
FloCon 2023
JAN 9-12, 2023 | SANTA FE, NM

databricks

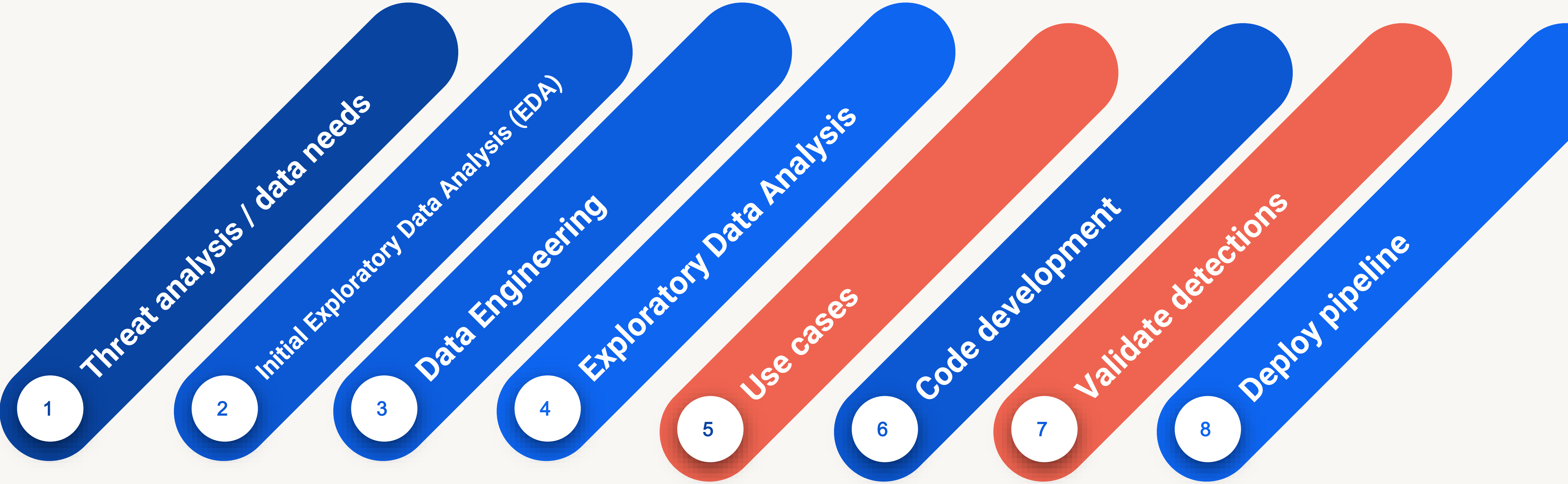
Data-Driven Detection Using PySpark

Markus De Shon

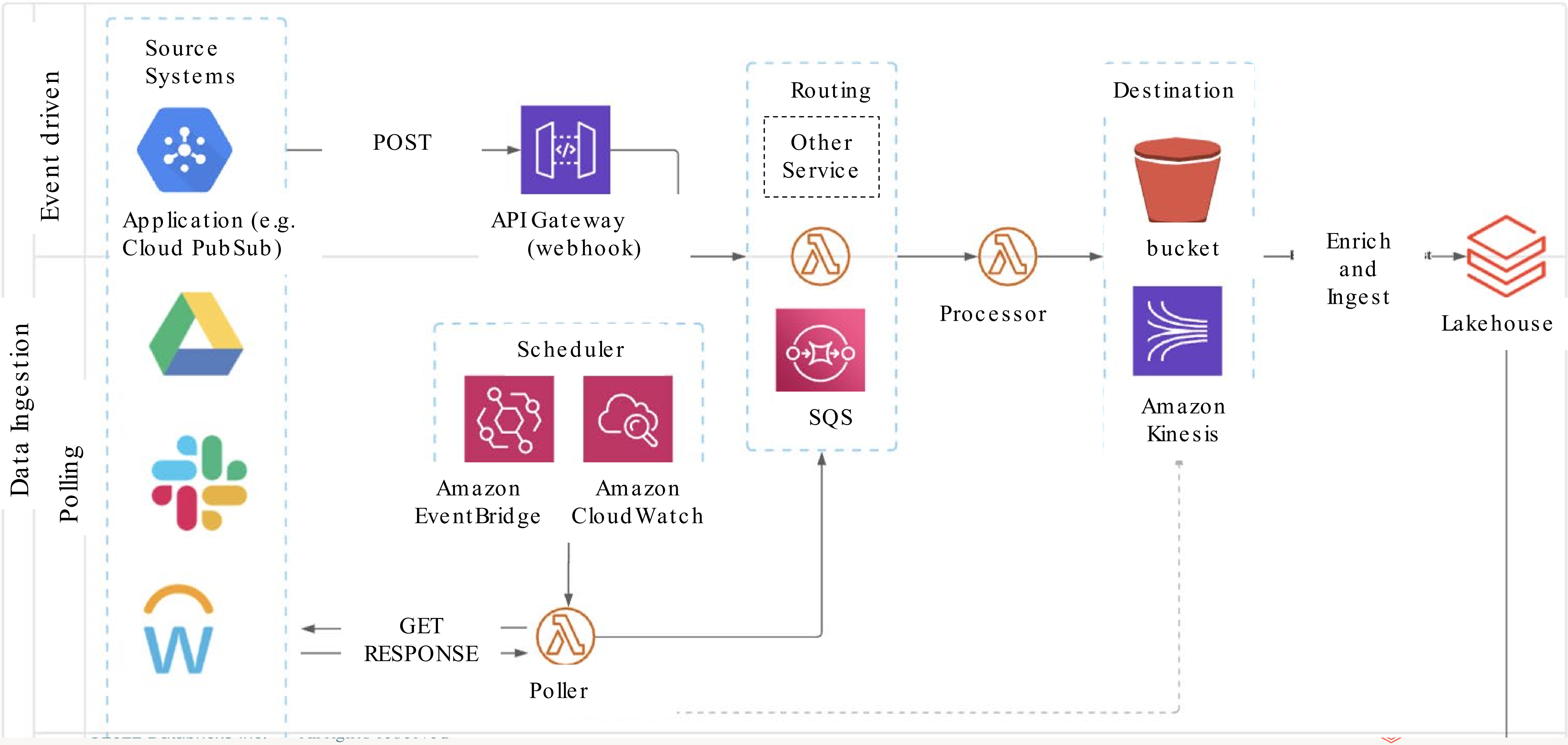
FloCon 2023

V1.0

Data-driven Detection Engineering process



Connect & Ingest: Event Driven or Polling



Connector configuration

```
{
  "poller_type": "okta",
  "url" : "https://<domain> /api/v1/logs",
  "custom_config" :{
  },
  "credentials": {
    "location" : "/security/creds/okta"
  },
  "state": {
    "location" : "/security/state/okta_cursor"
  },
  "output" : {
    "prefix" : "okta_logs"
  },
  "test_interval" : 4,
  "timestamp_format" : "%Y-%m-%dT%H%%3A%M%%3A%SZ"
}
```

```
{
  "date" : {
    "field_in" : "published",
    "format_in" : "%Y-%m-%dT%H:%M:%S.%fZ",
    "type_in" : "datetime",
    "type_out" : "date"
  },
  "timestamp" : {
    "cursor": true,
    "field_in": "published",
    "format_in": "%Y-%m-%dT%H:%M:%S.%fZ",
    "required": true,
    "type_in": "datetime",
    "type_out": "datetime"
  },
  "actor" : {},
  "authenticationContext" : {},
  "client" : {},

. . .
```

Feature Extractors

Unit-tested Python...

```
class OktaFeatures(object):
  @staticmethod
  def targetUserGroup():
    """UserGroup displayName"""
    return expr("""LOWER(
     FILTER(
      target,
      x -> x.type = 'UserGroup'
     )[0].displayName)
    """)
```

```
@pytest.mark.usefixtures("spark_session")
def test_targetUser(spark_session):
  result = (
   create_test_data(spark_session)
   .withColumn(
     "targetUser",
     OktaFeatures.targetUser(
       user_type = "AppUser"))
   ).head().targetUser
   assert (
     "firstname.lastname@example.com"
     == result)
```

Directed graph of jobs to produce views

YAML Rules

metadata, actor, target


```yaml
name: admin_grant_user
summary: admin permission
granted to user on <oktaURL>
severity: medium
source: okta
sourceDetails: <oktaURL>
alertClass: ALERT
ruleVersion: 1
eventTime: <timestamp>
```

```yaml
actor:
  type: USER
  domain: <oktaURL>
  id: <actor.alternateId>
  beliefCompromised: 0.1
target:
  type: USER
  domain: <oktaURL>
  id: <targetUser>
  beliefCompromised: 0.1
```

