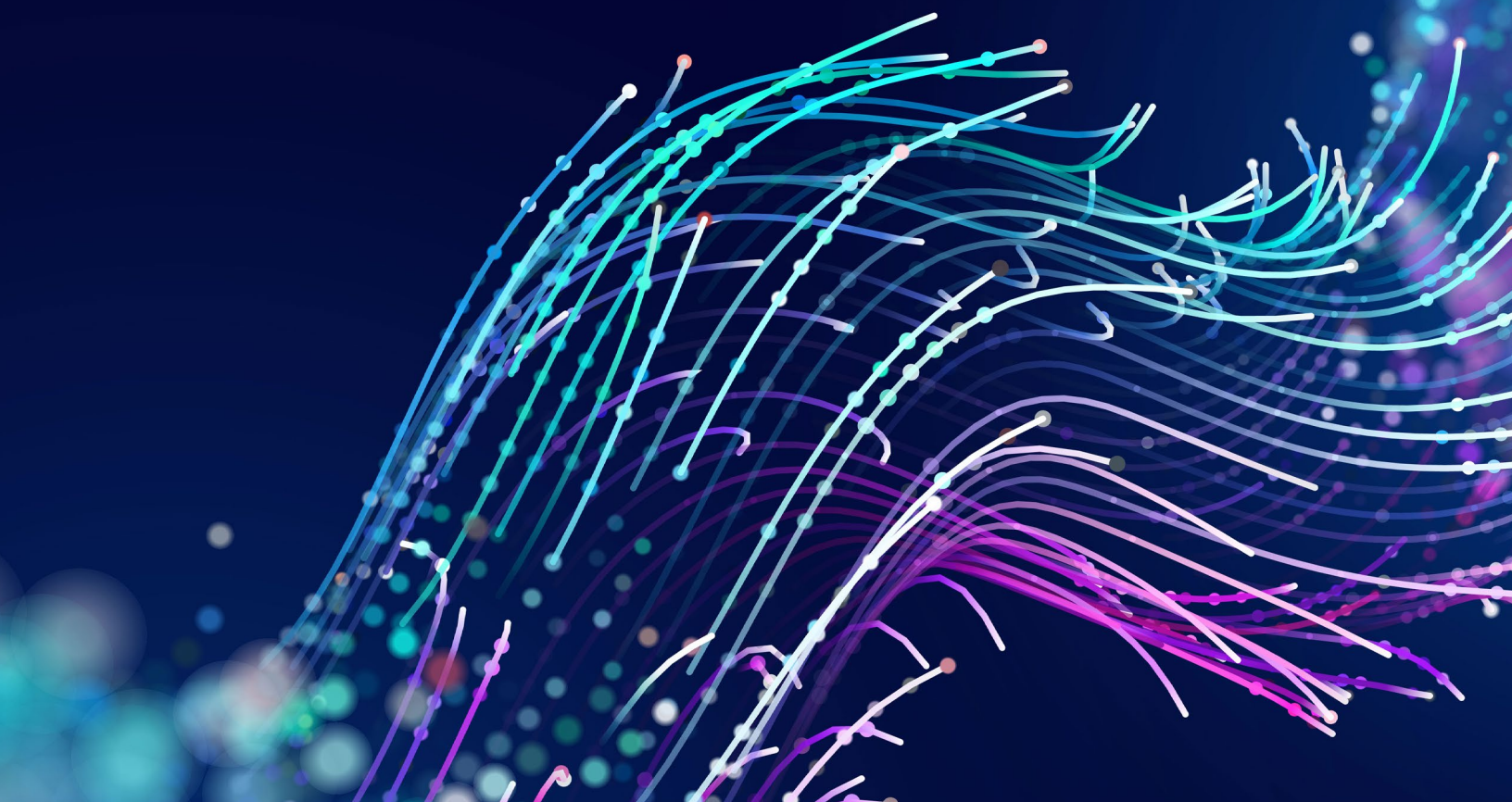
THREATQUOTIENT

Accelerate Threat Detection & Response with DataLinq Engine



Contents

3 Data & the SOC of the Future

4 Introducing DataLinq Engine

5 How the DataLinq Engine Works

11 Summary

Data & the SOC of the Future

All data is security data because data that provides the context needed to make the best decisions and take the right actions isn't limited to a few tools and feeds, it's everywhere. And harnessing all that data is problematic. No one understands this better than SOC teams battling to work smarter and faster all while facing internal challenges including staffing shortages, siloed organizations and disparate technologies, plus the ever-advancing threat.

