OASIS Committee Note

STIX Extension Definition Policy Version 1.0

Committee Note Draft 01

31 March 2025

This stage:
https://docs.oasis-open.org/cti/stix-edp/v1.0/cnd01/stix-edp-v1.0-cn0d1.docx (Authoritative)
https://docs.oasis-open.org/cti/stix-edp/v1.0/cnd01/stix-edp-v1.0-cnd01.html
https://docs.oasis-open.org/cti/stix-edp/v1.0/cnd01/stix-edp-v1.0-cnd01.pdf

Previous stage:
N/A

Latest stage:
https://docs.oasis-open.org/cti/stix-edp/v1.0/stix-edp-v1.0.docx (Authoritative)
https://docs.oasis-open.org/cti/stix-edp/v1.0/stix-edp-v1.0.html
https://docs.oasis-open.org/cti/stix-edp/v1.0/stix-edp-v1.0.pdf

Technical Committee:
OASIS Cyber Threat Intelligence TC

Chairs:
Marlon Taylor (Marlon.Taylor@mail.cisa.dhs.gov), DHS CISA
Alexandre Dulaunoy (alexandre.dulaunoy@circl.lu), Circl

Editor:
Richard Piazza (rpiazza@mitre.org), MITRE

Related work:
This document is related to:

STIX Version 2.1. Edited by Bret Jordan, Rich Piazza, and Trey Darley. Latest stage: http://docs.oasis-open.org/cti/stix/v2.1/os/stix-v2.1-os.html.

Abstract:

This document provides guidance and defines policy for creating and approving extension definitions to the STIX 2.1 specification.

Status:
This is a Non-Standards Track Work Product. The patent provisions of the OASIS IPR Policy do not apply.

This document was last revised or approved by the CTI TC on the above date. The level of approval is also listed above. Check the "Latest stage" location noted above for possible later revisions of this document. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at https://groups.oasis-open.org/communities/tc-community-home2?CommunityKey=c6c33da0-d1ee-42dd-9427-018dc7d32277.

TC members should send comments on this document to the TC's email list. Others should send comments to the TC's public comment list, after subscribing to it by following the instructions on TC's web

page at https://groups.oasis-open.org/communities/community-home?CommunityKey=d682c16c-d20d-45c1-ac8e-018f5aa7b6ae.

Citation format:
When referencing this document, the following citation format should be used:

[stix-edp-v1.0]

STIX Extension Definition Policy Version 1.0. Edited by Richard Piazza. 31 March 2025. OASIS Committee Note Draft 01. https://docs.oasis-open.org/cti/stix-edp/v1.0/cnd01/stix-edp-v1.0-cnd01.html. Latest stage: https://docs.oasis-open.org/cti/stix-edp/v1.0/stix-edp-v1.0.docx.

Table of Contents

1 Introduction

This document provides guidance and defines policy for creating and approving extension definitions to the STIX 2.1 specification.

1.1 Glossary

1.1.1 Acronyms and abbreviations

1.1.2 Document conventions

- Naming conventions
- Font colors and styles
- Typographical conventions

2 Extension Definition Policy Facets

This section defines the facets of the extension definition policy for the STIX 2.1 specification. A facet is an aspect of the status properties of an extension definition. All extensions must use an extension definition, as defined in section 7.3 of the STIX specification.

Custom properties or objects based on the deprecated method of customizing STIX (see section 11 of the STIX 2.1 specification) should be updated to use the extension definitions, since they might not be supported in future releases.

2.1 Location Facet

The Location facet defines where the extension definition can be found. The choices are:

- The Common Object Repository (COR)
- Alternative repositories - publicly available in repositories other than the COR.
- Private – not publicly available

The Common Object Repository is an OASIS repository in GitHub. The inclusion in the COR is at the discretion of the STIX Cyber Threat Intelligence Technical Committee (CTI TC), Inclusion in the COR requires the contributor to agree with the contributor license agreement (CLA). The CLA is discussed further in section 4.1.

The COR can also be thought of as a central clearinghouse for STIX extension definitions. Having an extension definition in the COR makes it easier to find and use during validation, as the needed JSON schemas can be located all in one place.

However, alternative repositories may be used, which may have their own requirements. To enable members of the community to easily discover such an extension definition the COR provides links to these alternative repositories.

