STIX & Semantic Equivalence

Background

One of the issues that STIX sharing communities eventually face is the sharing of intelligence that is either identical or very similar to intelligence that has already been shared within the community. While detecting purely identical intelligence is simple and can be accomplished through basic string comparison, detecting intelligence that has the same or similar underlying meaning and is therefore *semantically equivalent* to other intelligence is a more challenging proposition. Thus, the concept of detecting semantically equivalent STIX content is based around understanding the semantics of individual STIX Domain Objects (SDOs) – each object has its own unique set of properties that need to be considered when determining semantic equivalence.

As an illustration of this problem, consider the following two Indicators:


```
{
  "type": "indicator",
  "spec_version": "2.1",
  "id": "indicator--8e2e2d2b-17d4",
  "created": "2016-04-06T20:03:48.000Z",
  "modified": "2016-04-06T20:03:48.000Z",
  "indicator_types": ["malicious-activity"],
  "name": "Some Malware",
  "pattern": "[ipv4-addr:value = '203.0.113.2/31']",
  "valid_from": "2016-01-01T00:00:00Z"
}
```


Indicator 1


```
{
  "type": "indicator",
  "spec_version": "2.1",
  "id": "indicator--f3341337-0cab",
  "created": "2016-05-03T08:06:23.000Z",
  "modified": "2016-05-03T08:06:23.000Z",
  "indicator_types": ["malicious-activity"],
  "name": "Some Other Malware",
  "pattern": "[ipv4-addr:value = '203.0.113.2' OR
ipv4-addr:value = '203.0.113.3']",
  "valid_from": "2016-02-01T00:00:00Z"
}
```


Indicator 2

On the face of it, these Indicators are not identical, as they don't share any properties besides a common **indicator_types** property value. However, if one were to look at closely at their **pattern** values, which represents the actual malicious cyber activity that they're looking for, you'll see that they are in fact semantically equivalent as the "/31" CIDR block in Indicator 1 is equivalent to the two IP addresses OR'd together in Indicator 2.

Scope

Semantic equivalence in the context of STIX is a multi-faceted topic and this paper represents an initial attempt at covering some key aspects of this process. Accordingly, this paper is focused exclusively on the calculation of semantic equivalence between identical types of STIX SDOs, and does not cover or take into account equivalence between STIX Relationship Objects (SROs), equivalence between non-identical types of STIX SDOs, or the broader topic of graph-based equivalence.

Calculating Semantic Equivalence

The following section discusses the overarching concept and algorithm for calculating pair-wise semantic equivalence between STIX 2.x SDOs.

High-level Process

At a high-level, the process for calculating semantic equivalence between STIX SDOs is as follows:

1. Determine the “key” properties of primary semantic importance (i.e., those that are critical to the underlying meaning of the object) for each STIX Domain Object.
    a. Equivalence must be calculated differently for each type of STIX Domain Object.
2. Determine the set of properties for each STIX SDO that are not important with respect to its semantics and should be ignored. For example, the following common properties are not likely to be useful as they have no bearing on the overall semantics of an SDO:
    a. **id**
    b. **created_by_ref**
    c. **created/modified**
    d. **description**
3. For each pair of SDOs, calculate semantic equivalence by comparing only the values of the “key” properties found between the two SDOs and ignoring the rest.

Each of these steps are discussed in detail in the following sections.

Key SDO Properties

Captured below in Table 1 are the "key" properties of various STIX 2.1 SDOs, which play a significant role in capturing the underlying meaning of the object. Note that this concept is specific to semantic equivalence as outlined in this paper - the STIX specification has no notion of "key" properties in this sense Properties that are defined as required in the STIX specification for each SDO are highlighted in **bold**.

