Theory

Samba, Kerberos, configs, and how it all fits together — explained clearly

Quick TL;DR

Samba AD is Windows-style directory services (LDAP + DNS + AD data). Kerberos is the ticket-based authentication system AD uses. Samba provides the KDC, LDAP directory, and DNS when it runs as an AD DC. `sssd` + PAM + NSS let Linux processes (login, `getent`, `ssh`) consult AD for identity and authentication. WireGuard still uses keypairs for the wire; we enforce AD authentication at the management stage (who gets keys). `getent` is the standard system tool that asks the system name service (NSS) for user/group entries — it's the quick way to see what `nsswitch.conf` is returning.

Below I explain each piece, the important config files and what they *do*, the runtime flow (who talks to whom, in what order), and short troubleshooting/verification hints. I'll use plain language, small diagrams, and examples.

---

1) What Samba (as AD DC) actually is — plain and simple

- **Role:** Samba can act as an Active Directory Domain Controller (AD DC). In that mode it implements:
  - an **LDAP-compatible directory** (stores users, groups, attributes),
  - an internal **DNS** for AD service discovery (SRV records),
  - a **Kerberos KDC** (authentication tickets), and

    - Windows RPC/Netlogon interfaces that Windows clients expect.
- **Why it matters:** When Samba runs as an AD DC, it is the authoritative source for identity (who you are), authentication (prove it), and service discovery (where the Kerberos/LDAP services are).
- **Key storage:** Samba stores the directory in `/var/lib/samba` (and creates its own `smb.conf` and DB files there). It also creates a Kerberos keytab for service principals.

---

2) What Kerberos is — plain and simple

- **Concept:** Kerberos is a *ticket* system. Users log in once and get a Time-limited ticket (TGT). That ticket can be exchanged for service tickets (TGS) to access services (like LDAP, SMB, etc.) without re-entering a password each time.
- **Actors:**
    - **KDC (Key Distribution Center)** — issues tickets. On our setup this is provided by Samba AD.
    - **Client** — user or process that needs access (e.g., `ssh`, `sssd`).
    - **Service** — something that requires proof (e.g., LDAP, HTTP service).
- **Why use it:** Kerberos avoids sending passwords to every service and supports single sign-on inside the domain.

---

3) The important configuration files — full descriptions and what they do

I list the core files you'll see in the setup and explain each one's purpose and the important fields.

`/etc/samba/smb.conf` — Samba main config

What it is: Samba's configuration file. When Samba is provisioned as AD DC, `samba-tool domain provision` writes a complete `smb.conf` for AD operation.

Important responsibilities:

- Declares the **realm** and **domain** (e.g., `realm = VPN.LOCAL`, `netbios name = ...`).
- Configures **LDAP backend** or `SAMBA_INTERNAL` for directory storage.
- Configures **DNS backend** (internal or BIND9_DLZ).
- Controls Kerberos integration lines and service principals.

Why it matters: Samba reads this to know what role it is (AD DC), what DNS backend to run, where to store the DB, and what network services to expose.

Example (conceptual excerpt):

```
[global]
   netbios name = DC01
   realm = VPN.LOCAL
   server role = active directory domain controller
   idmap_ldb:use rfc2307 = yes
   dns forwarder = 8.8.8.8
```

`/etc/krb5.conf` — Kerberos client config

What it is: System-wide Kerberos configuration used by `kinit`, `ssh` (if using GSSAPI), and `sssd` Kerberos auth.

Important responsibilities:

- `default_realm` — which Kerberos realm to use by default (e.g., `VPN.LOCAL`).
- `realms` — mapping of realm → KDC/admin servers. For a single-host DC you often point to `localhost` or the server hostname.
- `domain_realm` — maps DNS domains to Kerberos realms.

Why it matters: If this is wrong, `kinit` and Kerberos-based authentication will fail (no TGTs).

Example:

```
[libdefaults]
 default_realm = VPN.LOCAL

[realms]
 VPN.LOCAL = {
  kdc = localhost
  admin_server = localhost
 }

[domain_realm]
 .vpn.local = VPN.LOCAL
 vpn.local = VPN.LOCAL
```

`/var/lib/samba/private/krb5.keytab` (and `/etc/krb5.keytab`) — Kerberos keytab

What it is: A binary file that stores Kerberos service principals and their secret keys (encrypted). Samba creates it during provisioning.

