

Supercharging Security with RAG

Anthony Tellez & Leo Meyerovich



louie.ai



[graphistry](https://graphistry.com)

Anthony G. Tellez

SVP, Digital Assets, Data & Intelligence at BNY

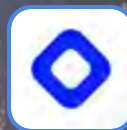
Cybersecurity expert & AI strategist with over 10 years of experience in the field.

Worked with various organizations, including government agencies, financial institutions, and healthcare providers, helping to improve their cyber security posture leveraging machine learning while at Splunk & Palo Alto Networks.

Anthony is a speaker at various industry conferences. Published several blogs on cyber security topics and AI.

Patent:11843622 “Providing machine learning models for classifying domain names for malware detection”

Publication:20230410072 “Systems and methods for enhanced non-fungible tokens”



Leo Meyerovich

Founder & CEO of Graphistry

Data-intensive investigations:

- **Graphistry & GFQL:** GPU visual graph AI
- **Louie.AI:** LLM investigations - genAI notebooks, dashboards, & pipelines
- **Ask me about Graph RAG**
Real-time continuous genAI ingest, monitoring, DB, & triggering

Won 2022 US Cyber Command AI RPE,

+ ACM SIGPLAN 10 Year Test of Time Award, multiple best paper awards

Helped start Apache Arrow & GPU dataframes, browser parallelization & security, first formal semantics for CSS, formal methods for analyzing access control policies, & functional reactive programming for the web



louie.ai



graphistry

TLDR;

Rule management sucks across security products.

Why not use AI to help humans write more efficient rules and find redundancy?

Better: *Use AI to automatically create rules and deploy?*

Similar RAG methods for:
AI event correlation, AI sitreps,
incident aggregation, auto-wiki, ...

RULES

**RULE
MANAGEMENT**

AI RULES

**AI RULE
MANAGEMENT**



Agenda

Large Language Models & RAG

General Guidance

Automating Rule

Creation

RAG with Graphs

Applicability to Security

Visualizing Netflow

References



Large Language Models & RAG



Large Language Models (LLMS) in Cyber Security

Large language models (LLMs) are powerful AI models that can process and generate human-like text.

LLMs can be used in cyber security to analyze and generate text-based data, such as logs, emails, and chat transcripts. [5]



Example of Retrieval Augmented Generation (RAG)

"What are the latest vulnerabilities in Apache web servers? Include reported Suricata rules + make new."

Retrieval: "Information Retrieval" component quickly finds recent reports related to Apache vulnerabilities.

Augmentation: Combine your query + retrieved information into the LLM prompt.

Generation: The LLM response summarizes the most critical vulnerabilities, potential impacts, recommended mitigations, and candidate rules.

Saves you time, ensures you're working with the most up-to-date and accurate information.

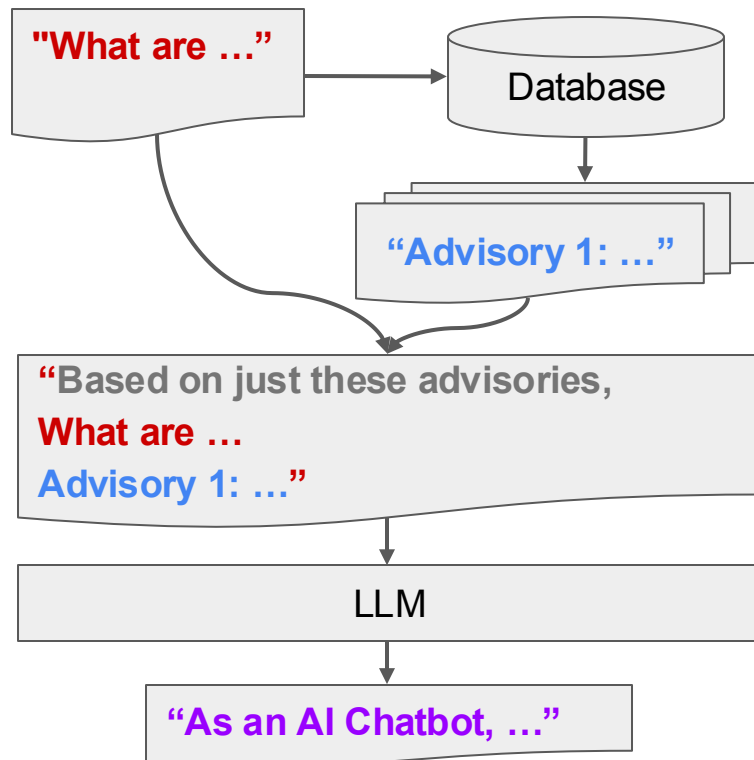


Figure: Single-shot RAG

Why is RAG Important for LLMs?



Information Availability: LLM models not trained on your database. RAG gives LLMs access to external knowledge that is precise and large.

Reduced “Hallucination”: LLMs emit facts that are statistically close... but wrong. RAG tells LLMs to only use retrieved facts.

Data Freshness: RAG adapts LLMs to new information. When knowledge bases update, the LLM gets instant access without needing retraining.

Data Confidentiality: LLM model training merges data without privacy controls. RAG retrievers can use regular access control checks to enforce only an authorized user gets access to its data.

Continuous Learning: RAG often re-used for memory and preferences. Store & retrieve past behavior to inform future ones.

High Cost/Benefit Ratio in Enhancing Security Applications with RAG



LLMs can be applied to tasks like writing Suricata rules based on security analytics for teams without access to a data scientist.

LLMs understand complex protocols and can identify correlations across data points makes them well-suited for security analysis. [6]

Improved threat detection: reasoning on text-based data (like logs).

