1) 50 Slash Slash: Testo, aiuti e soluzione

Testo

You own the application.
Free to use any resource you are given (e.g., you can have a look at the files contained in the 7z file).

In questo contesto, dettaglierò tutti i passi a differenza di chi dovrebbe farlo, ma non lo fa:

- Unzippare tutta la cartella in formato 7z
- Dentro, seguire il link: https://www.youtube.com/watch?v=N5vscPTWKOk&list=PL-osiE80TeTt66h8cVpmbayBKlMTuS55y&index=7 per imparare ad usare i *virtual environment* di Python. Un ambiente virtuale in Python è uno strumento che aiuta a mantenere separate le dipendenze richieste da diversi progetti creando per loro ambienti virtuali isolati.
- Le applicazioni sviluppate con Python useranno spesso pacchetti e moduli che non fanno parte della libreria standard. A volte le applicazioni necessitano di una versione specifica di una libreria, poiché l'applicazione potrebbe richiedere la correzione di un determinato bug o l'applicazione potrebbe essere scritta utilizzando una versione obsoleta dell'interfaccia della libreria.
  Ciò significa che potrebbe non essere possibile per un'installazione Python soddisfare i requisiti di ogni applicazione. Se l'applicazione A richiede la versione 1.0 di un particolare modulo ma l'applicazione B richiede la versione 2.0, i requisiti sono in conflitto e l'installazione della versione 1.0 o 2.0 non consente l'esecuzione di un'applicazione.
  La soluzione a questo problema è quella di creare un ambiente virtuale, un albero di directory autonomo che contiene un'installazione Python per una particolare versione di Python, oltre a una serie di pacchetti aggiuntivi.
- Quindi, occorre (i primi tre passaggi da saltare se tutto installato):
  - Installare Python → *sudo apt install python3* (per Arch: sudo pacman -S python-pip)
  - Installare PIP → *sudo apt install python3-pip* (per Arch: *sudo pacman -S python)*
  - Installare Virtualenv → *sudo pip install virtualenv* (per Arch: *sudo pacman -S python-virtualenv*)
  - Installare Flask → *sudo pip install flask*
  - Spostarsi all'interno della cartella *app* una volta unzippata la cartella
  - Attivare l'environment presente → *source env/bin/activate*
  - Installare i pacchetti con *pip* inclusi nel file *requirements* → *pip install -r requirements.txt*
  - Una volta finito di usare il virtualenv, si usa *deactivate* per uscire

Aiuti:

1) If you run the application with `python application.py` you see a meme telling you are in the wrong path. If you inspect the python code, you can see that the flag is contained inside a virtual environment call. Let's go to inspect the activation file of the environment, which is contained at "./env/bin/activate". There, you might find the FLAG.

Attenzione

Potrebbe esserci un errore del tipo:
Cannot import name Container from collections

Ciò è dovuto, nel caso si usi Python 3.10, all'uso di un pezzo deprecato all'interno della libreria.
Sarà quindi necessario Python 3.9.

Su Arch Linux si fa così:

- Installare *yay* → https://www.lffl.org/2021/01/guida-yay-arch-linux-manjaro.html
- Installare Python 3.9 → *yay python39*

Su Ubuntu si fa così:
sudo apt update
sudo apt install software-properties-common
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt install python3.9

Installare *pip* in versione 3.9
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
python3.9 get-pip.py
Usare i comandi per avviare l'esercizio come segue:

- *python3.9 -m pip install -r requirements.txt*
- *python3.9 application.py*

Soluzione

Ci viene fornito un file "7Z". La flag è contenuto all'interno, quindi dobbiamo guardarlo all'interno. Se decomprimiamo la cartella, otteniamo una cartella chiamata "app". Se ispezioniamo il suo interno, possiamo vedere un file chiamato "application.py", un'applicazione Flask (uno dei pacchetti installati con Pip). Possiamo ad esempio eseguirla; apriamo un terminale e digitiamo: *python application.py*

La flag non c'è, quindi possiamo ispezionare il codice python.
Come si vede, all'interno dello script *application.py* si trova una ruote che indica una flag che inizia con *encryptCTF*:

```
https://www.youtube.com/watch?v=N5vscPTWKOk&l
'''
FLAG = os.getenv("FLAG", "encryptCTF{}")

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/encryptCTF', methods=["GET"])
def getflag():
        return jsonify({
            'flag': FLAG
        })

if __name__ == '__main__':
    app.run(debug=False)
```

Sembra che la flag sia contenuta all'interno di un ambiente virtuale chiamato *env*.
Proviamo a dare un'occhiata ai singoli file presenti in questa cartella.
Andiamo a ispezionare il file di attivazione dell'ambiente virtuale, che è contenuto in "./env/bin/activate".

L'ultima linea del file ha il seguente pezzo:

export $(echo RkxBRwo= | base64 -d)="ZW5jcnlwdENURntjb21tZW50c18mX2luZGVudGF0aW9uc19tYWtlc19qb2hubnlfYV9nb29k X3Byb2dyYW1tZXJ9Cg==”

Possiamo decrittare il messaggio direttamente sul terminale, scrivendo:
echo "ZW5jcnlwdENURntjb21tZW50c18mX2luZGVudGF0aW9uc19tYWtlc19qb2hubnlfYV9nb29kX3Byb2dyYW1tZXJ9Cg==" | base64 -d

In output otteniamo:
encryptCTF{comments_&_indentations_makes_johnny_a_good_programmer}

2) Python: Testo, aiuti e soluzione

Qui la situazione cambia; abbiamo un file Python e dei file Docker, assieme al file di configurazione Dockerfile.
Quindi, per avviare l'esercizio (spostandosi (cd) in preventivo nella cartella che contiene tutto *python*):

- *chmod -R +rx ./*
- In un'altra finestra di terminale (diversa da quella dei successivi comandi) → *sudo dockerd*
  In questo modo, si attiva il daemon di Docker. Deve rimanere aperta questa finestra.
  (Basta farlo una volta sola, resta attivo dopo)
- *sudo ./docker_build.sh*
- *sudo ./docker_run.sh*

Testo - Attenzione (Richiesta versione 3.9 di Python e seguire i passaggi che masochisticamente sono stati riportati)

We are given the following description: "This is my raspberry pi at home. I have some secrets that only me can received. Do you want to try?"

