SUNIMOD FIELD NOTES

The Employee Login You Disabled May Not End Access: How Orphaned Tokens Turn Offboarding Into a Breach Path

Disabling a former worker's main login may not revoke every session, OAuth grant, personal token, shared credential, or standalone account. Learn how lingering access works, how to audit it safely, and how to build offboarding that produces verifiable evidence.

BY Christopher Jacobson

PUBLISHED July 21, 2026

TOPIC CISSP, Domain 1

https://sunimod.com/orphaned-oauth-tokens-employee-offboarding-breach-path/

The central lesson

Disabling a person's primary login is an important action, but it is not proof that every access path associated with that person has ended. Modern businesses distribute access across identity providers, cloud consoles, source repositories, customer relationship systems, marketing platforms, payment tools, website administration, hosting dashboards, browser sessions, mobile applications, automation, and third-party integrations. Some credentials are checked against the primary account on every use. Others represent an authorization that was issued earlier and may have a separate lifetime, owner, revocation method, or audit trail.

The durable control is therefore broader than "turn off the account." A dependable offboarding process identifies every access path, performs the correct revocation action for each credential type, transfers business-owned resources, rotates shared secrets when necessary, records evidence, tests that access is actually denied, and escalates anything that cannot be verified.

Educational and defensive scope

The examples below use fictitious systems, reserved example domains, invented token values, conceptual provider interfaces, and local-only Python code. The laboratory makes no network requests, uses no real account, and does not interact with an identity provider, cloud platform, email tenant, customer database, or software-as-a-service product.

Audit, test, or change only systems you own or are explicitly authorized to assess. Do not attempt to validate offboarding by using a former worker's real token, session, password, device, or account. Use approved administrative reports, provider-supported revocation functions, isolated test tenants, designated test identities, and documented verification procedures.

In this guide

1. What the vulnerability is

This weakness is an identity-lifecycle and authorization gap. It appears when the business treats a person's central account as the complete representation of that person's access, even though other systems have issued sessions, tokens, local accounts, integration grants, shared credentials, or ownership rights that are managed separately.

It is not necessarily a software defect with a public vulnerability number. It is often a design and process condition: one offboarding event reaches only part of the access estate, yet the business records the work as complete.

Authentication, authorization, and delegation are different

Three concepts are easy to collapse into one:

- **Authentication** answers, "Can this actor prove an identity right now?" A password, passkey, multifactor prompt, client certificate, or session cookie may participate.
- **Authorization** answers, "What may this identity or application do?" Roles, scopes, policies, groups, and resource permissions make that decision.
- **Delegation** allows one application to act with permission granted by a user, service, or administrator. OAuth access and refresh tokens are common examples.

Disabling interactive authentication can prevent a new password or passkey sign-in while leaving an already-issued authorization grant untouched. The exact behavior depends on the provider and token architecture. That uncertainty is itself operationally important: the business must verify the behavior rather than infer it from a green “account disabled” status.

When the gap exists

The offboarding gap exists when one or more of these conditions are true:

- The organization has no authoritative list of systems, accounts, grants, sessions, tokens, devices, automation, and shared credentials associated with a person.
- The workforce directory disables the primary identity, but connected systems do not receive or honor the event.
- A third-party application stores a refresh token or long-lived credential that remains active after the user’s interactive login is disabled.
- A resource server accepts a self-contained access token until expiration and has no immediate revocation check.
- A worker created a personal access token, app password, deployment key, local account, or integration outside the central identity provider.
- A business process depends on a shared password, API key, recovery code, mailbox, or service credential known to the departing person, but the secret is not rotated.
- Files, forms, automations, domains, advertising accounts, repositories, or customer records remain owned by a departed identity and become difficult to administer.
- The checklist records that a task was assigned, not that the provider confirmed the access path was closed.
- An exception has no owner, expiration date, compensating control, or management decision.

Do not assume every provider behaves the same way

Some platforms invalidate sessions and grants when an account is disabled. Others require separate actions for sessions, refresh tokens, application consents, app passwords, personal tokens, service principals, or local accounts. Some actions propagate quickly; others have a delay. Some access tokens are checked centrally on every request; others remain usable until a short expiration window closes. Use current official documentation and an authorized test identity to prove the behavior that matters to your environment.

Access layer	Example credential or right	What a primary-account disable may accomplish	What still needs verification
Interactive identity	Password, passkey, multifactor sign-in	May stop new interactive sign-ins	Sessions, recovery methods, emergency paths, federated access, and propagation
Browser or mobile session	Session cookie or device session	May or may not terminate an existing session	Session revocation, device sign-out, token cache, and session lifetime

Access layer	Example credential or right	What a primary-account disable may accomplish	What still needs verification
Delegated application	OAuth access or refresh token	May or may not revoke the authorization grant	Grant revocation, related tokens, scopes, client ownership, and last use
Developer or platform access	Personal access token, deployment token, SSH key	Often depends on the specific platform	Token inventory, key removal, repository access, runners, and automation
Standalone software account	Local username and password at a SaaS vendor	May be unaffected by the central directory	Direct disablement, ownership transfer, billing role, and recovery contact
Shared or nonhuman access	API key, shared password, service credential	Usually unaffected unless explicitly linked	Rotation, dependent jobs, secret storage, owner, and rollback plan

2. Credentials that can outlive the primary login

An accurate offboarding inventory distinguishes credential types because each one fails, expires, and revokes differently.

Browser sessions and cookies

A user who authenticated earlier may have a session cookie that the application accepts without asking for the password again. Resetting a password does not universally destroy every existing session. The application may maintain a server-side session record, use a signed self-contained cookie, cache authorization, or depend on a separate identity service. Offboarding must identify the provider's "revoke sessions," "sign out devices," "revoke sign-in sessions," or equivalent control and then verify the result.