Important responsibilities:

- Contains the host and service principals required for the server to authenticate clients and services (e.g., `HOST/servername@REALM`, `ldap/servername@REALM`, etc.).
- Needed by `sssd` and other services to accept Kerberos-authenticated users or to perform KRB5 operations as services.

Why it matters: Without a correct keytab, services cannot validate Kerberos tickets or present themselves as services to clients.

`/etc/sssd/sssd.conf` — SSSD config (system identity/auth broker)

What it is: SSSD configuration that tells the system how to get identity and authentication information (AD via `id_provider = ad`, Kerberos settings, caching, home

directory fallback, etc.)

Important responsibilities:

- `domains` — which domains to consult (e.g., `VPN.LOCAL`).
- `id_provider`, `auth_provider` — tells sssd to use `ad` and `krb5`.
- `fallback_homedir` and `default_shell` — how to create home dirs and shells for AD users.
- `enumerate`, `cache_credentials` — performance/security tuning.

Why it matters: `sssd` is the glue between Linux NSS/PAM and AD. It answers user lookups (`getent`), and performs authentication through Kerberos/SSO.

Example:

```
[sssd]
services = nss, pam
domains = VPN.LOCAL

[domain/VPN.LOCAL]
id_provider = ad
auth_provider = krb5
ad_domain = vpn.local
krb5_realm = VPN.LOCAL
fallback_homedir = /home/%u
use_fully_qualified_names = False
```

`/etc/nsswitch.conf` — Name Service Switch

What it is: Informs the system where to look for user and group information.
Typical lines:

```
passwd:     compat sss
group:      compat sss
```

```
shadow:     compat sss
```

Why it matters: If `sss` is not present here, `getent` and system calls (e.g., `login`, `sshd`) won't query `sssd` for AD users. `sssd` must be listed for Linux to find AD users.

PAM configuration (`/etc/pam.d/*`) — Pluggable Authentication Modules

What it is: A stack of modules that handle authentication, account checking, session setup, and password changes.

Key pieces used:

- `pam_sss.so` — authenticates against sssd.
- `pam_mkhomedir.so` — creates home directories on first login.
- `common-auth`, `common-account`, `common-session` — Debian/Ubuntu include these in service-specific files (e.g., `/etc/pam.d/sshd`).

Why it matters: This is what SSH and `login` use to authenticate users and set up sessions. Correct PAM configuration is essential so AD users can login using `sssd`.

`/etc/wireguard/wg0.conf` — WireGuard interface config

What it is: The persistent configuration file for the WireGuard interface. Contains the server `[Interface]` block and one or more `[Peer]` blocks for clients.

Important responsibilities:

- `PrivateKey` — server private key (keep secret).
- `Address` — WireGuard IP for server (e.g., `10.10.0.1/24`).
- `PostUp` / `PostDown` — firewall NAT rules to allow clients to reach the Internet.
- Each `[Peer]` has `PublicKey` and `AllowedIPs` (the client's internal WireGuard IP).

Why it matters: WireGuard enforces access by public keys in `[Peer]` blocks. The approach we used enforces AD authentication at the time of key issuance, not at handshake time.

Example excerpt:

```
[Interface]
PrivateKey = <server-private-key>
Address = 10.10.0.1/24
ListenPort = 51820

[Peer]
# Peer for alice
PublicKey = AAAA...
AllowedIPs = 10.10.0.11/32
```

`/etc/ssh/sshd_config` — SSH Daemon config

What it is: Configures how `sshd` authenticates users (PAM usage, password auth, public key auth, root login).

Important flags for AD integration:

- `UsePAM yes` — allows PAM sss flow to be used for SSH auth.
- `PasswordAuthentication yes/no` — when `yes` AD users can auth with domain passwords over SSH; when `no` they can still use keys if present.
- `AuthorizedKeysCommand` — can be used to fetch public keys from AD if desired.

Why it matters: SSH is typically the administrative access method and often the user path to obtain WireGuard client configs (in our control-plane design).

`/etc/sudoers.d/wg-client-create` — sudoers entry used in the plan

What it is: A small sudoers fragment that permits members of the AD group `VPNUsers` to run the client-generation script as root without a password.

