प्रॉक्सी सर्वर प्रतिबंध का विश्लेषण

क्या प्रॉक्सी सर्वर में □□□ ग्रेट फ़ायरवॉल (□□□) बैन से बच सकते हैं?

मैंने अपने □□□□□□□□□□□□ इंस्टेंस पर एक सरल सर्वर चलाया है, जिसका कोड निम्नलिखित है:

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

app = Flask(__name__)
CORS(app)  #          CORS

@app.route('/bandwidth', methods=['GET'])
def get_bandwidth():
    # eth0     5-                          vnstat
    result = subprocess.run(['vnstat', '-i', 'eth0', '-5', '--json'], capture_output=True, text=True)
    data = result.stdout

    #             JSON
    return jsonify(data)

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

और मैं □□□□□ का उपयोग करके पोर्ट 443 को सर्व करता हूं, जैसा कि नीचे दिखाया गया है:

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

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

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

```
    }
}
```

यह सर्वर प्रोग्राम नेटवर्क डेटा प्रदान करता है, और मैं इस सर्वर को अपने प्रॉक्सी सर्वर के रूप में उपयोग करता हूं, जिससे मैं नेटवर्क डेटा का उपयोग करके अपने ब्लॉग पर अपनी ऑनलाइन स्थिति प्रदर्शित कर सकता हूं।

दिलचस्प बात यह है कि यह सर्वर कई दिनों से ग्रेट फ़ायरवॉल (□□□) या किसी अन्य नेटवर्क नियंत्रण प्रणाली द्वारा बैन नहीं किया गया है। आमतौर पर, मेरे द्वारा सेट किया गया प्रॉक्सी सर्वर एक या दो दिनों के भीतर बैन हो जाता है। सर्वर 51939 जैसे पोर्ट पर □□□□□□□□□□□ प्रोग्राम चलाता है, इसलिए यह □□□□□□□□□□□ ट्रैफ़िक के साथ नियमित □□□ ट्रैफ़िक को मिलाकर चलता है। यह मिश्रण □□□ को यह विश्वास दिलाता है कि सर्वर एक समर्पित प्रॉक्सी नहीं है, बल्कि एक सामान्य सर्वर है, जिससे यह □□ को बैन नहीं करता है।

यह अवलोकन दिलचस्प है। ऐसा लगता है कि □□□ प्रॉक्सी ट्रैफ़िक और नियमित ट्रैफ़िक के बीच अंतर करने के लिए विशिष्ट तर्क का उपयोग करता है। जबकि चीन में □□□□□□□ और □□□□□□□ जैसी कई वेबसाइट्स ब्लॉक हैं, कई विदेशी वेबसाइट्स—जैसे कि अंतरराष्ट्रीय विश्वविद्यालयों और कंपनियों की वेबसाइट्स—अभी भी सुलभ हैं।

इससे पता चलता है कि □□□ संभवतः नियमित □□□□/□□□□□ ट्रैफ़िक और प्रॉक्सी-संबंधित ट्रैफ़िक के बीच अंतर करने के नियमों पर काम करता है। जो सर्वर दोनों प्रकार के ट्रैफ़िक को संभालते हैं, वे बैन से बच जाते हैं, जबकि केवल प्रॉक्सी ट्रैफ़िक संभालने वाले सर्वर अधिक संभावना से ब्लॉक हो जाते हैं।

एक सवाल यह है कि □□□ बैन के लिए डेटा एकत्र करने के लिए किस समय सीमा का उपयोग करता है—चाहे वह एक दिन हो या एक घंटा। इस समय सीमा के दौरान, यह पता लगाता है कि क्या ट्रैफ़िक विशेष रूप से प्रॉक्सी से है। यदि हां, तो सर्वर का □□ बैन हो जाता है।

मैं अक्सर अपने ब्लॉग पर जाकर यह समीक्षा करता हूं कि मैंने क्या लिखा है, लेकिन आने वाले हफ्तों में मेरा ध्यान ब्लॉग पोस्ट लिखने के बजाय अन्य कार्यों पर होगा। इससे पोर्ट 443 के माध्यम से `bandwidth` □□□ तक मेरी पहुंच कम हो जाएगी। यदि मुझे लगता है कि मैं फिर से बैन हो गया हूं, तो मुझे □□□ को धोखा देने के लिए इस □□□ को नियमित रूप से एक्सेस करने के लिए एक प्रोग्राम लिखना चाहिए।

यहां आपके टेक्स्ट का संशोधित संस्करण है जिसमें संरचना और स्पष्टता में सुधार किया गया है:

ग्रेट फ़ायरवॉल (□□□) कैसे काम करता है।

चरण 1: अनुरोधों को लॉग करना

```
import time

#
request_log = []

#
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` फ़ंक्शन स्रोत □□, लक्ष्य □□, लक्ष्य पोर्ट, अनुरोध बॉडी और टाइमस्टैम्प जैसी आवश्यक जानकारी के साथ आने वाले अनुरोधों को रिकॉर्ड करता है।

चरण 2: □□□ की जांच और बैन करना

```
#              IPs
def check_and_ban_ips():
    banned_ips = set()

