SIS v1.0 - Engineer Preflight Script

Skyes Over London LC | Effective 2026-02-27

This internal test script is the SKYE identity gate for production. Automated checks must pass, then complete the manual SSO verification steps.

Inputs required: APP_BASE, TENANT, SKYE_ADMIN_TOKEN. If SCIM is enabled, SCIM_TOKEN is required. If OIDC is used, OIDC_ISSUER is required.

sis_preflight.sh (part 1 of 6)

```
#!/usr/bin/env bash
# SKYE Identity Standard (SIS) — Engineer Preflight
# Version: SIS-1.0
#
# What it does:
# - Verifies SKYE identity discovery endpoint
# - Validates tenant identity status (SSO/SCIM enforcement flags)
# - Validates OIDC discovery + JWKS reachability (if configured)
# - Validates SAML cert expiry (if provided)
# - Runs SCIM CRUD smoke tests (Users + optional Groups)
# - Checks audit logging endpoints (basic reachability + last event presence)
# - Emits required manual SSO steps with pass/fail prompts
#
# Requirements:
# - bash, curl
# - jq optional (falls back to python for JSON parsing)
#
# Usage:
#   chmod +x sis_preflight.sh
#   APP_BASE="https://yourapp.example.com" \
#   TENANT="acme" \
#   SKYE_ADMIN_TOKEN="YOUR_ADMIN_BEARER_TOKEN" \
#   SCIM_TOKEN="YOUR_SCIM_BEARER_TOKEN" \
#   OIDC_ISSUER="https://issuer.example.com" \
#   ./sis_preflight.sh
#
# Optional:
#   SAML_CERT_PEM="/path/to/saml_cert.pem" ./sis_preflight.sh

set -euo pipefail

# ---------- Helpers ----------
red()   { printf "\033[31m%s\033[0m\n" "$*"; }
green() { printf "\033[32m%s\033[0m\n" "$*"; }
yellow(){ printf "\033[33m%s\033[0m\n" "$*"; }
blue()  { printf "\033[34m%s\033[0m\n" "$*"; }

need() {
  local name="$1"
  if [[ -z "${!name:-}" ]]; then
    red "MISSING env var: $name"
    exit 2
  fi
```

```
}

have_cmd() { command -v "$1" >/dev/null 2>&1; }

json_get() {
  # json_get <json> <jq_filter>
  local json="$1"
  local filter="$2"
  if have_cmd jq; then
    echo "$json" | jq -r "$filter"
  else
    python3 - <<PY
import json,sys
data=json.loads(sys.stdin.read())
# Minimal filter support for common paths (e.g., .field, .a.b)
path="${filter}".strip()
if not path.startswith("."):
```

sis_preflight.sh (part 2 of 6)

```
  print("")
  sys.exit(0)
keys=[k for k in path[1:].split(".") if k]
cur=data
for k in keys:
  if isinstance(cur, dict) and k in cur:
    cur=cur[k]
  else:
    print("")
    sys.exit(0)
print(cur if not isinstance(cur,(dict,list)) else json.dumps(cur))
PY
  fi
}

http() {
  # http <method> <url> <auth_header_optional> <data_optional>
  local method="$1"
  local url="$2"
  local auth="${3:-}"
  local data="${4:-}"
  local out
  if [[ -n "$data" ]]; then
    out=$(curl -sS -X "$method" "$url" \
      -H "Accept: application/json" \
      -H "Content-Type: application/json" \
      ${auth:+-H "$auth"} \
      --data "$data" \
      -w "\nHTTP_STATUS:%{http_code}\n")
  else
    out=$(curl -sS -X "$method" "$url" \
      -H "Accept: application/json" \
      ${auth:+-H "$auth"} \
      -w "\nHTTP_STATUS:%{http_code}\n")
  fi
  echo "$out"
}

status_code() {
  # Extract HTTP status appended by http()
  echo "$1" | sed -n 's/.*HTTP_STATUS:\([0-9][0-9][0-9]\).*/\1/p' | tail -n 1
}

body_only() {
  # Remove appended status line
  echo "$1" | sed '/^HTTP_STATUS:/d'
}

fail() { red "FAIL: $*"; exit 1; }
pass() { green "PASS: $*"; }

# ---------- Inputs ----------
need APP_BASE
need TENANT
need SKYE_ADMIN_TOKEN

APP_BASE="${APP_BASE%/}"

ADMIN_AUTH="Authorization: Bearer ${SKYE_ADMIN_TOKEN}"
```

