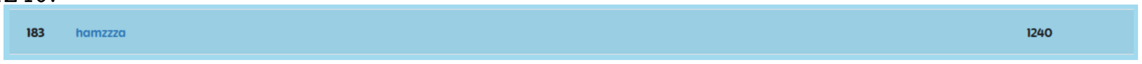
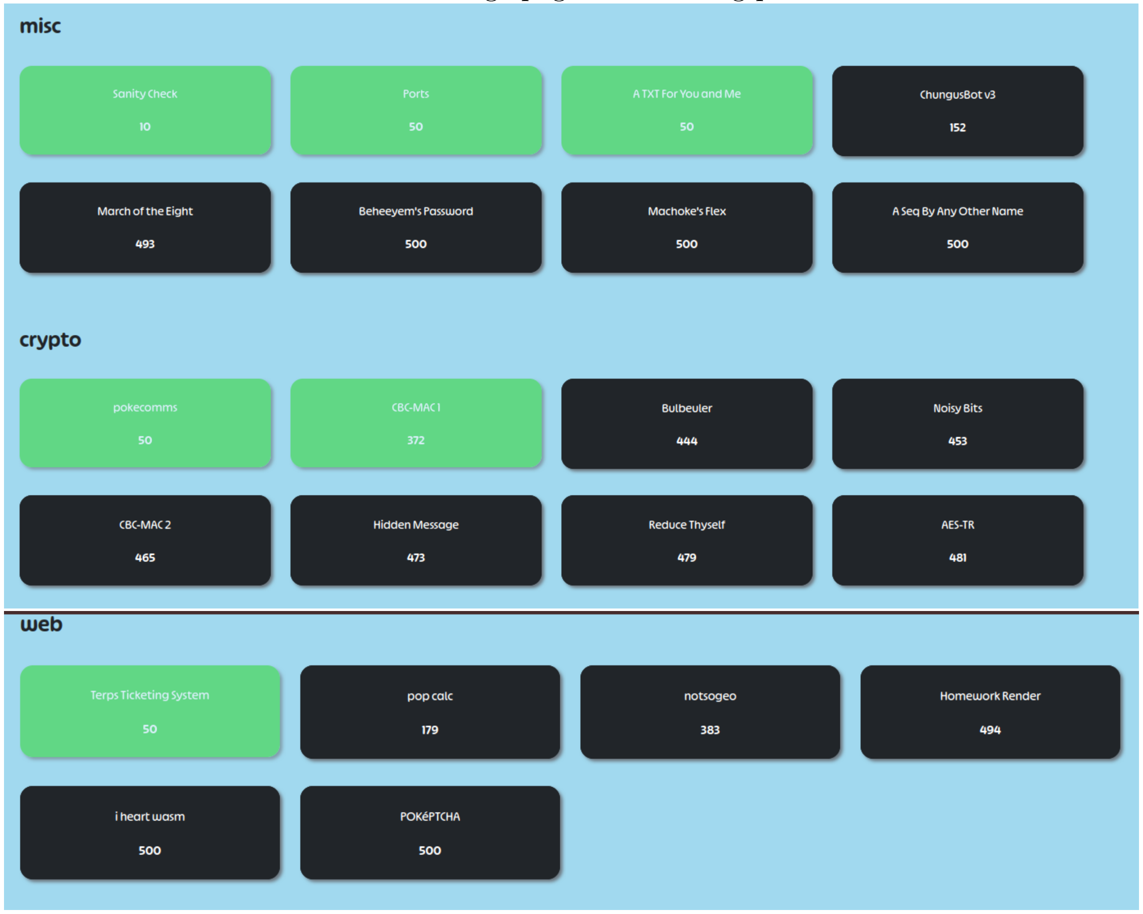
Computer Security **Name:** Ameer Hamza
CTF2: Online CTF UMD

The UMDCTF is at `https://umdctf.io/`, which revolves around Pokemons, hence some tasks will be specifically designed like that. My username at the CTF competition is `hamzzza`. My score is `1240`.



183	hamzzza	1240

I also took screenshots for the challenge page after solving problems:



misc

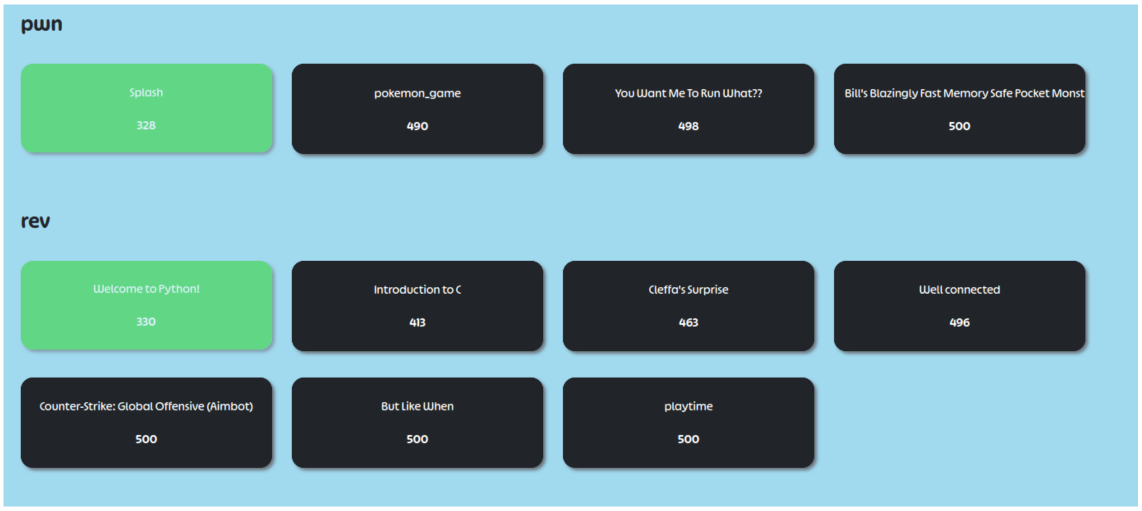
Sanity Check 10	Ports 50	A TXT For You and Me 50	ChungusBot v3 152
March of the Eight 493	Beheeyem's Password 500	Machoke's Flex 500	A Seq By Any Other Name 500

