THOMAS MORE

Lab 4.5

Hunting Through Application and Infrastructure Events

Toepasbare Cloud Security voor IT-Professionals

2025 - 2026

Contents

1. Hunting Through Application and Infrastructure Events

1.1. Context

In this lab you use KQL to investigate application and infrastructure events in Azure.

The focus is not on writing many separate queries, but on answering a few practical security questions:

- **What telemetry is available?**
- **What traffic is reaching the application?**
- **Is the internal backend reachable from outside?**
- **Can we trace one request across services?**
- **Were secrets, roles, or resources changed?**

1.2. Learning goals

By the end of this lab, you can:

[**Discover** available tables in Log Analytics] [**Investigate** application requests, traces, and dependencies] [**Detect** external access to internal endpoints] [**Trace** a request using `OperationId` or a correlation ID] [**Review** Key Vault and Azure Activity events for suspicious actions]

1.3. Estimated time

45-60 minutes

Time	Activity
5 minutes	Query 1: Discover available telemetry
10 minutes	Query 2: Investigate application traffic
15 minutes	Query 3: Detect external access to internal backend
10 minutes	Query 4: Trace one request across services
10 minutes	Query 5: Review sensitive and management-plane events

1.4. Before you start

Make sure you have:

[Access to the Log Analytics workspace] [Application Insights data from the frontend/API/internal backend exercise] [App Service HTTP logs enabled] [Key Vault diagnostics enabled, if you want to inspect secret access] [Azure Activity logs connected, if you want to inspect role or resource changes]

2. Query 1: Discover Available Telemetry

2.1. Purpose

Before investigating, first check which tables actually contain data.

2.2. Query

```
Usage
| where TimeGenerated > ago(24h)
| where IsBillable == true
| summarize
    DataGB = round(sum(Quantity) / 1024, 3)
    by DataType
| order by DataGB desc
```

2.3. What to check

Expected useful tables:

Table	Why it matters
`AppRequests`	Application HTTP requests captured by Application Insights
`AppDependencies`	Outgoing calls to services such as Key Vault or Storage
`AppTraces`	Application logs and structured traces
`AppServiceHTTPLogs`	Platform-level HTTP logs from App Service
`AZKVAuditLogs`	Key Vault secret access events
`AzureActivity`	Azure resource changes and role assignment changes

> **Question**
> Which expected tables have data? Which are missing? What might explain missing telemetry?

3. Query 2: Investigate Application Traffic

3.1. Purpose

Get a quick overview of application traffic, errors, and frequently used endpoints.

3.2. Query

```
AppRequests
| where TimeGenerated > ago(1h)
| summarize
    Requests = count(),
    Failed = countif(Success == false),
    AvgDurationMs = round(avg(DurationMs), 2)
    by Name, ResultCode
| order by Failed desc, Requests desc
```

3.3. Find recent failures

```
AppRequests
| where TimeGenerated > ago(1h)
| where Success == false or toint(ResultCode) >= 400
| project TimeGenerated, Name, ResultCode, ClientIP, DurationMs, OperationId
| order by TimeGenerated desc
| take 50
```

> **Question**
>
> Which endpoints fail most often? Are the failures expected, caused by your own tests, or suspicious?

4. Query 3: Detect External Access to the Internal Backend

4.1. Purpose

This is the main security investigation of the lab.

The internal backend should not be directly reachable from the internet. This query checks whether external clients reached `/internal` endpoints.

This could give a false positive you know why?

4.2. Query

```
AppServiceHTTPLogs
| where TimeGenerated > ago(24h)
| where _ResourceId has "Microsoft.Web/sites"
| extend
    Path = tostring(column_ifexists("CsUriStem", "")),
    ClientIP = tostring(column_ifexists("CIp", "")),
    Method = tostring(column_ifexists("CsMethod", "")),
    StatusCode = toint(column_ifexists("ScStatus", int(null)))
| where Path startswith "/internal"
| where ipv4_is_private(ClientIP) == false
| extend
    AccessResult = case(
        StatusCode between (200 .. 299), "Access succeeded",
        StatusCode in (401, 403), "Blocked by auth",
        StatusCode == 404, "Not found",
        StatusCode >= 500, "Server error",
        "Other"
    )
| project
    TimeGenerated,
    Alert = "External client requested internal endpoint",
    AccessResult,
    ClientIP,
    Method,
    Path,
    StatusCode
| order by TimeGenerated desc
```

4.3. Summarise by client IP

```
AppServiceHTTPLogs
| where TimeGenerated > ago(24h)
| extend
    Path = tostring(column_ifexists("CsUriStem", "")),
    ClientIP = tostring(column_ifexists("CIp", "")),
    StatusCode = toint(column_ifexists("ScStatus", int(null)))
| where Path startswith "/internal"
| where ipv4_is_private(ClientIP) == false
| summarize
```

```
    Attempts = count(),
    SuccessfulAttempts = countif(StatusCode between (200 .. 299)),
    FirstSeen = min(TimeGenerated),
    LastSeen = max(TimeGenerated),
    Paths = make_set(Path, 10),
    StatusCodes = make_set(StatusCode, 10)
    by ClientIP
| extend Finding = iff(
    SuccessfulAttempts > 0,
    "External access to internal endpoint succeeded",
    "External access attempted but did not clearly succeed"
)
| order by SuccessfulAttempts desc, Attempts desc
```

How odd ... could we only solve this somehow? :)

> **Question**
>
> Did an external IP reach the internal backend? Did the request succeed, or was it blocked? What does this tell you about the network controls from Day 2?

