Cyral

WHITE PAPER

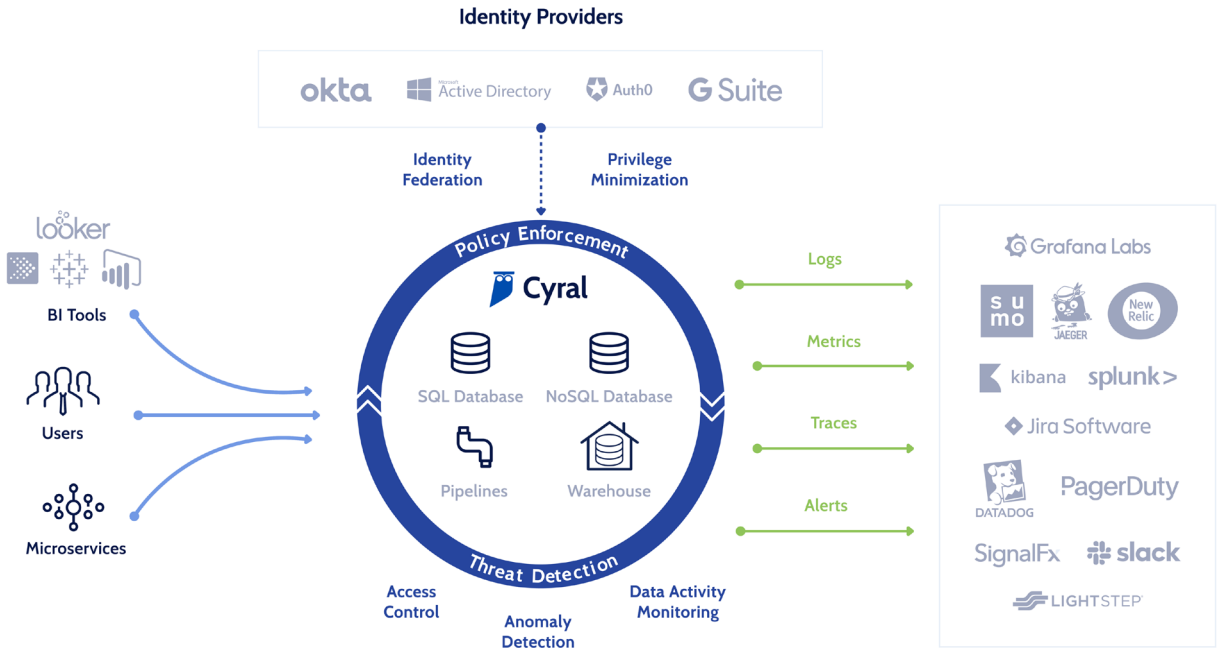
Using Observability to Detect Data Exfiltration

Contents

Using Observability to Protect Against Data Exfiltration

Most breaches are targeted at data repositories where the crown jewels of an organization reside. To protect themselves against a breach, the best that security teams can do today is deploy solutions across all their apps, infrastructure, networks and devices, but that does not give them the required visibility and control over their data flows. This is exacerbated by modern data mesh architectures deploying database-as-a-service instances, pipelines and cloud data warehouses that do not support traditional monitoring agents and host-based policy enforcements. Zero-trust systems also can't rely on monitoring agents to act as security boundaries for the precious crown jewels.

Cyral is the first cloud-native Security as Code solution to protect the modern data mesh. Our platform makes it easy for engineering teams to observe, protect, and control data endpoints in a cloud and DevOps-first world.



In this white paper, we describe how observability across the data mesh can be used by DevOps teams to protect themselves from data exfiltration attempts.

What is Observability?

The goal of observability is to provide granular insights into the behavior of a system along with a rich end-to-end context of all of its interactions with external users and services. In order to be observable, a system must generate the following artifacts with great detail.

