Rob Culbertson

CYOP – 300

* Lab 8 -

Test Table:

Screenshots:

1a – Successful Password UpdateA screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

1b – Incorrect Old PasswordA screenshot of a computer

AI-generated content may be incorrect.

1c – New Passwords Do Not Match

A screenshot of a computer

AI-generated content may be incorrect.

1d – Invalid New Password (Doesn’t Meet NIST or Complexity Reqs)

A screenshot of a computer

AI-generated content may be incorrect.

1e – New Password Too Short

A screenshot of a computer

AI-generated content may be incorrect.

1f – Username Invalid

A screenshot of a computer

AI-generated content may be incorrect.

2a – Successful pass\_check() validation (printed to console for testing)

A screenshot of a computer

AI-generated content may be incorrect.

2b – validpass12345! (No upper case)A screenshot of a computer

AI-generated content may be incorrect.

2c – Validpassword! (No Number)

A screenshot of a computer

AI-generated content may be incorrect.

2d – BadPass1234! (NIST List Trigger)

A screenshot of a computer screen

AI-generated content may be incorrect.

3a – Failed Login – Incorrect Password (Webpage and Log File)

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

3b – Failed Login: User Not Found

A screenshot of a computer

AI-generated content may be incorrect.

3c – Successful Login (No Line Added to Log File)

A screenshot of a computer

AI-generated content may be incorrect.

Pylint:

Writeup:

Flask Application Code:

I really enjoyed the implementation of the log file in this assignment. I think that being able to monitor failed attempts helps to provide information that can help me as a developer insulate my applications from bad actors. Since the log records not only the attempted username, but also the IP, I can detect when someone is trying brute force attempt to log into any given account, or spray attack across many different usernames and narrow it down to the IP address that is sending those requests. I can also review users who may be breaking (a hypothetical) ToS agreement where users aren’t supposed to share login information but we can see that a user is accessing an account from one IP and then seconds later from a different IP, etc. There is, of course, much refinement that can take place with this log, but I was surprised at how syntactically easy the implementation of the logging function was for how useful it could prove to be when developing applications like this one.

Using my old Password requirements from Lab 7, there wasn’t too much to add for checking against the NIST requirements. Once I was able to implement the file I/O for reading in the .txt file that provided the common password list, I was able to check against that list through a comparative loop to ensure that my users were not utilizing these common passwords. I then used print lines to test each of the complexity requirements and ensured that there were specific error messages for the different failure points. This meant that when something went wrong in my testing, I was able to almost instantly pinpoint the issue due to the unique error message for each possible error type, which also acted as feedback for the user to see that their password was too weak to be selected under my requirements.

Cyphers:

For the first cypher, it was apparent that the message had been put into morse code. When decoding the message, I wound up with “THIS SDEV 300 CLASS HAS SOME STRANGE REQUESTS.” Which appears to be a sensible decryption.

For the second message it appears to be some form of hash code at first glance which led me to think that it was a Base64 cipher. When reading about the Base64 ciphers, it’s use of uppercase, lowercase, and numbers stuck out to me as a possible solution for this code. When using the base64 algorithm to decrypt the output, the message became “So this is base64. Now I know.” Which seems like a practical output for the cipher.

The final code stuck out to me almost immediately as either a Caesar cipher, or something derived from one. The biggest indicators to me were the single letter ‘words’ that were surrounded by whitespace. This means that ‘W’ is shifted with the input being either ‘A’ or ‘I’ since those are the only single-letter words in English. From there we can test the two different values for N in the cypher. The first being N=22 (‘W’ -> ‘A’) which appears to be nonsense in the output. When we make N=14 (‘W’ -> ‘I’) everything falls into place and the output becomes “—Begin Key-- / I am so clever. No one could possibly figure this out. / -- End Key –". The output is immediately recognizable as plain English.