Enhanced incident response: Provide recommendations for remediation, helping analysts to effectively respond to incidents similar to SOAR without the coding.

Better situational awareness: Comprehensive view of the cyber security landscape, enabling analysts to make more informed decisions. [10]



Utilizing RAG for Rule Management Lifecycle

We propose a framework that utilizes LLMs in conjunction with existing security tools like Suricata.

Framework leverages Retrieval-Augmented Generation (RAG) and prompt engineering:

- **Improve Alert Comprehension:** explain complex security alerts.[11]
- **Automate Suricata Rule Writing:** assist in generating accurate Suricata rules. [12]
- **Security Workflow Automation:** Data collection, enrichment and reasoning tasks. [13]

Generalized RAG Architecture

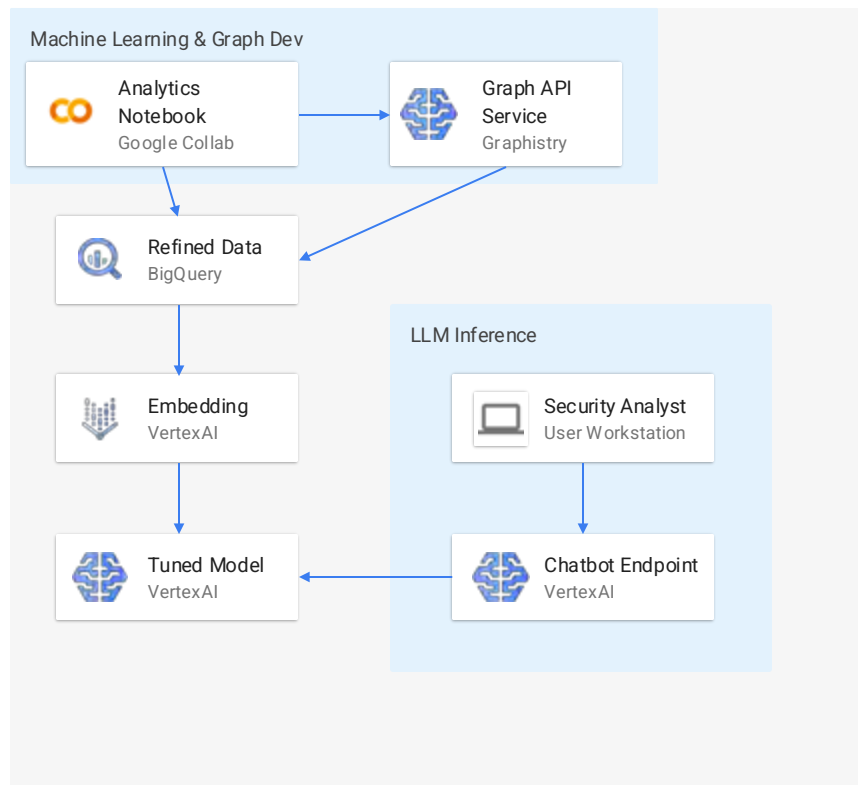
Cleaned data is processed into refined BigQuery

Netflow, Asset Information, Suricata Rules, & Threat Intelligence are processed for embedding

Vector index (+ others) contains information relevant to your network assets, threat history, and configurations

Analyst interacts with LLM getting more contextual information as they work through security alerts.

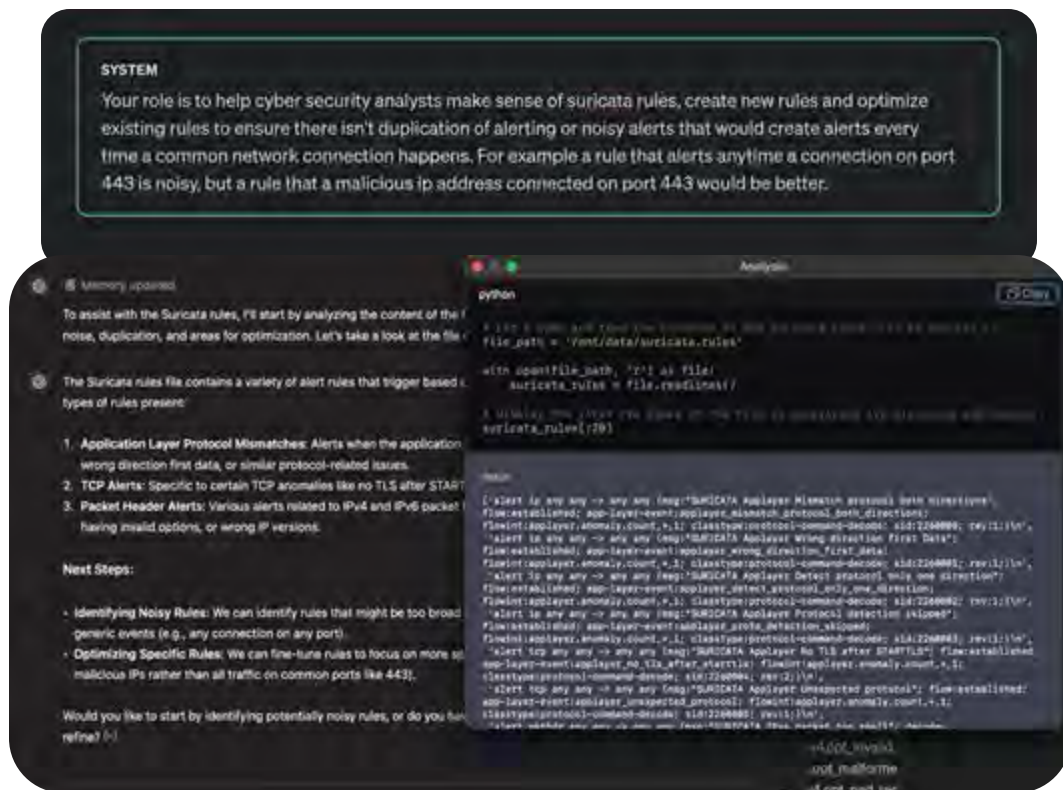
For example: *“Tell me about similar security incidents involving Apache Web Servers.”*



Prompt Engineering to Generate Rules: Attempt 1

OpenAI GPT-4o to generate rules given prompt related to the persona, conditions for what makes a good rule, and an upload of suricata.rules to learn.