STIX SDO (2.1 CSD01)	Key Properties
Indicator	**indicator_types** **pattern** **valid_from**
Malware	**name** **malware_types**

Tool	**name** **tool_types**
Attack Pattern	**name** external_references
Vulnerability	**name** external_references
Campaign	**name** aliases
Threat Actor	**name** **threat_actor_types** aliases
Location	latitude longitude region country
Identity	**name** **identity_class** sectors

Table 1: Key STIX 2.1 SDO Properties

Base Algorithm

The base algorithm for calculating semantic equivalence between a pair of STIX SDOs of the same type is as follows, where $prop_1$ … $prop_n$ refer to the union of key properties found on the pair of SDOs being compared (representing the properties that will be evaluated for equivalence), the percent matching is a floating point value from 0 to 1.0 (thus supporting partial matches), and the weight is an integer value from 0 to 100:

$$\%\ \boldsymbol{equivalence} = \frac{\sum_{\boldsymbol{prop_1}}^{\boldsymbol{prop_n}} \%\ \boldsymbol{matching} * \boldsymbol{weight}}{\sum_{\boldsymbol{prop_1}}^{\boldsymbol{prop_n}} \boldsymbol{weight}} * \mathbf{100}$$

It is necessary to weight each key property, as every property does not represent the same level of importance with regards to the semantic equivalence calculation for each SDO. For example,

on the Indicator SDO, the **pattern** property is of utmost importance and should be weighted accordingly.

It's also important to note that because the key properties for each object (as defined in this paper) are not always required in the STIX specification, the sum of the weights in each comparison is not always 100. For example, take the case of comparing two Location SDOs (see table below), with each having a **region** and **country** - the sum of these weights in this case is 66.

Equivalence Score

The above algorithm calculates the percent equivalence between two SDOs. Thus, this score can fall into thresholds such as the following, with the corresponding implications. Note that this is just an example meant to illustrate how such scoring thresholds may be used, and in practice is something that is dictated by the requirements of the consumer of this data.

- 0%-10%: there's no overlap between the key properties of the two SDOs, which implies that they have no semantic relationship.
- 10%-50%: there's some overlap between the key properties of the two SDOs, which implies that they may be related but are unlikely to be semantically equivalent.
- 50%-90%: there's significant overlap between the key properties of the two SDOs, which implies that they may be semantically equivalent.
- 100%: there is complete overlap between the key properties of the two SDOs, which implies that they're highly likely to be semantically equivalent.

Example 1

As an illustration of the score generated by this algorithm, take for example the two Indicators we've previously discussed. For this example, let's assume we have the following weights for their key properties:

- **indicator_types:** 15
- **pattern:** 80
- **valid_from:** 5

In addition, there are the following property values:

Property	Indicator 1 Value	Indicator 2 Value
indicator_types	`["malicious-activity"]`	`["malicious-activity"]`
pattern	`[ipv4-addr:value = '203.0.113.2/31']`	`[ipv4-addr:value = '203.0.113.2' OR ipv4-addr:value = '203.0.113.3']`

valid_from	`2016-01-01T00:00:00Z`	`2016-02-01T00:00:00Z`

With these weights and property values, we can now calculate the respective property match scores:

- **indicator_types**: 1.0 (perfect match) * 15 (weight) = 15
- **pattern**: 1.0 (perfect match) * 80 (weight) = 80
- **valid_from**: 0 (no match) * 5 (weight) = 0

Thus, the overall semantic equivalence percentage between these two Indicators is:

$$\frac{80 + 15 + 0\ (total\ matching\ score)}{80 + 15 + 5\ (sum\ of\ weights)} * 100 = \frac{95}{100} * 100 = 95\%$$

Note that although all of the property matches in this example result in scores of 1.0 (perfect match) or 0 (no match), it is possible to have partial matches across the **indicator_types** and **pattern** properties.