sis_preflight.sh (part 3 of 6)

```
SCIM_TOKEN="${SCIM_TOKEN:-}"
OIDC_ISSUER="${OIDC_ISSUER:-}"
SAML_CERT_PEM="${SAML_CERT_PEM:-}"

blue "SIS Preflight starting..."
echo "APP_BASE=$APP_BASE"
echo "TENANT=$TENANT"
echo

# ---------- 1) SKYE Identity Discovery ----------
blue "[1/7] Identity discovery: $APP_BASE/.well-known/skye-identity"
resp=$(http GET "$APP_BASE/.well-known/skye-identity")
code=$(status_code "$resp")
body=$(body_only "$resp")

[[ "$code" == "200" ]] || fail "Discovery endpoint missing or not 200 (got $code). Required by SKYE standard."
version=$(json_get "$body" ".version")
scim_base=$(json_get "$body" ".scim_base")
audit_base=$(json_get "$body" ".audit_base")
tenant_status_path=$(json_get "$body" ".tenant_status_path")

[[ -n "$version" ]] || fail "Discovery missing .version"
[[ -n "$tenant_status_path" ]] || fail "Discovery missing .tenant_status_path"
pass "Discovery OK (version=$version)"
echo "scim_base=$scim_base"
echo "audit_base=$audit_base"
echo "tenant_status_path=$tenant_status_path"
echo

# ---------- 2) Tenant Identity Status ----------
blue "[2/7] Tenant identity status"
status_url="$APP_BASE${tenant_status_path//\{tenant\}/$TENANT}"
resp=$(http GET "$status_url" "$ADMIN_AUTH")
code=$(status_code "$resp")
body=$(body_only "$resp")
[[ "$code" == "200" ]] || fail "Tenant status not reachable (got $code) at $status_url"

sso_enabled=$(json_get "$body" ".sso.enabled")
sso_required=$(json_get "$body" ".sso.required")
sso_protocol=$(json_get "$body" ".sso.protocol")
scim_enabled=$(json_get "$body" ".scim.enabled")
mfa_required=$(json_get "$body" ".policy.mfa_enforced")
pwd_disabled=$(json_get "$body" ".policy.password_login_disabled")
profile=$(json_get "$body" ".profile") # SIS-Core or SIS-Enterprise

echo "profile=$profile"
echo "sso.enabled=$sso_enabled  sso.required=$sso_required  sso.protocol=$sso_protocol"
echo "scim.enabled=$scim_enabled"
echo "policy.mfa_enforced=$mfa_required  policy.password_login_disabled=$pwd_disabled"

[[ "$sso_enabled" == "true" ]] || fail "SSO not enabled for tenant."
[[ "$sso_protocol" == "oidc" || "$sso_protocol" == "saml" || "$sso_protocol" == "both" ]] || fail "Invalid sso
if [[ "$profile" == "SIS-Enterprise" ]]; then
  [[ "$sso_required" == "true" ]] || fail "Enterprise requires SSO required=true"
  [[ "$pwd_disabled" == "true" ]] || fail "Enterprise requires password_login_disabled=true"
  [[ "$mfa_required" == "true" ]] || fail "Enterprise requires mfa_enforced=true"
  [[ "$scim_enabled" == "true" ]] || fail "Enterprise requires SCIM enabled=true"
fi
pass "Tenant status checks OK"
echo
```

sis_preflight.sh (part 4 of 6)

```
# ---------- 3) OIDC Discovery + JWKS (if OIDC configured) ----------
blue "[3/7] OIDC checks (if applicable)"
if [[ "$sso_protocol" == "oidc" || "$sso_protocol" == "both" ]]; then
  [[ -n "$OIDC_ISSUER" ]] || fail "OIDC issuer required (set OIDC_ISSUER)."
  issuer="${OIDC_ISSUER%/}"
  disc_url="$issuer/.well-known/openid-configuration"
  blue "Fetching OIDC discovery: $disc_url"
  resp=$(http GET "$disc_url")
  code=$(status_code "$resp")
  body=$(body_only "$resp")
  [[ "$code" == "200" ]] || fail "OIDC discovery not reachable (got $code)."

  jwks_uri=$(json_get "$body" ".jwks_uri")
  authz_ep=$(json_get "$body" ".authorization_endpoint")
  token_ep=$(json_get "$body" ".token_endpoint")
  [[ -n "$jwks_uri" && -n "$authz_ep" && -n "$token_ep" ]] || fail "OIDC discovery missing required fields."

  blue "Fetching JWKS: $jwks_uri"
  resp=$(http GET "$jwks_uri")
  code=$(status_code "$resp")
  [[ "$code" == "200" ]] || fail "JWKS not reachable (got $code)."
  pass "OIDC discovery + JWKS OK"
else
  yellow "Skipping OIDC checks (tenant not using OIDC)."
fi
echo

# ---------- 4) SAML Cert Expiry (if SAML configured and cert provided) ----------
blue "[4/7] SAML cert checks (if applicable)"
if [[ "$sso_protocol" == "saml" || "$sso_protocol" == "both" ]]; then
  if [[ -n "$SAML_CERT_PEM" ]]; then
    [[ -f "$SAML_CERT_PEM" ]] || fail "SAML_CERT_PEM file not found: $SAML_CERT_PEM"
    if have_cmd openssl; then
      exp=$(openssl x509 -enddate -noout -in "$SAML_CERT_PEM" | sed 's/notAfter=//')
      echo "SAML cert expires: $exp"
      pass "SAML cert parsed OK (ensure >60 days remaining manually or via policy)"
    else
      yellow "openssl not found; cannot parse SAML cert expiry automatically."
    fi
  else
    yellow "SAML in use but no SAML_CERT_PEM provided. Provide cert to validate expiry."
  fi
else
  yellow "Skipping SAML checks (tenant not using SAML)."
fi
echo

# ---------- 5) SCIM Smoke Tests ----------
blue "[5/7] SCIM checks (if applicable)"
if [[ "$scim_enabled" == "true" ]]; then
  [[ -n "$scim_base" ]] || fail "Discovery missing scim_base"
  [[ -n "$SCIM_TOKEN" ]] || fail "SCIM enabled but SCIM_TOKEN not provided."
  SCIM_AUTH="Authorization: Bearer ${SCIM_TOKEN}"
  scim_root="$APP_BASE${scim_base%/}"

  # Create a unique test user
  uid="sis-test-$(date +%s)"
  user_payload=$(cat <<JSON
{
```

