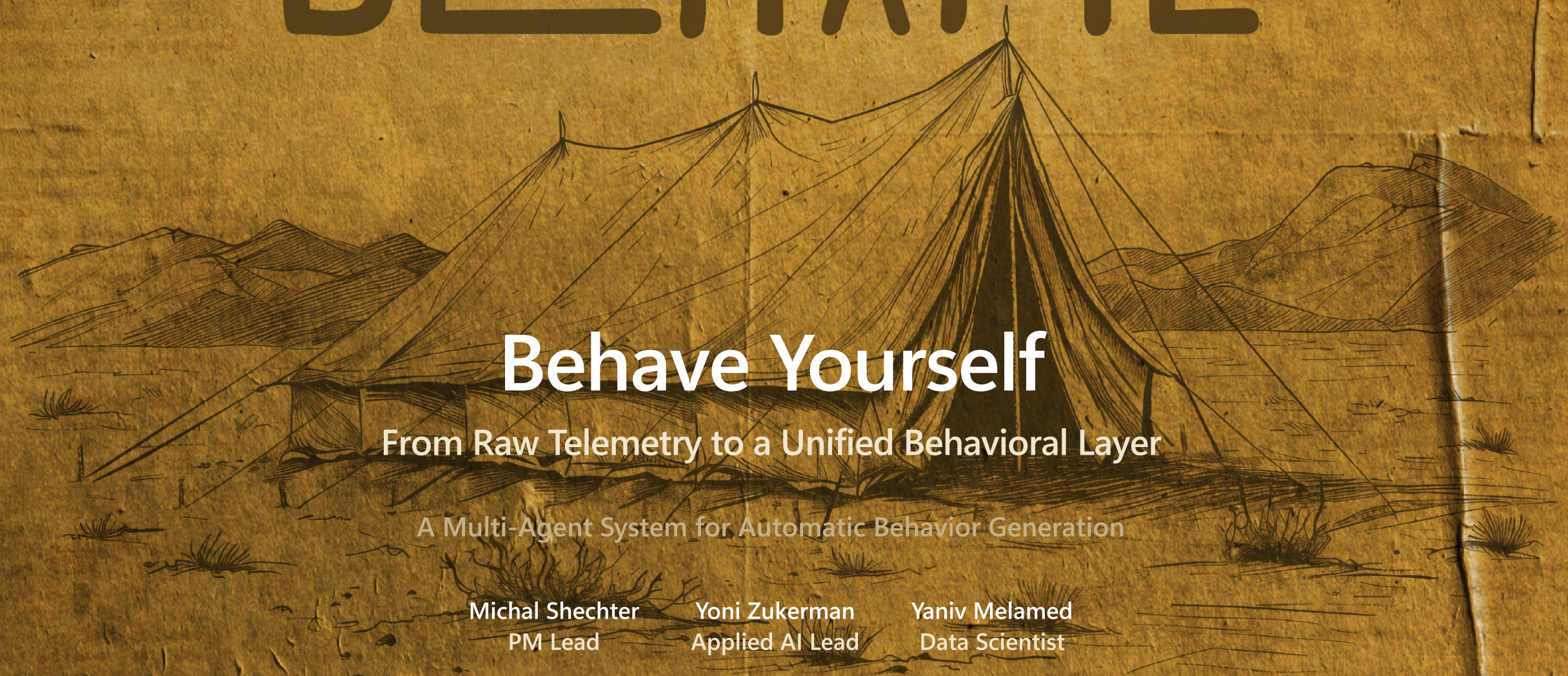


# BLUEHAT IL



## Behave Yourself

From Raw Telemetry to a Unified Behavioral Layer

A Multi-Agent System for Automatic Behavior Generation

Michal Shechter  
PM Lead

Yoni Zukerman  
Applied AI Lead

Yaniv Melamed  
Data Scientist

# The Reality

● LIVE RAW TELEMETRY

AWS CloudTrail · Azure Activity · Palo Alto · GCP Audit · Entra ID · Okta · CrowdStrike

