SUNIMOD FIELD NOTES

The Subdomain You Retired Can Be Claimed: How Dangling DNS Turns an Old SaaS Link Into an Attacker-Controlled Site.

Deleting a hosted service without removing its DNS record can leave a trusted company subdomain pointing at a resource another party may be able to claim. Learn the attack chain, run a safe localhost demonstration, assess the business impact, and build a disciplined offboarding process.

BY Christopher Jacobson

PUBLISHED July 16, 2026

TOPIC CISSP, Domain 1

https://sunimod.com/the-subdomain-you-retired-can-be-claimed-how-dangling-dns-turns-an-old-saas-link-into-an-attacker-controlled-site/

The central lesson

DNS can outlive the project, vendor, contract, employee, and account that originally justified it. A subdomain is not safe merely because the old service has been deleted. The organization must also remove or deliberately reroute the public DNS record, retire every surrounding trust relationship, and preserve an accountable record of who owns the name.

Educational and defensive scope: The lab in this guide runs only on `127.0.0.1`, uses reserved testing names, contacts no public DNS service or SaaS platform, and creates no real provider account. Test only domains, DNS zones, cloud accounts, and hosted services you own or are explicitly authorized to assess. Do not claim an apparently abandoned third-party resource as a test.

In this guide

1. What a dangling subdomain is
2. Why the weakness survives ordinary offboarding
3. How the takeover chain works
4. A safe localhost demonstration
5. What control of a subdomain does and does not grant
6. How to audit your own environment safely
7. Potential repercussions
8. What to do after suspected takeover
9. A durable remediation roadmap
10. A solvable business service

What a dangling subdomain is

A dangling subdomain is a company-controlled DNS name whose record still points toward an external resource that no longer exists, is no longer attached to the company's account, or is no longer exclusively reserved for that company. The most familiar pattern uses a DNS `CNAME` record to make a branded name an alias of a hosted platform's name.

DNS · intentionally vulnerable alias

```
$ORIGIN example.com.
support 300 IN CNAME customer-7421.saas.example.net.
```

The record says that `support.example.com` is an alias of `customer-7421.saas.example.net`. It does not say who currently owns the provider resource, whether the customer account still exists, whether the

hosted platform still reserves the identifier, or whether a different account can bind the same custom domain. DNS is performing name resolution, not contract management or tenant authorization.

Three conditions are normally required

1. **A trusted name remains published.** The organization's authoritative DNS still directs a subdomain toward a hosted platform, cloud resource, storage endpoint, CDN distribution, site builder, ticketing service, or another external destination.
2. **The intended resource is gone or detached.** The tenant, bucket, app, project, route, distribution, or custom-domain mapping has been deleted, disabled, renamed, moved, or released.
3. **The provider permits a new claim.** Another party can create or attach a resource that satisfies the provider's routing logic without proving a fresh, exclusive right to the company's subdomain.

> **A dangling record is not automatically exploitable.** Some providers reserve deleted names, require a random DNS verification value, prevent identifier reuse, retain a custom-domain ownership binding, or otherwise block a second account from receiving traffic. An error page, nonexistent target, or automated fingerprint is a reason to investigate—not proof that takeover is possible.

The business risk exists in the gap between the organization's DNS and the provider's resource lifecycle. One side still says "send traffic there." The other side may say "that identifier is available." When those two states overlap, a trusted company address can become a routing instruction to somebody else's tenant.

Why this is a business lifecycle failure

The defect is visible in DNS, but DNS is rarely the root cause. The deeper problem is that several teams can participate in one service without sharing one complete offboarding process:

- Marketing or customer operations purchases the hosted platform.
- An agency, developer, or IT administrator creates the DNS record.
- The provider controls the tenant identifier and custom-domain workflow.
- Finance cancels the subscription.
- A project manager closes the launch or migration ticket.
- Nobody is explicitly accountable for removing the public name and its surrounding trust relationships.

The service may disappear from the budget and from the provider dashboard while remaining live in the organization's public namespace. This is why modern risk-management guidance emphasizes inventories of supplier-provided services, management throughout the service lifecycle, monitoring during the relationship, and provisions for activities after a partnership or service agreement concludes. Those outcomes are directly relevant to DNS aliases, hosted applications, custom domains, and third-party integrations.

