Duo MFA Integration

Scope

Duo MFA is enforced in two places:

1. **SSH login** (v2.3+) — via PAM keyboard-interactive after publickey auth
2. **sudo** (v2.4+) — via PAM at the auth phase, with AD password as the post-Duo factor

The role uses Duo Security's official `duo-unix` package, not Ubuntu universe's `libpam-duo` (which is outdated and has Duo API incompatibilities).

Package Installation

Source: `roles/ssh-baseline/tasks/duo.yml`. The flow:

1. Download Duo's GPG signing key from `https://duo.com/DUO-GPG-PUBLIC-KEY.asc`
2. Convert to a dearmored keyring at `/etc/apt/trusted.gpg.d/duo.gpg`
3. Add APT repository: `deb [arch=amd64] https://pkg.duosecurity.com/Ubuntu {{ ansible_distribution_release }} main`
4. Purge any legacy `libpam-duo` / `libduo3` from Ubuntu universe
5. Install `duo-unix` package

Inline comment from the role explaining why we don't use Ubuntu universe:

> 1. Ubuntu universe ships 1.11.3 (2022) which has incompatibilities with current Duo Auth API and returns HTTP 403 in some scenarios.
> 2. Duo's 2.1.0+ is required for the April 2026 CA bundle rotation.
> 3. Duo's docs explicitly target the duo-unix package on Ubuntu 22.04.

The package installs `pam_duo.so` at `/usr/lib64/security/` — not in Ubuntu's default PAM module search path. Both PAM stack templates reference the module by absolute path for this reason.

Duo PAM Configuration File

Template: `roles/ssh-baseline/templates/pam_duo.conf.j2`. Deployed to `/etc/duo/pam_duo.conf` with mode 0600 (contains `skey`). The task that writes it has `no_log: true`.

```
# Managed by Ansible - PBR ssh-baseline role
# Source: roles/ssh-baseline/templates/pam_duo.conf.j2
#
# pam_duo.conf - configuration for Duo Security PAM module
# Permissions MUST be 0600 owned by root (contains skey).

[duo]
ikey = {{ duo_ikey }}
skey = {{ duo_skey }}
host = {{ duo_api_host }}

# failmode controls behaviour when Duo cloud is unreachable:
#   safe   = allow login (single-factor publickey fallback)
#   secure = deny login (locks out during Duo outage)
failmode = {{ duo_failmode }}

# Include hostname + command in push notification
pushinfo = {{ duo_pushinfo }}

# Max retries at the Duo prompt
prompts = {{ duo_prompts }}

# Auto-push to user's primary device (true) vs prompt for factor (false)
autopush = {{ duo_autopush }}

# Restrict Duo to AD server-access group members.
# Users not in this group (e.g. {{ break_glass_user }} break-glass) bypass Duo automatically.
groups = {{ ad_server_access_group | lower }},{{ ad_sudo_group | lower }}
```

The `groups` directive is the key Duo-level filter: `pam_duo.so` only challenges users in the listed groups. Local accounts (`pbr_admin`, `ansible`) are not in those groups, so they bypass Duo entirely — even before our pam_succeed_if carve-outs fire.

Group names are lowercased because SSSD normalises AD group names to lowercase when surfacing them via NSS.

SSH PAM Stack (pam_sshd.j2)

Deployed to `/etc/pam.d/sshd`. This is a custom file (not `@include common-auth` at the top) so we can control the order of Duo vs. password validation precisely.

```
# Managed by Ansible - PBR ssh-baseline role
# === Auth section ===
auth    [success=2 default=ignore]  pam_succeed_if.so user = pbr_admin quiet

# AD users: Duo MFA is required, failure terminates the stack
auth    requisite                   /usr/lib64/security/pam_duo.so

# Duo succeeded → exit stack with success (do not fall through to pam_unix)
auth    [success=done default=die]  pam_permit.so

# pbr_admin lands here (jumped past pam_duo + pam_permit)
auth    required                    pam_unix.so try_first_pass nullok_secure

# === Account section ===
account    required     pam_nologin.so
@include common-account

# === Session section ===
session    [success=ok ignore=ignore module_unknown=ignore default=bad]   pam_selinux.so close
session    required     pam_loginuid.so
session    optional     pam_keyinit.so force revoke
@include common-session
session    optional     pam_motd.so motd=/run/motd.dynamic
session    optional     pam_motd.so noupdate
session    optional     pam_mail.so standard noenv
session    required     pam_limits.so
session    required     pam_env.so
session    required     pam_env.so user_readenv=1 envfile=/etc/default/locale
session    [success=ok ignore=ignore module_unknown=ignore default=bad]   pam_selinux.so open

# === Password section ===
@include common-password
```

Auth section dissection

Four lines of auth, each with deliberate control flow. Reading from the top:

Line 1: pbr_admin detection & branching

```
auth    [success=2 default=ignore]  pam_succeed_if.so user = pbr_admin quiet
```

- `pam_succeed_if.so user = pbr_admin` returns success if the authenticating user is `pbr_admin` .
- `success=2` means: on success, skip the next 2 modules (pam_duo and pam_permit).
- `default=ignore` means: for any other return value (the user is NOT pbr_admin), continue to the next module.