crypto

pokecomms 50	CBC-MAC 1 372	Bulbeuler 444	Noisy Bits 453
CBC-MAC 2 465	Hidden Message 473	Reduce Thyself 479	AES-TR 481

web

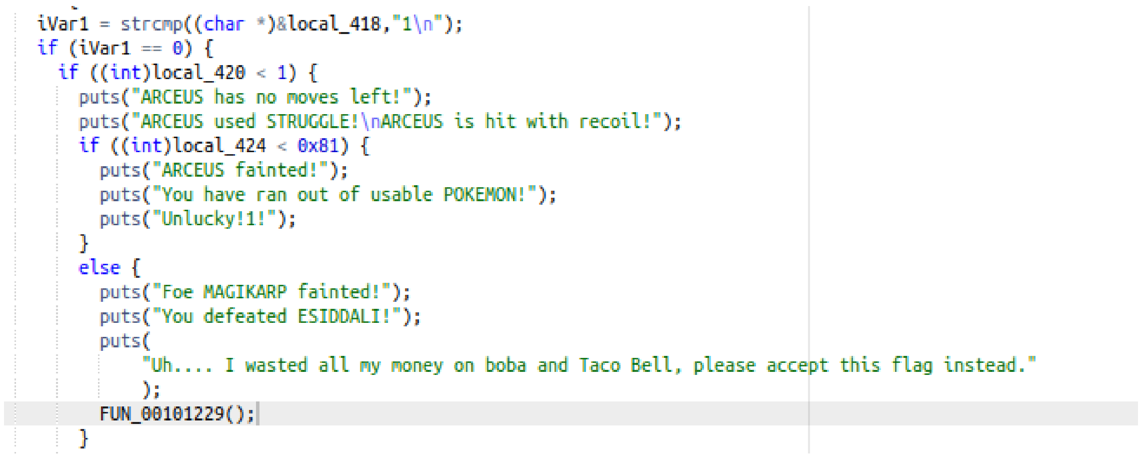
Terps Ticketing System 50	pop calc 179	notsogeo 383	Homework Render 494
i heart wasm 500	POKéPTCHA 500		



I was able to screenshot the description of some of the challenges but not all, before they ended the competition. Not sure if an official write-up will be available for the challenges, but it seems so, from the communication. I will explain in words if an actual challenge description is unavailable.

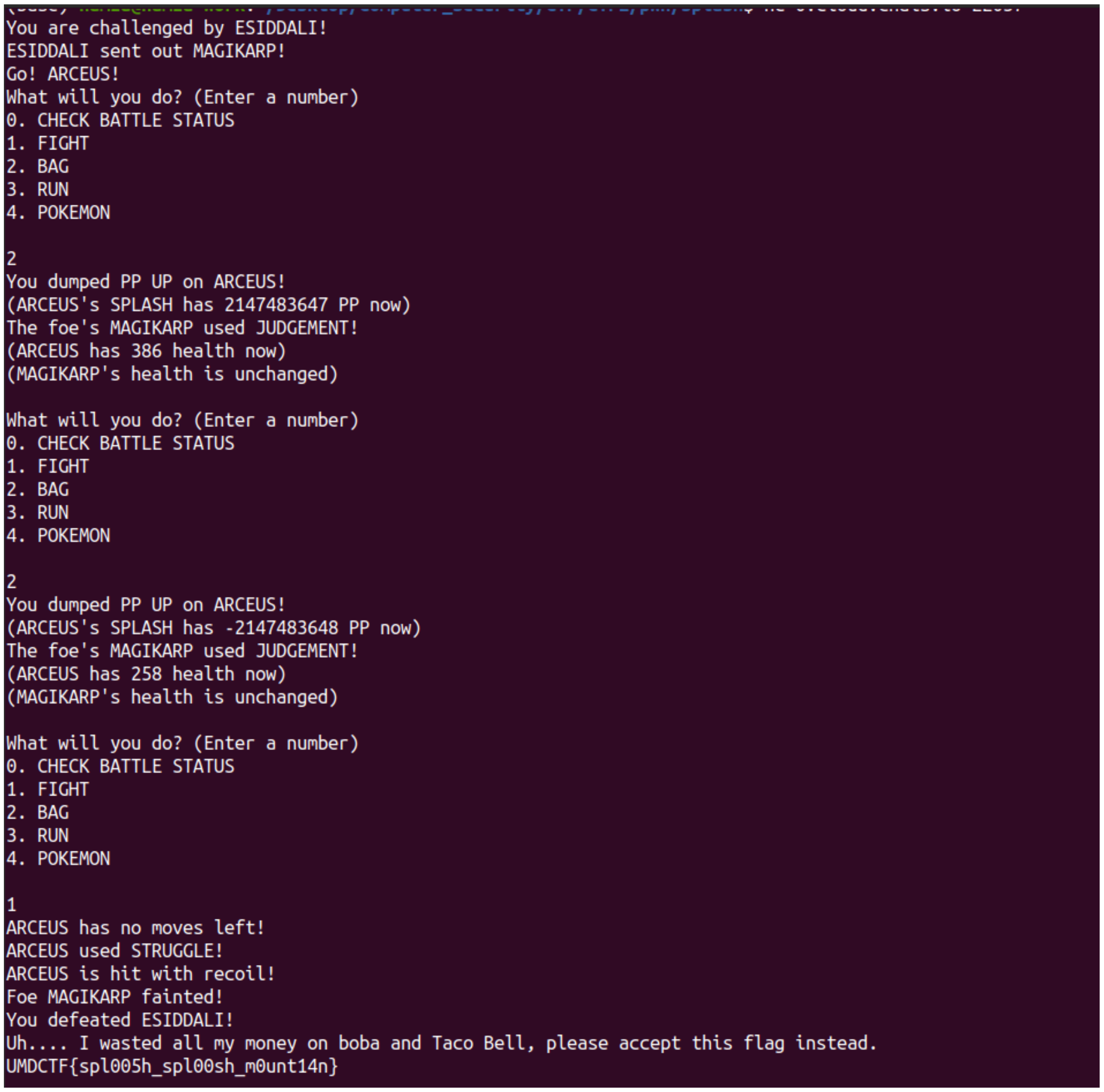
1. **Pwn**:

   (a) **Splash**: I do not have a screenshot of the challenge description. We have a binary `splash` whose partial decompiled version, using Ghidra decompiler, is given here:



```
  iVar1 = strcmp((char *)&local_418,"1\n");
  if (iVar1 == 0) {
    if ((int)local_420 < 1) {
      puts("ARCEUS has no moves left!");
      puts("ARCEUS used STRUGGLE!\nARCEUS is hit with recoil!");
      if ((int)local_424 < 0x81) {
        puts("ARCEUS fainted!");
        puts("You have ran out of usable POKEMON!");
        puts("Unlucky!1!");
      }
      else {
        puts("Foe MAGIKARP fainted!");
        puts("You defeated ESIDDALI!");
        puts(
            "Uh.... I wasted all my money on boba and Taco Bell, please accept this flag instead."
            );
        FUN_00101229();
      }
```

The basic idea of the task is a fight between 2 Pokemons. Your Pokemon has limited health, which decreases every time your Pokemon attacks the opponent because of some special power (a counterattack) of other Pokemon. Hence, if you keep fighting, you will lose. Also, your Pokemon has attack power as a very large number. Another option than `Fight` is `Bag`, which I do not exactly know what it does but it can increase the attack power of your Pokemon, but every time the health decreases drastically. In the above screenshot, we have two variables, `local_424` as health and `local_420` as attack power. The function `FUN_00101229` gives us the flag. We need to pass certain checks. The attack power needs to be less than 1, but the health needs to be not less than 0x80, which seems counter-intuitive as health will end before attack power goes to 0. It seems we can keep adding to the attack power and it overflows, making it a very large negative value, which is less than 1, while the health hasn't yet decreased as much. This gives us the flag as shown.

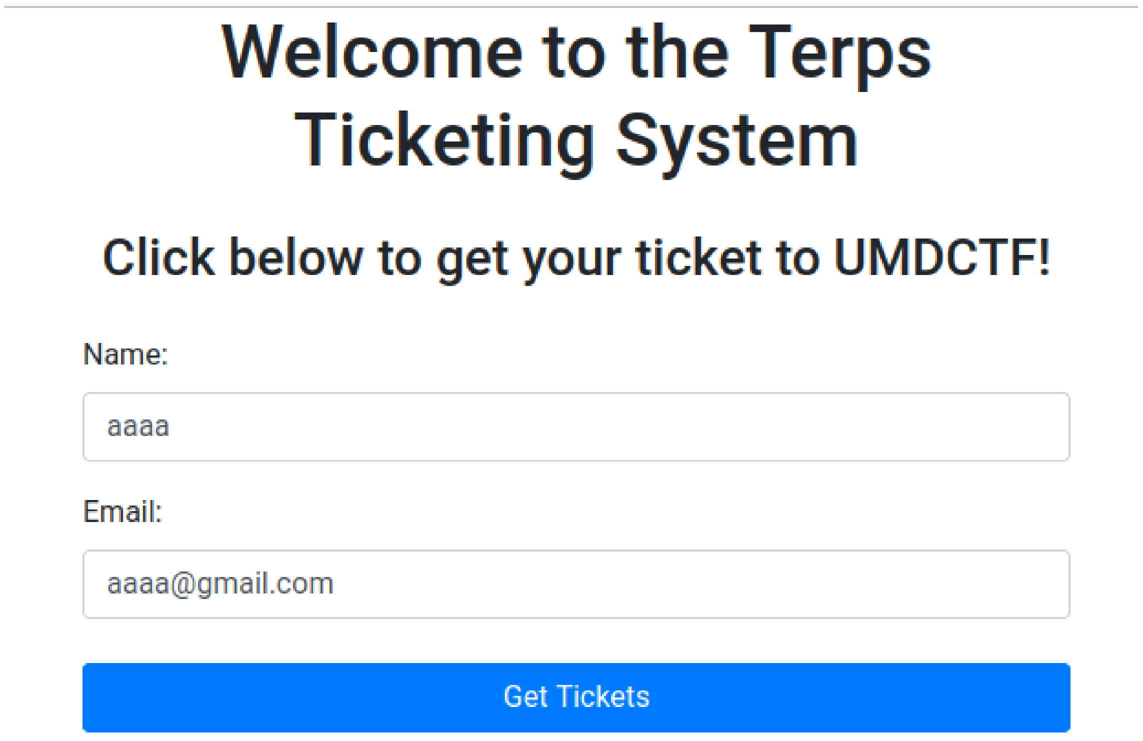


```
You are challenged by ESIDDALI!
ESIDDALI sent out MAGIKARP!
Go! ARCEUS!
What will you do? (Enter a number)
0. CHECK BATTLE STATUS
1. FIGHT
2. BAG
3. RUN
4. POKEMON

2
You dumped PP UP on ARCEUS!
(ARCEUS's SPLASH has 2147483647 PP now)
The foe's MAGIKARP used JUDGEMENT!
(ARCEUS has 386 health now)
(MAGIKARP's health is unchanged)

What will you do? (Enter a number)
0. CHECK BATTLE STATUS
1. FIGHT
2. BAG
3. RUN
4. POKEMON

2
You dumped PP UP on ARCEUS!
(ARCEUS's SPLASH has -2147483648 PP now)
The foe's MAGIKARP used JUDGEMENT!
(ARCEUS has 258 health now)
(MAGIKARP's health is unchanged)

What will you do? (Enter a number)
0. CHECK BATTLE STATUS
1. FIGHT
2. BAG
3. RUN
4. POKEMON

1
ARCEUS has no moves left!
ARCEUS used STRUGGLE!
ARCEUS is hit with recoil!
Foe MAGIKARP fainted!
You defeated ESIDDALI!
Uh.... I wasted all my money on boba and Taco Bell, please accept this flag instead.
UMDCTF{spl005h_spl00sh_m0unt14n}
```