YAML Rules (2)

attacks, indicators, context

```yaml
attacks:
- mitre:
    taxonomy: ENTERPRISE
    tactic: PRIVILEGE_ESCALATION
    techniqueId: T1548
    technique: Abuse Elevation
Control Mechanism
indicators:
  ipAddresses:
  - <client.ipAddress>
  domains: []
  fileHashes: []
  urls: []
```

```yaml
context:
  Okta URL: <oktaURL>
  UserAgent:
<client.userAgent.rawUserAgent>
  ipAddress: <client.ipAddress>
  who: <actorUser> from
<client.ipAddress> grant
<admin_grant> privilege to
<targetUser>
```

YAML Rules (3)

filter, test cases

```
filter: |-
  eventType =
'user.account.privilege.grant' and
  admin_grant IS NOT NULL and
   (actor.type is NULL or actor.type !=
'SystemPrincipal' or actor.alternateId !=
'system@okta.com')

test_cases:
- test_name: 'expected hit'
  expected_result: true
  expected_title: 'admin permission granted to user on
okta.example.com'
  expected_context:
    {
      "Okta URL": "okta.example.com",
      "UserAgent": "OktaVerify",
      "ipAddress": "1.2.3.4",
      "who": "bob@example.com from 1.2.3.4 grant
app123-admin privilege to alice@example"
    }
```

```
test_input:
  {
    "actor": {"type": "User", "alternateId":
"bob@example.com"},
    "eventType": "user.account.privilege.grant",
    "oktaURL": "okta.example.com",
    "client": {"userAgent": {"rawUserAgent":
"OktaVerify"}, "ipAddress": "1.2.3.4"},
    "actorUser": "bob@example.com",
    "targetUser": "alice@example",
    "admin_grant": "app123-admin",
    "timestamp": "2022-01-01"
  }
```

(Beware NULL in string matching..)

```
SELECT
  field != "bla",
  field IS NULL OR field != "bla"
```

```
(NOT (NULL = bla))   ((NULL IS NULL) OR (NOT (NULL = bla)))
null                                   true
```

Operationalizing ML in Security

Key to ML success: Clarify value, deliver quality

Tied to threat analysis

Covers a gap that cannot be addressed with rules

Success stories:

- Burner domains detection
- Volumetric anomalies for data exfiltration detection
- Cloud API error rate anomalies (e.g. AWS CloudTrail) to detect compromised credential/user/role/service
- Agglomerative cluster model of Cloud API call characteristics to identify automated activity, admin activity, anomalies

Challenges: ML for Intrusion Detection

Data quality

Labeling

Model performance

IR Acceptance

Integrations

Scalability

Maintenance

Model Drift

IR Acceptance

Base acceptance criteria, not just ML (Cory Altheide)

1. **Alerts contain relevant information**
- 1.1 Alerts should contain all information required to resolve
  - 1.1.1 Event timestamp/time information
  - 1.1.2 Entity Identifier
  - …
- 1.2 Playbook contains source/tracing information

1. **Alerts are associated with a clear action/response**

3. **Alert volume is reasonable**
- 3.1 Analyzed over 30 days of data OR staged to test alert destination and analyzed for 7 days.

4. **Alerts correspond to true -positive events**
- 4.1 Alert false/true positive ratio has been measured
- 4.2 Measured alert true alarm rate goal by severity (High (Pager): 99-100%; Medium: 90+% Low: 75+%)
- 4.3 Mechanism for suppression.

