JupiterOne

BLOG

Turn 10,000 AppSec Findings into 10 Actual Risks

You've seen it, you've been a part of it. Alert fatigue sets in with warnings coming from multiple domains: cloudsec, infrasec, netsec, data security, appsec, seceng, identity and access management (IAM), secops, vulnerability management... and a hundred other sources. Each of these security domains is complex enough on their own, but in today's cloud computing environment, all of these areas are not only **interrelated**, they are **interconnected**.

The complexity of relationships between your cyber assets is impossible to recognize in siloed systems, resulting in the creation of thousands of alerts of "possible risk", burying the real issues in meaningless reports. **The key to separating actual risk from hypothetical musings** (and actually getting remediation prioritized) **is using cyber asset relationships to provide context** around application security vulnerabilities.

Why is Appsec so challenging?

Development, engineering, and security teams frequently have a challenge answering seemingly innocent questions:

- **Who are the developers** that generated the greatest number of vulnerabilities in production?
- **Which code repos or projects** have the most 'high' vulnerabilities deployed, that are exposed to the internet (publicly accessible)?
- **Which events** trigger my Lambda functions?
- **What open code and software vulnerabilities** do I have on applications that are deployed to production infrastructure, that isn't encrypted, that exposes personally identifiable information (PII) to the internet?

As the questions get more difficult, additional complexity in your tools is required. In order to answer the question, **an organization needs to have complete understanding and visibility** of their code pipeline tools and the disparate systems that make up their infrastructure.

Aggregation of the data for analysis, investigation, and remediation is a technical challenge. **The sheer volume of findings reduces clarity** when teams need to prioritize their finite resources for corrective action.

How to achieve the reduction of noise

A cloud-native security platform providing visibility and security into your entire cyber asset universe is needed to manage the complexity of assets and relationships within your systems. Our mission at JupiterOne is to reduce the noise to deepen understanding, and help you focus on high priority risks. To achieve this, we ask questions of the data using the JupiterOne Query Language (J1QL).

What follows is a suggested process for refining a J1QL query that could potentially return tens of thousands of results. We help you narrow the results into a limited set that is easily visualized within an expandable/contractable graph.

The starting point: integrations with existing domains

JupiterOne has managed integrations for:

- **Code repositories**, e.g., GitHub, Bitbucket, Gitlab
- **Software/code scanners**, e.g., Snyk, Veracode, WhiteHat, Detectify, Checkmarx
- **Cloud service providers**, e.g., Amazon Web Services (AWS), Azure, Google

All of the above fall under what are considered "cyber assets" - any hardware or software that can be defined by code: containers, virtual machines, data stores, user accounts, identities, code repos, pull requests, laptops, scanners, agents, vulnerabilities, findings, and much more.

A simple query: Define a starting point using a severity level

Many shops have security scanning tools for different purposes - one for containers, one for software development, one for the network, and another couple for infrastructure. Logging into each of the individual tools to attempt investigation and remediation efforts is a task in itself, let alone when teams need to start managing several of these scanners that each return numerous findings. Wouldn't it be nice to see ALL of them in one dashboard?

Let's start here with a simple J1QL query:

```
find Finding with numericSeverity > 5
```

Since JupiterOne integrates with and collects findings from other vulnerability scanners, this can **return all findings from integrated security and code scanner tools**, e.g., Snyk, Qualys, Tenable, Gitleaks, etc.

Limit the query to code repos

A simple query with no qualifiers can result in a lot of noise and volume. To limit results to just code findings, let's focus our results on code repositories:

```
find Finding with numericSeverity > 5
that HAS CodeRepo
```

This cuts down the findings significantly as now we're **only looking at findings that are related to a code repository**. Notice that the query does not specify whether it is Snyk findings or Veracode or WhiteHat. That is because **JupiterOne applies a high level model to abstract the data**. This reduces the need for tuning and allows the same query to work across multiple vendors/products.