OAuth access and refresh tokens

An access token is presented to an API to reach a protected resource. A refresh token can be used to request new access tokens without repeating the full interactive authorization flow. The OAuth framework treats refresh tokens as credentials bound to a client, and the token-revocation specification defines a separate mechanism for invalidating them. The current OAuth security best-practice document also describes refresh tokens as attractive targets because successful replay can mint access tokens on behalf of the resource owner.

This matters during offboarding because the authorization server may treat "user cannot sign in interactively" and "this previously approved client grant is revoked" as separate state. The standard allows automatic revocation after security events such as a password change or logout, but that behavior is not a universal guarantee. A responsible workflow invokes the provider's supported revocation mechanism and records the result.

Access-token architecture also affects the remaining window:

- **Reference or introspected tokens** can be checked against current server-side state, allowing a resource server to learn that a token is no longer active.
- **Self-contained tokens** may be validated locally by signature and claims. Immediate invalidation can require a deny list, a revocation epoch, a changed signing or subject state, backend coordination, or a deliberately short lifetime.

- **Refresh-token rotation** can help detect replay, but it does not replace intentional grant revocation when a person's relationship with the business ends.
- **Sender-constrained tokens** reduce the usefulness of a stolen token when the attacker lacks the associated key, but they do not eliminate the need to remove authorization that is no longer legitimate.

API keys, app passwords, and personal tokens

Developer platforms, hosting providers, ecommerce systems, source repositories, analytics tools, and automation services may allow a user to create a personal access token or app-specific credential. These values can be stored in a command-line configuration file, continuous-integration secret, password manager, environment variable, desktop application, or vendor integration. A central password reset may have no effect on them unless the platform explicitly links their validity to the account state.

The review must capture the token's owner, scopes, creation time, last use, expiration, allowed network context, dependent jobs, and revocation status. Do not copy token values into an inventory or ticket. Store identifiers and evidence locations, not secrets.

Service identities and shared secrets

A departing worker may know or control credentials that are not named after them: a shared hosting administrator, a generic marketing account, a database export key, a service principal, a DNS provider credential, an automation webhook, or a recovery code. Simply deleting the person's account cannot remove knowledge of a shared secret. The secret must be rotated, dependent systems must be updated safely, and ownership must move to a current accountable role.

Service identities should not be silently treated as employee accounts. They need their own owner, purpose, least-privilege scope, noninteractive authentication method, review schedule, expiration or rotation policy, and emergency disable procedure.

Standalone SaaS accounts and resource ownership

Not every application participates in single sign-on or automated provisioning. Teams often adopt niche tools with a direct username and password. A former worker may also remain the sole owner of dashboards, advertising accounts, forms, reports, repositories, API clients, scheduled exports, domain properties, or billing relationships. Offboarding must close unauthorized access without destroying business records or breaking operations.

That requires an ordered process: identify the asset, preserve required records, transfer ownership, replace recovery contacts, verify current administrators, remove the former identity, and test the business workflow.

3. How the attack chain works

MITRE ATT&CK documents token theft and the use of application access tokens as ways adversaries can bypass the normal authentication process and reach cloud or software-as-a-service resources. The offboarding gap makes that behavior more damaging because the organization may believe the identity is already closed.

1. **Access accumulates over time.** A worker signs into the central directory, authorizes applications, creates tokens, receives local vendor accounts, joins groups, configures automations, and learns shared operational credentials.
2. **The relationship changes.** The person leaves, changes role, completes a contract, or no longer needs a particular system. Human resources or management creates an offboarding request.
3. **The workflow reaches only the obvious account.** IT disables the primary login, resets the password, collects a device, and marks the ticket complete.
4. **A separate authorization remains active.** A refresh token, session, personal token, local account, app password, service credential, or shared secret was not included in the workflow.
5. **The credential is retained or obtained.** It may remain on a returned device, in a browser profile, in a legitimate integration, in a personal development environment, in an automation service, or in another location compromised by an attacker. The scenario does not require the former worker to be malicious.
6. **The service receives a technically valid credential.** The request reaches an API or application that checks the token, session, key, or account according to its own state. If the credential is still active, the primary directory's disabled-login flag may never be consulted.
7. **Authorized-looking activity occurs.** The actor reads records, exports data, changes configuration, deploys code, creates additional credentials, alters forwarding or automation, changes billing details, or accesses internal conversations within the permissions already granted.
8. **Detection is delayed.** Logs may show the original user, client application, or service identity rather than an obviously malicious account. The business may suppress concern because the offboarding ticket says "complete."
9. **The blast radius expands.** A surviving token or account can expose connected data, allow additional persistence, disrupt customer-facing systems, or create a path into partners and downstream services.

Conditions that determine severity

The same gap can range from a harmless stale account to a serious incident. Severity depends on:

- The scopes and privileges attached to the surviving credential.
- Whether the credential can read, export, alter, delete, deploy, administer users, change billing, or create more credentials.
- How long the token, session, key, or account remains valid.
- Whether access is bound to a managed device, client certificate, source network, or proof-of-possession key.
- Whether the resource server checks current subject state or only validates a token locally.
- Whether logs identify the subject, client, credential type, source context, and exact actions.
- How quickly offboarding exceptions and post-termination activity are detected.
- The sensitivity of customer, employee, financial, operational, source-code, and authentication data reachable through the access path.
- Whether one credential can create another, change policy, disable logging, or reach connected systems.

4. Why account disablement is not enough

An intentionally incomplete offboarding handler

The following conceptual function performs two useful actions, but then records completion too early. The interfaces are fictitious and the example is not a production template.

