Bypassing CloudTrail in AWS Service Catalog, and Other Logging Research | Datadog Security Labs

By Nick Frichette

Published: 2023-03-20 · Archived: 2026-04-06 00:35:20 UTC

CloudTrail is a crucial AWS service that provides a record of API calls and other important activities in AWS environments. Teams can use this information for auditing purposes and to identify potential security incidents. If an attacker who has gained a foothold in an environment can perform actions without CloudTrail logging them, they'll be able to conceal their activities and become functionally invisible to the victim.

Bypassing CloudTrail for AWS services is an active field of research. In previous publications, we demonstrated how we were able to enumerate IAM permissions without logging to CloudTrail for a number of services and perform a subset of IAM actions while bypassing CloudTrail, allowing an adversary to carry out reconnaissance activities undetected.

In this blog post, we'll share some of our latest research into bypassing CloudTrail. We'll cover a method that allowed CloudTrail bypass with both read and write API actions for the Service Catalog service. This now-fixed vulnerability is noteworthy, because it was the first publicly known CloudTrail bypass that could permit an attacker to alter an AWS environment. In addition, we'll discuss another bug we identified in CloudTrail logging for AWS Control Tower.

Like with all CloudTrail bypasses, this would be used as a post-exploitation technique. An adversary would need to gain initial access to an AWS account through some means and the compromised identity would need to have sufficient privileges to interact with Service Catalog. It is at this point that an adversary could have used this vulnerability to bypass CloudTrail logging, for both read and write API actions. CloudTrail bypasses are important because they allow an adversary to avoid detection during activities that might otherwise be suspicious.

We shared this information with AWS, who confirmed our findings and have since remediated both issues.

Disclosure timeline

January 30, 2023: Datadog reports both issues to AWS.
January 30, 2023: AWS responds that they received the report.
January 31, 2023: Datadog sends two proof-of-concept scripts.
February 7, 2023: AWS confirms that fixes are in development for both issues.
February 7, 2023: AWS deploys fix to Service Catalog.
February 13, 2023: AWS deploys fix to Control Tower.
March 20, 2023: Datadog releases public disclosure.

CloudTrail bypass in AWS Service Catalog

As cloud security researchers, we are constantly exploring attack paths in cloud environments, trying to understand how real-world adversaries could compromise or exploit cloud resources. This research is typically focused on the customer side of the shared responsibility model—however, we sometimes identify opportunities to cross into the cloud service provider's domain.

While working on a research project, we noticed an interesting entry in the Content-Security-Policy meta tag for a page in the AWS Console.

```
 [···snip···]
https://controltower.us-east-1.amazonaws.com
https://sts.amazonaws.com/
https://cloudformation.us-east-1.amazonaws.com/
https://aws242-servicecatalog-beta.us-east-1.amazonaws.com
https://widgets.marketplace.us-west-2.amazonaws.com
https://clientlogger.marketplace.aws.a2z.com
[···snip···]
```

What caught our eye here was the reference to `aws242-servicecatalog-beta.us-east-1.amazonaws.com`. It looks similar to the normal API endpoint for AWS Service Catalog, `servicecatalog.us-east-1.amazonaws.com`, but it seemed strange that it had the `aws242` prefix. Because Service Catalog is such a critical AWS service—it allows organizations to create and maintain catalogs of approved AWS resources—a suspicious endpoint for this service could introduce additional attack surface, so we decided to dig deeper.

When using the AWS Console, a number of JavaScript files are loaded in the user's browser. These files contain the logic for interacting with the various AWS APIs. After digging around, we eventually found the service definition information for `AWS242ServiceCatalogService`.

```
{
 "version": "2.0",
 "metadata": {
   "apiVersion": "2015-12-10",
   "endpointPrefix": "servicecatalog",
   "jsonVersion": "1.1",
   "protocol": "json",
   "serviceFullName": "AWS Service Catalog",
   "serviceId": "Service Catalog",
   "signatureVersion": "v4",
   "targetPrefix": "AWS242ServiceCatalogService",
   "uid": "servicecatalog-2015-12-10"
 },
[···snip···]
```

Armed with the service definition information and the suspicious endpoint, we knew how to speak to the service and what endpoint to interact with. We used valid IAM credentials with the necessary privileges to sign our requests using the SIGv4 signing protocol, and the results we got surprised us.