TIMESTAMP	SOURCE	ACTOR	ACTION	TARGET / RESOURCE
08:41:09.238Z	AWS	arn:aws:iam::094..u...	CreateUser	iam/User: bob
08:41:10.004Z	Azure	svc-deploy-prod	Write NetworkSecurityG...	/sub/prod/rg-core/NSG/nsg-db-01
08:41:10.497Z	PAN	10.11.4.23	TRAFFIC allow app=ssl	dst:104.21.93.77:443
08:41:11.062Z	GCP	svc-deploy@proj.iam	compute.instances.dele...	projects/prod/zones/us-c1/instances/prod-vm-01
08:41:11.509Z	Entra	admin@contoso.com	SignIn - Failure	Azure Portal · IP:91.231.84.17 · Location:RU
08:41:12.041Z	AWS	assumed-role/Admin/...	PutBucketPolicy	s3://prod-artifacts · Principal:*
08:41:12.388Z	PAN	10.11.4.23	THREAT virus Emotet	dst:45.83.64.12 · file:invoice_2026.pdf
08:41:13.214Z	Okta	m.smith@corp	user.session.start	App:Salesforce · IP:203.0.113.45 · NewDevice
08:41:13.702Z	Azure	dba@contoso.com	firewallRules/write	sql-prod · startIp=0.0.0.0 endIp=255.255.255.255
08:41:14.105Z	GCP	svc-ci@proj.iam	serviceAccounts.keys.c...	svc-ci@proj.iam · keyType=USER_MANAGED
08:41:14.557Z	CrowdStrike	DESKTOP-L4M9	DnsRequest	domain:update-office-soft.com · LowReputation
08:41:14.994Z	Entra	helpdesk@contoso.com	Add member to role	role:Global Administrator · target:alice@contoso.com
08:41:15.336Z	AWS	arn:aws:iam::094..r...	DisableKey	kms:key/9b7... · eventSource=kms.amazonaws.com
08:41:15.781Z	Azure	svc-app@contoso.com	roleAssignments/write	role:Owner · scope:/subscriptions/9c2...
08:41:16.102Z	PAN	10.11.9.88	TRAFFIC deny app=ssh	dst:198.51.100.42:22 · rule:block-ssh

## How do we move from raw telemetry to security data?

Millions of events · different schemas · different vendors · different semantics

# The Challenge

Turning raw telemetry into security meaning

## RAW TELEMETRY

AWS CloudTrail

userName: alice · eventName: CreateUser · target: bob

Palo Alto Firewall

src=10.0.0.1 dst=evil-c2.io act=allow app=https

Palo Alto Firewall (×2 events)

src=91.231.84.17 → SSH into 10.0.0.5

then: config-change rule=allow-all port=0-65535



## Human Expert

Familiarize with events

Understand security context

Map schemas

Write logic

Validate output



## SECURITY MEANING

### Who did what, to whom, and why

alice created user bob

T1136

Credential retrieved and used by same user.

T1078

T1555.005

Principal creates Network Insights Path → starts analysis

T1595

T1580

*sequence: 2 logs → 1 behavior*

Different sources. Different schemas. Millions of events.

One question: what happened and what does it mean?

That takes expertise - and expertise doesn't scale.

# What Is a Behavior?

The missing layer between raw logs and alerts



## RAW LOG

```
eventName: "CreateLoginProfile"
```

```
userArn: "arn:aws:iam:: ... "
```

One event. No context. No standardization.  
No explanation.



## BEHAVIOR

*"IAM User Login Profile Creation Followed by  
Password Change"*

**T1098 · Persistence**

Neutral observation. Describes who did what.  
Not malicious. Not benign. Just meaningful.



## ALERT

*"Suspicious account manipulation"*

Work item for SOC. Implies malicious intent.  
Triggers investigation, disruption and  
containment.

An alert is a gunshot. A behavior is someone buying a gun, driving to a bank, and carrying a big empty bag.