Important note: The group name format when domain-qualified may need escaping: `%VPN\\Users` or similar depending on `sssd` configuration.

Why it matters: This controls who may create WireGuard client configs (enforcing AD group membership for issuance).

---

4) How the system actually works — step-by-step runtime flow (sequence)

I'll show two flows: one for **authentication lookups** (login) and one for **WireGuard client issuance** (how we restrict VPN access to AD users).

A — AD user login / identity lookup (when `alice` logs in via SSH)

1. **SSH receives connection** and hands control to PAM (because `UsePAM yes`).
2. **PAM stack** calls `pam_sss.so` for auth (configured via `pam-auth-update` / `/etc/pam.d`).
3. `pam_sss` asks **SSSD** to authenticate the credentials.
4. **SSSD**:
   - If the client needs a Kerberos ticket, `sssd` will use the local keytab / Kerberos configuration to contact the **KDC** (Samba) and obtain/validate tickets.
   - For identity lookups, `sssd` queries AD LDAP (Samba's LDAP) to fetch user attributes (uidNumber, gidNumber, unixHomeDirectory, etc.) — or uses `ldap_id_mapping`.
5. `sssd` returns **success/failure** back to PAM. If success, PAM creates a session. `pam_mkhomedir` may create `/home/alice`.
6. The login is allowed and the user gets a shell. `id alice` and `getent passwd alice` now work because `nsswitch.conf` is configured to query `sss`.

Diagram (simplified):

```
ssh → PAM → pam_sss → sssd → (KDC ldap services provided by Samba)
              ^               ^
              |               |
```

```
/etc/krb5.conf          /etc/samba/smb.conf & /var/lib/samba
```

B — WireGuard client issuance (how we ensure only AD users get configs)

1. AD user `alice` SSHes to the server and runs:

   ```
   sudo /usr/local/bin/wg-create-client.sh alice
   ```

   (sudo is allowed because the `VPNUsers` AD group is in the sudoers fragment).

2. The sudoers rule ensures **only AD users in VPNUsers** can execute the script as root (no password).

3. The script:

   - Validates the invoker is in `VPNUsers` (double check).
   - Generates a WireGuard keypair.
   - Creates a client config file and places it in `/home/alice/...`.
   - Appends a `[Peer]` entry into `/etc/wireguard/wg0.conf` (persistent).
   - Applies the peer to the running `wg0` with `wg set ...`.

4. Now the client config contains the client private key and server endpoint. Importing it into a WireGuard client allows a successful handshake using WireGuard keypairs.

5. Because only AD users could run the script and receive the client private key, the VPN is effectively restricted to AD-authenticated users — even though the runtime handshake is still key-based.

Diagram (simplified):

```
alice@ssh → sudo (allowed via AD group) → wg-create-client.sh (runs as roo
t)
  → writes /home/alice/alice_wg.conf (private key) & updates /etc/wireguard/
```

```
wg0.conf (server side)
WireGuard handshake: client (private key) ↔ server (peer list contains client public key)
```

---

5) What `getent` is and why it's useful

- **Definition:** `getent` is a standard Unix command that queries the Name Service Switch (NSS) databases using the same system libraries as the OS — it asks for entries like `passwd`, `group`, `hosts`, `services`, etc.
- **Common usage:**
  - `getent passwd alice` — asks NSS for the `alice` account. If `sssd` (via `nsswitch.conf`) supplies it, `getent` will show it.
  - `getent group VPNUsers` — shows the AD group if `sss` or `winbind` is configured.
- **Why it matters:** It's the canonical way to verify the system sees the AD-provided users and groups via `sssd`/`winbind`. CLI tools and services use the same NSS mechanism; `getent` uses that same path so it's the best test.

Examples:

```
getent passwd alice
# prints the passwd line for alice if resolvable

getent group VPNUsers
# prints the group line if resolvable
```

If `getent` returns nothing for an AD user, that means NSS/SSSD lookup is not working — start debugging `sssd` and `nsswitch.conf`.

