UNIVERSITÉ DE LIMOGES
Faculté de Science et Techniques

Master 1
Sécurité de l'Information et Cryptologie
(CRYPTIS)
Parcours Informatique

Protocoles et Programmation Réseaux - Semestre I

Projet Rapport: Proxy

SALAME Joe
MAKHOUL Vladimir
DUONG Dang-Hung

18 decembre, 2022

Table des matières

Chapitre 1

Principales fonctionnalités du proxy

Dans ce chapitre, nous aborderons les principales fonctionnalités de notre programme Proxy : gestion et transfert des requêtes, filtrage du contenu et utilisation de la mémoire cache pour optimiser le débit. Le filtrage peut être activé avec 2 options : `activate_filter` ou `deactivate_filter`

Cependant, lors de nos travaux, la fonctionnalité de filtrage n'est appliquée qu'aux paquets avec des protocoles HTTP mais pas HTTPS en raison de certains problèmes liés au tunnel sécurisé entre le client et le serveur, et nous devons également faire face à l'échange de données cryptées, où le proxy sera bloqué s'il essaie d'envoyer des données.

1.1 Filtrage du contenu du package

Le filtrage du contenu, tel que précisé dans la description du projet fournie par le Professeur, consiste principalement à insérer des textes dans le titre, à remplacer les textes ou encore à bloquer tout le site en fonction d'une liste de certains mots interdits. Cette liste est contenue dans un fichier *json*, qui peut être manipulé par l'utilisateur à l'aide de l'interface utilisateur graphique (GUI) ou de l'interface Web (plus de détails seront abordés au chapitre 2). Les données de configuration ont été enregistrées sous la forme d'un fichier *json* à l'aide du package `json` fourni par Python.

Voici à quoi ressemble un exemple de fichier de configuration :

```
{
    "filter_choice": "activate_filter",
    "forbidden_keywords": [
        "server",
        "adresse",
        "GPGPU",
        "bonnefoi",
        "port"
    ]
}
```

Dans notre application, il traite simplement de 2 choses : le toggle activation/désactivation du filtre, et la liste des mots-clés interdits, mais nous nous intéresserons uniquement au filtrage des mots-clés interdits dans cette section.

Nous avons 2 types de filtres : bloquer l'intégralité du site et donner à l'utilisateur une page d'erreur si l'URL contient un mot interdit, ou simplement remplacer les mots interdits dans le contenu du site. De plus, nous sommes en mesure de modifier l'en-tête de la requête et le titre de la page également.

1.1.1 Changer le titre de la page Web

```
def chgTitle(page):
    page=page.splitlines()
    p=''
    re_debut_titre = re.compile(r'(.*)<title>(.*)</title>(.*)',re.I)
    for line in page:
        if re_debut_titre.match(line):
            resultat = re_debut_titre.search(line)
            line = '<title>Proxy</title>'
        p+=line+'\n'
    return p
```

Pour changer le titre du site, on prend simplement en entrée tout le fichier HTML sous forme de texte brut, puis on trouve la balise `title` dans le `header` et on injecte le titre comme on veut à l'intérieur.

1.1.2 Filtrer l'entête de la requête

```
def filter(msg):
    packet = ""
    msg = msg.splitlines()
    print(msg)
    urlSplit = re.compile(r'(^.*) HTTP/1.1')

    for i in msg:
        if(re.match('(^.*) HTTP/1.1',i)):
            result1 = urlSplit.search(i)
            part1 = result1.group(1)
            part1+= ' HTTP/1.0'
            packet +=part1 + '\r\n'
            continue
        if(i=="Connection: keep-alive" or i == "Proxy-Connection: keep-
   alive" or i == "Accept-Encoding: gzip, deflate"):
            continue
        packet+=i +'\r\n'
    return packet
```

Comme spécifié dans les exigences, pour le bon fonctionnement du proxy, il est nécessaire de :

- Dans la requête envoyée au serveur et provenant du navigateur :
  - Supprimez les lignes commençant par `Connection : Keep-Alive` et `Proxy-Connection : Keep-Alive`
  - Supprimez la ligne commençant par `Accept-Encoding : gzip;`
- Faire des requêtes en version `HTTP/1.0`