Aiuti:
1) This is a classic example of language vulnerability.

We need to insert an IP and a Port in order to do something, i.e., reach the flag. In addition, the app contains a source (see the link): if we open it, we can see a Python code. Let's copy this code in local and try to figure it out what it's doing.
But first, let's try to see the output of the program. If we insert random values, and we are going to receive a message that tells that it is mandatory to insert a specific IP and PORT.

So, let's try to use as an IP 8.8.8.8 and as port 8000.

"The flag has been sent". What does it mean?

Let's go and analyze the source. First of all, we need, with patience, to clean the code. After that we can understand the various if-else statements.

Let's start with the GET function, where three variables are involved:
● IP : the IP that we provide;
● Port: the port that we provide;
● flag : a string containing (?) the flag.
A dictionary called allowed contains a restriction on the IP number, which must be 8.8.8.8

2) You need to exploit the following line:

```
return ("You have chosen IP " + ip + ", but only %(allowed_ip)s will
receive the key\n") % allowed
```

The language vulnerability can be exploited using a smart "dictionary access"

Si avvia l'esercizio facendo le seguenti cose:

- Crearsi un virtualenv → *virtualenv project*

- Spostarsi nella cartella project → *cd project*
- Attivare l'environment presente → *source bin/activate*
- Spostarsi nella cartella *www*
- Eseguire *python3.9 -m pip install flask*
- Eseguire l'applicazione → *FLASK_APP=chall.py python3.9 -m flask run*
- Aprire un browser e andare all'indirizzo: http://127.0.0.1:5000/

Soluzione

Ci viene data questa pagina:

Complete the following field: source
IP: Port: Submit

Dobbiamo inserire un IP e una porta per poter fare qualcosa, cioè raggiungere la flag. Inoltre, l'applicazione contiene un *source* (si veda il link, quello della foto in viola): se lo apriamo, possiamo vedere un codice Python (il file *source.py*). Copiamo questo in locale e cerchiamo di capire cosa sta facendo.
Ma prima proviamo a vedere l'output del programma.
Immettendo un qualsiasi valore, riceveremo il seguente messaggio:

You have choose IP 79.51.180.84, but only 8.8.8.8 will receive the key

Proviamo quindi a utilizzare come IP 8.8.8.8 e come porta una generica, ad esempio 8000.

SUCCESS: The flag have been sent to DST IP 8.8.8.8 and DST PORT 8000

Andiamo ad analizzare il sorgente di *chall.py* (ad esempio con *cat chall.py* per leggerne il contenuto o con un editor di testo):

```
def get():
    if request.method == 'GET':
        ip = request.args.get('ip')
        port = request.args.get('port')


        flag = open('flag.txt').readline()
        allowed = {"allowed_ip": "8.8.8.8", "allowed_port": port, "allowed_flag": flag}
        if ip and ip != '' and port and port != '':
            if port.isdigit():
                if ip == allowed.get("allowed_ip"):
                    subprocess.Popen("cat flag.txt > /dev/tcp/" + str(ip) + "/" + str(port), shell=True,
                                                        executable="bash")
                    return ("SUCCESS: The flag have been sent to DST IP %s and DST PORT %s\n") % (ip, port)
                else:
                    return ("You have choose IP " + ip + ", but only %(allowed_ip)s will receive the key\n") % allowed
            else:
                return ("Port invalid\n")
        else:
            return ("Please choose an IP and a PORT\n")
    else:
        return ("FAIL: Method HTTP not allowed (%s)\n") % (request.method)
```

Cominciamo con la funzione GET, in cui sono coinvolte tre variabili:
● IP: l'IP che forniamo;
● Port: la porta che forniamo;
● flag: una stringa contenente (?) la flag.
Un dizionario chiamato *allowed* contiene una restrizione sul numero di IP, che deve essere 8.8.8.8.

Il primo IF controlla che IP e Porta non siano vuoti. In caso contrario, si controlla se la porta è un numero: in caso contrario, riceviamo un messaggio di errore.
In base a ciò, si ottiene che:
● IP = 8.8.8.8;
● Porta = numero a piacere
Quando si inserisce questa combinazione corretta, viene aperto un sottoprocesso contenente la flag e inviato all'indirizzo IP e alla porta indicati.
Tuttavia, dobbiamo concentrarci sull'istruzione else generata da un IP sbagliato. Come si può vedere, l'istruzione restituisce due variabili:
● Il nostro valore IP inserito;

● Il valore IP corretto (quello obbligatorio).

Con un po’ di attenzione, si può notare che la flag viene stampata in corrispondenza di “allowed_flag”. La stampa utilizza il formato "%(allowed_flag)s" per stampare il valore contenuto nel dizionario (questo serve a formattarlo come IP).

Usiamo questa informazione per stampare la flag come si vede qui:

Questo restituisce un messaggio (come prima), ma questa volta l'IP inserito ci mostra la flag:
You have choose IP INSA{Y0u_C@n_H@v3_fUN_W1Th_pYth0n}, but only 8.8.8.8 will receive the key