2. **Web**:

   (a) **Terps Ticketing System**

   We are given the following link: `https://tts.chall.lol/` which is a ticketing system.

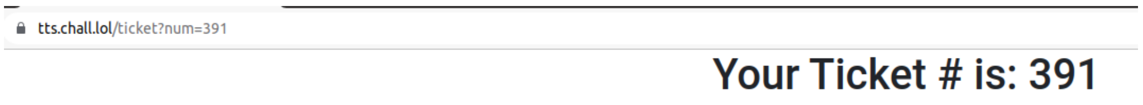

Welcome to the Terps Ticketing System

Click below to get your ticket to UMDCTF!

Name: aaaa

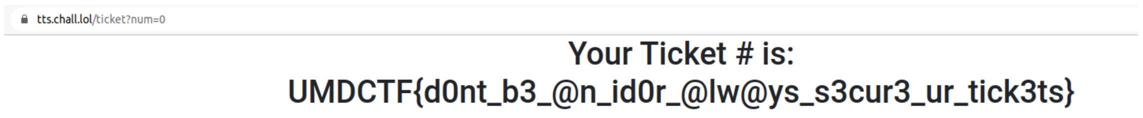
Email: aaaa@gmail.com

If we provide a name and an email, it gives us a random ticket number from 1 to some number.


tts.chall.lol/ticket?num=391

Your Ticket # is: 391

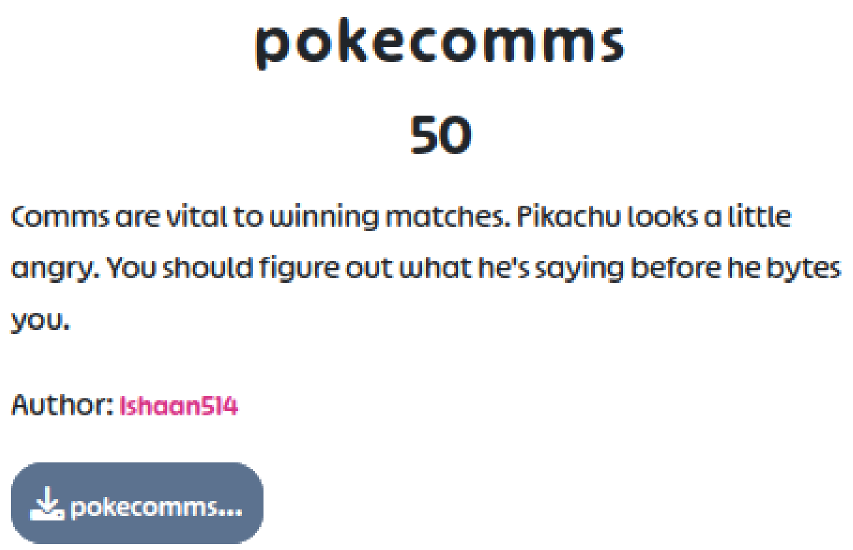
Turns out, if you directly give it the num=0, it will give you the flag.


tts.chall.lol/ticket?num=0

Your Ticket # is:
UMDCTF{d0nt_b3_@n_id0r_@lw@ys_s3cur3_ur_tick3ts}

3. **Crypto**:

   (a) **Pokecomms** I have a screenshot for the challenge description.