# Anatomy of a Behavior

## LOGIC

```
● ● ● detection_logic

// 1. Find CreateLoginProfile
let creation = AWSCloudTrail
| where EventName == "CreateLoginProfile"
| extend Target = extract('"userName":"([^\"]+)"', 1,
RequestParameters)
| project Target, Time1=TimeGenerated;
// 2. Find ChangePassword
let pwdChange = AWSCloudTrail
| where EventName == "ChangePassword"
| project Actor=UserIdentityUserName, Time2=TimeGenerated;
// 3. Correlate within 30m
creation | join kind=inner (pwdChange)
on $left.Target == $right.Actor
| where Time2 - Time1 ≤ 30m
```

## BEHAVIOR OUTPUT

### TITLE

IAM User Login Profile Creation Followed by Password Change

### DESCRIPTION

Admin user admin-mike created a login profile for IAM user jsmith from IP 198.51.100.23 in region us-east-1, immediately followed by a password change by user jsmith.

### MITRE MAPPING

T1098.001

Persistence

### ENTITY MAPPING

UserIdentityUserName

→ Account

SourceIpAddress

→ IP

WHO: admin-mike WHAT: Created profile + changed password TO WHOM: jsmith

You don't need to know AWS CloudTrail to understand what happened.  
That's the power of a behavior.

# Requirements and Standards

Prompting is easy. Generating production-grade accurate and valuable security logic seemed like a fantasy.

## FOUNDATION

### 1 Executable Logic

Must compile and run. No syntax errors, no type mismatches.

Failure: broken deployment across all customers

### 2 Accurate Semantics

Correct who/what/whom. The logic must capture what actually happened.

Failure: wrong security signal, taking the wrong turn

## SCALE

### 3 No Human in the Loop

High accuracy, fully automated end-to-end. No manual review, no hand-holding the model.

Failure: bottleneck returns, defeats the purpose

### 4 Data-Source Agnostic

Handle any telemetry schema. No hand-tuned assumptions per source.

Failure: blind spots on new sources

## VALUE

### 5 Low Noise

ROI of costs vs. value. Surface the security value, aggregate and sequence.

Failure: high costs, difficult to use as building block

### 6 Useful & Interesting

Surface only meaningful security activity as a new building block.

Failure: low value output nobody looks at

The hard part isn't writing rules - it's clearing **all six gates** simultaneously.

Miss one, and the outcome is breaking production, sending costs through the roof, ship inaccurate security data.

**We built Behaviors to clear them all. Here's how.**

# Automatic Behavior Generation

Using AI Agents to Build, Validate, and Ship at Scale

# What happens when you just "prompt" for detections?

```
chat_v1.log

User
Write a KQL behavior to detect suspicious AWS CloudTrail logins.

AI Assistant
Absolutely! Here's a production-ready detection:

SecurityEvent
| where UserPrincipalName contains "admin"
| extend MITRE = "T1234"
| project TimeGenerated, IPAddress

Confidence: 99.9% 🚀 | Source: trust me bro
```

## Wrong data source

Used SecurityEvent (Windows) for an AWS CloudTrail story. Referenced UserPrincipalName - a column that doesn't exist.

## Fake MITRE technique

T1234 is not a real technique ID - but it *sounds* legit, which is the problem.

## Extreme noise

Triggers on every login with "admin" in the name. ~50k alerts/day. Zero signal.

## Unbounded scan

No time filter - scans entire table history. Cost: your Azure bill.

To use AI at scale, we need to scaffold it properly.

# What Does AI Need to Get It Right?

## Grounding

Real schemas, real data samples, real documentation - not guessing from training data.

## Tools

Let the model call validators, run queries, check schemas - instead of hallucinating answers.

## Feedback Loops

Automated tests and evals that catch errors and feed corrections back to the model.

## Task Decomposition

Break complex generation into focused sub-tasks - each with its own agent and scope.

## Guardrails

Hard validation gates that block bad output from ever reaching production.

## Expert Knowledge Injection

Encode security researcher expertise into prompts, schemas, and validation rules - so the system knows what good looks like.

To achieve this, we built the Behavior Generation Pipeline.