Threat actors are becoming more sophisticated in their tactics, techniques and procedures. Advances with ransomware, thanks to the ease with which it can be monetized, plus the growing attack surface resulting from cloud, remote workers and an increasingly digital supply chain, have all yielded even more data for SOC teams to consume.

Data has perhaps never been more important to the SOC, which is why security-forward leaders recognize that a data-driven approach is the keystone for the SOC of the future.

Data has perhaps never been more important to the SOC, which is why security-forward leaders recognize that a data-driven approach is the keystone for the SOC of the future.

When security is data driven, SOC teams have the context provided by a wide range of sources including threats, vulnerabilities and identities, that enables them to focus on relevant, high priority issues, make the best decisions and take the right actions. Data-driven security also provides a continuous feedback loop that enables teams to store and use data to improve future analysis.

Data is spread throughout the typical organization, so bi-directional integrations are required to bring that data together into a common work surface, and an open integration architecture provides the best approach to do this. An open approach to data integration offers the widest access to the range of technologies, threat feeds and other third-party sources that are relevant to detection and investigation, and also enables teams to drive response back to those same technologies.

Response may take the form of machine automation or manual action. Teams have benefited by automating repetitive, low-risk, time-consuming tasks, but the need for human analysis remains. Irregular, high impact, time-sensitive investigations are best led by a human analyst with automation simply augmenting the work. A balance between manual investigations and machine automation ensures that teams always have the best tool for the job, while a data-driven approach to both improves the speed and thoroughness of the work.

The need for a data-driven approach, open integration architecture, and balanced use of automation is best when dealing with the evolving nature of attacks. As threat actors now work across the entire organization, it's critical for SOC teams to "connect the dots" across all data sources, tools and teams to accelerate threat detection and response.

Introducing DataLinq Engine

ThreatQ DataLinq Engine takes a unique approach to make sense of data in order to accelerate detection, investigation and response. The DataLinq Engine starts by enabling data in different formats and languages from different vendors and systems to work together. From there, it focuses on getting the right data to the right systems and teams at the right time to make security operations more data driven, efficient and effective.

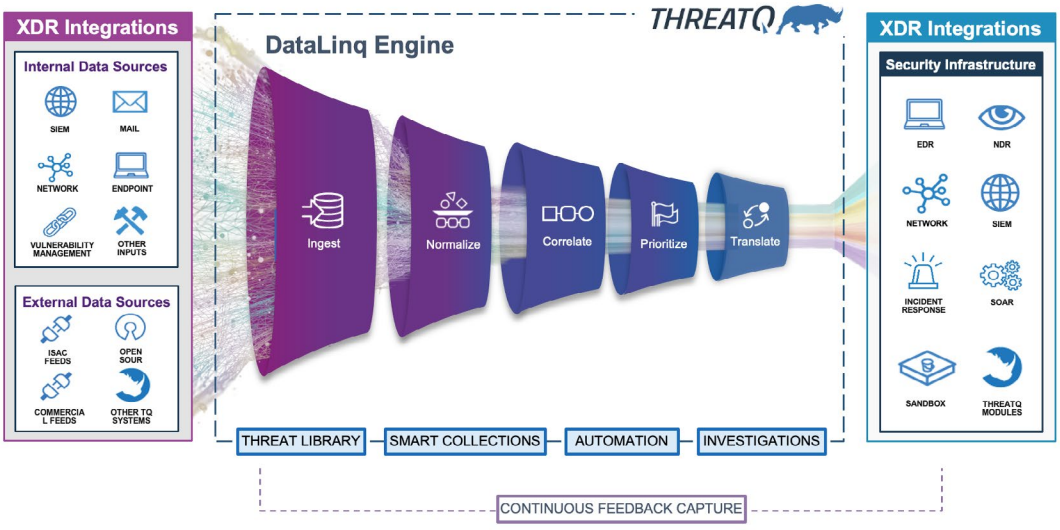
Many security problems can be remedied by focusing on the right, smaller sets of data.

