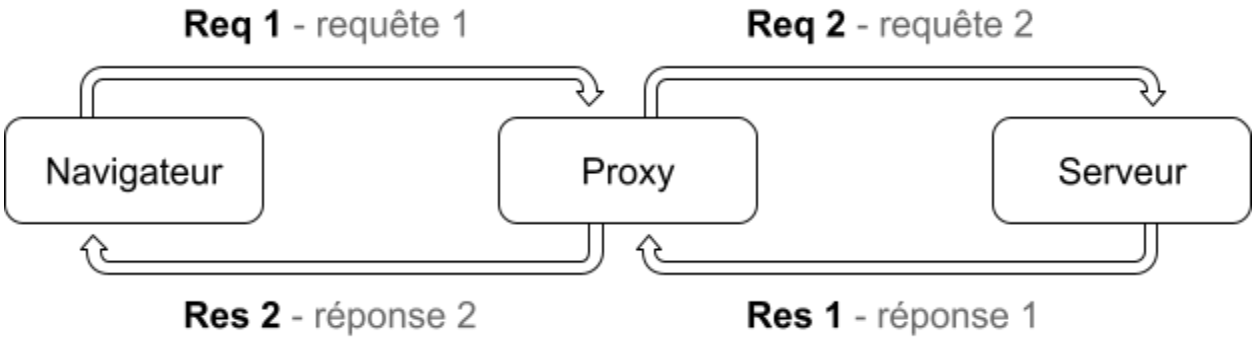
Protocoles et Programmation Réseaux

Projet - Serveur proxy

Sommaire

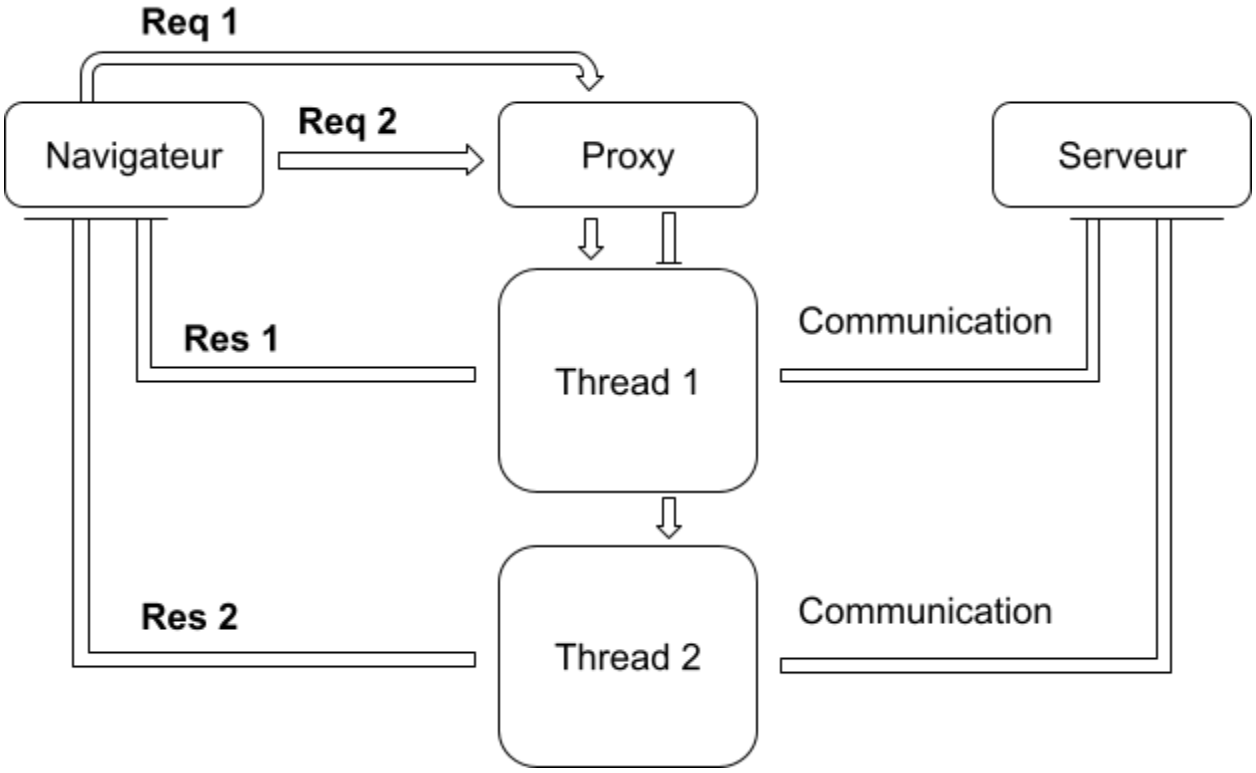
Proxy, programmation socket en python

L'objectif de ce projet était de réaliser un proxy utilisant les sockets en python. Ci-dessous un schéma précisant le fonctionnement de celui-ci *(nous ferons référence à req ½ et rep ½ par la suite)* :



Le proxy possède une socket TCP en écoute sur un port donné *(admettons 1234)*, pour chacune des nouvelles connexions acceptées, la requête en provenance du serveur **Req 1** est transmise au serveur à qui elle était destinée *(en tant que **Req 2**)*. Le Proxy joue le rôle de "man in the middle" entre le client et le serveur.

Afin de maintenir le proxy disponible pour les prochaines connexions, les différentes requêtes sont gérées par des fils d'exécution indépendants.



Utilisation pratique

Pour le bon fonctionnement du proxy, il est préférable d'utiliser google chrome ainsi que de vérifier que le **cache du navigateur** a bien été vidé, penser à le vider de nouveau à chaque changement de configuration.

Les paramètres de configuration sont stockés dans le fichier **config** au format json et peuvent être directement *(une fois le proxy configuré)* :

- accessibles, via le lien http://config/home
- modifiés, via le lien http://config

Le proxy ne supporte actuellement que le protocole HTTP.

Fonctionnement standard

Afin de permettre la transmission de **Req 1** par le proxy, il nous faut mettre ce dernier en écoute, *(ici en localhost:4567, toutefois, le proxy pourrait se trouver sur un serveur distant)* :

```
def start(self):
    self.serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.serverSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    self.serverSocket.bind((
        Self.config["parameters"]['host'],
        self.config["parameters"]['port']))
    self.serverSocket.listen(socket.SOMAXCONN)
    print("[+] Listening on port " , self.config["parameters"]['port'])
```

A chaque nouvelle connexion, lancer une thread afin de permettre au proxy de continuer à servir les autres clients :