pokecomms

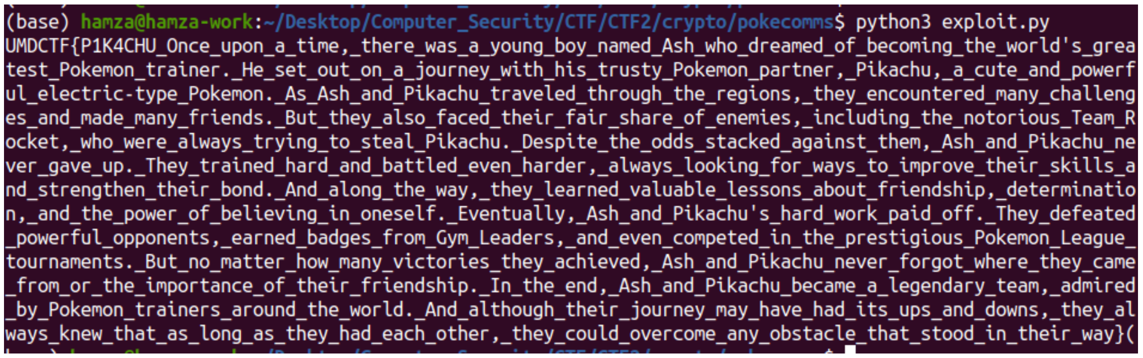
50

Comms are vital to winning matches. Pikachu looks a little angry. You should figure out what he's saying before he bytes you.

Author: Ishaan514

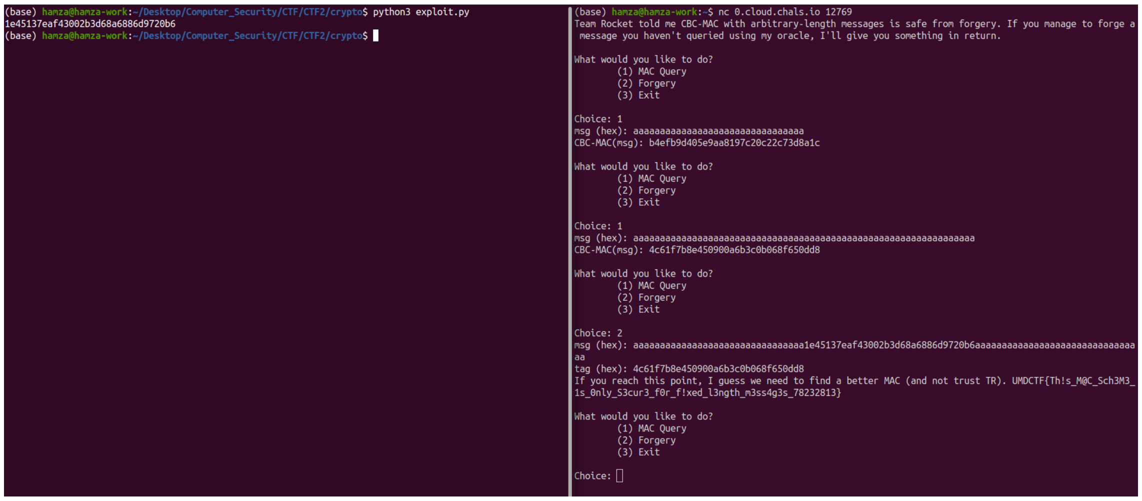
pokecomms...

We are also given a long text file with the sounds of Pikachu. The sound is either `CHU!` or `PIKA`. However, we notice that all sounds are grouped as 8 sounds in each line. Also, there are only two possible sounds. Made me think that may be each line is a byte and each sound is a bit, with `CHU!` as 0 and `PIKA` as 1 (I tried both combinations and this worked). I wrote a small program in `exploit.py` to convert the given file to binary code, then to a string considering we are given ascii values. Here is the result as a flag:



```
(base) hamza@hamza-work:~/Desktop/Computer_Security/CTF/CTF2/crypto/pokecomms$ python3 exploit.py
UMDCTF{P1K4CHU_Once_upon_a_time,_there_was_a_young_boy_named_Ash_who_dreamed_of_becoming_the_world's_grea
test_Pokemon_trainer._He_set_out_on_a_journey_with_his_trusty_Pokemon_partner,_Pikachu,_a_cute_and_powerf
ul_electric-type_Pokemon._As_Ash_and_Pikachu_traveled_through_the_regions,_they_encountered_many_challeng
es_and_made_many_friends._But_they_also_faced_their_fair_share_of_enemies,_including_the_notorious_Team_R
ocket,_who_were_always_trying_to_steal_Pikachu._Despite_the_odds_stacked_against_them,_Ash_and_Pikachu_ne
ver_gave_up._They_trained_hard_and_battled_even_harder,_always_looking_for_ways_to_improve_their_skills_a
nd_strengthen_their_bond._And_along_the_way,_they_learned_valuable_lessons_about_friendship,_determinatio
n,_and_the_power_of_believing_in_oneself._Eventually,_Ash_and_Pikachu's_hard_work_paid_off._They_defeated
_powerful_opponents,_earned_badges_from_Gym_Leaders,_and_even_competed_in_the_prestigious_Pokemon_League_
tournaments._But_no_matter_how_many_victories_they_achieved,_Ash_and_Pikachu_never_forgot_where_they_came
_from_or_the_importance_of_their_friendship._In_the_end,_Ash_and_Pikachu_became_a_legendary_team,_admired
_by_Pokemon_trainers_around_the_world._And_although_their_journey_may_have_had_its_ups_and_downs,_they_al
ways_knew_that_as_long_as_they_had_each_other,_they_could_overcome_any_obstacle_that_stood_in_their_way}(
```

