Using TypeDB to Build a Cyber Threat Intelligence Knowledge Graph

London, April 2022

Tomás Andrés Sabat Stöfsel, Vaticle
tomas@vaticle.com

Dr. Srujan Kotikela, Texas A&M University
srujan.kotikela@tamuc.edu

Brett Forbes, Innovolve
brett@innovolve.ai

Abstract

Knowledge of cyber threats is a key focus in cyber security. TypeDB CTI is an open source knowledge graph to store and manage such knowledge. It enables Cyber Security Intelligence (CTI) professionals to bring together their disparate CTI information into one knowledge graph, enabling them to more easily manage such data and discover new insights about cyber threats.

In this article, we describe how to use TypeDB to represent STIX 2.1, the most widely used language and serialization format to exchange cyber threat intelligence. We cover how to leverage TypeDB's modelling constructs such as type hierarchies, nested relations, hyper relations, unique attributes, and logical inference to build this CTI knowledge graph.

1 Introduction

Cybercrime is expected to cost organisations worldwide over $10 trillion annually by 2025, up from $3 trillion in 2015. That represents an annual growth of 15% — one of the greatest transfers of economic wealth in history.

COVID-19 has only exacerbated this situation. Cyber threats have been on the rise and will continue to do so. The UN estimates that cybercrime has increased by 600% due to the pandemic, while the rate of detection has been estimated to be as low as only 0.05%.

As a consequence, there exists a massive need for innovative technologies to address the state of the industry. In particular, in order to improve our understanding of attacks and mitigate potential ones, we require novel approaches to our Cyber Threat Intelligence (CTI). CTI, as a sub-field within cybersecurity, refers to the discipline of collecting knowledge on threats and threat actors, from either technical or human indicators.

This is why we have built an open source knowledge graph to massively accelerate our understanding of CTI. Based on the widely used STIX standard, TypeDB CTI enables organisations to accurately structure their cybersecurity data and discover the right insights.

2 Problems with Current Approaches

Cybersecurity data is inherently connected. In order to obtain a cohesive picture to determine the severity of a particular threat, a cybersecurity analyst needs to refer to multiple pieces of information from a variety of tools and sources.

This becomes more difficult as most cybersecurity data is not completely standardised across vendors and tools. This is what the STIX data standard tries to solve, which has now become one of the most widely used standards within CTI. To represent the connected nature of this data, the committee behind STIX has been converting the standard into a proper graph format. Compared to tabular, or flat data, this would already help an analyst to visualise and reason with CTI data much more easily.

Furthermore, the industry suffers from a huge shortage of cybersecurity professionals. It is, therefore, essential that tools exist to help them become more effective and efficient. A threat analyst needs primarily to discover connections across entities such as threat actors, malwares, campaigns, vul-

nerabilities, etc., and having a tool that can reveal implicit knowledge is extremely valuable to making CTI much more efficient.

For these reasons, we built a Knowledge Graph for CTI with TypeDB, which you can access on Github. TypeDB is ideal to solve these problems in CTI primarily because:

1. **Data Expressivity**: TypeDB provides an expressive concept level schema that enables you to model your domain based on logical and object-oriented principles, making modelling complex data straightforward

2. **Logical Inference**: TypeDB provides native inferencing and reasoning capabilities out of the box, allowing for the discovery of facts and patterns that would otherwise be too hard to find

3 Building a Cyber Threat Intelligence Knowledge Graph

To enable a unified approach for the description of different kinds of cybersecurity data, we based the data model on the Structured Threat Information Expression (STIX™) standard. This makes it straightforward for cyber security analysts to ingest heterogeneous CTI data through a single common language that describes the data they work with.

