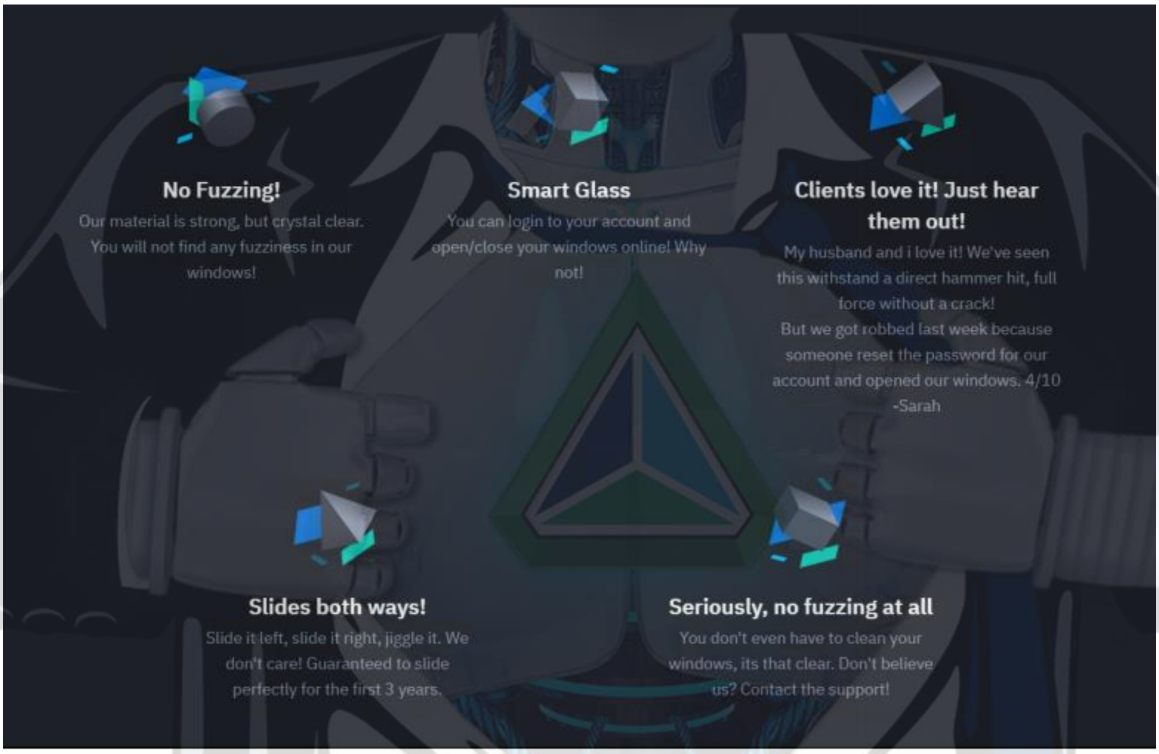
Challenge Name: Sliding Windows

Category: Crypto

Flag: STMCTF{UncR4cKaB1e_bUt_Sl1dEs_r1ghT_0ff}

This challenge has a small http service, and it hints that password reset functionality might have some trouble in the challenge text.

Some of the site texts hints us that we don't need to fuzz anything, as well as 404 page.



No Fuzzing!

Our material is strong, but crystal clear. You will not find any fuzziness in our windows!

Smart Glass

You can login to your account and open/close your windows online! Why not!

Clients love it! Just hear them out!