In addition, extension definitions might not be publicly available. There are many reasons to create a "private" extension definition such as: a short-term need, when sharing only within a trust group[1], or an experiment. The TC is not involved in their development.

Extension definitions that are not available in the COR do not have to follow the informative text in this document. However, they must be compliant with section 7.3 of the specification to be considered a STIX extension definition.

2.2 Management Facet

This facet defines who is responsible for the management of the extension definition and the state of the intellectual property. Many different community members will have varied interests and reasons for creating an extension definition. However, this facet only distinguishes between those extension definitions that are under the CTI TC or not. The choices are:

- Managed by the TC under the Contributor License Agreement (TC CLA)
- Managed by others

Extension definitions developed under the control of the TC must adhere to the intellectual property requirements of OASIS. In particular, once an extension definition is under the TC's CLA (inclusion in the

[1] a self-selected group of users of STIX that share cyber threat intelligence and use the same extension definitions and other STIX features, such as consistent labels.

COR), contributors must be members of the TC. Non-members can suggest ideas via the TC's comment email list. The TC only has control of its copy of the extension definition in the COR. Extension definitions copies outside the COR are under the open-source license, and as such not controlled by the TC. This is discussed further in section 4.1.

An extension definition can be created to share information based on frameworks and standards developed outside of the CTI 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]. The evolution of such frameworks or standards do not adhere to the STIX release schedule – they have their own release schedule. The property extensions or new STIX object types created to support them may never be officially part of STIX. Therefore, they will always be expressed using extension definition syntax (see section 6.1) unless they are later incorporated into the STIX specification document. These extension definitions can be thought of as being external to the STIX specification. Such extension definitions need not be created by the developers of the framework or standard on which they are based. However, any changes should involve the external developers. These extensions definitions are probably not available in the Common Object Repository in order to avoid them possibly being out of sync with a new version of the definition. Such extension definitions are usually not under the TC's CLA.

Lastly, extension definitions can be created by individuals and organizations who want to share their work with the community but independent of the CTI TC. This can be because of IP considerations, or the TC might not be interested in its domain. These extension definitions, however, should be publicly available in an alternative repository in order for the community to evaluate and/or use them.

To be clear, individuals that are members of the TC can develop or consult on an extension definition that is not under the TC's CLA. However, any contributions that are made are their own as an individual and not associated with the TC.

2.3 TC Stance Facet

The TC Stance facet defines the position the TC is taking on the extension definition. The TC may take the following stances:

- Specification Candidate
- Specification Candidate Beta
- Advised Use Only
- None

The TC may advise the use of any extension definition that is publicly available, even if it is not controlled by the TC. In other words, it is preferred for use by the community by the TC but this stance does not imply anything about the other facets of the extension definition. It may be available in the COR. A list of the extension definitions that the TC advises using will be listed in the COR, in the repository's README file.

The TC can decide that an extension definition in the COR is a specification candidate. Extensions definitions that are of the specification candidate stance or specification candidate beta stance are by definition also controlled by the TC. Extension definitions that have this facet indicates there is TC consensus that it provides a valuable capability.

For an extension definition to be considered a specification candidate it must be approved as such by the TC. Approval of an extension definition is based on the expectation that the extension definition will become an official part of a future version of the STIX specification. Once it is approved, further work is under the direction of the TC. Work on the specification candidate must follow the specification candidate lifecycle workflow as documented in Section 5. The TC can be consulted before the commencement of work on an extension definition but that does not imply it is a specification candidate.

The work on a specification candidate takes place in the COR.

Multiple extension definition implementations of the same concepts may be specification candidates.

When an extension definition becomes a specification candidate it does not imply that the TC is advising its use. For example, the TC may believe a definition provides a valuable capability but that the definition is not yet mature enough for operational use. Once a specification candidate has reached a certain level of maturity, it may become a specification candidate beta. When it does, it indicates that the TC is advising its use.

In addition, many extension definitions do not have any association with the TC, which is referred to as the None stance. They can be alternatives to the ones the TC advises, extension definitions not publicly available or any other.