---

6) Key verification & troubleshooting checklist (short)

- **Kerberos time skew**: Kerberos requires clocks to be roughly in sync (≤5 min). Check `timedatectl`.
- **Keytab present**: Ensure `/etc/krb5.keytab` exists and contains host principals (`sudo klist -k`).
- **SSSD running**: `sudo systemctl status sssd` and check `/var/log/sssd/` logs.
- **NSS configured**: `/etc/nsswitch.conf` must contain `sss` for `passwd` / `group` / `shadow`.
- **PAM configured**: `pam_sss` present in `common-auth`; `pam_mkhomedir` in `common-session`.
- **DNS SRV records**: `host -t SRV _kerberos._tcp.vpn.local` should return the KDC record (Samba provides it).
- **Samba DC status**: `sudo samba-tool domain level show` and `sudo systemctl status samba-ad-dc`.
- **WireGuard peers**: `sudo wg show` — check peer public keys and last handshake.

---

7) Short glossary (one-liners)

- **AD (Active Directory):** Windows directory service storing users, groups, policies, and service discovery.
- **Samba (as AD DC):** Linux implementation of AD services (LDAP + KDC + DNS + RPC).
- **Kerberos:** Ticket-based authentication protocol (TGT/TGS).
- **Keytab:** File with service principal keys used by services to prove identity.
- **sssd:** System Security Services Daemon — Linux identity/auth gateway to remote providers (AD, LDAP).
- **PAM:** Authentication stack used by SSH, login, etc.
- **NSS:** System resolver for names (passwd/group), configured in `nsswitch.conf`.
- **getent:** Tool to query NSS databases (e.g., `passwd`, `group`).
- **WireGuard:** Modern VPN that authenticates with public/private keypairs; does not natively do username/password authentication.

---

8) Final short example: verifying an AD user can be seen and authenticate

Run these on the server to check the main pieces:

```
# Is sssd running?
sudo systemctl status sssd

# Can we look up the user?
getent passwd alice

# Does Kerberos issue a ticket for AD admin?
kinit administrator@VPN.LOCAL
klist

# Is Samba AD DC healthy?
sudo samba-tool domain level show

# Does wg show the interface and peers?
sudo wg show
```

If these commands show reasonable output (sssd active, `getent` prints user, `kinit` returns a ticket, samba-tool reports a functional level, and `wg show` lists the interface), your core stack is functioning.

AD is not natively built into

```
wg-easy
```

or most WireGuard UIs — they authenticate by local users or web passwords, not LDAP/AD. You can still integrate AD

indirectly

by putting the UI behind an AD-authenticated reverse proxy or by enforcing AD-auth when users request configs (via scripts or PAM).

Below I explain each concept (AD, NSS, PAM, LDAP, Samba AD/DC), how the authentication chain works, what happens when a user connects and gets a config, and what the iptables PostUp/PostDown rules do.

🧩 1. Active Directory (AD) integration with WireGuard UIs

✅ What's possible

- `wg-easy` (a simple Node.js WireGuard UI) **does not have native AD/LDAP integration**.
  - It stores credentials in its internal user database.
  - You can't directly make it ask AD for user passwords.
- **Workaround (practical):**
  1. Run an **NGINX reverse proxy** in front of `wg-easy`.
  2. Use **Kerberos (SPNEGO)** or **LDAP auth module** in NGINX.
  3. Users log in with AD credentials at the proxy layer.
  4. Only authenticated sessions can access the `wg-easy` UI.

  Example:

  ```
  user(browser) → nginx (AD auth) → wg-easy (localhost:51821)
  ```

- Alternatively, use more enterprise UIs like:
  - **NetBird** (WireGuard + SSO, supports AD/Azure AD/LDAP)

- **Headscale + Glauth** (for LDAP auth)
- **Tailscale Enterprise** (supports SSO via AzureAD, Google Workspace, Okta)

These are easier for centralized auth than hacking AD into `wg-easy`.

🧠 2. How AD concepts fit into your setup

Let's clarify the logic chain when AD is tied into the system for user control:

Step-by-step of what happens:

1. **User (AD account) connects via SSH** to the server.

   SSH uses **PAM**, which calls **sssd**, which talks to **AD (Samba)** to validate the username/password.

   → This confirms the user is a valid domain user.

2. **User runs the WireGuard provisioning script** (`wg-create-client.sh`).

   The `sudoers` rule allows *only* AD users in a specific AD group (e.g. `VPNUsers`) to execute the script.