Example 2
As an another example, consider two Location objects, with the following key properties and weights:

- **latitude/longitude:** 34
- **region:** 33
- **country:** 33

In addition, there are the following property values:

Property	Location 1 Value	Location 2 Value
latitude/longitude	`n/a`	`n/a`
region	`australia-new-zealand`	`australia-new-zealand`
country	`AU`	`NZ`

As seen above, neither of these SDOs contains the latitude/longitude property; accordingly, because this property is optional, we can exclude it and its corresponding weight from the equivalence calculation. We can now calculate the respective property match scores:

- **region**: 1.0 (perfect match) * 33 (weight) = 33
- **country**: 0 (no match) * 33 (weight) = 0

Thus, the overall semantic equivalence percentage between these two Indicators is:

$$\frac{33 + 0\ (total\ matching\ score)}{33 + 33\ (sum\ of\ weights)} * 100 = \frac{33}{66} * 100 = 50.0\%$$

Types of Matches

Depending on the key SDO property that is being compared, different types of matches must be calculated. For example, certain properties such as timestamp values can only match if both values are exactly the same. Other properties, such as lists of strings, can have partial matching results if there is any intersection between the list entries. The types of matches we have considered for this initial draft are outlined below in Table 2.

Match Type	Description	Possible Resulting Values
Exact	Exact match between two property values.	Boolean: 0 (no match) or 1 (match)
Partial (list-based)	Partial match, based on the intersection of the two lists (normalized on the size of the largest of the lists). For example, the following two lists would have a matching value of 0.50: List 1: ["foo", "bar", "foobar", "f00"] List 2: ["foo", "bar"] Matching value = 2 (number of intersecting values)/4 (size of the largest list) = 0.50	Float: 0.0 (no intersection) – 1.0 (complete intersection)
Partial (string-based)	Partial match, using the Jaro-Winkler algorithm[1] to calculate the distance between the two strings. Note that this approach does not account for synonyms or alternate spellings of the same word.	Float: 0.0 (no match) - 1.0 (perfect match)

[1] https://en.wikipedia.org/wiki/Jaro%E2%80%93Winkler_distance

Partial (timestamp-based)	Partial match, based on the intersection of the timestamps. **Algorithm**: convert date timestamps to seconds from the epoch (T1 and T2 respectively) and compare to the number of seconds in a day. Matching value = 1 – min((abs(T1 – T2) / 86400), 1) For example, the following two timestamps would have a matching value of 0.459 T1: Wednesday, October 31, 2018 12:25:40 PM T2: Thursday, November 1, 2018 1:25:23 AM Matching value = 1 - min((abs(1540988740-1541035523)/86400,1) = 1 - min(46783/86400,1) = 1 - 0.541 = 0.459	Float: 0.0 (no match) - 1.0 (exact match)
Partial (external-reference)	A special class of partial matching on external references (see section below).	Float: 0.0 (no match) - 1.0 (perfect match)
Custom	Custom type of match, depending on the property.	0 – 1 (depends on the property)

Table 2: Types of Matches for Calculating Equivalence

External Reference Matching

When matching on external references, if the **source_name** property of one of the external references on each SDO has a value that refers to a type defined in STIX (i.e., “capec”, “mitre-attack”, or “cve”) and if one of its **external_id** or **url** properties match the value in the external reference of the other SDO being compared, then it should be considered a perfect match, and the other external references on the SDO can be ignored.
For example, the following two external references would be considered a perfect match (1.0):

```
{
  "source_name": "capec",
  "external_id": "CAPEC-550",
  "url": "http://capec.mitre.org/550.html"
}
```

```
{
  "source_name": "capec",
  "url": "http://capec.mitre.org/550.html"
}
```

For matching on external references that don't include a STIX-defined **source_name** value, standard list-based matching should take place, with the **source_name**, **external_id**, and **url** properties being compared.

Specific Object Equivalence

The following section delves into the details of calculating semantic equivalence between specific types of STIX 2.x SDOs, including their key properties, weights, matching types, and associated corner cases.