Python · intentionally incomplete offboarding handler

```
def offboard_user(user_id: str) -> None:
    directory.disable_interactive_login(user_id)
    directory.reset_password(user_id)

    audit.write(
        "worker_offboarded",
        {"user_id": user_id, "status": "complete"},
    )
```

The function disables interactive login and resets the password. It does not ask what else represents the user's authority. Missing actions may include:

- Revoking browser and mobile sessions.
- Revoking OAuth grants, refresh tokens, access tokens, and application consents.
- Removing app passwords, personal access tokens, deployment tokens, SSH keys, and device certificates.
- Disabling local accounts in systems that do not use the central directory.
- Removing group, role, repository, cloud, hosting, website, database, and vendor access.
- Rotating shared secrets and generic account passwords the person knew.
- Transferring ownership of business records, automations, forms, dashboards, repositories, and integrations.
- Checking for access used after the effective termination time.
- Capturing provider evidence and an independent verification result.

The most dangerous line is not an API call. It is the audit record that says `status` is `complete` without evidence that the complete access estate was addressed.

The hidden assumption in token exchange

The next conceptual function illustrates a provider that validates the delegated grant but does not re-check current workforce status. That can be a legitimate architectural choice: an authorization server is not automatically connected to the employer's personnel system.

Python · conceptual refresh-token exchange with no workforce-state check

```
def exchange_refresh_token(token: str) -> AccessToken:
    grant = grants.find_active(token)
    if grant is None:
        raise InvalidGrant("refresh token is unknown or revoked")

    return issue_access_token(
        subject=grant.subject,
        client_id=grant.client_id,
        scopes=grant.scopes,
```

```
    )
```

If the grant remains active, the function issues a new access token. The user's inability to start a new interactive session is irrelevant because the refresh token represents an earlier delegated authorization. The control failure happened upstream: the offboarding workflow did not revoke the grant.

> **This is not automatically a provider bug**
>
> OAuth intentionally separates clients, authorization servers, resource owners, grants, access tokens, refresh tokens, and protected resources. A provider may offer correct revocation capabilities while the customer's offboarding process never calls them. The business problem is to connect authoritative personnel events to the actual access paths and verify the outcome.

5. Safe local demonstration: one departure, two outcomes

This laboratory models two offboarding policies in memory. Both scenarios terminate the same fictitious identity and disable the same primary login. Under the vulnerable policy, the delegated grant remains active and can mint an access token. Under the hardened policy, the grant is revoked and the exchange is denied.

- Requires Python 3.10 or later.
- Makes no network requests.
- Creates no accounts and changes no external system.
- Uses only fictitious identifiers and token values.
- Models one authorization decision; it does not reproduce a real provider's complete behavior.

Save the following file as `offboarding_token_lab.py`.

Python · local-only offboarding and token-revocation demonstration

```
#!/usr/bin/env python3
"""Local-only model of offboarding and delegated token revocation."""

from __future__ import annotations

from dataclasses import dataclass, field
from typing import Callable


@dataclass
class User:
    user_id: str
    employment_status: str = "active"
    login_enabled: bool = True


@dataclass
class Grant:
    grant_id: str
    subject: str
    client_id: str
    refresh_token: str
    scopes: set[str] = field(default_factory=set)
    revoked: bool = False
```

```
@dataclass
class State:
    users: dict[str, User]
    grants: dict[str, Grant]


def build_state() -> State:
    user = User(user_id="alex.morgan@example.com")
    grant = Grant(
        grant_id="grant-demo-001",
        subject=user.user_id,
        client_id="reports.example.com",
        refresh_token="rt_demo_alex_001",
        scopes={"crm.records.read", "crm.records.export"},
    )
    return State(
        users={user.user_id: user},
        grants={grant.refresh_token: grant},
    )


def vulnerable_offboard(state: State, user_id: str) -> None:
    user = state.users[user_id]
    user.employment_status = "terminated"
    user.login_enabled = False
    # The delegated grant is left active.


def hardened_offboard(state: State, user_id: str) -> None:
    user = state.users[user_id]
    user.employment_status = "terminated"
    user.login_enabled = False

    for grant in state.grants.values():
        if grant.subject == user_id:
            grant.revoked = True


def exchange_refresh_token(
    state: State,
    refresh_token: str,
) -> tuple[bool, dict[str, object] | str]:
    grant = state.grants.get(refresh_token)
    if grant is None:
        return False, "unknown_refresh_token"
    if grant.revoked:
        return False, "grant_revoked"

    # This intentionally models a service that validates the grant
    # but does not re-check the worker's current employment status.
    access_token = {
        "token_id": "at_demo_001",
        "subject": grant.subject,
        "client_id": grant.client_id,
        "scopes": sorted(grant.scopes),
    }
    return True, access_token


def read_customer_records(
    access_token: dict[str, object],
) -> tuple[bool, str]:
    scopes = set(access_token.get("scopes", []))
    if "crm.records.read" not in scopes:
        return False, "missing_scope"
    return True, "records=3"


def run_scenario(
    label: str,
    offboard: Callable[[State, str], None],
```

```
) -> None:
    state = build_state()
    user_id = "alex.morgan@example.com"
    offboard(state, user_id)

    user = state.users[user_id]
    print(f"\n=== {label} ===")
    print(f"employment_status={user.employment_status}")
    print(f"primary_login_enabled={str(user.login_enabled).lower()}")

    exchanged, result = exchange_refresh_token(
        state,
        "rt_demo_alex_001",
    )
    if not exchanged:
        print(f"refresh_exchange=DENIED reason={result}")
        print("api_read=NOT_ATTEMPTED")
        return

    print("refresh_exchange=ALLOWED")
    api_allowed, api_result = read_customer_records(result)
    outcome = "ALLOWED" if api_allowed else "DENIED"
    print(f"api_read={outcome} {api_result}")


def main() -> int:
    run_scenario(
        "VULNERABLE OFFBOARDING",
        vulnerable_offboard,
    )
    run_scenario(
        "HARDENED OFFBOARDING",
        hardened_offboard,
    )
    print("\nNo network service or real account was used.")
    return 0


if __name__ == "__main__":
    raise SystemExit(main())
```

Run the lab

Bash · run the local demonstration

```
python3 offboarding_token_lab.py
```

A tested run produces:

Plaintext · tested local output

```
=== VULNERABLE OFFBOARDING ===
employment_status=terminated
primary_login_enabled=false
refresh_exchange=ALLOWED
api_read=ALLOWED records=3

=== HARDENED OFFBOARDING ===
employment_status=terminated
primary_login_enabled=false
refresh_exchange=DENIED reason=grant_revoked
api_read=NOT_ATTEMPTED

No network service or real account was used.
```

What the lab proves

The personnel event and primary-login state are identical in both scenarios. The result changes because the delegated grant state changes. In the vulnerable scenario, a valid refresh token is enough to obtain API access. In the hardened scenario, the authorization server's modeled grant lookup rejects the token.