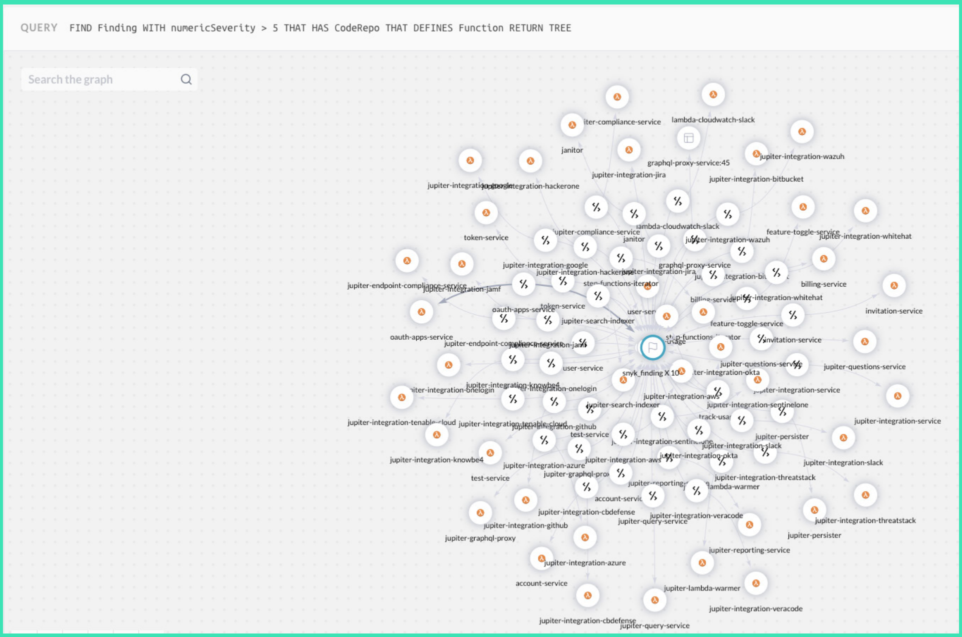
Return a graph for easy visualization

Limiting the results to those found within code repos is still likely to return too many findings to be of use. A unique answer to simplifying the reporting process is through **cyber asset and relationship graphs**. These graphs can be used to create a high level overview, with the ability to drilldown to the granular level.

We extend our query to create a graph (tree):

```
find Finding with numericSeverity > 5
  that HAS CodeRepo
  that DEFINES Function
return tree
```

We have further filtered our results to only **return findings related to code repositories that define functions**. At JupiterOne, our architecture is heavily serverless, using AWS Lambda functions. The query could **easily be modified for virtual instances, hosts, containers, clusters**, etc.



```
QUERY  FIND Finding WITH numericSeverity > 5 THAT HAS CodeRepo THAT DEFINES Function RETURN TREE
```

Refine the results set to see access roles

Refinement of this query continues by looking to specifically identify the findings related to software code changes that define our compute functions, and those that isolate individual functions that define and allow access - via access roles and access policies.

Extend our query to include access roles and policies:

```
find Finding with numericSeverity > 5
  that HAS CodeRepo
  that DEFINES Function
  that ASSIGNED AccessRole
  that ASSIGNED AccessPolicy
  that ALLOWS DataStore
return tree
```

At this point, the remaining results are software findings that define Lambda functions that have access (via AWS IAM Roles and Policies) to data stores, including S3 buckets, DynamoDB tables, RDS databases, Elasticsearch clusters, etc. This is an **example showing the benefits of the abstracted data model** - `DataStore` covers any and all data stores matching the relationships, both current and future.

```
find Finding with numericSeverity >= 7
  that HAS CodeRepo
  that DEFINES Function
  that ASSIGNED AccessRole
  that ASSIGNED AccessPolicy
  that ALLOWS DataStore
return tree
```

Final tweaks

In a small adjustment to the tuning of the query, we raise the severity from a '5' to an '7', which will expose **the high or critical findings**, only:

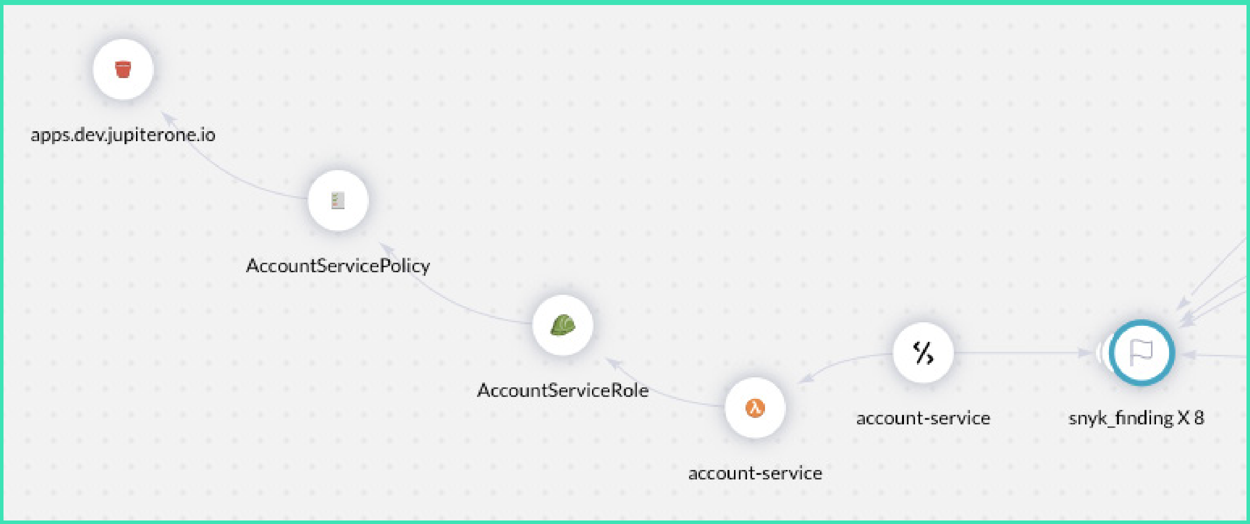
```
find Finding with numericSeverity > 8
  that HAS CodeRepo
  that DEFINES Function
  that ASSIGNED AccessRole
  that ASSIGNED AccessPolicy
  that ALLOWS DataStore with
    encrypted!=true and
    classification='PII'
return tree
```

