security` scheme" using `HTTPBasic`.
- Use that `security` with a dependency in your *path operation*.
- It returns an object of type `HTTPBasicCredentials`:
  - It contains the `username` and `password` sent.

`Python hl_lines="2  6  10" {!../../../docs_src/security/tutorial006.py!}`

When you try to open the URL for the first time (or click the "Execute" button in the docs) the browser will ask you for your username and password:

Check the username

Here's a more complete example.

Use a dependency to check if the username and password are correct.

For this, use the Python standard module `secrets` to check the username and password:

`Python hl_lines="1  11-13" {!../../../docs_src/security/tutorial007.py!}`

This will ensure that `credentials.username` is `"stanleyjobson"`, and that `credentials.password` is `"swordfish"`. This would be similar to:

```python
if not (credentials.username == "stanleyjobson") or not (credentials.password == "swordfish"
    # Return some error
    ...
```

But by using the `secrets.compare_digest()` it will be secure against a type of attacks called "timing attacks".

Timing Attacks

But what's a "timing attack"?

Let's imagine some attackers are trying to guess the username and password.

And they send a request with a username `johndoe` and a password `love123`.

Then the Python code in your application would be equivalent to something like:

```
if "johndoe" == "stanleyjobson" and "love123" == "swordfish":
    ...
```

But right at the moment Python compares the first `j` in `johndoe` to the first `s` in `stanleyjobson`, it will return `False`, because it already knows that those two strings are not the same, thinking that "there's no need to waste more computation comparing the rest of the letters". And your application will say "incorrect user or password".

But then the attackers try with username `stanleyjobsox` and password `love123`.

And your application code does something like:

```
if "stanleyjobsox" == "stanleyjobson" and "love123" == "swordfish":
    ...
```

Python will have to compare the whole `stanleyjobso` in both `stanleyjobsox` and `stanleyjobson` before realizing that both strings are not the same. So it will take some extra microseconds to reply back "incorrect user or password".

The time to answer helps the attackers At that point, by noticing that the server took some microseconds longer to send the "incorrect user or password" response, the attackers will know that they got *something* right, some of the initial letters were right.

And then they can try again knowing that it's probably something more similar to `stanleyjobsox` than to `johndoe`.

A "professional" attack Of course, the attackers would not try all this by hand, they would write a program to do it, possibly with thousands or millions of tests per second. And would get just one extra correct letter at a time.

But doing that, in some minutes or hours the attackers would have guessed the correct username and password, with the "help" of our application, just using the time taken to answer.

Fix it with `secrets.compare_digest()` But in our code we are actually using `secrets.compare_digest()`.

In short, it will take the same time to compare `stanleyjobsox` to `stanleyjobson` than it takes to compare `johndoe` to `stanleyjobson`. And the same for the password.

That way, using `secrets.compare_digest()` in your application code, it will be safe against this whole range of security attacks.

Return the error

After detecting that the credentials are incorrect, return an `HTTPException` with a status code 401 (the same returned when no credentials are provided) and add the header `WWW-Authenticate` to make the browser show the login prompt again:

```
Python hl_lines="15-19" {!../../../docs_src/security/tutorial007.py!}
```