1.1.3 Bloquer les sites dont l'URL contient des mots interdits

```
# Check if the url contains forbidden words
def forbidden(address: str) -> bool:
    with open('config.json', 'r') as file:
        config_data_dict = json.loads(file.read())
        forbidden_keywords = config_data_dict["forbidden_keywords"]
        forbidden = False
        for w in forbidden_keywords:
            aline = w.strip()
            if aline in address:
                forbidden = True
                break
    return forbidden
```

L'URL sera vérifiée si elle contient un ou plusieurs mots dans la liste interdite.

```
        if forbidden(address) and filter_choice == "activate_filter":
            reply = "HTTP/1.0 403 Forbidden\r\n"
            reply += "Proxy-agent: Pyx\r\n"
            reply += "\r\n"
            with open('index.html', 'r') as file:
                lines = file.readlines()
                for line in lines:
                    aline = line.strip()
                    reply += aline
            socket_client.sendall(reply.encode())
            socket_client.close()
```

Et si c'est le cas, le site est totalement bloqué et une page d'erreur (dans ce cas, elle s'appelle `index.html`) est renvoyée à l'utilisateur à la place.

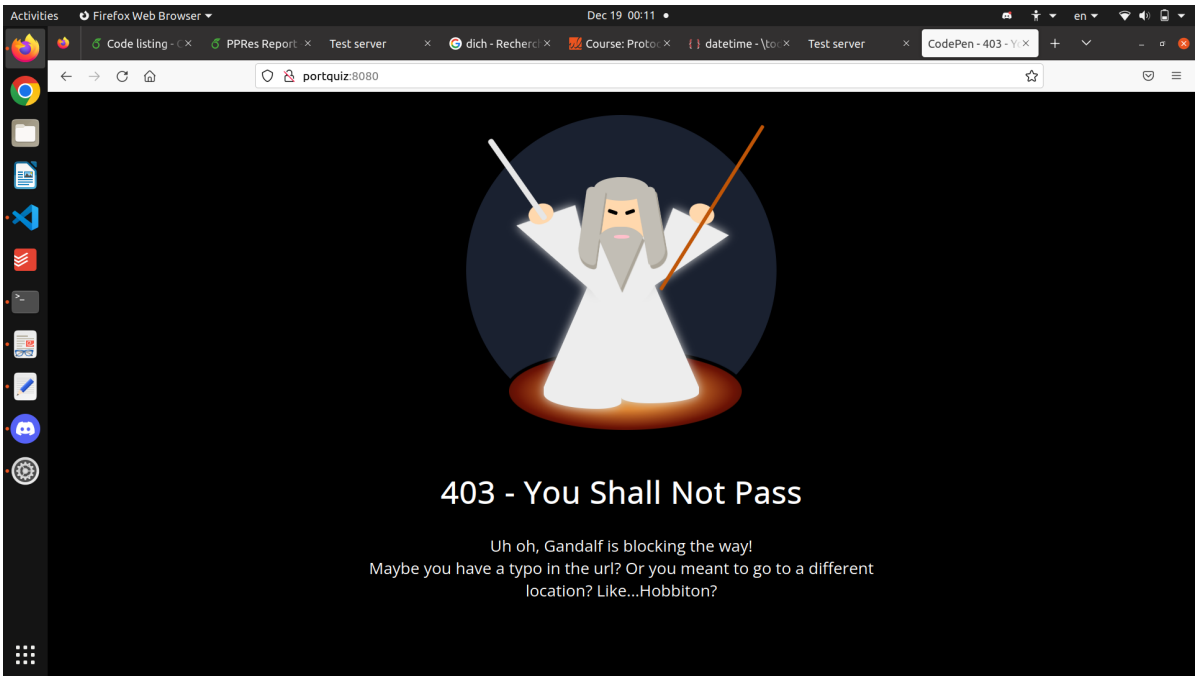

Figure 1.1: Page d'erreur renvoyée à l'utilisateur en cas de blocage du site

1.1.4 Remplacer/censurer les mots interdits dans le contenu des sites Web

```python
# Replace forbidden words in the content of the HTML page
def replaceforbid(page):
    with open('config.json', 'r') as file:
        config_data_dict = json.loads(file.read())
        forbidden_keywords = config_data_dict["forbidden_keywords"]
        for w in forbidden_keywords:
            aline = w.strip()
            if aline.casefold().lower() in page.lower():
                page = page.replace(aline,"*" * len(aline))
    return page
```