Location			Management		TC Stance			
COR	Alternative Repository	Private	TC CLA	Others	Specification Candidate	Specification Candidate Beta	Advise Only	None
X			X		X			
X			X			X		
X			X				X	
X			X					X
X			* See 4.3.				X	
X			* See 4.3.					X
	X			X				X
	X			X			X	
		X		X				X

Table 1: Valid Combinations of Extension Definition Facets

2.4 Extension Definition Maturity

The facets associated with an extension definition do not indicate the level of maturity of the extension definition. The maturity of an extension definition is an informal measure of its completeness with respect to the guidelines in section 3. An extension definition that is not publicly available might be fully mature and adhere to the informative language in that section (but not shared), whereas a specification candidate extension definition might just be an idea for an extension that the TC is supporting as a specification candidate.

3 Contents of an Extension Definition

An extension definition in the COR is expected to follow most of these guidelines but (limited) non-conformance does not preclude inclusion in the COR. The TC retains the right to require that any extension definition that the TC controls follow the statements in this section. Developers of private extension definitions should also endeavor to follow these guidelines.

3.1 JSON Schema

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

- Object definitions must use the JSON schema property ‘additionalProperties’ or 'unevaluatedProperties', and the value must be False.
- Deprecated custom properties (see section 11.1 of the STIX 2.1 specification) must not be used.
- Array definitions must contain the property ‘minItems’, and the value must be set to at least 1 (to adhere to the requirement for Lists as stated in the STIX 2.1 specification).
- Open vocabularies should be added to the definitions section of the schema for documentation purposes.

Previously developed JSON schemas for an extension definition (e.g., for an existing standard) can be referenced in an extension definition's JSON schema (using the "$ref" property) but should adhere to the previous bulleted statements.

Best practices for developing a JSON schema for an extension definition can be found in Appendix A.

3.2 Extension Definition Object

An extension definition object must be created. It must 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 should link directly to the root of a directory tree, accessible from the internet, 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 sections 3.3) for the extension definition. However, the location should be in the same root directory.
- The **created_by_ref** property should point to an identity object, which should be available in the COR and should include contact information about the individual or group that is responsible for creating the definition.

As stated in the STIX specification, extension definition STIX objects cannot themselves be extended.

The extension definition STIX object itself should be stored in the COR, unless it is not publicly available. Alternative repositories created by others may be the location of the extension definition STIX object.

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 format of the document (tables, sections, etc.) should be similar to the STIX 2.1 specification. The document should be created using Asciidoc, but MS Word, Google Docs, Markdown, or a similar product are acceptable. This document should be referenced by the extension definition using an external_references URL property.

3.4 Examples

Examples of objects using the extension definition must be provided. They should cover some of the common use cases for the extension definition. They must validate using the extension definition's JSON schema. The example files should be referenced by the extension definition using an external_references URL property. Additionally, they can be included in the documentation as examples.

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 by defining the properties and objects for the STIX 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 definition should exist in its own directory. Tests for both legal and illegal instances of the extension definition should be provided.

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

3.6 Validation

This section assumes that a producer or consumer wishes to support and validate content from one or more extension definitions and specifies informative text to support implementations of a STIX validator. Assuming that an extension definition will be shared with the community (i.e., it is publicly available), a uniform way to validate its use is desirable.

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. 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 (data) specified using extension definitions is stored by consumers is beyond the scope of this document.

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 informative statements in this section. However, the OASIS STIX validator itself does not currently adhere to all of the following informative statements.

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.

   For example, the OASIS STIX validator uses a command line argument:

   stix2_validator <stix_file.json> -s <directory containing json schemas>

2. If the JSON schema is not locally available, it should be accessible using the value of schema URL property of the extension definition STIX object. The STIX validator may directly obtain the json schema using the schema URL of the extension definition. The extension definition object 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. Having the extension definition object in the COR is a straightforward way to make the JSON schema URL available.

4. The base schema must validate, by default, any use of an extension definition for which the schema is not available. This is always true when using the OASIS STIX validator.

5. The schema file must be named using the extension definition object's **id** property, in order for the validator to 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 STIX content must be valid in both the base JSON schemas, and the one associated with the extension definition id. As the base schema must accept any use of an extension definition (see 4 above), 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 The Extension Definition Ecosystem

Through the use of the OASIS GitHub repositories, the TC is creating an Extension Definition ecosystem, where extension definitions of various facet combinations as described in Table 1 can be found. The Common Object Repository, as described in section 2.1 can be found at https://github.com/oasis-open/cti-stix-common-objects.

A list of extension definitions is listed in COR in a README document. Each extension definition is described using the facets described in section 2. Links to each extension definition in the COR are provided. Additionally, links can be provided to the alternative repositories for extension definitions not in the COR. Private extension definitions are not listed.