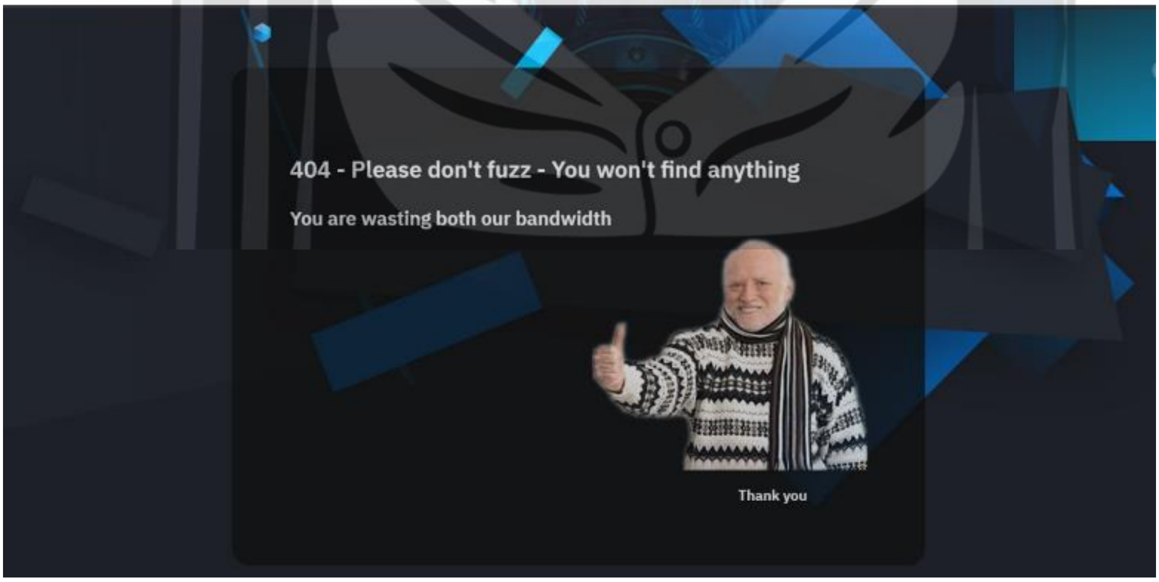
My husband and i love it! We've seen this withstand a direct hammer hit, full force without a crack! But we got robbed last week because someone reset the password for our account and opened our windows. 4/10 -Sarah

Slides both ways!

Slide it left, slide it right, jiggle it. We don't care! Guaranteed to slide perfectly for the first 3 years.

Seriously, no fuzzing at all

You don't even have to clean your windows, its that clear. Don't believe us? Contact the support!