```
thr =threading.Thread(target=self.mainThread , args=(line , conn, addr))
thr.setDaemon(True)
thr.start()
```

Dans la fonction permettant la communication *(exécutée dans la thread)*, on cherche dans un premier temps à extraire le **nom de domaine** ainsi que le **numéro de port** de la requête reçue, une fois ces variables définies, la connexion au serveur peut être établie :

```
def mainThread(self ,line , conn , addr):
    try :
        decodedLine = line.decode("utf_8")
        url = decodedLine.split("\n")[0].split(' ')[1]
        Patern = re.search("https?://", url)
        url = url if patern == None else url[patern.span()[1]:]
        patern = re.search("[a-zA-Z0-9\._-]+\.?[a-zA-Z0-9]*:?[0-9]{0,5}", url)
```

```
        url = patern.group()
        Patern = re.search(":", url)
        port = 80 if patern == None else int(url[patern.span()[1]:])
        url = url if patern == None else url[:patern.span()[1]-1]

        self.connectToDestination(conn, line, url, port, addr)

    except Exception as e:
        print(e)
```

Pour ce faire, une nouvelle socket est créée afin d'installer une communication avec le serveur en question, la connexion est maintenue tant que le proxy reçoit des données. Toutes les informations reçues sont transmises au client *(la fonction modifyRequest() s'occupe de passer HTTP 1.1 en 1.0 ainsi que de faire les modifications nécessaires expliquées dans le sujet)* :

```
def connectToDestination(self, conn, data, webHost, webPort, addr):
    print("[+] Connecting to server ( ", webHost , " ) on port :" , webPort)
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((webHost, webPort))
    sock.settimeout(30)

    newData = self.modifyRequest(data)
    sock.sendall(newData)

    while 1:
        rep = sock.recv(self.config["parameters"]['bufferSize'])

        if not rep:
            conn.sendall(fullResponse)
            break

        conn.sendall(rep)
```

Et voilà, nous avons un proxy qui transmet chacune des requêtes qu'il reçoit sans se poser de questions.