The purpose of this ecosystem is to make available a "marketplace" for the STIX community to discover extension definitions that are useful to them while they are not part of the current specification. As discussed in section 2.4, extension definitions found in the ecosystem are of various levels of maturity. However, the existence of the ecosystem provides the community with a glimpse to the future of the STIX specification and enables the community to experiment with and vet the usefulness of proposed extension definitions.

The process of including an extension definition into the COR is detailed in section 5.

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

4.1 Licenses, Intellectual Property, CLAs…

If an extension definition is added to the COR, it must be done under the rules and directives set up by OASIS for making contributions to the OASIS Open repositories (see https://www.oasis-open.org/policies-guidelines/ipr/#contributions). The individual (and possibly the entity the individual is doing the work under) must sign a Contributor License Agreement (CLA) that grants OASIS non-revocable rights to use and create derived works from the contributed extension definition (see https://cla-assistant.io/oasis-open/Open-Repo-admin).

In agreeing to the CLA, the CTI TC will have sole authority over the copy of the contribution stored in the COR. The contributor remains the sole authority over their contribution outside of the COR. To avoid ambiguity of these two copies of the contribution, it is required that the COR and non-COR versions of the contribution be given different names.

Contributing an extension definition to the repository does not change the intellectual property rights of the contributor, but makes the extension definition available for use by OASIS and others under the license agreement (https://github.com/oasis-open/cti-stix-common-objects/blob/main/LICENSE.md). A contributed extension definition can be used under any rules stated in the license, which allows for inclusion in future versions of the STIX specification. Contributing an extension definition to the repository does not imply it will be included in a future version.

Note that OASIS has no responsibility to make use of any contribution.

It is important to note that acknowledging the CLA is stating that the creator adheres to the BSD-3-Clause license, which is an "open source" license. The only additional restriction is that OASIS owns the "name" of the extension definition that is in the COR. The CLA does not prohibit anyone, including the original creator, from doing further development of the extension definition independent of the TC on their copy, but they must use a different name and work outside of the COR. This paragraph explains what is indicated by the * entries in Table 1.

Contributions to extension definitions that reside in the COR from outside of the TC can only be made through the comment mailing list (see section 1.7 of https://www.oasis-open.org/policies-guidelines/tc-

process-2017-05-26). The comment mailing list address for the CTI-TC is: cti-comment@lists.oasis-open.org.

4.2 Overall Structure of the Common Object Repository

Extension definitions can be found in the COR in the subdirectory extension-definition-specifications. Each extension definition is contained in its own directory. Upon inclusion in the COR the TC assigns a name for the directory. It will be the name of the STIX object involved followed by the extension definition id's UUID first three digits. For example: `incident-ef7` is the directory name for an extension of the STIX incident object whose id starts with “ef7”. This directory should contain the documentation file, the JSON schema file (whose name is the id of the extension definition STIX object) and a directory that contains examples, which may be referenced in the documentation file.

The subdirectory `objects/extension-definition` should contain the extension definition STIX object.

Alternate repositories that seek to serve the same ecosystem should mirror the structure of the COR, since users will find it easier to navigate and use a familiar structure.

5 Extension Definition Lifecycle

As stated above, not all extensions 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. As discussed in section 2.2, extension definitions that capture information or structures specified in non-STIX standards will not usually be part of the STIX specification, because those frameworks and other standards are not controlled by the TC and may have different release schedules. As stated in section 4.1, any extension definition that is not controlled by the TC can be revised according to creators' own needs and schedules in its own copy. However, an extension definition copy in the COR is under control of the TC and all revisions are subject to OASIS rules.

Additionally, multiple extension definitions for the same STIX object type might be useful or be created to meet a temporary need. How they may be used together, and whether they are included in a future specification will be determined by the TC on a case-by-case basis.

