🛠️ Custom HTTPS Request Tool with TLS Fingerprint Spoofing and Anti-Bot Evasion

🎯 Objective

Build a minimal yet robust Go-based tool that performs HTTPS requests designed to **bypass sophisticated anti-bot protections** (such as Cloudflare, PerimeterX, Akamai) by mimicking legitimate browser behavior across multiple layers:

- Spoof TLS fingerprints (ClientHello) to resemble popular browsers like Chrome, Firefox, or Safari
- Mimic browser HTTP headers consistent with the selected TLS profile
- Support proxy usage with reliable proxy rotation, including both HTTP and SOCKS5 proxies
- Detect and respond to anti-bot challenges, including HTTP status codes (e.g., 403), CAPTCHA pages, and redirect-based challenges
- Manage cookies and HTTP redirects similarly to a real browser session
- Implement timing randomization between requests to avoid behavioral fingerprinting
- Avoid any use of headless browsers or browser automation frameworks (e.g., Puppeteer, Playwright, ChromeDP)

The tool should be configurable, stable during continuous use, and provide minimal but clear logging of request success, failure, and detection events.

🔧 Core Technical Requirements

TLS Fingerprint Spoofing

- Use [uTLS] or an equivalent for **custom ClientHello** construction

- Support at least **3 preset TLS profiles** (e.g., Chrome, Firefox, Safari)
- Support **randomized fingerprint** selection per request
- **Optional**: Allow manual construction of a ClientHello (custom cipher suites, extensions, ALPN, etc.)
- Log or hash **JA3 fingerprints** per request (optional)

Proxy Support

- All requests must be made **through proxies**
- Proxies are provided in `user:pass@ip:port` format and can be **HTTP or SOCKS5**
- Implement **proxy rotation**:
    - Rotate per request
    - Or rotate on timeout/error/block
- Validate proxies before use (optional)

Request Logic

- Support **basic HTTPS GET requests**
- Implement logic to detect:
    - Success (e.g., `200 OK`)
    - Blocks or challenges (`403`, CAPTCHA in body, redirects to challenge pages)
- If blocked or timed out:
    - Retry with a new proxy and TLS fingerprint

- Support configurable inputs:
    - Target URL(s)
    - Proxy list
    - Number of requests
    - TLS profile/randomization mode
    - Delay range between requests

🛡️ Extended Detection Evasion Features

To effectively bypass layered detection systems, the following **enhanced anti-bot techniques** must be implemented:

Header Mimicry

- Construct browser-consistent headers:
    - `User-Agent`, `Accept`, `Accept-Encoding`, `Accept-Language`, `Sec-CH-UA`, etc.
- Match headers to the TLS profile (e.g., Chrome TLS + Chrome headers)
- Allow header customization via config

Cookie Handling & Redirects

- Maintain a simple cookie jar (in-memory, per session or per proxy)
- Optionally follow HTTP redirects (e.g., 302s to challenge pages)
- Support toggling cookie persistence on/off

Challenge Detection (Body Heuristics)

- Detect bot blocks or CAPTCHA challenges by inspecting response body
    - Look for markers like `cf-challenge`, `g-recaptcha`, `verify you're human`, etc.
- Categorize responses as `Success`, `Blocked`, or `Challenged`

HTTP/2 & ALPN Support

- Use **HTTP/2** where negotiated via ALPN
- Allow fallback to HTTP/1.1 if unsupported
- Respect ALPN negotiation based on TLS fingerprint

JA3 Awareness

- Log or calculate **JA3 hashes** for each request (optional)
- Allow user to view or configure known benign JA3 profiles

Fingerprint Pools

- Maintain a rotating pool of TLS+header fingerprints
- Randomly select from the pool per request or per session
- Allow adding/removing profiles via config

Timing Randomization

- Add jitter/delay between requests to mimic human behavior
- User-configurable delay range (e.g., `500ms–3000ms`)

📦 Implementation Requirements

- Language: **Go 1.20+**
- Tool can be a **CLI app**, **library**, **daemon**, or **script**
- Must handle at least **50 consecutive requests** reliably
    - No memory leaks
    - No crashes or hangs
- Minimal output/logging showing:
    - Request success/failure
    - Blocked/challenged status
    - Proxy used
    - (Optional) TLS fingerprint or JA3 hash used

🔌 Input Data for Testing

Proxy list format:

ruby
CopyEdit
`user:pass@ip:port`

Sample target URLs:

- https://www.viagogo.co.uk/Concert-Tickets/Rock-and-Pop/Sting-Tickets/E-157332132
- https://www.viagogo.co.uk/Concert-Tickets/Alternative-and-Indie/Coldplay-Tickets/E-155391504
- https://www.stubhub.com/stardew-valley-denver-tickets-9-13-2025/event/156264784
- https://www.viagogo.com/Concert-Tickets/Alternative-Music/Coldplay-Tickets/E-155741198

✅ Completion Criteria

- Requests successfully routed through rotating proxies
- TLS fingerprints are randomized or matched to known browsers
- Browser-like headers are used
- Challenges and blocks are correctly detected and retried
- No browser/headless automation frameworks are used
- Tool performs at least 50 sequential requests without instability
- Clean, maintainable Go code
- README or usage instructions (minimal CLI documentation)

💡 Notes

- Developer has full freedom over architecture and internal tooling
- Creative approaches to detection evasion are encouraged

- Focus is on stability and stealth, not flashy output