Análisis de la Prohibición del Servidor Proxy

¿Pueden las API en servidores proxy evitar las prohibiciones del Gran Cortafuegos (GFW)?

Ejecuto un servidor simple en mi instancia de Shadowsocks con el siguiente código:

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

app = Flask(__name__)
CORS(app)  # Habilitar CORS para todas las rutas

@app.route('/bandwidth', methods=['GET'])
def get_bandwidth():
    # Ejecutar el comando vnstat para obtener las estadísticas de tráfico en intervalos de 5 minutos pa
    result = subprocess.run(['vnstat', '-i', 'eth0', '-5', '--json'], capture_output=True, text=True)
    data = result.stdout

    # Devolver los datos capturados como una respuesta JSON
    return jsonify(data)

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

Y uso nginx para servir el puerto 443 como se muestra a continuación:

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

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

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

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

Este programa de servidor proporciona datos de red, y uso el servidor como mi servidor proxy, lo que me permite mostrar mi estado en línea en mi blog utilizando los datos de red.

Lo interesante es que el servidor no ha sido prohibido por el Gran Cortafuegos (GFW) ni por ningún otro sistema de control de red durante varios días. Normalmente, el servidor proxy que configuro sería prohibido en uno o dos días. El servidor ejecuta un programa de Shadowsocks en un puerto como el 51939, por lo que opera con tráfico de Shadowsocks mezclado con tráfico de API regular. Esta mezcla parece llevar al GFW a creer que el servidor no es un proxy dedicado, sino un servidor normal, evitando que prohíba la IP.

Esta observación es intrigante. Parece que el GFW utiliza una lógica específica para diferenciar el tráfico de proxy del tráfico regular. Mientras que muchos sitios web como Twitter y YouTube están bloqueados en China, numerosos sitios web extranjeros, como los de universidades y empresas internacionales, siguen siendo accesibles.

Esto sugiere que el GFW probablemente opera en base a reglas que distinguen entre el tráfico HTTP/HTTPS normal y el tráfico relacionado con proxies. Los servidores que manejan ambos tipos de tráfico parecen evitar las prohibiciones, mientras que los servidores que manejan solo tráfico de proxy tienen más probabilidades de ser bloqueados.

Una pregunta es qué rango de tiempo utiliza el GFW para acumular datos para prohibir, ya sea un día o una hora. Durante este rango de tiempo, detecta si el tráfico es exclusivamente de un proxy. Si lo es, la IP del servidor es prohibida.

A menudo visito mi blog para revisar lo que he escrito, pero en las próximas semanas, mi enfoque se centrará en otras tareas en lugar de escribir publicaciones en el blog. Esto reducirá mi acceso a la API de `bandwidth` a través del puerto 443. Si descubro que me prohíben nuevamente, debería escribir un programa para acceder regularmente a esta API y engañar al GFW.

Aquí está la versión refinada de tu texto con una estructura y claridad mejoradas:

Cómo funciona el Gran Cortafuegos (GFW).

Paso 1: Registro de Solicitudes

```
import time

# Base de datos para almacenar datos de solicitudes
request_log = []

# Función para registrar solicitudes
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 función `log_request` registra las solicitudes entrantes con información esencial como la IP de origen, la IP de destino, el puerto de destino, el cuerpo de la solicitud y la marca de tiempo.

Paso 2: Verificación y Prohibición de IPs

```
# Función para verificar solicitudes y prohibir IPs
def check_and_ban_ips():
    banned_ips = set()

    # Iterar sobre todas las solicitudes registradas
    for request in request_log:
        if is_illegal(request):
            banned_ips.add(request['target_ip'])
        else:
            banned_ips.discard(request['target_ip'])

    # Aplicar prohibiciones a todas las IPs identificadas
    ban_ips(banned_ips)
```

La función `check_and_ban_ips` itera a través de todas las solicitudes registradas, identificando y prohibiendo las IPs asociadas con actividades ilegales.

Paso 3: Definición de lo que Hace que una Solicitud sea Ilegal

