# Summary of Findings

- Frequent use of 'terraform taint' and 'untaint' is symptomatic of deeper architectural issues in the solution.

- Mixing Terraform and direct Akamai API scripts leads to state drift, ownership ambiguity, and automation breakdowns.

- Manual tainting disrupts the declarative and idempotent nature of Terraform.

- This practice introduces significant operational risk, audit gaps, and inefficiencies in CI/CD pipelines.

- A clean separation of responsibilities, or migrating to a single toolset for resource management, is strongly recommended.

# Risks of Frequent Tainting in Terraform with Mixed Akamai API Scripts

Using 'terraform taint' and 'untaint' frequently as a workaround in an Akamai solution that mixes Terraform and direct API calls is indicative of architectural and operational issues. While these commands can resolve short-term inconsistencies, relying on them regularly is not recommended.

## Why Frequent Tainting is Problematic

Frequent use of 'terraform taint' and 'untaint' is symptomatic of deeper architectural issues in the solution.

Mixing Terraform and direct Akamai API scripts leads to state drift, ownership ambiguity, and automation breakdowns.

Manual tainting disrupts the declarative and idempotent nature of Terraform.

This practice introduces significant operational risk, audit gaps, and inefficiencies in CI/CD pipelines.

A clean separation of responsibilities, or migrating to a single toolset for resource management, is strongly recommended.

## Impacts in Akamai Solutions

|  |  |
| --- | --- |
| Impact | Description |
| Unstable CI/CD pipelines | Pipelines may fail unless tainting is manually performed before execution. |
| Broken team workflows | Inconsistent understanding of resource ownership leads to errors. |
| Drift-prone infrastructure | Terraform loses control over actual infrastructure state. |
| Security and compliance issues | Audit trails become incomplete due to manual interventions. |
| Operational overhead | Engineers spend time resolving taint-related issues instead of building value. |

## Recommended Alternatives

- Avoid mixing Terraform and API calls for managing the same resources.

- Choose a single source of truth for configuration and resource management.

- If necessary, use 'lifecycle ignore\_changes' to safely handle non-Terraform-managed fields.

- Use data sources in Terraform to reference externally managed resources instead of defining them.

- Clearly define team ownership and boundaries between Terraform and scripting responsibilities.