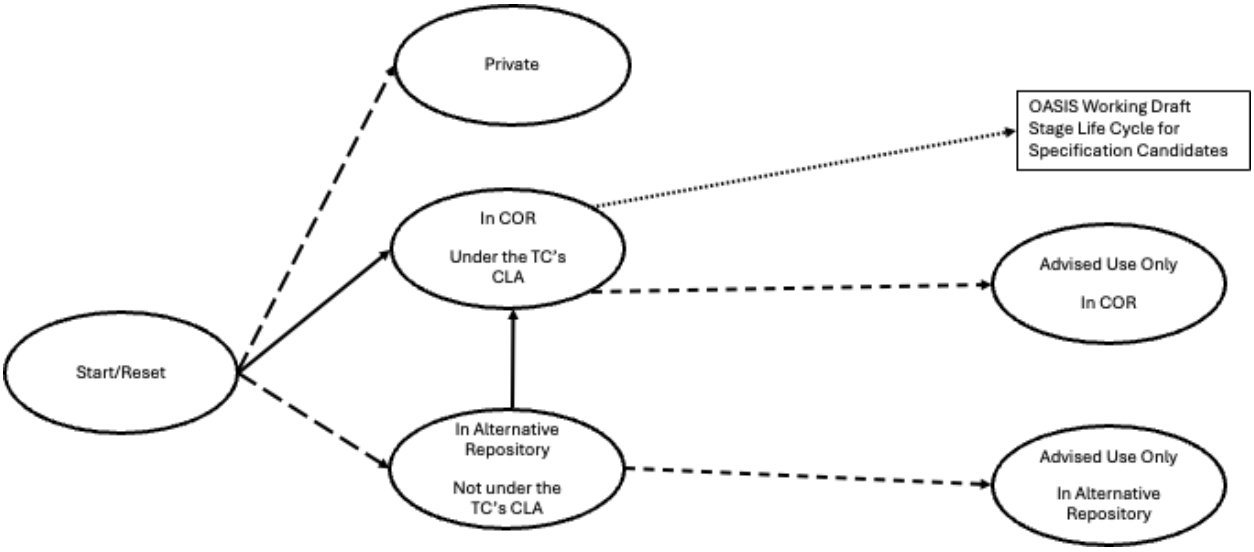

Figure 1. Extension Definition Life Cycle

The Extension Definition Lifecycle (Figure 1) describes the various states of an extension definition based on the facets discussed in section 2. Transitions between states could require TC approval. Note that it is possible for any state to transition to the Start/Reset state, but those edges have been omitted for the sake of clarity.

The transitions in this life cycle are:

- Transitions to "In COR/Under TC CLA" state (solid line)

  Creators of an extension definition can propose to the TC that it want to contribute an extension definition under the TC's CLA. A TC vote is needed to make this transition. A copy of the extension definition will then be added to the COR, which requires the creators to agree to the TC's CLA. Further contributions by the creators or any other non-TC members to the COR copy is constrained by the OASIS rules (section 4.1).

- Transition to "OASIS Working Draft Stage Life Cycle" for Specification Candidates" (dotted line)

  For an extension definition to become a specification candidate it must be put to a ballot by the TC and then follow the OASIS Working Document Stage life cycle (see Figure 2). This transition

is discussed in more detail in section 5.1.

- Transition to "Advised Use Only" states (short dash line)

  TC members can suggest that the TC advises an extension definition's use by the STIX community. As discussed in section 2.3, the TC might prefer that a STIX community uses a particular extension definition regardless of whether it is controlled by the TC. The TC can make this determination with or without direction from the creators of the extension definition. A TC vote is needed to make this transition.

- Other transitions (long dash line) -

  The TC is not involved in these transitions.

5.1 OASIS Working Draft Stage Life Cycle for Specification Candidates

The work on an extension definition follows the OASIS process for shepherding new content into future releases of the STIX specification. Using this process allows the TC to keep track of the various extension definition ideas that might be included in a future release, avoiding unnecessary duplication of work and supporting the individual or group responsible for the extension definition. For details of this process see https://www.oasis-open.org/policies-guidelines/document-life-cycle-best-practices.

