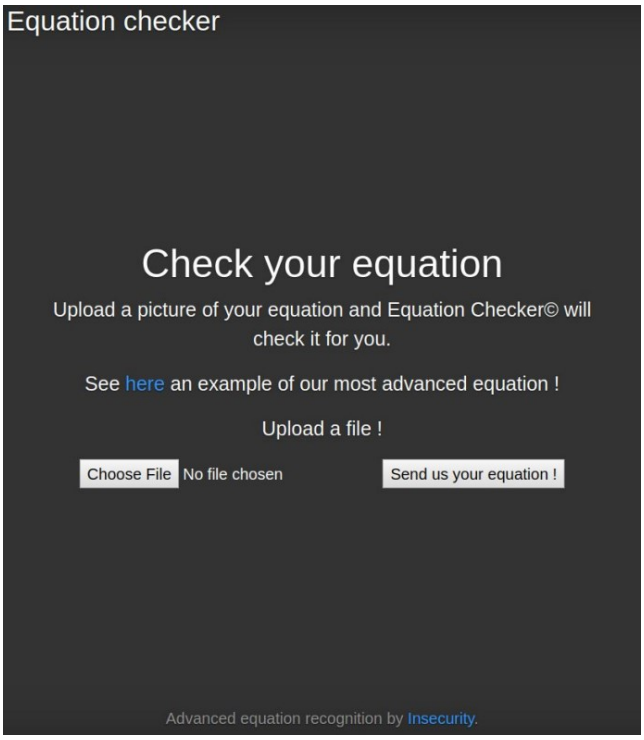
Soluzione

Abbiamo la seguente pagina:

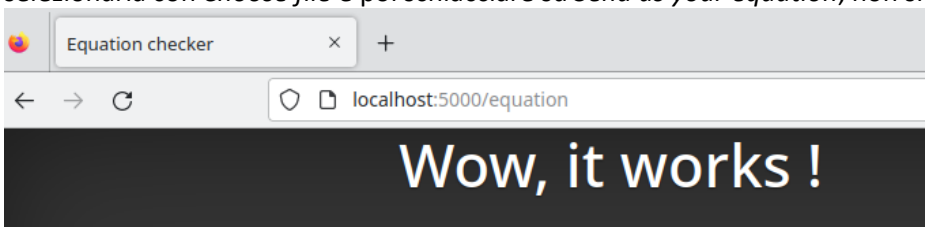


Se facciamo clic su "here", vediamo un esempio di equazione. L'APP riceve in input un'immagine un'immagine contenente un'equazione e calcola il risultato. Possiamo ispezionare la pagina web e ottenere subito un'idea:

```
  <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity
</body>
<!-- TODO : Remove me : -->
<!-- /debug-->

</html>
```

Di per sé, la pagina non fa molto; il pulsante "Send us your equation" non funziona da solo e dal link cliccabile come *here* si ha un .png di un'equazione $2 + 2 = 4$; anche provando a salvarsela come immagine, selezionarla con *Choose file* e poi schiacciare su *Send us your equation*, non si ha un grande risultato.



Si capisce quindi che l'approccio da seguire deve essere un altro; si nota dal commento nel codice precedente *debug*, segnato però come link (*/debug*); forse si può provare ad entrarci.

Andiamo al link *http://127.0.0.1:5000/debug* : la pagina contiene il codice sorgente che esegue l'applicazione (in pratica scarica il codice Python del server).

Come si vede a lato, accetta come configurazione un contenuto con una lunghezza massima e, al metodo GET, reindirizza una pagina *index.html*; si nota però che esiste una variabile *x* che apre un file *flag.txt*; direi che ci può interessare.

```
#!/usr/bin/python3
from flask import Flask,request,send_from_directory,render_template,abort
import pytesseract
from PIL import Image
from re import sub
from io import BytesIO
app = Flask(__name__)
app.config.update(
    MAX_CONTENT_LENGTH = 500 * 1024
)
x = open("private/flag.txt").read()


@app.route('/',methods=['GET'])
def ind():
    return render_template("index.html")

@app.route('/debug',methods=['GET'])
def debug():
    return send_from_directory('./', "server.py")
```

