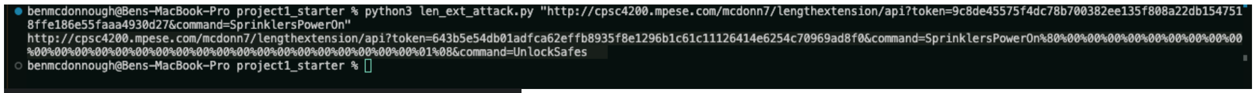
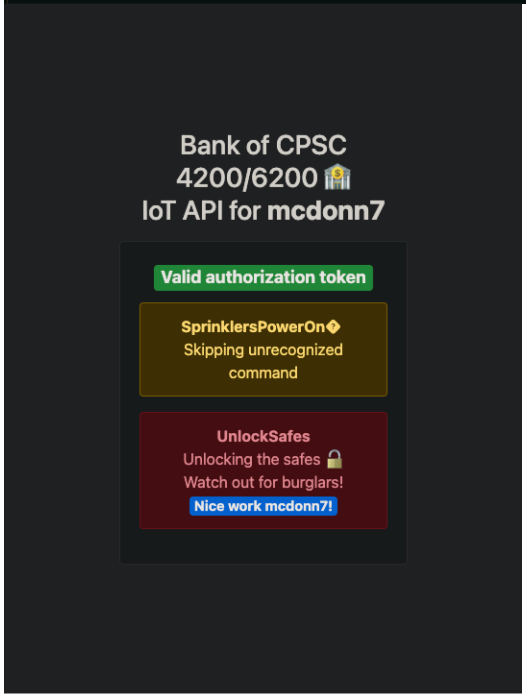
# Writeup Homework 1
Benjamin McDonnough

I worked on this homework by myself.
Question 1 (len_ext_attack):

I started off by setting variables in the code: the length of the secret password, the length of the suffix, and the padding. These lengths were used to create the padding needed and also the count for the hash. I then created the hash using sha256 with the state being from the url token and the count being the length of the secret password, the url suffix, and the padding. I then created a variable for the command needing to be added (&command=UnlockSafes). I updated the hash to include this command. I then updated the token for the url and the suffix so that both included the new command. All of this allows for the API to believe that the privileges needed to call the Unlock Safes command are there and for the Unlock Safes command to be cal



```
● benmcdonnough@Bens-MacBook-Pro project1_starter % python3 len_ext_attack.py "http://cpsc4200.mpese.com/mcdonn7/lengthextension/api?token=9c8de45575f4dc78b700382ee135f808a22db154751
  8ffe186e55faaa4930d27&command=SprinklersPowerOn"
  http://cpsc4200.mpese.com/mcdonn7/lengthextension/api?token=643b5e54db01adfca62effb8935f8e1296b1c61c11126414e6254c70969ad8f0&command=SprinklersPowerOn%80%00%00%00%00%00%00%00%00
  %00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%01%08&command=UnlockSafes
○ benmcdonnough@Bens-MacBook-Pro project1_starter % 
```



Bank of CPSC 4200/6200
IoT API for **mcdonn7**

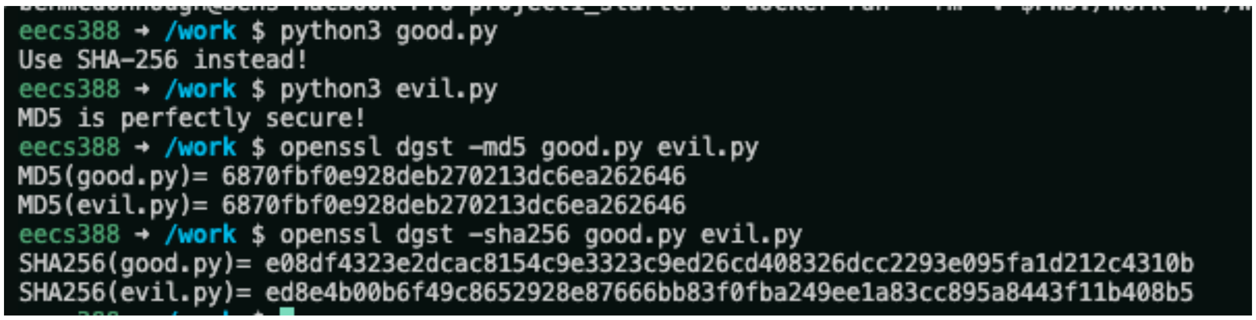
Valid authorization token

**SprinklersPowerOn�**
Skipping unrecognized command

**UnlockSafes**
Unlocking the safes
Watch out for burglars!
Nice work mcdonn7!

Question 2:

For the good.py and evil.py, I started off by making the prefix and suffix files. I made the prefix to be exactly like the example given in the instructions as this is all that was needed. In the suffix, I wrote code to read the file so that I could then check for a specific part of the hex later in the suffix code. Before the rest of the suffix code, I made sure to use the prefix, the files (good.py and evil.py) and fastcoll to create the files col1 and col2 and check for collisions. I took both col1 and col2, converted them to hex, and used the diff -y on both to find the specific section of hex that was different between the two. I then went back to the suffix file and made an if statement that looked for the specific section of hex in the read-in file. If the section was in it, then I told it to print the requirement for good.py. If it was not found, then it printed the requirement for evil.py. I then ensured that the MD5 was the same for both files and the SHA-256 was different.



```
eecs388 → /work $ python3 good.py
Use SHA-256 instead!
eecs388 → /work $ python3 evil.py
MD5 is perfectly secure!
eecs388 → /work $ openssl dgst -md5 good.py evil.py
MD5(good.py)= 6870fbf0e928deb270213dc6ea262646
MD5(evil.py)= 6870fbf0e928deb270213dc6ea262646
eecs388 → /work $ openssl dgst -sha256 good.py evil.py
SHA256(good.py)= e08df4323e2dcac8154c9e3323c9ed26cd408326dcc2293e095fa1d212c4310b
SHA256(evil.py)= ed8e4b00b6f49c8652928e87666bb83f0fba249ee1a83cc895a8443f11b408b5
```