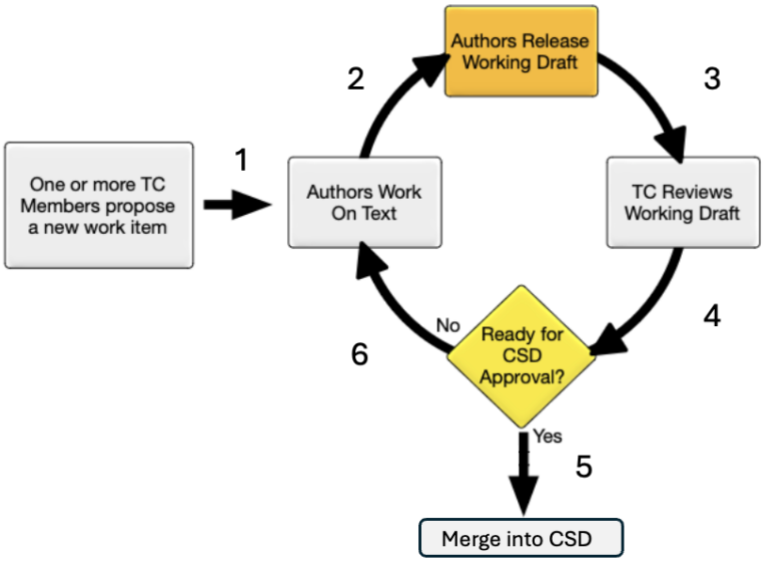

Figure 2: OASIS Working Draft Stage Life Cycle (for Specification Candidates)

Members of the TC can propose a new work item for the development of an extension definition. This may be an idea that is already under development, and perhaps already available in the COR and/or advised for use by the TC, or simply a request to start work on an extension definition concept (e.g., a new STIX object type, or additional properties to an existing object to support a particular use case).

A proposal can come from outside of the TC, but it must first be approved by the TC for inclusion into the COR. Once this takes place, it must follow the OASIS rules for the COR. The TC can then vote to decide to start work on the extension definition under the OASIS process.

If the TC determines that work should begin on a working draft of the extension definition (step 1), the TC proposers of the extension definition may suggest forming a mini-working group to help develop the extension definition (alternatively, development may continue informally) (step 2).

The mini-working group is open to anyone who is a member of the TC, without exception. A chair of the mini-working group may be chosen. Individuals working on the extension definition are referred to as the working draft's "authors."

When the authors feel their working draft is complete and ready, they then release it for review by the TC. The artifacts of the extension definition must adhere to section 3 (step 3). After the review, a decision is made (step 4). If the TC determines it is ready, it proceeds to the next phase, which is inclusion in the committee specification draft (CSD) (step 5). This only signifies that the working draft will be included in the CSD, not that it will necessarily be part of the final committee specification (CS) document.

If it is not ready to be included in the CSD version of the specification, the TC might still advise its use, but it will not be merged into the CSD and it will return to the authors to respond to any comments made by the TC (step 6).

The authors can decide to stop development of an extension definition at any time. If the work has value as a reference or for future development the TC can choose to archive the extension definition. To resume work on an archived extension definition it must be proposed as a new work item.

As stated above, OASIS has no responsibility to make use of any contribution. This does not mean that the extension definition is not usable by the community. It can be used in any manner described in the license.

6 Incorporating an Approved Extension Definition into a Committee Specification Draft (CSD)

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

This section describes the various ways in which extension definitions are incorporated into the STIX specification. The example extension definitions in this section are for expository purposes only and are not intended to reflect any existing content in the COR. They may or may not be proposed extension definitions.

6.1 Definitions

These terms will be used in this section.

Extension definition syntax - the syntax used when an object uses an extension definition. The keys of the **extensions** property are extension definition ids. See the first and second examples in section 6.2.

Standard syntax - the syntax for objects as used in the STIX specification. It does not contain any extension definition ids as keys of the object in the **extensions** property. See the third example in section 6.2.

Top-level properties - all STIX objects have top-level properties. These are expressed in JSON by the keys of the object. Not all properties in STIX 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 in the STIX 2.1 specification.

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 second example in section 6.2.

Subtype extensions - 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 sections 6.3 and 6.4 for examples.

6.2 Using Standard Syntax

If an extension definition is to be incorporated into a future release, the extension definition syntax would need to be re-expressed as standard syntax.

Here is an example of using an extension definition object to extend 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 extending 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 standard 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"
        }
    ]
}
```

6.3 Defining Additional 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.

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 an additional subtype extension.

If an extension definition defines a subclass of an existing STIX object, whose properties are not common to the STIX object, 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"
       }
    }
}
```

6.4 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 could be desirable to use subtype extensions when defining new SCOs.

Here is an example of what this would look like as an extension definition 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": { … }
    }
}
```

6.5 Using Property Extensions of Existing SubType Extensions

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 STIX 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"
        }
     }
}
```

6.6 Incorporating New Object Extension Definitions

Some extension definitions will define a new SDO/SCO/SRO object type. Content using the extension definition syntax and standard 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 definition 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:

```
   {
        "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",
        …
      }
```

6.7 Handling Existing Data

Extension definitions that have been added to the CSD do not use the extension property in its text definition or examples, since they have been expressed using standard syntax. However, any use of the extension definition in data still uses the extension definition syntax until the new specification is released.

