STIX TC Extension Definition Policy

Version 1.1

Abstract: This document aims to provide guidance and defines policy for creating and approving extension definitions to the STIX 2.1 specification.

1 Definitions Used in this Document

Top-level properties - all STIX objects have top-level properties. These are expressed in JSON by the keys of the object. Not all properties appear at the top-level, because top-level properties can have as their value a JSON object. See the **`body_multipart`** property of the `email-message` SCO for an example.

Top-level extension properties - the extension definition facility allows property extensions to appear as top-level properties when using the `toplevel-property-extension` extension_type. See the third example in Section 6.1.1.

Subtype extensions - this is the term that will be used in this document for the predefined extensions currently in the specification. See the `file` SCO as an example. In addition, certain proposed extension definitions will have a similar purpose - i.e., as a subtype of an existing or new object extension. The term subtype extensions will be used to describe these also. See Section 6.1.2 for an example.

Extension definition syntax - the syntax used when an object uses an extension definition. The keys of the **`extensions`** property will be extension definition ids. See the first example in Section 6.1.1.

Regular syntax - the syntax for objects as used in the specification. It will not contain any extension definition ids as keys of the object in the **`extensions`** property. See the second example in Section 6.1.1.

Common Object Repository (COR) - A common repository of STIX objects available to the community managed by the CTI-TC. See ttps://github.com/oasis-open/cti-stix-common-objects

See section 7.3 of the STIX 2.1 specification [STIX 2.1] for the definitions of other terms related to extension definitions.

2 Extension Definition Proposal Categories

This document defines four categories of extension proposals to allow organizations or individuals to define extensions to the STIX specification. An extension proposal **MUST** use an extension definition, as defined in section 7.3 of the STIX 2.1 specification. Any custom properties or objects using the deprecated method of customizing STIX (see section 11 of the STIX 2.1 specification) **MUST** be converted to use the extension definitions to be included in any of the categories defined in this section.

2.1 Open

Extension definitions that are available globally are called "Open." The criteria for inclusion in the Common Object Repository is determined by the TC, but they will be minimally vetted (review document, syntactic, but not the semantics or correctness) by the repository maintainer. The expectation is that the extension definition is in a usable form, although it may not be complete. It **SHOULD** adhere to the normative text in Section 3 as much as possible. Alternative repositories **MAY** be used, which **MAY** have their own requirements, although it is **RECOMMENDED** to use the Common Object Repository.

Extension definitions at this proposal category can be elevated to the Specification Candidate proposal category with approval of the TC (see section 2.4). Such Extension Definitions stored in alternative repositories **SHOULD** be moved to the Common Object Repository. Whether to delete it from the alternative repository is left to the proposer.

The TC reserves the authority to require any changes to the extension definitions that will be incorporated in the Common Object Repository.

2.2 External

Certain extension definitions are based on frameworks and standards developed outside of the STIX Technical Committee (TC) which need to be expressible in STIX. Examples are TLP 2.0 data markings [TLP 2.0] and the ATT&CK framework [ATT&CK]. Their development does not adhere to the STIX release schedule – they have one of their own. The property extensions or new STIX object types they introduce may never be officially part of STIX. These extension definitions can be thought of as another type of extension called "External." Any changes would be under the control of the external developers and will always be expressed using extension definition syntax unless they are incorporated into the STIX specification.

The TC **MAY** advise the use of an external extension definition. In other words, it is preferred by the TC, but not part of the specification. External extension definitions need not be created by the developers of the framework or standard. When an external extension definition is considered for inclusion in the Common Object Repository (see Section 4), the TC has the final determination concerning the contents. External extensions definitions **MAY** not be available in the Common Object Repository in order to avoid it possibly being out of sync with a new version of the definition.

If an External extension definition is present in the common object repository it needs to meet all of the requirements of an Open extension definition.

2.3 Specification Candidate

Extension definition proposals in this category have the full support of the TC. Approval of a proposal is based on the expectation that the extension definition will become an official part of

a future version of the STIX specification. Different extension implementations of the same concepts MAY exist. For an extension definition to be considered a specification candidate it **MUST** be approved as such by the TC. Once it is approved further work is under the direction of the TC. The TC can be consulted before the commencement of work on a proposal, but that does not imply it is a specification candidate. Depending on the proposal, a mini-working group **MAY** be formed to discuss and create or update the contents of the extension definition. Work on the extension definition **SHOULD** follow the specification candidate lifecycle workflow as documented in Section 5. The advancement in the life cycle will include specific criteria under the control of the TC. Before being approved by the CTI TC, in the life cycle the extension definition **MUST** adhere to the normative language in Section 3. The work on a Specification candidate **SHOULD** take place in the COR.

It is a best practice that extension definitions to be considered as specification candidates use the `property-extension` extension type rather than the `toplevel-property-extension` extension type. This should help avoid property name collisions prior to integration into a future specification. When an extension is approved, any name collisions **MUST** be resolved.

The `toplevel-property-extension` extension type might be appropriate when converting deprecated custom properties to the extension definition facility, or for open vocabulary extension definitions (see Section 3.6).

2.4 Other

Extension definitions can be created by any user of STIX.

