STIX 2.0 Specification - Pre-Draft

STIX Core Concepts - Version 0.2

Document Table of Contents

Document Development Status

TODO - This section should be removed from the document prior to completion. It is included here to help visually track where things are at in the process.

Each physical documents contains a table that defines 4 levels of development for each TLO and CTI concept. The first level is called Concept. Content coming in to one of the documents starts as a Concept. Once the community starts to work on it it will move to Development. During this phase, the group will flesh out the design and come up with normative text. As the group comes to general consensus the TLO will move to a Review phase. During this phase the community can comment and offer suggestions on the normative text and design. After a period of time of no comments or feedback, the TLO will move to its final stage of Draft.

Object / Concept	Status	MVP
Common Types		
`controlled-vocabulary`	Review	Yes
`granular-marking`	Review	Undecided
`identifier`	Draft	Yes
`timestamp`	Draft	Yes
`timestamp-precision`	Draft	Yes
`vocab-ext`	Review	Yes
STIX Concepts		
Custom Properties	Draft	Yes
Custom Top Level Objects	Development	Yes

Open Questions:

1. How do we track serialization concerns to highlight JSON MTI while allowing alternate bindings?

2. How do we represent examples? Are they in a section that appears in the TOC or just a bolded heading?
3. Need to decide what to do about controlled vocabularies, recent discussion has led us away from the previous compromise.

1. Abstract

< to do >

2. Introduction

STIX is an information model and serialization for representing cyber threat information. The STIX information model is graph-based, with STIX Top Level Objects (TLOs) representing nodes and edges of the graph.

TODO: Second paragraph about JSON MTI and JSON Examples

STIX supports two primary activities: shared threat analysis and machine automation.

<TODO> ## probably worth having something about document organization.##

2.1. Purpose

<to do>

2.2. Requirements

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [34].

An implementation is not compliant if it fails to satisfy one or more of the MUST or REQUIRED level requirements. An implementation that satisfies all the MUST or REQUIRED level and all the SHOULD level requirements is said to be "unconditionally compliant"; one that satisfies all the MUST level requirements but not all the SHOULD level requirements is said to be "conditionally compliant."

2.3. Document Conventions

2.3.1. Font Colors and Style

The following color, font and font style conventions are used in this document:

- The `Consolas` font is used for all type names, property names and literals.
  - type names are in red with a light red background - `package`
  - property names are in bold style - **`created_at`**
  - literals are in green with a green background - `IP Watchlist`
  - as kinds of relationship are string literals, they will also appear in green with a green background - `related-to`
- In property tables if the property is being redefined from an inherited value in some way, then the background is dark grey.

All type names, property names and literals are in lower-case. Words in property names are separated with an underscore (_), while words in type names and string enumerations are separated with a dash (-).

Reserved property names are marked with a type called `RESERVED` and a description text of "RESERVED FOR FUTURE USE". Any property name that is marked as `RESERVED` **MUST NOT** be used in any implementation.

Examples are included, using the JSON MTI serialization. They are in `Consolas` 9 pt font, with black text and a light blue background. JSON examples have a 2 character space indentation.

2.4. Changes from STIX 1.x

< TO DO>

3. Terms and Definitions

Term	Definition
Top Level Object	Top level objects (TLOs) capture information about things that exist in the cybersecurity threat domain: threat actors, observations, attack patterns, malware, campaigns, etc.
ID reference	A string that contains an ID that references a different existing STIX TLO.

Object Creator	The object creator is the entity (e.g. system, organization, tool) that generates the `identifier` field for a given object. Entities (proxies) that re-publish a top-level object from another entity, maintaining the original `identifier`, are not considered the object creator. Entities (brokers) that accept objects and republish them with a new identifier are considered the object creator of the new objects.
Object Series	A top-level object series is the set of STIX TLOs with the same ID. All objects are members of a top-level object series. a top-level object series may only have a single member.
Properties	< TO DO, the fields in a TLO>
Producer	See: Object Creator
Consumer	
Object Owner	

4. Common Types

This section defines common types used throughout STIX. These types will be referenced by the "Type" column in other sections. This section defines the names and allowable values of common types that are used in the STIX information model; however does not define the meaning of any fields using these types. These types may be further restricted elsewhere in the document.

Type	Description
`array`	Contains an ordered sequence of values. Often, the phrasing `array` of type `<type>` is used to indicate that all values within the array must conform to a specific type.
`boolean`	Contains a value of true or false.
`controlled-vocab`	Contains a value from a STIX-default controlled vocabulary.
`granular-marking`	Contains a field-level data marking.
`identifier`	Contains an identifier (ID) for a STIX top-level object.