Fonctions de configuration

La page de configuration de notre proxy propose de :

- définir les paramètres généraux
- restreindre des noms de domaine
- rediriger des noms de domaine
- interdire certaines extensions *(.png .mp4 .ppm …)*

Chacune des fonctions suivantes modifient le fichier de configuration :

```
self.goToConfigure()
self.showConfigPage()
self.setProxyParameters()
self.setForbiddenHosts()
self.setForbiddenExtensions()
self.setRedirectHosts()
self.setChangeHTMLWord()
```

`self.goToConfigure()`, `self.showConfigPage()` ont pour objectif de rediriger le client respectivement vers la page de configuration et la page d'affichage des paramètres.

`self.setProxyParameters()` permet de modifier les paramètres généraux du proxy *(port, host et buffer size)*, les autres fonctions effectuent le même traitement sur des paramètres différents.

Pour ce faire, nous testons si les url reçus font partie de ceux "réservés" au maintien du proxy, *(auquel cas une fonction de configuration est lancée)* dans le cas contraire le proxy agit normalement :

```
    if url == 'http://config/':
        thr =threading.Thread(target=self.goToConfigure (conn))

    elif url == 'http://config/home':
        thr =threading.Thread(target=self.showConfigPage (conn))

    elif re.search('(?:https?:\/\/config\/proxySetting\?host=)[...]', url):
        thr =threading.Thread(target=self.setProxyParameters , args=(url , conn ))
    else:
        thr =threading.Thread(target=self.mainThread , args=(line , conn, addr))
```

Fonctions de Filtrage

Les variables de filtrage ayant été définies, il nous faut maintenant appliquer les filtres en question.

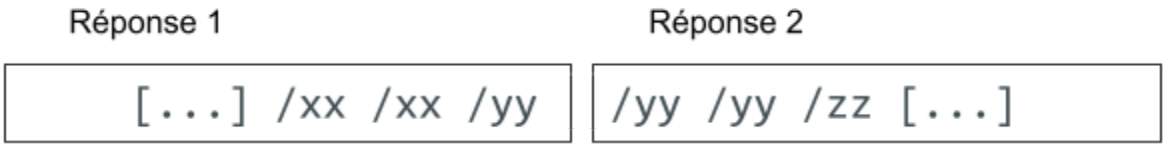
Commençons par le remplacement de mots :

```
def changeWordsinHTMLPage(self , res , isBodyTAG):

	try:
		val = res.decode('ISO-8859-1', errors='replace')
	except Exception as e:
		print(e)
		return res, isBodyTAG
	res = val

	htmlStart = res.find('<body')
	htmlEnd = res.find('</body>')

	if htmlStart == -1 and htmlEnd == -1 and not isBodyTAG:
		return res.encode('ISO-8859-1') , isBodyTAG

	if htmlStart != -1:
		isBodyTAG = True

	if htmlEnd !=-1:
		isBodyTAG =False

	if htmlStart == -1:
		htmlStart = 0

	if htmlEnd == -1:
		htmlEnd = len(res)


	resStart = res[0:htmlStart]
	resEnd = res[htmlEnd:len(res)]

	resSub = res[htmlStart:htmlEnd]
	for val in self.config['changeWords']:
		resSub = resSub.replace(val['word'], val['changeTo'])

	res = resStart + resSub + resEnd

	return res.encode('ISO-8859-1') , isBodyTAG
```

L'objectif de cette fonction est de remplacer toutes les occurrences d'un mot *(ou suite de lettres)* par un autre *(vide ou non)*.

La première chose à faire est de décoder les octets reçus, cette opération peut sembler triviale néanmoins elle pose de nombreuses questions. On sait que nos données sont encodées en UTF-8. C'est un encodage multibytes ce qui signifie qu'un caractère peut être composé de plusieurs octets. Ceci étant dit, nous recevons nos données par blocs qui dépendent de la taille du buffer que nous avons autorisé.

