Analyse de l’interdiction des serveurs proxy

Les API dans les serveurs proxy peuvent-elles éviter les bannissements du GFW ?

Je gère un serveur simple sur mon instance Shadowsocks avec le code suivant :

```
from flask import Flask, jsonify
from flask_cors import CORS
import subprocess

app = Flask(__name__)
CORS(app)  # Activer CORS pour toutes les routes

@app.route('/bandwidth', methods=['GET'])
def get_bandwidth():
    # Exécuter la commande vnstat pour obtenir les statistiques de trafic sur un intervalle de 5 minute
    result = subprocess.run(['vnstat', '-i', 'eth0', '-5', '--json'], capture_output=True, text=True)
    data = result.stdout

    # Retourner les données capturées sous forme de réponse JSON
    return jsonify(data)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Et j’utilise nginx pour servir le port 443 comme indiqué ci-dessous :

```
server {
    listen 443 ssl;
    server_name www.some-domain.xyz;

    ssl_certificate /etc/letsencrypt/live/www.some-domain.xyz/fullchain.pem; # géré par
    # ...
    location / {

        proxy_pass http://127.0.0.1:5000/;
```

```
        # ...
    }
}
```

Ce programme serveur fournit des données réseau, et j'utilise ce serveur comme mon serveur proxy, me permettant d'afficher mon statut en ligne sur mon blog en utilisant les données réseau.

Ce qui est intéressant, c'est que le serveur n'a pas été banni par le Grand Firewall (GFW) ou tout autre système de contrôle réseau depuis plusieurs jours maintenant. Normalement, le serveur proxy que je configure serait banni en un ou deux jours. Le serveur exécute un programme Shadowsocks sur un port comme 51939, donc il fonctionne avec un trafic Shadowsocks mélangé à un trafic API régulier. Ce mélange semble amener le GFW à croire que le serveur n'est pas un proxy dédié, mais plutôt un serveur normal, empêchant ainsi le bannissement de l'IP.

Cette observation est intrigante. Il semble que le GFW utilise une logique spécifique pour différencier le trafic proxy du trafic régulier. Alors que de nombreux sites comme Twitter et YouTube sont bloqués en Chine, de nombreux sites étrangers—comme ceux des universités et entreprises internationales—restent accessibles.

Cela suggère que le GFW fonctionne probablement sur la base de règles qui distinguent le trafic HTTP/HTTPS normal du trafic lié aux proxys. Les serveurs qui gèrent les deux types de trafic semblent éviter les bannissements, tandis que les serveurs qui ne gèrent que le trafic proxy sont plus susceptibles d'être bloqués.

Une question est de savoir sur quelle plage de temps le GFW accumule les données pour bannir—que ce soit un jour ou une heure. Pendant cette plage de temps, il détecte si le trafic provient exclusivement d'un proxy. Si c'est le cas, l'IP du serveur est bannie.

Je visite souvent mon blog pour revoir ce que j'ai écrit, mais dans les semaines à venir, je vais me concentrer sur d'autres tâches au lieu d'écrire des articles de blog. Cela réduira mon accès à l'API `bandwidth` via le port 443. Si je constate que je suis à nouveau banni, je devrais écrire un programme pour accéder régulièrement à cette API pour tromper le GFW.

Voici la version raffinée de votre texte avec une structure et une clarté améliorées :

Comment fonctionne le Grand Firewall (GFW).

Étape 1 : Journalisation des Requêtes

```
import time

# Base de données pour stocker les données de requête
request_log = []

# Fonction pour journaliser les requêtes
def log_request(source_ip, target_ip, target_port, body):
    request_log.append({
        'source_ip': source_ip,
        'target_ip': target_ip,
        'target_port': target_port,
        'body': body,
        'timestamp': time.time()
    })
```

La fonction `log_request` enregistre les requêtes entrantes avec des informations essentielles comme l'IP source, l'IP cible, le port cible, le corps de la requête et l'horodatage.

Étape 2 : Vérification et Bannissement des IPs

```
# Fonction pour vérifier les requêtes et bannir les IPs
def check_and_ban_ips():
    banned_ips = set()

    # Parcourir toutes les requêtes journalisées
    for request in request_log:
        if is_illegal(request):
            banned_ips.add(request['target_ip'])
        else:
            banned_ips.discard(request['target_ip'])