The control objective is simple to state: when a person's authority ends, every credential and delegated grant that derives from that authority must either be revoked, transferred to a legitimate owner, rotated, or explicitly documented as an approved exception.

What the lab does not prove

The lab does not simulate a real identity provider, token signature, browser cookie, device-bound credential, token introspection endpoint, propagation delay, refresh-token rotation, client authentication, source-IP restriction, multifactor authentication, insider threat, malware, data exfiltration, or a provider-specific offboarding API. It isolates one design fact: disabling one login state does not alter a separate grant unless the workflow connects them.

6. How to audit your own environment

Start with people and business services, not a product list. For each employee, contractor, agency, vendor, volunteer, intern, and administrator, the business should be able to identify what they can reach, why they have it, who approved it, how it is revoked, and what evidence proves closure.

Build an access-path inventory

For each identity and each important system, record:

- The authoritative workforce identifier and current relationship status.
- The effective time for termination, transfer, leave, or role change.
- The business owner, technical owner, and person responsible for revocation.
- The central account, local account, groups, roles, privileges, and resource ownership.
- Active browser sessions, device sessions, application passwords, and recovery methods.
- OAuth clients, grants, scopes, refresh tokens, access-token behavior, and consent owner.
- Personal access tokens, API keys, deployment credentials, SSH keys, certificates, and automation secrets.
- Shared accounts and secrets that require rotation rather than simple account disablement.
- The last successful use, last review, expiration, network or device restrictions, and logging coverage.
- The provider-supported revocation action and expected propagation behavior.
- The evidence identifier, timestamp, reviewer, and verification method.
- Any exception, its business justification, compensating control, owner, approval, and expiration.

Do not place passwords, tokens, private keys, session cookies, recovery codes, or secret values in the inventory. Record metadata and secure evidence references.

A fictitious inventory with deliberate gaps

The following JSON contains no secrets. It represents a terminated user whose directory account was disabled while several other access paths remain active, unknown, unowned, or unverified.

JSON · fictitious access-lifecycle inventory


```
{
  "as_of": "2026-07-21",
  "identities": [
    {
      "identity": "alex.morgan@example.com",
      "employment_status": "terminated",
      "termination_effective_at": "2026-07-18T17:00:00Z",
      "access_paths": [
        {
          "type": "directory-account",
          "system": "Fictitious Directory",
          "status": "disabled",
          "owner": "IT Operations",
          "last_used_at": "2026-07-18T16:42:00Z",
          "revocation_verified_at": "2026-07-18T17:03:00Z",
          "scopes": []
        },
        {
          "type": "oauth-grant",
          "system": "Fictitious CRM",
          "status": "active",
          "owner": "Sales Operations",
          "last_used_at": "2026-07-20T09:14:00Z",
          "revocation_verified_at": null,
          "scopes": [
            "crm.records.read",
            "crm.records.export"
          ]
        },
        {
          "type": "personal-access-token",
          "system": "Fictitious Hosting",
          "status": "active",
          "owner": null,
          "last_used_at": "2026-07-19T04:08:00Z",
          "revocation_verified_at": null,
          "scopes": [
            "deployments.write"
          ]
        },
        {
          "type": "standalone-account",
          "system": "Fictitious Newsletter",
          "status": "unknown",
          "owner": "Marketing",
          "last_used_at": null,
          "revocation_verified_at": null,
          "scopes": [
            "campaigns.read"
          ]
        },
        {
          "type": "shared-secret",
          "system": "Fictitious Analytics Export",
          "status": "active",
          "owner": "Data Operations",
          "last_used_at": null,
          "revocation_verified_at": null,
          "rotation_required": true,
          "rotated_at": null,
          "scopes": [
            "exports.write"
          ]
```

```
        }
      ]
    }
  ]
}
```

Run a read-only gap check

The next script reads the JSON file and flags modeled conditions. It makes no provider calls, performs no revocation, and cannot certify that an account is secure. Use it only with an inventory you are authorized to review.

