ENTERPRISE VPN INTEGRATION

Automated WireGuard Access for an Active Directory Environment

Windows Server 2022 • Group Policy • Ubuntu Routing • WireGuard • PowerShell • Flask/Gunicorn

PROJECT SNAPSHOT **Designed and integrated an enterprise-style VPN control and data plane that automatically onboards domain-managed Windows endpoints, assigns per-client WireGuard identities, and routes only protected server-network traffic through the encrypted tunnel.**

3	1	GPO	SPLIT
NETWORK ZONES	**CENTRAL ENROLLMENT**	**AUTOMATED ONBOARDING**	**TUNNEL POLICY**
.56 client • .57 protected • 10.10.10 VPN	API-driven peer allocation and config generation	Install, enroll and start tunnel from SYSVOL	Only 192.168.57.0/24 enters WireGuard

OWNERSHIP What I Engineered

VPN architecture
Designed a segmented topology joining AD-managed clients, an Ubuntu VPN/router, and a protected Windows Server network.

Automated enrollment
Built the client-to-server flow for hostname/public-key registration, VPN IP allocation, configuration generation and peer activation.

Domain deployment
Integrated WireGuard onboarding with Group Policy and SYSVOL so domain endpoints can be provisioned centrally.

Split tunneling
Restricted client VPN routing to 192.168.57.0/24, keeping local .56 traffic and normal Internet traffic outside the tunnel.

Central operations
Implemented health endpoints, peer status, revoke/restore controls, audit data and handshake-based client visibility.

Systems integration
Connected Windows services, Linux routing, systemd/Gunicorn, Flask, WireGuard, PowerShell and NAT into one operational workflow.

DESIGN Architecture & End-to-End Flow

1	2	3	4	5
DOMAIN CLIENT	**GPO / POWERSHELL**	**ENROLLMENT API**	**WIREGUARD**	**PROTECTED NET**
192.168.56.x DHCP + AD policy	Install WireGuard Generate keys	Ubuntu :5000 Allocate 10.10.10.x	wg0 10.10.10.1 Encrypted tunnel	192.168.57.0/24 AD/DNS/resources

Traffic rule: **192.168.57.0/24 → WireGuard | 192.168.56.0/24 → local LAN | Internet → normal/default route**

IMPLEMENTATION Engineering Depth

CLIENT AUTOMATION

- Startup execution under LOCAL SYSTEM
- 64-bit PowerShell handling
- WireGuard MSI detection/install path
- Client key-pair generation with wg.exe
- Enrollment retry logic and health checks
- Restricted ACLs on generated tunnel config
- Automatic WireGuard tunnel service startup

UBUNTU CONTROL PLANE

- Flask enrollment API behind Gunicorn/systemd
- Persistent peer registry in peers.json
- Dynamic 10.10.10.x address allocation
- Server configuration regeneration per peer
- wg-quick service restart after peer changes
- Handshake/traffic status via wg show
- Revoke/restore and audit endpoints

NETWORKING Configuration I Had to Reason About

Component	Address / Port	Role	Key Engineering Point
Windows Server	192.168.56.106 192.168.57.106	AD / DNS / DHCP	Dual-homed server; direct-route behavior matters
Ubuntu	192.168.56.110 192.168.57.110	Router / VPN host	Forwards between client, protected and VPN networks
WireGuard	10.10.10.1/24 UDP 51820	Encrypted data plane	Per-peer tunnel IPs; protected subnet selected on client
Enrollment API	TCP 5000	Control plane	Health, enrollment, dashboard and peer operations

EVIDENCE Troubleshooting That Demonstrates Systems Thinking

DHCP / network conflict

Eliminated competing VirtualBox DHCP behavior so clients received the intended Windows DHCP configuration and could join the domain.

Layer-2/3 + AD dependency analysis

Runtime port conflict

Found multiple systemd/Gunicorn units competing for TCP 5000. Stopped/disabled obsolete services and retained the intended wg-enroll service.

Linux service ownership + socket diagnostics

Flask /health returned 404

Compared the imported Flask URL map with the process actually owning :5000, proving the source file and runtime service were not the same execution path.

Application/runtime isolation

PowerShell startup failure Traced a Test-Path/-and binding error from the deployment log and corrected the Boolean expression before the installer stage.	**PowerShell debugging under GPO**
MSI exit code 1603 Correctly treated 1603 as a symptom, not a root cause; improved the deployment path with local MSI staging and verbose /L*v logging for evidence-driven diagnosis.	**Windows Installer troubleshooting**

CAPABILITY Why This Project Matters

This was not a single-tool VPN exercise. It required coordinating identity, endpoint policy, Windows execution context, Linux routing, application services, VPN route semantics and NAT while preserving a clear boundary between the control plane and encrypted data plane.

CORE SKILLS DEMONSTRATED

Active Directory • DNS/DHCP • Group Policy • PowerShell • Windows Installer • Ubuntu/Linux • IP Routing • Split Tunneling • WireGuard • Flask • Gunicorn • systemd • NAT/Firewall Analysis • Troubleshooting

Portfolio positioning: Enterprise Infrastructure / Systems Engineering / Network & Security Engineering

FESTUS AWE COPY