There are many reasons to create an extension definition:

- A short term need
- Sharing only within a trust group
- Experiments

Extension definition proposals in this category do not have to follow the normative text in this document. In other words, the guidance and procedures described here need not be strictly followed when creating an extension definition, as long as it is compliant with Section 7.3 of the specification.

Other extension definitions are those that **MAY** not correspond to the normative text of the other three categories and are not generally publically available.

2.5 Proposal Maturity

The four proposal categories should not be thought of as representing a level of maturity of the proposals. An Other extension definition might be fully mature and adhere to the normative language (but not shared), whereas a specification candidate extension definition might just be an idea for an extension that the TC is supporting.

3 Contents of an Extension Definition

Except where noted, the following **MUST** be followed for Specification Candidate extension proposals, **SHOULD** be followed for Open extension proposals, and **MAY** be followed for Other extension definitions.

3.1 JSON Schema

A JSON schema **MUST** exist and be available to validate any content that uses the extension definition. It must adhere to the following:

- It **MUST** use the JSON schema property ‘additional_properties’ or 'unevaluatedProperties' for all object definitions, and the value **MUST** be False.
- Deprecated custom properties are not allowed.
- To adhere to the requirement for Lists as stated in the STIX specification, all array definitions **MUST** contain the property ‘minItems’ which **MUST** be set to 1.

Any JSON schemas developed by an external specification committee (e.g., an existing standard) which are then referenced by "$ref" properties in the JSON schema for the extension definition **SHOULD** adhere to the previous bulleted normative statements.

3.2 Extension Definition Object

An extension definition object **MUST** be created. It **SHOULD** conform to the specification in Section 7.3 of the STIX 2.1 specification.

Additional constraints are as follows:

- The schema property **MUST** be a URL that links directly to the root of a directory tree that contains the JSON schema defining the syntax and semantics of the extension definition.
- The **`external_references`** property **MAY** contain a URL that links to the location of the human-readable documentation (see Section 3.3) for the extension definition. However, the preferred place for it is in the same root directory.
- The **`created_by_ref`** property **SHOULD** point to an identity object that **SHOULD** be available in the common object repository and includes contact information about the individual or group that is responsible for the definition.

As stated in the specification, Extensions Definition objects cannot themselves be extended.

The object itself **SHOULD** be stored in the Common Object Repository (see Section 4), unless it is an Other extension definition. Alternative repositories **MAY** be created by others.

3.3 Documentation

Documentation in the style of the STIX 2.1 OASIS specification (see Section 1.1 of the specification) **MUST** be created. The document can be created using Asciidoc, MS Word, Google Docs, Markdown, or a similar product. It **MUST** be available at the URL in the `external_references` of the extension definition, if a URL is provided.

3.4 Examples

Examples of objects using the extension **MUST** be provided. They **SHOULD** cover some of the common use cases for the extension definition. They **MUST** be validated by the JSON schema. They **MUST** be available at the URL in the `external_references` property of the extension definition, if a URL is provided.

3.5 API Implementation

A Python API implementation has been provided for the STIX 2.1 specification by the CTI TC. APIs using other languages are allowed and encouraged for those who cannot use the provided Python API.

An extension definition **SHOULD** be complemented with an API implementation, preferably using Python, following the style of the Python API for the STIX 2.1 specification.

The repository https://github.com/oasis-open/cti-python-stix2-extensions should be used to store the API implementation. Each extension should exist in its own directory. Tests for both legal and illegal instances of the extension should be provided.

If an API implementation is created and written using Python, it **SHOULD** be available via PyPi.

3.6 Guidance for Creating a STIX Extension Definition JSON Schema

The following guidance assumes that the extension definition will reside in the STIX Common Object Repository. The example is based on an extension definition used by the STIX elevator to support a STIX 1.x Indicator. See Appendix A for the JSON schema of this extension definition.

1. The base schemas [STIX 2.1 Schemas] **SHOULD** be used (see items 4 and 6 below). Also consult them for examples on how to write JSON schemas, in general.

2. The main schema file **MUST** be named using the extension definition id.

3. The type of an open vocabulary property should be **string**. For documentation purposes, the schema **SHOULD** include the list of values of the open vocabulary using the **enum** keyword. See Section 3.6 for details.

4. When using known STIX data types, reference the base JSON schema using a URL to the base schema file. For example:

```
"create_date_time": {
        "$ref": "http://raw.githubusercontent.com/oasis-open/cti-stix2-json-schemas/stix2.1/schemas/common/timestamp.json",
}
```

5. For consistency, version draft/2020-12 of the JSON schema **SHOULD** be used and the header should resemble the one below:

```
        "$id":
"https://raw.githubusercontent.com/oasis-open/cti-stix-common-objects/main/extension-definition-specifications/extension-definition--7c8ca481-f0e9-4389-94f5-90df472eb01d.json",
        "$schema": "https://json-schema.org/draft/2020-12/schema"",
```

   If other versions of the JSON schema are used, they may not be consistent with the STIX validator.