Admettons le cas suivant :



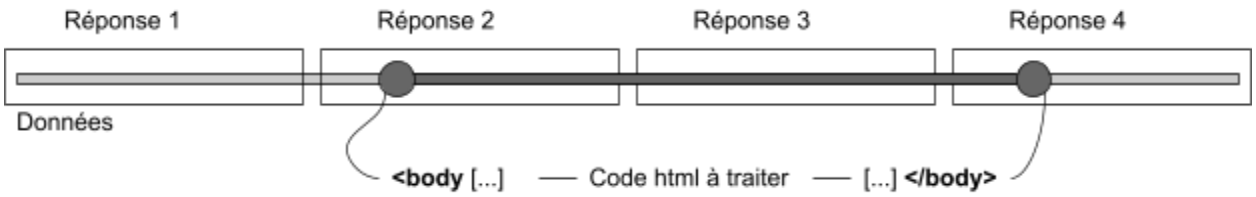
Le mot **y**, codé sur trois octets, se retrouve dans deux réponses différentes. En temps normal cela ne pose aucun problème car le client peut reconstruire la réponse complète avant de la traiter. Dans le cas de notre proxy, nous devons effectuer des modifications sur ces données.

Nous avons donc eu l'idée de :

- attendre la fin de la réponse du serveur
- effectuer notre traitement
- envoyer les données modifiées au client

Toutefois cette idée n'a pas été fructueuse et nous avons corrigé nos erreurs de décodage en utilisant le format ISO-8859-1. Suite à de nombreuses recherches, nous n'avons pas compris la source de notre erreur.

Une fois les données au bon format, nous cherchons les mots `<body` *(début des données à traiter)* et </body> *(fin des données à traiter)* afin de nous situer sur les informations reçues. Nous pouvons alors traiter les données présentes entre `htmlStart` et `htmlEnd` :



Ensuite la suppression d'extension :

```
def removeExtensions (self, str, extensions):
    try:
        str =str.decode('ISO-8859-1' , errors='ignore')
    except Exception as e:
        print(e)
        return str
    for i in range(len(str)):
        for ext in extensions :
            if str.endswith(ext,0 , i):
                for j in range(i+1 ,-1 ,-1):
                    if str.startswith('="' ,j , i):
                        str=str.replace(str[j:i], '="')
                        break
    return str.encode('ISO-8859-1')
```

Cette fonction peut sembler complexe, toutefois son fonctionnement est très simple, on itère sur la chaîne de caractères fournie en entrée *(la réponse du serveur)*. Pour chacune des extensions à supprimer une recherche est effectuée.

Prenons un exemple, on trouve la sous-chaîne suivante :

On sait qu'il existe une extension .png à la position i, on cherche le nom de l'image et on le remplace par une chaîne vide afin d'obtenir :

L'image vient d'être supprimée

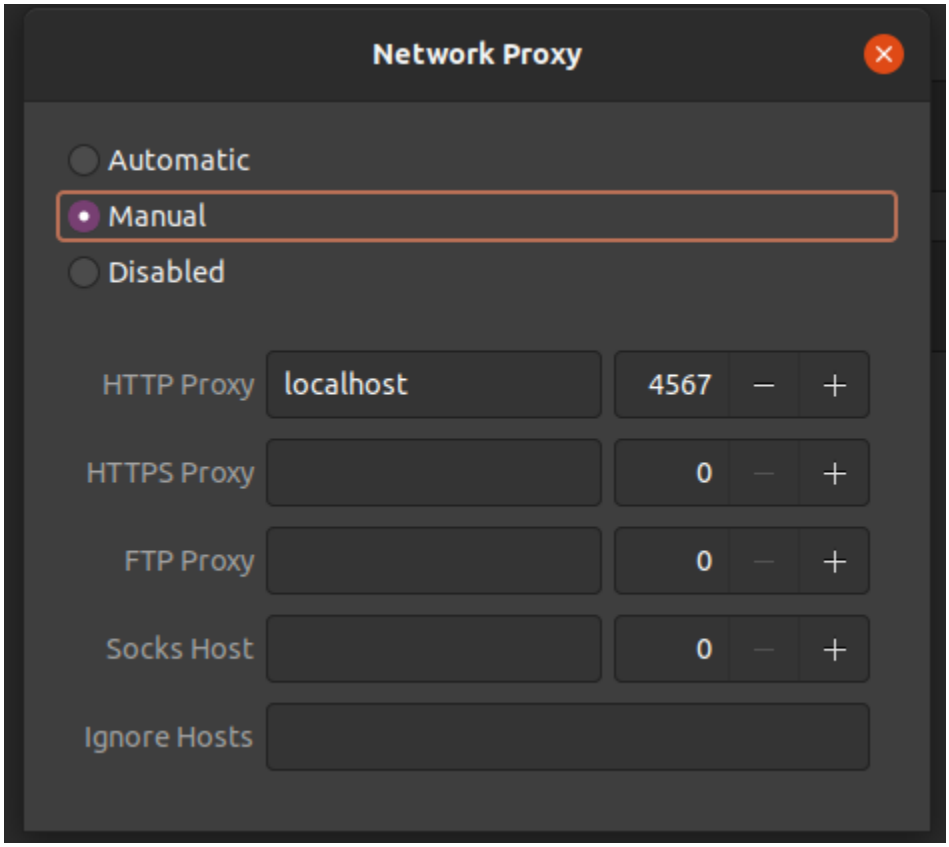
Les dernières fonctionnalités sont celles permettant la redirection et l'interdiction de noms de domaines :

```
try :
    for forbidenHost in self.config['forbiddenHosts']:
        if socket.gethostbyname(forbidenHost) == socket.gethostbyname(webHost) :
               print("[+] Forbidden !! try later")
               conn.close()
               return
except Exception as e:
    print(e)

try :
     for host in self.config['redirectHosts']:
         if socket.gethostbyname(host['host']) == socket.gethostbyname(webHost) :
                webHost = host['changeTo']
                data = data.decode('utf_8').replace(host['host'], webHost).encode('utf_8')
except Exception as e:
    print(e)
```