```
# Función para simular la verificación de si una solicitud es ilegal
def is_illegal(request):
    # Marcador de posición para la lógica real de verificación de solicitudes ilegales
    # Por ejemplo, verificar el cuerpo de la solicitud o el destino
    return "ilegal" in request['body']
```

Aquí, `is_illegal` verifica si el cuerpo de la solicitud contiene la palabra "ilegal". Esto puede ampliarse a una lógica más sofisticada dependiendo de lo que constituya una actividad ilegal.

Paso 4: Prohibición de IPs Identificadas

```
# Función para prohibir una lista de IPs
def ban_ips(ip_set):
    for ip in ip_set:
        print(f"Prohibiendo IP: {ip}")
```

Una vez que se identifican las IPs ilegales, la función `ban_ips` las prohíbe imprimiendo sus direcciones IP (o, en un sistema real, podría bloquearlas).

Paso 5: Método Alternativo para Verificar y Prohibir IPs Basado en el 80% de Solicitudes Ilegales

```
# Función para verificar solicitudes y prohibir IPs basado en el 80% de solicitudes ilegales
def check_and_ban_ips():
    banned_ips = set()
    illegal_count = 0
    total_requests = 0

    # Iterar sobre todas las solicitudes registradas
    for request in request_log:
        total_requests += 1
```

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

    # Si el 80% o más de las solicitudes son ilegales, prohibir esas 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'])

    # Aplicar prohibiciones a todas las IPs identificadas
    ban_ips(banned_ips)
```

Este método alternativo evalúa si una IP debe ser prohibida en función del porcentaje de solicitudes ilegales. Si el 80% o más de las solicitudes de una IP son ilegales, se prohíbe.

Paso 6: Verificación Mejorada de Solicitudes Ilegales (por ejemplo, Detección de Protocolos Shadowsocks y Trojan)

```
def is_illegal(request):
    # Verificar si la solicitud utiliza el protocolo Shadowsocks (el cuerpo contiene datos binarios)
    if request['target_port'] == 443:
        if is_trojan(request):
            return True
    elif is_shadowsocks(request):
        return True
    return False
```

La función `is_illegal` ahora también verifica protocolos específicos como Shadowsocks y Trojan: - **Shadowsocks**: Podríamos verificar si el cuerpo de la solicitud contiene datos cifrados o binarios. - **Trojan**: Si la solicitud llega a través del puerto 443 (HTTPS) y coincide con patrones específicos (por ejemplo, características del tráfico de Trojan), se marca como ilegal.

Paso 7: Ejemplo de Solicitudes Legales

Por ejemplo, solicitudes como `GET https://some-domain.xyz/bandwidth` son seguramente legales y no activarán el mecanismo de prohibición.

Paso 8: Características del Tráfico de Servidores Proxy

Los servidores proxy tienen características de tráfico muy diferentes en comparación con los servidores web o API regulares. El GFW necesita distinguir entre el tráfico de un servidor web normal y el tráfico de un servidor proxy, que puede parecer completamente diferente.

Paso 9: Modelos de Aprendizaje Automático e IA para Detección Inteligente

Dada la amplia gama de solicitudes y respuestas que pasan a través de Internet, el GFW podría emplear modelos de IA y aprendizaje automático para analizar patrones de tráfico y detectar de manera inteligente comportamientos ilegales. Al entrenar el sistema en una variedad de tipos de tráfico y utilizar técnicas avanzadas, podría prohibir o filtrar el tráfico de manera más efectiva en función de los patrones observados.

Actualización

A pesar de mis esfuerzos, mi servidor proxy sigue siendo prohibido. Para mitigar esto, he implementado una solución alternativa utilizando la función de IP inversa de Digital Ocean, que me permite asignar rápidamente una nueva dirección IP cada vez que ocurre una prohibición.