In building the schema in TypeQL, we followed the official STIX specification as closely as possible. The core entity objects in STIX include:

1. SDOs: STIX Domain Objects (Attack Patterns, Campaigns, . . .)

2. SCOs: STIX Cyber-observable objects (Artifacts, Autonomous Systems, . . .)

3. SMOs: STIX Meta Objects (Marking Definitions, External References, . . .)

To represent these in TypeQL, first, we created one entity type that serves as the parent to all other entities in the schema — this is called stix-object. This entity has two subtypes — stix-core-object and stix-meta-object (SMOs). The former has two further subtypes — stix-domain-object (SDOs)

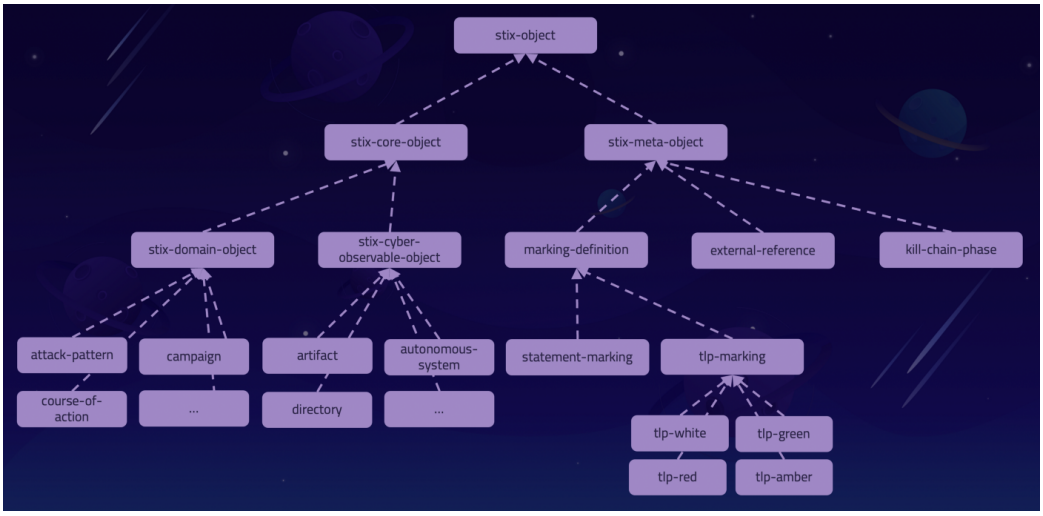

Figure 1

and stix-cyber-observable (SCOs). Modelling their remaining subtypes, gives us an entity type hierarchy that ends up looking as follows:

Written in TypeQL, this schema looks like this:

```
define
stix-object sub entity;
  stix-core-object sub stix-object;
    stix-domain-object sub stix-core-object;
    stix-cyber-observable-object sub stix-core-object;
  stix-meta-object sub stix-object;
    marking-definition sub stix-meta-object;
    statement-marking sub marking-definition;
    tlp-marking sub marking-definition;
      tlp-white sub tlp-marking;
      tlp-green sub tlp-marking;
      tlp-red sub tlp-marking;
      tlp-amber sub tlp-marking;
    external-reference sub stix-meta-object;
    kill-chain-phase sub stix-meta-object;
```

To create these mappings between STIX and TypeQL, we went through the STIX Specification and for all the the objects, relations and properties defined, we mapped them to their equivalent types in TypeDB. You can find

the full schema here.

