代理服务器封锁分析

代理伺服器中的 API 能否避免 GFW 封禁？

我在我的 Shadowsocks 實例上運行了一個簡單的伺服器，代碼如下：

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

app = Flask(__name__)
CORS(app)  # 為所有路由啟用 CORS

@app.route('/bandwidth', methods=['GET'])
def get_bandwidth():
    # 運行 vnstat 命令以獲取 eth0 的 5 分鐘間隔流量統計數據
    result = subprocess.run(['vnstat', '-i', 'eth0', '-5', '--json'], capture_output=True, text=True)
    data = result.stdout

    # 將捕獲的數據作為 JSON 響應返回
    return jsonify(data)

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

並且我使用 nginx 來服務 443 端口，如下所示：

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

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

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

```
    }
}
```

這個伺服器程序提供網絡數據，我將該伺服器用作我的代理伺服器，使我能夠使用網絡數據在我的博客上顯示我的在線狀態。

有趣的是，這個伺服器已經幾天沒有被防火長城（GFW）或其他網絡控制系統封禁了。通常，我設置的代理伺服器會在一兩天內被封禁。該伺服器在像 51939 這樣的端口上運行 Shadowsocks 程序，因此它運行的是 Shadowsocks 流量與常規 API 流量的混合。這種混合似乎讓 GFW 認為該伺服器不是專用代理，而是一個正常伺服器，從而避免封禁 IP。

這個觀察很有趣。似乎 GFW 使用特定的邏輯來區分代理流量和常規流量。雖然許多網站如 Twitter 和 YouTube 在中國被屏蔽，但許多外國網站——如國際大學和公司的網站——仍然可以訪問。

這表明 GFW 可能基於區分正常 HTTP/HTTPS 流量和代理相關流量的規則來運作。處理這兩種類型流量的伺服器似乎可以避免封禁，而僅處理代理流量的伺服器更有可能被封禁。

一個問題是 GFW 使用多長時間範圍來累積數據以進行封禁——是一天還是一小時。在這個時間範圍內，它檢測流量是否完全來自代理。如果是，則伺服器的 IP 會被封禁。

我經常訪問我的博客來查看我寫的內容，但在接下來的幾週內，我的注意力將轉移到其他任務上，而不是寫博客文章。這將減少我通過 443 端口訪問 `bandwidth` API 的次數。如果我發現再次被封禁，我應該編寫一個程序來定期訪問這個 API 以欺騙 GFW。

以下是改進後的文本版本，結構更清晰，表達更明確：

防火長城（GFW）的工作原理。

第一步：記錄請求

```
import time

# 存儲請求數據的數據庫
request_log = []

# 記錄請求的函數
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()
    })
```

`log_request` 函數記錄了傳入請求的基本信息，如源 IP、目標 IP、目標端口、請求主體和時間戳。

第二步：檢查並封禁 IP

```
# 檢查請求並封禁 IP 的函數
def check_and_ban_ips():
    banned_ips = set()

    # 遍歷所有記錄的請求
    for request in request_log:
        if is_illegal(request):
            banned_ips.add(request['target_ip'])
        else:
            banned_ips.discard(request['target_ip'])

    # 對所有識別的 IP 應用封禁
    ban_ips(banned_ips)
```

`check_and_ban_ips` 函數遍歷所有記錄的請求，識別並封禁與非法活動相關的 IP。

第三步：定義什麼是非法請求

```
# 模擬檢查請求是否非法的函數
def is_illegal(request):
    # 實際非法請求檢查邏輯的佔位符
    # 例如，檢查請求主體或目標
    return "illegal" in request['body']
```

在這裡，`is_illegal` 檢查請求主體是否包含單詞 “illegal” 。這可以根據什麼構成非法活動擴展到更複雜的邏輯。

第四步：封禁識別的 IP

```
# 封禁 IP 列表的函數
def ban_ips(ip_set):
    for ip in ip_set:
        print(f" 封禁 IP: {ip}")
```

一旦識別出非法 IP，ban_ips 函數通過打印其 IP 地址（或在真實系統中，可以阻止它們）來封禁它們。

第五步：基於 80% 非法請求的檢查和封禁 IP 的替代方法

```
# 基於 80% 非法請求檢查並封禁 IP 的函數
def check_and_ban_ips():
    banned_ips = set()
    illegal_count = 0
    total_requests = 0

    # 遍歷所有記錄的請求
    for request in request_log:
        total_requests += 1
        if is_illegal(request):
            illegal_count += 1

    # 如果 80% 或更多的請求是非法的，則封禁這些 IP
    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'])

    # 對所有識別的 IP 應用封禁
    ban_ips(banned_ips)
```

這種替代方法根據非法請求的百分比評估是否應封禁 IP。如果來自某個 IP 的 80% 或更多的請求是非法的，則封禁該 IP。

第六步：增強非法請求檢查（例如，Shadowsocks 和 Trojan 協議檢測）

```
def is_illegal(request):
    # 檢查請求是否使用 Shadowsocks 協議（主體包含類似二進制的數據）
    if request['target_port'] == 443:
        if is_trojan(request):
            return True
    elif is_shadowsocks(request):
        return True
    return False
```

`is_illegal` 函數現在還檢查特定協議，如 Shadowsocks 和 Trojan：- **Shadowsocks**：我們可能會檢查請求主體中的加密或類似二進制的數據。- **Trojan**：如果請求通過 443 端口（HTTPS）傳入並匹配特定模式（例如，Trojan 流量特徵），則標記為非法。

第七步：合法請求示例

例如，像 `GET https://some-domain.xyz/bandwidth` 這樣的請求肯定是合法的，不會觸發封禁機制。

第八步：代理伺服器流量特徵

代理伺服器的流量特徵與常規 Web 或 API 伺服器非常不同。GFW 需要區分正常 Web 伺服器流量和代理伺服器流量，這兩者看起來可能完全不同。

第九步：用於智能檢測的機器學習和 AI 模型

考慮到通過互聯網傳遞的各種請求和響應，GFW 可以採用 AI 和機器學習模型來分析流量模式並智能檢測非法行為。通過在各種流量類型上訓練系統並使用先進技術，它可以更有效地根據觀察到的模式封禁或過濾流量。