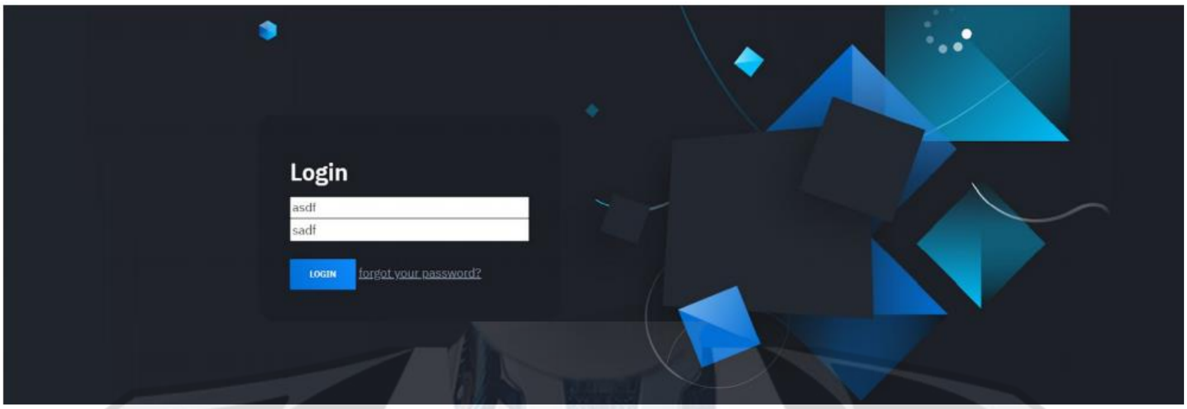


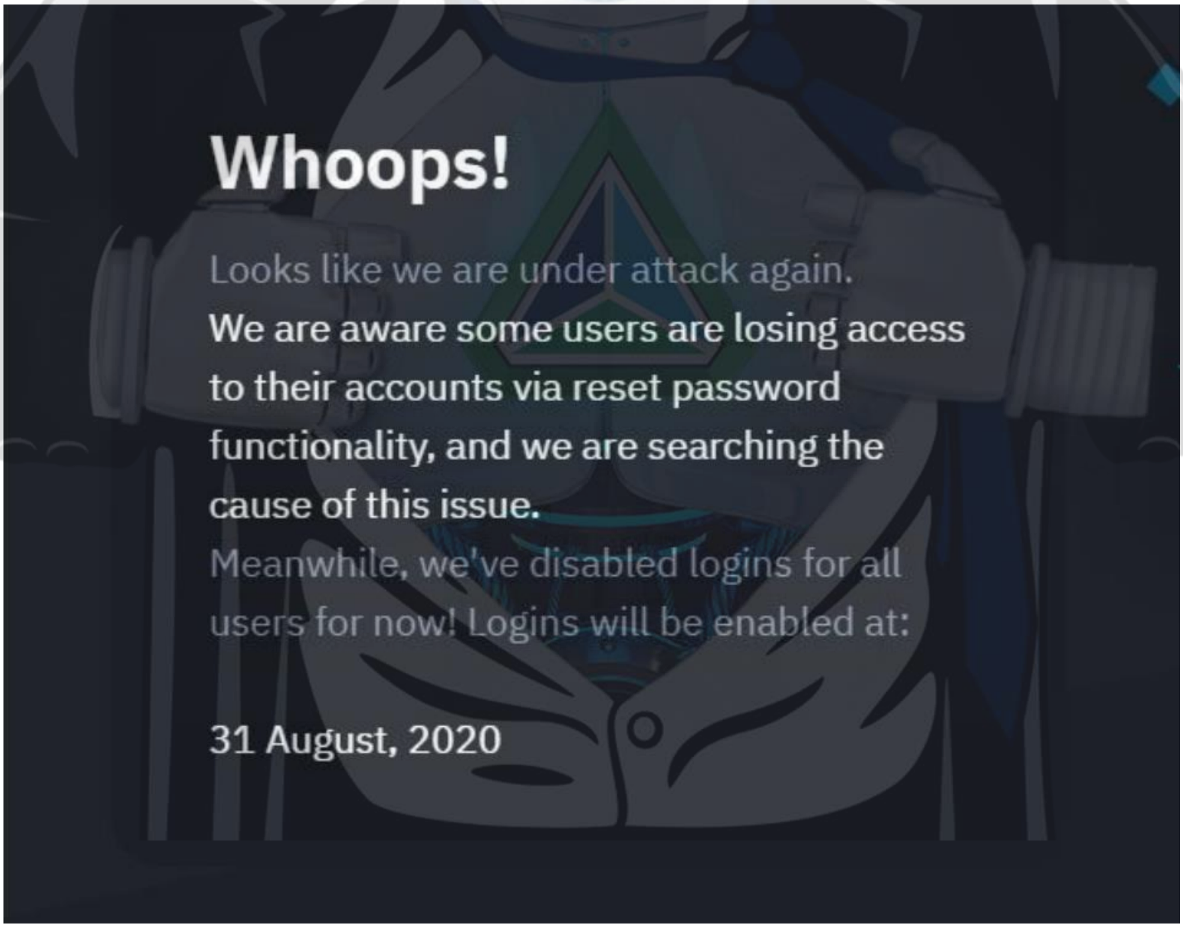
404 - Please don't fuzz - You won't find anything

You are wasting both our bandwidth

Thank you

There is a login page, but whatever we enter, it says logins are disabled, and login unlock time keeps getting delayed.





Whoops!

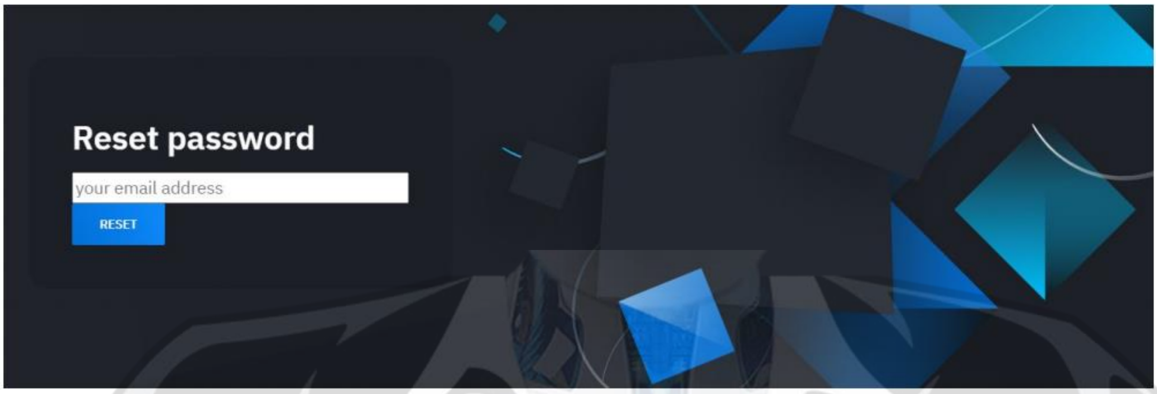
Looks like we are under attack again.