Once the new specification is released, an extension definition that was approved for inclusion in the new release might already be being used and therefore there exists content that uses the extension definition syntax. There are several options available to handle this situation:

1. Either syntax is valid, possibly forever.
2. Either syntax is valid, but the extension definition syntax is deprecated, and a warning is issued by any validators.
3. The extension definition syntax is marked as invalid. Producers/consumers must update to the standard syntax or ignore the extension definition usage.

For an extension definition not included in the CS, it is most likely that the extension definition syntax will remain the only syntax that is valid.

Option 2 supports interoperability the best. However, another option can be chosen by the TC for each extension definition, probably based on the amount of content that already exists using the extension definition syntax.

Appendix A. Best Practices for Creating Extension Definitions

A.1 Guidance for Creating a STIX Extension Definition JSON Schema

The example is based on an extension definition used by the STIX elevator to support a STIX 1.x Indicator.

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 A.2 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 of the JSON schema 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 might not be consistent with the OASIS STIX validator.

6. The schema of an extension definition for property extensions of an existing STIX object should be split between two files.

   a. The first schema file, which is named using the extension definition id, should contain the JSON syntax for the 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. 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.

      iii. 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"
    ]
}
```

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

ii. 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 definition.

7. The best practice for new object extensions schemas is to use one file.

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"
}
```

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
}
```

A.2 Extension Definitions for Open Vocabularies and Enumerations

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". 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 and schema.

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, the specification of such properties states that values "should" come from the open vocabulary. A STIX validator may issue warnings whenever the suggested values are not used. This is useful when the value is a typo of a suggested value and therefore not the intended suggested value from the open vocabulary.

Continuing with the implementation-language-ov open vocabulary example, at a certain point, newly defined languages will commonly be specified for this property and the warnings would be undesirable and somewhat irrelevant. Using an extension definition to extend an open vocabulary allows users to understand what new values are suggested for the open vocabulary.

This could enable a STIX validator to avoid issuing these warnings, although this is not implemented in the current 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.

The value of a property defined using an enumeration must be one of the enumeration values. Therefore, adding a value using an extension definition 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 it is necessary 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
}
```

Appendix B. Informative References

This appendix contains the references that are used in this document.

[ATT&CK] MITRE ATT&CK®, Version 16.0. The MITRE Corporation. [Online]. Available: https://attack.mitre.org/

[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.

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

Appendix C. Acknowledgments

C.1 Special Thanks

Substantial contributions to this document from the following individuals are gratefully acknowledged:

Desire Beck, MITRE
Jeff Mates, DC3/TSD
Charles Schmidt, MITRE
Marlon Taylor, CISA

C.2 Participants

The following individuals were members of this Technical Committee during the creation of this document and their contributions are gratefully acknowledged:

First Name	Last Name	Company
Alexandre	Dulaunoy	CIRCL
Marlon	Taylor	US DHS Cybersecurity and Infrastructure Security Agency (CISA)
Leszek	Adamiak	IBM
David	Ailshire	DHS Cybersecurity and Infrastructure Security Agency (CISA)
Syam	Appala	Cisco Systems
Drew	Armstrong	Australian Signals Directorate
Nick	Ascoli	Cyware Labs
Jorge	Aviles	Johns Hopkins University Applied Physics Laboratory
Stephen	Banghart	US NIST
Desiree	Beck	MITRE Corporation
Ted	Bedwell	Cisco Systems
Eldan	Ben-Haim	IBM
Jeremy	Berthelet	Capgemini
David	Bizeul	SEKOIA . IO
Georges	Bossert	SEKOIA . IO
Mike	Boyle	National Security Agency
James	Cabral	James E. Cabral Jr. (Personal)
Michael	Chisholm	MITRE Corporation
Mike	Cokus	MITRE Corporation
Sam	Cornwell	MITRE Corporation
James	Crossland	Northrop Grumman
Scott	Dowsett	Anomali
Terrence	Driscoll	Cyware Labs
Jessica	Fitzgerald-McKay	National Security Agency
Steven	Fox	DHS Cybersecurity and Infrastructure Security Agency (CISA)
Utkarsh	Garg	Cyware Labs