4.4. What to notice

Observation	Meaning
2xx status code	The internal endpoint was reachable and returned data
401 or 403	The service was reachable, but access was blocked by authentication or authorisation
404	The request reached the service, but the path was not found
No results	No external access was observed in the selected time window

5. Query 4: Trace One Request Across Services

5.1. Purpose

Use one request identifier to reconstruct what happened across requests, traces, and dependencies.

You can use either:

[`OperationId`: generated by Application Insights/OpenTelemetry] [`x-correlation-id`: set by your application code, if propagated correctly]

5.2. Step 1: Find a useful OperationId

```
AppRequests
| where TimeGenerated > ago(1h)
| project TimeGenerated, Name, ResultCode, Success, DurationMs, ClientIP, OperationId
| order by TimeGenerated desc
| take 20
```

Copy one `OperationId`.

5.3. Step 2: Trace request, dependencies, and logs

```
let opId = "<paste-operation-id-here>";

let request_events =
    AppRequests
    | where TimeGenerated > ago(1h)
    | where OperationId == opId
    | project
        TimeGenerated,
        EventType = "HTTP Request",
        Service = tostring(AppRoleName),
        Detail = tostring(Name),
        ResultCode = toint(ResultCode),
        Success = tobool(Success),
        DurationMs = todouble(DurationMs);

let dependency_events =
    AppDependencies
    | where TimeGenerated > ago(1h)
    | where OperationId == opId
    | project
        TimeGenerated,
        EventType = "Dependency",
        Service = tostring(AppRoleName),
        Detail = strcat(tostring(Name), " -> ", tostring(Target)),
        ResultCode = toint(ResultCode),
        Success = tobool(Success),
        DurationMs = todouble(DurationMs);

let trace_events =
    AppTraces
    | where TimeGenerated > ago(1h)
```

```
    | where OperationId == opId
    | project
        TimeGenerated,
        EventType = "Trace",
        Service = tostring(AppRoleName),
        Detail = tostring(Message),
        ResultCode = int(null),
        Success = true,
        DurationMs = real(null);

union request_events, dependency_events, trace_events
| order by TimeGenerated asc
```

> **Question**
>
> Which event types appear for this request? Do you see dependency calls to Azure services? Are any expected services missing from the trace?

5.4. Optional: trace by correlation ID

Use this only if your application writes an `x-correlation-id` into logs.

```
let correlationId = "<paste-correlation-id-here>";

union AppRequests, AppTraces, AppDependencies
| where TimeGenerated > ago(1h)
| where tostring(Properties["x-correlation-id"]) == correlationId
   or OperationId == correlationId
| project TimeGenerated, Type, AppRoleName, OperationId, Properties
| order by TimeGenerated asc
```

6. Query 5: Review Sensitive and Management-Plane Events

6.1. Purpose

Check whether sensitive operations occurred, such as secret access, role assignment changes, or resource updates.

6.2. Key Vault secret access

```
AZKVAuditLogs
| where TimeGenerated > ago(24h)
| where OperationName has "Secret"
| project
    TimeGenerated,
    OperationName,
    CallerIpAddress,
    Identity,
    ResultDescription,
    _ResourceId
| order by TimeGenerated desc
| take 50
```

6.3. Role assignment changes

```
AzureActivity
| where TimeGenerated > ago(24h)
| where OperationNameValue in (
    "Microsoft.Authorization/roleAssignments/write",
    "Microsoft.Authorization/roleAssignments/delete"
)
| project
    TimeGenerated,
    Operation = OperationNameValue,
    Status = ActivityStatusValue,
    ChangedBy = Caller,
    Scope = ResourceId,
    ResourceGroup,
    SubscriptionId,
    CorrelationId
| order by TimeGenerated desc
```

6.4. Resource changes

```
AzureActivity
| where TimeGenerated > ago(24h)
| where OperationNameValue has_any ("write", "delete", "action")
| project
    TimeGenerated,
    OperationNameValue,
    ActivityStatusValue,
    Caller,
```

```
    ResourceGroup,
    ResourceProviderValue,
    ResourceId
| order by TimeGenerated desc
| take 50
```

> **Question**
>
> Which sensitive operations occurred? Who triggered them? Are the changes expected based on your own lab actions?

7. Reflection

Answer the following questions briefly.

1. Which telemetry source was most useful for detecting direct external access to the internal backend?
2. What is the difference between `OperationId` and `x-correlation-id`?
3. Why can IP-based correlation be useful but imperfect?
4. If you found successful external access to `/internal`, what would you check next?
5. Which query from this lab would be most useful to convert into an analytics rule? Why?

8. Summary

In this lab you used KQL to investigate application and infrastructure events in Azure.

You learned how to:

- discover available telemetry,
- investigate application traffic,
- detect external access to internal endpoints,
- trace one request across services,
- and review sensitive Azure operations.

The most important takeaway is that security investigation is not about one perfect log table. It is about combining application telemetry, platform logs, and management-plane events to understand what really happened.