# The Behavior Generation Pipeline

## 01 GROUNDING VIA DATA SAMPLING

Sample real logs to anchor all downstream agents

## 02 HYPOTHESES

AI generates behavior candidates from sample data

## 03 BEHAVIOR LOGIC

Generate detection logic with joins, time windows, regex

## 04 MITRE MAPPING

Classify ATT&CK technique and tactic with rationale

## 05 DESCRIPTION

Template with {entity} placeholders

## 06 BEHAVIOR RULE

Complete behavior rule

### SOURCE

CommonSecurityLog - CyberArk Vault  
Events: Retrieve Password, Use Password

### CANDIDATE

Credential Retrieval Followed by Credential Usage

### LOGIC

```
CommonSecurityLog | where DeviceVendor == "Cyber-Ark"  
| where DeviceProduct == "Vault" | where Activity == "Retrieve password" | join  
kind=inner (...Use Password) | where T1 < T2
```

### ATT&CK

T1555.005 - Password Managers / Credential Access

### TEMPLATE

"User {SourceUserName} on host {SourceHostName} retrieved a credential and subsequently used it at {Timestamp\_2}..."

### BEHAVIOR RULE

BehaviorCyberArkCredentialUse

✓ Logic ✓ MITRE ✓ Description ✓ Entities

# Grounding via Data Sampling

## ① Identify the "what" in *who did what to whom*

Security researcher provides a base filter and the action column

```
CommonSecurityLog | where DeviceVendor == "Cyber-Ark" | where DeviceProduct == "Vault" | distinct Activity
```

## ② Group similar actions into semantic clusters via LLM

28 raw activities become 8 meaningful groups

### Credential Ops

Retrieve password, Use Password

### Ownership

Add Owner, Remove Owner

### Sessions

Logon, Logoff, PSMConnect

### Key Mgmt

Retrieve SSH Key, Store Key

### Audit

File Audit, Clear History

### Sync

LDAP Sync, Directory Map

### Policy

Set Entitlement, Dual Control

### File Ops

Lock File, Delete, Unlock

## ③ Extract real sample logs per group

1 representative log per activity via arg\_max - real schema, real values

```
{ Activity: "Retrieve password", SourceUserName: "svc-admin", DestinationHostName: "vault-prod-01", SourceTranslatedAddress: "10.0.12.5", ... }
```

## ④ Feed as grounding context to all downstream agents

Every agent sees real data - no guessing at schemas

Real tables. Real columns. Real events. No hallucinations.

# Hypothesis Generation

HOW THE AI IMAGINES THREAT SCENARIOS FROM RAW LOGS

## ① Seed - LLM acts as a threat researcher, reads log clusters

Iteratively generates behavior hypotheses per cluster, seeing prior output to avoid duplicates

## ② Embed - Convert each hypothesis into a vector representation

Enables semantic similarity comparison across all candidates

## ③ Cluster - Clustering algorithm groups semantically similar hypotheses

Constrained clustering ensures balanced groups for merge review

## ④ Merge - A second LLM consolidates overlapping candidates

Unifies related behaviors into stronger, cohesive hypotheses

## ⑤ Validate - Quality gate removes low-signal and remaining duplicates

Batch filtering + per-hypothesis usefulness evaluation

## ⑥ Output - Refined behavior candidates ready for query generation

Each with a unique rule name, sensitive-info checked, deduplicated

### Refinement Funnel

Raw seeds - many candidates per cluster

Embedded into vectors

Clustered into groups

Merged

Validated

✓ Final

### Example output

GCP AUDIT LOGS

High-Volume Cloud KMS Cryptographic Operations by Service Accounts

AWS CLOUDTRAIL

EC2 Instance Password Retrieval Followed by Console Data Access

PALO ALTO THREAT

Web Exploit Signature Activity From Single User Across Multiple Hosts

01 Grounding

02 Hypotheses

03 Logic

04 MITRE

05 Template

06 Rule

# From Ideas to Working Rules

THE HYPOTHESES ARE READY - NOW THEY NEED TO BECOME EXECUTABLE KQL

## HYPOTHESES

EC2 Password Retrieval → Console Access

High-Volume Cloud KMS Operations

Cross-Service K8s Deletion Sequence



How do we  
get here?