We are aware some users are losing access to their accounts via reset password functionality, and we are searching the cause of this issue.

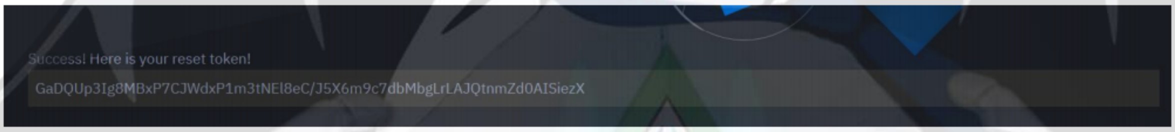
Meanwhile, we've disabled logins for all users for now! Logins will be enabled at:

31 August, 2020

Reset password link just under the login page gives us this panel.



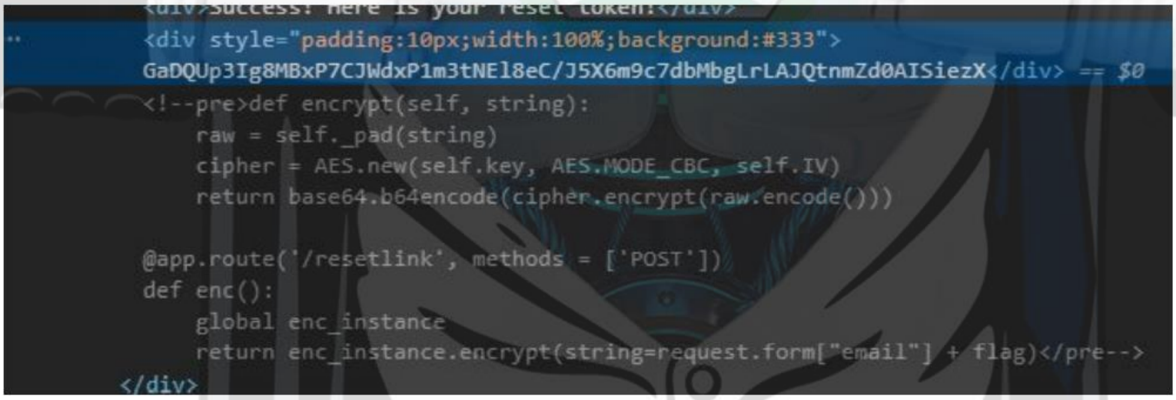
Whenever an email is given, it generates a token. Same emails give same token every time.



Success! Here is your reset token!

GaDQUp3Ig8MBxP7CJWdxP1m3tNEl8eC/J5X6m9c7dbMbgLrLAJQtnmZd0AISiezX

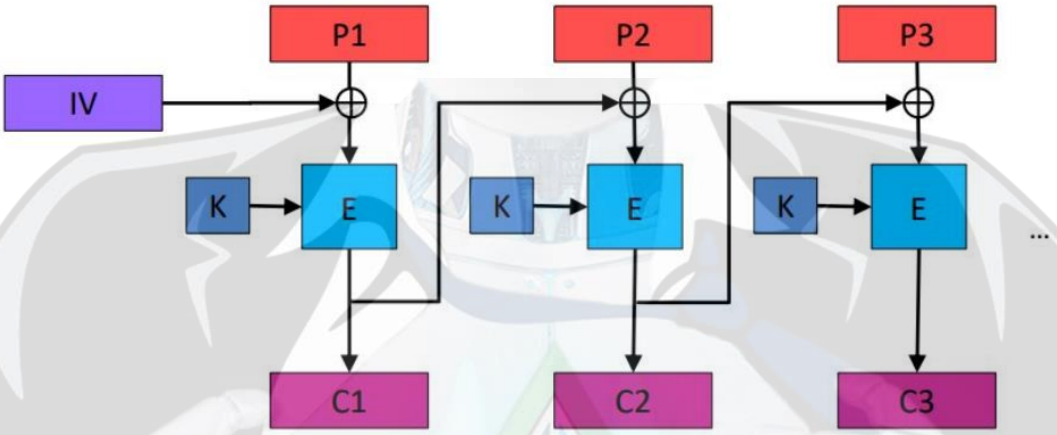
There are backend flask functions which are commented here, showing flag is appended to the end of our email and then encrypted to give us the token. It looks like both IV and key is reused because we get the same output everytime with the same input.



```
        <div>Success! Here is your reset token!</div>
        <div style="padding:10px;width:100%;background:#333">
        GaDQUp3Ig8MBxP7CJWdxP1m3tNEl8eC/J5X6m9c7dbMbgLrLAJQtnmZd0AISiezX</div> == $0
        <!--pre>def encrypt(self, string):
            raw = self._pad(string)
            cipher = AES.new(self.key, AES.MODE_CBC, self.IV)
            return base64.b64encode(cipher.encrypt(raw.encode()))

        @app.route('/resetlink', methods = ['POST'])
        def enc():
            global enc_instance
            return enc_instance.encrypt(string=request.form["email"] + flag)</pre-->
      </div>
```