> **The problem is finite and solvable.** A useful engagement does not begin with indiscriminate internet scanning. It begins with the company's authoritative DNS zones, registrar accounts, hosting platforms, cloud subscriptions, SaaS inventory, website configuration, and named business owners. Each published hostname is mapped to a real resource, a purpose, an owner, a provider, and a retirement procedure.

How the takeover chain works

A subdomain takeover does not usually require breaking DNS cryptography, compromising the registrar, or exploiting memory corruption. The attacker takes advantage of a valid DNS record whose destination has fallen out of the organization's control.

1. **The organization publishes a branded alias.** A hostname such as `support.example.com` points to a provider-specific resource such as `customer-7421.saas.example.net`.
2. **The provider routes by tenant or custom domain.** Requests reach shared provider infrastructure. The platform uses the requested hostname, provider resource identifier, or both to decide which customer's content to serve.
3. **The business retires the service.** A subscription ends, a migration finishes, a campaign closes, a vendor is replaced, or an employee deletes the old project.
4. **The hosted resource is removed before DNS.** The company's provider tenant or mapping disappears, but the CNAME, A, AAAA, NS, or other relevant record remains in the authoritative zone.
5. **The abandoned state becomes observable.** Public DNS still reveals the target. The provider may return a recognizable "resource not found" response, a generic setup page, or another indication that the expected tenant no longer exists.
6. **A different party attempts to create the reusable resource.** If the provider allows the same identifier or accepts the existing DNS record as sufficient proof, the new account may bind the company subdomain.
7. **Traffic begins reaching the new tenant.** Employees, customers, search engines, links, bookmarks, QR codes, old emails, integrations, and automated clients continue using the branded hostname.
8. **Impact expands through surrounding trust.** The claimant may imitate the old service, collect information users submit, receive broadly scoped cookies, receive an authorization response sent to a stale redirect URI, or benefit from application rules that trust company subdomains too broadly.

DNS keeps routing after the contract ends

A CNAME is an alias in the Domain Name System. Resolvers follow the alias toward the target and eventually obtain an address. The DNS protocol does not ask whether the target belongs to the same legal entity that controls the alias. It also does not receive an automatic "subscription canceled" event from a SaaS vendor.

Cached DNS answers can remain usable until their time to live expires. Removing a record from the authoritative zone is therefore the correct containment action, but the previous answer may continue to exist in recursive resolver caches for the remainder of its earlier TTL. Planned retirement should account for that delay rather than deleting the provider resource immediately after the authoritative edit.

The provider makes the tenant decision

Hosted platforms commonly serve many customers from shared infrastructure. The IP address alone does not identify the customer. The platform receives the requested hostname and maps it to a tenant, route, site, project, bucket, or distribution. If that mapping can be recreated by another account, the provider can begin serving different content without any change to the company's DNS.

Plaintext · simplified request and routing path

```
1. Browser asks DNS for support.example.com.
2. DNS returns the CNAME customer-7421.saas.example.net.
3. The browser resolves the provider target to an address.
4. The browser connects to that address but keeps the original hostname:

GET /signin HTTP/1.1
Host: support.example.com

5. The hosted platform uses the Host value to select a tenant.
```

The browser preserves the trusted hostname

The browser may resolve the provider's target name to reach an IP address, but the HTTP request still identifies the original host the user entered. HTTP's `Host` field—or the corresponding `:authority` value in newer HTTP versions—allows one server address to route requests for many hostnames. That is what makes legitimate custom domains work, and it is also why a stale custom-domain binding can be dangerous.

Why the order of operations matters

Provisioning and deprovisioning should use opposite sequences:

- **Provision:** create and secure the provider resource, complete fresh ownership verification, then publish DNS last.
- **Deprovision:** remove or reroute public DNS first, verify the authoritative change, allow prior TTLs to expire, then remove the provider binding and resource.

Provider-specific instructions can differ, especially when a platform requires the custom-domain binding to be removed in a particular order. The invariant is that there must never be a period when public DNS points at a provider identifier that a different account can claim.

> **DNSSEC is not the missing control.** DNSSEC can authenticate that the published DNS answer came from the signed zone. It cannot determine whether the third-party tenant behind an authentic CNAME is still owned by the intended customer. The dangerous record can be perfectly authentic.