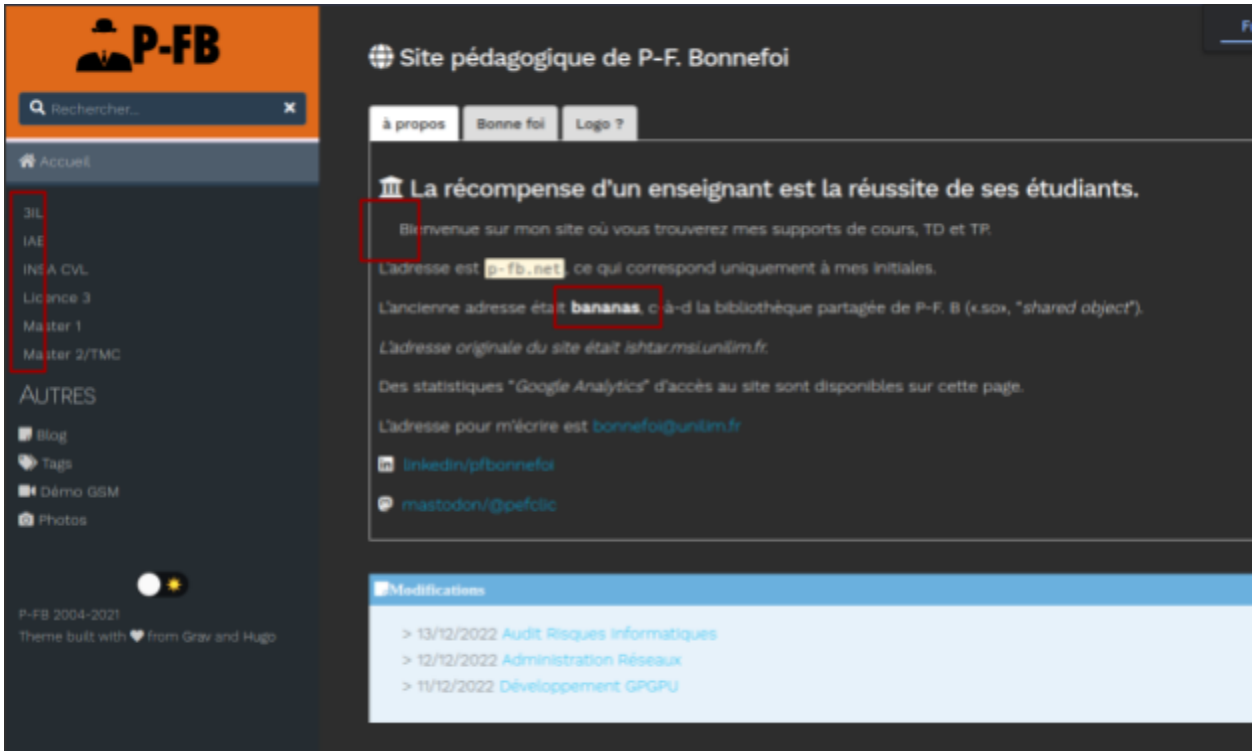
Pour ce faire, nous comparons les différents noms de domaines présents dans notre configuration, si on obtient une équivalence, alors soit le nom de domaine est remplacé soit la connexion n'aboutira pas.