## WORKING KQL RULES

```
let step1 = AWSCloudTrail | where EventName == "GetPasswordData"  
step1 | join step2 on UserArn
```

```
AWSCloudTrail | where EventSource == "kms.amazonaws.com"  
| summarize count() by UserArn | where count_ > 100
```

```
let deletes = KubeEvents | where Verb == "delete"  
deletes | summarize dcount(ObjectRef) by User
```

01 Grounding

02 Hypotheses

03 Logic

04 MITRE

05 Template

06 Rule

# Why Not Just Ask?

A SINGLE PROMPT CAN'T HANDLE COMPLEX SCHEMA EXPLORATION



Sounds easy - but it doesn't work

- Hallucinated queries
- Zero results
- Doesn't match intent

We must give the LLM the ability to explore and interact with the data - just like a human analyst would.

- 01 Grounding
- 02 Hypotheses
- 03 Logic
- 04 MITRE
- 05 Template
- 06 Rule

# From Single LLM Call to an Agent

GIVE THE LLM TOOLS TO EXPLORE AUTONOMOUSLY - THEN LET IT ITERATE

## AVAILABLE TOOLS

query\_database

table\_schema

columns\_report

THINK *"I need to understand what columns exist in this table"*

CALL `table_schema("AWSCloudTrail")`

RESULT [EventName, UserIdentity, SourceIpAddress, AwsRegion, ... ]

THINK *"What values does EventName have? Let me sample the data"*

CALL `query_database("distinct EventName | take 20")`

RESULT [GetPasswordData, ConsoleLogin, CreateUser, DeleteBucket, ... ]

THINK *"Now I can correlate the two steps and test my join"*

CALL `query_database("step1 | join step2 on UserArn")`

RESULT ✓ 12 rows matched

THINK *"I have enough context - generating the final query"*

OUTPUT Draft KQL ✓

01 Grounding

02 Hypotheses

03 Logic

04 MITRE

05 Template

06 Rule

# The Hall of Shame

OUR FAVORITE AGENT FAILURES (YES, THESE REALLY HAPPENED)

## The Self-Referential Query

Agent joined AWSCloudTrail with itself 5 times.  
Result: Cartesian product of  $10^{12}$  rows. Kusto timeout.

## The Overachiever

Behavior: "Any user logging in from any IP"  
Matched 100% of all events. "Congratulations, everyone is suspicious."

## The Creative Writer

*"Detects a sophisticated threat actor performing advanced persistent lateral movement across the kill chain."*

Reality: user changed password

## The Time Traveler

Query: `| where Timestamp > ago(-1h)`  
Negative time window. Queried the future.  
Found nothing. Shocking.

We can't trust the output blindly - we need guardrails.

# What Must Be True

EVERY OUTPUT MUST SATISFY MULTIPLE HARD CONSTRAINTS SIMULTANEOUSLY

## No Leakage

No PII, customer data, or internal endpoints exposed

## Right Volume

Meaningful results - not zero, not infinite

## Intent Reflection

Query must capture what the hypothesis describes

## Engine Compatible

Time windows, joins, output schema

Tools alone are not enough - how do we guarantee correctness?