A vulnerable business scenario

Imagine a company that operated a hosted customer-help center at `support.example.com`. The implementation record looked complete when the service launched:

- The help-center tenant was created.
- The provider supplied the name `customer-7421.saas.example.net`.
- IT published a CNAME for `support.example.com`.
- The provider attached the custom domain and issued the normal site configuration.

Two years later, the company migrates support content into its primary website. Customer operations cancels the old platform. The provider tenant is deleted, but the migration ticket contains no DNS task and the DNS zone has no business owner metadata. The subdomain continues resolving to the provider.

The technically vulnerable state can be summarized as:

- **Company control:** the company still controls the DNS alias.
- **Missing control:** the company no longer controls the provider resource.
- **Potential claim path:** the provider may allow a different account to reuse the identifier or bind the custom domain.
- **Residual trust:** old email templates, bookmarks, search results, support documents, cookies, redirect URIs, or allowlists still reference the subdomain.

The absence of a legitimate page is not the main risk. The main risk is that the company continues lending its name to an endpoint whose operational ownership is unknown.

Safe localhost demonstration: the DNS route stays while the tenant changes

The following lab models the essential behavior without touching public DNS or a real provider. It starts a temporary HTTP server on `127.0.0.1`, creates an in-memory DNS-to-provider mapping, and sends requests with the custom hostname. The script shows three states: the legitimate tenant exists, the tenant is deleted while the routing record remains, and a different claimant receives the same provider resource.

- No public hostname is resolved.
- No third-party service is contacted.
- No account is created or claimed.
- No credential, browser data, or local secret is read.
- The server binds only to the local loopback interface.
- All state disappears when the process exits.

Save the file as `dangling_subdomain_lab.py`.

Python · localhost-only dangling subdomain model

```
#!/usr/bin/env python3
"""Local-only model of a dangling custom-domain route."""

from __future__ import annotations

from dataclasses import dataclass
from http.client import HTTPConnection
from http.server import BaseHTTPRequestHandler, ThreadingHTTPServer
from threading import Thread

CUSTOM_DOMAIN = "support.example.test"
PROVIDER_RESOURCE = "customer-7421.saas.example.test"


@dataclass(frozen=True)
class Tenant:
    owner: str
    content: str


DNS = {CUSTOM_DOMAIN: PROVIDER_RESOURCE}
```

```
CLAIMS: dict[str, Tenant] = {
    PROVIDER_RESOURCE: Tenant(
        owner="Example Company",
        content="Legitimate customer help center",
    )
}


class ProviderHandler(BaseHTTPRequestHandler):
    def do_GET(self) -> None:
        requested_host = self.headers.get("Host", "").split(":", 1)[0].lower()
        provider_target = DNS.get(requested_host)
        tenant = CLAIMS.get(provider_target) if provider_target else None

        if tenant is None:
            status = 404
            body = (
                "UNCLAIMED\n"
                f"custom_domain={requested_host}\n"
                f"provider_resource={provider_target}\n"
            )
        else:
            status = 200
            body = (
                f"owner={tenant.owner}\n"
                f"content={tenant.content}\n"
                f"custom_domain={requested_host}\n"
                f"provider_resource={provider_target}\n"
            )

        encoded = body.encode("utf-8")
        self.send_response(status)
        self.send_header("Content-Type", "text/plain; charset=utf-8")
        self.send_header("Content-Length", str(len(encoded)))
        self.end_headers()
        self.wfile.write(encoded)

    def log_message(self, format: str, *args: object) -> None:
        return


def fetch(port: int) -> tuple[int, str]:
    connection = HTTPConnection("127.0.0.1", port, timeout=3)
    connection.putrequest("GET", "/", skip_host=True)
    connection.putheader("Host", CUSTOM_DOMAIN)
    connection.endheaders()
    response = connection.getresponse()
    body = response.read().decode("utf-8")
    connection.close()
    return response.status, body


def show(label: str, result: tuple[int, str]) -> None:
    status, body = result
    print(f"\n=== {label} ===")
    print(f"HTTP status: {status}")
    print(body.rstrip())


def main() -> int:
    server = ThreadingHTTPServer(("127.0.0.1", 0), ProviderHandler)
    thread = Thread(target=server.serve_forever, daemon=True)
    thread.start()

    try:
        show("1. Legitimate provider tenant exists", fetch(server.server_port))

        del CLAIMS[PROVIDER_RESOURCE]
        show(
            "2. Tenant deleted while DNS remains",
            fetch(server.server_port),
        )
```

```
        CLAIMS[PROVIDER_RESOURCE] = Tenant(
            owner="Untrusted claimant",
            content="Imitation sign-in page - simulation only",
        )
        show(
            "3. Reusable provider resource claimed by another party",
            fetch(server.server_port),
        )
    finally:
        server.shutdown()
        server.server_close()
        thread.join(timeout=3)

    print("\nLocalhost only: no public DNS or SaaS platform was contacted.")
    return 0


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

Run it with the system Python interpreter:

Bash · run the localhost demonstration