Démonstration d’utilisation

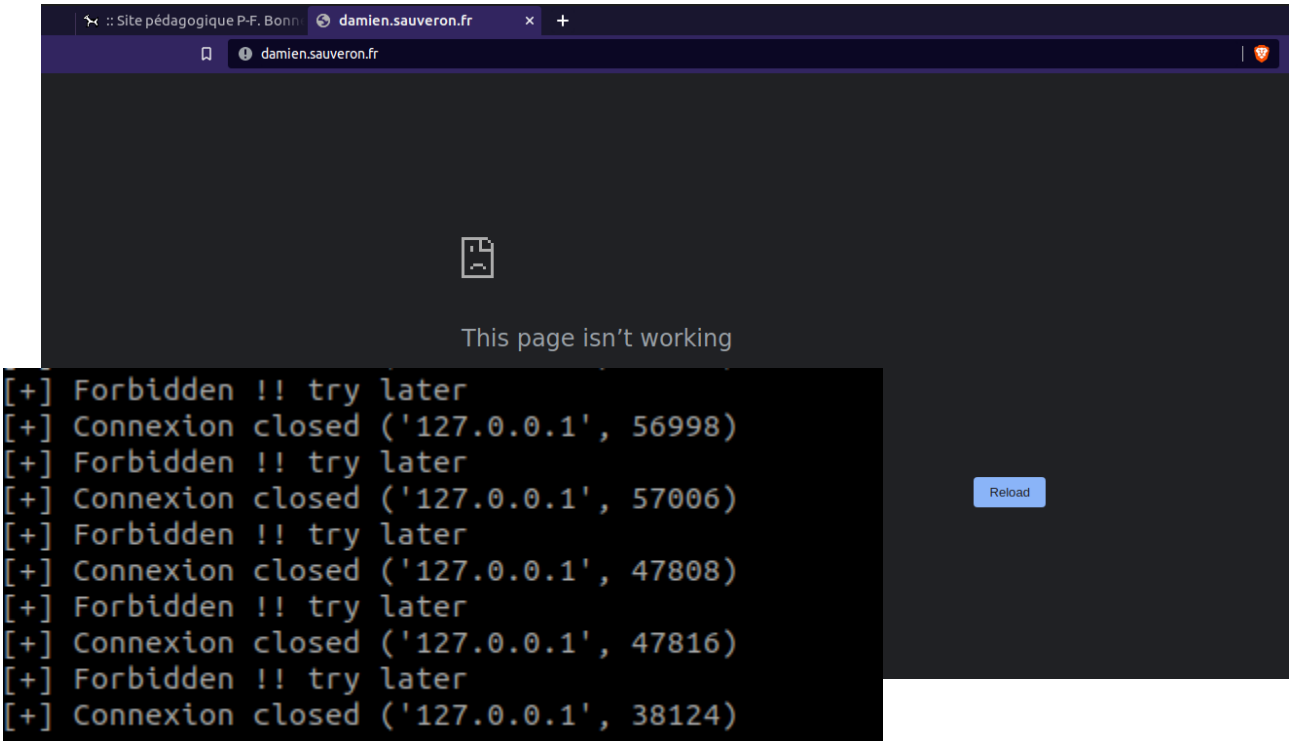
Commencer par lancer le proxy `python3 main.py` dans un terminal, configurer ensuite le proxy HTTP du navigateur sur le port adéquat :