6. It is a best practice that the schema of an extension definition for property extensions of an existing STIX object, is split between two files.

   a. The first schema file, which is named using the extension definition id, should contain the syntax for the JSON that is an id/value pair that is contained in the **`extensions`** property. The top level of the schema should use an **allOf** clause.

      i. If defining a property extension, the first part of the **allOf** clause should contain the URL to the schema of the object that is being extended from the base schema. For example: If extending the indicator SDO, the following should be the first part of the **allOf** clause:

```
{
    "$ref": "http://raw.githubusercontent.com/oasis-open/cti-stix2-json-schemas/stix2.1/schemas/sdos/indicator.json"
}
```

      ii. If defining a new SCO, the first part of the **allOf** clause should contain the URL to the schema of the common properties of SCOs from the base schemas.

```
{
    "$ref":
"http://raw.githubusercontent.com/oasis-open/cti-stix2-json-schemas/stix2.1/schemas/common/cyber-observerable-core.json"
}
```

iii. If defining a new SDO or SRO, the first part of the **allOf** clause should contain the URL to the schema of the common properties of SDO/SROs from the base schemas.

```
{
    "$ref": "http://raw.githubusercontent.com/oasis-open/cti-stix2-json-schemas/stix2.1/schemas/common/core.json"
}
```

iv. The second part of the **allOf** clause should define the **`extensions`** property for this extension definition. It should include one property, **`extensions`**, which itself contains one property - the extension definition id. The extension definition id property should use the **$ref** keyword, which should refer to the second schema file that contains the properties of the extension definition. The name of the second schema file should be suggestive of what it contains. Notice that the **`extensions`** property **MUST** be required.

For example:

```
{
    "properties": {
      "extensions": {
        "type": "object",
        "properties": {
          "extension-definition--7c8ca481-f0e9-4389-94f5-90df472eb01d": {
            "type": "object",
            "$ref": "stix1x-indicator-.json"
          }
        }
      }
    },
    "required": [
      "extensions"
    ]
}
```

b. The second schema file should define the content of the extension definition. Notice that the first property defined **MUST** be **extension_type**. It **MUST** be defined using a single enumeration value from the `extension-type-enum` type. For example:

```
"extension_type": {
   "type": "string",
   "enum": [
       "property-extension"
   ]
}
```

c. The **additionalProperties** keyword should be set to **false**.

d. Alternatively, if a property extension definition implies extra conditions on the extended object type (e.g., certain base properties are deprecated and should not be used), then the second schema file should redefine the whole object type, not just the properties of the extension.

3.7 Open Vocabulary and Enumeration Extension Definitions

STIX uses the concept of "open vocabularies" for various properties. An open vocabulary is one where there is a defined suggested list of values as part of the STIX specification, but other values are permitted to be used.

For instance, the **`implementation_languages`** property of the Malware object contains a list of the implementation languages used in the malware. Suggested values come from the `implementation-language-ov` open vocabulary. Because there are always new languages being defined, this list could quickly become "stale". However, because this property has values from an open vocabulary, values not listed in the `implementation-language-ov` open vocabulary can be used and are still valid with respect to the specification.

The JSON schema rule for such properties simply enforces that the value(s) is a string. As a practice, the list of suggested values is included in the JSON schema for documentation purposes, using the enum keyword. However, reading the specification of such properties more closely, it states that values "SHOULD" come from the open-vocabulary. A STIX validator **MAY** issue warnings whenever "SHOULD" normative text is not adhered to. This is useful when the value is a typo of a suggested value and therefore not a suggested value from the open vocabulary.

Continuing with the `implementation-language-ov` open vocabulary example, at a certain point, newly defined languages will be commonly specified for this property, and the warnings would be undesirable and somewhat irrelevant. Including new values in an open vocabulary should be possible using an extension definition to avoid the warning, however, this is not currently implemented in the OASIS STIX validator.

Notice that the specification (see Section 7.3) explicitly prohibits adding values to enumeration types. It states:

> This extension mechanism **MUST NOT** be used to redefine existing standardized objects or properties.

An enumeration is tied to the exact definition of a property which states it **MUST** be one of the enumeration values. Therefore, adding a value using an extension would be in violation of this normative statement.

Because the STIX 2.1 specification defined the `extension-type-enum` type as an enumeration (see Section 10.5 of the specification), no additional type value is permitted. This implies that we need to use one of the existing extension type values to support the addition of a new value to an open vocabulary. `toplevel-property-extension` was chosen because the open vocabulary property exists at the top level.

```
{
    "id": "extension-definition--320740a0-26cd-4347-9020-a951d5d3ce29",
```

```
    "type": "extension-definition",
    "spec_version": "2.1",
    "name": "Additional values for implementation-language-ov",
    "description": "This extension adds the value 'zig' to the open vocabulary",
    "created": "2022-02-20T09:16:08.989000Z",
    "modified": "2022-02-20T09:16:08.989000Z",
    "created_by_ref": "identity--11b76a96-5d2b-45e0-8a5a-f6994f370731",
    "schema":"https://github.com/oasis-open/cti-stix-common-objects/tree/master/extension-definition-specifications/implementation-language-ov/additional-values.json",
    "version": "1.1",
    "extension_types": [ "toplevel-property-extension" ]
}
```

The documentation in the json schema would redefine the `implementation-language-ov` open vocabulary to include "zig".

```
"implementation-language-ov": {
      "type": "string",
      "enum": [
        "applescript",
        "bash",
        "c",
        "c++",
        "c#",
        "go",
        "java",
        "javascript",
        "lua",
        "objective-c",
        "perl",
        "php",
        "powershell",
        "python",
        "ruby",
        "rust"
        "scala",
        "swift",
        "typescript",
        "visual-basic",
        "x86-32",
        "X86-64",
        "zig"
      ]
    }
```