sis_preflight.sh (part 5 of 6)

```
    "schemas":["urn:ietf:params:scim:schemas:core:2.0:User"],
    "userName":"$uid",
    "name":{"givenName":"SIS","familyName":"Test"},
    "emails":[{"value":"$uid@example.invalid","primary":true}],
    "active":true,
    "externalId":"$uid"
  }
  JSON
  )

    blue "SCIM Create User: $scim_root/Users"
    resp=$(http POST "$scim_root/Users" "$SCIM_AUTH" "$user_payload")
    code=$(status_code "$resp")
    body=$(body_only "$resp")
    [[ "$code" == "201" || "$code" == "200" ]] || fail "SCIM create user failed (got $code): $body"
    scim_user_id=$(json_get "$body" ".id")
    [[ -n "$scim_user_id" ]] || fail "SCIM create response missing user id"
    pass "SCIM create user OK (id=$scim_user_id)"

    blue "SCIM Get User"
    resp=$(http GET "$scim_root/Users/$scim_user_id" "$SCIM_AUTH")
    code=$(status_code "$resp")
    [[ "$code" == "200" ]] || fail "SCIM get user failed (got $code)"
    pass "SCIM get user OK"

    blue "SCIM Deactivate User (PATCH active=false)"
    patch_payload='{"schemas":["urn:ietf:params:scim:api:messages:2.0:PatchOp"],"Operations":[{"op":"Replace","p
    resp=$(http PATCH "$scim_root/Users/$scim_user_id" "$SCIM_AUTH" "$patch_payload")
    code=$(status_code "$resp")
    [[ "$code" == "200" ]] || fail "SCIM patch deactivate failed (got $code): $(body_only "$resp")"
    body=$(body_only "$resp")
    active=$(json_get "$body" ".active")
    [[ "$active" == "false" ]] || fail "SCIM patch did not result in active=false"
    pass "SCIM deactivate OK"

    blue "SCIM Optional Delete User"
    resp=$(http DELETE "$scim_root/Users/$scim_user_id" "$SCIM_AUTH")
    code=$(status_code "$resp")
    if [[ "$code" == "204" || "$code" == "200" || "$code" == "202" ]]; then
      pass "SCIM delete OK (code=$code)"
    else
      yellow "SCIM delete not supported or blocked (code=$code). Disable-only can be acceptable if tenant uses c
    fi

  else
    yellow "Skipping SCIM checks (SCIM not enabled)."
  fi
  echo

  # ---------- 6) Audit Log Reachability ----------
  blue "[6/7] Audit logging checks"
  if [[ -n "$audit_base" ]]; then
    audit_url="$APP_BASE${audit_base%/}/events?tenant=$TENANT&limit=5"
    resp=$(http GET "$audit_url" "$ADMIN_AUTH")
    code=$(status_code "$resp")
    body=$(body_only "$resp")
    [[ "$code" == "200" ]] || fail "Audit events endpoint not reachable (got $code): $audit_url"
    pass "Audit endpoint reachable (verify events contain auth + scim + admin config entries)"
  else
    yellow "Discovery missing audit_base; cannot verify audit endpoint."
```

sis_preflight.sh (part 6 of 6)

```
fi
echo

# ---------- 7) Manual SSO Validation Steps ----------
blue "[7/7] Manual SSO validation (required)"
echo "MANUAL STEP A - Login:"
echo "  1) Open: $APP_BASE/org/$TENANT/login  (or your tenant login route)"
echo "  2) Complete IdP login."
echo "  3) Confirm you land in the correct tenant context."
echo "  PASS CRITERIA: correct tenant, correct user identity, no unexpected prompts."
echo
echo "MANUAL STEP B - Policy enforcement:"
echo "  - If profile is SIS-Enterprise:"
echo "    * Confirm password login is disabled."
echo "    * Confirm IdP MFA is enforced (user experiences MFA when policy demands)."
echo
echo "MANUAL STEP C - Deprovision cut-off:"
echo "  1) Disable a real test user in IdP (or via SCIM active=false)."
echo "  2) Attempt access using an existing session."
echo "  PASS CRITERIA: access revoked within SLA (Enterprise target <2 min; Core max <15 min)."
echo
echo "SIS Preflight script completed."
green "AUTOMATED CHECKS PASSED (manual steps still required for GO)."
```