`number`	Contains a number.
`string`	Contains text.
`timestamp`	Contains a timestamp (date and time).
`timestamp-precision`	Contains a precision value for timestamps.
`vocab-ext`	Contains a value for a field in an extended vocabulary.

4.1. Array

An `array` contains an ordered sequence of values. When the phrasing `array` of type `<type>` is used, all values in the array **MUST** be of the specified type. When a type is not specified, values in the array **MAY** be of any type. Note that this section does not specify an upper limit on the number of values present in an array.

The JSON MTI serialization uses the JSON array type, which is an ordered list of zero or more values. Binding specifications for other serializations **MUST** support ordered lists for this type.

4.2. 4.2.Boolean

A `boolean` contains a value of true and false. Fields with this type **MUST** have a value of `true` or `false`.

The JSON MTI serialization uses the JSON boolean type, which is a literal (unquoted) true or false. Binding specifications for other serializations **MAY** use other mechanisms for capturing these values.

4.3. Controlled Vocabulary

	Status: Review **MVP**: Yes

All vocabularies in the CTI specifications are either "*Uncontrolled*" or "*Controlled*". Each vocabulary also supports an extension point to support additional / external vocabularies. The key name for the extension point (of `vocab-ext` type) is **`[vocabulary_field_name]_ext`**. The vocabulary fields also support the ability to have a fallback value. Using the value of `other` in the main field indicates no fallback value.

Uncontrolled - This type represents a vocabulary that is not yet defined in the specification, but may be defined in a future version. This type means you can you can use any string value you want or use the extension point to specify a value from your own controlled vocabulary.

Controlled - This type represents a vocabulary that is defined in the specification. You can use this controlled vocabulary or use the extension point to specify a value from your own controlled vocabulary.

4.3.1. Examples

In this example the field *cti_type* is an uncontrolled vocabulary, which means you can use any string value you want

```
{
  ...,
  "cti_type": "foo bar"
  ...
}
```

In this example the field *cti_type* is a controlled vocabulary, where you use something from the defined vocabulary in the specification.

```
{
  ...,
  "cti_type": "malware",
  ...
}
```

In this example the field *cti_type* is a controlled vocabulary (example is the same for an uncontrolled vocabulary), however, you want to use your own controlled vocabulary with no fallback / default value. Using the value of `other` in the main field indicates no fallback value.

```
{
  ...,
  "cti_type": "other",
  "cti_type_ext": {
    "value": "malware type foo",
    "vocab": "my name or url to my super cool vocab"
  },
  ...
}
```

In this example the field *cti_type* is a controlled vocabulary (example is the same for an uncontrolled vocabulary), however, you want to use your own controlled vocabulary with a fallback / default value from the defined vocabulary.

```
{
  ...,
  "cti_type": "malware",
  "cti_type_ext": {
```

```
    "value": "malware type foo",
    "vocab": "my name or url to my super cool vocab"
  },
  ...
}
```

In this example the field *cti_type* is a controlled vocabulary (not valid for an uncontrolled vocabulary), however, you want to use some arbitrary string value that is not part of the defined vocabulary or any other vocabulary with no fallback / default value. Note the use of the value `other` in this case.

```
{
  ...,
  "cti_type": "other",
  "cti_type_ext": {
    "value": "malware type foo"
  },
  ...
}
```

In this example the field cti_type is a controlled vocabulary (not valid for an uncontrolled vocabulary), however, you want to use some arbitrary string value that is not part of the defined vocabulary or any other vocabulary and you want to add a fallback / default value from the defined vocabulary.

```
{
  ...,
  "cti_type": "malware",
  "cti_type_ext": {
    "value": "malware type foo"
  },
  ...
}
```

4.4. Granular Marking

Type Name: `granular-marking`	**Status:** Review **MVP**: Undecided

This section defines the `granular-marking` type. Granular markings are used within STIX to allow producers to mark individual fields and data points within a package or top-level object.

The consumer must evaluate each selector in the **`content_selectors`** list and apply each marking in the **`marking_refs`** list to that content. More information and normative usage is available in <todo ref to data marking section number>.

Property Name	Type	Description
content_selectors (required)	array of type string	A list of selectors for content within this object. The markings referenced in the **content_selectors** field are applied to the selected content. In the JSON MTI specification, this is a list of JSONPath statements. The context root that these statements must be applied against is the object that contains the granular-marking list that this is contained in (i.e. for a package, the package, and for a top-level object, that object).
marking_refs (required)	array of type identifier	The list of markings that apply to the fields selected by **content_selectors**.