```
python3 dangling_subdomain_lab.py
```

A successful run produces output like this:

Plaintext · tested localhost output

```
=== 1. Legitimate provider tenant exists ===
HTTP status: 200
owner=Example Company
content=Legitimate customer help center
custom_domain=support.example.test
provider_resource=customer-7421.saas.example.test

=== 2. Tenant deleted while DNS remains ===
HTTP status: 404
UNCLAIMED
custom_domain=support.example.test
provider_resource=customer-7421.saas.example.test

=== 3. Reusable provider resource claimed by another party ===
HTTP status: 200
owner=Untrusted claimant
content=Imitation sign-in page - simulation only
custom_domain=support.example.test
provider_resource=customer-7421.saas.example.test

Localhost only: no public DNS or SaaS platform was contacted.
```

What the lab proves

The custom hostname never changed. The simulated DNS mapping never changed. The browser-side request never changed. Only the provider's ownership state changed.

In the first state, the provider routed the company hostname to the legitimate tenant. In the second state, the same hostname pointed to an unclaimed resource. In the third state, the same routing instruction delivered content associated with a different owner. That is the core of a dangling-subdomain takeover: stable public trust combined with unstable backend ownership.

The lab deliberately does not model a real provider's registration workflow because the decisive controls vary. A real assessment must determine whether the provider reserves identifiers, requires fresh random ownership proof, validates the custom domain continuously, prevents cross-account reuse, or offers a platform-specific protection such as an alias record tied to the resource lifecycle.

What control of a subdomain does—and does not—grant

Severity should be based on evidence, not on the phrase "subdomain takeover" alone. Control of `support.example.com` does not automatically compromise `www.example.com`, the registrar, the authoritative DNS zone, every sibling subdomain, or the internal network. Browsers treat different hostnames as different origins even when they share a registrable parent domain.

The separate-origin boundary still matters

Web origin is based on scheme, host, and port. An attacker-controlled `https://support.example.com` page normally cannot read the DOM of `https://www.example.com` merely because both names end in `example.com`. It also does not inherit application sessions that are correctly scoped only to the primary host.

That boundary is important, but organizations often build additional trust above it. Severity rises when other systems treat "any subdomain of the company" as equivalent to a specifically approved application.

Parent-domain cookies can cross the boundary

A cookie created with `Domain=example.com` is eligible for matching subdomains. If the taken-over endpoint can serve HTTPS and the browser sends that cookie to `support.example.com`, the server controlling that hostname can receive the cookie in the HTTP request. The `Secure` attribute requires encrypted transport, and `HttpOnly` blocks ordinary JavaScript access, but neither attribute narrows the cookie's DNS domain scope.

Plaintext · broader and host-only cookie scope

```
# Broader than necessary: the cookie may be sent to matching subdomains.
Set-Cookie: session=DEMO_ONLY; Domain=example.com; Path=/; Secure; HttpOnly; SameSite=Lax

# Host-only: omitting Domain confines the cookie to the issuing host.
Set-Cookie: session=DEMO_ONLY; Path=/; Secure; HttpOnly; SameSite=Lax
```

Omitting the `Domain` attribute creates a host-only cookie, which is generally safer when cross-subdomain sharing is not required. Cookie scoping is not a substitute for fixing dangling DNS; it is a way to reduce the blast radius if a subdomain is ever misrouted.

Stale OAuth redirects and broad trust rules can magnify impact

An authorization server should compare redirect URIs exactly, but an exact URI can still be dangerous when the registered hostname has been retired and then taken over. A forgotten redirect such as `https://support.example.com/oauth/callback` may continue receiving authorization responses until it is removed from the client registration.

JSON · fictitious OAuth redirect inventory