Here is an example of the extension definition in use:

```
{
         "type": "malware",
         "spec_version": "2.1",
         "id": "malware--0c7b5b88-8ff7-4a4d-aa9d-feb398cd0061",
         "created": "2016-05-12T08:17:27.000Z",
         "extensions": {
             “extension-definition---320740a0-26cd-4347-9020-a951d5d3ce29" : {
                "extension_type": "toplevel-property-extension",
              }
         }
         "implementation_languages": [
```

```
            "zig"
        ]
        "modified": "2016-05-12T08:17:27.000Z",
        "name": "zig ransomware",
        "description": "ransomware implemented in zig",
        "malware_types": ["ransomware"],
        "is_family": false
}
```

3.8 Validation

Producing or consuming content expressed via an extension definition is optional, as noted in the Conformance section (see Section 12.3.3) of the STIX 2.1 specification.

The rest of this section assumes that a producer or consumer wishes to support and validate content from one or more extension definitions. A producer/consumer does not need to support content for all extension definitions, and is free to ignore those in which they have no interest. It is also possible to process such content without validating it.

How content specified using Extension Definitions is stored by consumers is beyond the scope of this document.

Assuming that an extension definition will be shared with the community (i.e., it is not an Other extension definition), a uniform way to validate its use is desirable and described in this section.

The following method to validate STIX data that contains extension definitions is based on the implementation of the STIX validator application maintained by OASIS (see https://github.com/oasis-open/cti-stix-validator). Any other validator implementation **SHOULD** adhere to the algorithm described in this section.

1. The JSON schema for the extension definition **MAY** be available locally. Any schemas locally stored **MUST** be placed in a location that is known to the STIX validator.
2. If the JSON schema is not locally available, it **SHOULD** be accessible using the value of schema url property of the extension definition. The STIX validator **MAY** directly obtain the json schema using the schema URL of the extension definition. The extension definition may be found in a local store, or in a known TAXII server.
3. If the extension definition object is not found the STIX validator **MAY** be provided with URLs of repositories or other TAXII servers to search to discover it, or **MAY** be found at locations known to the trust group.
4. The base schema **MUST** validate, by default, any use of an extension definition for which the schema is not available.
5. Because a schema file **MUST** be named using the Extension Definition's `id` property, the validator can identify which schema needs to be used to validate the use of the extension definition.
6. For any extension definition used, for property extensions, the validator **MUST** be valid in both the base JSON schemas, and the one associated with the extension definition id.

As the base schema will accept any use of an extension definition, the schema associated with the extension definition **MUST** only accept valid uses of the extension definition.

7. The validator **MAY** report which JSON schema was used to validate a use of an extension definition.

4 OASIS Github Repositories for Extensions

4.1 Maintaining The Repositories

The TC assigns individuals who maintain the repository. Policies for maintaining the repository are set by the TC.

4.2 Inclusion of a Proposal

The inclusion of an extension definition into the repository will be based on the Git workflow. Individuals or groups interested in making a proposal should start by creating a fork of the repository.

For an Open proposal, once the definition is fairly complete, a pull request should be created so the repository maintainers can determine whether it should be included in the main branch of the repository. The repository maintainer will send the TC an email asking if there are any objections to inclusion. Appeals to objections are handled by the TC.

A Specification Candidate proposal can start as an Open proposal, or simply a request to the TC to begin work on the extension definition. Because this extension definition will be supported by the TC and will probably be included in a future version of the STIX specification, an email should be sent asking if there are any objections to this change of status. A fork of the repository should be made for work to begin, if not already in existence. Because this will be a work-in-progress, from time to time a pull request **MAY** be made to update the main branch.

4.4 Licenses, Intellectual Property, CLAs…

If an extension proposal is added to the Common Object Repository, it must be done under the rules and directives set up by OASIS for making contributions to the OASIS Open repositories.

This implies that the individual (and possibly the entity the individual is doing the work under) must sign a Contributor License Agreement (CLA) (see https://cla-assistant.io/oasis-open/Open-Repo-admin). Contributing an extension proposal to the repository does not change the intellectual property rights of the contributor, but makes the proposal available for use by others under the license agreement (https://github.com/oasis-open/cti-stix-common-objects/blob/main/LICENSE.md).

A contributed extension proposal can be used under any rules stated in the license, which allows for inclusion in future versions of the STIX specification. If an individual or entity does not want their extension proposal to be included in the specification, they should not contribute it to the Common Object Repository, but make it available elsewhere. Contributing an extension proposal to the repository does not imply it will be included in a future version. The TC must approve the inclusion.

4.5 Common Object Repository

The TC supports the maintenance of the Common Object Repository at https://github.com/oasis-open/cti-stix-common-objects.

The manner in which a proposal is included in this repository will depend upon the category of the proposal as discussed in Section 2.