4.5. Identifier

Type Name: identifier	**Status:** Draft **MVP**: Yes

An identifier uniquely identifies a STIX top-level object. Identifiers **MUST** follow the form [object-type]--[UUIDv4], where **[object-type]** is the exact value from the type field of the object being identified or referenced and **[uuid]** is an RFC 4122 compliant Version 4 UUID. The **uuid** field **MUST** be generated according to the algorithm(s) defined in RFC 4122, Section 4.4 (Version 4 UUID).

4.5.1. Examples

```
{
  "type": "indicator",
  "id": "indicator--e2e1a340-4415-4ba8-9671-f7343fbf0836",
  ...
}
```

4.6. Number

A number contains any number that can be expressed as a decimal (e.g., -10, 0, 10, 10.1, 10.123213).

In the JSON MTI serialization, numbers are represented by the JSON number type.

4.7. String

The `string` data type represents arbitrary-length text strings, including Unicode characters.

The JSON MTI serialization uses the JSON string type.

4.8. Timestamp

Type Name: `timestamp`	**Status:** Draft **MVP**: Yes

This section defines the `timestamp` type. All discrete timestamps (i.e. not time ranges or relative times) in the CTI specifications are made up of two fields: the timestamp field itself, containing the time, and an optional field that indicates the precision of the timestamp.

Fields with a type of timestamp have two parts: a mandatory timestamp field and an optional precision field. The timestamp field MUST be present on all fields with a type of timestamp. The precision field MAY be present on all fields with a type of timestamp.

4.8.1. Format

```
YYYY-MM-DDTHH:mm:ss.ssssssZ
```

- The timestamp field **MUST** be a valid RFC 3339-formatted timestamp.
- The timestamp **MUST** be represented in the UTC timezone and **MUST** use the 'Z' designation to indicate this.
- The optional precision field, if present, **MUST** a value from the `timestamp-precision` enumeration.
    - The default value for the precision field is "second", so omitting the field is equivalent to explicitly specifying "second".
    - A value of "second" indicates that the value in the timestamp field is precise to the number of digits in the seconds value (including any fractional seconds, such as milliseconds or microseconds).
    - A value of “minute”, “hour”, “day”, “month”, or “year” indicates that the timestamp value is precise to that as a lower bound (the precision window is the timestamp value plus one unit of the precision value). For example, if the timestamp value is 2016-04-25T13:00:00Z and the precision value is "hour", the time is greater than or equal to 2016-04-25T13:00:00Z and less than 2016-04-25T14:00:00Z.
    - When specifying a precision other than "second", the time portion of the timestamp field **MUST** contain zeroes for all fields beyond the specified precision

while the date portion MUST contain "01" for all fields beyond the specified precision.
        - *For example, if the precision field is "month", the timestamp field must contain "01" for the day field and "00" for the hour, minute, and second fields such as 2016-12-01T00:00:00Z.*
- The timestamp precision field is always nested at the same level as the timestamp field.
- The property name for the precision field is **`[timestamp_field_name]_precision`**.
    - *For example, if the key of the timestamp field is "created_at", the key of the precision field is "created_at_precision".*

4.8.2. Examples

A timestamp know only to a year would look like:

```
{
  "timestamp": "2016-01-01T00:00:00Z",
  "timestamp_precision": "year"
}
```

A timestamp known only to an hour would look like:

```
{
  "timestamp": "2016-01-20T12:00:00Z",
  "timestamp_precision": "hour"
}
```

A timestamp known to a second would look like:

```
{
  "timestamp": "2016-01-20T12:31:12Z"
}
```

A timestamp known to 5 digit sub second precision would look like:

```
{
  "timestamp": "2016-01-20T12:31:12.12345Z",
}
```

4.9. Timestamp Precision

Type Name: `timestamp-precision`	**Status:** Draft **MVP**: Yes

A `timestamp-precision` represents the precision options for a given timestamp. Its value **MUST** be one of "year", "month", "day", "hour", "minute", or "second".

4.10. Vocabulary Extension

Type Name: `vocab-ext`	**Status:** Review **MVP**: Yes

<enter description>

Property Name	Type	Description
value (required)	`string`	Arbitrary value or value from an alternate vocab.
vocab (optional)	`string`	Name or location of alternate vocab

4.10.1. Examples

```
{
  "type": "indicator",
  "labels": ["other"],
  "labels_ext": [{
     "value": "malware type foo",
     "vocab": "my name or url to my super cool vocab"
  }]
}

{
  "type": "indicator",
  "labels": ["other"]
  "labels_ext": [{
     "value": "malware type foo"
  }]
}
```

5. Customizing STIX

5.1. Custom Properties

	Status: Draft **MVP**: Yes

The authors of this specification recognize that there will be cases where certain information exchanges can be improved by adding fields that are not specified nor reserved in this document; these fields are called **Custom Properties**. This section provides guidance and requirements for how producers can use Custom Properties and how consumers should interpret them in order to extend STIX in an interoperable manner.