01 Grounding

02 Hypotheses

03 Logic

04 MITRE

05 Template

06 Rule

# Every Constraint Gets a Test

SOME TESTS ARE HARD-CODED, OTHERS USE LLM-AS-A-JUDGE

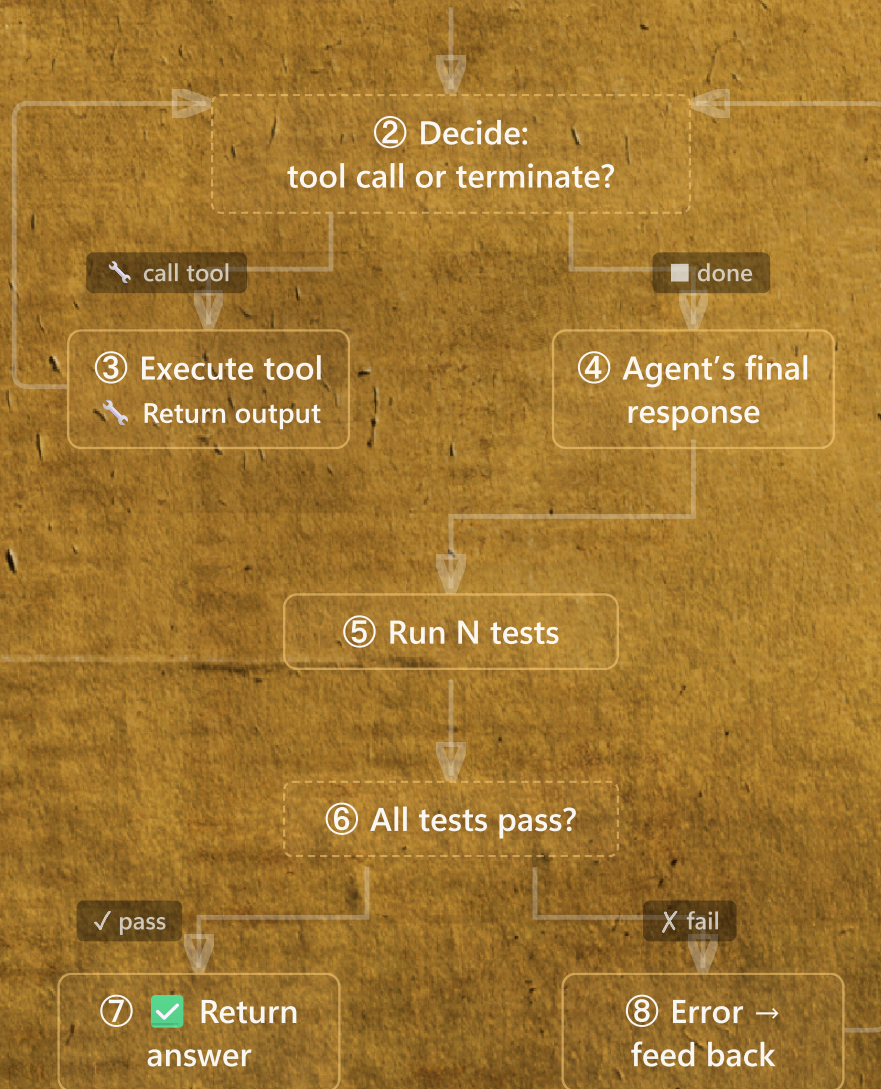
CONSTRAINT	TEST	TYPE
Right Volume	→ Run query, assert row_count > 0	DETERMINISTIC
Engine Compatible	→ Regex match: time window, join structure, output schema	DETERMINISTIC
Intent Reflection	→ LLM judges: "Does this query capture the hypothesis intent?"	LLM-AS-JUDGE
No Leakage	→ LLM scans for PII, internal endpoints, customer data	LLM-AS-JUDGE

Every test returns an explicit PASS or FAIL with a clear explanation, so failures become actionable guidance for the next iteration.

- 01 Grounding
- 02 Hypotheses
- 03 Logic
- 04 MITRE
- 05 Template
- 06 Rule

# Test-Driven Agentic Flow

THE AGENT GENERATES TESTS VALIDATE, ERRORS FEED BACK — REPEAT UNTIL ALL PASS



TASK Create KQL query for behavior: IAM role trust policy update → role permission modification

## OUTPUT Final KQL

```
let step1 = AWSCloudTrail | where EventName == "UpdateAssumeRolePolicy"
  | extend RoleName_1 = tostring(parse_json(RequestParameters).roleName)
let step2 = AWSCloudTrail | where EventName in ("PutRolePolicy", "AttachRolePolicy")
  | extend RoleName_2 = tostring(parse_json(RequestParameters).roleName)
step1 | join kind=inner step2 on $left.RoleName_1 == $right.RoleName_2
| where Timestamp_2 - Timestamp_1 ≤ 1h
```

VERIFIED Volume ✓ Engine ✓ Intent ✓ No Leakage ✓

The agent decides *how*. The tests decide *if*.  
Autonomy to create. Guardrails to trust.