Anuj	Goel	Cyware Labs
Pavan	Gudimetta	MITRE Corporation
Roseann	Guttierrez	IBM
Sandra	Hernandez	IBM
Taneika	Hill	US DHS Cybersecurity and Infrastructure Security Agency (CISA)
Wei	Huang	Anomali
Tim	Hudson	Cryptsoft Pty Ltd.
Caitlin	Huey	Cisco Systems
Andras	Iklody	CIRCL
Rachel	James	Rachel James (Personal)
Elysa	Jones	Elysa Jones (Personal)
Bret	Jordan	Afero
Avkash	Kathiriya	Cyware Labs
David	Kemp	National Security Agency
Qin	Long	Alibaba Cloud Computing Ltd.
Qem	Lumi	Peraton
Terry	MacDonald	Individual
Vasileios	Mavroeidis	University of Oslo
Evette	Maynard-Noel	US DHS Cybersecurity and Infrastructure Security Agency (CISA)
Shaun	McCullough	National Security Agency
Julie	Modlin	Johns Hopkins University Applied Physics Laboratory
Luca	Morgese Zangrandi	TNO
John	Morris	IBM
Mark	Moss	Johns Hopkins University Applied Physics Laboratory
Mark	Munoz	Johns Hopkins University Applied Physics Laboratory
Timothy	O'Neill	MITRE Corporation
Jackie Eun	Park	US DHS Cybersecurity and Infrastructure Security Agency (CISA)
Nicole	Parrish	MITRE Corporation
Katie	Pelusi	Anomali
Josh	Poster	National Council of ISACs (NCI)
Pavan	Reddy	Cisco Systems
Nathan	Reller	Johns Hopkins University Applied Physics Laboratory
Daniel	Riedel	GL Venture Studio HoldCo LLC
Mark	Risher	Google Inc.
Larry	Rodrigues	MITRE Corporation
Nick	Rossmann	IBM
Zach	Rush	MITRE Corporation
Laura	Rusu	IBM
Jon	Salwen	MITRE Corporation
Omar	Santos	Cisco Systems
Thomas	Schaffer	Cisco Systems

Charles	Schmidt	MITRE Corporation
Michael	Simonson	Cisco Systems
Florian	Skopik	AIT Austrian Institute of Technology
Sean	Sobieraj	US DHS Cybersecurity and Infrastructure Security Agency (CISA)
Ben	Sooter	Electric Power Research Institute (EPRI)
Duncan	Sparrell	sFractal Consulting LLC
Richard	Struse	MITRE Corporation
Sam	Taghavi Zargar	Cisco Systems
Allan	Thomson	Individual
Alex	Tweed	MITRE Corporation
Sulakshan	Vajipayajula	IBM
Robert	Van Dyk	Northrop Grumman
Emmanuelle	Vargas-Gonzalez	MITRE Corporation
Jyoti	Verma	Cisco Systems
Raphav´l	Vinot	CIRCL
Preston	Werntz	US DHS Cybersecurity and Infrastructure Security Agency (CISA)
Ron	Williams	IBM
Andrew	Windsor	Cisco Systems
Mateusz	Zych	University of Oslo
Jane	Ginn	Individual
Jonathan	Matkowsky	Microsoft Corporation
Sean	Carroll	National Security Agency
Marco	Caselli	Siemens AG
Kartikey	Desai	MITRE Corporation
Jason	Keirstead	Pobal Cyber Ltd
Kevin	Klein	US DHS Cybersecurity and Infrastructure Security Agency (CISA)
Chenta	Lee	IBM
Chris	Lenk	MITRE Corporation
Patrick	Maroney	AT&T Services, Inc.
Jeffrey	Mates	US Department of Defense (DoD)
Richard	Piazza	MITRE Corporation
Emily	Ratliff	IBM
Stephan	Relitz	Peraton
Chris	Ricard	Financial Services Information Sharing and Analysis Center (FS-ISAC)
Aviv	Ron	IBM
Michael	Rosa	National Security Agency
Christian	Studer	CIRCL
Dean	Thompson	Australia and New Zealand Banking Group (ANZ Bank)

Appendix D. Revision History

Revisions made since the initial stage of this numbered Version of this document may be tracked here.

Revision	Date	Editor	Changes Made
1.0	[Rev Date]	Rich Piazza	Initial draft

Appendix E. Notices