It's common to hear it stated that *"Cybersecurity is a big data problem"*. This can be interpreted in a couple of ways. You can interpret the statement to mean that security problems can only be solved with big data. However, another perspective is that cybersecurity has a set of big problems, caused by the volume of data now available to teams. Many security problems can be remedied by focusing on the right, smaller sets of data.

Too much data can cause a serious impediment for organizations in terms of scale and execution. While cloud computing drastically reduces the cost of storage and processing, it ushered in a world where data proliferation is a mounting issue. With more copies of existing data being made continuously, each with minor modifications, in different locations, analysts lack of a single source of truth which causes confusion. ThreatQ DataLinq Engine focuses on augmenting key existing data stores, so that they can interoperate, reference each other, and enable cross product and data workflows that simplify how defenders approach response.

For many years, ThreatQuotient has been operating inside a diverse ecosystem of hundreds of different security products, threat intelligence feeds, data enrichment services and security operations teams. We've seen first-hand the challenges professionals have in making sense of security data in order to determine if, and how, to respond or contain a threat, or simply ignore it. To better serve our customers, we've developed the DataLinq Engine with the specific goal of optimizing the process of making sense out of data in order to reduce the unnecessary volume and resulting burden.

The DataLinq Engine follows a specific processing pipeline leading to a dynamic end-state that is constantly updating, evolving and learning. This method of processing is vastly different from a SIEM, Log manager, or legacy Threat Intelligence Platforms.



How the DataLinq Engine Works

To make sense out of data and operationalize it where required, we must first deconstruct it, and then merge it into a collective many-to-many relational model that has multiple dimensions. The engine must focus on the goal of adding more value to existing data stores and systems that exist within the operational environment rather than merely duplicating or replacing them.

The DataLinq Engine accomplishes this goal by working through five key stages:

Ingest

The first stage in processing data is to gain access to input data. The DataLinq Engine supports a wide variety of sources including both internal and external; structured and unstructured; and standard or custom (see examples at https://marketplace.threatq.com).

Parsers can be applied for industry recognized formats such as email (EML), PDFs, YARA, OpenIOC, Snort/Suricata, XML, JSON, and plain text. The goal of the ingest stage is to identify key elements within a data object that can be useful for understanding it.

Consider a hypothetical single security data record that includes some form of observation or metadata in describing a threat or incident. It may be represented as a simple JSON object:

```
{
     "observation_time": 1634208760,
     “event_id”: 1234,
     "malware_family": "emotet",
     “md5”: d8e8fca2dc0f896fd7cb4cb0031ba249,
     "connection": {
          "from": "source.example.com",
          "to": "destination.example.com"
     },
     “URL”: “http://example.com:80/some/call/back?id=1”,
     "detection type": "C2",
     "source_identity": {
          "first_name": "Joe",
          "last_name": "smith"
     ,
     “first_256_bytes_of_flow”: “xxxxxxxxxxx<omitted>”,
     “pe_header”: {
          <omitted>
     }
}
```

There is much that we could pull from this sample record that is useful for analysis, detection, investigation and response activities. However, for the scope of this paper, we'll summarize the elements within a data object into three different categories:

1. **Threat Objects:** Elements that can help to identify key objects to statefully track, and to enable pivots across tools: Examples: Malware family names, identity data, file hash, urls, IP addresses.

