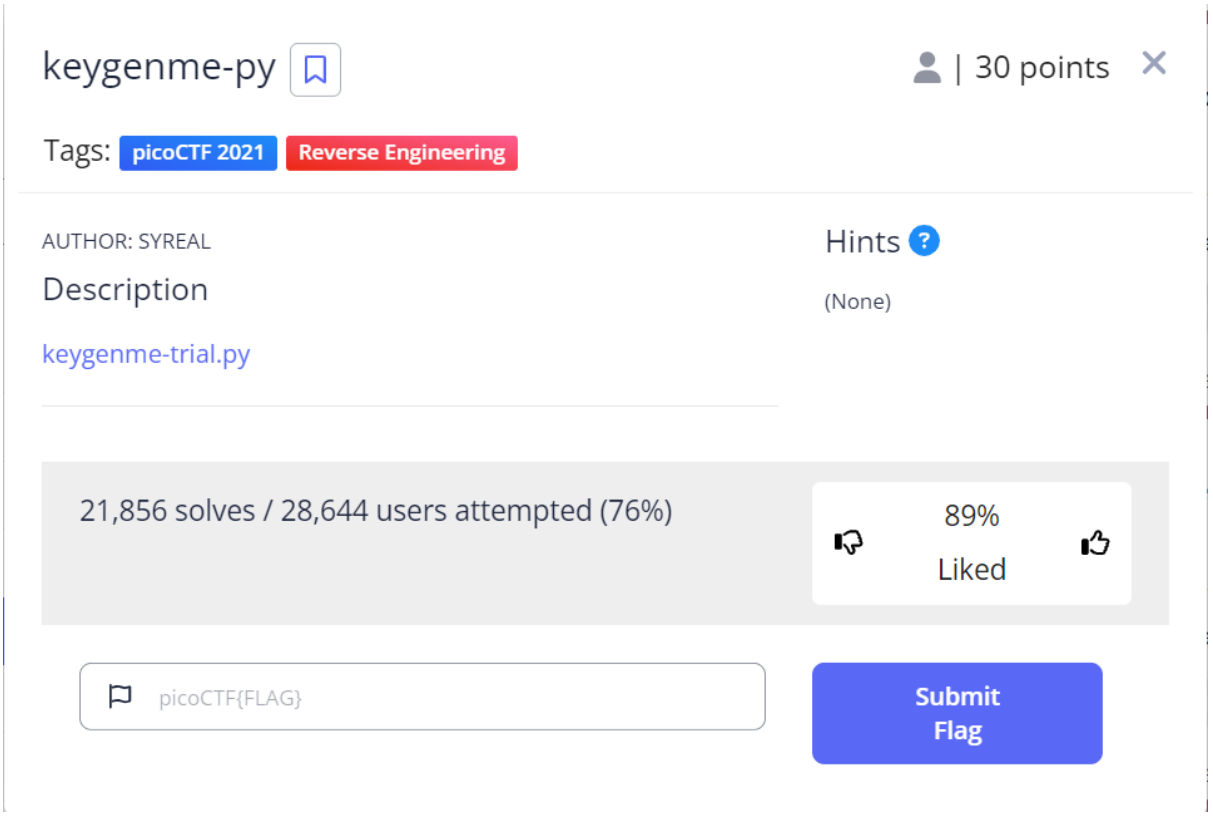
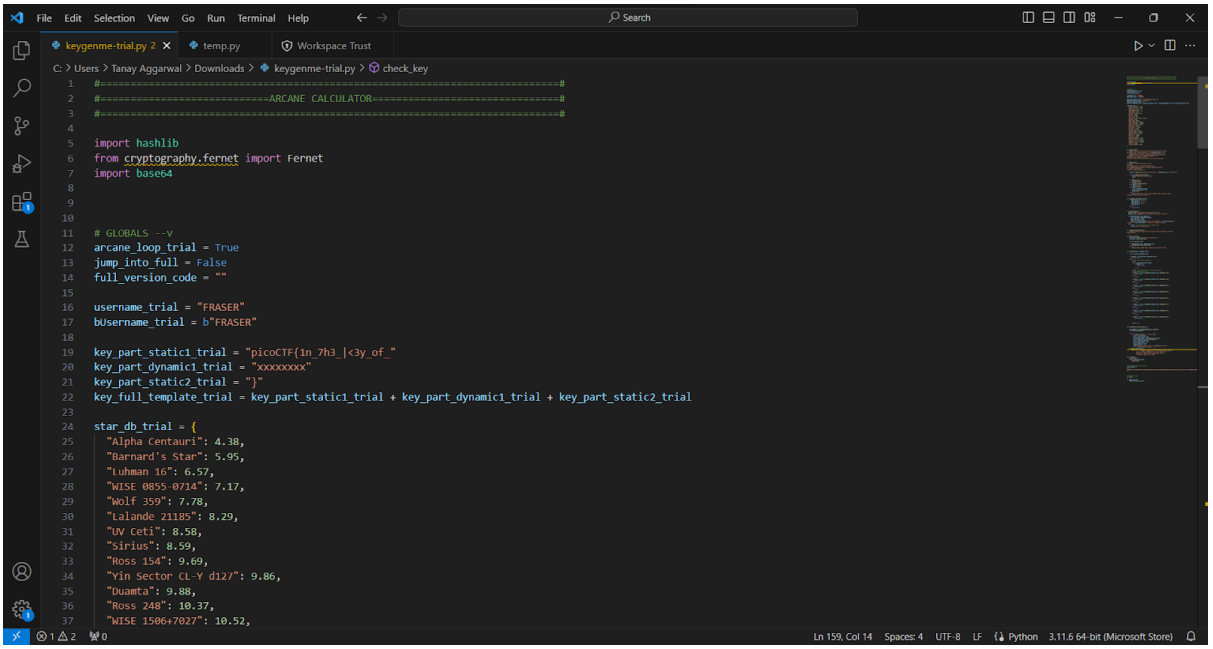
Keygenme-py