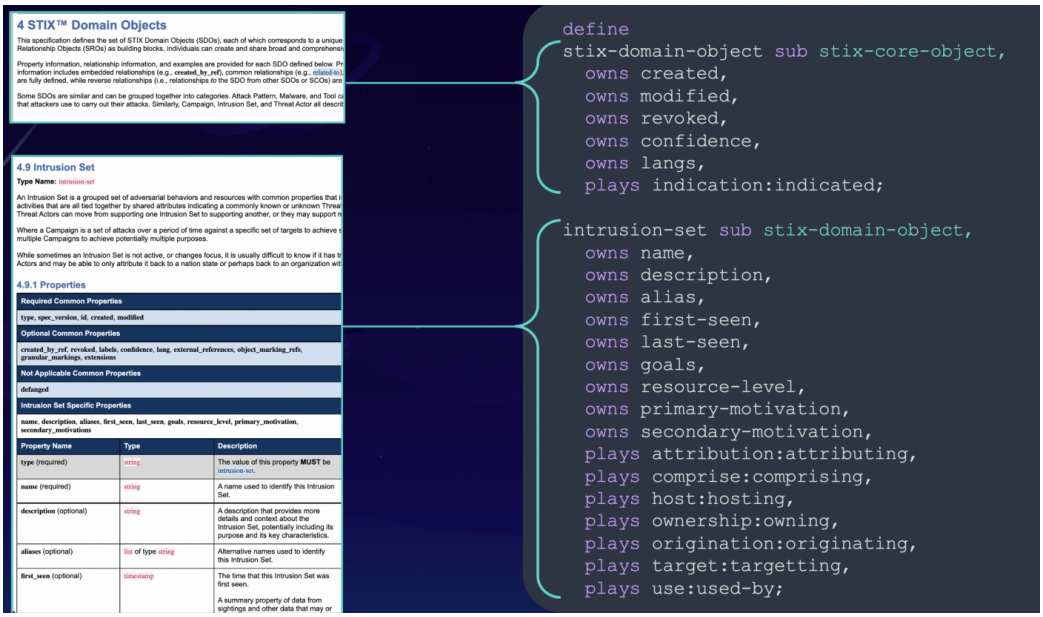

Figure 2

4 What are the Benefits of TypeDB for CTI Data?

The reason it is so straightforward to build STIX in TypeDB is because it enables data to be modelled based on logical and object-oriented principles. Concepts such as type hierarchies, nested relations and n-ary relations are just some of the ways in which TypeDB makes it easy to model complex domains such as cybersecurity data.

4.1 Type Hierarchies

TypeDB supports type hierarchies for any entities, relations and attributes. With the entity type hierarchy definitions mentioned in the previous section, we could query for every single subtype of `stix-object` by simply writing:

```
match $so isa stix-object;
```

Through type inference, TypeDB will automatically fetch entities of type `stix-object` or its subtypes, without needing to explicitly specify all the subentities in our query. On the other hand, if we only wanted to retrieve all subtypes of `stix-domain-objects`, we would simply write:

```
match $sdo isa stix-domain-object;
```

4.2 Hyper Relations

A central component of TypeDB's type system is the ability to represent hyper-relations. While in a property graph a relation is just a pair of things, a hyper-relation refers to a set of things, enabling us to natively represent n-ary relations. Below we model a ternary relation called `network-traffic-source` that connects three entities: `ipv4`, `network-traffic`, and `artifact`. This represents the network traffic which is coming from an IPV4 source, and contains an artifact as a payload (Chapter 6.12 Network Traffic Object).

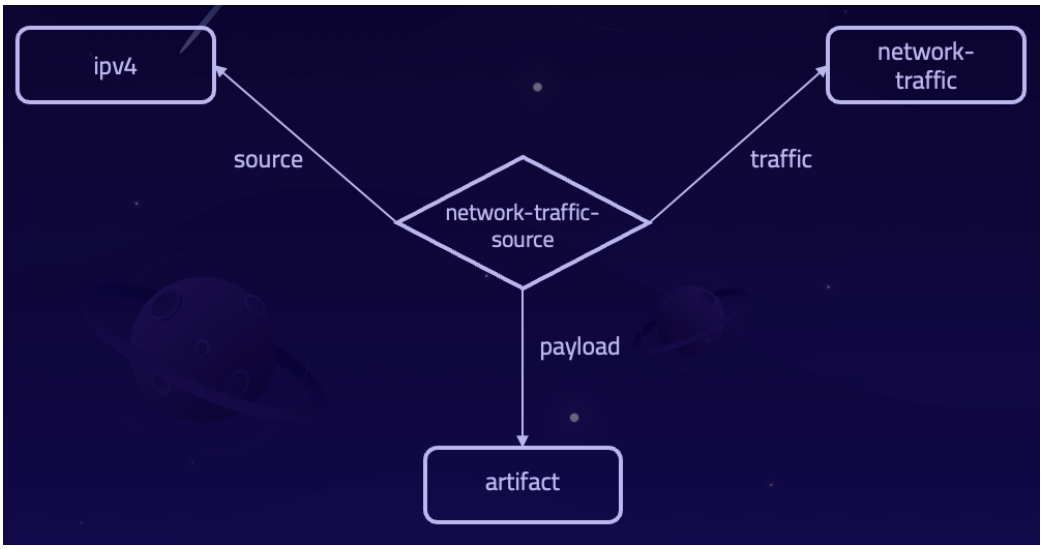

Figure 3 Squares denote entities, diamond shapes denote relations.