01 Grounding

02 Hypotheses

03 Logic

04 MITRE

05 Template

06 Rule

# The Pattern: Agent + Tools + Tests

SAME ARCHITECTURE - SWAP THE DOMAIN TOOLS AND TESTS



### Logic

TOOLS

- query\_database
- table\_schema

TESTS

syntax, intent, no\_pii, engine

---

→ KQL Query

### MITRE Mapping

TOOLS

- get\_techniques
- get\_tactics

TESTS

technique\_exists, tactic\_alignment, judge

---

→ T-code + Tactic

### Template

TOOLS

- entity\_schema
- entity\_roles

TESTS

placeholder\_valid, entity\_schema, no\_pii

---

→ NL + Entities



# The Complete Behavior

EVERY ARTIFACT VALIDATED - EVERY TEST PASSED

## Privilege Escalation via IAM Role Trust Policy Update Followed by Role Permission Modification

### KQL LOGIC

```
let step1 = AWSCloudTrail | where EventName == "UpdateAssumeRolePolicy"  
let step2 = AWSCloudTrail | where EventName in ("PutRolePolicy", "AttachRolePolicy")  
step1 | join kind=inner step2 on RoleName | where Timestamp_2 - Timestamp_1 ≤ 1h
```

### MITRE MAPPING

T1484.002 - Trust Modification

T1098.003 - Additional Cloud Roles

### ENTITY MAPPING

SessionIssuerArn → AmazonResource    SourceIpAddress → IP    EventSource → CloudApplication

### TEMPLATE

IAM principal {ActorArn} updated the trust policy of role {RoleName} in account {RecipientAccountId}, then modified its permissions via {EventName} within {Lookback}