```
{
  "client_id": "sunimod-demo-client",
  "redirect_uris": [
    "https://app.example.com/oauth/callback",
    "https://support.example.com/oauth/callback"
  ]
}
```

The same principle applies to other trust lists:

- API gateways or CORS middleware that approve every company subdomain.
- Content Security Policy entries that permit scripts, frames, images, or connections from `*.example.com`.
- Single sign-on or logout callbacks that still include a retired hostname.
- Webhook destinations, password-reset links, deep links, QR codes, and mobile-app universal-link configurations.
- Monitoring systems, status pages, or documentation that keep sending users to the old address.

> **HTTPS is not proof of current business ownership.** HTTPS proves that the connection is encrypted and that the endpoint presented a certificate valid for the hostname. If a provider's custom-domain workflow allows a new claimant to complete certificate issuance, the browser can show a normal secure connection to content the business did not authorize.

How to audit your own environment safely

The strongest review starts with records the organization already controls. A public subdomain-enumeration tool can find clues, but an authoritative zone export and provider inventory are more complete, less ambiguous, and easier to map to owners.

Start with an accountable inventory

For every internet-facing hostname, record at least:

- The hostname and DNS record type.
- The complete target chain, including intermediate CNAMEs.
- The business service and business owner.
- The technical owner and emergency contact.
- The provider, account, subscription, project, tenant, or resource identifier.
- Whether the resource currently exists and is visible in the expected provider account.
- The custom-domain verification method and current binding state.
- The data handled, user population, and criticality.
- Related cookies, OAuth redirects, SSO settings, CORS rules, CSP entries, webhooks, and application references.
- The last verification date, planned retirement date, and decommission evidence.

Plaintext · minimal DNS and service inventory

```
hostname,record_type,target,service_owner,technical_owner,provider,resource_id,status,last_v
    erified
support.example.com,CNAME,customer-7421.saas.example.net,Customer Operations,IT
    Operations,Fictitious SaaS,customer-7421,active,2026-07-17
events.example.com,CNAME,event-8834.saas.example.net,Marketing,Web Operations,Fictitious
    Events,event-8834,retiring,2026-07-17
```

A useful inventory is not merely a zone-file dump. The target must be connected to a real business purpose and a person who can say whether it should still exist.

Use read-only DNS checks as one evidence source

On Ubuntu, the `dig` command from the `dnsutils` package can show how an owned hostname currently resolves. The following commands are read-only, but they should still be used only for domains you are authorized to assess.

Bash · read-only DNS review for an owned hostname

```
#!/usr/bin/env bash
set -Eeuo pipefail

host="support.example.com"

printf 'Authoritative alias answer for %s:\n' "$host"
dig +noall +answer "$host" CNAME