We used this as our benchmark to determine how effective a generalized LLM could be at creating & analyzing rules.



Prompt Engineering: Lessons Learned

Providing Suricata rules helps the LLM accurately understand syntax, protocols and how to accomplish the task.

Problems with this approach:

- Session based output from the model
- Requires full data for each session
- No prior knowledge of conversation history (*this is starting to change*)

2. Creating a Rule for Windows Remote Desktop Connections

Windows Remote Desktop Protocol (RDP) typically uses port 3389. We can create a Suricata rule to detect connections to this port, which might indicate an RDP session.

Here's an example of such a rule:

plaintext

Copy

```
alert tcp any any -> any 3389 (msg:"ALERT: Windows RDP Connection Detected"; flow:to_server,established)
```

Explanation:

- `alert tcp any any -> any 3389`: This triggers on any TCP connection to port 3389 (RDP).
- `msg:"ALERT: Windows RDP Connection Detected"`: Custom message to identify the alert.
- `flow:to_server,established`: Ensures the rule only triggers on established connections to the server.
- `content:"Cookie"; http_header; content:"msthash="`: These content matches are specific to RDP session negotiation.
- `pcre:"/Cookie: msthash=/"`: Regular expression to further refine the match, looking for the "msthash" parameter in the cookie header.
- `classtype:protocol-command-decode`: Classification type for protocol-related alerts.
- `sid:1000001`: Unique identifier for this rule.
- `rev:1`: Rule revision number.

This rule should be more precise in detecting actual RDP sessions rather than just any connection on port 3389, reducing noise while still alerting you to potential remote desktop activities.

Evolving from Prompt Engineering to RAG Approach

Challenge: OpenAI defaults for
~Redditors

Step 1: Clarify context in the prompt

Persona

Detection Engineer vs Redditor

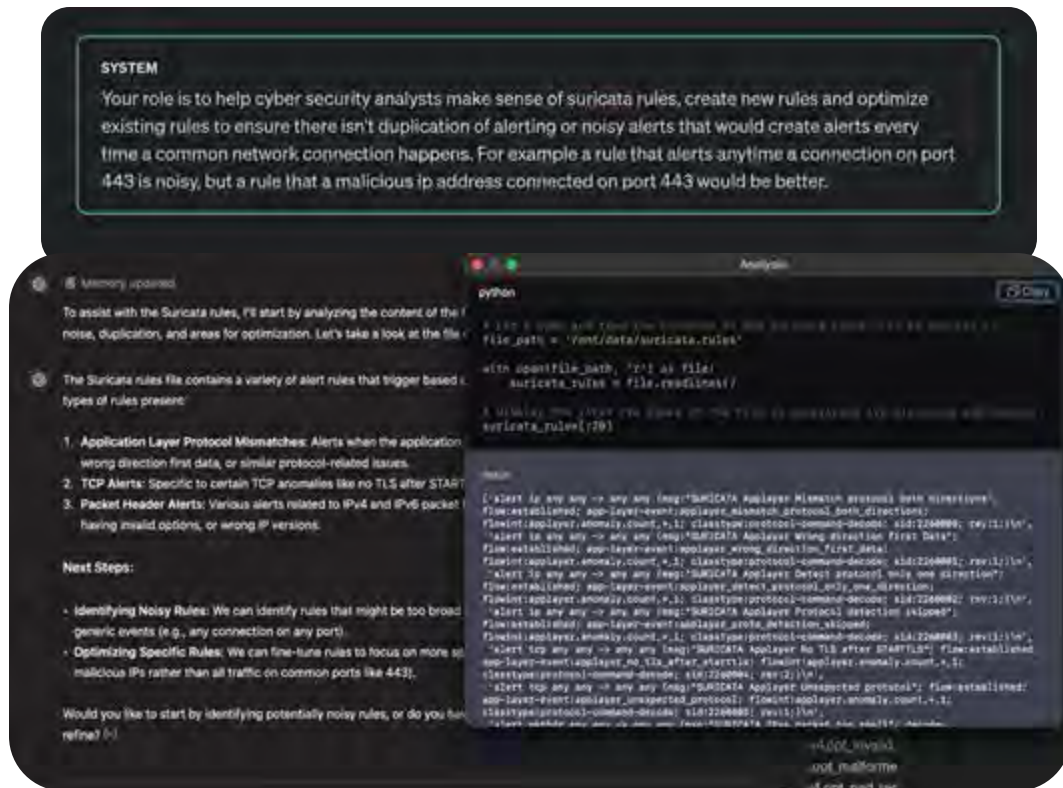
Examples

Exact good vs bad rules

Single task & its goals

General criteria like not noisy

Step 2: RAG to go from hard-coded
prompts to auto-filled & auto-optimized



Vectorizing Suricata Rules: Vector Index for Semantic Search

Semantic search is fuzzy search over DB's *vector index* of *embedded text*

Ex: Dataframe with all suricata rules
`df_rules`

1. **Rewriting:** Make rules closer to user queries by asking gpt-4o (OpenAI) to describe the purpose of each rule, append as new column `_map`. Implemented via Lotus [1].