(b) **CBC-MAC** We are given a program that does `CBC-MAC` encryption and claims that CBC-MAC with arbitrary-length messages is safe from forgery, so if I can provide a forged message that the Oracle hasn't seen yet, it will give the flag. For each message, the Oracle sends its tag (the cipher), and we can send a message and its tag to verify. We can construct such a message which is not seen by the Oracle means we haven't sent it before, still, we know its tag already [1]. Specifically, we first send a message as `m: "aa"*16`, we get a tag `t`. We then send a multi-block message `m_prime="aa"*32` (2 blocks). We get a tag `t_prime`. We construct another message as `m_prime_prime = m || m_prime[0] XOR t || m_prime[1]`. The claim is that the tag for it will still be `t_prime`, because tag of `m` is `t`, which will be passed to the second part of `m_prime_prime`, so `m_prime[0] XOR t XOR t = m_prime[0]`. Hence, the rest of the message becomes `m_prime` whose tag is given as `t`. We show the result below:



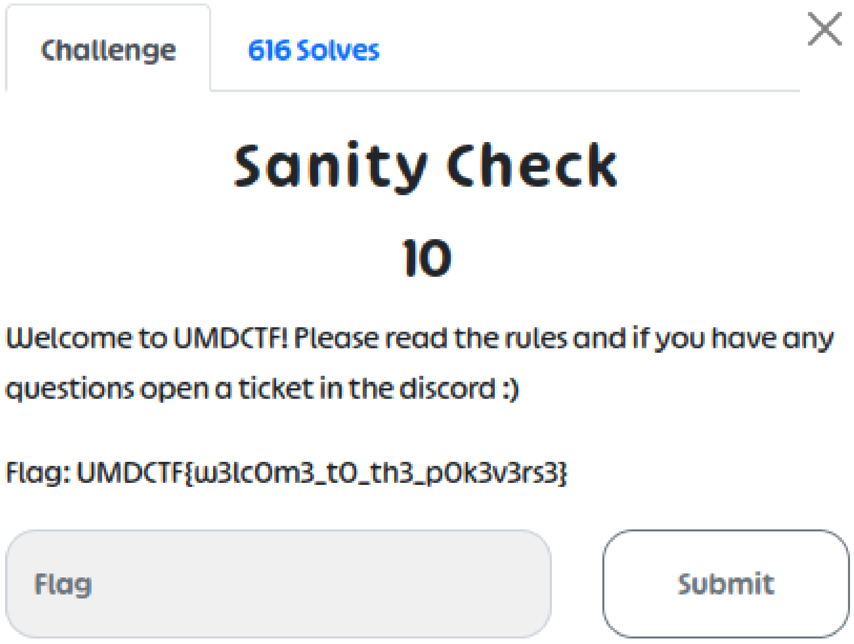
```
(base) hamza@hamza-work:~/Desktop/Computer_Security/CTF/CTF2/crypto$ python3 exploit.py
1e45137eaf43002b3d68a6886d9720b6
(base) hamza@hamza-work:~/Desktop/Computer_Security/CTF/CTF2/crypto$ 
```

```
(base) hamza@hamza-work:~$ nc 0.cloud.chals.io 12769
Team Rocket told me CBC-MAC with arbitrary-length messages is safe from forgery. If you manage to forge a
 message you haven't queried using my oracle, I'll give you something in return.

What would you like to do?
        (1) MAC Query
        (2) Forgery
        (3) Exit

Choice: 1
msg (hex): aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
CBC-MAC(msg): b4efb9d405e9aa8197c20c22c73d8a1c

What would you like to do?
        (1) MAC Query
        (2) Forgery
        (3) Exit

Choice: 1
msg (hex): aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
CBC-MAC(msg): 4c61f7b8e450900a6b3c0b068f650dd8

What would you like to do?
        (1) MAC Query
        (2) Forgery
        (3) Exit

Choice: 2
msg (hex): aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa1e45137eaf43002b3d68a6886d9720b6aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aa
tag (hex): 4c61f7b8e450900a6b3c0b068f650dd8
If you reach this point, I guess we need to find a better MAC (and not trust TR). UMDCTF{Th!s_M@C_Sch3M3_
1s_0nly_S3cur3_f0r_f!xed_l3ngth_m3ss4g3s_78232813}

What would you like to do?
        (1) MAC Query
        (2) Forgery
        (3) Exit

Choice: 
```

The code for `CBC-MAC` is in `cbc-mac1.py` and `exploit.py` just computes XOR of two numbers. On the left, I just compute `m_prime XOR t`.

4. **Misc**:

   (a) **Sanity Checks** It simply gives the flag.



Sanity Check

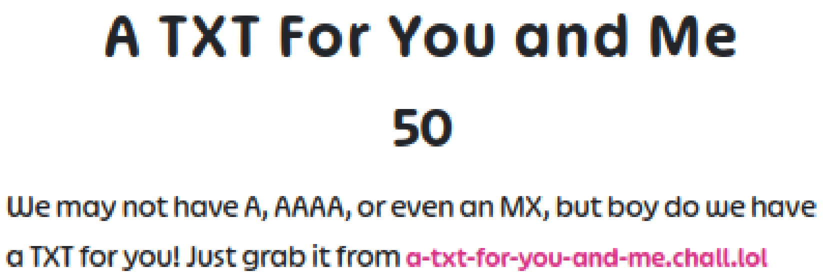
10

Welcome to UMDCTF! Please read the rules and if you have any questions open a ticket in the discord :)

Flag: UMDCTF{w3lc0m3_t0_th3_p0k3v3rs3}

   (b) **A TXT for you and me**

   Here is the screenshot for the description.