Below is a good demonstration of AES CBC mode



When both key and IV are reused, P1 block will give out the same C1 block every time. P2 block will give out same C2 if and only if C1 block was also same, and so on.

P=	AAAAAAAAAAAAAAAA	????????????????	????????????????
C=	C1	C2	C3

Removing 1 character from the left padding

P=	AAAAAAAAAAAAAAA?	????????????????	????????????????
C=	C4	C5	C6

Now, attacker can brute force just the last character of P1, and if encrypts to C4, attacker will know the first character of the unknown plaintext.

Since flag format is STMCTF, first character will be “S”. Now, attacker can remove another character from the left padding and brute force the next character.

P=	AAAAAAAAAAAAAA??	????????????????	????????????????
C=	C5	CX	CX

Now that the attacker knows first character is S, they can Brute force last character of P1 again by adding S to the end of left padding.

P=	AAAAAAAAAAAAAAS?	????????????????	????????????????
C=	C5	CX	CX

If P1 encrypts to C5, attacker will know the second character of the flag. It is possible to repeat this as many times as necessary and with as many blocks as needed.

Automated Script

Getting reset token

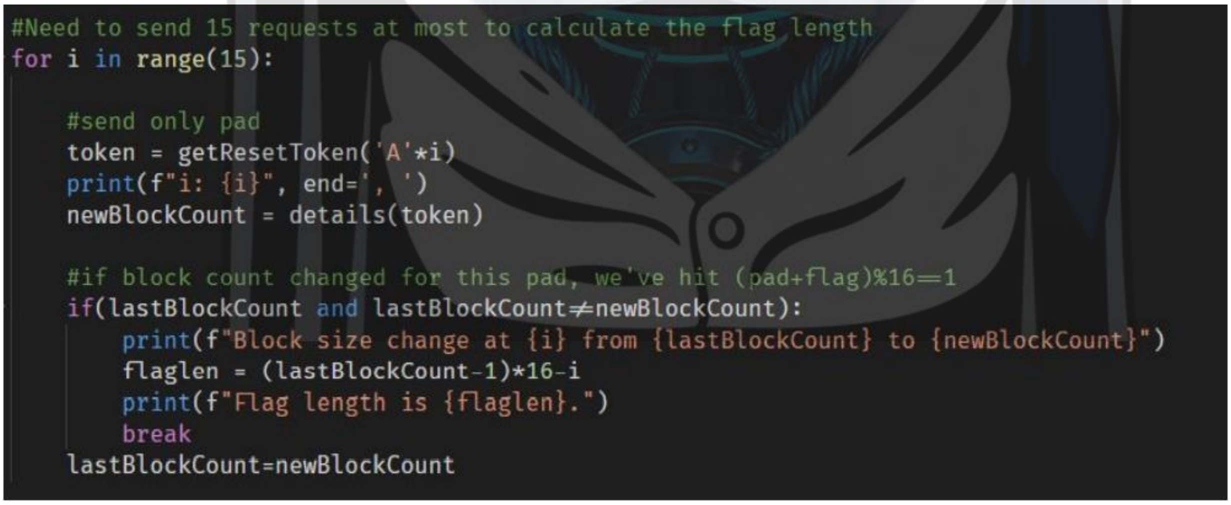
```
def getResetToken(msg):
    data["email"]=msg
    r = requests.post(url,data=data)
    return base64.b64decode(r.text)
```

Flag alphabet:

```
import string

flag_alphabet = string.digits + string.ascii_uppercase +
string.ascii_lowercase + '{}_'
```

First we need to determine how long the flag is, so we can add left-pad accordingly.