Effect: If you're `pbr_admin`, jump straight to the `pam_unix.so` line. If you're not, continue to pam_duo.

Line 2: Duo MFA

```
auth    requisite                   /usr/lib64/security/pam_duo.so
```

- `requisite` means: if this module fails, terminate the auth stack immediately with that failure code. Do not try further modules.
- This is for AD users (who reach this line because Line 1's pam_succeed_if didn't match).
- Inside `pam_duo.so`, the `groups` filter in `pam_duo.conf` applies — if the user is not in `sg_serveraccess` or `sg_sudo`, Duo skips them and returns success without prompting. (In practice, sshd's `AllowGroups` would have rejected them earlier, so this is defence-in-depth.)

Line 3: success exits the stack

```
auth    [success=done default=die]  pam_permit.so
```

- `pam_permit.so` always returns success.
- `success=done` means: terminate the auth stack with overall success. Do not run later auth modules.
- Reached only after pam_duo passes. AD users land here on success and exit the stack cleanly.

Line 4: pbr_admin's destination

```
auth    required                    pam_unix.so try_first_pass nullok_secure
```

- Reached only by `pbr_admin` (who jumped here via Line 1's `success=2`).
- `pam_unix.so` validates the local password against `/etc/shadow`.
- `try_first_pass` uses the password already supplied (sshd passes it via the keyboard-interactive PAM conversation).
- `required` means: failure makes the stack fail, but later modules still run (none in this stack).

The full sshd authentication picture

Putting sshd's `AuthenticationMethods publickey,keyboard-interactive` together with the PAM stack:

User	sshd Step 1: publickey	sshd Step 2: keyboard-interactive (PAM)
AD user (e.g. a.mfraser)	Validates against AD-stored `sshPublicKey` via SSSD	pam_succeed_if doesn't match → pam_duo prompts → success exits stack
pbr_admin	(see below)	pam_succeed_if matches → jump to pam_unix → validates local password
ansible	Local `~/.ssh/authorized_keys`; `AuthenticationMethods publickey` in Match block bypasses keyboard-interactive	Never enters PAM auth

Wait: how does pbr_admin authenticate at all if sshd requires publickey first?

The `Match User pbr_admin Address ...` block in `sshd_hardening.conf.j2` overrides `AuthenticationMethods` for that user to `password` only:

```
Match User pbr_admin Address {{ pbr_admin_allowed_sources }}
    PasswordAuthentication yes
    AuthenticationMethods password
```

So `pbr_admin` enters PAM via password auth (not keyboard-interactive), but the PAM stack handles both flows the same way — `pam_succeed_if` matches and jumps to `pam_unix` which validates the password.

---

sudo PAM Stack (pam_sudo.j2)

Deployed to `/etc/pam.d/sudo`. Replaces the Ubuntu-default file.

```
#%PAM-1.0
# Managed by Ansible - PBR ssh-baseline role v2.4

# Standard Ubuntu sudo session environment setup
session    required   pam_env.so readenv=1 user_readenv=0
session    required   pam_env.so readenv=1 envfile=/etc/default/locale user_readenv=0

# Skip Duo for users not in the AD sudo group (covers ansible, pbr_admin,
# and any local user with sudo rights).
auth       [success=1 default=ignore]  pam_succeed_if.so quiet user notingroup sg_sudo

# Require Duo MFA for AD users in the sudo group.
auth       requisite                   /usr/lib64/security/pam_duo.so

# Validate the user's password (AD via pam_sss for AD users, local via
# pam_unix for break-glass account). NOPASSWD entries in sudoers bypass
# this entire auth phase regardless.
@include common-auth
@include common-account
@include common-session-noninteractive
```

Auth section dissection

Line 1: AD sudo group check

```
auth       [success=1 default=ignore]  pam_succeed_if.so quiet user notingroup sg_sudo
```

- `pam_succeed_if user notingroup sg_sudo` returns success if the user is **not** in `sg_sudo`.
- `success=1` jumps over the next module (pam_duo).
- `default=ignore` continues to pam_duo for users IN `sg_sudo`.

Group name is lowercase because SSSD normalises AD group names. The template uses `{{ ad_sudo_group | lower }}` for safety.

Line 2: Duo for AD sudo users

```
auth       requisite                   /usr/lib64/security/pam_duo.so
```

- Reached only by users in `sg_sudo`.
- `requisite` aborts the stack on Duo failure (denied push, timeout, etc.).

- On success, falls through to common-auth.

Line 3: Password validation

```
@include common-auth
```

- `common-auth` runs `pam_sss.so` for AD users (validates AD password) or `pam_unix.so` for local users.
- NOPASSWD entries in sudoers bypass this entire auth phase — `ansible` sudo never reaches PAM auth at all.