- Logs that describe discrete events such as error/debug messages, audit trail events, and state changes in semi-structured formats suitable for processing and analytics after the fact.
- Metrics that report metadata about a system that can be aggregated in various ways — the rate of request processing, a histogram of response times, and the average request queue length. Metrics are often used to reason about the health of a system.
- Traces that stitch together context from various components of a distributed system to help understand the lifecycle of a request — where it originated, where it traveled, and where it was finally fulfilled.

Together, the above three artifacts provide visibility into the internal states of the system and help answer questions around how and why service failures, performance degradations, and security breaches occur for SRE, DevOps and SecOps teams.

What is the observability gap in the data mesh?

Observability is a well understood problem for self-hosted services (homegrown web applications, ETL jobs, microservices) and self-managed infrastructure (cloud VM instances such as EC2). Services may be instrumented through a combination of manual and/or dynamic instrumentation mechanisms to generate logs, metrics and traces of all activity. Similarly, agents may be deployed on cloud VM instances for infrastructure monitoring and alerting. The artifacts generated by instrumentation and monitoring are shipped to a dashboard for alerting, analytics and troubleshooting. Datadog, Grafana and New Relic are examples of products that do this for services and infrastructure across programming languages and cloud environments.

These solutions, however, break down at the data mesh. DBaaS endpoints such as RDS, Aurora, Redshift, Snowflake, MongoDB Atlas, Bigquery, and Athena can neither be instrumented, nor monitored using agents as both the software and the hardware comprising these endpoints are in the control of their respective cloud and service providers. Consequently, the artifacts needed for these endpoints to be observable either do not have the right granularity, or are entirely missing in certain cases, as described below:

Often the only source of logs in both traditional on-prem DB deployments and DBaaS is when the data repository itself logs the activity to a file system. However, logging is commonly turned off for the following reasons:

- **Degradation in performance** The QPS (queries per second) typically drops by 25-30% in MySQL and PostgreSQL databases when query logging is turned on due to the additional I/O incurred in the critical query execution path.

- **Risk of PII leaks** The queries/requests being logged do not have PII redacted from them. This ends up being a security concern.

There are two sources of metrics in DBaaS:

- **Non-granular DBaaS metrics** These are aggregate metrics published by the DBaaS engine itself. They are coarse grained and cannot be broken down by specific human users or services interacting with the DBaaS. Examples include (among many others):
  - Connections per second
  - SELECTs/UPDATEs/INSERTs per second
  - Slow queries per second
- **Rolled-up Cloud provider metrics** AWS Cloudwatch, for example, publishes the number of bytes and throughput corresponding to the DBaaS engine's network I/O activity. However, it's not possible to determine how many rows from a certain table, or documents from a collection, correspond to the number of bytes observed at the network layer. It's also not possible to attribute the observed values of these metrics to specific human users or services interacting with the DBaaS.

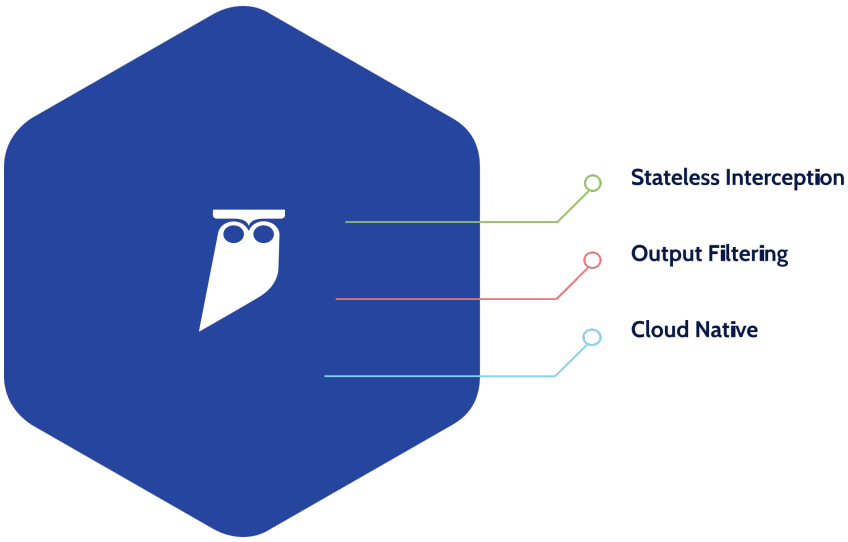
Traditional database deployments add some visibility. Often these logs are disabled because of the performance impact or storage cost.