To create a schema that represent a ternary relation, we simply define three roles in the `network-traffic-source` relation:

```
define
artifact sub stix-cyber-observable-object,
```

```
  plays network-traffic-source:payload;
ipv4 sub stix-cyber-observable-object,
  plays network-traffic-source:source;
network-traffic sub stix-cyber-observable-object,
  plays network-traffic-source:traffic;
network-traffic-source sub relation,
  relates payload,
  relates source,
  relates traffic;
```

Then, to query for this ternary relation, we can simply assign the relevant entities (`ipv4`, `network-traffic`, and `artifact`) as variables in the relation `network-traffic-source`:

```
match
$ipv4 isa ipv4-address,
  has stix-id "ipv4-addr--e42c19c8-f9fe-5ae9-9fc8-22c398f78fb7";
$artifact isa artifact,
  has stix-id "artifact--6f437177-6e48-5cf8-9d9e-872a2bddd641";
$network-traffic isa network-traffic,
  has stix-id "network-traffic-2568d22a-8998-58eb-99ec-3c8ca74f527d";
$hyper-relation (source: $ipv4, payload: $artifact, traffic:
    $network-traffic) isa network-traffic-source;
```

4.3 Nested Relations

Given that TypeDB considers relations first-class citizens, we can also natively model nested relations. This becomes useful when, for example, we want to represent object markings (Chapter 7.2.2 on Object Markings).

In this example, we have a relation of type `indication` between two entities of types `indicator` and `threat-actor`. In order to label the `indication` relation as `tlp-red`, we create a relation of type `object-marking` between the entity `tlp-red` and the `indication` relation (see Figure 4).