Indicator

Key Property	Proposed Weight	Matching Type	Comments
indicator_types	15	Partial (list-based)	Straightforward list-based matching. Custom values are ignored.
pattern	80	Custom	Lots of corner cases and open questions, see below.
valid_from	5	Partial (timestamp-based)	n/a

The most difficult aspect of calculating semantic equivalence for Indicators is with respect to the **pattern** property, as this involves parsing the actual pattern expression, extracting and performing pair-wise comparisons across each of the Comparison Expressions within. Notionally, each pair of Comparison Expressions (e.g., `url:value = 'http://example.com/foo'` and `url:value = 'http://example.com/bar'`) must be compared in the context of an exact match, and with basic pattern expressions that don't make use of patterning features such as Observation Expression Qualifiers this is a fairly simple task.

However, there are also a number of corner cases that must be accounted for:

- As in the previous examples used in this paper, when comparing Comparison Expressions with a base object of type **ipv4-addr** or **ipv6-addr,** CIDR blocks need to be accounted for in terms of the IP range that they represent
- When comparing Comparison Expressions with a base object of type **file**, any intersection across the **hashes** property must be considered a full match.
    - E.g., if one Comparison Expression includes only an MD5 hash and another includes both an MD5 and a SHA-256 hash, the two Expressions can be considered a perfect match if the MD5 hash is the same in both.
- When comparing Comparison Expressions that represent object references (i.e., those properties ending in **_ref** or **_refs**), the references must point to the same type of object in order to be considered a match.
- When comparing Comparison Expressions with a base object of type **windows-registry-key**, any comparisons across the **key** and **values/name** properties must be case insensitive.
    - E.g., *HKEY_LOCAL_MACHINE/FOO/BAR* and *hkey_local_machine/foo/bar* should be considered to be the same value.

In addition to the above corner cases, there are a number of open questions on dealing with other pattern constructs with respect to semantic equivalence:

- How should Observation Expression Qualifiers such as *REPEATS* or *WITHIN* be handled with respect to pattern comparison?
    - For example, if two patterns share a Comparison Expression but have different Qualifiers, should they be considered matching at all? What if they both contain the same Observation Expression Qualifier but with different values (e.g., WITHIN 5 SECONDS vs. WITHIN 30 SECONDS).
- How should Observation Operators such as *AND* and *FOLLOWEDBY* be handled?
    - For example, if two patterns share the same Comparison Expressions but have different Observation Operators (e.g., two IP addresses that are AND'd in one pattern and are OR'd together in another), should they be considered matching at all?
- How should object references be taken into account when calculating a match across two patterns?
    - For example, if two patterns share a set of Comparison Expressions, but also have references to different objects as part of their patterns, should they be considered matching at all?
    - E.g., if one pattern has an AND between a File and a Registry Key, and another pattern has an AND between a File and a Process – if the two Files are the same, should this still constitute a match at all, since the other parts of the two patterns are completely different?

Note that the above lists are not exhaustive, and that there are likely to be other corner cases and open questions. That said, there are a few rules that can be used to simplify the comparison of patterns:

- Pair-wise comparisons only make sense to do across the same type of base object, which should limit the number of comparisons that are made.
    - E.g., an **ipv4-addr** cannot be compared with a **process**
- If there is no overlap between the Cyber Observable Objects in the patterns, no match can be made and thus the resulting value for the match can automatically be set to 0.
    - E.g., if one pattern contains only **file** expressions and the other contains only **domain-name**, no match can be made.

Malware

Key Property	Proposed Weight	Matching Type	Comments
malware_types	20	Partial (list-based)	Straightforward list-based matching. Custom values are ignored.
name	80	Partial (string-based)	n/a

The key properties for Malware in its current state are **malware_types** (the “type” of malware) and name (what the producer calls the malware).

Tool

Key Property	Proposed Weight	Matching Type	Comments
tool_types	20	Partial (list-based)	Straightforward list-based matching. Custom values are ignored.
name	80	Partial (string-based)	n/a

The key properties for Tool in its current state are **tool_types** (the “type” of tool) and **name** (what the producer calls the tool).