Interacting with the beta endpoint

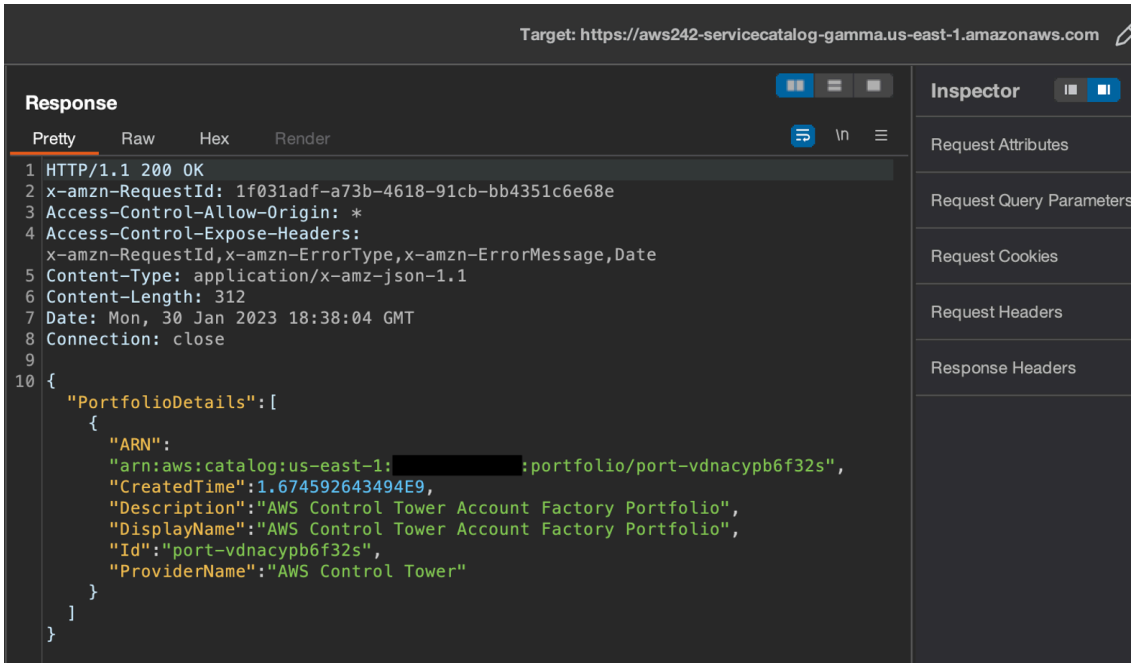
When we interacted with the `beta` endpoint, none of our requests failed or received an error message—however, they did not return results that we would normally expect. For example, our attempts to call `servicecatalog:ListPortfolios` returned no results, even when we intentionally created a portfolio in the account.

We found, however, that if we used the `beta` endpoint to create a portfolio, it would show up when we called `servicecatalog:ListPortfolios` with the `beta` endpoint. We hypothesize that the `beta` endpoint is in a different namespace than the normal commercial environment, so resources created in this partition are not accessible by normal means. In terms of security impact, we could not identify any way this endpoint would be useful to an attacker, since it seemed to be siloed from the normal production environment.

Having reached what appeared to be a dead end and looking for other opportunities, we checked if there was a DNS entry for a `gamma` endpoint. Through their research on the AWS console, our researchers knew that `gamma` is a relatively common suffix for various AWS API endpoints (along with `alpha` and `beta`) and has been noticed by the security research community on multiple occasions. Based on their naming conventions, we presume these endpoints are used to test new features and services. Sure enough, the `aws242-servicecatalog-gamma.us-east-1.amazonaws.com` domain resolved to an IP address.