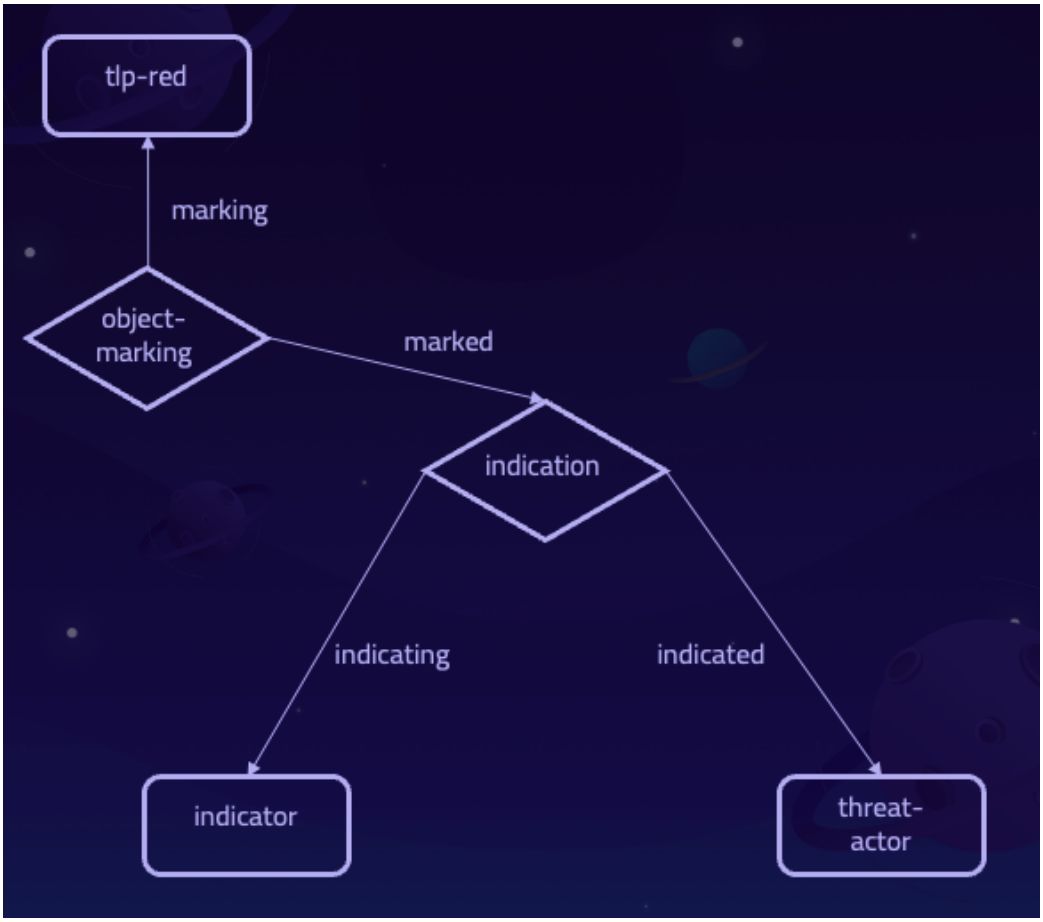

Figure 4 Squares denote entities, diamond shapes denote relations.

To write the schema for this, we declare that the relation `indication` can play the role `marked` in the `object-marking` relation:

```
define
indicator sub stix-domain-object,
  plays indication:indicating;
tlp-marking sub tlp-marking,
  plays object-marking:marking;
threat-actor sub stix-domain-object,
  plays indication:indicated;
object-marking sub relation,
  relates marked,
```

```
  relates marking;
indication sub relation,
  relates indicating,
  relates indicated,
  plays object-marking:marked;
```

We can then query for the nested relation as shown below. Note how we assign the variable `\$ind` to the `indication` relation, which allows us to let it play the role `marked` in the `object-marking` relation.

```
match
$indicator isa indicator, has name "Fake email address";
$threat-actor isa threat-actor, has name "The Joker";
$tlp isa tlp-red, has stix-id
    "marking-definition--5e57c739-391a-4eb3-b6be-7d15ca92d5ed";
$ind (indicating: $indicator, indicated: $threat-actor) isa
    indication;
$mark (marking: $tlp, marked: $ind) isa object-marking;
```

4.4 How do we Discover new CTI Insights?

TypeDB's ability to perform logical inference during query runtime enables the discovery of new insights from existing CTI data. For example, let's take the following question:

Does the "Restrict File and Directory Permissions" course of action mitigate the "BlackTech" intrusion set, and if so, how?

```
match
$course isa course-of-action, has name "Restrict File and Directory
    Permissions";
$in isa intrusion-set, has name "BlackTech";
$mit (mitigating: $course, mitigated: $in) isa mitigation;
```

This returns a relation of type `inferred-mitigation` between the two entities:

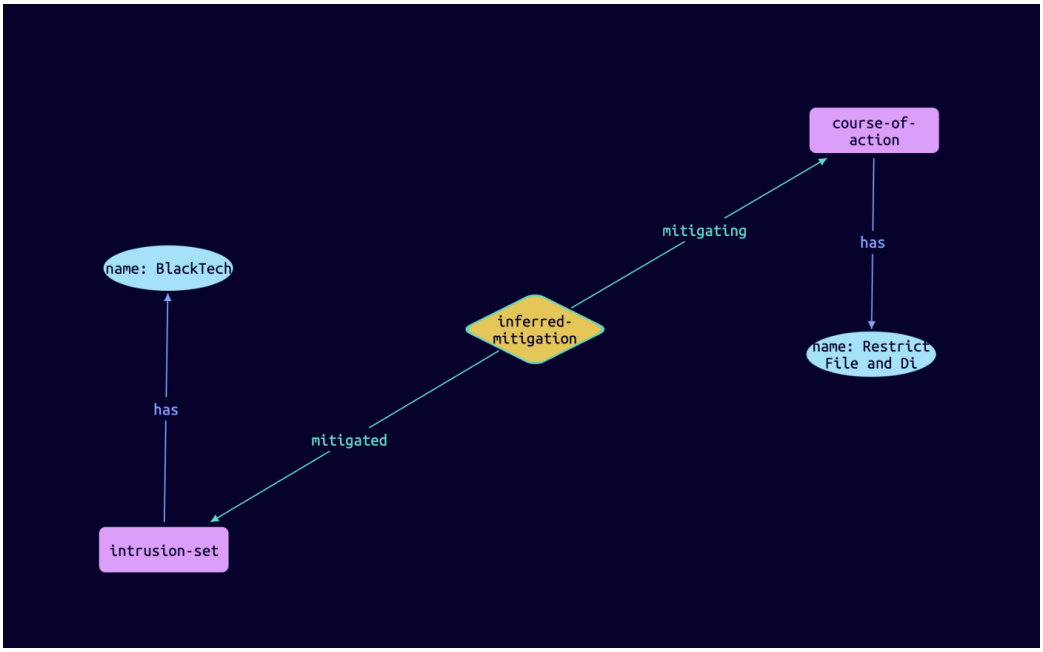

Figure 5

But the `inferred-mitigation` relation does not actually exist in the database, it was inferred at query runtime by TypeDB's reasoner. By double clicking on the inferred relation, the explanation shows that the `course-of-action` actually mitigates an `attack-pattern` with the name `Indicator Blocking`, which has a use relation with the `intrusion-set`.

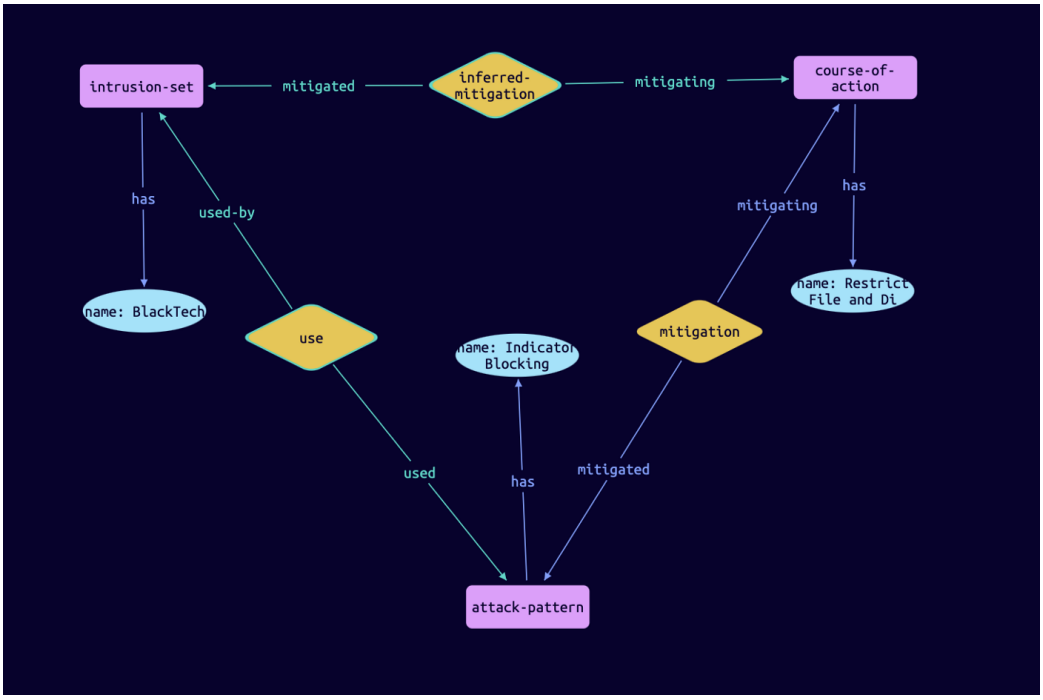

Figure 6

However, that `use` relation (between the `intrusion-set` and the `attack-pattern`) is also inferred. Double clicking on it shows that the `attack-pattern` is not directly used by the `intrusion-set`. Instead, it is used by a `malware` called `Waterbear`, which is used by the `intrusion-set`.