Attack Pattern

Key Property	Proposed Weight	Matching Type	Comments
name	30	Partial (string-based)	n/a
external_references	70	Partial (external-reference)	Special form of list-based matching for external references.

The key properties for Attack Pattern in its current state are **name** (what the producer calls the Attack Pattern) and **external_references** (any external references to the Attack Pattern, such as to an existing threat framework).

Vulnerability

Key Property	Proposed Weight	Matching Type	Comments
name	30	Exact	n/a
external_references	70	Partial (external-reference)	Special form of list-based matching for external references.

The key properties for Vulnerability in its current state are **name** (what the producer calls the Vulnerability) and **external_references** (any external references to the Attack Pattern, such as to an existing vulnerability database).

Campaign

Key Property	Proposed Weight	Matching Type	Comments
name	60	Partial (string-based)	String values should be normalized to exclude common terms (see list below). Cross-compare with **aliases** – see below.
aliases	40	Partial (list-based)	Cross-compare with **name** – see below.

The key properties for Campaign in its current state are **name** (what the producer calls the Campaign) and **aliases** (what other sources may call the Campaign).

Values of the **name** property should be normalized to exclude the following common terms (case insensitive) before comparison (otherwise, values such as “Campaign Foo” and “Campaign Bar” would be considered substantial matches):

- campaign
- operation

Additionally, since they both capture a name (or variant thereof), **name** and **aliases** should be “cross-compared” – that is, values from one property should be compared directly against values of the other. For example, the following two Campaigns would be identified as being identical using this technique:

```
{
 "type": "campaign",
 "name": "Foo Campaign",
 "aliases": ["Bar Campaign"]
}
```

```
{
 "type": "campaign",
 "name": "Bar Campaign",
 "aliases": ["Foo Campaign"]
}
```

One open question around Campaign is whether to incorporate matching against the **external_references** property as well – this could be useful as an additional means of deduplicating Campaign data, as multiple Campaigns may have different producer assigned names but could share one or more external references (e.g., to threat reports that describe the Campaign).

Threat Actor

Key Property	Proposed Weight	Matching Type	Comments
name	60	Partial (string-based)	String values should be normalized to exclude common terms (see list below). Cross-compare with **aliases** – see below.
threat_actor_types	20	Partial (list-based)	Straightforward list-based matching. Custom values are ignored.

aliases	20	Partial (list-based)	Cross-compare with **name** – see below.

The key properties for Threat Actor in its current state are **name** (what the producer calls the Threat Actor), **threat_actor_types** (the "type" of Threat Actor), and **aliases** (what other sources may call the Threat Actor).

Values of the **name** property should be normalized to exclude the following common terms (case insensitive) before comparison (otherwise, values such as "APT26" and "APT24" would be considered substantial matches):

- APT
- group
- threat group

Additionally, since they both capture a name (or variant thereof), **name** and **aliases** should be "cross-compared" – that is, values from one property should be compared directly against values of the other. For example, the following two Threat Actors would be identified as being identical using this technique:

```
{
 "type": "threat-actor",
 "name": "APT29",
 "aliases": ["The Dukes"]
}
```

```
{
 "type": "threat-actor",
 "name": "The Dukes",
 "aliases": ["APT 29"]
}
```

As with Campaign, another open question around Threat Actor is whether to incorporate matching against the **external_references** property – this could be useful as an additional means of deduplicating Threat Actor data, as multiple Threat Actors may have different producer assigned names but could share one or more external references (e.g., to threat reports that describe the Threat Actor).