Si la page n'est pas bloquée, cette fonction prend tout le contenu de la page HTML en texte brut et censure tous les mots présentés dans la liste interdite par les astérisques (*).

```python
                if b'Content-Type: text/html' in page:
                    page = page.decode()
                    if filter_choice == "activate_filter":
                        page = replaceforbid(page)
                    page = chgTitle(page)
                    page = bytes(page,'utf8')
                socket_client.sendall(page)
                socket_client.close()
```

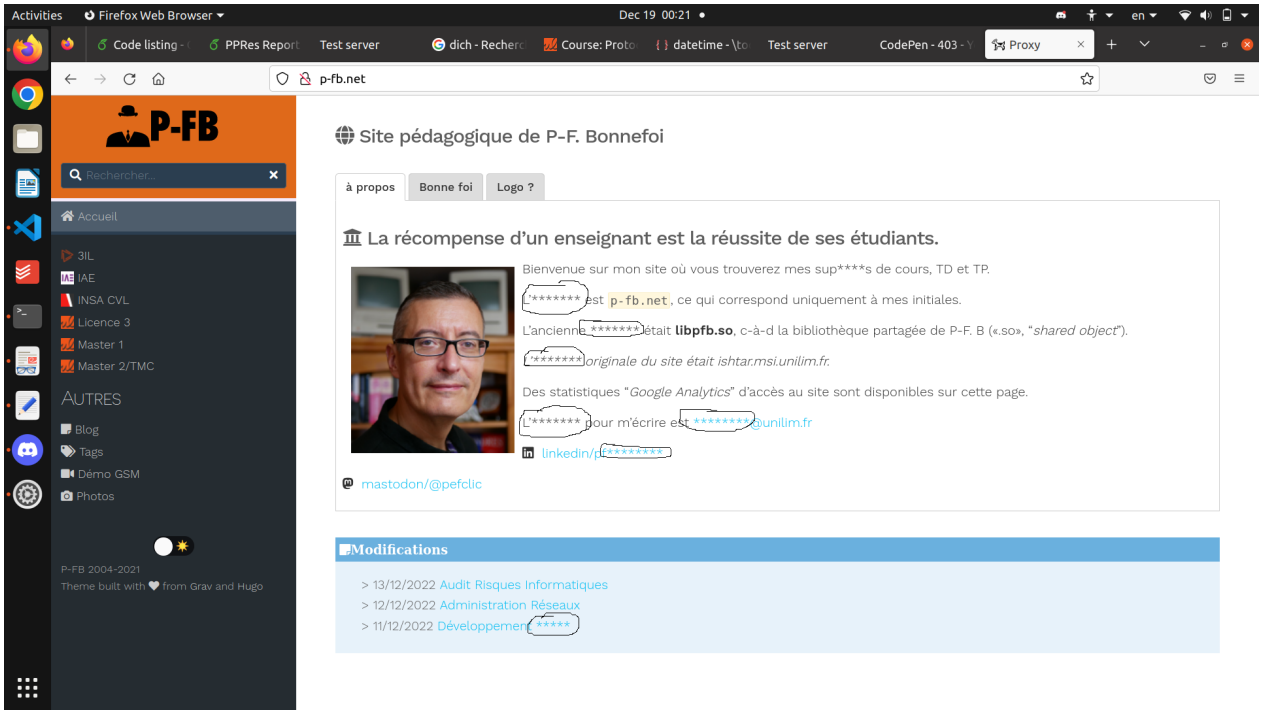

Figure 1.2: Les mots interdits sont censurés

1.2 Gestion et transmission des demandes et la mémoire cache

Chaque expression régulière divise l'URL en 3 groupes : adresse, port, chemin.

```
    # In case HTTP protocol was used
    if re.match(rx_pattern, url):
        urlSplit = re.compile(rx_pattern)
        result = urlSplit.search(url)
        address, port, chemin = result.groups()
```

Tout d'abord nous accédons à un fichier spécifique nommé `nbVisits.txt` qui contient tous les sites Web visités précédemment avec le nombre de visites. Ensuite nous ouvrons un autre fichier `url.txt` qui contient les paquets de chaque site web sauvegardé en mémoire et on l'insère dans un dictionnaire.

```
try:
    visitedSites=open("nbVisits.txt",'r')

except Exception as e:
    print(e.args)
    sys.exit(1)

while 1:
    ligne=visitedSites.readline()
    if not ligne:
        break
    l=ligne.split(':')
    nb_of_visits[l[0]]=int(l[1])

visitedSites.close()

try:
    memoRead=open('urls.txt','r')
except Exception as e:
    print(e.args)
    sys.exit(1)
memoRead.readline()

while 1:
    ligne=memoRead.readline()
    if not ligne:
        break
    lig=ligne
    if ligne=='end\n':
         continue
    ligne=ligne.split(':')
    url=ligne[0]
    #print(url)
    memo[url]=ligne[1]
    while 1:
        l=memoRead.readline()
        if l == 'end\n' or l=='end':
```

```
            break
        response+=l
    if len(url)!=0:
        memo[url]+=response
    response=''
memoRead.close()
```

Nous comparons les clés du dictionnaire `memo` avec l'URL actuelle.