2. **Vectorization:** Embed – convert – columns rule, `_map`, etc into one column of vectors and use as database vector index

```
1 full_annotation = df_rules.sem_map("Describe the purpose of {rule} in 15 words")
1 display(full_annotation)
```

	rule	_map
0	alert ip any any -> any any (msg:"SURICATA App...	The rule detects mismatched application layer ...
1	alert ip any any -> any any (msg:"SURICATA App...	The rule detects incorrect application layer d...
2	alert ip any any -> any any (msg:"SURICATA App...	The rule detects one-directional application l...
3	alert ip any any -> any any (msg:"SURICATA App...	The rule detects and alerts when application l...
4	alert tcp any any -> any any (msg:"SURICATA Ap...	The rule detects TCP traffic lacking TLS alter...
...
49999	alert tcp [95.217.28.112,95.217.30.201,95.217...	The rule detects non-exit Tor relay traffic to...
50000	alert tcp [95.23.252.197,95.234.237.101,95.85...	The rule detects non-exit Tor relay traffic to...
50001	alert tcp [96.20.102.87,96.21.250.174,96.232.1...	The rule detects traffic from known Tor relay ...
50002	alert tcp [96.35.102.247,96.43.209.183,96.65.6...	The rule detects non-exit Tor relay traffic to...
50003	alert tcp [98.115.87.163,98.121.68.25,98.128.1...	The rule detects traffic from known Tor relay ...
50004

50004 rows x 2 columns

[1] <https://github.com/TAG-Research/lotus/>

⇒ [0.16, 0.32, ...], [0.22, 0.86, ...], ...

Why Query Rules as Vectors?

K-nearest-neighbor search (k-nn) to get a *ranked list* or *similarity graph*:

1. Search keywords to find related rules, with similar purposes
2. Find near duplicates or verify existing rules
3. Look for stale regions, new regions, outliers, ...
4. Embedding can be used to improve rule generation for similar threats

```
152 # k-NN search
153 query_text = "Tor"
154 k = 30
155 results = knn_search(query_text, k)
156 if results:
157     for text, doc_id in results:
158         print(f"ID: {doc_id}, Text: {text}")
159
```



Vector similarity graph

DEMO: Louie.ai Prompting for Rules & Embeddings

Hey louie, get 100 events to do with ssh traffic in bnym-rag-index-d384

Here's a model prompt, and vector, if you need help debugging or visualization, just let me know!

```
▼ (
  semantic: "SSH traffic events",
  label: "Similarity",
  index: "bnym-rag-index-d384",
  reasoning: "The query aims to find events that are semantically related to 'SSH traffic'. The user specifies the index to use, which supports semantic search."
```

▼ qds1: {

▼ query: {

▼ knn: {

▼ vector: {

vector: [

>0.8141809811850000,

>0.127002555111111,

>0.202111570444719,

>0.170400000000000,

>0.111111111111111,

>0.021111111111111,

>0.000000000000000,

>0.101010101010101,

>0.101010101010101,

>0.101010101010101,

>0.101010101010101,

>0.101010101010101,

>0.101010101010101,

>0.101010101010101,

text

```
alert http EXTERNAL_NET any -> SHOME_NET any (msg:"ET WEB_CLIENT Evil Redirect Command")
alert t1s EXTERNAL_NET any -> SHOME_NET any (msg:"ET INFO Observed Lats Encrypt Cert")
alert http SHOME_NET any -> EXTERNAL_NET any (msg:"ET POLICY Akamai NetSession Interf")
alert tcp SHOME_NET any -> EXTERNAL_NET any (msg:"ET MALWARE Linux/djvu! IRC Command")
alert dns SHOME_NET any -> any any (msg:"ET POLICY Observed DNS Query to IP Lookup Dom")
alert http EXTERNAL_NET any -> SHOME_NET any (msg:"ET EXPLOIT_KIT Yeez JS/Encryption")
alert t1s SHOME_NET any -> EXTERNAL_NET any (msg:"ET MALWARE Observed PyPI Malicious")
alert http EXTERNAL_NET any -> SHOME_NET any (msg:"ET PHISHING Coinbase Credential Ph")
alert http EXTERNAL_NET any -> SHOME_NET any (msg:"ET WEB_CLIENT Possible Javascript")
alert http EXTERNAL_NET any -> SHOME_NET any (msg:"ET HUNTING Suspicious IAT Enable")
alert http EXTERNAL_NET any -> SHOME_NET any (msg:"ET HUNTING Suspicious Script Load")
alert http EXTERNAL_NET any -> SHOME_NET any (msg:"ET POLICY Googlebot User Agent")
alert tcp EXTERNAL_NET any -> SHOME_NET any (msg:"ET WEB_CLIENT Apple Outcastime RTS")
alert http EXTERNAL_NET any -> SHOME_NET any (msg:"ET ADWARE_POP LNKX Possible Weapon")
alert tcp EXTERNAL_NET any -> SHOME_NET any (msg:"ET EXPLOIT_0wnscript illegal dele")
alert http EXTERNAL_NET any -> SHOME_NET any (msg:"ET POLICY Localtunnel Successful Co")
alert ftp SHOME_NET any -> EXTERNAL_NET any (msg:"ET MALWARE W32/Dokbot - cs File Ex")
alert http EXTERNAL_NET any -> SHOME_NET any (msg:"ET CURRENT_EVENTS [Fireeye] Backdo")
alert http SHOME_NET any -> EXTERNAL_NET any (msg:"ET POLICY File Being Uploaded to S")
alert t1s SHOME_NET any -> EXTERNAL_NET any (msg:"ET EXPLOIT_KIT Malware Domain In TLS")
```

great, now do a umap on column rule



Case Study: Scaling Vector Indexes to 1B Rows (TI, OSINT, logs)