- **Service accounts:** It is common for a user to login to the database using a BI tool or application, and the application will query the database using a shared service account. The original user's identity is not available to database logs.
- **User activities:** Authentication logs can highlight authentication failures including the date & time of the event. These logs don't include the contextual data including the query performed or the source IP of the caller.
- **Query activities:** These logs are generally used for database tuning and include performance details including the query plan and execution time.
- **System Health:** Aggregate metrics about the database health including memory and storage used, the last time performance tuning was run, and hardware or corruption issues if any.

Above are just a few examples of observability artifacts not being of the right granularity, or being completely absent in DBaaS endpoints. This leads to blind spots and a lack of visibility when it comes to troubleshooting failures, performance degradations and security issues.

How Does Cyral Plug this Observability Gap?

The observability gap exists at the data mesh because in zero-trust environments, monitoring endpoints can't be used as boundary fences. Further, cloud-based DBaaS endpoints can neither be instrumented nor monitored using agents. Cyral solves this problem by intercepting all data endpoint requests coming from human users, applications, and 3rd party tools. The key to this is a featherweight, stateless interception service that can be easily deployed in the customer's environment. We call this the Data Mesh Sidecar.



The Cyral sidecar intercepts requests in real time to generate logs, metrics, traces of all data activity. Each of these artifacts are generated at the granularity of each human user or service interacting with the DBaaS endpoint, thus providing rich semantic information for policy evaluation, monitoring and analytics.

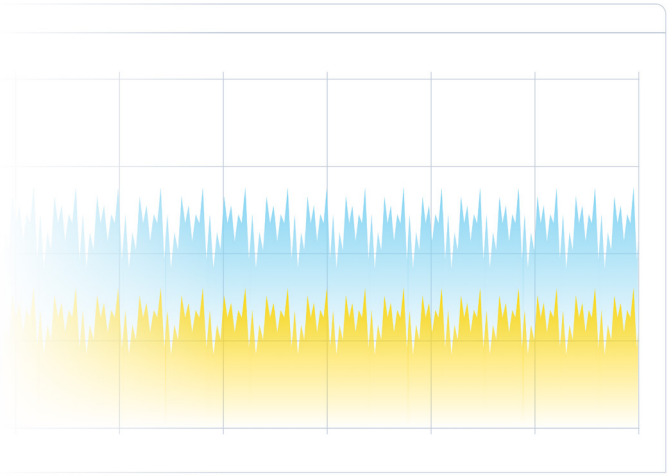
Logs

Cyral logs capture the intent and cardinality for each request and response pair, respectively, annotated with the identity of the human users, applications and 3rd party tools issuing them. Additionally, PII embedded in the requests is redacted in order to make it safe to send the logs to a SIEM for analytics.

```
"endUser": "bsmith@barnfeed.com"
"ssoGroup": "Analysts",
"dbUser": "analysts",
"request": {
    "statement": "select ccn from orders where name = :redacted1",
    "statementType": "Select",
    "tablesReferenced": [ "barnfeed.orders" ],
    "columnsReferenced": {
      "barnfeed.orders": [ "ccn" ]
    },
}
"response": {
    "status": "OK",
    "rowsAffected": 10,
    "bytesAffected": 1560,
    "executionTime": "1.54225ms"
}
```