    # Appliquer les bannissements à toutes les IPs identifiées
    ban_ips(banned_ips)
```

La fonction `check_and_ban_ips` parcourt toutes les requêtes journalisées, identifiant et bannissant les IPs associées à des activités illégales.

Étape 3 : Définition de ce qui Rend une Requête Illégale

```
# Fonction pour simuler la vérification si une requête est illégale
def is_illegal(request):
    # Placeholder pour la logique réelle de vérification des requêtes illégales
    # Par exemple, vérifier le corps de la requête ou la cible
    return "illegal" in request['body']
```

Ici, `is_illegal` vérifie si le corps de la requête contient le mot “illegal”. Cela peut être étendu à une logique plus sophistiquée en fonction de ce qui constitue une activité illégale.

Étape 4 : Bannissement des IPs Identifiées

```
# Fonction pour bannir une liste d'IPs
def ban_ips(ip_set):
    for ip in ip_set:
        print(f"Bannissement de l'IP: {ip}")
```

Une fois les IPs illégales identifiées, la fonction `ban_ips` les bannit en imprimant leurs adresses IP (ou, dans un système réel, en les bloquant).

Étape 5 : Méthode Alternative pour Vérifier et Bannir les IPs Basée sur 80% de Requêtes Illégales

```
# Fonction pour vérifier les requêtes et bannir les IPs basée sur 80% de requêtes illégales
def check_and_ban_ips():
    banned_ips = set()
    illegal_count = 0
    total_requests = 0

    # Parcourir toutes les requêtes journalisées
    for request in request_log:
        total_requests += 1
```

```
        if is_illegal(request):
            illegal_count += 1

    # Si 80% ou plus des requêtes sont illégales, bannir ces IPs
    if total_requests > 0 and (illegal_count / total_requests) >= 0.8:
        for request in request_log:
            if is_illegal(request):
                banned_ips.add(request['target_ip'])

    # Appliquer les bannissements à toutes les IPs identifiées
    ban_ips(banned_ips)
```

Cette méthode alternative évalue si une IP doit être bannie en fonction du pourcentage de requêtes illégales. Si 80% ou plus des requêtes d'une IP sont illégales, elle est bannie.

Étape 6 : Vérification Améliorée des Requêtes Illégales (par exemple, Détection des Protocoles Shadowsocks et Trojan)

```
def is_illegal(request):
    # Vérifier si la requête utilise le protocole Shadowsocks (le corps contient des données de type bi
    if request['target_port'] == 443:
        if is_trojan(request):
            return True
    elif is_shadowsocks(request):
        return True
    return False
```

La fonction `is_illegal` vérifie maintenant également des protocoles spécifiques comme Shadowsocks et Trojan : - **Shadowsocks** : Nous pourrions vérifier la présence de données chiffrées ou de type binaire dans le corps de la requête. - **Trojan** : Si la requête arrive sur le port 443 (HTTPS) et correspond à des motifs spécifiques (par exemple, des caractéristiques du trafic Trojan), elle est signalée comme illégale.

Étape 7 : Exemple de Requêtes Légales

Par exemple, les requêtes comme `GET https://some-domain.xyz/bandwidth` sont sûrement légales et ne déclencheront pas le mécanisme de bannissement.

Étape 8 : Caractéristiques du Trafic des Serveurs Proxy

Les serveurs proxy ont des caractéristiques de trafic très différentes par rapport aux serveurs web ou API réguliers. Le GFW doit distinguer entre le trafic normal d'un serveur web et le trafic d'un serveur proxy, qui peut sembler complètement différent.

Étape 9 : Modèles d'Apprentissage Automatique et d'IA pour une Détection Intelligente

Étant donné la large gamme de requêtes et de réponses qui transitent par Internet, le GFW pourrait employer des modèles d'IA et d'apprentissage automatique pour analyser les modèles de trafic et détecter intelligemment les comportements illégaux. En entraînant le système sur une variété de types de trafic et en utilisant des techniques avancées, il pourrait bannir ou filtrer plus efficacement le trafic en fonction des modèles observés.

Mise à Jour

Malgré mes efforts, mon serveur proxy continue d'être banni. Pour atténuer cela, j'ai mis en place une solution de contournement en utilisant la fonction d'IP inversée de Digital Ocean, qui me permet d'attribuer rapidement une nouvelle adresse IP chaque fois qu'un bannissement se produit.