- S'il correspond, nous récupérons le contenu de ce dictionnaire et nous l'envoyons directement au client sans établir une connexion avec le serveur.

```
            if address + chemin in memo.keys():
                response=bytes(memo[address+chemin],'utf8')
                socket_client.sendall(response)
                socket_client.close()
```

- Sinon il se connecte au serveur et après filtrage le proxy envoie la requête au serveur, et celui-ci commence à recevoir des réponses qui seront assemblées en une seule réponse. Si le type de contenu est Html alors on appelle 2 fonctions qui filtrent le contenu de la page et changent le titre en "proxy". Enfin il envoie la page au navigateur. S'il a été visité plus de 5 fois, il ouvre le fichier `urls.txt` et y ajoute l'adresse et le contenu de la page. Ensuite, nous incrémentons le nombre de visites. S'il n'existe pas dans `number_of_visits` on l'ajoute au dictionnaire `number_of_visits` puis au fichier `nbVisits.txt`.

```
            else:
                if len(port)==0:
                    port="80"
                adresse_serveur = socket.gethostbyname(address)
                # if address in memo.keys():
                #     socket_client.sendall(memo[address])
                # else:
                print("Fetch")
                socket_server = socket.socket(socket.AF_INET, socket.
    SOCK_STREAM, socket.IPPROTO_TCP)
                try:
                    port = int(port)
                    socket_server.connect((adresse_serveur, port))
                except Exception as e:
                    print("Probleme de connexion", e.args)
                req = filter(msg)
                page = b''
                req = bytes(req, 'utf-8')
                socket_server.sendall(req)
                while 1:
                    data = socket_server.recv(4096)
                    if not data:
                        break
                    page += data

                if b'Content-Type: text/html' in page:
                    page = page.decode()
```

```
                    if filter_choice == "activate_filter":
                        page = replaceforbid(page)
                    page = chgTitle(page)
                    page = bytes(page,'utf8')
                socket_client.sendall(page)
                socket_client.close()

                try:
                    updateVisits=open("nbVisits.txt",'w')
                except Exception as e:
                    print(e.args)
                    sys.exit(1)

                if address+chemin not in memo.keys():
                    if address+chemin in nb_of_visits.keys():
                        if nb_of_visits[address+chemin] >= 5:
                            try:
                                memoAdd=open('urls.txt','a+')
                            except Exception as e:
                                print(e.args)
                                sys.exit(1)
                            memoAdd.write('\n'+address+chemin+':'+str(
page,'utf8'))
                            memoAdd.write('\nend')
                            memoAdd.close()
                        nb_of_visits[address+chemin]+=1
                        for site in nb_of_visits.keys():
                            updateVisits.write(site+":"+str(
nb_of_visits[site])+'\n')
                        updateVisits.close()
                    else:
                        nb_of_visits[address+chemin]=1
                        for site in nb_of_visits.keys():
                            updateVisits.write(site+":"+str(
nb_of_visits[site])+'\n')
```