4.5.1 Status Within the Repository

Each extension proposal will have a status within the repository. The status enumeration is described in Section 5, and includes such terms as "open", "started", "complete", "archived", etc. The status will change based on the lifecycle workflow (see Figure 1 in Section 5). The status of a proposal informs users of the extension definition of the degree of vetting and refinement that has gone into the proposal. The status of each extension definition will be available to all users, preferably at the repository's home page.

5 Specification Candidate Lifecycle

The TC defines a formal process for shepherding extension definition proposals into future releases of the STIX specification. Using this process allows the TC to keep track of the various proposals, avoiding unnecessary duplication of work and supporting the individual or group responsible for the proposal.

As stated above, not all extension definition proposals will become part of the STIX specification. Trust groups might have their own use cases for extension definitions that are not general enough for inclusion in the specification. External extension definitions will not usually be part of the STIX specification. Additionally, alternative extension definitions for the same STIX object type might be desirable. For instance, an extension definition for Incident has been created to support the conversion of STIX 1.x content to STIX 2.x by the STIX elevator (see ref). It is based on the STIX 1.x definition of an Incident, which is different then the extension definition for Incident that is under development with the awareness of the TC (see ref). The

STIX 1.x related Incident extension definition will never be part of any future specification, although others might be.

Here are the steps and ground rules of the specification candidate lifecycle:

1. A member of the TC requests to work on an extension proposal. This may be a proposal that is already under development, and perhaps already available in the Common Object Repository (and would be in the "open" status), or simply a request to start work on a extension concept (e.g., a new STIX object type, or additional properties to an existing object to support a particular use case). The proposal will then be in the "submitted" status.

   A proposal request **SHOULD** be submitted to the TC via the CTI email list, or be presented at a TC meeting. TC approval can be as simple as there being no objections made within a certain time frame (or on the TC call), or a formal vote. The proposal will then be in the "accepted" status.

2. The proposers **MAY** request a mini-working group to help develop the proposal. The mini-working group is open to anyone who is a member of the TC, without exception.

3. A leader(s) of the mini-working group **SHOULD** be chosen.

4. A fork of the Common Object Repository should be created by the mini-working group for all content of the extension definition. The proposal will then be in the "started" status.

5. A proposal **SHOULD** be developed using Asciidoc, if possible, and kept in the fork. Suggested changes can be made via pull requests. The document should be viewable by anyone, however only TC members can contribute suggestions and changes.

6. The artifacts in the fork **SHOULD** adhere as closely to the Extension Definition Object specification (as described in Section 3) as possible.

7. From time to time, assuming the artifacts adhere to Section 3, the current state of the fork can be merged into the main (master) branch, as long as there are no objections from the TC. This can be viewed as providing a beta version of the extension definition to the community. The proposal will then be in the "beta" status. Extension proposals in the beta status **MUST** adhere to the normative language in Section 3.

8. Once the TC initiates development of a working document for the next version of the specification, the TC will determine which, if any, Specification Candidates should be included. Candidate proposals need not be complete to be included.

9. The editors of the specification will incorporate any approved extension definition's documentation into the working document, using the guidance in Section 6. The proposal will be in the "approved" status. The approved status only signifies that the proposal has been included in the working document, not that it will necessarily be part of the committee specification document.

10. The proposers can decide to stop development of an extension definition at any time. The mini-working group (if one exists) can then choose to put the proposal in "archived" status (if no mini-working group exists, then the proposers may “archive” the proposal). An accepted extension definition's status can be changed to "archived" only by approval of the TC.

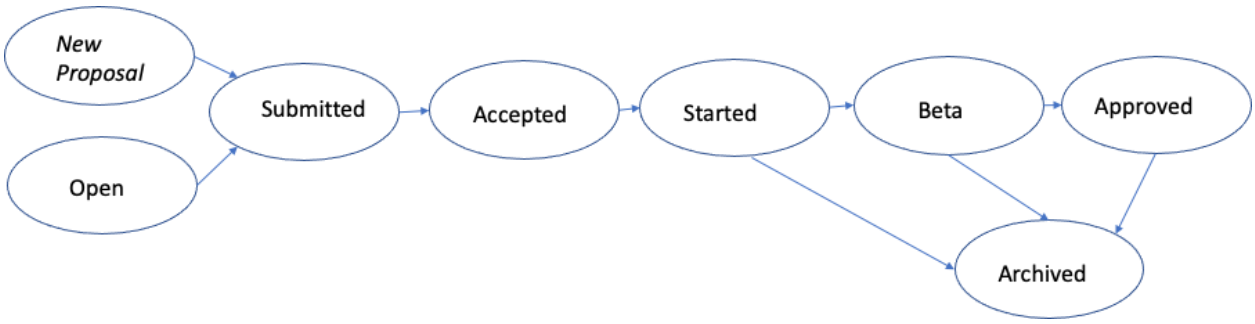

Figure 1: Lifecycle of an Extension Definition Controlled by the TC

6 Incorporating an Approved Extension Definition into a Future Specification

Once the TC has approved the inclusion of an extension definition into the specification's working document the editors **SHOULD** publish a new version of the working document with the extension definition expressed using standard syntax. Note that approving the inclusion of an extension definition does not ensure that it will be part of a future version of the specification, but only that it is included in the current working document.

The example extension definitions in this section are for expository purposes only. They may or may not be proposed extension definitions.