Il resto del codice, alla POST, non fa altro che fare dei controlli sui file caricati (non ne è stato caricato nessuno oppure file caricato vuoto), poi usa la libreria *pytesseract* per aprire l'immagine di input, convertirla in byte, e formattare il testo aggiungendo un "=" ogni split di riga, rimpiazzando "=" con "==" come padding e con la funziona *sub* ricerca come pattern "===+", lo rimpiazza con "==" sulla base della stringa *formated_text*. Successivamente controlla che il testo sia nell'alfabeto, oppure se ci sono delle parentesi, se nel testo ci sono librerie/pezzi di codice specifici o se si abbia una lunghezza troppo lunga.

```
oute('/equation',methods=['POST'])
uation():
 'file' not in request.files:
  return render_template('result.html', result = "No file uploaded")
le = request.files['file']
int(file)
 file and file.filename == '':
  return render_template('result.html', result = "No correct file uploaded")
 file:
  input_text = pytesseract.image_to_string(Image.open(BytesIO(file.read())))
  print(input_text)
  formated_text = "=".join(input_text.split("\n"))
  formated_text = formated_text.replace("=","==")
  formated_text = sub('===+','==',formated_text)
  formated_text = formated_text.replace(" ","")
  print(formated_text)
  if any(i not in 'abcdefghijklmnopqrstuvwxyz0123456789()[]=+-*' for i in formated_text):
      return render_template('result.html', result = "Some features are still in beta !")
  if formated_text.count('(') > 1 or formated_text.count(')') > 1 or formated_text.count('[') > 1
      return render_template('result.html', result = "We can not solve complex equations for now !
  if any(i in formated_text for i in ["import","exec","compile","tesseract","chr","os","write","sl
      return render_template('result.html', result = "We can not understand your equation !")
  if len(formated_text) > 15:
      return render_template('result.html', result = "We can not solve complex equations for now !
```

Da tenere d'occhio la successiva funzione *eval* e al blocco di codice che controlla il padding "==" sia presente; se questo capita, considera le due parti come interi e, se uguali, ritorna "Wow, it works!", altrimenti afferma che non sono uguali. Se non appare il padding, allora chiede di inserire una equazione valida.

```
        try:
            if "==" in formated_text:
                parts = formated_text.split("==",maxsplit=2)
                pa_1 = int(eval(parts[0]))
                pa_2 = int(eval(parts[1]))
                if pa_1 == pa_2:
                    return render_template('result.html', result = "Wow, it works !")
                else:
                    return render_template('result.html', result = "Sorry but it seems that %d is
            else:
                return render_template('result.html', result = "Please import a valid equation !")
        except (KeyboardInterrupt, SystemExit):
            raise
        except:
            return render_template('result.html', result = "Something went wrong...")

@app.route('/is/<path:path>')
```

Dobbiamo infatti concentrarci sulla funzione *eval*, che è una funzione ben conosciuta che permette codici simili alle iniezioni. Possiamo considerare un esempio semplice di funzionamento:

1 + 1 = 3 - 1

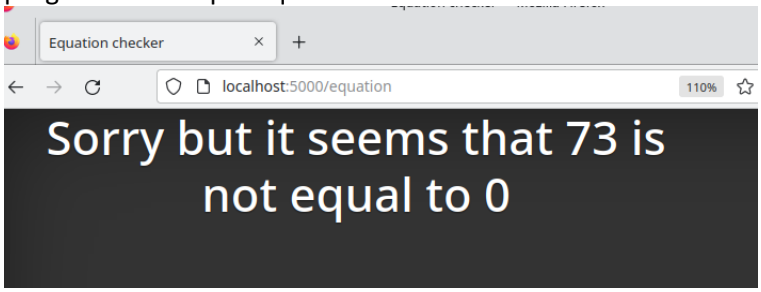
Il codice divide i due lati, *(1 + 1, 3 - 1)* e poi esegue ciò che è contenuto all'interno di *(2,2).*
Questi due numeri sono uguali, ottimo. Come si può notare, l'immagine dell'equazione $2 + 2 = 4$ non è in effetti a caso; sono due numeri uguali e dice proprio "Wow, it works".