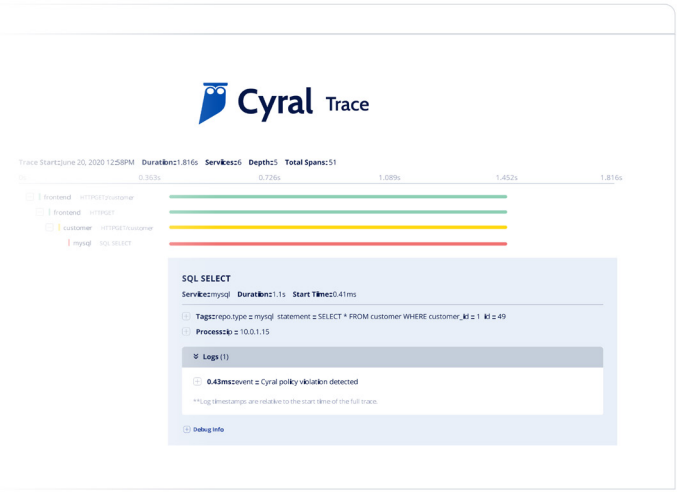
Metrics

Unlike rolled up DBaaS and infrastructure metrics, Cyral reports detailed performance metrics (latency, QPS) as well as security metrics (rate of sensitive requests, rows/documents of sensitive data read) tagged with the identity of the human users or application.



Traces

Cyral adds OpenTelemetry spans describing each request-response issued at the data mesh to ongoing traces in a distributed system of microservices. This allows developers, DevOps and SecOps teams access to complete end to end traces to easily diagnose failures, performance degradation and security issues.



For complete details of metrics, log structure and tracing information generated by Cyral, please refer to: **www.cyral.com/docs**

Use Case: Data Exfiltration Detection

Data exfiltration is a security threat that involves an external attacker or a malicious insider stealing an organization's monetizable data for their personal gain. In the context of DBaaS endpoints, a data exfiltration attack may be carried in several different ways, which may be broadly classified into the following two categories.

Bulk Exfiltration

This attack exploits APIs that support bulk reading of data. For endpoints supporting SQL (RDBMS, NoSQL, Kafka KSQL) this usually means running a SELECT statement without any constraints in the form of a WHERE clause or a JOIN condition. For endpoints driven by SDKs (S3, memcache, redis) this ends up being some variant of a bulkGET() or a bulkFetch() method. In all the cases, however, the consequence is that a large number of rows or documents of data are read from the endpoint.

Detection Without Using Cyral

Without Cyral, one needs to resort to cobbling up logs and metrics from multiple sources, and adding tooling and automation for alerting upon detection of an exfiltration attempt.

On AWS, the Cloudwatch metrics Network IO Bytes and Network IO Throughput MB/s quantify the outgoing network activity from the data endpoint's perspective. Alerting built on top of these metrics would identify when a bulk exfiltration attempt is in progress based on predetermined thresholds.

Subsequently, scanning database logs within the timeframe when the Cloudwatch metrics were alerted upon will help identify the specific dataset (table, collection, topic) that was the target of exfiltration. However, CloudWatch logs are delivered on a best-effort basis and often not for many hours making the logs only useful for a post-mortem and not for real-time exfiltration prevention.

However, Cloudwatch network activity metrics are low level and do not capture any user/service and dataset level information. This results in a lot of false positives, which makes these metrics a low fidelity signal for exfiltration detection. Similarly, DBaaS logging is often turned off for performance and security reasons. Consequently, this signal may not even be present.

Detection Using Cyral

Bulk exfiltration detection using Cyral is very straightforward. Cyral sidecar intercepts data endpoint requests and responses to generate rich activity logs. The logs feed into a policy engine that determines whether thresholds for a specific dataset will be violated as a result of a request.

For example user "bob" issues a request to access customer SSNs and email addresses but due to lack of an explicit constraint in the form of a WHERE clause, almost 1500 rows of customer data end up being read.
The "safe" threshold previously determined for "bob" was 10 rows, hence an alert is immediately generated.
Optionally, the connection used by "bob" may be terminated resulting in blocking behavior.

Due to the rich and granular insights into request intent and response cardinality, this approach leads to a very high fidelity signal (hence, no false positives).