3. **Script generates keys and config file.**

   It uses `wg genkey` / `wg set` commands to add a peer, and writes a config file for the user in `/home/<username>`.

4. **User downloads that config** (via SSH, SCP, or UI protected by AD login).

   Once imported into the WireGuard client, the tunnel works using public/private keys.

5. **When the client connects:**
   - WireGuard itself only validates the *public key* (not username/password).
   - So AD is used *beforehand* — to determine *who is allowed* to get a key/config, not at handshake time.

In short:

> AD is your gatekeeper for who gets a WireGuard profile, not part of the WireGuard handshake itself.

🧩 3. Key Linux identity/auth systems: NSS, PAM, and LDAP

Component	Purpose	Example of Role
NSS (Name Service Switch)	Lets Linux resolve *who* a user/group is and their UID/GID/home shell, regardless of where they come from (local `/etc/passwd`, LDAP, AD). Controlled by `/etc/nsswitch.conf`.	When you run `id alice`, the system asks NSS → SSSD → AD.
PAM (Pluggable Authentication Modules)	Handles *how users authenticate* — login, SSH, sudo. Configured in `/etc/pam.d/`.	SSHD calls PAM → `pam_sss.so` → AD via SSSD.
LDAP (Lightweight Directory Access Protocol)	A network protocol for querying directory info (users, groups, etc.). AD exposes LDAP.	SSSD uses LDAP to fetch user/group data from Samba AD.

Relationship:

```
Application (SSH, sudo, login)
  ↳ PAM (auth)
    ↳ SSSD (delegates to AD for password/Kerberos)
  ↳ NSS (lookup)
    ↳ SSSD (fetches user/group info from AD LDAP)
```

🏛️ 4. Samba AD vs Samba AD DC vs AD DC

✅ Samba AD

General term meaning Samba is providing Active Directory services (Kerberos, LDAP, DNS).

✅ Samba AD DC

"Samba Active Directory Domain Controller" — a specific *server role* in Samba.

It's the Linux equivalent of a Windows Domain Controller.

✅ AD DC (Active Directory Domain Controller)

The **core server** in any Active Directory environment:

- Hosts **LDAP directory** (user/group database)
- Hosts **Kerberos KDC** (issues tickets)
- Hosts **DNS** for domain lookups
- Issues **Group Policy** and manages replication

In Samba, `server role = active directory domain controller` makes it a full AD DC, combining:

- LDAP server
- Kerberos server
- DNS
- AD replication interfaces (if multiple DCs)

Summary table:

Term	Full Name	Role
AD	Active Directory	Directory service concept
AD DC	Active Directory Domain Controller	The actual server providing AD
Samba AD DC	Samba's implementation of an AD DC	Linux version of a Windows DC

⚙️ 5. What the WireGuard script does

Example: `/usr/local/bin/wg-create-client.sh`

It automates five steps:

1. **Generate key pair**

```
wg genkey | tee /etc/wireguard/clients/alice.key | wg pubkey > /etc/wireg
uard/clients/alice.pub
```

→ Creates private/public keys for this user.

2. **Assign IP**

```
CLIENT_IP="10.10.0.11"
```

→ Each user gets a unique tunnel IP.

3. **Add peer to server config**

```
echo -e "\n[Peer]\nPublicKey = $(cat /etc/wireguard/clients/alice.pub)\nAl
lowedIPs = ${CLIENT_IP}/32" >> /etc/wireguard/wg0.conf
```

→ Tells the server “accept traffic from this public key”.

4. **Apply change live**

```
wg set wg0 peer $(cat /etc/wireguard/clients/alice.pub) allowed-ips ${CLI
ENT_IP}/32
```

→ Updates the running interface instantly.

5. **Generate client config file**

```
cat > /home/alice/alice_wg.conf <<EOF
[Interface]
PrivateKey = $(cat /etc/wireguard/clients/alice.key)
Address = ${CLIENT_IP}/32
DNS = 1.1.1.1
```

```
[Peer]
PublicKey = $(cat /etc/wireguard/server.pub)
Endpoint = your.server.ip:51820
AllowedIPs = 0.0.0.0/0
EOF
```

→ This is what the user imports into WireGuard client.