6.1 Incorporating New Property Extension Definitions

This section describes the various ways in which extension definitions are incorporated into the specification.

6.1.1 Using Regular Syntax

If an extension definition is to be incorporated into a future release, the current implication is that its extended properties syntax would be re-expressed as regular syntax.

Here is an example extending the vulnerability SDO:

```
{
    "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",
    "created_by_ref": "identity--f431f809-377b-45e0-aa1c-6a4751cae5ff",
    "name": "CVE-2016-1234",
    "external_references": [
        {
            "source_name": "cve",
            "external_id": "CVE-2016-1234"
        }
    ],
    "extensions": {
        "extension-definition--4b5a2e3b-1ce9-41d9-9af7-77590a0dd93b ": {
             "extension_type": "property-extension",
             "cvss_score": "CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H",
        }
      }

 }
```

Although using the `toplevel-property-extension` extension_type is not viewed as a best practice, there is no restriction of its use, and in certain circumstances it may be a better choice.

This is an example of using the `toplevel-property-extension` extension type for an extension of the Vulnerability object

```
{
    "type": "vulnerability",
    "spec_version": "2.2",
    "id": "vulnerability--0c7b5b88-8ff7-4a4d-aa9d-feb398cd0061",
    "created": "2016-05-12T08:17:27.000Z",
    "modified": "2016-05-12T08:17:27.000Z",
    "cvss_score": "CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H",
    "created_by_ref": "identity--f431f809-377b-45e0-aa1c-6a4751cae5ff",
    "name": "CVE-2016-1234",
    "external_references": [
        {
            "source_name": "cve",
            "external_id": "CVE-2016-1234"
        }
    ]
    "extensions": {
        "extension-definition--4b5a2e3b-1ce9-41d9-9af7-77590a0dd93b ": {
             "extension_type": "toplevel-property-extension"
        }
```

```
        }
}
```

The **cvss_score** property of a Vulnerability can be a property of every Vulnerability, so it makes sense to add it as a top-level property in the next STIX release. Regardless of the choice of the extension type, it would look the same when incorporated into the specification using the regular syntax.

```
{
    "type": "vulnerability",
    "spec_version": "2.2",
    "id": "vulnerability--0c7b5b88-8ff7-4a4d-aa9d-feb398cd0061",
    "created": "2016-05-12T08:17:27.000Z",
    "modified": "2016-05-12T08:17:27.000Z",
    "cvss_score": "CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H",
    "created_by_ref": "identity--f431f809-377b-45e0-aa1c-6a4751cae5ff",
    "name": "CVE-2016-1234",
    "external_references": [
        {
            "source_name": "cve",
            "external_id": "CVE-2016-1234"
        }
    ]
}
```

6.1.2 Using "new" Subtype Extensions

Some subtype extensions have already been defined in the 2.1 specification. Consider the File SCO. The STIX 2.1 specification defines five predefined object extensions to the File SCO (`ntfs-ext`, `raster-image-ext`, `pdf-ext`, `archive-ext`, `windows-pebinary-ext`). Each extension can be thought of as a "subclass" of the File SCO, as previously discussed in Section 1.

An alternative to expressing extension properties as top-level properties when an extension definition is incorporated into a future release of the specification is to package the extension properties as one more subtype extension.

If an extension definition defines a subclass of an existing STIX object, instead of re-expressing extension properties as top-level properties, it might make more sense to create a new subtype extension instead.

For instance, continuing with the File SCO object, assume we add an extension definition for a symbolic link file. Here is an example of its use:

```
{
    "type": "file",
    "spec_version": "2.1",
    "id": "file--a951e0c7-2232-4246-9072-be17fe7e3130",
    "name": "a-file-link.c":
    "extensions": {
          "extension-definition--e916f1a5-0121-4e38-a3e6-604486bbaf6c": {
                "extension_type": "property-extension",
```

```
                "linked_file_ref: "file--5890b0bb-94a6-4476-b3b4-b9153f73cc50"
            }
        }
    }
```

Assuming this extension definition is approved by the TC for the next release, it might make sense to introduce a new subtype extension instead of making the **`linked_file_ref`** property a top-level property of the File SCO.

Once approved for inclusion, the extension definition id would be replaced with the name of the new subtype extension - `link-ext`:

```
{
    "type": "file",
    "spec_version": "2.2",
    "id": "file--a951e0c7-2232-4246-9072-be17fe7e3130",
    "name": "a-file-link.c":
    "extensions": {
        "link-ext": {
           "linked_file_ref: "file--5890b0bb-94a6-4476-b3b4-b9153f73cc50"
       }
    }
}
```

6.1.3 Using SubType Extensions as part of an Object Extension Definition

When new STIX object types are defined via an extension definition, they may also have subtypes that should be expressed as subtype extensions. As an example, consider a new SCO for network devices. There are common properties among the various devices (e.g., Routers, Switches, Firewalls), but they all have other properties that are specific to each of them. It would be desirable to use subtype extensions when defining new SCOs.