Interacting with the `gamma` endpoint

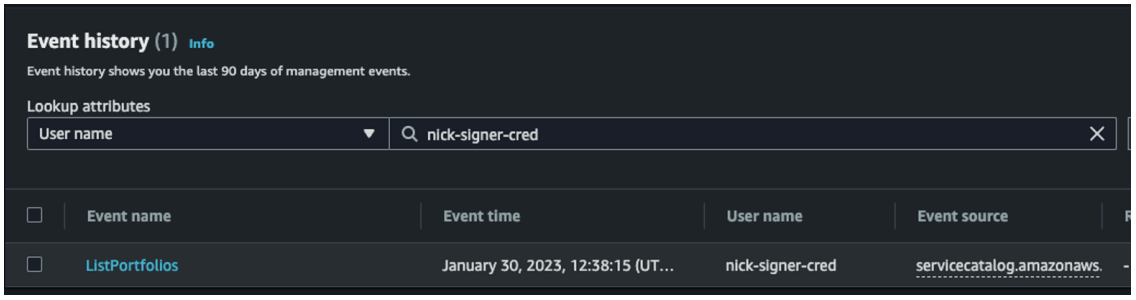
The first thing we noticed was that we got normal results when interacting with `aws242-servicecatalog-gamma.us-east-1.amazonaws.com`. We could, for example, perform `servicecatalog:ListPortfolios`, and the result would show all portfolios we could normally see in the account from the console.



We could now interact with Service Catalog normally using the gamma endpoint and get results for our API calls. The new question became, what does this activity look like in CloudTrail? From prior research on the subject, we

knew that when the AWS Console interacts with private AWS APIs, there is a potential to bypass CloudTrail.

Here is what it looks like in CloudTrail when you invoke the `servicecatalog:ListPortfolios` call using the AWS CLI and the normal endpoint.



However, by comparison, after waiting we saw that no traffic sent to the `gamma` endpoint would show up in CloudTrail. We had just found a new way to bypass CloudTrail.

