Challenge: Fast Carmichael

Challenge Description :

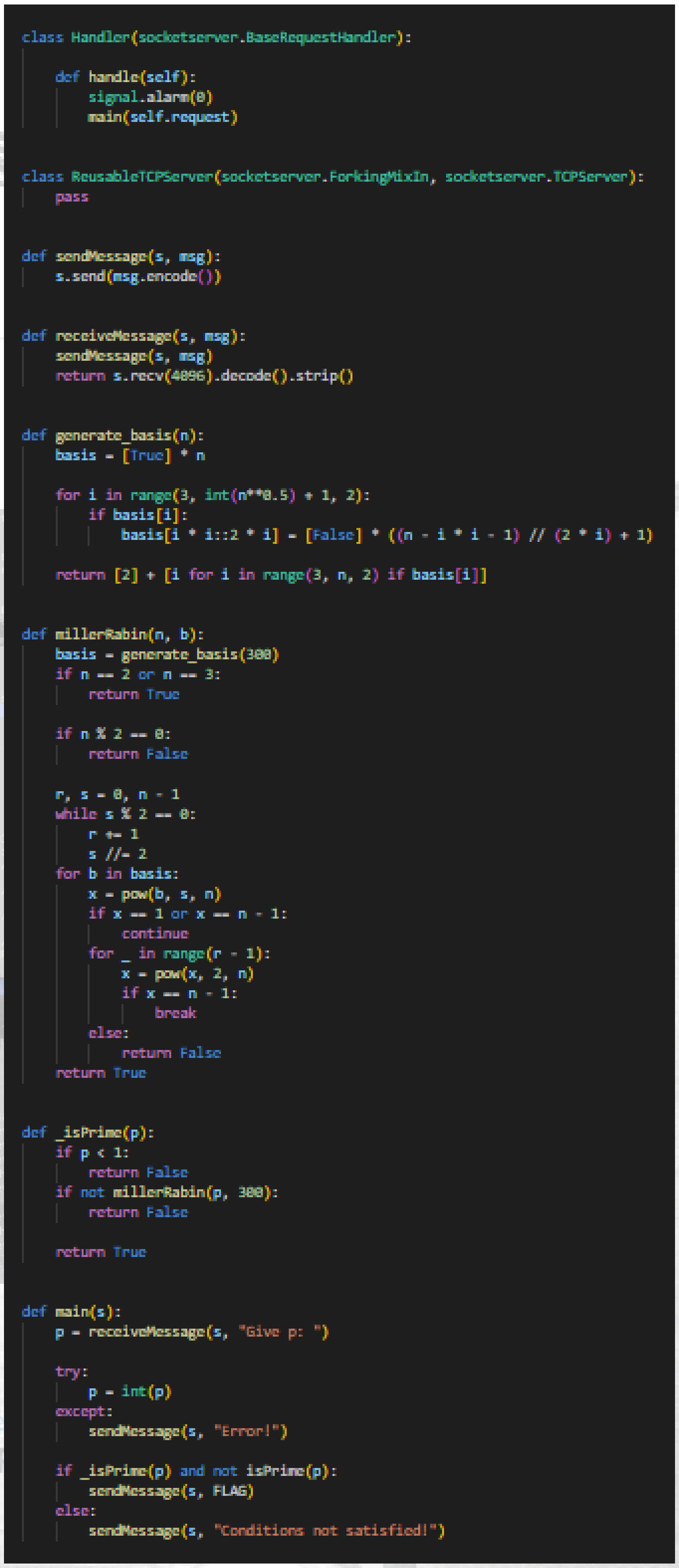
You are walking with your friends in search of sweets and discover a mansion in the distance. All your friends are too scared to approach the building, so you go on alone. As you walk down the street, you see expensive cars and math papers all over the yard. Finally, you reach the door. The doorbell says &quot;Michael Fastcar&quot;. You recognize the name immediately because it was on the news the day before. Apparently, Fastcar is a famous math professor who wants to get everything done as quickly as possible. He has even developed his own method to quickly check if a number is a prime. The only way to get candy from him is to pass his challenge.

Context :

- You are given a .EML file, a weird plain text format for emails, basically just analyze it, open the file up and get both base64 encoded strings. From there analyze the code you are given and find the flag.