The full sudo authentication picture

User	PAM flow	Effective auth
AD user in sg_sudo	pam_succeed_if doesn't match → pam_duo prompts → common-auth → pam_sss	Duo push + AD password
pbr_admin (NOT in sg_sudo)	pam_succeed_if matches → jump past pam_duo → common-auth → pam_unix	Local password
ansible (NOPASSWD sudoers)	sudoers NOPASSWD bypasses PAM auth entirely	None

---

sudo Credential Cache Extension

The role drops `/etc/sudoers.d/sudo_timestamp_timeout`:

```
# Managed by Ansible - PBR ssh-baseline role v2.4
# Extends sudo credential cache from default 15min to {{ sudo_timestamp_timeout }}min
# to reduce Duo MFA push frequency for AD sudo users without significantly
# weakening the control (session hijack window unchanged).
Defaults timestamp_timeout={{ sudo_timestamp_timeout }}
```

Default value: `sudo_timestamp_timeout: 30` (minutes). Ubuntu's default is 15.

The drop-in is validated with `visudo -cf` before being written. The file is mode 0440 (per sudoers convention).

Why extend: A typical maintenance session involves many sudo invocations. With the default 15-minute cache, an AD user gets repeated Duo pushes. Extending to 30 minutes reduces noise

without meaningfully changing the security envelope — the session-hijack window is per-tty and the underlying authentication is unchanged.

Failure Mode (failmode = safe)

If Duo's cloud is unreachable (DNS broken, Duo outage, firewall change), pam_duo returns success and the stack proceeds. For SSH this means single-factor publickey is sufficient; for sudo, common-auth still requires a password.

The trade-off:

- **With failmode = safe (chosen):** Duo outages don't lock administrators out. Single-factor publickey is still strong — AD-managed keys with revocation in effect.
- **With failmode = secure:** Stronger MFA guarantee but Duo outages cause fleet-wide lockout. `pbr_admin` break-glass would be the only path in.

Chosen: `safe`. PBR has acceptable compensating controls (key-based auth, AD password for sudo, source-IP-restricted break-glass) such that single-factor degradation during a Duo outage is acceptable.

Validation Tasks in the Role

After deploying both PAM stacks and pam_duo.conf, the role runs validation checks to fail fast if something is wrong:

```
- name: Validate Duo module is referenced in sudo PAM stack
  ansible.builtin.command: grep -c "pam_duo.so" /etc/pam.d/sudo
  failed_when: sudo_pam_duo_check.stdout | int &lt; 1

- name: Sanity check - sudo still works for non-Duo automation accounts
  ansible.builtin.command: sudo -n true
  become: false
  # Runs as the ansible_user (ansible). ansible has NOPASSWD in sudoers
  # and is not in sg_sudo, so it should bypass Duo entirely. If this fails,
  # the new PAM stack has broken local sudo - red flag, terminate deploy.

- name: Validate Duo module is referenced in sshd PAM stack
  ansible.builtin.command: grep -E "pam_duo\.so" /etc/pam.d/sshd
```

```
- name: Validate pam_duo.so exists at the absolute path used by PAM stack
  ansible.builtin.stat: path: /usr/lib64/security/pam_duo.so
  failed_when: not pam_duo_stat.stat.exists
```

The sanity sudo check is particularly important: it runs as the `ansible` user (non-Duo automation) and verifies that sudo still works. If the new PAM stack broke local sudo, the deploy halts immediately rather than continuing through subsequent tasks that depend on sudo working.

Compliance Note

From the inline comment in `defaults/main.yml`:

> Duo MFA on sudo (v2.4)
> Essential Eight ML2: MFA for privileged users performing privileged actions.

This is the only Essential Eight reference in the role's source. Broader compliance mappings (VPDSS, VG-CISO) are out of scope for this documentation — refer to PBR's separate compliance documentation if needed.

Troubleshooting Duo

"Permission denied" without a Duo prompt

Most likely the user is not in `SG_ServerAccess` or `SG_Sudo` — sshd's `AllowGroups` rejected them before PAM ran. Verify:

```
ssh -vvv a.mfraser@host.pbr.org.au 2&gt;&amp;1 | grep -i 'permission denied\|allowgroups'
```

Duo prompt arrives but auth fails

Check the host's Duo PAM logs:

```
sudo journalctl -u sshd --since "5 minutes ago" | grep -i duo
```

Common causes: Duo Auth API `ikey`/`skey`/`host` wrong in `/etc/duo/pam_duo.conf` (vault credentials mismatch), system clock drift (Duo requires NTP), user disabled in Duo admin console.

sudo asks for password but never prompts for Duo

Indicates the user is not in `sg_sudo`, so the pam_succeed_if branch skipped pam_duo. Verify:

```
id a.mfraser | tr ',' '\n' | grep -i sg_sudo
```

If empty, either the user isn't in the AD group (intended) or SSSD cache is stale (`sudo sss_cache -E`).

Where to Read Next

- **SSH Hardening Reference** — how sshd's Match blocks interact with the PAM stack
- **AD Integration & SSSD** — how pam_sss validates AD passwords post-Duo
- **Known Limitations, Troubleshooting & Version History** — Royal TS Rebex authentication caveats

---

Revision #1
Created 2026-05-13 05:28:54 UTC by PBR_Documentation
Updated 2026-05-13 05:28:54 UTC by PBR_Documentation