5.1.1. Requirements

- Producers **MAY** create any number of Custom Properties in a STIX TLO.
- Custom Properties **SHOULD** start with "x_" followed by a source unique identifier (like a domain name), an underscore and then the name. For example: **`x_examplecom_customfield`**.
- Custom Properties **MUST** be uniquely named when produced by the same source and **SHOULD** use a consistent namespace prefix (e.g., a domain name).
- Rules for processing Custom Properties **SHOULD** be well defined and accessible to any consumer that would be reasonably expected to parse them.
- Custom Properties **SHOULD** only be used when there is no existing field defined by the STIX specification that fulfills that need.

Any consumer that receives a STIX document with one or more Custom Properties **MAY**:

- process the properties in the manner intended by the producer, if known
- refuse to process the document further
- silently ignore non-understood properties and continue processing the document

Producers of STIX documents that contain Custom Properties **SHOULD** be well aware of the variability of consumer behavior depending on whether or not the consumer understands the Custom Properties present in a STIX TLO. Custom Properties that are not prefixed with "x_" may be used in a future version of the specification for a different meaning. If compatibility with future versions of this specification is required, the "x_" prefix **MUST** be used.

The reporting and logging of errors originating from the processing of Custom Properties depends heavily on the technology used to transport the STIX document and is therefore not covered in this specification.

Consumers that receive a STIX TLO that contains one or more Custom Properties that are understood **MUST** process the Custom Properties according to the rules for that Custom Property.

5.1.2. Examples

```
{
  ...,
  "x_acmeinc_scoring": {
```

```
    "impact": "high",
    "probability": "low"
  },
  ...
}
```

5.2. Custom Top Level Objects

	Status: Development **MVP**: Yes

The authors of this specification recognize that there will be cases where certain information exchanges can be improved by adding Top Level Objects that are not specified nor reserved in this document; these objects are called **Custom Top Level Objects**. This section provides guidance and requirements for how producers can use Custom Top Level Objects and how consumers should interpret them in order to extend STIX in an interoperable manner.

Open Questions:

1. Do we want to require producers that are going to use a Custom Top Level Objects to also include a special manifest TLO object that lists out all of the Custom Top Level Objects that they are using? Kind of like a custom TLO object inventory? This could look like:

```
{
  "type": "custom-tlo-objects",
  "id": "custom-44298a74-ba52-4f0c-87a3-1824e67d7fad",
  "custom-tlo-objects": [
    "x_example_vendor-TLO1",
    "x_example_vendor-TLO2",
    "x_example_vendor_TLO3"
  ]
}
```

5.2.1. Requirements

- Producers **MAY** create any number of Custom Top Level Objects in a STIX document.
- Custom Top Level Objects **MUST** contain at least a `type` field and an `id` field in accordance with the standard STIX TLO design.
- The value of the `type` field in a Custom Top Level Objects **SHOULD** start with “x_” followed by a source unique identifier (like a domain name), an underscore and then the name. For example: **`x_examplecom_customobject`**.
- The value of the `id` field in a Custom Top Level Objects **SHOULD** use the same identifier type that the rest of the STIX TLOs uses. Namely, name--uuid

- Custom Top Level Objects **MAY** use all of the STIX TLO Common Properties not just the required `type` and `id` properties.
- Custom Top Level Objects **MUST** be uniquely named when produced by the same source and **SHOULD** use a consistent namespace prefix (e.g., a domain name).
- Rules for processing Custom Top Level Objects **SHOULD** be well defined and accessible to any consumer that would be reasonably expected to parse them.
- Custom Top Level Objects **SHOULD** only be used when there is no existing TLO defined by the STIX specification that fulfills that need.

Any consumer that receives a STIX document with one or more Custom Top Level Objects **MAY**:

- process the TLO in the manner intended by the producer, if known
- refuse to process the document further
- silently ignore non-understood TLOs and continue processing the document

Producers of STIX documents that contain Custom Top Level Objects **SHOULD** be well aware of the variability of consumer behavior depending on whether or not the consumer understands the Custom Top Level Objects present in the STIX document. Custom Top Level Objects that are not prefixed with “x_” may be used in a future version of the specification for a different meaning. If compatibility with future versions of this specification is required, the “x_” prefix **MUST** be used.

The reporting and logging of errors originating from the processing of Custom Top Level Objects depends heavily on the technology used to transport the STIX document and is therefore not covered in this specification.

Consumers that receive a STIX document that contains one or more Custom Top Level Objects that are understood **MUST** process the Custom Top Level Objects according to the rules for that Custom Top Level Object.

5.2.2. Example

<TODO>