Location

Key Property	Proposed Weight	Matching Type	Comments
latitude/longitude	34	Custom	See comments below.
region	33	Exact	Custom values should be ignored.
country	33	Exact	n/a

The key properties for Location in its current state are **latitude**, **longitude**, **region** (the geographic region that the location belongs to), and **country**. This SDO is unique because unlike the others described here, it has no required properties and instead requires one of either **latitude** and **longitude**, **region**, or **country**. Accordingly, because **latitude** and **longitude** must be present together, we've assigned them weights that are roughly half of the weights of the other key properties.

Matching on **latitude** and **longitude** is defined as a custom type because it's possible to calculate distances between these coordinates. Accordingly, distances between non-equivalent latitude and longitude coordinates should be calculated, in kilometers, using the Haversine formula[2] and factored into the following calculation for determining the matching score:

$$matching\ score = 1 - \frac{distance}{1000\ km}$$

Accordingly, as an example, this will yield the following scores:

Distance (km)	Matching Score
0	1.00
100	0.90
300	0.70
500	0.50
700	0.30
1000	0.00

Identity

Key Property	Proposed Weight	Matching Type	Comments
name	60	Exact	n/a
identity_class	20	Exact	n/a

[2] http://www.movable-type.co.uk/scripts/latlong.html

sectors	20	Partial (list-based)	Custom values should be ignored.

The key properties for Identity in its current state are **name**, **identity_class** (the “class” of identity), and **sectors** (the industry sectors the identity belongs to).

Use Cases

This section discusses the applications of semantic equivalence, particularly with respect to the sharing of STIX data in one or more communities.

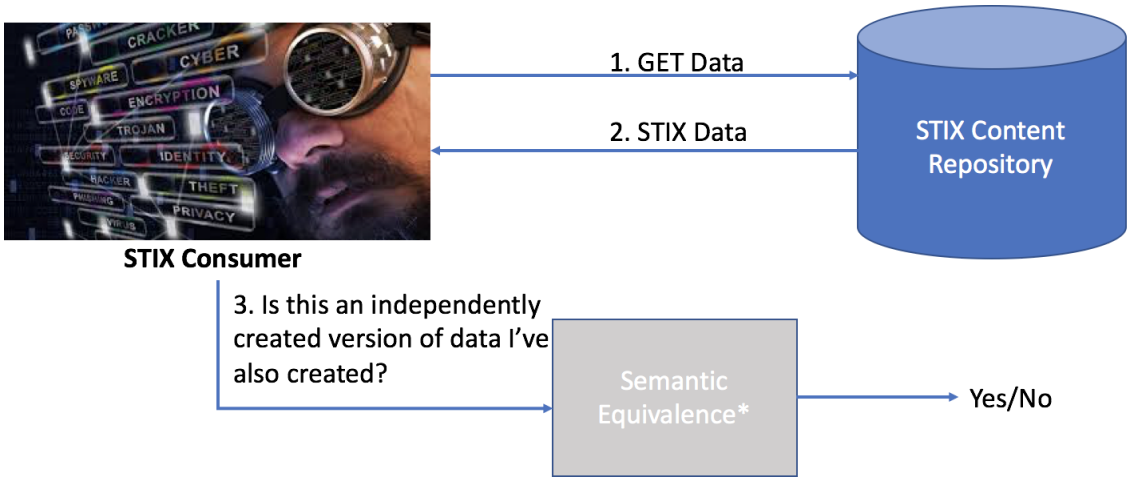
Definitions

The following concepts with respect to management of STIX SDOs are referred to throughout the use cases section, and so are defined for quick reference below.

- Merge: the action of combining two or more SDOs into a single representation by creating a new SDO that contains a union of their properties.
- Delete/deprecate: the action of deleting an SDO from a repository or data store.
- Deduplication: the action of iterating through the SDOs stored in a repository or data store and deleting any duplicates (i.e., keeping only a single unique copy of each SDO).

Echo Detection

When a STIX content producer consumes STIX data from external threat feeds and other sources, there is always the possibility that this may include data that is duplicative with respect to the content previously created by the producer. Thus, the problem of echo detection in this case revolves around understanding whether the STIX content that one is receiving is an independently submitted version of content that the consumer has previously created.