    #
    for request in request_log:
        if is_illegal(request):
            banned_ips.add(request['target_ip'])
        else:
            banned_ips.discard(request['target_ip'])

    #         IPs
    ban_ips(banned_ips)
```

`check_and_ban_ips` फ़ंक्शन सभी लॉग किए गए अनुरोधों पर पुनरावृत्ति करता है, और अवैध गतिविधि से जुड़े □□□ की पहचान करता है और उन्हें बैन करता है।

चरण 3: अनुरोध को अवैध बनाने वाले कारकों को परिभाषित करना

```
#
def is_illegal(request):
    #
    #        ,
    return "illegal" in request['body']
```

यहां, `is_illegal` जांचता है कि क्या अनुरोध बॉडी में "□□□□□□□□" शब्द है। इसे अवैध गतिविधि के आधार पर अधिक परिष्कृत तर्क में विस्तारित किया जा सकता है।

चरण 4: पहचाने गए □□□ को बैन करना

```
# IPs
def ban_ips(ip_set):
    for ip in ip_set:
        print(f"IP        : {ip}")
```

एक बार अवैध □□□ की पहचान हो जाने पर, `ban_ips` फ़ंक्शन उन्हें बैन करता है (या, वास्तविक सिस्टम में, उन्हें ब्लॉक कर सकता है)।

चरण 5: 80% अवैध अनुरोधों के आधार पर □□□ की जांच और बैन करने के लिए वैकल्पिक विधि

```
# 80%                          IPs
def check_and_ban_ips():
    banned_ips = set()
    illegal_count = 0
    total_requests = 0

    #
    for request in request_log:
        total_requests += 1
        if is_illegal(request):
            illegal_count += 1

    #    80%             ,     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'])

    #         IPs
    ban_ips(banned_ips)
```

यह वैकल्पिक विधि □□□ को बैन करने के लिए अवैध अनुरोधों के प्रतिशत का मूल्यांकन करती है। यदि किसी □□ से 80% या अधिक अनुरोध अवैध हैं, तो उसे बैन कर दिया जाता है।

चरण 6: अवैध अनुरोध जांच को बढ़ाना (उदाहरण के लिए, □□□□□□□□□□□□ और □□□□□□□ प्रोटोकॉल का पता लगाना)

```
def is_illegal(request):
    #            Shadowsocks                (        -        )
    if request['target_port'] == 443:
        if is_trojan(request):
            return True
    elif is_shadowsocks(request):
        return True
    return False
```

`is_illegal` फ़ंक्शन अब □□□□□□□□□□□□ और □□□□□□□ जैसे विशिष्ट प्रोटोकॉल की भी जांच करता है: - □□□□□□□□□□□□: हम अनुरोध बॉडी में एन्क्रिप्टेड या बाइनरी-जैसा डेटा की जांच कर सकते हैं। - □□□□□□□: यदि अनुरोध पोर्ट 443 (□□□□□□) पर आता है और विशिष्ट पैटर्न से मेल खाता है (उदाहरण के लिए, □□□□□□□ ट्रैफ़िक की विशेषताएं), तो इसे अवैध माना जाता है।

चरण 7: वैध अनुरोध उदाहरण

उदाहरण के लिए, `GET https://some-domain.xyz/bandwidth` जैसे अनुरोध निश्चित रूप से वैध हैं और बैन तंत्र को ट्रिगर नहीं करेंगे।

चरण 8: प्रॉक्सी सर्वर ट्रैफ़िक की विशेषताएं

प्रॉक्सी सर्वर में नियमित वेब या □□□ सर्वर की तुलना में बहुत अलग ट्रैफ़िक विशेषताएं होती हैं। □□□ को नियमित वेब सर्वर ट्रैफ़िक और प्रॉक्सी सर्वर ट्रैफ़िक के बीच अंतर करने की आवश्यकता होती है, जो पूरी तरह से अलग दिख सकते हैं।

चरण 9: स्मार्ट पहचान के लिए मशीन लर्निंग और □□ मॉडल

इंटरनेट के माध्यम से गुजरने वाले अनुरोधों और प्रतिक्रियाओं की विस्तृत श्रृंखला को देखते हुए, □□□ ट्रैफ़िक पैटर्न का विश्लेषण करने और अवैध व्यवहार का स्मार्ट तरीके से पता लगाने के लिए □□ और मशीन लर्निंग मॉडल का उपयोग कर सकता है। विभिन्न प्रकार के ट्रैफ़िक पर सिस्टम को प्रशिक्षित करके और उन्नत तकनीकों का उपयोग करके, यह देखे गए पैटर्न के आधार पर ट्रैफ़िक को अधिक प्रभावी ढंग से बैन या फ़िल्टर कर सकता है।

अपडेट

मेरे प्रयासों के बावजूद, मेरा प्रॉक्सी सर्वर लगातार बैन हो रहा है। इसे कम करने के लिए, मैंने □□□□□□□□ □□□□□□ के उलटे □□ फ़ीचर का उपयोग करके एक वर्कअराउंड लागू किया है, जो मुझे जब भी बैन होता है तो तुरंत एक नया □□ पता असाइन करने की अनुमति देता है।