2. **Object Context:** Data that can enable the threat objects to be better understood. Examples: The Malware family name, the fact that c2 communications has been observed (and to where), identity information showing who has been impacted by it.

3. **Event Observation Metadata:** This is information about the occurrence of a sighting or event raised, including elements that are specific to the use case of the reporting product or system. Examples: Detailed information about file contents, network flow or event identifiers. Generally, details are related to one specific observation.

If the goal is a dataset that can enable better coordination across existing systems and help users to take actions that span across systems, then we should focus on the first two categories, since doing so will enable simplified pivoting between tools, systems and datasets via a common denominator, but with added security context. To achieve this goal, we must also normalize the data.

The third category of data, event observation metadata, always has value and it is common to need some of it, but this is classically where existing data stores (e.g., Log repositories and SIEMs) have focused. So, it's best to only take tactical elements to avoid falling into the "proliferation of datastores" trap.

Normalize

Normalization of data is a complex subject, but the scope of our needs allows us to simplify the goal. We need confidence that it is possible to identify functionally identical objects across different data sources when they are described in slightly different ways. For object values, the following must be considered with data normalization:

- Character encodings
- Whitespace
- Text case, in the context of the data being reported
- De-fang/re-fang processes (e.g., hxxp://example[.]com)
- Protocol specific elements, such as port number handling in urls (http://example.com:80 vs. http://example.com)

Normalization requirements extend beyond the object values to all the object context that is provided with them. The DataLinq Engine uses a configuration-driven approach to control how this context is normalized. Here are two small example snippets showing how timestamps are automatically normalized, and normalized mappings can be applied.

```
filters:
     - parse-json
     - get: results
     - iterate
     - filter-mapping:
          created: timestamp
          modified: timestamp
```

Figure 1: Configuration controlled parsing of an array of JSON objects, with timestamp parsing applied to created and modified keys

```
- name: Target Industry
     value: !expr value.industries or []
     published_at: !expr value.created
```

Figure 2: Iterating over an array of context in an industries key

The DataLinq Engine breaks apart these complex data structures related to events, incidents and threat intelligence into atomic elements, so it can rebuild them with an aggregated view of the data from across all the sources, tools, services and users. This is critical to allow the correlation into a unified object to occur between different reporting sources, and also to relate other unified objects.

Correlate

By ensuring a solid foundation of object values, we can also use it for automatically or manually gathering supporting information about an object from integrated products. For example, a file hash alone may involve:

- **EDR:** Does this file hash appear on any endpoints, and what are they?
- **AV/SIEM:** Are any security events reported about this hash?
- **Sandbox:** Are there malware analysis reports, and what were the results?
- **Incident/Ticketing System:** Has this hash been reported as an artifact associated with a breach/incident in the past?
- **Intelligence:** What adversaries, campaigns or malware families has this hash been attributed to, and what were the goals, motivations and methods behind them? (Including mappings to MITRE ATT&CK TTPs)

This is just the beginning of the detection and response investigation process. Consider the other objects that exist within our single event, and the benefits realized by quickly answering key questions by pulling from across the deployed defenses:

- Who is this identity?
- What are the methods behind the malware family detected?
- What's the history of that URL, or domain?
- What additional context can be pulled in to validate if the C2 traffic detection is valid?

This is where correlation comes in. As more data and context is learned, the engine consistently updates the object records which also links it to additional sources, events and tools. This correlation into a single record is essential to enable an assessment of priority or relevance on an object, not only when it's first seen, but continuously throughout its lifespan as additional information is learned about it based on the sum of all information known.

Figure 3: A visual representation of security events, broken down into objects and then presented as a correlated whole

Prioritize

Though the prioritization of data objects is automated, it is under the control of the teams using ThreatQ. Flexibility in how data is scored is key to ensure that the right threat objects are identified allowing protections to be exported automatically to the right tools. Getting the right data, to the right tools and at the right time, for the right organization has always been the goal. Once the score of an object passes a threshold, it may trigger the auto deployment of mitigations, additional analysis or responses.