6. **Set permissions and show message**

```
chown alice:alice /home/alice/alice_wg.conf
echo "Config ready for alice"
```

So, it gives AD-approved users a ready `.conf` file and updates the server — no manual editing.

🔥 6. What the PostUp/PostDown rules do

These lines in `wg0.conf` are **iptables firewall rules**.

They're executed *after* (`PostUp`) or *before down* (`PostDown`) the interface comes up or goes down.

The exact lines:

```
PostUp = /usr/sbin/iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
PostUp = /usr/sbin/iptables -A FORWARD -i %i -j ACCEPT
PostUp = /usr/sbin/iptables -A FORWARD -o %i -j ACCEPT

PostDown = /usr/sbin/iptables -t nat -D POSTROUTING -o eth0 -j MASQUERA
DE
PostDown = /usr/sbin/iptables -D FORWARD -i %i -j ACCEPT
PostDown = /usr/sbin/iptables -D FORWARD -o %i -j ACCEPT
```

What they do:

Command	Explanation
`iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE`	Enables NAT (Network Address Translation) so VPN clients' traffic going out via `eth0` (Internet) appears as coming from the server's IP. Without this, clients wouldn't reach the Internet.
`iptables -A FORWARD -i %i -j ACCEPT`	Allows forwarding of packets coming **from** the WireGuard interface (e.g. `wg0`) to other interfaces.
`iptables -A FORWARD -o %i -j ACCEPT`	Allows forwarding of packets **to** the WireGuard interface.
`PostDown` lines	Remove these rules when the interface is brought down.

So, these rules **enable routing + NAT**, making the VPN a full tunnel.

Diagram:

```
[Client 10.10.0.x] → wg0 (VPN) → iptables NAT → eth0 → Internet
```

🧭 7. Summary chain (how everything connects)

```
AD User (domain account)
  |
  |--- SSH login ——→ PAM → SSSD → Samba AD (Kerberos+LDAP)
  |
  |--- Runs wg-create-client.sh (sudo allowed by AD group)
       |
       |--- Generates keys + WireGuard config
       |--- Adds peer to /etc/wireguard/wg0.conf
       |--- Gives user a .conf file
       |
  |--- Imports config into WireGuard client
       |
       |--- WireGuard handshake (key-based)
            |
            |--- NAT (iptables PostUp)
```

```
        |
        +—> Internet or private network
```

🧩 Samba Domain Setup

- `sudo rm -f /etc/samba/smb.conf`

  Removes the default Samba configuration file to ensure the system starts clean before provisioning the AD Domain Controller. This avoids conflicts between standalone server settings and domain controller configuration.

- `sudo samba-tool domain provision --realm=VPN.LOCAL --domain=VPN --adminpass='Str0ngAdminPass!' --server-role=dc --use-rfc2307 --dns-backend=SAMBA_INTERNAL --option="dns forwarder = 8.8.8.8"`

  Provisions a new Samba Active Directory Domain Controller with Kerberos, LDAP, and internal DNS. It defines the realm (VPN.LOCAL), creates initial databases, configures DNS forwarding, and sets the admin password.

- `sudo apt install -y samba-ad-dc samba-dsdb-modules samba-vfs-modules python3-samba samba-common-bin smbclient winbind libnss-winbind libpam-winbind krb5-user`

  Installs all required Samba DC, authentication, and Kerberos modules. This ensures AD DC components, PAM/NSS integration, and Kerberos libraries are all present for domain functionality.

⚙️ Kerberos & Host Configuration

- Editing `/etc/krb5.conf`

  Defines the Kerberos realm (VPN.LOCAL), KDC, and admin server. It ensures the system knows where to contact Kerberos for authentication requests.

- Editing `/etc/hosts`

  Maps the server's hostname and FQDN (like `dc1.vpn.local`) to its local IP address. This guarantees Kerberos and Samba resolve the DC hostname correctly.

🧠 Samba DC Service & Verification

- `sudo systemctl enable samba-ad-dc --now`

  Enables and starts the Samba AD DC service immediately, ensuring it autostarts at boot. This runs LDAP, Kerberos, and DNS as an integrated directory service.

- `sudo samba-tool domain level show`

  Displays the current domain and forest functional levels to verify provisioning succeeded.