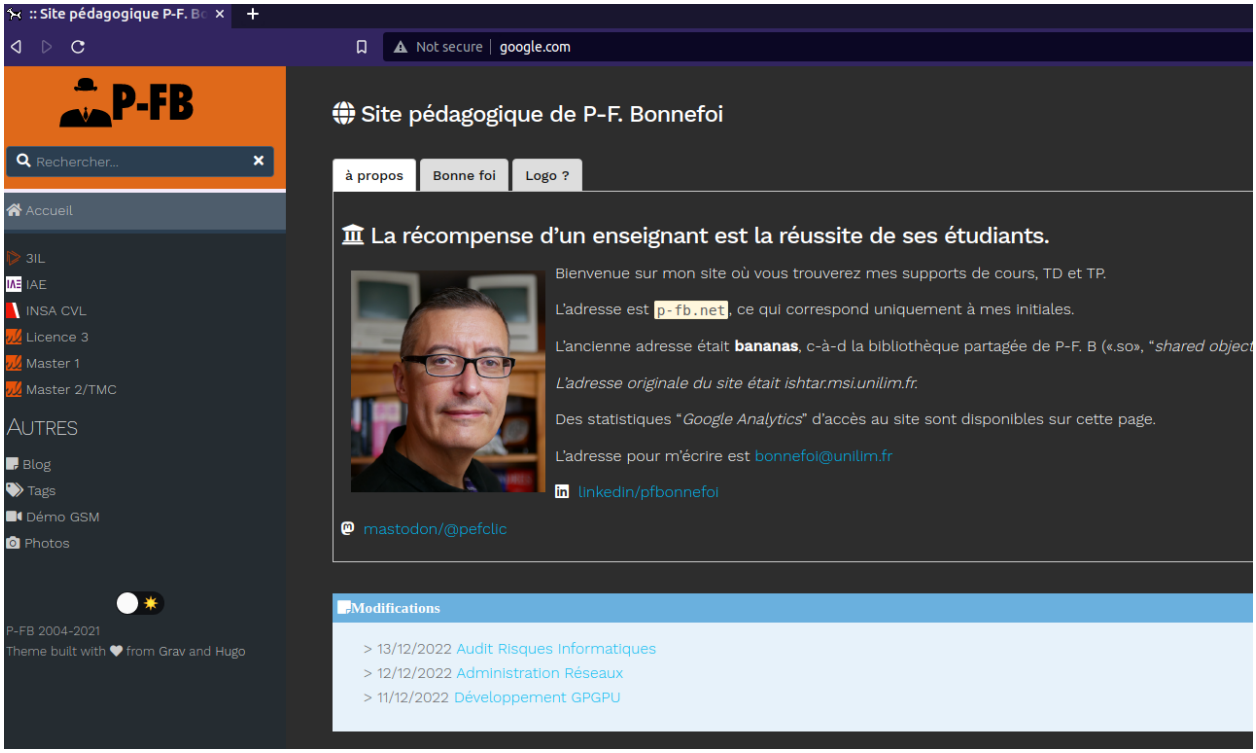
Exemple sur le site http://p-fb.net : *(libpfb.so → bananas, extension png supprimée)*



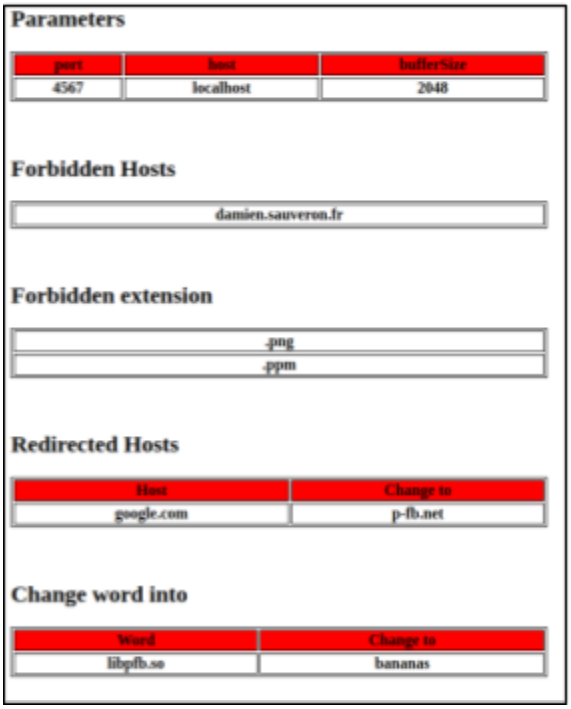
Exemple sur le site https://damien.sauveron.fr/ : *(Accès à un site non autorisé)*



Exemple sur le site http://p-fb.net : *(redirection de google.com)*



Affichage des parametres



Modification des parametres