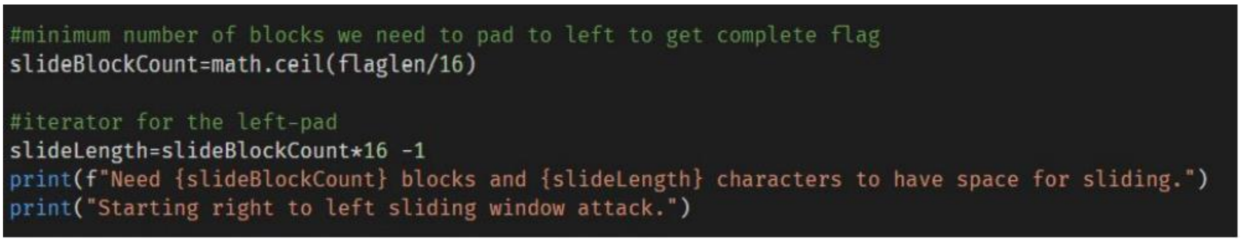
```
#Need to send 15 requests at most to calculate the flag length
for i in range(15):

    #send only pad
    token = getResetToken('A'*i)
    print(f"i: {i}", end=', ')
    newBlockCount = details(token)

    #if block count changed for this pad, we've hit (pad+flag)%16==1
    if(lastBlockCount and lastBlockCount!=newBlockCount):
        print(f"Block size change at {i} from {lastBlockCount} to {newBlockCount}")
        flaglen = (lastBlockCount-1)*16-i
        print(f"Flag length is {flaglen}.")
        break
    lastBlockCount=newBlockCount
```

If block count changes, we will know that our padding + flag is overflowing to the next block. Then we can calculate it by removing our padding length from the current size.

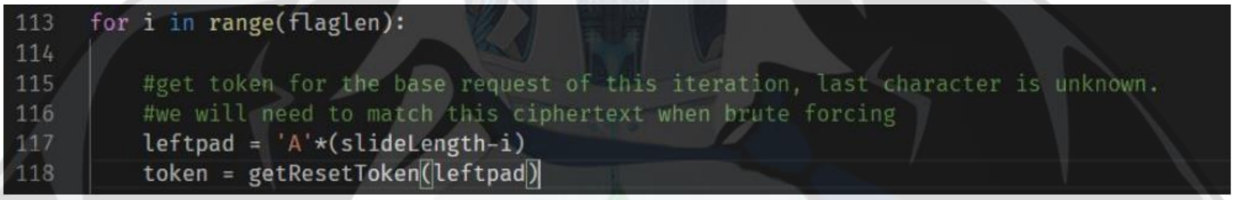
We can easily determine number of blocks needed to pad, as well as the initial pad length from the flag length.



```
#minimum number of blocks we need to pad to left to get complete flag
slideBlockCount=math.ceil(flaglen/16)

#iterator for the left-pad
slideLength=slideBlockCount*16 -1
print(f"Need {slideBlockCount} blocks and {slideLength} characters to have space for sliding.")
print("Starting right to left sliding window attack.")
```

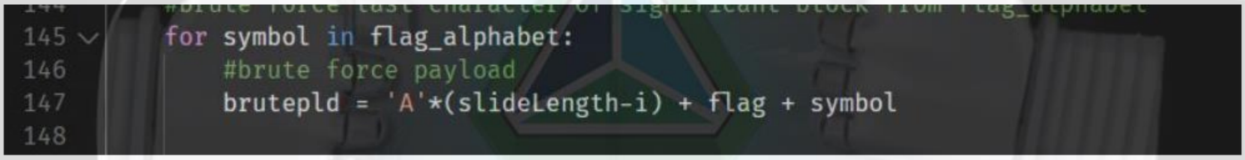
For each iteration on brute force, we will need to get the state of ciphertext when last block of our padding has only one unknown character first



```
for i in range(flaglen):

    #get token for the base request of this iteration, last character is unknown.
    #we will need to match this ciphertext when brute forcing
    leftpad = 'A'*(slideLength-i)
    token = getResetToken(leftpad)
```