Python · read-only access-lifecycle inventory check

```
#!/usr/bin/env python3
"""Read-only review of a fictitious access-lifecycle inventory."""

from __future__ import annotations

import argparse
import json
from datetime import datetime
from pathlib import Path
from typing import Any


CLOSED_STATES = {"disabled", "revoked", "removed", "expired"}
SENSITIVE_SCOPE_WORDS = {
    "admin",
    "billing",
    "delete",
    "export",
    "impersonate",
    "users",
    "write",
}


def parse_args() -> argparse.Namespace:
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "inventory",
        type=Path,
        help="Path to an authorized access inventory JSON file.",
    )
    return parser.parse_args()


def parse_time(value: str | None) -> datetime | None:
    if not value:
        return None
    return datetime.fromisoformat(value.replace("Z", "+00:00"))


def sensitive_scopes(scopes: list[str]) -> list[str]:
    return sorted(
        scope
        for scope in scopes
        if any(
            word in scope.lower()
            for word in SENSITIVE_SCOPE_WORDS
        )
    )


def evaluate_identity(identity: dict[str, Any]) -> list[str]:
    if identity.get("employment_status") != "terminated":
        return []
```

```
    subject = identity.get("identity", "unknown-identity")
    terminated_at = parse_time(
        identity.get("termination_effective_at")
    )
    findings: list[str] = []

    for path in identity.get("access_paths", []):
        system = path.get("system", "unknown-system")
        path_type = path.get("type", "unknown-type")
        label = f"{system} [{path_type}]"
        status = str(path.get("status", "unknown")).lower()

        if status not in CLOSED_STATES:
            findings.append(
                f"{subject}: {label} remains {status}"
            )

        if not path.get("owner"):
            findings.append(
                f"{subject}: {label} has no accountable owner"
            )

        if not path.get("revocation_verified_at"):
            findings.append(
                f"{subject}: {label} lacks revocation evidence"
            )

        last_used_at = parse_time(path.get("last_used_at"))
        if (
            terminated_at is not None
            and last_used_at is not None
            and last_used_at > terminated_at
        ):
            findings.append(
                f"{subject}: {label} was used after termination"
            )

        risky = sensitive_scopes(path.get("scopes", []))
        if status not in CLOSED_STATES and risky:
            findings.append(
                f"{subject}: {label} retains sensitive scopes: "
                + ", ".join(risky)
            )

        if (
            path_type == "shared-secret"
            and path.get("rotation_required")
            and not path.get("rotated_at")
        ):
            findings.append(
                f"{subject}: {label} requires secret rotation"
            )

    return findings


def main() -> int:
    args = parse_args()
    data = json.loads(
        args.inventory.read_text(encoding="utf-8")
    )

    findings: list[str] = []
    for identity in data.get("identities", []):
        findings.extend(evaluate_identity(identity))

    if not findings:
        print("OK: no modeled offboarding gaps found")
        return 0

    for finding in findings:
```

```
        print(f"FINDING: {finding}")

    print(f"Total findings: {len(findings)}")
    return 1


if __name__ == "__main__":
    raise SystemExit(main())
```

Save the script as `access_lifecycle_check.py`, save the JSON as `access-inventory.json`, and run:

Bash · scan the authorized local inventory

```
python3 access_lifecycle_check.py access-inventory.json
```

The intentionally weak inventory produces findings like these:

Plaintext · tested scanner output

```
FINDING: alex.morgan@example.com: Fictitious CRM [oauth-grant] remains active
FINDING: alex.morgan@example.com: Fictitious CRM [oauth-grant] lacks revocation evidence
FINDING: alex.morgan@example.com: Fictitious CRM [oauth-grant] was used after termination
FINDING: alex.morgan@example.com: Fictitious CRM [oauth-grant] retains sensitive scopes:
    crm.records.export
FINDING: alex.morgan@example.com: Fictitious Hosting [personal-access-token] remains active
FINDING: alex.morgan@example.com: Fictitious Hosting [personal-access-token] has no
    accountable owner
FINDING: alex.morgan@example.com: Fictitious Hosting [personal-access-token] lacks
    revocation evidence
FINDING: alex.morgan@example.com: Fictitious Hosting [personal-access-token] was used after
    termination
FINDING: alex.morgan@example.com: Fictitious Hosting [personal-access-token] retains
    sensitive scopes: deployments.write
FINDING: alex.morgan@example.com: Fictitious Newsletter [standalone-account] remains unknown
FINDING: alex.morgan@example.com: Fictitious Newsletter [standalone-account] lacks
    revocation evidence
FINDING: alex.morgan@example.com: Fictitious Analytics Export [shared-secret] remains active
FINDING: alex.morgan@example.com: Fictitious Analytics Export [shared-secret] lacks
    revocation evidence
FINDING: alex.morgan@example.com: Fictitious Analytics Export [shared-secret] retains
    sensitive scopes: exports.write
FINDING: alex.morgan@example.com: Fictitious Analytics Export [shared-secret] requires
    secret rotation
Total findings: 15
```

Interpret findings as questions, not automatic verdicts

A finding means the inventory contains a condition that deserves review. It does not prove compromise. An active shared secret may already be isolated from the former worker; an unknown account status may be a documentation defect; an OAuth grant may have been revoked at the provider while evidence failed to sync. Resolve each item against authoritative provider state and business context.

The most valuable outcome is not a zero-finding report produced by incomplete data. It is an inventory whose coverage, ownership, evidence, and exceptions are trustworthy enough to support decisions.

7. How to design verifiable offboarding

A reliable process coordinates personnel, business ownership, identity administration, application administration, security review, records handling, and operational continuity. It should be fast enough for urgent departures and structured enough to avoid destructive improvisation.

Use an authoritative trigger and effective time

The workflow needs one approved source for the person's status and an exact effective time. That event should include the unique subject identifier, relationship type, manager or sponsor, risk level, required timing, and systems known to be critical. Email alone is a weak system of record because it is difficult to correlate, retry, measure, and prove.

Urgent or involuntary departures may require coordinated action at the effective time. Routine contract completion may allow preparation in advance. Role changes should remove old access before or as new access is granted, rather than accumulating both indefinitely.

Revoke by credential type

Each access path needs the action that actually closes it:

- **Primary account:** disable sign-in, remove active roles and groups, remove recovery methods, and verify propagation.
- **Sessions:** revoke current sessions and device tokens, then test that an existing session can no longer reach protected resources.
- **OAuth grants:** revoke the grant or refresh token through the provider-supported mechanism, remove unnecessary consent, and confirm related tokens are no longer accepted according to the provider's documented behavior.
- **Personal tokens and keys:** enumerate and revoke them at the issuing platform; inspect dependent automation before removal.
- **Standalone accounts:** transfer ownership where needed, change recovery contacts, disable or remove the account, and verify current administrators.
- **Shared credentials:** rotate the secret, update authorized dependencies through a controlled deployment, confirm the old value fails, and monitor for breakage or reuse.
- **Managed devices:** remove access certificates, management enrollment, local profiles, cached secrets, and remote-access capability according to policy and legal requirements.
- **Physical and support channels:** close badges, help-desk verification paths, call-center PINs, vendor support contacts, and emergency reset routes that can restore digital access.