Viene suggerito di dare un'occhiata al blog:
https://medium.com/swlh/hacking-python-applications-5d4cd541b3f1
Si può sfruttare la funzione iniettando informazioni dannose, ad esempio, possiamo creare le seguenti due immagini per dedurre i primi due caratteri della bandiera:

ord(x[0]) = 0

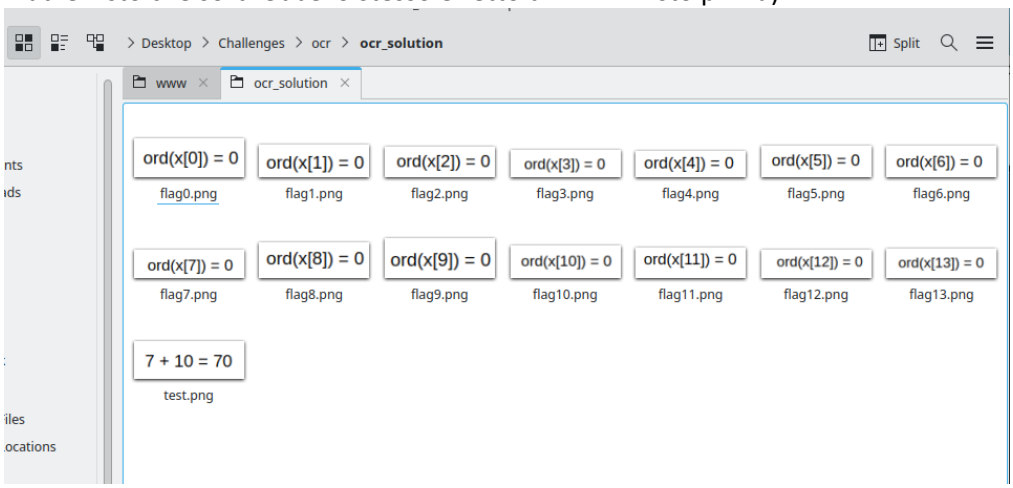
ord(x[1]) = 0

In questo modo si stamperà il valore numerico del carattere 0 nel flag; poiché non sarà uguale a 0, il programma stamperà quel contenuto!



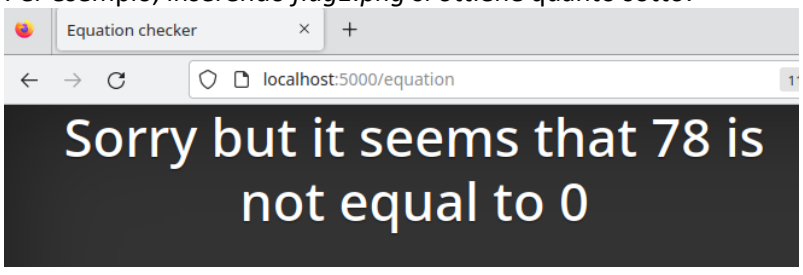
Se guardiamo bene il codice, notiamo che bisogna iniettare un testo almeno 14 volte (infatti, la condizione `if len(formated_text) > 15` fa capire esattamente questo. Qua viene l'idea dal nome OCR (riconoscimento ottico dei caratteri); in poche parole, prendiamo una serie di immagini, convertite ad intero e aggiungiamo ad "x", sfruttando la vulnerabilità della funzionalità *eval*).

Dobbiamo solo creare diverse immagini "ad hoc" per tutta la lunghezza della bandiera, testando ogni singolo carattere (si completa anche con un'immagine di test, comunque inutile visto che sortirebbe lo stesso effetto di 2+2=4 visto prima):



I log di errore come quello di sopra non sono casuali; seguono infatti questa logica a lato.
Riferimento dell'immagine da:
https://0fa.ch/writeups/web/2018/04/06/INSHACK2018-ocr.html

Per esempio, inserendo *flag1.png* si ottiene quanto sotto:



Inserendo uno ad uno i caratteri come sopra e prendendo la corrispondente posizione, otteniamo quindi la flag:
INSA{0cr_L0ng}