printf '\nResolved IPv4 addresses:\n'
dig +short "$host" A

printf '\nResolved IPv6 addresses:\n'
dig +short "$host" AAAA
```

Interpret the result with the provider inventory:

- If the CNAME target differs from the approved inventory, investigate the change.
- If the target resolves but no matching resource exists in the expected provider account, treat it as urgent.
- If the target does not resolve, determine whether the provider resource is missing, the record is stale, or the service is in a failed state.
- If the provider returns a generic setup or “not found” page, identify the provider-specific ownership and reuse rules without creating a claim.
- If a hostname uses A, AAAA, NS, ALIAS, ANAME, CDN, load-balancer, or cloud-specific records instead of CNAME, trace that record type and its resource lifecycle as well.

Why a 404 or fingerprint is not proof

Automated takeover scanners compare responses with known provider fingerprints. They are valuable for prioritization, but false positives occur. A provider can return the same error for an unclaimed resource, a suspended but reserved resource, a misconfigured active resource, a regional outage, or a resource that only the original organization can restore.

Manual validation should answer two separate questions:

1. **Is the company’s DNS pointing at a resource that the company does not currently control?**

2. **Can an unrelated account actually obtain the specific provider binding required to receive traffic?**

The first question is often enough to justify remediation. The second must be answered from provider documentation, account records, support confirmation, or a written authorized test plan—not by opportunistically claiming the resource.

> **Treat unknown ownership as a finding.** Even when takeover cannot be conclusively demonstrated, a public DNS record with no accountable owner, no verified resource, and no documented purpose is an unmanaged business dependency. Remove it or bring it under control.

Potential repercussions for your business or infrastructure

The technical event is "a trusted hostname routes to content controlled by a different party." The business effect depends on how the hostname is used and what other systems trust it.

Repercussion	How it can happen	Business effect
Branded phishing	The claimant reproduces the old support, billing, document, or sign-in experience under the real company subdomain.	Employees, customers, or vendors may submit credentials, payment details, documents, or personal information because the address appears legitimate.
Customer and vendor deception	Old links, QR codes, bookmarks, invoices, email templates, and search results continue sending people to the retired hostname.	Fraud, false instructions, fake downloads, support impersonation, and disputes over whether the company controlled the page.
Cookie exposure	A sensitive cookie is scoped to the parent domain and is sent to the taken-over subdomain.	Session compromise, account takeover, privacy exposure, forced session invalidation, and difficult investigation. This impact is conditional on the actual cookie scope and endpoint capabilities.
OAuth or SSO response leakage	A retired hostname remains registered as a redirect, callback, relay, or logout destination.	Authorization codes, tokens, assertions, or account-flow data may reach the wrong endpoint. The exact result depends on protocol configuration and additional safeguards.
Application trust bypass	An API, CORS rule, CSP rule, webhook validator, or internal application broadly trusts company subdomains.	The claimant may gain access to data or browser capabilities that would not be available to an unrelated internet origin.
Malicious content or downloads	The hostname serves deceptive scripts, documents, installers, browser prompts, or redirects.	Malware exposure, credential theft, fraudulent software updates, support burden, and containment work on user devices.
Brand and search damage	The subdomain hosts spam, scams, adult content, counterfeit offers, or search-engine manipulation.	Loss of trust, search-quality issues, complaints, partner concern, and reputational repair work even when the primary website was never breached.
Privacy, contractual, or legal review	Personal, customer, employee, financial, or confidential information is submitted to or delivered through the taken-over endpoint.	Evidence preservation, counsel review, contractual notifications, insurance coordination, and regulatory analysis based on the facts and jurisdictions involved.
Operational interruption	Emergency DNS removal breaks legacy links or dependencies that nobody knew still used the hostname.	Support volume, failed integrations, unavailable customer workflows, rushed migration work, and management distraction.

Repercussion	How it can happen	Business effect
Incident-response cost	Teams must reconstruct DNS history, provider ownership, certificate activity, logs, user exposure, application references, and the duration of control.	Unplanned labor, outside assistance, delayed projects, customer communications, credential rotation, and restoration work.

Severity is determined by the full trust graph: who visits the hostname, what data they submit, which cookies are sent, which applications redirect to it, which APIs trust it, how long the condition existed, and what evidence is available.

What to do after suspected takeover

If an owned subdomain appears to be serving unauthorized content, treat it as an incident rather than an ordinary website edit.

1. **Preserve essential evidence.** Export the DNS zone, record the authoritative answer and TTL, preserve registrar and DNS change logs, capture the full HTTP response and headers, document the time discovered, and retain provider account and audit records. Do not delay containment to collect perfect evidence.
2. **Cut the public route.** Remove the vulnerable DNS record or point it to a controlled destination. Confirm the change at the authoritative nameservers and track the prior TTL so responders understand how long cached answers may remain.
3. **Engage the hosted provider.** Report the unauthorized custom-domain binding or resource claim through the provider's security or abuse process. Ask the provider to preserve tenant, access, ownership-verification, and certificate records.
4. **Remove residual trust.** Delete stale OAuth redirect URIs, SSO callbacks, webhooks, CORS and CSP allowlist entries, application links, QR codes, mobile deep links, and any other configuration that treats the retired hostname as trusted.
5. **Protect sessions and credentials.** Review cookie scope and authentication flows. Invalidate sessions, rotate credentials, revoke tokens, or reset affected accounts when evidence shows—or cannot reasonably exclude—that secrets or session material reached the endpoint.
6. **Identify affected users and data.** Determine which pages, forms, APIs, files, campaigns, email messages, or automated systems referenced the hostname during the exposure window.
7. **Search for related abandoned assets.** A takeover often reveals a process weakness. Review other domains, subdomains, cloud resources, old campaigns, development environments, and terminated vendors for the same lifecycle gap.
8. **Coordinate business response.** Involve leadership, legal, privacy, insurance, communications, customer support, affected vendors, and law enforcement as appropriate to the evidence and impact.
9. **Document the corrected lifecycle.** Assign owners, create a decommission gate, record the final state, and schedule verification so the same hostname or provider pattern does not recur.

> **Do not "test" by trying to out-claim an unknown account.** Racing to register a provider resource can alter evidence, violate provider terms, create ownership disputes, or unintentionally take control of somebody else's asset. Contain through DNS you own and coordinate with the provider.

A durable remediation roadmap

Phase 1: contain exposed names and map ownership

- Export all authoritative DNS zones and registrar-managed forwarding records.
- Trace CNAME chains and cloud-specific aliases to their final services.
- Match every external target to a provider account, resource, business purpose, and owner.
- Remove records for services that are no longer required.
- Escalate records whose provider resource is missing, unknown, suspended, or outside the expected account.
- Record criticality based on users, data, authentication, payments, integrations, and brand exposure.

Phase 2: make provisioning and retirement one controlled workflow

Turn a collection of technical edits into a repeatable service record:

YAML · fictitious third-party service retirement record