Transfer ownership before removing the identity

Security and continuity can conflict when a departed account is the sole owner of customer files, automations, dashboards, billing profiles, source repositories, forms, calendars, domains, or integration clients. The answer is not to preserve an active personal identity indefinitely. Transfer business-owned assets to an approved role or service identity, preserve records according to policy, test the workflow, and then close the personal access path.

A provider-neutral workflow example

The following YAML is a conceptual checklist for a workflow engine. It is not syntax for a specific identity, human-resources, ticketing, or SaaS product.

YAML · conceptual evidence-driven offboarding workflow

```yaml
event:
  type: workforce.access-end
  subject: alex.morgan@example.com
  effective_at: 2026-07-18T17:00:00Z
  reason: employment-ended

workflow:
  completion_policy: all-required
  steps:
    - id: preserve-authorized-business-records
      owner: records-owner
      evidence_required: true
    - id: disable-primary-identity
      owner: identity-operations
      evidence_required: true
    - id: revoke-browser-sessions
      owner: identity-operations
      evidence_required: true
    - id: revoke-oauth-grants
      owner: application-owners
      evidence_required: true
    - id: revoke-personal-tokens
      owner: platform-owners
      evidence_required: true
    - id: disable-standalone-accounts
      owner: application-owners
      evidence_required: true
    - id: rotate-shared-credentials
      owner: secret-owners
      evidence_required: true
    - id: transfer-resource-ownership
      owner: business-owner
      evidence_required: true
    - id: verify-post-termination-denial
      owner: security-reviewer
      evidence_required: true

exceptions:
  unresolved_action: block-close-and-escalate
  require_owner: true
  require_expiration: true
```

Make proof of completion a first-class output

For each required action, retain an evidence record that answers:

- Which subject and system were affected?
- Which access path and credential type were addressed?
- What exact action was requested?
- When did the provider accept it?
- What provider event, object, or audit identifier supports the claim?
- Who independently verified the result, and how?
- Was access tested after the effective time?
- What failed, what was retried, and what remains unresolved?
- If an exception exists, who owns it and when does it expire?

A better definition of done

Offboarding is complete when all required access paths are closed or deliberately transferred, shared secrets are rotated where necessary, business assets remain usable, provider evidence is retained, post-termination access checks show the expected denial, and every exception has a current owner and expiration. A ticket status by itself is not proof.

8. How to detect use after offboarding

Prevention can fail, provider behavior can change, and inventories can miss systems. Detection should therefore compare access events with authoritative workforce status.

High-value detection signals

- Any successful sign-in, session refresh, token exchange, API call, repository action, cloud-console action, or administrative change after the effective termination time.
- Use of an OAuth grant or personal token issued long before a password change or departure.
- Activity from a client application that no current owner recognizes.
- Access from a new device, automation platform, network, geography, or user agent after termination.
- Data export, mailbox forwarding, repository cloning, deployment, credential creation, role change, billing change, or logging change by a terminated subject.
- A standalone account that remains active after the central identity is disabled.
- A shared credential used after its documented rotation time.
- Revocation tasks that failed, timed out, returned ambiguous responses, or were never independently verified.

A conceptual post-termination query

This provider-neutral SQL assumes access events and workforce identities have been normalized into local tables. Adapt the schema, time zones, event semantics, and retention to your authorized environment.

SQL · detect allowed activity after termination

```sql
SELECT
    e.occurred_at,
    e.subject_id,
    e.system_name,
    e.credential_type,
    e.action,
    e.outcome,
    e.source_context
FROM access_events AS e
JOIN workforce_identities AS i
    ON i.subject_id = e.subject_id
WHERE i.employment_status = 'terminated'
  AND e.occurred_at >= i.termination_effective_at
  AND e.outcome = 'allowed'
ORDER BY e.occurred_at ASC;
```

The query should feed an investigation workflow, not an automatic accusation. Time synchronization, delayed ingestion, service-account attribution, approved preservation work, and provider-specific event semantics can create false positives. Preserve the raw event, identify the credential type, confirm the effective time, and involve the correct business and security owners.

Measure control health

Useful internal measures include:

- Percentage of in-scope systems connected to an authoritative identity inventory.
- Percentage of departures whose required revocations were verified by the effective time.
- Median and maximum time from authoritative event to verified denial.
- Number and age of unresolved offboarding exceptions.
- Number of terminated identities with active accounts, grants, sessions, personal tokens, or owned resources.
- Number of shared secrets requiring rotation after a departure.
- Percentage of critical applications that expose adequate revocation and audit evidence.
- Time to detect and investigate post-termination activity.
- Percentage of role changes that removed prior access instead of only adding new access.

These are management indicators, not universal thresholds. Set targets according to business criticality, data sensitivity, contractual obligations, available provider controls, and tolerance for delayed revocation.

9. How to respond to suspected lingering access

When a token, session, account, or shared secret may have remained usable after offboarding, treat the situation as a potential identity incident until the facts are known.

> **Do not test by replaying a real credential**
>
> Using a former worker's token or session can alter evidence, exceed authorization, expose data, and create legal or personnel complications. Validate through administrative state, audit logs, provider-supported test identities, controlled revocation, and authorized investigation procedures.

1. **Establish authority and leadership.** Assign an incident owner, technical lead, business owner, and communication path. Involve legal counsel, human resources, privacy, insurance, or law enforcement as appropriate to the facts and jurisdiction.
2. **Preserve evidence.** Retain relevant identity, token, application, API, cloud, source-control, endpoint, help-desk, and workforce events. Record time sources and collection methods.
3. **Contain in layers.** Disable the primary identity, revoke sessions and grants, remove personal tokens and keys, disable local accounts, rotate shared secrets, and restrict affected applications or networks as needed.
4. **Protect continuity.** Transfer business-owned resources, update automations safely, preserve required records, and avoid destroying data needed for operations or investigation.

