Wow! This is hilariously bad!

With all the extraneous stuff stripped, out, the basic sequence of events is:

- The client sends a GET request to the server.
- The server sends back a 401 Unauthorized response with a `WWW-Authenticate` header.
- Your user agent, hopefully, puts together that it should query the user with
  a username/password popup.
- The client sends back a request to the same URL, with the `Authorization`
  header set to the auth scheme (in this case "Basic") and then the username, a
  colon, and the password, base64 encoded. In this case, `Y3MzMzg6cGFzc3dvcmQ=`
- The server checks to see that the username and password are all in order, and
  if so presents you with the contents of whatever behind's the URL.
  = And if the credentials *don't* match, the server sends another 401.
- After that, it looks like the browser saves the username and password
  somewhere, because I can reload the page and load files in that folder
  without it asking for authentication here.

"Extraneous stuff" here includes:
- The original TCP handshake to get the client/server talking
- The server helpfully telling me via code 454 that I should be querying
  `/basicauth/` with a slash, not `/basicauth`
- Several TCP keep-alive packets because I was taking my time punching in the
  password, so the server doesn't drop the connection.
  = As it turned out the server did eventually drop the connection, so I guess you
    can't just infinitely send Keep-Alives and tie up the whole server.
  = There is a thing called a Slow Loris attack though, where you make an HTTP
    request and send the body of it really slowly. (Usually this is done by just
    sending random nonsense in the user agent because you can put whatever you want
    there.) The server can't tell that this isn't just some poor schmuck's really
    bad wifi, and so will happily wait there until the end of time for the request
    to end.

I also manually made a request with:
  curl -vu cs338:password cs338.jeffondich.com/basicauth
and it looks like the server doesn't need to have you previously tried and
failed to authenticate in order to accept the password. (Which makes sense;
HTTP is a stateless protocol.)

Trying to `curl` for one of the files also requires me to give a password,
which is what leads me to believe that Firefox is saving the username and
password for the convenience of the user.

=====

Why is this so bad? Say I join a league of supervillains and they install me in
a position as a network sysadmin. If someone uses this authentication scheme I can
just read out the `Authentication` header and turn it back into their username and
password using `base64 -d`.

And from the RFC:

> ... users are known to choose weak passwords and to reuse them across
  authentication realms.

(By the way, by "joining a league of supervillains" I mean "going down to Blue
Monday and turning on Wireshark")

(I hope this being a single page is OK. I'm writing this in a plaintext
document and Firefox prints it to PDF very small.)