Here is an example of what this would look like as an extension specification candidate:

```
{
    "type": "network-device",
    "spec_version": "2.1",
    "id": "network-device--b443fcb9-e106-4d10-a5ed-a8f3ff57b328",
    "mac_addr_ref": "mac-addr–85ab2b71-7f61-4aa1-a218-1ecea24daf01",
    "manufacturer_ref": "identity–45458637-9cb4-4811-bee8-7f92bedbeca5",
    "model_number": "WQ234",
    "extensions": {
      "extension-definition--9c59fd79-4215-4ba2-920d-3e4f320e1e62" : {
           "extension_type" : "new-sco",
           "extensions": {
                "router-ext": { … }
           }
       }
    }
}

{
    "type": "network-device",
    "spec_version": "2.1",
```

```
    "id": "network-device--90001f29-5ea8-4c97-8d8a-ea7282720fc8",
    "mac_addr_ref": "mac-addr–f1a306d1-4969-4653-8f72-13e3a1234583"
    "manufacturer_ref": "identity–1c5506e9-a436-4b0e-aa1c-1825ff6de4cd",
    "model_number": "BT-103",
    "extensions": {
      "extension-definition--9c59fd79-4215-4ba2-920d-3e4f320e1e62" : {
           "extension_type" : "new-sco",
           "extensions": {
                "firewall-ext": { … }
           }
       }
    }
}
```

Once approved for inclusion, these objects would look like:

```
{
    "type": "network-device",
    "spec_version": "2.1",
    "id": "network-device--b443fcb9-e106-4d10-a5ed-a8f3ff57b328",
    "mac_addr_ref": "mac-addr–85ab2b71-7f61-4aa1-a218-1ecea24daf01",
    "manufacturer_ref": "identity–45458637-9cb4-4811-bee8-7f92bedbeca5",
    "model_number": "WQ234",
    "extensions": {
        "router-ext": { … }
    }
}

{
    "type": "network-device",
    "spec_version": "2.1",
    "id": "network-device--90001f29-5ea8-4c97-8d8a-ea7282720fc8",
    "mac_addr_ref": "mac-addr–f1a306d1-4969-4653-8f72-13e3a1234583"
    "manufacturer_ref": "identity–1c5506e9-a436-4b0e-aa1c-1825ff6de4cd",
    "model_number": "BT-103",
    "extensions": {
        "firewall-ext": { … }
    }
}
```

6.1.4 Using Property Extensions of Existing SubType Extensions

If you want to add properties to a subtype extension already defined in the specification (e.g. tcp-ext), it is suggested you create a new Extension Definition object. It will not conflict with the existing sub-type extension since it will use the extension definition id for the additional properties.

For instance, here is a current network-traffic SCO that uses the existing sub-type extension `tcp-ext`.

```
{
     "type": "network-traffic"
     "spec_version": "2.1",
```

```
    "id": "network-traffic--09ca55c3-97e5-5966-bad0-1d41d557ae13",
    "src_ref": "ipv4-addr--89830c10-2e94-57fa-8ca6-e0537d2719d1",
    "dst_ref": "ipv4-addr--45f4c6fb-2d7d-576a-a571-edc78d899a72",
    "src_port": 3372,
    "dst_port": 80,
    "protocols": [ "tcp" ],
    "extensions": {
       "tcp-ext": {
           "src_flags_hex": "00000002"
       }
    }
}
```

Assuming we want to add properties to the tcp extension, we would create a new extension definition and use it in addition to the `tcp-ext` extension.

```
{
    "type": "network-traffic"
    "spec_version": "2.1",
    "id": "network-traffic--09ca55c3-97e5-5966-bad0-1d41d557ae13",
    "src_ref": "ipv4-addr--89830c10-2e94-57fa-8ca6-e0537d2719d1",
    "dst_ref": "ipv4-addr--45f4c6fb-2d7d-576a-a571-edc78d899a72",
    "src_port": 3372,
    "dst_port": 80,
    "protocols": [ "tcp" ],
    "extensions": {
       "tcp-ext": {
           "src_flags_hex": "00000002"
       },
       "extension-definition--8727bd6d-969f-4f74-a45e-e17c5a562c0d": {
           "extension_type": "property-extension",
           "checksum_hex": "34db"
       }
    }
}
```

Once approved for inclusion, the extension definition id would be dropped and the properties would be folded into the existing subtype extension, *not the top-level*. The object would look like:

```
{
    "type": "network-traffic"
    "spec_version": "2.1",
    "id": "network-traffic--09ca55c3-97e5-5966-bad0-1d41d557ae13",
    "src_ref": "ipv4-addr--89830c10-2e94-57fa-8ca6-e0537d2719d1",
    "dst_ref": "ipv4-addr--45f4c6fb-2d7d-576a-a571-edc78d899a72",
    "src_port": 3372,
    "dst_port": 80,
    "protocols": [ "tcp" ],
    "extensions": {
       "tcp-ext": {
           "src_flags_hex": "00000002",
           "checksum_hex": "34db"
       }
    }
}
```

6.2 Incorporating New Object Extension Definitions

Some extension definitions will define a new SDO/SCO/SRO object type. Content using the extension definition syntax and regular syntax are similar except the **`extensions`** property will no longer contain the extension definition id entry.