5. **Determine exposure.** Identify the credential, its scopes, issuance and last-use history, source context, actions taken, data reached, changes made, and any credentials created from it.
6. **Search for persistence.** Review new OAuth applications, service principals, tokens, forwarding rules, API clients, webhooks, users, roles, deployment keys, scheduled jobs, and recovery methods.
7. **Eradicate and verify.** Remove unauthorized persistence, confirm old credentials fail through approved methods, validate current administrators, and monitor for renewed activity.
8. **Assess obligations.** Evaluate customer, employee, contractual, regulatory, insurer, and partner notification duties based on confirmed facts. Requirements vary; obtain qualified advice.
9. **Improve the system.** Update the inventory, workflow, connectors, evidence model, detection rules, training, and exception process that allowed the gap.

10. Potential business repercussions

The consequences depend on the surviving credential and the systems behind it. Potential effects include:

Repercussion	How the offboarding gap can contribute	What determines severity
Unauthorized data access	A still-active session, grant, token, or local account reads customer, employee, financial, operational, or source-code data.	Scope, data sensitivity, volume, logging, duration, and whether information was exported.
Fraud or unauthorized transactions	Existing access changes payment details, issues refunds, alters orders, modifies advertising spend, or reaches billing functions.	Approval controls, transaction limits, segregation of duties, detection speed, and recoverability.
Website or application compromise	A personal token, hosting account, deployment key, repository role, or shared secret changes code or configuration.	Deployment protections, review requirements, token privilege, rollback capability, and monitoring.
Operational disruption	An actor deletes records, disables integrations, changes ownership, revokes legitimate administrators, or breaks automation.	Business dependence, backup quality, alternate workflows, and administrative recovery paths.
Additional persistence	A surviving credential creates another token, OAuth application, account, key, forwarding rule, webhook, or service identity.	Privilege level, policy controls, alerting, and the completeness of the investigation.
Privacy and contractual response	Unauthorized access or disclosure may trigger review under applicable laws, contracts, customer commitments, and insurance terms.	Jurisdiction, data type, affected people, confirmed actions, agreements, and qualified legal analysis.
Evidence and attribution problems	Activity appears under the former user or a legitimate client application, while incomplete logs make intent and scope unclear.	Audit retention, event detail, time synchronization, client identification, and investigation readiness.
Customer and partner trust loss	The business cannot explain why access remained active or what information and systems are trustworthy.	Impact, transparency, response quality, contractual relationships, and prior control maturity.
Emergency cost and strategic delay	Specialists, overtime, legal review, customer support, system rebuilding, secret rotation, and reconciliation interrupt planned work.	System complexity, documentation, vendor support, recovery capability, and incident duration.

Secondary and delayed effects

- **Lost business continuity.** A team may discover that essential automations or accounts were personally owned only after access is removed.
- **Over-retention of risky accounts.** Fear of breaking operations can lead staff to keep former identities active indefinitely.
- **Inaccurate compliance evidence.** A completed checklist may not withstand scrutiny if it lacks provider records and verification.
- **Insurance friction.** Delayed discovery, unclear timelines, incomplete evidence, or failure to follow policy can complicate a claim; policy terms vary.
- **Broader supply-chain exposure.** A surviving credential may reach customer tenants, partner portals, managed websites, integrations, or downstream data flows.
- **Insider-threat confusion.** A malicious former worker is only one scenario. Malware, a stolen device, a compromised integration, or an unrelated attacker can exploit the same neglected credential.
- **Long-tail cleanup.** The business may need to review months of access, reissue credentials, reconcile changed records, contact customers, and rebuild trust in affected workflows.

11. How to build a durable access-lifecycle program

Offboarding becomes dependable when it is part of a full identity lifecycle rather than an emergency task performed only when someone leaves.

Govern the whole lifecycle

- **Join:** create only approved access, assign an owner, record purpose, and use least privilege.
- **Change:** remove access that no longer matches the person's role before adding new privilege.
- **Review:** periodically confirm accounts, grants, tokens, roles, service identities, owners, and exceptions.
- **Leave:** revoke, transfer, rotate, verify, and preserve evidence at the effective time.
- **Retire systems:** remove connectors, accounts, tokens, integrations, DNS, automation, and ownership records when a service is no longer used.

Assign clear ownership

Human resources or management owns the workforce event. Business leaders own the decision about who still needs access and who receives transferred assets. Application owners know the system's account and revocation model. Identity and platform administrators execute technical actions. Security defines risk-based requirements and detection. Records, privacy, and legal stakeholders guide preservation and obligations. No single team can infer the complete process alone.

Design for failure and exceptions

External APIs fail, vendors lack automation, ownership records drift, and urgent departures happen outside business hours. The workflow should support retries, idempotent actions, manual steps, secure evidence upload, escalation, and a visible incomplete state. An exception must never be a permanent blank field. Require an owner, reason, approval, compensating control, review date, and expiration.

Test with designated identities

Use test accounts and representative applications to prove that:

- The authoritative event starts the workflow.
- Each connector addresses the intended account, grant, session, token, or role.
- Revocation reaches the provider within the required time.
- Existing sessions and delegated access fail as expected.
- Transferred files and automations still work for the new owner.
- Evidence is complete and understandable to an independent reviewer.
- Alerts fire when post-termination access is allowed.
- Failed connectors remain visible and are not silently marked complete.

12. A solvable Sunimod project: Access Lifecycle Assurance

For a business using WordPress, custom applications, ecommerce, cloud hosting, customer portals, source repositories, productivity software, marketing platforms, or connected workflows, this is a finite business-analysis and engineering problem. The work can be scoped around the systems and identities that create the greatest operational and data risk.