This problem can be addressed by STIX content producers by attempting to detect semantically equivalent data upon ingest or through a “garbage collection” process that runs periodically. Content that is determined to have a high level of equivalence (e.g., >=70%, depending on the type of SDO) with respect to data previously created by the producer could be handled automatically (i.e., deleted or merged into a new SDO that represents the union of the various properties in the content).

However, the challenge here (and with any similar use cases) is how to handle content that is determined to be similar but not completely equivalent (e.g., content that is determined to be 40-60% equivalent). One possibility is to mark this content and place it in a queue for human review; however, this approach is unlikely to scale given the potential volume of STIX content and is also unlikely to be a good use of an analyst’s time. Another approach is to simply reject such content, which means that any additional value it may provide would be lost. Clearly, neither approach is ideal, so some further thought on this topic is warranted.

Common Object Detection

With certain STIX SDOs, there is a higher likelihood that duplicate or near-duplicate versions of these SDOs will exist in threat feeds or sharing communities. This is particularly true of SDOs used primarily to provide supporting evidence or context that will be commonly created by STIX content producers, including the Attack Pattern, Vulnerability, and Location SDOs.

Detecting semantically equivalent STIX data can help address this problem through the identification of identical or nearly identical versions of such content. For example, consider the following scenario of three different versions of the same Vulnerability being shared:

```
{
  "type": "vulnerability",
  "spec_version": "2.1",
  "id": "vulnerability--0c7b5b88-8ff7-4a4d-
aa9d-feb398cd0061",
  "created": "2016-05-12T08:17:27.000Z",
  "modified": "2016-05-12T08:17:27.000Z",
  "name": "FooVuln",
  "external_references": [
    {
      "source_name": "cve",
      "external_id": "CVE-2016-1234"
    }
  ]
}
```

Vulnerability 1

```
{
  "type": "vulnerability",
  "spec_version": "2.1",
  "id": "vulnerability--bbb23987-b8a2-4638-
a779-4507d40c55f2",
  "created": "2016-05-14T10:33:11.000Z",
  "modified": "2016-05-14T10:33:11.000Z",
  "name": "CVE-2016-1234",
  "external_references": [
    {
      "source_name": "cve",
      "external_id": "CVE-2016-1234"
    }
  ]
}
```

Vulnerability 2

```
{
  "type": "vulnerability",
  "spec_version": "2.1",
  "id": "vulnerability--da2da90e-94d9-483f-
b600-615c8935864f",
  "created": "2016-05-09T21:03:44.000Z",
  "modified": "2016-05-09T21:03:44.000Z",
  "name": "VulnFoo",
  "external_references": [
    {
      "source_name": "cve",
      "external_id": "CVE-2016-1234"
    }
  ]
}
```

Vulnerability 3

Although each of these vulnerabilities has a different **name**, they all share the same external reference (to a CVE entry). Accordingly, using the semantic equivalence algorithm and weights previously outlined, these vulnerabilities would be calculated as being 70% equivalent. Using this calculation, a STIX content consumer could infer that these are in fact referencing the same vulnerability and use this knowledge to store and reference only a single variant of this object.

This use case can also be extended to searching across STIX content repositories for variants of the same object (assuming that the repositories don't do any merging or deduplication of such data). For instance, in the vein of the above example, one could search for variants of the same Vulnerability SDO in order to determine if there is any additional information available on the vulnerability.

Managing & Sharing Semantically Equivalent Data

STIX threat feeds, content aggregators, and other data feed providers that incorporate data from various sources will face issues and questions similar to those outlined for STIX content producers in the context of echo detection. Namely, how can they detect similar STIX data in order to keep the quality of their feed high, and what do they do with this data once detected. The mechanism (algorithm and key properties/weights) for detecting semantically equivalent STIX data has been discussed previously, so the larger question here is with regards to what such feed providers should do once they've detected some set of such data as being semantically equivalent.