Consider a new SDO type for weakness content as an extension example:

```
    {
        "type": "weakness",
        "spec_version": "2.1",
        "id": "weakness--0c7b5b88-8ff7-4a4d-aa9d-feb398cd0061",
        "created": "2022-12-12T08:17:27.000Z",
        "modified": "2022-12-12T08:17:27.000Z",
        "name": "Use After Free",
        "description": "Referencing memory after it has been freed can cause a program to crash, use
unexpected values, or execute code.",
        "external_references": [
            {
                "source_name": "cwe",
                "external_id": "CWE-416"
            }
          ],
        "languages": [ "c", "c++" ],
        "likelihood_of_exploit": "high",
                <other possible properties not included in the example>
        "extensions": {
            "extension-definition--25660cad-8c7b-4198-85f8-f57ac710c7ce" : {
                "extension_type" : "new-sdo",
          }
     }
```

New SDO type content once approved for inclusion in a future release example:

```
    {
        "spec_version": "2.1",
        "id": "weakness--0c7b5b88-8ff7-4a4d-aa9d-feb398cd0061",
        "created": "2022-12-12T08:17:27.000Z",
        "modified": "2022-12-12T08:17:27.000Z",
        "name": "Use After Free",
        "description": "Referencing memory after it has been freed can cause a program to crash, use
unexpected values, or execute code.",
        "external_references": [
            {
                "source_name": "cwe",
                "external_id": "CWE-416"
              }
          ],
        "languages": [ "c", "c++" ],
        "likelihood_of_exploit": "high",
        …
      }
```

6.3 Handling Existing Data

Because an extension definition approved for inclusion might have at one time been in the "beta" status (or another status) there might exist content that uses the extension definition syntax. There are several ways to handle this situation:

1. For an external extension definition, it is most likely that the extension definition syntax will remain the only syntax that is valid, although this is not required.
2. Either syntax is valid, possibly forever, or just until the new specification has been released.
3. Either syntax is valid, but the extension definition syntax is deprecated and a warning is issued by any validators. This extension definition syntax **SHOULD** be invalid once the new specification has been released.
4. The extension definition syntax is marked as invalid. Producers/consumers **MUST** update to the regular syntax, or ignore the extension definition usage.

Currently, there is no guidance for which choice is appropriate for each extension definition. Therefore, an option will be chosen by the TC for each extension definition, probably based on the amount of content that already exists using the extension definition syntax.

7 Appendix A: JSON Schema of the Indicator Extension for STIX 1.x

File extension-definition--7c8ca481-f0e9-4389-94f5-90df472eb01d.json

```
{
  "$id":
"https://raw.githubusercontent.com/rpiazza/cti-stix-common-objects/stix-1x/extension-definition-specificat
ions/stix-1x/extension-definition--7c8ca481-f0e9-4389-94f5-90df472eb01d.json",
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "stix1x-indicator-extension-schema",
  "description": "This is the wrapper for the indicator extension is used by the stix-elevator",
  "type": "object",
  "allOf": [
    {
      "$ref":
"http://raw.githubusercontent.com/oasis-open/cti-stix2-json-schemas/stix2.1/schemas/sdos/indicator.json"
    },
    {
      "properties": {
        "extensions": {
          "type": "object",
          "properties": {
            "extension-definition--7c8ca481-f0e9-4389-94f5-90df472eb01d": {
                "type": "object",
                "$ref": "stix1x-indicator.json"
            }
          }
        }
```

```
      },
      "required": [
        "extensions"
      ]
    }
  ]
}
```

File stix1x-indicator.json

```
{
        "$id":
"https://raw.githubusercontent.com/rpiazza/cti-stix-common-objects/stix-1x/extension-definition-specificat
ions/stix-1x/stix1x-indicator.json",
        "$schema": "http://json-schema.org/draft-07/schema#",
        "title": "stix1x-indicator-extension-schema",
        "description": "This is the wrapper for the indicator extension is used by the stix-elevator",
        "type": "object",
        "properties": {
                "extension_type": {
                        "type": "string",
                        "enum": [
                                "property-extension"
                        ]
                },
                "likely_impact": {
                        "description": "Specifies the likely potential impact within the relevant context
if this Indicator were to occur. This is typically local to an Indicator consumer and not typically
shared. This field includes a Description of the likely potential impact within the relevant context if
this Indicator were to occur and a Confidence held in the accuracy of this assertion. If the
extension-definition is being used, then it is required",
                        "type": "object",
                        "$ref": "hml-statement-type.json"
                }
        },
        "required": [
                "extension_type",
                "likely_impact"
        ]
}
```

8 Appendix B: References

8.1 B1: Normative References

[STIX 2.1] *STIX Version 2.1*. Edited by Bret Jordan, Rich Piazza, and Trey Darley. 10 June 2021. OASIS Standard. https://docs.oasis-open.org/cti/stix/v2.1/os/stix-v2.1-os.html.

[STIX 2.1 Schemas] OASIS Cyber Threat Intelligence (CTI) TC, "cti-stix2-json-schemas", OASIS. [Online]. Available: https://github.com/oasis-open/cti-stix2-json-schemas.

8.2 B2: Informative References

[ATT&CK] MITRE ATT&CK®, Version 12.0. The MITRE Corporation. [Online]. Available: https://attack.mitre.org/

[TLP] Traffic Light Protocol, Version 2.0 (TLP). (August 2022). FIRST. [Online]. Available: https://first.org/tlp.