Before - *1B vectors x 3,072 vector length x float32 => 12,288 GB RAM*

After - *1B vectors x 64 vector length x float16 => 128 GB RAM*

Local model: Embed 20 sentences / CPU / second (OSS Triton inferencing server)

Tricks:

- Shorter vector length: Smaller base model, fine-tune model, Matryoshka, ...
- Quantization: Smaller floats - SQ, ...
- Volume reduction: Keyword tuning, spam filter model, ...
- Invest in manual/synthetic evaluation set => easily autotune each decision
- Self-host embedding when steady ingest & CPU when low volume

Going bigger: Disk inverse-mapping (IVF), cold cloud storage shards, ...

The background features a complex network graph with numerous black nodes and thin connecting lines. Overlaid on this are several large, wavy, translucent shapes in shades of orange, brown, and grey-blue, which appear to be flowing or undulating across the frame.

RAG with Graphs

RAG != Vector search: Typically retrieve from multiple index types & APIs!

Multiple query types \Rightarrow multiple DB index types

Fuzzy - Vector: “Azure AD alert” \Rightarrow “Microsoft Entra AD event”

Keyword - Text: “174.10.3.102” \Rightarrow “174.10.3.102:443” \leftarrow vectors inexact on IDs!

N-hop - Graph: (IP:A) \rightarrow (User:B) \rightarrow (IP:{X,Y,Z, ...})

Pick retrieval strategy

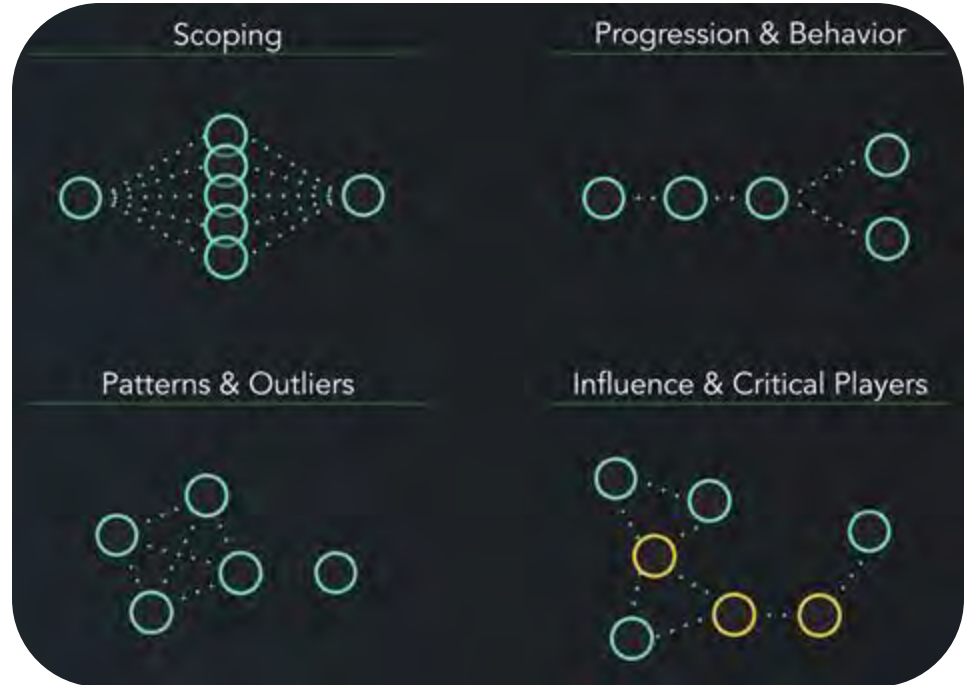
Start simple: Single-shot retrieval on one index

Advance: Multi-shot retrieval on multiple indexes - reranking, reasoning loops, ...

Graph-based Approaches for Security Analytics

Graphs model complex relationships between entities that help analyze and visualize cyber security threats.

Correlate across all security data:
Network traffic, system logs, cloud resources, shadow IT, identity, ...[1]



“Defenders think in Lists. Attackers think in Graphs. As long as this is true, attackers win.”

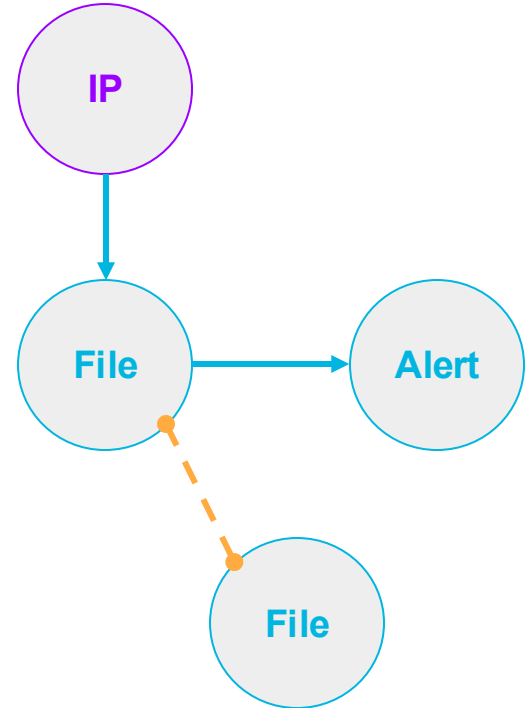
- John Lambert, Microsoft

How to think of your data as different graphs: Network, Attack, Anomaly

Physical - **Ex: Network graphs:** Representing network topology, where nodes are devices or systems, and edges represent connections between them.

