GABRIELE ROMANATO

Menu

Password casuali con apg su macOS con Docker, Python e Bash

L'utility da riga di comando apg per la generazione di password casuali al momento non è disponibile su macOS tramite il noto package manager Homebrew. Possiamo ovviare a questo problema usando una combinazione di un container Docker con Python e di un semplice script Bash.

apg è interessante perché consente anche la generazione di password robuste ma facili da ricordare (l'algoritmo o dell'opzione a è infatti quello predefinito).

Possiamo predisporre un container Docker con un'immagine Python basata su Debian su cui andremo ad installare l'utility apg e creeremo un'interfaccia web minimale in Python che restituisca l'output di tale comando. Iniziamo creando il file `Dockerfile`.

```
FROM python:bullseye

WORKDIR /usr/src/app

RUN apt-get update && apt-get install -y \
    libpq-dev \
    gcc \
    apg \
    && rm -rf /var/lib/apt/lists/*

COPY server.py /usr/src/app/

EXPOSE 9943

CMD ["python", "server.py"]
```

apg viene installato tramite APT poiché stiamo operando all'interno di un'istanza Debian. Possiamo quindi usare Docker Compose per rendere più agevole la gestione del container.

```
services:
  apgweb:
    build: .
    ports:
      - "9943:9943"
```

Il servizio è esposto sulla porta 9943 sia a livello della nostra macchina host che a livello di rete Docker. A questo punto possiamo creare il file `server.py`.

```
import subprocess
from http.server import BaseHTTPRequestHandler, HTTPServer

def execute_command(command):
    return subprocess.Popen(command, shell=True,
                             stdout=subprocess.PIPE, stderr=subprocess.PIPE).communicate()

class Server(HTTPServer):
    def __init__(self, address, request_handler):
        super().__init__(address, request_handler)

class RequestHandler(BaseHTTPRequestHandler):
    def __init__(self, request, client_address,
                 server_class):
        self.server_class = server_class
        super().__init__(request, client_address,
                         server_class)

    def do_GET(self):
        pass

    def do_POST(self):
        pass_length = int(self.headers['Content-Length'])
        apg_cmd = f"apg -a 1 -m {pass_length} -x"
```

```

{pass_length} -n 1"
        password, _ = execute_command(apg_cmd)
        response = password.decode('utf-8')
        self.send_response(200)
        self.send_header("Content-type", "text/plain")
        self.send_header("Content-Length",
str(len(response)))
        self.end_headers()
        self.wfile.write(str(response).encode('utf8'))

def start_server(addr, port, server_class=Server,
handler_class=RequestHandler):
    server_address = (addr, port)
    http_server = server_class(server_address, handler_class)
    print(f"Starting server on {addr}:{port}")
    http_server.serve_forever()

def main():
    addr = "0.0.0.0"
    port = 9943
    start_server(addr=addr, port=port)

if __name__ == "__main__":
    main()

```

Il nostro server gestisce una richiesta POST avente come header HTTP Content-Length che indicherà la lunghezza della password generata. Il comando apg, invocato tramite subprocess, restituirà una sola password (opzione n) di lunghezza minima (opzione m) e massima (opzione x) uguale al valore dell'header HTTP Content-Length passato dal client che effettua la richiesta.

Non ci resta che creare lo script Bash per automatizzare la generazione della password casuale dal Terminale.

```
#!/bin/bash
```

```
url="http://localhost:9943"

read -p "Inserisci la lunghezza della password desiderata: "
PASSWORD_LENGTH

post_request() {
    local content_length_header="Content-Length:
${PASSWORD_LENGTH}"
    local response=$(curl -s -X POST -H
"${content_length_header}" "${url}")
    echo $response
}

password=$(post_request)

echo $password
```

Conclusione

Docker offre un ottimo modo per bypassare i limiti imposti dall'architettura di macOS quando abbiamo bisogno di utilizzare una libreria o un utility presente principalmente nel mondo Linux. I container in tal senso si rivelano essere anche il modo migliore per distribuire le nostre soluzioni su più installazioni.