GeekSTunnel: Micro-Level Architecture & Engineering Deep Dive

1. Introduction

This document provides an exhaustive, micro-level technical analysis of the GeekSTunnel VPN Control Plane. It explores the "how" and "why" behind every design decision, from kernel-level packet manipulation to high-level application security.

---

2. The VPN Kernel: WireGuard Deep Dive

2.1 Key Exchange & Cryptography

WireGuard uses **Noise Protocol Framework** for its handshake. - **Micro-Level**: When a user is created, we generate a Curve25519 keypair. The Private Key is used by the client to sign handshakes, and the Public Key is stored on the server to identify the peer. - **Reasoning**: Unlike OpenVPN/IPsec, WireGuard is "stealthy." It does not respond to any packets unless they are correctly signed by a known peer's public key, making the server invisible to port scanners.

2.2 Atomic Configuration Management (`wg.py`)

- **The Process**: When adding a user, we don't just append to `wg0.conf`. We use **File Locking** (`fcntl.flock`) to prevent race conditions and **Atomic Writes** (writing to a `.tmp` file and then `os.rename`).
- **The Sync**: We use `wg syncconf` instead of `wg-quick down/up`.
- **Micro-Level**: `wg syncconf` reads the new config and calculates the "delta" (difference) between the current kernel state and the file. It then only adds/removes the necessary peers without dropping existing connections.
- **Reasoning**: This ensures **Zero Downtime**. If a reload fails, the system automatically rolls back to the `.conf.bak` file.

2.3 The "Homeostatic" Sync (Immune System)

- **The Process**: A background worker periodically compares the **Kernel State** (`wg show dump`) with the **Database State**.
- **Micro-Level**: If a peer exists in the kernel but is marked as `disabled` or is missing from the DB, the system executes `wg set wg0 peer [KEY] remove`.
- **Reasoning**: This prevents "Zombie Peers"—unauthorized users who might have remained in the kernel due to a system crash or manual tampering.

---

3. Networking & Netfilter (iptables) Engineering

3.1 The Packet Lifecycle

When a packet arrives from a VPN client (`10.50.0.x`), it undergoes several transformations:

A. PREROUTING (The Hijack)

- **Rule**: `iptables -t nat -I PREROUTING -i wg0 -p udp --dport 53 -j DNAT --to-destination 10.50.0.1:53`
- **Micro-Level**: This rule intercepts packets *before* the routing decision is made. Even if a client manually sets their DNS to `8.8.8.8`, the kernel overwrites the destination IP to our internal CoreDNS.
- **Reasoning**: This is "Network-Level Enforcement." It is impossible for the client to bypass our DNS filtering at the OS level.

B. FORWARD (The ACL)

- **Rule**: `iptables -A VPN_ACL -s 10.50.0.5 -d 192.168.1.0/24 -j DROP`
- **Micro-Level**: The `VPN_ACL` chain is checked for every packet traversing the server.
- **Reasoning**: This implements **Micro-Segmentation**. We can isolate users from the local network (Intranet-Only vs. Internet-Only) without changing the VPN configuration itself.

C. POSTROUTING (The NAT)

- **Rule**: `iptables -t nat -A POSTROUTING -s 10.50.0.0/24 -j MASQUERADE`
- **Micro-Level**: The kernel replaces the client's internal IP (`10.50.0.x`) with the server's public IP.
- **Reasoning**: This allows the client to access the public internet using the server's identity.

3.2 MSS Clamping (The "Hanging Website" Fix)

- **Rule**: `iptables -t mangle -A FORWARD -p tcp --tcp-flags SYN,RST SYN -j TCPMSS --clamp-mss-to-pmtu`
- **Micro-Level**: WireGuard adds an 80-byte header to every packet. If a client sends a standard 1500-byte packet, it becomes 1580 bytes, which exceeds the standard MTU and gets dropped. MSS Clamping forces the TCP handshake to negotiate a smaller segment size (usually 1380 bytes).
- **Reasoning**: Without this, many websites (like Netflix or Google) would simply hang or fail to load over the VPN.

---

4. DNS Engine: CoreDNS & Anti-Bypass

4.1 The CoreDNS Pipeline

CoreDNS uses a chain of plugins to process a query: 1. `hosts`: Checks `blocked.hosts`. If found, returns `0.0.0.0`. 2. `template`: Handles wildcards (e.g., `*.doubleclick.net`). 3. `forward`: If not blocked, forwards the query to upstream providers (Cloudflare/Google) over **DNS-over-TLS** for privacy.

4.2 Anti-Bypass Mechanics

Modern browsers use **DNS-over-HTTPS (DoH)** to bypass local DNS. We counter this at two levels: 1. **Domain Level**: We return `NXDOMAIN` for "Canary Domains" like `use-application-dns.net`. When Firefox sees this, it automatically disables DoH and falls back to system DNS (which we hijack). 2. **IP Level**: We block port 443 (HTTPS) traffic to known DoH provider IPs (e.g., `8.8.8.8`, `1.1.1.1`). This forces the browser to time out on DoH and use standard DNS.

---

5. Application Security & Auth Logic

5.1 Session Integrity

- **Signed Cookies**: We use `itsdangerous` to sign session cookies.
- **Micro-Level**: The cookie contains the `admin_id` and a `timestamp`. If the server's `SECRET_KEY` doesn't match the signature, or if the timestamp is older than `SESSION_MAX_AGE`, the session is rejected.
- **Reasoning**: This prevents session tampering and replay attacks.

5.2 CSRF Protection (The Double-Submit Pattern)

- **The Process**:
  1. Frontend fetches a token from `/auth/csrf`.
  2. Backend sets a `csrf_token` cookie.
  3. Frontend sends the token in the `X-CSRF-Token` header.
  4. Backend compares the header value with the cookie value.
- **Reasoning**: This ensures that requests can only be made from our own frontend, protecting against malicious sites trying to trigger actions in the admin's browser.

5.3 WebSocket Authentication

- **Micro-Level**: WebSockets start as an HTTP request. We intercept this "Upgrade" request and validate the session cookie *before* allowing the connection to switch to the binary WebSocket protocol.
- **Reasoning**: This prevents unauthorized users from streaming real-time system metrics.

---

6. Database Schema & Persistence

- **`admins`**: Stores hashed passwords and TOTP secrets.
- **`users`**: Stores the "Source of Truth" for the VPN mesh (Public Keys, Assigned IPs, ACL Profiles).
- **`audit_logs`**: Tracks every action (who created which user, when the firewall was reloaded).
- **`sessions`**: (Optional/Redis) Tracks active administrative sessions for instant revocation.

---

7. Deployment Initialization Sequence

1. **`init_db()`**: Sets up the relational schema.
2. **`init_firewall_chains()`**: Creates the `VPN_ACL` chain and hooks it into the kernel's `FORWARD` path.
3. **`sync_to_wildcards()`**: Generates the initial CoreDNS configuration.
4. **`sync_wireguard_state()`**: Reads the DB and populates the WireGuard kernel module.

Conclusion: GeekSTunnel is not just a UI; it is a sophisticated orchestration layer that manages the Linux kernel's networking subsystem to provide a secure, high-performance, and un-bypassable VPN environment.