```yaml
service:
  name: "Fictitious Customer Help Center"
  custom_domain: "support.example.com"
  provider_target: "customer-7421.saas.example.net"
  business_owner: "Customer Operations"
  technical_owner: "IT Operations"
  planned_retirement: "2026-08-15"

required_sequence:
  - export provider and DNS configuration
  - reduce DNS TTL before the planned change
  - remove the public DNS record
  - verify authoritative removal and allow prior TTLs to expire
  - remove the provider custom-domain binding
  - delete the provider resource and close the account
  - remove stale OAuth, cookie, CORS, CSP, and application references
  - monitor the retired hostname and record final evidence
```

The workflow should require evidence at each gate. “Subscription canceled” is not completion. Completion means the public name no longer points to a reusable external resource, the provider mapping is removed, related trust settings are retired, application references are updated, and a named owner signs off.

For planned changes, lowering TTL in advance can shorten cache persistence, but only the TTL that resolvers previously received controls existing cached answers. Lowering it at the moment of deletion does not retroactively shorten old cached records.

Phase 3: reduce the value of any future takeover

- Use host-only cookies unless cross-subdomain sharing is a documented requirement.
- Review parent-domain cookies for sensitivity, lifetime, and necessity.
- Register exact OAuth and SSO redirect URIs and remove retired entries immediately.
- Avoid application rules that trust every subdomain solely because it ends in the company’s domain.

- Use explicit CORS origins and narrow CSP sources instead of broad wildcard subdomain trust where practical.
- Separate marketing, customer, development, and privileged application surfaces onto intentional hostnames and trust zones.
- Keep secrets, tokens, or private data from being sent automatically to endpoints simply because their hostname appears company-owned.

Phase 4: monitor the lifecycle, not just the DNS answer

- Diff authoritative DNS zones and alert on unapproved additions, target changes, and deletions.
- Compare DNS targets with cloud and SaaS resource inventories.
- Review provider custom-domain bindings and ownership-verification state.
- Require a scheduled revalidation date for every third-party hostname.
- Include DNS cleanup in employee, agency, vendor, campaign, and project offboarding.
- Monitor retired hostnames long enough to find legacy traffic and references.
- Review provider security features such as non-reusable identifiers, random TXT verification, resource-linked alias records, and deletion locks.
- Test the incident process using a harmless internal exercise, not a public resource claim.

> **The durable control is accountability.** Automation can identify differences, but only the organization can decide whether a hostname is still needed, who owns it, what it is allowed to trust, and what must happen when the service ends.

Turn the risk into a solvable business service

This weakness is well suited to a focused **Domain and SaaS Lifecycle Assurance Review**. The engagement connects public DNS, websites, hosted tools, cloud resources, vendor accounts, application trust settings, and business ownership into one practical remediation plan.

What Sunimod can deliver

- An authorized inventory of domains, subdomains, DNS records, redirects, and third-party targets.
- Mapping from each hostname to its business owner, technical owner, provider, account, tenant, resource identifier, and purpose.
- Identification of stale, unknown, failed, or potentially dangling DNS records without claiming third-party resources.
- Provider-account verification and custom-domain lifecycle review for websites, WordPress, ecommerce, support tools, status pages, landing-page systems, CDNs, cloud applications, and integrations.
- Risk ranking based on user exposure, authentication, cookies, OAuth and SSO callbacks, payment or support workflows, application trust, and data sensitivity.
- Coordinated DNS cleanup, redirect design, website and application updates, and provider offboarding.

- Review of cookie scope, redirect URIs, CORS, CSP, webhooks, deep links, and other trust relationships tied to retired hostnames.
- A repeatable provisioning and decommission checklist with evidence, owners, approvals, rollback considerations, and review dates.
- Ongoing monitoring and maintenance options so new projects do not recreate the same gap.