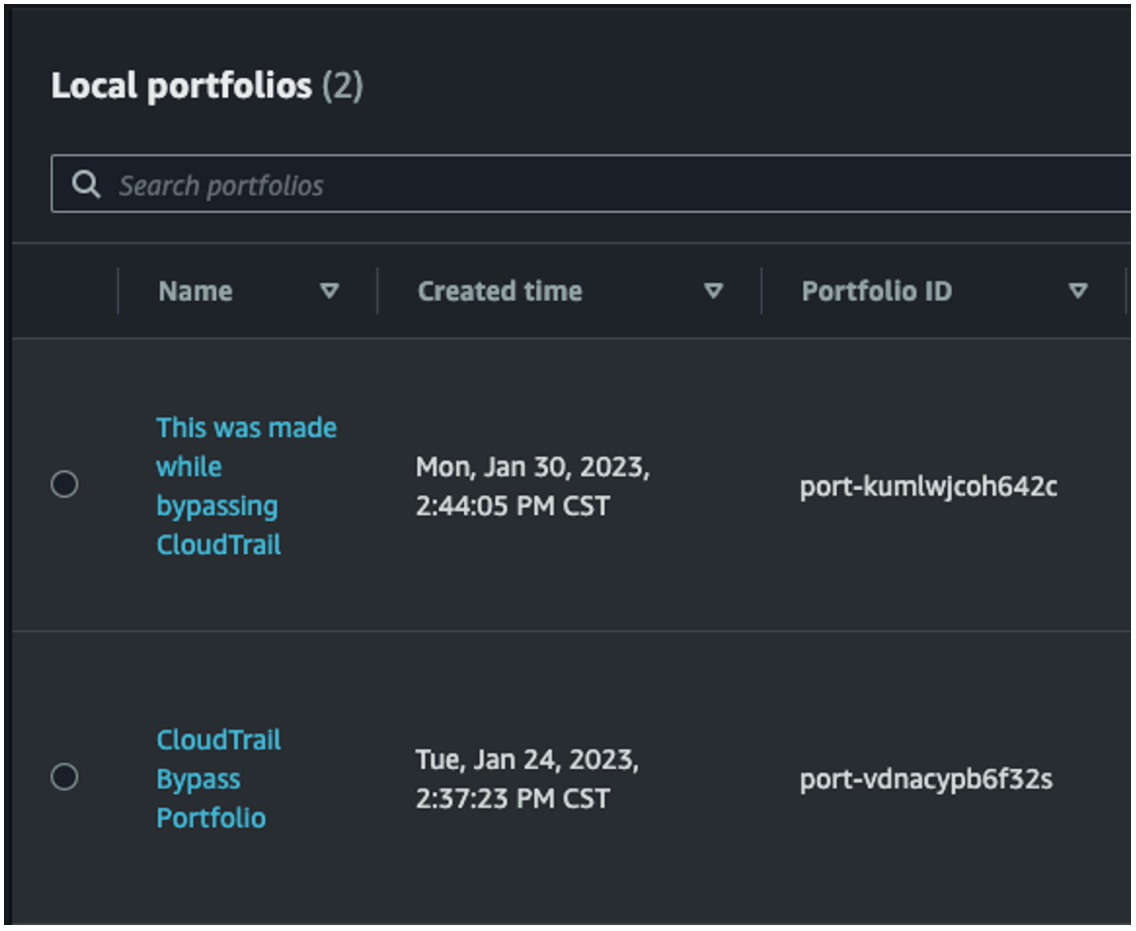
Write actions

The next order of business was to try to perform write actions, which are API calls that create or alter resources. During this step, we encountered an interesting error message.

```
 HTTP/1.1 400 Bad Request
[···snip···]
{
  "__type":"AccessDeniedException",
  "Message":"Access Denied - The caller doesn't have permission to call this API."
}
```

This was discouraging—however, to our surprise, the request that returned this error message appeared to complete successfully. For example, when calling `servicecatalog:CreatePortfolio`, we received the same error message, but the portfolio was nevertheless created.

```
 [...snip···]
{
  "ProviderName":"my-provider",
  "DisplayName":"This was made while bypassing CloudTrail",
  "IdempotencyToken":"blahblahblah"
}
```



Other write actions such as `servicecatalog:UpdatePortfolio` also returned a similar result: We would get an error as the response, but the action would complete normally.

For a practical example of how an attacker could abuse this flaw, imagine a company using AWS Service Catalog has a portfolio called “Production Portfolio” that they use to share CloudFormation stacks of production-ready infrastructure. An adversary who has compromised access to a role or user with sufficient permissions to modify the portfolio and the products within it could do so by using the `gamma` endpoint—without leaving a CloudTrail log of their actions.

Missing CloudTrail logging in Control Tower

AWS Control Tower is a service that makes it easy to manage common controls across a multi-account AWS organization by orchestrating the capabilities of AWS Service Catalog, IAM Identity Center, and other services. While using Control Tower from the AWS Console, the application will make requests to an additional service called `AWSBlackbeardService`. This service handles API requests for most of the console functionality of Control Tower.