We download the given python script:



This is what it looks like:

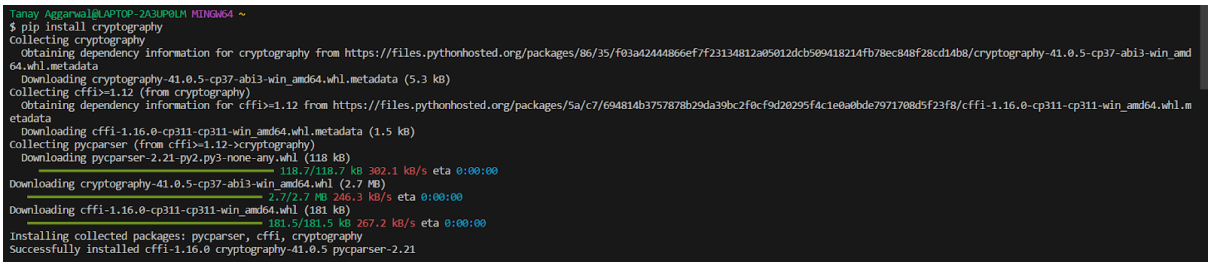


(an ss of the script from VSCode)

To run the program, we first need to install the cryptography package.

We do so using the pip install command.

pip is a package manager for python that allows us to import specific packages or modules into our code. A module can be thought of as a function in a program while a package can be thought as a collection of modules.



Now the cryptography package is installed.

I ran the program, and it displayed various options. Testing each one did not give any meaningful output.

For option (a)

```
Menu:
(a) Estimate Astral Projection Mana Burn
(b) [LOCKED] Estimate Astral Slingshot Approach Vector
(c) Enter License Key
(d) Exit Arcane Calculator
What would you like to do, FRASER (a/b/c/d)? a


SOL is detected as your nearest star.
To which system do you want to travel? Alpha Centauri

Alpha Centauri will cost between 19.1844 and 84.027672 stone(s) to project to
```

Option (b)

```
___Arcane Calculator___

Menu:
(a) Estimate Astral Projection Mana Burn
(b) [LOCKED] Estimate Astral Slingshot Approach Vector
(c) Enter License Key
(d) Exit Arcane Calculator
What would you like to do, FRASER (a/b/c/d)? b


You must buy the full version of this software to use this feature!
```

Option (c):

```
Menu:
(a) Estimate Astral Projection Mana Burn
(b) [LOCKED] Estimate Astral Slingshot Approach Vector
(c) Enter License Key
(d) Exit Arcane Calculator
What would you like to do, FRASER (a/b/c/d)? c

Enter your license key: a

Key is NOT VALID. Check your data entry.
```

Option (d) caused me to exit the output. (as expected)

My first line of thought was that there must be a licence key hidden somewhere in the script, which would enable me to access option (b). So I tried looking for it, but it was a false lead.

As it turned out, I did not need a licence key. This clicked when I looked closely at the start of the code.

```
username_trial = "FRASER"
bUsername_trial = b"FRASER"

key_part_static1_trial = "picoCTF{1n_7h3_|<3y_of_"
key_part_dynamic1_trial = "xxxxxxxx"
key_part_static2_trial = "}"
key_full_template_trial = key_part_static1_trial + key_part_dynamic1_trial + key_part_static2_trial
```

As we see, the flag is made of 3 parts. It is the second part, ‘key_part_dynamic1_trial’ that we need to determine to complete the flag.

Further down in the program:

```
if len(key) != len(key_full_template_trial):
    return False
else:
    # Check static base key part --v
    i = 0
    for c in key_part_static1_trial:
        if key[i] != c:
            return False

        i += 1

    # TODO : test performance on toolbox container
    # Check dynamic part --v
    if key[i] != hashlib.sha256(username_trial).hexdigest()[4]:
        return False
```

The for loop is meant to increment the value of ‘i’ , which is clearly indicating the index of a certain character in the key.