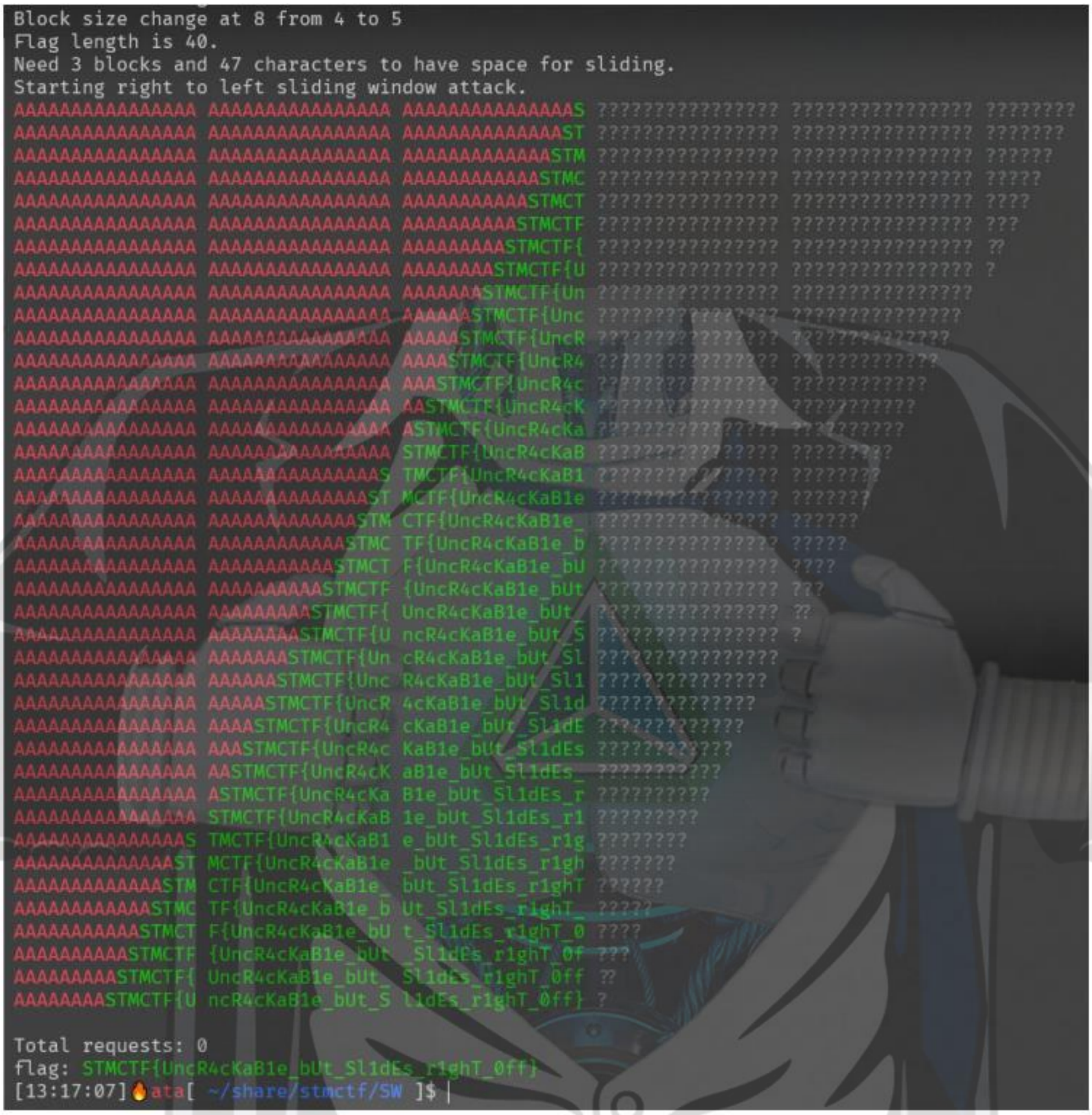
Then, we can add padding + part of the flag we brute forced so far + next symbol from the alphabet, and check if it matches the state of the ciphertext.



```
    #brute force last character of significant block from flag_alphabet
    for symbol in flag_alphabet:
        #brute force payload
        brutepld = 'A'*(slideLength-i) + flag + symbol