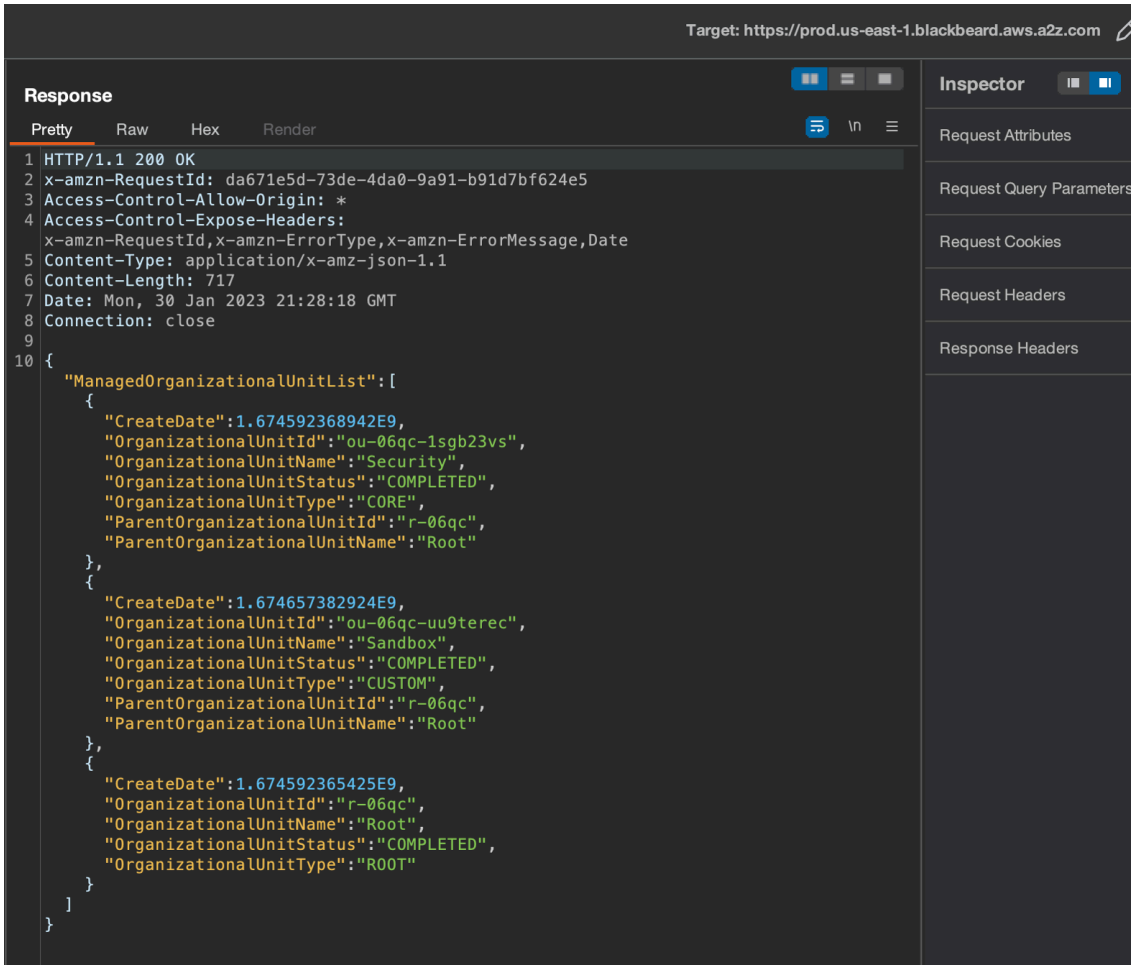
```
{
 "version": "2.0",
 "metadata": {
   "apiVersion": "2018-11-26",
   "endpointPrefix": "controltower",
```

```
    "jsonVersion": "1.1",
    "protocol": "json",
    "serviceFullName": "AWS Control Tower",
    "serviceId": "ControlTower",
    "signatureVersion": "v4",
    "signingName": "controltower",
    "targetPrefix": "AWSBlackbeardService",
    "uid": "controltower-2018-11-26"
  },
[···snip···]
```