Turn scattered offboarding tasks into one operating workflow

A common problem is fragmentation: workforce status lives in one system, application ownership in a spreadsheet, tokens in vendor consoles, local accounts in team knowledge, evidence in tickets, and exceptions in email. A focused Access Lifecycle Assurance application can connect those records without becoming a vault for secrets.

The application could track identities, employment or contract state, system owners, access paths, credential types, scopes, required actions, connector results, manual tasks, evidence references, verification, exceptions, and post-termination alerts. It can show the difference between “request created,” “provider accepted,” and “access independently verified as denied.”

SQL · conceptual revocation-evidence table

```
CREATE TABLE access_revocation_evidence (
    id INTEGER PRIMARY KEY,
    subject_id TEXT NOT NULL,
    system_name TEXT NOT NULL,
    access_path_type TEXT NOT NULL,
    required_action TEXT NOT NULL,
    status TEXT NOT NULL
        CHECK (
            status IN (
                'pending',
                'verified',
                'failed',
                'exception'
            )
        ),
    provider_event_id TEXT,
    verified_at TEXT,
    verified_by TEXT,
```

```
    evidence_location TEXT,
    exception_owner TEXT,
    exception_expires_at TEXT
);

CREATE INDEX access_revocation_subject_status
    ON access_revocation_evidence (subject_id, status);
```

The real schema should match the organization's data-classification rules, identity sources, providers, retention needs, and reporting requirements. The system should store evidence metadata and secure references, not passwords, token values, private keys, or recovery codes.

What Sunimod can help deliver

- An authorized inventory of workforce identities, administrators, contractors, service identities, applications, websites, hosting accounts, repositories, integrations, and standalone vendor accounts.
- An access-path map that distinguishes primary accounts, sessions, OAuth grants, personal tokens, keys, shared secrets, service credentials, recovery methods, and resource ownership.
- A business-impact review that prioritizes customer data, revenue systems, operational workflows, administrative control, and high-risk credentials.
- A practical offboarding standard with owners, effective-time rules, required actions, evidence, verification, escalation, and exception handling.
- Provider-specific runbooks for systems that require manual administration.
- API, webhook, or workflow integrations where the selected providers expose suitable supported interfaces.
- A custom Access Lifecycle Assurance dashboard or internal tool that coordinates tasks, records evidence, highlights incomplete revocation, and tracks exceptions without storing secret values.
- Read-only reporting that identifies terminated identities with active or unknown access paths and highlights use after the effective time.
- Test-identity exercises that verify the workflow and document provider behavior, propagation, limitations, and rollback needs.
- Ownership-transfer workflows for files, dashboards, automations, repositories, websites, forms, domains, and integration clients.
- Documentation and handoff so the business can operate, review, and improve the process after implementation.

The result should answer practical questions: Which access paths remain after the main login is disabled? Which applications can revoke grants automatically? Which require a manual owner? Can the business prove that an old session fails? Who rotates a shared credential? What breaks if an account is removed? Which exceptions are overdue? Has any terminated identity been allowed to act since the effective time?

13. Key takeaways

- Disabling a primary login is necessary, but it is not universal proof that all access has ended.

- Sessions, OAuth grants, refresh tokens, personal tokens, app passwords, standalone accounts, service identities, and shared secrets have different revocation paths.
- A refresh token can mint new access tokens while its grant remains valid; intentional revocation is therefore part of offboarding.
- Provider behavior varies. Verify current documentation and test with authorized identities instead of assuming a password reset closes every path.
- Transfer business-owned resources before removing the personal identity so security work does not destroy continuity.
- Do not store secret values in inventories, tickets, dashboards, or quote forms. Track metadata, ownership, and evidence references.
- Offboarding is complete only when required actions are verified, post-termination access is denied, and exceptions are owned and time-limited.
- Detection should correlate allowed access events with authoritative workforce status and the exact effective time.
- A custom workflow can turn fragmented tickets and vendor consoles into measurable, repeatable, evidence-driven access lifecycle management.

14. Sources and further reading

- NIST SP 800-53 Rev. 5: Security and Privacy Controls for Information Systems and Organizations — the official control catalog includes account management and personnel-termination controls relevant to disabling access and revoking credentials.
- RFC 6749: The OAuth 2.0 Authorization Framework — defines access tokens, refresh tokens, authorization grants, clients, and protected-resource access.
- RFC 7009: OAuth 2.0 Token Revocation — specifies a revocation mechanism for refresh and access tokens and discusses architectural implications for immediate revocation.
- RFC 9700: Best Current Practice for OAuth 2.0 Security — current OAuth security guidance covering refresh-token protection, replay detection, rotation, sender constraint, privilege restriction, expiration, and revocation.
- MITRE ATT&CK: Use Alternate Authentication Material — Application Access Token, T1550.001 — describes using stolen application access tokens in place of the typical authentication process.
- MITRE ATT&CK: Steal Application Access Token, T1528 — describes theft of application access tokens used to reach cloud and software-as-a-service resources.
- MITRE ATT&CK: Valid Accounts — Cloud Accounts, T1078.004 — covers abuse of valid cloud and SaaS accounts for access, persistence, privilege, and defense evasion.
- MITRE ATT&CK: Cloud Application Integration, T1671 — describes cloud application integrations, including OAuth-based access in SaaS environments.

Sources accessed July 21, 2026. Token lifetimes, session behavior, revocation semantics, provider APIs, audit fields, and product capabilities can change. Verify current official documentation and test controls in an authorized environment before relying on them.

Turn offboarding into proof, not assumption

Hire Sunimod to map the access paths your business actually uses, identify sessions and credentials that can survive a primary-account disable, design a practical revocation workflow, connect supported systems, build an evidence dashboard, and test the result with authorized identities.

Request an access lifecycle project quote

Describe the applications, website, hosting environment, repositories, workforce process, and business outcome you need to protect. Do not submit passwords, API keys, access tokens, private keys, session cookies, recovery codes, or other secrets through the form; a safer handoff can be arranged when authorized access is required.