```

If it matches the initial ciphertext, we’ve replayed the scenario, and found the missing character. We can repeat until all the flag is revealed.



```
Block size change at 8 from 4 to 5
Flag length is 40.
Need 3 blocks and 47 characters to have space for sliding.
Starting right to left sliding window attack.
AAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAS ???????????????? ???????????????? ????????
AAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAA AAAAAAAAAAAAAAST ???????????????? ???????????????? ???????
AAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAA AAAAAAAAAAAAASTM ???????????????? ???????????????? ??????
AAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAA AAAAAAAAAAAASTMC ???????????????? ???????????????? ?????
AAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAA AAAAAAAAAAASTMCT ???????????????? ???????????????? ????
AAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAA AAAAAAAAAASTMCTF ???????????????? ???????????????? ???
AAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAA AAAAAAAAASTMCTF{ ???????????????? ???????????????? ??
AAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAA AAAAAAAASTMCTF{U ???????????????? ???????????????? ?
AAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAA AAAAAAASTMCTF{Un ???????????????? ????????????????
AAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAA AAAAAASTMCTF{Unc ???????????????? ???????????????
AAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAA AAAAASTMCTF{UncR ???????????????? ??????????????
AAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAA AAAASTMCTF{UncR4 ???????????????? ?????????????
AAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAA AAASTMCTF{UncR4c ???????????????? ????????????
AAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAA AASTMCTF{UncR4cK ???????????????? ???????????
AAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAA ASTMCTF{UncR4cKa ???????????????? ??????????
AAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAA STMCTF{UncR4cKaB ???????????????? ?????????
AAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAS TMCTF{UncR4cKaB1 ???????????????? ????????
AAAAAAAAAAAAAAAA AAAAAAAAAAAAAAST MCTF{UncR4cKaB1e ???????????????? ???????
AAAAAAAAAAAAAAAA AAAAAAAAAAAAASTM CTF{UncR4cKaB1e_ ???????????????? ??????
AAAAAAAAAAAAAAAA AAAAAAAAAAAASTMC TF{UncR4cKaB1e_b ???????????????? ?????
AAAAAAAAAAAAAAAA AAAAAAAAAAASTMCT F{UncR4cKaB1e_bU ???????????????? ????
AAAAAAAAAAAAAAAA AAAAAAAAAASTMCTF {UncR4cKaB1e_bUt ???????????????? ???
AAAAAAAAAAAAAAAA AAAAAAAAASTMCTF{ UncR4cKaB1e_bUt_ ???????????????? ??
AAAAAAAAAAAAAAAA AAAAAAAASTMCTF{U ncR4cKaB1e_bUt_S ???????????????? ?
AAAAAAAAAAAAAAAA AAAAAAASTMCTF{Un cR4cKaB1e_bUt_Sl ????????????????
AAAAAAAAAAAAAAAA AAAAAASTMCTF{Unc R4cKaB1e_bUt_Sl1 ???????????????
AAAAAAAAAAAAAAAA AAAAASTMCTF{UncR 4cKaB1e_bUt_Sl1d ??????????????
AAAAAAAAAAAAAAAA AAAASTMCTF{UncR4 cKaB1e_bUt_Sl1dE ?????????????
AAAAAAAAAAAAAAAA AAASTMCTF{UncR4c KaB1e_bUt_Sl1dEs ????????????
AAAAAAAAAAAAAAAA AASTMCTF{UncR4cK aB1e_bUt_Sl1dEs_ ???????????
AAAAAAAAAAAAAAAA ASTMCTF{UncR4cKa B1e_bUt_Sl1dEs_r ??????????
AAAAAAAAAAAAAAAA STMCTF{UncR4cKaB 1e_bUt_Sl1dEs_r1 ?????????
AAAAAAAAAAAAAAAS TMCTF{UncR4cKaB1 e_bUt_Sl1dEs_r1g ????????
AAAAAAAAAAAAAAST MCTF{UncR4cKaB1e _bUt_Sl1dEs_r1gh ???????
AAAAAAAAAAAAASTM CTF{UncR4cKaB1e_ bUt_Sl1dEs_r1ghT ??????
AAAAAAAAAAAASTMC TF{UncR4cKaB1e_b Ut_Sl1dEs_r1ghT_ ?????
AAAAAAAAAAASTMCT F{UncR4cKaB1e_bU t_Sl1dEs_r1ghT_0 ????
AAAAAAAAAASTMCTF {UncR4cKaB1e_bUt _Sl1dEs_r1ghT_0f ???
AAAAAAAAASTMCTF{ UncR4cKaB1e_bUt_ Sl1dEs_r1ghT_0ff ??
AAAAAAAASTMCTF{U ncR4cKaB1e_bUt_S l1dEs_r1ghT_0ff} ?

Total requests: 0
flag: STMCTF{UncR4cKaB1e_bUt_Sl1dEs_r1ghT_0ff}
[13:17:07]🔥ata[ ~/share/stmctf/SW ]$ |
```