Trickle Exfiltration

Trickle exfiltration involves more sophisticated mechanisms to exfiltrate data from an endpoint. The objective of an attacker is to stay “under the radar” of network and security monitoring tools by adopting a low and slow approach to exfiltration.

Regardless of the type of the data endpoint, all trickle exfiltration attacks rely on what are called point requests (fetching a small set of specific rows), range requests (fetching a small range of rows) and offset requests (fetching a small set of rows from a different offset each time).

Detection Without Using Cyral

AWS Cloudwatch network metrics will not show any anomalies because the attacker is intentionally staying under the radar. With database logging also turned off, trickle exfiltration detection becomes virtually impossible using existing tools.

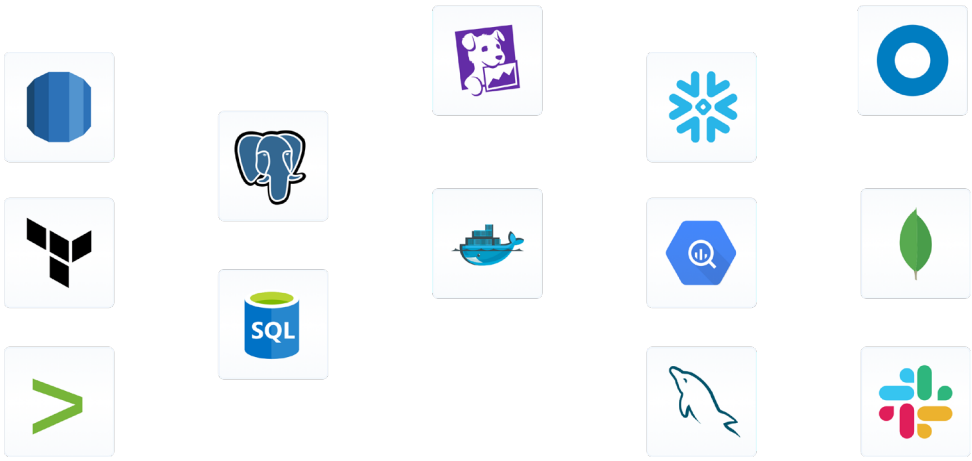
Detection Using Cyral

A combination of logs, metrics and traces generated by Cyral make it easy to detect trickle exfiltration attempts. For each user and service, Cyral tracks the number of rows or documents of sensitive data read and the rate at which such read requests are issued. Tracking these metrics over time establishes a baseline for each user and service, and serves as a threshold against which presently issued requests by them can be checked. When these threshold checks continue failing over a sustained period of time (usually in the order of minutes), it becomes a very strong and reliable signal of a trickle exfiltration attempt.

While metrics help with detection, the generated logs show which particular dataset (a table or a document collection) was the target of the attack by tracing the attacker’s data activity across time and across all of their sessions.

Cyral was designed to be plugged seamlessly into existing DevOps and SecOps workflows. Deploy using Terraform, Cloudformation, Chef, Ansible, Puppet. Logs, metrics, traces and alerts can be sent to popular DevOps tools like Splunk, DataDog, ELK, Grafana, PagerDuty and the service can integrate popular security tools used for identity management (Okta, Gsuite, Microsoft AD), incident response (Jira, Hive), forensics and compliance management.

Cyral Integrations



About Cyral

Cyral delivers enterprise data security and governance across all data services such as S3, Snowflake, Kafka, MongoDB, Oracle and more. The cloud-native service is built on a stateless interception technology that monitors all data endpoint activity in real-time and enables unified visibility, identity federation and granular access controls. Cyral automates workflows and enables collaboration between DevOps and Security teams to automate assurance and prevent data leakage. Cyral is venture-backed by Redpoint, A.Capital, Costanoa and SVCI. Follow the company on Twitter at @CyralInc.

Want to learn more about how you can use observability to stop data exfiltration? Sign up for a tech talk with one of our engineers to see how Cyral's unique architecture can help!

cyral.com/tech-talk