1.3 Connexion à l'aide du protocole HTTPS

Concernant le protocole HTTPS, nous ne pouvions pas beaucoup interférer avec le paquet, à la place, le client envoie d'abord la demande de connexion TLS avec un serveur au proxy, le serveur proxy extrait ensuite l'URL du serveur de destination et établit une connexion avec lui puis il envoie au client un paquet pour l'informer que la connexion est établie. Après cela, le proxy crée un tunnel à 2 canaux (un pour transmettre les données du client vers le serveur, et l'autre pour le sens inverse) . Le proxy dans ce cas ne joue que le rôle de transmettre les données, au lieu de manipuler directement les données comme illustré dans les sections précédentes.

```
    # In case HTTPS Protocol was used
    else:
        urlSplit = re.compile(r'([^:]*):?([^:\D/$]*)?[/]?([^$]{0,})?')
        result = urlSplit.search(url)
```

```
        address, port, chemin = result.groups()

        if forbidden(address):
            reply = "HTTP/1.0 403 Forbidden\r\n"
            reply += "Proxy-agent: Mozilla/5.0\r\n\r\n"
            reply += "\r\n"
            socket_client.sendall(reply.encode())
        else:
            adresse_serveur = socket.gethostbyname(address)
            socket_server = socket.socket(socket.AF_INET, socket.
    SOCK_STREAM, socket.IPPROTO_TCP)
            try:
                port = int(port)
                socket_server.connect((str(adresse_serveur), port))
            except Exception as e:
                print ("Probleme de connexion", e.args)
                sys.exit(1)
            # Connect to port 443
            # If successful, send 200 code response
            reply = "HTTP/1.0 200 Connection established\r\n"
            reply += "Proxy-agent: Mozilla/5.0\r\n"
            reply += "\r\n"
            socket_client.sendall(bytes(reply, "utf-8"))


            client_to_server_thread = threading.Thread(target=forward,
    args=(socket_client, socket_server,))
            client_to_server_thread.start()

            server_to_client_thread = threading.Thread(target=forward,
    args=(socket_server, socket_client,))
            server_to_client_thread.start()
```

La fonction `forward()` :

```
# Forward data stream from one socket to another
def forward(socket_source: socket.socket, socket_dest: socket.socket) ->
    None:
     while 1:
        stream_data = socket_source.recv(1)
        socket_dest.sendall(stream_data)
```

Chapitre 2

Web Interface for Proxy Configuration

Dans ce chapitre, nous expliquerons brièvement comment nous avons implémenté l'interface Web afin que l'utilisateur puisse configurer le proxy via l'accès Web.