Final modifications to the query will filter the previous results to **yield those findings that allow access to unencrypted data stores that contain 'critical' data**.

Drill down into the final result set

Clicking through the various node and relationship entities in JupiterOne's dynamic graph visualization allows **real-time traversal of the knowledge base**, to facilitate additional inquiries such as:

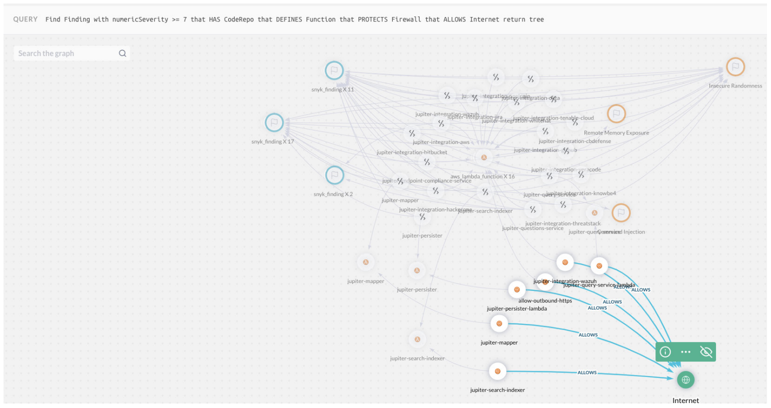
- **What else within the subnet is accessible** via the IAM role/policy assigned by the code defining the function?
- **What does the function have access to?** For example, it's triggered by a specific Amazon API Gateway.



- **Which developer** created, approved, or merged this pull request (PR)?
- **Who owns the infrastructure** in case of remediation?

Similarly, you can find vulnerable code deployed to be internet facing:

```
Find Finding with numericSeverity >= 7
  that HAS CodeRepo
  that DEFINES Function
  that PROTECTS Firewall
  that ALLOWS Internet
return tree
```



QUERY Find Finding with numericSeverity >= 7 that HAS CodeRepo that DEFINES Function that PROTECTS Firewall that ALLOWS Internet return tree

With simple tuning/modification of queries, we can start with hundreds, thousands, maybe even tens of thousands of findings across all sorts of security and code scanners, and **quickly prioritize the findings that pose the greatest risk** to our organization's crown jewel assets - in this case, customer data.

The results of a refined query: simplicity, visibility, and clarity

The seemingly impossible question we asked at the beginning, "What open code and software vulnerabilities do I have on applications that are deployed to production infrastructure, that isn't encrypted, that exposes personally identifiable information (PII) to the internet?", is just one of numerous questions developers and security teams can find the answers to with JupiterOne's graph knowledge base.

Without using JupiterOne, an application security engineer would have to reconcile a list of findings from Snyk and work with a DevOps engineer to figure out which Lambda functions are affected. These engineers may even need to loop in development or security team members to figure out which repos, PRs, and customer data stores could be affected.

What you can do right now

In the current security reporting environments where there is too much finding/alert fatigue due to excess volume and noisiness of signal, not all findings equate to risk. It's through the systematic interrogation of our cyber environments and connecting all of the disparate systems which compose what we define as an "asset", that we are able to identify true business risks and the who, what, where, when, and how to take corrective action.

We invite you to **run your own queries on JupiterOne**. Our basic platform is free. This is not a "trial" version, there is no expiration. Our hope is that you will see immediate value as you begin to surface your cyber assets and be able to determine your highest security priorities through the query refinement process.

If you have questions or comments, **I monitor our slack channel daily**, and look forward to hearing from you.

George Tang, Principal Security Architect
JupiterOne