Logical - **Ex: Attack graphs:** Modeling the sequence of attacks and vulnerabilities, where nodes represent physical & logical entities, and edges represent events & alerts connecting them

Statistical - **Ex: Anomaly graphs:** Identifying unusual patterns, or behavior in network traffic or system logs, where nodes represent events or entities, and edges represent relationships between them, including AI-inferred correlations



Benefits of Graph-Based Approaches

Improved threat detection: Graph-based approaches can identify complex patterns and relationships that may not be visible using traditional methods.

Enhanced incident response: Graph-based approaches can help analysts quickly identify the scope and impact of an attack to prioritize response efforts.

Better situational awareness: Graph-based approaches can provide a comprehensive view of the network and its relationships, enabling analysts to make more informed decisions. [3]



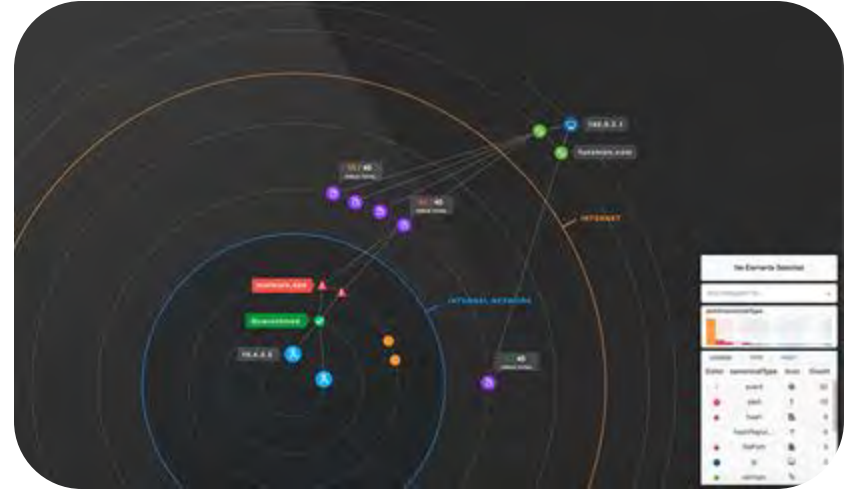
Popular Solutions for Graph Processing and Analysis

NetworkX: A Python library for creating, manipulating, and analyzing complex networks. [4]

GraphX: is Apache Spark's API for graphs and graph-parallel computation, with a built-in library of common algorithms.

GFQL & RAPIDS cuGraph: Python dataframe libraries for GPU graphs

Graphistry: GPU visual graph AI on top of GFQL/RAPIDS ecosystem



GRAPHISTRY



RAPIDS

Graph Wrangling: Last century - Networkx

```
import networkx as nx

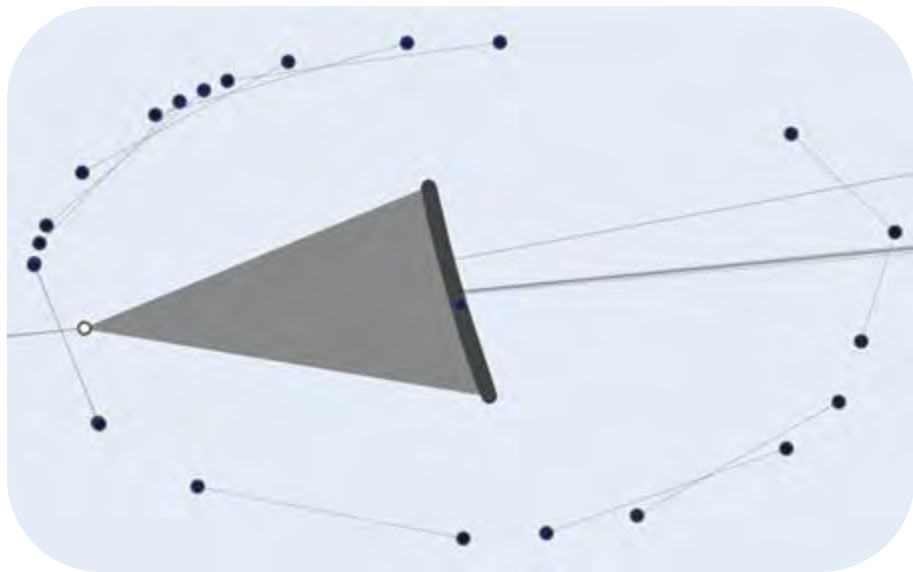
# Create a directed graph
G = nx.DiGraph()

# Iterate over the DataFrame and add edges
for _, row in df_cleaned.iterrows():
    src_ip = row['src_ip']
    dest_ip = row['dest_ip']
    src_port = row['src_port']
    dest_port = row['dest_port']
    flow_id = row['flow_id']

# Add nodes (if they don't exist)
G.add_node(src_ip)
G.add_node(dest_ip)

# Add edge with attributes
G.add_edge(src_ip, dest_ip, src_port=src_port,
            dest_port=dest_port, flow_id=flow_id)

# Draw the graph
nx.draw(G, with_labels=True)
```



Manual data wrangling, manual graph visualization, ...