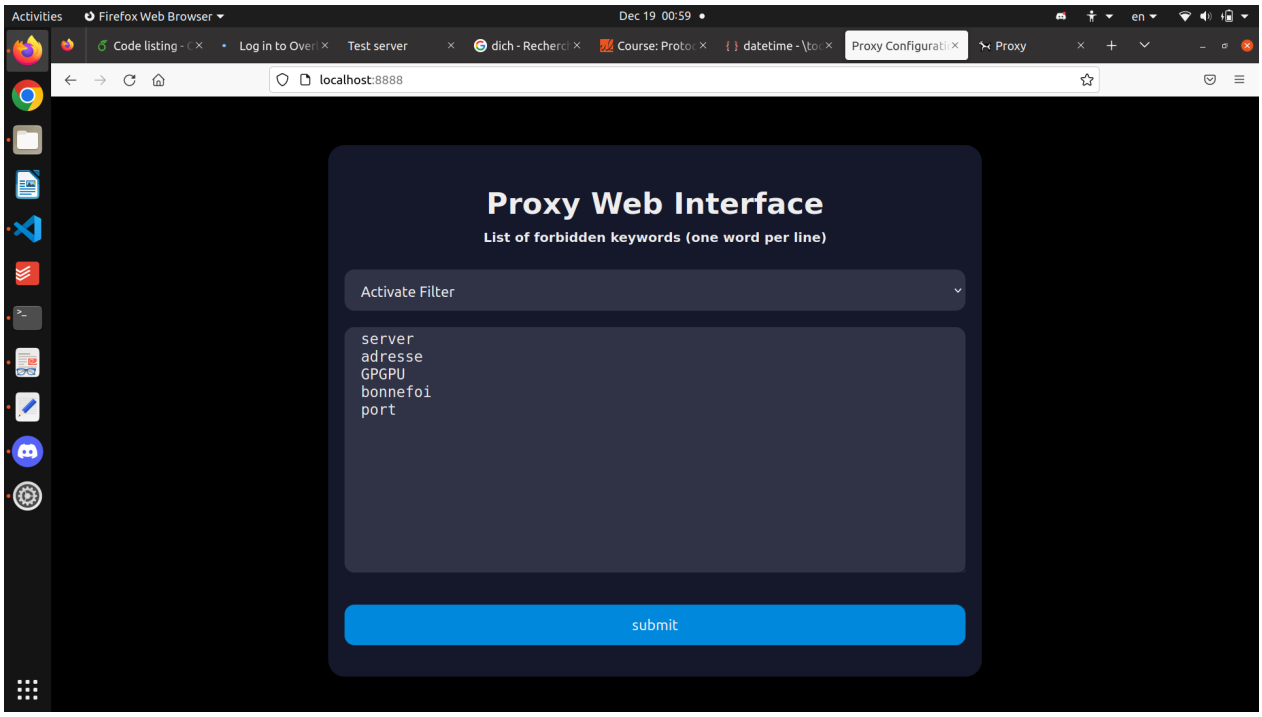

Figure 2.1: Web Interface for Proxy Configuration

Le code source de l'implémentation de l'interface Web a été principalement placé dans le fichier `config_interface.py`. Afin d'héberger un serveur local spécifiquement pour cela, nous avons principalement utilisé la bibliothèque `http.server` fournie par Python, et les données sont enregistrées dans le fichier `config.json`, comme mentionné dans la section 1.3.

Pour gérer les requêtes de l'interface Web, nous avons défini une classe enfant de `BaseHTTPRequestHandler` et l'avons nommée `ConfigProxyInterface`. Cette classe sert principalement à 2 fins : retourner l'interface Web à configurer via la fonction `do_GET()`, et prendre en compte l'entrée de l'utilisateur pour personnaliser la configuration du proxy à l'aide de la fonction `do_POST()`.

```
    def do_GET(self):
        f = open("config.json", "r")
        config_data_dict = json.loads(f.read())
        f.close()
        filter_choice = config_data_dict["filter_choice"]
        forbidden_keywords = config_data_dict["forbidden_keywords"]

        self.send_response(200)
        self.send_header("Content-type", "text/html")
        self.end_headers()
        with open("web.html", "r") as f:
            html_contents = f.readlines()
        if filter_choice == "deactivate_filter":
            html_contents[10], html_contents[11] = html_contents[11],
    html_contents[10]
        if len(forbidden_keywords) > 0:
            temp = html_contents[17].split("><")
            temp[0] += ">"
            temp[1] = "<" + temp[1]
            words = ""
            for w in forbidden_keywords:
                words += w + "&#13;&#10"
            html_contents[17] = temp[0] + words + temp[1]
        for l in html_contents:
            self.wfile.write(bytes(l, "utf-8"))
```

La fonction `do_GET()` lit d'abord le fichier json pour obtenir le contenu prédéfini, après cela, ces données sont injectées dans le fichier HTML et affichées à l'utilisateur.

```
    def do_POST(self):
        content_length = self.headers['Content-Length']
        if content_length and content_length != '0':
            try:
                body = self.rfile.read(int(content_length))
                config_data = body.decode().split('&')
                config_data_dict = {}
                for entry in config_data:
                    key, value = entry.split("=", 1)
                    if key == "forbidden_keywords":
                        value = value.split("%0D%0A")
                        value = list(filter(lambda a: len(a) > 0, value))
                    config_data_dict[key] = value
                config_data_json = json.dumps(config_data_dict, indent=4)
                print("Config data in JSON format:")
                print(config_data_json)
                with open("config.json", "w") as f:
                    f.write(config_data_json)
                resp_msg = 'Proxy configuration saved!'
            except Exception as e:
                resp_msg = "Submission problem:" + str(e.args)
                print(resp_msg)
                return
        else:
            resp_msg = "No data was submitted"
        self.send_response(200)
```

```
        self.send_header('Content-type', 'text/plain')
        self.send_header('Content-length', len(resp_msg))
        self.end_headers()
        response = BytesIO()
        response.write(b'Proxy configuration saved!\n')
        response.write(bytes(str(config_data_json), "utf-8"))
        self.wfile.write(response.getvalue())
```

La fonction `do_POST()` va récupérer le choix de (dés)activer le filtre et la liste des mots-clés interdits à l'utilisateur, et modifier le fichier json. Il répondra également à l'utilisateur pour montrer si sa demande a été soumise avec succès.

```
def run_config_server():
    config_server_host = 'localhost'
    config_server_port = 8888
    webServer = HTTPServer((config_server_host, config_server_port),
   ConfigProxyInterface)
    print("Proxy configuration server listening at http://%s:%s" % ('
   localhost', 8888))
    try:
        webServer.serve_forever()
    except KeyboardInterrupt:
        pass
    webServer.server_close()
```

L'interface Web est hébergée sur `localhost`, port `8888` par défaut.

```
config_sv_thread = threading.Thread(target=run_config_server)
config_sv_thread.start()
```

Il est géré par un thread qui s'exécute en même temps que le proxy.