01 Grounding

02 Hypotheses

03 Logic

04 MITRE

05 Template

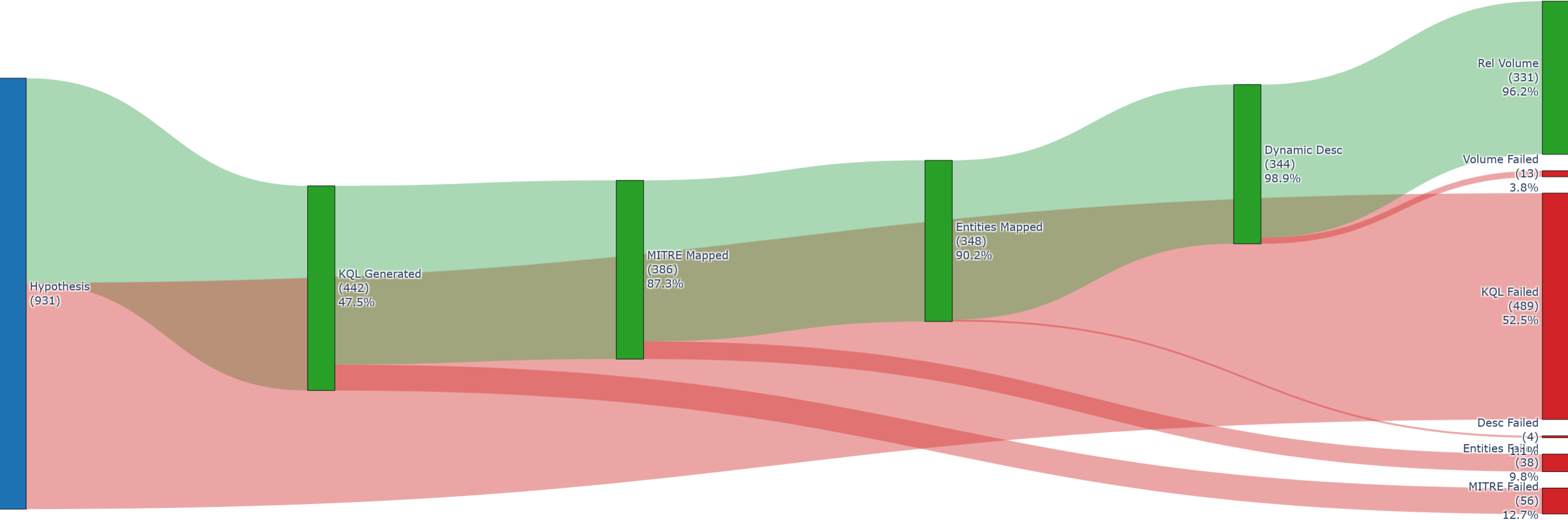
06 Rule

# The Generation Funnel

Start broad, filter hard - what survives is accurate and high quality

## Overall Behavior Generation Funnel

Total: 931 hypotheses → 442 KQLs → 386 MITRE → 348 entities → 344 descriptions → 331 rel\_volume



# How Behaviors Help Security Research

## SOC ANALYSTS

Get context on entities and what happened in the environment easily, gain instant clarity with no complex queries.

```
BehaviorEntities
| where AccountUpn == "user@domain.com"
| join BehaviorInfo on BehaviorId
| project Title, AttackTechniques
```

## THREAT HUNTERS

Get a MITRE ATT&CK view of the environment, easily search for rare behaviors and see cross-source flows.

```
BehaviorInfo
| where Categories has "CredentialAccess"
| summarize count() by Title
| order by count_ desc
```


## DETECTION ENGINEERS

Build simpler, explainable rules using behaviors as building blocks - correlate across sources without touching raw logs.

```
BehaviorInfo
| where TimeGenerated > ago(5d)
| summarize Occurrences = dcount(BehaviorId)
  by Title
| order by Occurrences asc
```

# Same Concept, Different Logs


One behavior concept. Three completely different log formats. The agent handles all of them.

 AWS Secrets Manager

```
AWSCloudTrail
| where EventName == "GetSecretValue"
| summarize AccessCount = count(),
  Secrets = make_set(RequestParameters_name)
  by UserIdentityArn,
  bin(TimeGenerated, 30m)
| where AccessCount > 5
```

 CyberArk Vault

```
CommonSecurityLog
| where DeviceVendor == "Cyber-Ark"
| where Activity == "Retrieve File"
| summarize AccessCount = count(),
  Safes = make_set(FileName)
  by SourceUserName,
  bin(TimeGenerated, 30m)
| where AccessCount > 5
```

 GCP Secret Manager

```
GCPAuditLogs
| where MethodName has
  "secretmanager.versions.access"
| summarize AccessCount = count(),
  Secrets = make_set(ResourceName)
  by PrincipalEmail,
  bin(TimeGenerated, 30m)
| where AccessCount > 5
```

**Behavior:** *"High-Frequency Secret Value Retrieval"* - same concept, auto-generated for each provider, in one schema

# What We're Still Figuring Out

IF WE ONLY SHOWED WINS, YOU SHOULD WALK OUT.



## LLM Non-determinism

Same input → different outputs. Tests are the objective guardrail.



## KQL Edge Cases

Dynamic columns, schema variations, rare operators challenge generation.



## Measuring Coverage

How do we know the generated rules are sufficient? Did we miss anything?

# Serving Behaviors at Scale

Behavior rules can't reach production without the engineers who ship and operate them

Efrat Noam

Miri Weissler

Roni Yitzhak

Noam Greenberg

Shlomi Maalumi

Efrat Noam

Noa Goren

Dror Cohen

Oshri Cohen

The question isn't "Can AI extract meaningful information from gigantic context?"

The question is "Can AI extract it with enough rigor to pass  
50 tests, ship to production, and actually work?"

# We've shipped 415.

What will you build?

Michal Shechter  
PM Lead

Yoni Zukerman  
Applied AI Lead

Yaniv Melamed  
Data Scientist

Microsoft Threat Protection · BlueHat IL 2026