Common questions addressed when prioritizing data include:

- Is this IP address related to an active campaign we're tracking?
- Is this malware known to exploit a vulnerability we're exposed to?
- Is the actor behind this attack known to target organizations in our vertical?
- Elements like these help to ensure the right objects are prioritized for action. However, before taking action, additional translation back into the appropriate language and format is needed.

Translate

The translation approach and process varies depending on the type of actions to be taken against what form of external service or product. Consider the detection of a malicious FQDN: evil.example.com. In order to have different tools take action on it, the detection data may need to be defined in very different languages:


```
{
  "feedinfo": {
    "name": "threatq",
    "display_name": "ThreatQ",
    "provider_url": "https://rhino.threatq.online",
    "summary": "This feed exports indicators from the ThreatQ platform.",
    "tech_data": "This ThreatQ feed exports FQDN, IP Address, and MD5 indicators    with an \"Active\" status.",
    "icon": "iVBORw0KGgoAQmCC<SNIP>"
  },
  "reports": [
    {
      "id": "601811634734482",
      "timestamp": 1634734482,
      "link": "https://rhino.threatq.online/indicators/60181/details",
      "title": "Indicators from ThreatQ",
      "score": 100,
      "iocs": {
        "dns": ["example.com"]
      }
    }
  ]
}
```


Figure 4: VMWare Carbon Black

```
alert udp any any -> any 53 (msg:"ThreatQ: DNS Query - example.com"; \
    content:"|01 00 00 01 00 00 00 00 00 00|"; depth:10; offset:2; \
    content:"|07|example|03|com"; nocase; fast_pattern:only; \
    classtype:bad-unknown; \
    sid:8460181; \
    rev:1;)
```

Figure 5: Cisco FirePOWER (Snort) language

Being able to automatically generate sets of detection data for consumption by detection and prevention tools, focused on the use case of that tool, but with data that is prioritized to ensure that resources are used intelligently is now possible because of DataLinq.

The Feedback Loop

Priorities, threats, campaigns and vulnerabilities are forever changing, therefore it's important to avoid considering any dataset to be dependable unless it takes this situation into account. The DataLinq Engine supports a feedback loop where it consumes context that is detected from those tools that have detection content deployed into them in addition to systems that are providing enrichment services. This feedback loop enables additional context to be learned over time as sightings can provide almost immediate updates to prioritization and scoring data.

Summary

The SOC of the future is built on data, and teams have never had greater access to it. While the data available from the various technologies, threat feeds and other third-party sources is essential for threat detection and response, teams can quickly become overwhelmed by the volume of data if care is not taken to properly manage it.

Security-forward teams are adopting a data-driven approach supported by an open integration architecture and balanced use of automation when dealing with the evolving nature of attacks. This approach best equips SOC teams to "connect the dots" across all data sources, tools and teams to accelerate threat detection and response across the entire organization.

the ThreatQ DataLinq Engine enables a data-driven approach across five key stages including ingestion, normalization, correlation, prioritization and translation. Finally, SOC teams have an efficient and effective way to make sense out of data and operationalize it where required. To learn more about how to take a strategic approach to using data in the SOC to accelerate detection, investigation and response, contact us at info@threatq.com for a demo.

ThreatQuotient improves security operations by fusing together disparate data sources, tools and teams to accelerate threat detection and response. ThreatQuotient's data-driven security operations platform helps teams prioritize, automate and collaborate on security incidents; enables more focused decision making; and maximizes limited resources by integrating existing processes and technologies into a unified workspace. The result is reduced noise, clear priority threats, and the ability to automate processes with high fidelity data. ThreatQuotient's industry leading data management, orchestration and automation capabilities support multiple use cases including incident response, threat hunting, spear phishing, alert triage and vulnerability prioritization, and can also serve as a threat intelligence platform. ThreatQuotient is headquartered in Northern Virginia with international operations based out of Europe and APAC. For more information, visit www.threatquotient.com.