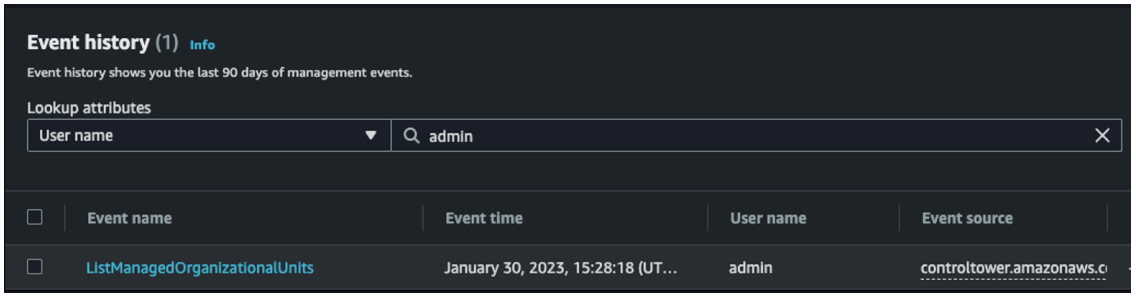

While working with `AWSBlackbeardService` during our research, it quickly became clear that it behaved differently from other AWS services. Normally, services that log events to CloudTrail log both successes **and** failures. Any action configured to log to CloudTrail will produce a log there, regardless of whether it succeeds or fails.

With `AWSBlackbeardService`, however, failures due to insufficient privileges were **not** logged to CloudTrail. We can demonstrate this by making calls to the service with two different user accounts: admin and no-perm (for no-permissions). With both of them, we invoke `controltower:ListManagedOrganizationalUnits`.

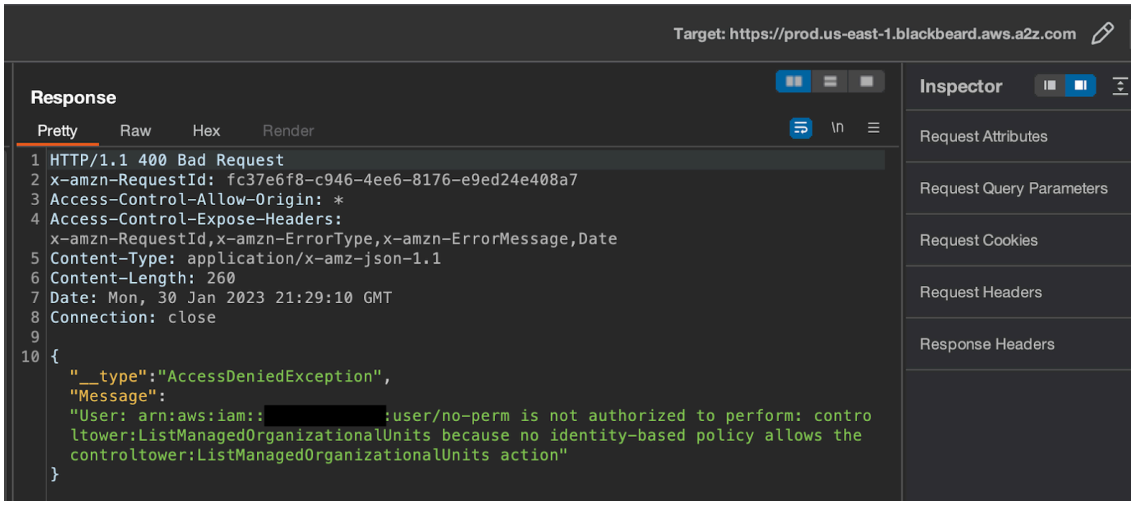
Here is the admin user's response:



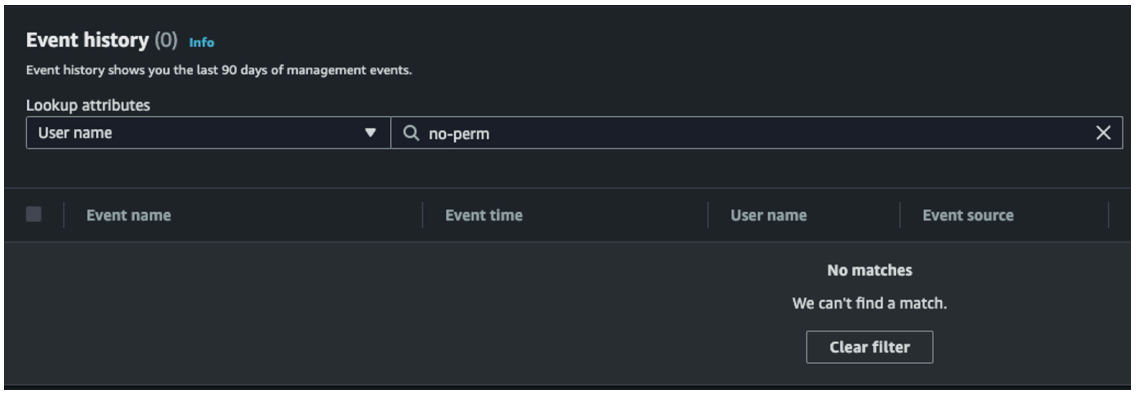
And here is the resulting CloudTrail event:



Here is the no-perm user's invocation:



And here, we see that the result does not appear in CloudTrail:



Knowing when an API action fails due to a lack of privilege can sometimes be as important as knowing when an action succeeds. Attackers who have compromised IAM credentials will often use brute-force techniques to determine what permissions the compromised entity has—and in this scenario, because there is no log in CloudTrail, they could avoid detection when enumerating Control Tower permissions.

You may be wondering, is it possible to use a `gamma` endpoint with `AWSBlackbeardService` to bypass CloudTrail logging, similar to what we did with Service Catalog? As it turns out, it's not, but the reason sheds further light on the differences between `alpha` / `beta` / `gamma` and the normal endpoints. `gamma.us-east-1.blackbeard.aws.a2z.com` resolves to an IP address (the normal endpoint is `prod.us-east-`

`1.blackbeard.aws.a2z.com`). When you made an API call to the `gamma` endpoint with `AWSBlackbeardService`, you would get the following error:

```
 {
  "__type":"AccessDeniedException",
  "Message":"Failed to assume role arn:aws:iam::123456789012:role/service-role/AWSControlTowerAdmin"
}
```

Normally, when making requests through the Blackbeard service, Control Tower will [assume](#) the `AWSControlTowerAdmin` role in the account. When you execute this request on the normal production endpoint, the service assuming the role is `controltower.amazonaws.com`. However, when you make this request via `gamma`, the `AWSInternal` identity assumes the role. This means the `gamma` endpoint does not assume the role with the same identity that the production service does.

This can be validated by setting the trust policy on the `AWSControlTowerAdmin` role to allow `*` to assume the role. (**Never do this in your AWS account**, as it will allow any AWS principal to assume the role.)

```
 {
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2023-02-13T21:52:45Z",
    "eventSource": "sts.amazonaws.com",
    "eventName": "AssumeRole",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "roleArn": "arn:aws:iam::123456789012:role/service-role/AWSControlTowerAdmin",
        "roleSessionName": "GetGuardrailComplianceStatus"
    },
[···snip···]
```

In the excerpt of the above CloudTrail log above, we can see that “AWS Internal” assumes the role, rather than the Control Tower service. When this happens, we get a different error message:

```
 {
  "__type":"ResourceNotFoundException",
  "Message":"Unable to get details for account=123456789012 in namespace=123456789012"
}
```

This further validates our theory that `gamma` and `beta` endpoints can be (though are not always) in a separate partition or namespace, where their resources are different from the normal production environment used by customers.

Fixes

In response to these findings, AWS has implemented a fix for each service. For Service Catalog, attempts to make requests to the `gamma` endpoint (even with sufficient privileges) now return the following error:

```
 HTTP/1.1 400 Bad Request
[...snip…]
{
  "__type":"AccessDeniedException",
  "Message":"Access Denied - The caller doesn't have permission to call this API."
}
```

For Control Tower—and, by extension, the `AWSBlackbeardService` —failed requests will now appropriately log to CloudTrail.

AWS's Response

In addition, AWS has provided the following statement in regard to this disclosure:

"AWS is aware of the issues reported by Datadog and has fully addressed them. No customer action is required. We have conducted a review of the services' logs with respect to the reported issues, and did not discover any evidence of unauthorized activity related to either Service Catalog or Control Tower. We would like to thank Datadog for the coordinated disclosure of their security research and collaboration with us on this report."

Acknowledgements

We’d like to give major thanks to Alexis Fahrney and the AWS Security Outreach Team for being pleasant to work with, as always.