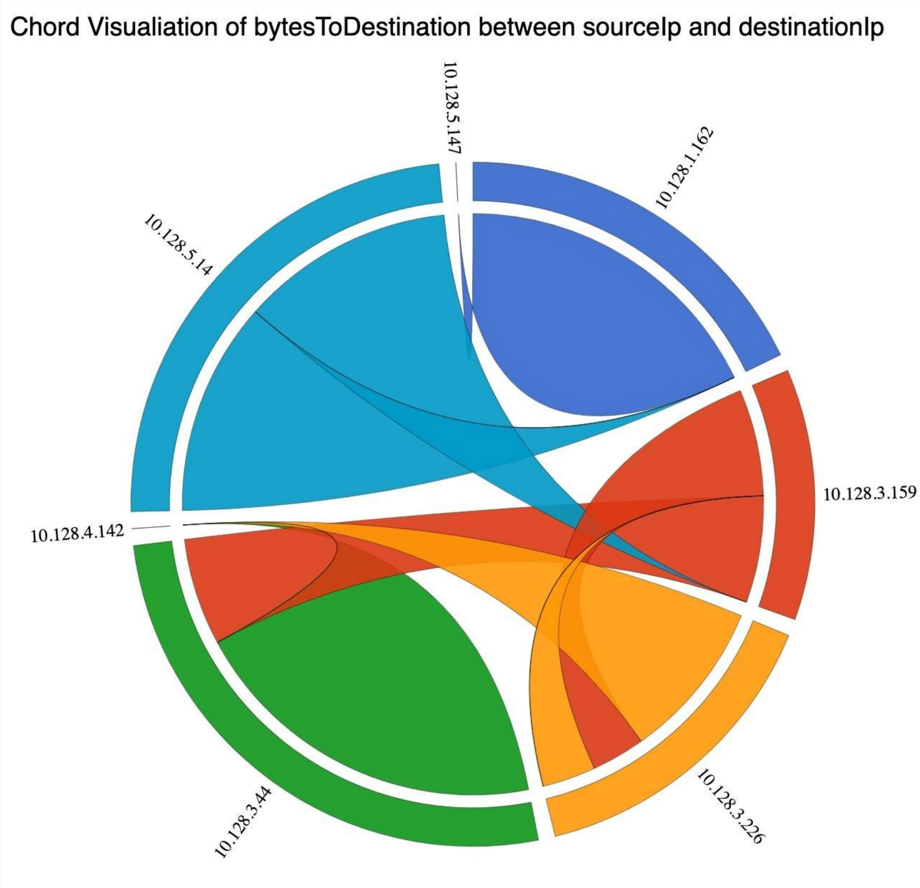
Further criteria for ML

Explainable

- Relevant feature(s) are identified in the alert
- Feature sets mapped to intrusion scenarios

Playbook

- Explains design of the model & expected results
- Pulls relevant context
- Visualizes patterns
  - Time series
  - Graphs
  - Volumetric representations (e.g. Chords)
- Suggests further investigation steps



Contributors

Engineering Team

Derek King * Jason Trost * Jason Sommerfield * Julian Shalaby * Matt Yang * Alex Ott * Maximiliano Lagos * Franco Mennucci

SME Team

Arjun Chakraborty * Siamac Mirzaie * Silvio Fiorito * Cory Altheide * David Wells * Arun Pamulapati * Kristin Dahl * Zafer Bilaloglu

Leadership

David Veuve * Kishore Fernando * Lipyeow Lim * Markus De Shon

Backup slides

Data quality

Expectations (cf. Great Expectations package)

Check for bogus null values ("", "NULL", "n/a", etc.)

Check certain columns are not NULL

Check certain columns are in a given set

Beware hard failures in prod – remember we're doing statistics

Labeling

Pre-labeling: sample of feature sets evaluated by SecOps

Post-labeling: Unsupervised approach with feedback from SecOps

No labeling: Just plain unsupervised

It's just an event: ML detections alert only when correlated

Model performance

Require a high True Alarm Rate: TP / (TP + FP)

Sensitivity / Precision: Base rate fallacy

What is "true" when there are almost no actual intrusions in the data?

→ Worth investigating (in the judgement of SecOps)

Keep track of risk

→ How much do I believe this is a real intrusion? (belief)

→ How bad would it be if this is a real intrusion? (impact)

→ Risk = belief * impact

Integrations

Implement in a framework, OR Inject to an existing events/alerts API

Run on same timeline (batch interval vs. streaming)

Standardize alerts

- Use correlatable metadata
- Focus on more persistent entities (e.g. cluster not instance)
- Standardize entity identifiers

→ Map temporary IDs (e.g. IP address) to more permanent IDs (e.g. hostname/serial number)

Scalability

Data volume / variety / velocity

Simultaneous models (relevant entities, beware key explosion)

PySpark: Pandas UDFs not Python UDFs

Maintenance

Model, data, and use case documentation

Track dependencies, test & update

Monitor for errors, crashes

Model versioning

Model drift

Monitor for changes in alert volume

Monitor for false positive surges

If unsupervised, re-train occasionally