Flag :

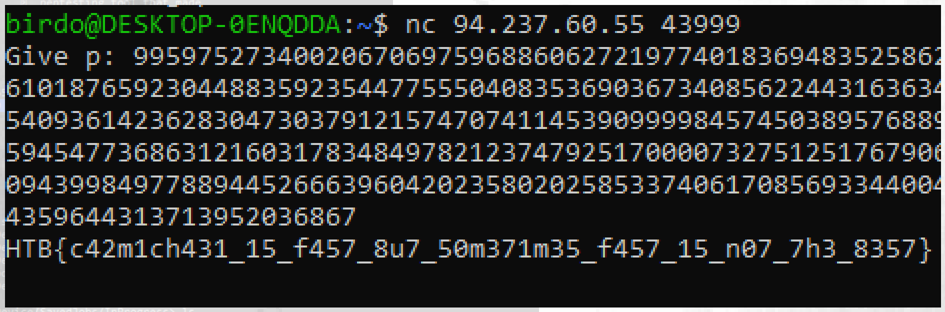
- First, install the necessary files and open them with a basic text editor, such as Notepad.
- The provided Python server code includes a custom implementation to check whether a given number is prime using the Miller-Rabin algorithm with some additional conditions. Our goal is to exploit this to retrieve the flag from the server.
- The server prompts the client to provide a number p and checks if p satisfies the custom _isPrime function but fails the standard isPrime check from the Crypto.Util.number module. If these conditions are met, the server sends the flag; otherwise, it responds with an error message.
- The _isPrime function incorporates multiple conditions, one of which involves testing the input against a generated basis of prime numbers to determine if
- p is likely prime. Certain numbers, known as pseudoprimes, can fool specific primality tests, including the Miller-Rabin test. Our task is to find a pseudoprime that passes the server's _isPrime check but fails the standard isPrime test.



```python
class Handler(socketserver.BaseRequestHandler):

    def handle(self):
        signal.alarm(0)
        main(self.request)


class ReusableTCPServer(socketserver.ForkingMixIn, socketserver.TCPServer):
    pass


def sendMessage(s, msg):
    s.send(msg.encode())


def receiveMessage(s, msg):
    sendMessage(s, msg)
    return s.recv(4096).decode().strip()


def generate_basis(n):
    basis = [True] * n

    for i in range(3, int(n**0.5) + 1, 2):
        if basis[i]:
            basis[i * i::2 * i] = [False] * ((n - i * i - 1) // (2 * i) + 1)

    return [2] + [i for i in range(3, n, 2) if basis[i]]


def millerRabin(n, b):
    basis = generate_basis(300)
    if n == 2 or n == 3:
        return True

    if n % 2 == 0:
        return False

    r, s = 0, n - 1
    while s % 2 == 0:
        r += 1
        s //= 2
    for b in basis:
        x = pow(b, s, n)
        if x == 1 or x == n - 1:
            continue
        for _ in range(r - 1):
            x = pow(x, 2, n)
            if x == n - 1:
                break
        else:
            return False
    return True


def _isPrime(p):
    if p < 1:
        return False
    if not millerRabin(p, 300):
        return False

    return True


def main(s):
    p = receiveMessage(s, "Give p: ")

    try:
        p = int(p)
    except:
        sendMessage(s, "Error!")

    if _isPrime(p) and not isPrime(p):
        sendMessage(s, FLAG)
    else:
        sendMessage(s, "Conditions not satisfied!")
```

- To exploit the server, we connect to it using netcat (nc). Once connected, we input a carefully selected pseudoprime that meets the custom conditions set by the server while being recognized as non-prime by the standard isPrime function.

- To generate a pseudoprime number I recommend researching it more, as it was a bit weird trying to find some logic for it. This is what i found to be useful :
    - https://github.com/awslabs/fast-pseudoprimes/
    - https://gist.github.com/keltecc/b5fbd533d2f203e810b43c26ff9d17cc < Contains a pseudoprime example.
    - https://www.semanticscholar.org/paper/Breaking-a-Cryptographic-Protocol-with-Pseudoprimes-Bleichenbacher/e9f1f083adc1786466d344db5b3d85f4c268b429?p2df < Mostly for learning and reading more about it.



```
birdo@DESKTOP-0ENQDDA:~$ nc 94.237.60.55 43999
Give p: 99597527340020670697596886062721977401836948352586
6101876592304488359235447755504083536903673408562244316363
5409361423628304730379121574707411453909999845745038957688
5945477368631216031783484978212374792517000073275125176790
0943998497788944526663960420235802025853374061708569334400
43596443137139520368 67
HTB{c42m1ch431_15_f457_8u7_50m371m35_f457_15_n07_7h3_8357}
```

- The flag, after successfully finding the correct pseudoprime, should be:

    HTB{c42m1ch431_15_f457_8u7_50m371m35_f457_15_n07_7h3_8357}