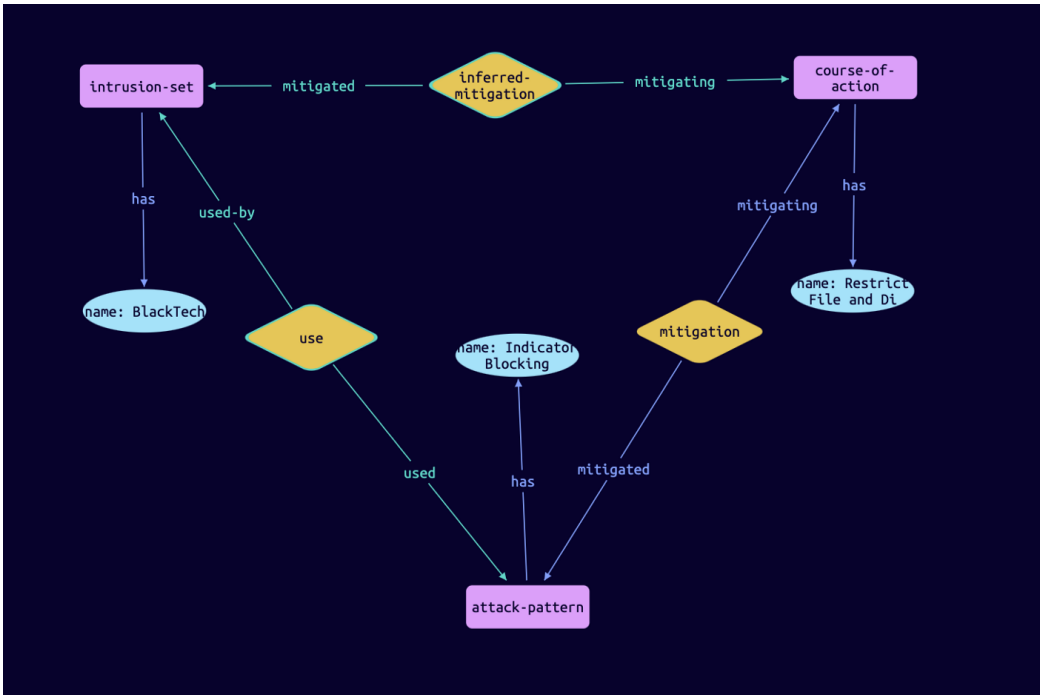

Figure 7

TypeDB is able to make these inferences when we define rules in our schema (more rules are defined here in the repo). Specifically, for the two inferred relations shown above, we use two separate rules, each inferring one relation: the first rule below infers the relation `inferred-mitigation`, while the second rule infers the relation `use`:

```
rule mitigating-course-of-action-with-intrusion-set:
when {
 $course-of-action isa course-of-action;
 $sdo isa stix-domain-object;
 $intrusion-set isa intrusion-set;
 $mitigation (mitigating: $course-of-action, mitigated: $sdo) isa
    mitigation;
 $use (used: $sdo, used-by: $intrusion-set) isa use;
} then {
 (mitigating: $course-of-action, mitigated: $intrusion-set) isa
    inferred-mitigation;
};
```

```
rule attribution-when-using:
when {
 (attributing: $x, attributed: $y) isa attribution;
 (used-by: $y, used: $z) isa use;
} then {
 (used-by: $x, used: $z) isa use;
};
```

5 Conclusions

Cybersecurity data is complex, dynamic and heterogeneous. This makes it a perfect fit for TypeDB. Its expressive schema language, that allows for concepts such as type hierarchies and hyper relations, give us a level of expressivity to model the most complex cybersecurity data as accurately as possible. And through that semantic richness, TypeDB makes it easy to discover new insights, improving prevention of cyberattacks and the understanding of our CTI.

We believe TypeDB Data — CTI can be a real game changer to push forward the cybersecurity industry. You can access it on Github, it's open source.