

generate

PY

Identify Suspicious Activity by Repeated Deletion

Actions select actor_handle, count(*) as delete_count from pipes_audit_log where action_type = 'workspace.delete' group by actor_handle;

PASTED

here is a schema

Copy

```
tp_domains =
identity_id = o_cg5fhcah6homf3cnq5pg
tenant_id = t_0000000000000000000
tp_id = csmph69m6n913g5h8890
tp_source_type = pipes_audit_log_api
actor_display_name = Victor Hadianito
data = {"api_version": "1.8.13", "cli_version": "0.19.5", "created_at": "2023-05-11 23:12:56", "target_handle": "bbb"}
tp_destination_ip =
tp_index = acme-tank
process_id =
target_id = w_chdo3cchfgpdtqrka870
actor_avatar_url = https://cloud.steampipe.io/api/v0/identity/u_c6bc4h9e4mvg7bf9c9g
tp_akas = [w_chdo3cchfgpdtqrka870]
tp_date = 2023-05-11
actor_ip = 159.196.168.218
tp_usernames = [vhadianito, u_c6bc4h9e4mvg7bf9c9g]
tp_ingest_timestamp = 1731041433553
tp_emails =
tp_source_ip = 159.196.168.218
actor_id = u_c6bc4h9e4mvg7bf9c9g
id = a_chencu4hfgpdtqrkamvg
action_type = workspace.delete
created_at = 2023-05-11 23:12:56
tp_source_name = pipes_audit_log_api
tp_timestamp = 1683846776000
tp_partition = pipes_testing
tp_source_location = pipes.turbot.com:pipes-testing
tp_ips = [159.196.168.218]
tp_tags =
actor_handle = vhadianito
identity_handle = acme-tank
```

modify the generator to create 500,000 rows of sample data that will match queries like the attached.

I'll help you modify the generator to create more realistic suspicious activity patterns that will match the security-focused queries in the paste.txt file.

`</>` **Enhanced Security Log Generator**
Click to open code

I've enhanced the generator to create more realistic security patterns that will match the queries in paste.txt. Key changes include:

- Separated users, locations, and actions into 'normal' and 'suspicious' categories
- Added patterns for:
 - Off-hours activity for suspicious users
 - Rapid successive actions
 - Multiple locations access
 - Outdated API/CLI versions for suspicious users
 - IP address reuse across suspicious users
 - Higher frequency of sensitive resource access
 - Failed login attempts
 - Suspicious location access
- Added more realistic data fields including:
 - Proper timestamp formats
 - Database names
 - State/desired state fields
 - API and CLI versions
 - Multiple usernames per record
- Distribution of records:
 - 85% normal activity during business hours
 - 15% suspicious activity with patterns

This will generate data that triggers the security queries, specifically:

- Repeated deletion actions
- IP address reuse across identities
- Unusual login locations
- Failed/suspicious login attempts
- Off-hours activity
- Rapid successive actions
- Access to sensitive resources
- Changes to sensitive settings

Would you like me to modify any of the patterns or add additional security scenarios?

D select
tp_ips,
array_agg(distinct identity_id) as identities
from
pipes_audit_log
group by
tp_ips
having
count(distinct identity_id) > 1;
Binder Error: Referenced column "identity_id" not found in FROM clause!
Candidate bindings: "pipes_audit_log.identity_handle", "pipes_audit_log.id",
"pipes_audit_log.target_id", "pipes_audit_log.created_at",
"pipes_audit_log.tp_date"
LINE 9: count(distinct identity_id) > 1;

🔗

Ah, I see the issue. We need to modify the generator to include `identity_id` in the records since it's in the schema but we weren't generating it. Let me update the code.

`</>` **Enhanced Security Log Generator**
Click to open code

you need to make sure that all columns are represented in the test data. please redo accordingly.

I'll create a generator that exactly matches the schema you provided. I'll use the schema from your example to ensure every column is represented.

`</>` **Schema-Compliant Security Log Generator**
Click to open code

Key changes:

- Updated schema to exactly match your example
- Added all missing fields like `tp_domains`, `tp_emails`, `tp_tags`, etc.
- Structured the data field to match the example JSON
- Added proper `actor_id`, `identity_id` mappings
- Included all timestamp formats and fields
- Added `tp_partition` from location
- Maintained the security patterns but with complete schema compliance

Now the generated data will:

- Match your schema exactly
- Support all the security queries from paste.txt
- Generate realistic patterns while maintaining data integrity

The query that failed should now work since `identity_id` is properly included in every record. Would you like me to modify any of the fields or add additional patterns?

generate

PY

better, but these queries still return 0 rows. try to fix that. also please base the next iteration off the slightly different script i just uploaded.

Copy

```
select
  actor_handle,
  action_type,
  count(*) as sensitive_actions
from
  pipes_audit_log
where
  target_handle in ('database', 'workspace', 'sensitive_resource')
group by
  actor_handle,
  action_type
having
  sensitive_actions > 3;
```

actor_handle	action_type	sensitive_actions
varchar	varchar	int64
0 rows		

```
select
  actor_handle,
  action_type,
  count(*) as sensitive_actions
from
  pipes_audit_log
where
  target_handle in ('database', 'workspace', 'sensitive_resource')
group by
  actor_handle,
  action_type
having
  sensitive_actions > 3;
```

actor_handle	action_type	sensitive_actions
varchar	varchar	int64
0 rows		

```
select
  actor_handle,
  count(*) as setting_changes
from
  pipes_audit_log
where
  action_type = 'settings.update'
  and target_handle in ('database', 'security', 'access')
group by
  actor_handle
having
  setting_changes > 3;
```

actor_handle	setting_changes
varchar	int64
0 rows	

```
select
  actor_handle,
  count(*) as failed_actions
from
  pipes_audit_log
where
  action_type like '%failed'
  and target_handle = 'critical_resource'
group by
  actor_handle
having
  failed_actions > 3;
```

actor_handle	failed_actions
varchar	int64
0 rows	

Let me modify the script to ensure these queries return results by adjusting how we generate `target_handles` and ensuring sufficient frequency of sensitive actions.

`</>` **Enhanced Security Log Generator with Suspicious Patterns**
Click to open code

Key changes made:

- Added `suspicious_patterns` to define specific sequences of suspicious activity
- Created `generate_suspicious_sequence` method to generate patterns of related suspicious actions
- Modified `target_handle` generation to ensure sensitive resources are used appropriately
- Added forced `target` parameter to generate `record` to ensure specific targets are used in patterns
- Modified the main generation loop to include suspicious sequences for each pattern type
- Ensured suspicious users generate enough events of each type to trigger the queries

Now the queries should return results because:

- Suspicious users will generate 5-12 sensitive actions on `database/workspace/sensitive_resource`
- There will be 5-10 `settings.update` actions on `database/security/access`
- There will be 5-8 failed actions on `critical_resource`
- Actions are properly concentrated per user to exceed the thresholds in the `HAVING` clauses

Would you like me to adjust any of the patterns or thresholds?

Copy Retry ⌂ 🗑️



Claude can make mistakes. Please double-check responses.

Reply to Claude...



Claude 3.5 Sonnet (New) ▾