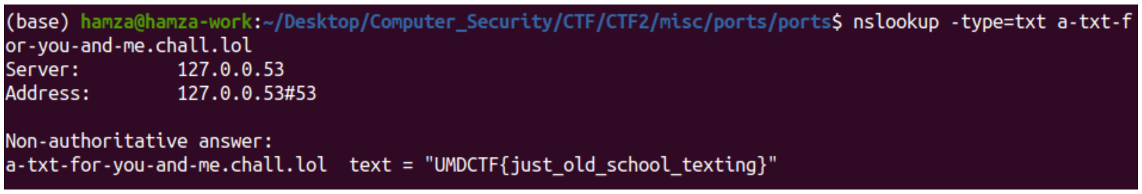


A TXT For You and Me

50

We may not have A, AAAA, or even an MX, but boy do we have a TXT for you! Just grab it from a-txt-for-you-and-me.chall.lol

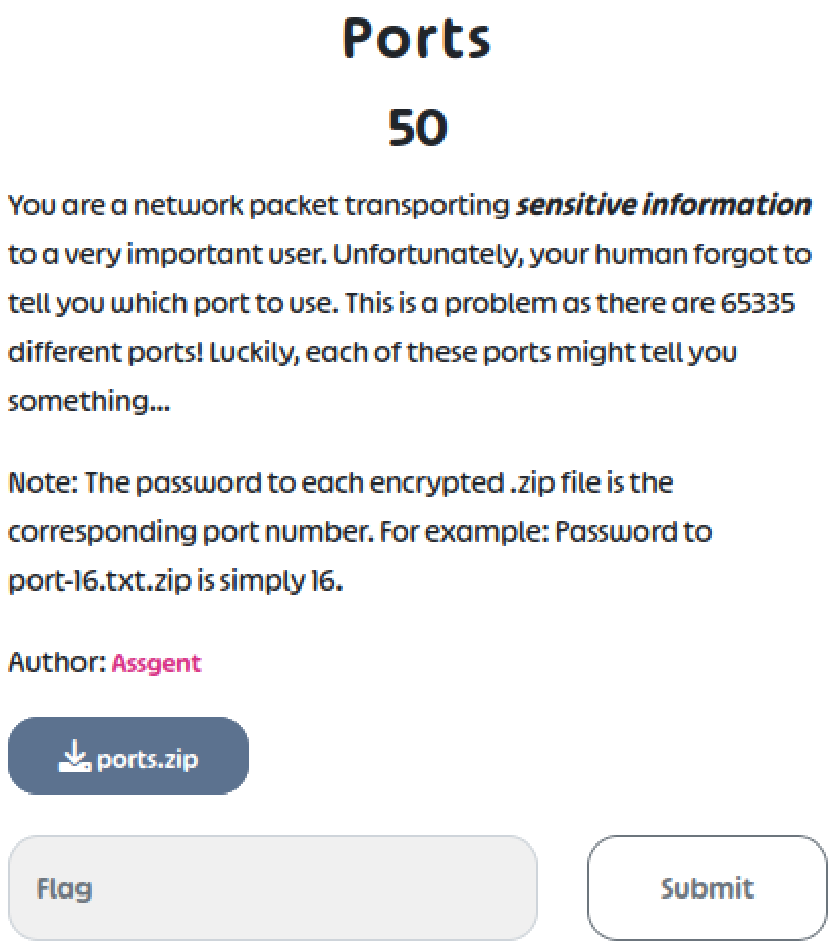
The link does not work. However, I looked into what `A, AAAA, MX, TXT` are and realized that these are DNS record types. We can access `TXT` in a number of ways including `nslookup`. Hence, I just used it and got the flag:



```
(base) hamza@hamza-work:~/Desktop/Computer_Security/CTF/CTF2/misc/ports/ports$ nslookup -type=txt a-txt-f
or-you-and-me.chall.lol
Server:         127.0.0.53
Address:        127.0.0.53#53

Non-authoritative answer:
a-txt-for-you-and-me.chall.lol  text = "UMDCTF{just_old_school_texting}"
```

(c) **ports**

Here is the screenshot for the ports challenge.



Ports

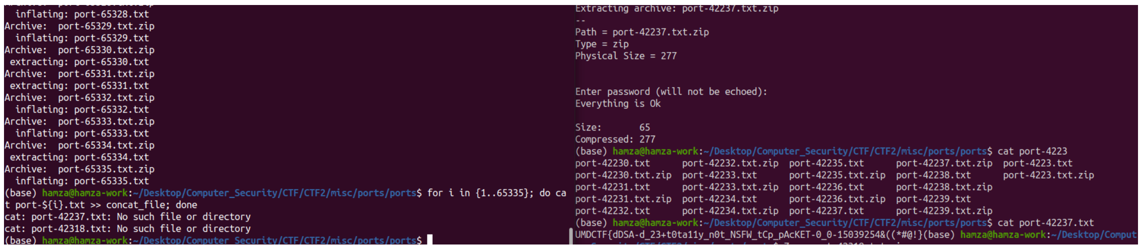
50

You are a network packet transporting ***sensitive information*** to a very important user. Unfortunately, your human forgot to tell you which port to use. This is a problem as there are 65335 different ports! Luckily, each of these ports might tell you something...

Note: The password to each encrypted .zip file is the corresponding port number. For example: Password to port-16.txt.zip is simply 16.