Modern PyData for graphs: Dataframes!

src_ip	dest_ip	flow_id	protocol
1.1.1.1	1.2.3.4	13411	ssh
2.2.2.2	8.8.8.8	78701	dns
1.1.1.1	4.4.6.6	36919	https

```
df = spark.sql("...").collect()

g = graphistry.edges(df, 'src_ip' 'dest_ip')

g.plot()
```

Shape a graph by mapping how table columns map to nodes vs edges vs properties

Easy ML & AI algorithmic enrichments for different measures of importance on nodes, edges, subgraphs:

Graph pattern search - Multi-hop correlation hits

Clustering Coefficient & Embeddings - Measure degree to which nodes tend to cluster together.

Betweenness Centrality - Importance measure based on frequency of occurring on shortest paths.

Link Prediction: How anomalous vs likely is a relationship between two nodes?

Comparing NetworkX => Graphistry

Productivity

- 20+ lines => 3 lines (prev slide!)
- PyData ecosystem:
ML/AI/viz one-liners!
- Interactivity & intuitive
- Next level: Talk to graphs - Louie!

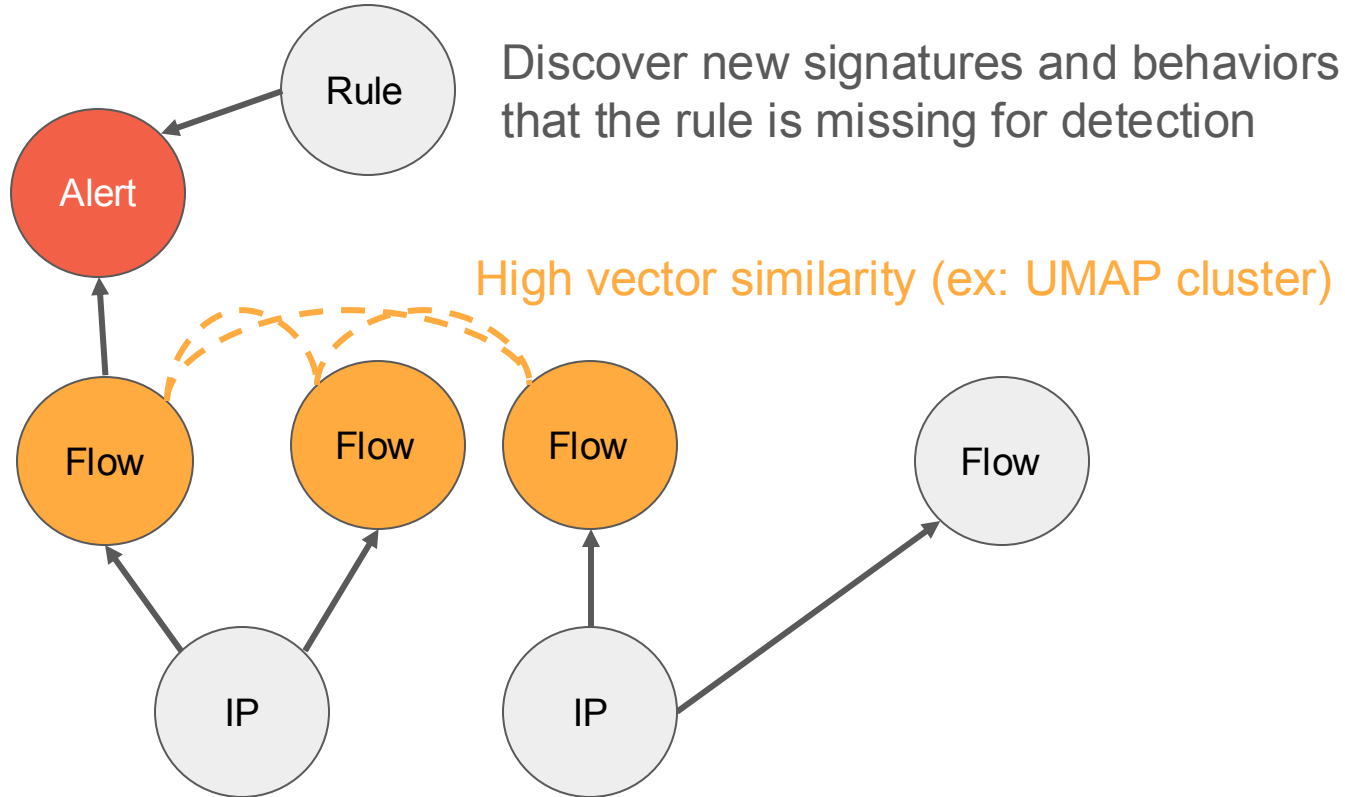


Speed & Scale

- Hunting, tuning, etc need scale, incl. viz
- Data pulls: Spark CPU scaleout fetching
- Compute: Graphistry GPU processing
- 10min+ => 10s:
many rows or non-linear algorithms