The customer is not buying a list of DNS errors. The customer is buying confidence that every public company name has a current purpose, a verified destination, an accountable owner, and a safe retirement path.

Good reasons to start now

- You have changed agencies, hosting companies, cloud providers, support platforms, ecommerce tools, or marketing systems.
- Your DNS zone contains records nobody recognizes.
- You inherited a website or domain portfolio without complete documentation.
- A campaign, microsite, status page, customer portal, or SaaS subscription was recently retired.
- You use parent-domain cookies, OAuth, SSO, webhooks, or broad subdomain allowlists.
- An old company link now shows a provider setup page, generic error, unexpected redirect, or unfamiliar content.
- You are preparing a migration, acquisition, divestiture, rebrand, vendor termination, or domain consolidation.

Frequently asked questions

Is every dangling CNAME exploitable?

No. The provider may reserve the identifier, require fresh domain verification, prohibit cross-account reuse, or retain an ownership binding. A dangling record is still a configuration and ownership problem, but exploitability must be validated provider by provider.

Does DNSSEC stop subdomain takeover?

No. DNSSEC can prove that the DNS answer is authentic. It cannot prove that the SaaS tenant, cloud resource, storage endpoint, or custom-domain binding behind that answer is still controlled by the intended organization.

Does HTTPS stop subdomain takeover?

Not by itself. HTTPS protects the connection to the endpoint that presents a valid certificate. Some hosted platforms automate certificates for successfully bound custom domains. The core defense is preventing an unauthorized account from obtaining the binding and removing stale DNS before the resource is released.

Should we claim the resource to prove the finding?

Not without a written authorization plan, provider approval where required, and clear ownership boundaries. Begin with authoritative DNS, provider-account evidence, provider documentation, and support confirmation. Remove or reroute the DNS record when the resource is not required.

Is removing the DNS record enough?

It is the primary containment step, but the response should also account for cached answers, remove the provider mapping, retire OAuth and SSO callbacks, review cookie scope and application trust, update links and integrations, preserve evidence, and determine whether unauthorized content was served.

Key takeaways

- A company-controlled hostname can remain trusted after the underlying third-party resource is gone.
- A CNAME authenticates neither current tenant ownership nor the continuing business relationship.
- Takeover normally requires a dangling record, a missing intended resource, and a provider path that permits a new claim.
- A 404 or automated fingerprint is a lead, not proof of exploitability.
- Provision provider resources before publishing DNS; remove DNS before releasing provider resources.
- Parent-domain cookies, stale OAuth redirects, and broad subdomain allowlists can increase impact.
- Authoritative inventory, named ownership, change control, and supplier offboarding are the durable controls.
- Contain suspected takeover through DNS you own and coordinate with the hosted provider.

Sources and further reading

1. Microsoft Learn: Prevent dangling DNS entries and avoid subdomain takeover
2. Microsoft Azure Architecture Center: Dangling DNS and subdomain takeover attacks
3. OWASP Web Security Testing Guide: Test for Subdomain Takeover
4. NIST Cybersecurity Framework 2.0
5. RFC 1034: Domain Names—Concepts and Facilities
6. RFC 9110: HTTP Semantics—Host and :authority
7. RFC 6265: HTTP State Management Mechanism
8. RFC 9700: Best Current Practice for OAuth 2.0 Security
9. RFC 2606: Reserved Top Level DNS Names

Provider behavior changes over time. Verify the current documentation and account-specific controls for every hosted service before changing production DNS or concluding that a resource is claimable.

Make every published subdomain accountable

Hire Sunimod to trace the hosted services behind your domains, identify stale or unowned routing, remove dangerous leftovers, repair website and application dependencies, and build an offboarding process that closes both the provider account and the public trust path.

Request a Sunimod project quote

Describe the domains, websites, hosted platforms, migration, or retired services you are concerned about. Do not send passwords, API keys, access tokens, recovery codes, or other secrets through the quote form; Sunimod can arrange a safer handoff when access is required.