Author: Assgent

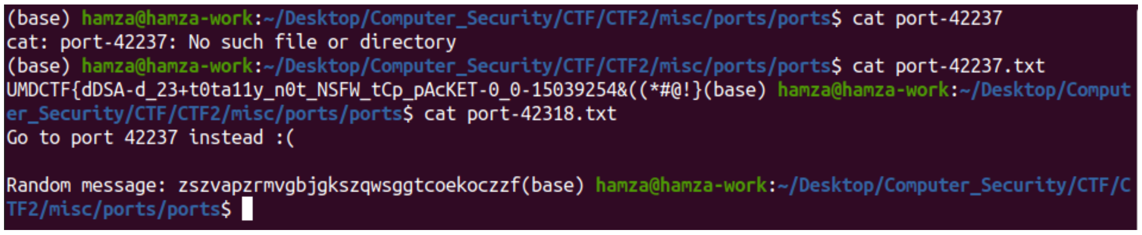
I extract all files in `ports.zip`. I then get another 65335 zip files, one for each port. I tried extracting each one of them using a bash script, however, two files do not get extracted, which I didn't immediately know. But when I tried concatenating all 65335 text files to a single one, I got errors on two. Which can be seen on the left in the below screenshot.



```
  inflating: port-65328.txt
Archive:  port-65329.txt.zip
  inflating: port-65329.txt
Archive:  port-65330.txt.zip
 extracting: port-65330.txt
Archive:  port-65331.txt.zip
 extracting: port-65331.txt
Archive:  port-65332.txt.zip
  inflating: port-65332.txt
Archive:  port-65333.txt.zip
  inflating: port-65333.txt
Archive:  port-65334.txt.zip
 extracting: port-65334.txt
Archive:  port-65335.txt.zip
  inflating: port-65335.txt
(base) hamza@hamza-work:~/Desktop/Computer_Security/CTF/CTF2/misc/ports/ports$ for i in {1..65335}; do ca
t port-${i}.txt >> concat_file; done
cat: port-42237.txt: No such file or directory
cat: port-42318.txt: No such file or directory
(base) hamza@hamza-work:~/Desktop/Computer_Security/CTF/CTF2/misc/ports/ports$
```

```
Extracting archive: port-42237.txt.zip
--
Path = port-42237.txt.zip
Type = zip
Physical Size = 277


Enter password (will not be echoed):
Everything is Ok

Size:       65
Compressed: 277
(base) hamza@hamza-work:~/Desktop/Computer_Security/CTF/CTF2/misc/ports/ports$ cat port-4223
port-42230.txt      port-42232.txt.zip  port-42235.txt      port-42237.txt.zip  port-4223.txt
port-42230.txt.zip  port-42233.txt      port-42235.txt.zip  port-42238.txt      port-4223.txt.zip
port-42231.txt      port-42233.txt.zip  port-42236.txt      port-42238.txt.zip
port-42231.txt.zip  port-42234.txt      port-42236.txt.zip  port-42239.txt
port-42232.txt      port-42234.txt.zip  port-42237.txt      port-42239.txt.zip
(base) hamza@hamza-work:~/Desktop/Computer_Security/CTF/CTF2/misc/ports/ports$ cat port-42237.txt
UMDCTF{dDSA-d_23+t0ta11y_n0t_NSFW_tCp_pAcKET-0_0-15039254&((*#@!}(base) hamza@hamza-work:~/Desktop/Comput
```

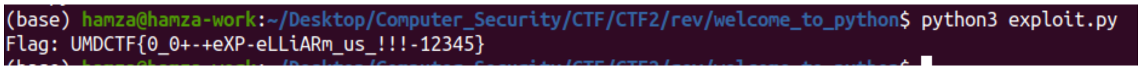
On the right, I extract one of the files using another unzipper which succeeds and gives the flag. Also, the contents of the two files are given here, one contains the flag, other points to the flag file.



```
(base) hamza@hamza-work:~/Desktop/Computer_Security/CTF/CTF2/misc/ports/ports$ cat port-42237
cat: port-42237: No such file or directory
(base) hamza@hamza-work:~/Desktop/Computer_Security/CTF/CTF2/misc/ports/ports$ cat port-42237.txt
UMDCTF{dDSA-d_23+t0ta11y_n0t_NSFW_tCp_pAcKET-0_0-15039254&((*#@!}(base) hamza@hamza-work:~/Desktop/Comput
er_Security/CTF/CTF2/misc/ports/ports$ cat port-42318.txt
Go to port 42237 instead :(

Random message: zszvapzrmvgbjgkszqwsggtcoekoczzf(base) hamza@hamza-work:~/Desktop/Computer_Security/CTF/C
TF2/misc/ports/ports$
```

5. **rev**:

   (a) **Welcome to Python** The challenge says that a given executable file is compiled from a Python script. We need to decompile it. I checked for a lot of tools to decompile executable to python code but many had issues with python version compatibility. Finally, I found this article that helped me decompile [4]. First, I extracted the dump section from the file, that is in `pydata.dump` using `objcopy`. Then, created a compiled copy `chal.pyc` from the dump using `pyinstxtractor` [3]. Finally, used `pycdc` [2] to decompile into `decompiled.py`. However, this does not give the flag as it is, hence I create an exploit in `exploit.py` that constructs the flag character by character.



```
(base) hamza@hamza-work:~/Desktop/Computer_Security/CTF/CTF2/rev/welcome_to_python$ python3 exploit.py
Flag: UMDCTF{0_0+-+eXP-eLLiARm_us_!!!-12345}
```

References

[1] *CBC-MAC*. URL: `https://en.wikipedia.org/wiki/CBC-MAC`.

[2] *pycdc*. URL: `https://github.com/zrax/pycdc`.

[3] *pyinstxtractor*. URL: `https://github.com/extremecoders-re/pyinstxtractor`.

[4] *Unpacking Python Executables on Windows and Linux*. URL: `https://www.fortinet.com/blog/threat-research/unpacking-python-executables-windows-linux`.