There are also several if-else conditions that follow, which are comparing the values of the

character at the ‘i’ th index of the key to the actual key that must be the flag.

```
if key[i] != hashlib.sha256(username_trial).hexdigest()[5]:
    return False
else:
    i += 1

if key[i] != hashlib.sha256(username_trial).hexdigest()[3]:
    return False
else:
    i += 1

if key[i] != hashlib.sha256(username_trial).hexdigest()[6]:
    return False
else:
    i += 1

if key[i] != hashlib.sha256(username_trial).hexdigest()[2]:
    return False
else:
    i += 1

if key[i] != hashlib.sha256(username_trial).hexdigest()[7]:
    return False
else:
    i += 1

if key[i] != hashlib.sha256(username_trial).hexdigest()[1]:
    return False
else:
    i += 1

if key[i] != hashlib.sha256(username_trial).hexdigest()[8]:
    return False
```

So we can now reverse engineer the characters from these conditions.

We see the following code in each condition:

```
key[i] != hashlib.sha256(username_trial).hexdigest()[
```

Here is the meaning of the various terms involved in this line:

hashlib: It is a library containing various hash functions to form the hash (non-decryptable form) of plaintext.

sha256(<bstring>): This produces a 256 bit hash of the binary string passed as a function to it. It uses 32 bit words. sha performs various mathematical operations on a bytes string, making it practically impossible to brute-force it back to plaintext. It is part of the SHA-2 family.

hexdigest(): It converts a hash digest into a hexadecimal string. It consists of characters from 0-9 and A to F.

In our case, the value of username_trial is “FRASER”, which we are supposed to pass as a bytes string to the above function. A byte string is a string that isn’t human readable. For example, an image when stored as a .jpg file is encoded in a certain manner and stored in the memory of the computer. Similarly, the system stores this string in such a manner.

We now use the same function in a different program to get the characters one by one.

```
if key[i] != hashlib.sha256(username_trial).hexdigest()[4]:
    return False
else:
    i += 1

if key[i] != hashlib.sha256(username_trial).hexdigest()[5]:
    return False
else:
    i += 1

if key[i] != hashlib.sha256(username_trial).hexdigest()[3]:
    return False
else:
    i += 1

if key[i] != hashlib.sha256(username_trial).hexdigest()[6]:
    return False
else:
    i += 1

if key[i] != hashlib.sha256(username_trial).hexdigest()[2]:
    return False
else:
    i += 1

if key[i] != hashlib.sha256(username_trial).hexdigest()[7]:
    return False
else:
    i += 1

if key[i] != hashlib.sha256(username_trial).hexdigest()[1]:
    return False
else:
    i += 1

if key[i] != hashlib.sha256(username_trial).hexdigest()[8]:
    return False
```

The characters we need are at the [<integer>] index.

Here is the code we will be using to decode.

C: > Users > Tanay Aggarwal > Downloads > temp.py

```
1  import hashlib
2  i=int(input("Index: "))
3  print (hashlib.sha256(b"FRASER").hexdigest()[i])
```

We need the characters at indices: 4,5,3,6,2,7,1,8

```
$ "C:/Users/Tanay Aggarwal/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "c:/Users/Tanay Aggarwal/Downloads/temp.py"
Index: 4
a

Tanay Aggarwal@LAPTOP-2A3UP0LM MINGW64 ~
$ "C:/Users/Tanay Aggarwal/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "c:/Users/Tanay Aggarwal/Downloads/temp.py"
Index: 4
a

Tanay Aggarwal@LAPTOP-2A3UP0LM MINGW64 ~
$ "C:/Users/Tanay Aggarwal/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "c:/Users/Tanay Aggarwal/Downloads/temp.py"
Index: 5
c

Tanay Aggarwal@LAPTOP-2A3UP0LM MINGW64 ~
$ "C:/Users/Tanay Aggarwal/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "c:/Users/Tanay Aggarwal/Downloads/temp.py"
Index: 3
7

Tanay Aggarwal@LAPTOP-2A3UP0LM MINGW64 ~
$ "C:/Users/Tanay Aggarwal/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "c:/Users/Tanay Aggarwal/Downloads/temp.py"
Index: 6
3

Tanay Aggarwal@LAPTOP-2A3UP0LM MINGW64 ~
$ "C:/Users/Tanay Aggarwal/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "c:/Users/Tanay Aggarwal/Downloads/temp.py"
Index: 2
d

Tanay Aggarwal@LAPTOP-2A3UP0LM MINGW64 ~
$ "C:/Users/Tanay Aggarwal/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "c:/Users/Tanay Aggarwal/Downloads/temp.py"
Index: 7
c

Tanay Aggarwal@LAPTOP-2A3UP0LM MINGW64 ~
$ "C:/Users/Tanay Aggarwal/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "c:/Users/Tanay Aggarwal/Downloads/temp.py"
Index: 1
2

Tanay Aggarwal@LAPTOP-2A3UP0LM MINGW64 ~
$ "C:/Users/Tanay Aggarwal/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "c:/Users/Tanay Aggarwal/Downloads/temp.py"
Index: 8
9
```

Characters are:

ac73dc29

This completes our flag:

picoCTF{1n_7h3_|<3y_of_ac73dc29}

Upon passing this,

This completes this level.