One possibility is to simply capture this fact using a STIX relationship object and a custom relationship value such as "semantically-equivalent", and thereby include information about semantically SDOs in their data feed. This would also allow the capture of the actual equivalence score between semantically equivalent data, either as a custom property or through the use of the existing **confidence** property.

However, this approach raises the question of what value this provides to consumers – what do they gain by knowing that some set of objects in the data feed are semantically equivalent? On one hand, it offloads this decision process to the consumer, so that those who have the ability to do more interesting things with this data (e.g., merge together all semantically equivalent objects) can take advantage of it. On the other hand, this could just end up being extra noise that consumers will have to deal with, in terms of the extra relationships and data that they'll have to parse.

One practical approach is to send feedback of semantic equivalence only to the original content submitter. In theory they could potentially receive feedback with the ID of the equivalent indicator, which will allow data submitters to receive feedback on their submissions and curate their data in a way that makes it more relevant for sharing. This also provides the means for submitters to remove content that they understand is duplicate and not relevant.

Another possibility is for threat feeds to "garbage collect" semantically identical content by periodically checking for similarities amongst the STIX data in their collection and then either merging, deleting any copies of objects that have a high degree of similarity, or (primarily for cases of low similarity) simply keeping the objects as-is. As with echo detection, this process is highly dependent on the level of similarity found, with high and low levels of similarity being easier to deal with. For example, if two SDOs are highly similar, it may not make sense to keep

both in the data feed; conversely, SDOs that have a low level of similarity should likely both be kept. However, deciding what to do with SDOs that show some middle level of similarity is tricky; one option is to attempt to automatically merge them. The problem is that there are likely to be many corner cases with such a process, including what to do with properties that are contained in multiple SDOs but are not identical across all of the them. Again, it's clear that this warrants further investigation.

An additional possibility is for the threat feed to store all of their semantically equivalent content in a separate repository that can be queried by users as a means of data enrichment. For instance, if a user wishes to see if there exists any additional data on an SDO that they already possess, they could query this special repository to find any SDOs that may be semantically equivalent and use this information to enrich their existing data or just gain additional context.

Material Change Detection

Another usage of semantic equivalence is around the automated detection of material changes when versioning STIX SDOs. The STIX specification defines material change as a "change to the meaning of the object"; this is an important concept in the context of versioning because any changes to an SDO that are material in nature suggest that a new SDO should be created instead of updating the old one. Accordingly, when versioning an object, semantic equivalence can used as a means of detecting whether material changes have been made in the new version – if the new version and the previous version of an SDO are not highly equivalent (~>80%), then this is likely to be a strong indicator that material changes have been made and a new SDO should instead be created.

Practical Use of Semantic Equivalence

An important fact to keep in mind when discussing semantic equivalence, especially in the context of sharing platforms like AIS, is that *most* STIX 2.x content in the near future will be relatively simplistic and thus straightforward to calculate semantic equivalence for. This is particularly true for Indicators, where many of the corner cases and questions that we've outlined will not be applicable. For example, the majority of Indicators today consist of "facts" such as domain names or IP addresses, which result in very basic Indicator patterns that should be easy to compare in terms of semantic equivalence. In addition, the basic semantic equivalence algorithm is easy to apply as it operates directly on the facts that are exchanged in STIX data.

Accordingly, it would be prudent to employ at most a basic form of semantic equivalence in the near future for the purposes of producer-based echo detection and basic garbage collection in threat feeds (i.e., detecting and deprecating data that is highly equivalent). Other use cases and corner cases associated with semantic equivalence can be addressed in conjunction with the increase in complexity of the STIX 2.x content in the wild, especially when considering the shift

from exchange of more basic data such as indicators/observables to more complex/analysis-based data such as campaigns and TTPs.