Demo: Find signatures based on missed vector-similar flows



Demo: Find signatures based on missed vector-similar flows

Aggregate attacker's
unalerted flows



UMAP-similar aggregate
attacks 2 hops out

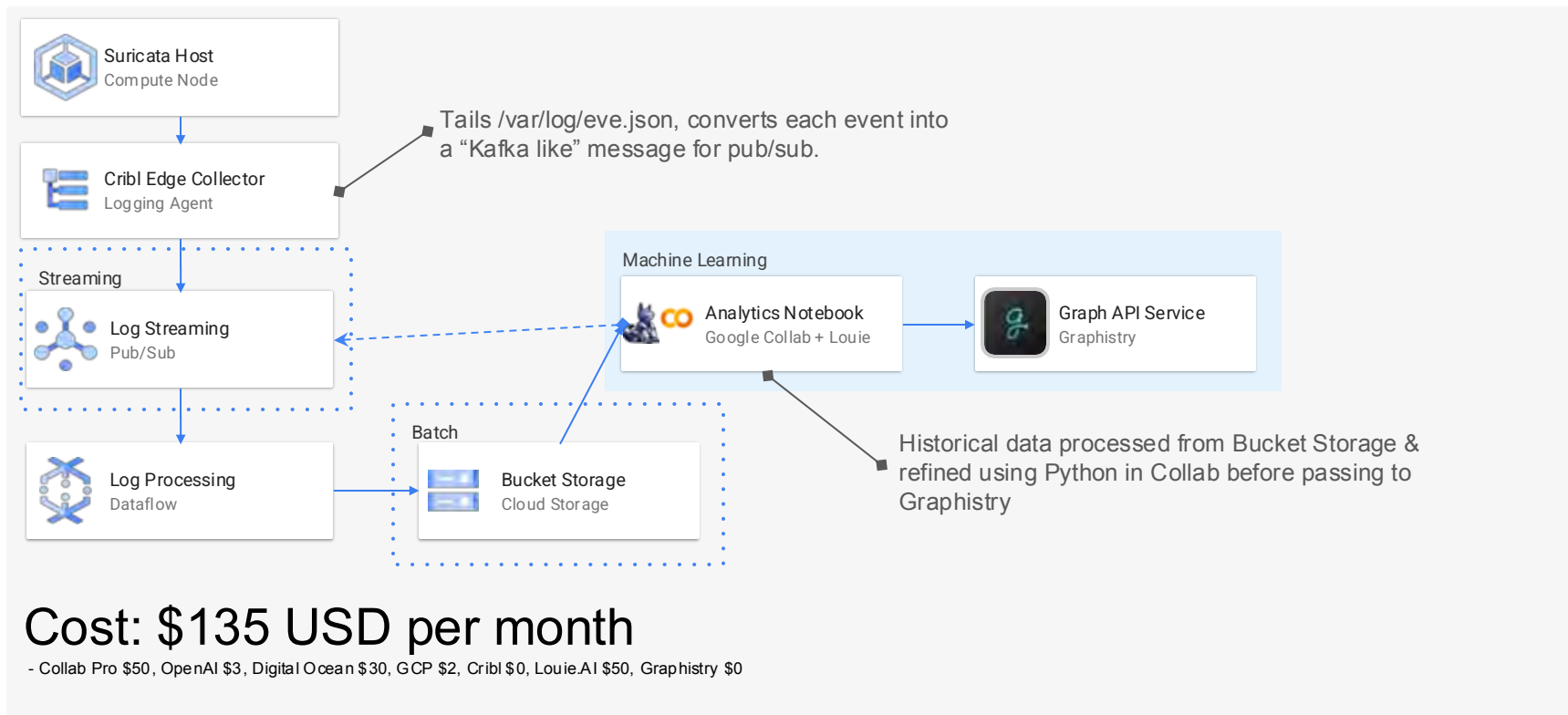


Mine unalerted clusters
& update related rules



- Variants: Individual flows vs aggregates, rule similarity vs flow similarity, ...
- Use cases like retiring, updating/generalizing, deduplicating, tuning FP/FNs, ...
- All variants of working with *embedding nearest neighbor graphs*

Suricon 2024: Deployment Architecture



References

- [1] "Leveraging Graph Analytics for Enhanced Cybersecurity" by Palo Alto Networks (<https://start.paloaltonetworks.com/unit-42-network-threat-trends-report-malware-2023.html>)
- [2] "The MITRE ATT&CK Framework™ for Threat Modeling and Detection" by The MITRE Corporation (<https://attack.mitre.org/>)
- [3] "Security Automation: Streamlining Incident Response" by McAfee (<https://www.trellix.com/services/advanced-cyber-threat-services/incident-response-services/>)
- [4] "Graphistry: A graph-based analytics platform for cyber security and threat intelligence" by Graphistry (<https://www.graphistry.com/>)
- [5] "Large Language Models and Cybersecurity: A Primer" by Cybersecurity & Infrastructure Security Agency (CISA) (<https://www.cisa.gov/>)
- [6] "Enhancing Threat Detection and Response with AI-powered Security Operations" by IBM (<https://www.ibm.com/blog/ibms-new-watson-large-speech-model-brings-generative-ai-to-the-phone/>)
- [7] "Using Artificial Intelligence and Machine Learning to Automate Threat Intelligence Tasks" by Darktrace (<https://darktrace.com/resources/darktrace-ai-combining-supervised-and-unsupervised-machine-learning>)
- [8] "The Future of Incident Response: How AI is Transforming Security Operations" by McAfee (<https://www.linkedin.com/pulse/future-cyber-protection-mcafees-latest-innovations-threats-karton-navec>)
- [9] "Deep Learning for Malware Analysis" by Palo Alto Networks (<https://unit42.paloaltonetworks.com/>)
- [10] "The Advantages of Artificial Intelligence in Cybersecurity" by Fortinet (<https://www.fortinet.com/resources/cyberglossary/artificial-intelligence-in-cybersecurity>)
- [11] "Using AI to Reduce Alert Fatigue and Improve Security Operations" by McAfee (<https://dl.acm.org/doi/abs/10.1145/3474718.3474723>)
- [12] "Automating Security Rule Generation with Machine Learning" by Palo Alto Networks (<https://www.paloaltonetworks.com/unit42/assess/ai-security-assessment>)
- [13] "Security Automation: How It Works and Why It Matters" by Varonis (<https://www.varonis.com/blog>)

Art Generated by Grok



Thank You

Anthony Tellez & Leo Meyerovich