- `sudo host -t SRV _kerberos._tcp.vpn.local`

  Queries DNS for Kerberos service SRV records, confirming that the DC publishes the Kerberos endpoint.

- `sudo host -t SRV _ldap._tcp.vpn.local`

  Checks DNS for LDAP SRV records, validating proper LDAP service advertisement for AD operations.

- `sudo kinit administrator@VPN.LOCAL <<< 'Str0ngAdminPass!' && klist`

  Authenticates as the AD administrator via Kerberos and lists the obtained ticket. Verifies that Kerberos trust and time sync are functioning.

👥 AD User and Group Management

- `sudo samba-tool group add VPNUsers`

  Creates a new AD group named `VPNUsers` within the directory. This group will control which users have WireGuard access privileges.

- `sudo samba-tool user add alice MySecurePassword123 --given-name="alice" --description="VPN user"`

  Adds a new AD user account called `alice` with the given password and metadata. Automatically assigns her to the AD database and sets required attributes.

🔐 SSSD & PAM Integration

- `sudo samba-tool domain exportkeytab /etc/krb5.keytab -U administrator`

Exports the Samba domain keytab to `/etc/krb5.keytab`. This allows system daemons like SSSD to authenticate against AD using stored Kerberos keys.

- `sudo chown root:root /etc/krb5.keytab`

  Secures ownership of the keytab file so only the root user can read it, preventing credential exposure.

- `sudo chmod 0600 /etc/krb5.keytab`

  Restricts permissions on the keytab to owner-only access, ensuring confidentiality of Kerberos service keys.

- Editing `/etc/sssd/sssd.conf`

  Configures the SSSD service to use the `ad` provider for VPN.LOCAL. Enables Kerberos-based authentication and automatic UID/GID mapping.

- `sudo chmod 600 /etc/sssd/sssd.conf`

  Limits access to the SSSD configuration to root only, protecting stored secrets and keys.

- `sudo systemctl enable sssd --now`

  Starts and enables SSSD on boot. This activates AD user lookup and Kerberos authentication for PAM/NSS.

- Editing `/etc/nsswitch.conf`

  Adds `sss` to passwd, group, and shadow entries, telling the system to resolve user info from AD through SSSD.

- `sudo pam-auth-update --enable mkhomedir`

  Enables automatic home directory creation for AD users upon first login. It modifies PAM to use the `pam_mkhomedir` module.

Testing AD Integration

- `getent passwd alice`

  Queries NSS for the `alice` account, proving that AD user resolution via SSSD works.

- `id alice`

Displays the user's UID, GID, and group memberships to verify SSSD mappings.

- `sudo su -s /bin/bash alice -c "whoami"`

  Tests PAM/Kerberos authentication by switching to the AD user context and running a shell command.

🔐 SSH Configuration for AD Users

- `sudo cp /etc/ssh/sshd_config /etc/ssh/sshd_config.bak`

  Creates a backup of the SSH configuration file before modifications, allowing rollback if needed.

- Editing `/etc/ssh/sshd_config`

  Enables password authentication, disables key-based logins if desired, and allows AD users to authenticate via PAM.

- `sudo systemctl restart sshd`

  Restarts the SSH daemon to apply new authentication rules.

⚡ Sudo Configuration for AD Group

- `sudo visudo`

  Opens the sudoers file safely for editing.

  Adding `%VPNUsers ALL=(ALL:ALL) NOPASSWD: /usr/local/bin/wg-create-client.sh` gives AD users in `VPNUsers` passwordless permission to run the script as root.

🧰 WireGuard Client Creation Script

- `sudo chmod 700 /usr/local/bin/wg-create-client.sh`

  Restricts execution of the WireGuard client creation script to root only. Prevents non-privileged access to sensitive operations.

- `sudo chown root:root /usr/local/bin/wg-create-client.sh`

  Assigns ownership of the script to root for security and integrity control.

🧪 Full System Test

- `ssh podarochek@209.38.40.145 sudo /usr/local/bin/wg-create-client.sh podarochek`

  Logs in as an AD user and executes the client configuration script through sudo. Tests group permissions, authentication, and config generation.

- `scp podarochek@209.38.40.145:/home/podarochek/podarochek_wg.conf /`

  Copies the generated WireGuard configuration file from the server to the local system. Confirms end-to-end AD-authenticated VPN setup.