Where did this come from?

This originated from the discussions around CTI Common and whether it's needed as a separate work product/spec; this boiled over into discussion on the common properties that CybOX Objects need (IDs? Versioning?) and from there went on to CybOX relationships.

This was also driven by concerns that the proposed CybOX Object model for v3.0 makes heavy use of relationships, which could make CybOX content more verbose and difficult to parse. E.g., for the proposed Network Connection Object, source and destination addresses would be specified via relationships:

- Network Connection → *hasSource* → IPv4 Address
- Network Connection → *hasDestination* → IPv4 Address

What is CybOX trying to do?

Fundamentally, CybOX is scoped around the ability to represent *instances* of both static (e.g., a particular file on an endpoint) and dynamic (e.g., the creation of said file) cyber entities, for use in various contexts. Accordingly, we need to consider the following questions:

- What are the core capabilities needed by CybOX?
    - Objects?
        - Observable objects?
        - Observable actions?
    - Relationships?
    - Patterns?
- How is CybOX currently used elsewhere (besides in STIX)?
    - In MAEC, CybOX Objects are defined as TLOs and referenced from corresponding entities (Malware Subjects, Actions, etc.)
    - In DFAX, CybOX Objects are defined as TLOs and referenced from corresponding entities

CybOX Relationships

Use Cases

- One object **is composed of** another object
    - Network connection A *has a source address* of Address B

- One object **acts on** another object
    - Process A *created* File B

- One object is related to another object in some **particular context**
    - Hostname A *resolves to* Address B at a particular time/times

- An action **acts on** an object
    - File Delete *affects* File B

- More generally, relationships allow for the expression of graph-based data
    - CybOX Objects (et. al) are nodes in the graph
    - Relationships are the edges between the nodes in the graph
    - E.g., Domain Name **A** *resolves to* IP Address **B** which is *part of* AS **C**

Fundamental Questions

- Are CybOX Object relationships necessary?
    - This has fundamental implications on how CybOX Objects would be constructed
    - If *NO*:
        - CybOX Objects would have to revert to a completely embedded approach
            - Simple to define in some cases, not so much in others
                - In many cases it would have to be an OR between several other Object types
            - The semantics of embedded Objects may be unclear or awkward
        - Actions could not reference existing Objects
            - E.g., multiple Actions that operate on the same Object, such as a file that was created, written to, and then deleted
    - If *YES*:
        - CybOX Objects could be defined using either an embedded or relationship driven approach
        - Which properties belong on CybOX relationships? Do we need any?
            - ID
            - Information source
            - Created timestamps
    - Could also be done on a case-by-case basis
- If we support relationships between CybOX Objects, which *classes* of relationships should we support?
    - Also, do we need a special class of relationship for Object <-> Object relationships? Or are STIX and CybOX relationships effectively identical?
- If we do not support relationships between CybOX Objects, could they be replaced with a different capability?
    - Deterministic IDs?
    - Patterns?
- Can there be some middle ground where we support CybOX Object relationships, without extensive global relationships?

CybOX Objects

Use Cases

- Observations
    - File A was observed at time X
    - Malware analysis (via Actions): Malware binary X created File A at time X
    - Digital forensics: File A was found starting at Sector X on Disk Y

Fundamental Questions

- Are CybOX objects themselves TLOs or are they always included in the context of some other object?
    - Can they be independently versioned?
    - Do they need to be marked separately?
- If they *are not* TLOs:
    - They're just definitions for things without the context for what they mean
    - What you include them in determines the meaning:
        - Observation: I saw this
        - Pattern: Look out for this
        - MAEC: This malware does these things
    - It makes it more difficult to relate Objects found in different contexts, e.g., in MAEC and STIX
        - As an example, say that a MAEC report highlights a File created by a particular malware instance; the same File is also part of an Observation in STIX. If CybOX Objects are not TLOs, then MAEC and STIX wouldn't be able to reference the same File Object
- If they *are* TLOs, what does this mean? Are they effectively subsumed by STIX?
    - As TLOs, they have a common implementation that can referenced and used in any supporting language
        - STIX
        - MAEC
        - etc.
    - Which properties belong on CybOX Objects?

Existing CybOX Relationships

The following is an examination of a subset of the Object-->Object relationships that exist in CybOX, particularly around whether they can be replaced or implemented using a different method:

- Equivalent action: the relationship can be replaced with an equivalent CybOX Action.
    - E.g., "created" for a File Object can be replaced with a "create file" Action that operates on a File Object

- Embedded object: the relationship can be replaced with an Object directly embedded inside of another Object.
    - E.g., a File Object that “contains” another File Object could be replaced with a field directly on the File Object (such as “contained_objects”) that contains a list of File Objects

Relationship	Relationship Type	Possible Replacement	Open Questions/Comments
created	acts on	Equivalent action.	n/a
deleted	acts on	Equivalent action.	n/a
modified_properties_of	acts on	Equivalent action.	n/a
downloaded	acts on	Equivalent action.	n/a
uploaded	acts on	Equivalent action.	n/a
values_enumerated	acts on	Equivalent action.	n/a
killed	acts on	Equivalent action.	n/a
locked	acts on	Equivalent action.	n/a
unlocked	acts on	Equivalent action.	n/a
listed_on	acts on	Equivalent action.	n/a
renamed	acts on	Equivalent action.	n/a
moved	acts on	Equivalent action.	n/a
copied	acts on	Equivalent action.	n/a
connected_to	acts on	Equivalent action.	n/a
suspended	acts on	Equivalent action.	n/a
paused	acts on	Equivalent action.	n/a
resumed	acts on	Equivalent action.	n/a
wrote_to	acts on	Equivalent action.	n/a
read_from	acts on	Equivalent action.	n/a
allocated	acts on	Equivalent action.	n/a

freed	acts on	Equivalent action.	n/a
opened	acts on	Equivalent action.	n/a
closed	acts on	Equivalent action.	n/a
downloaded_to	acts on/contextual	Equivalent action.	n/a
sent_to	acts on/contextual	Equivalent action.	n/a
renamed_to	acts on/contextual	Equivalent action.	n/a
moved_to	acts on/contextual	Equivalent action.	n/a
copied_to	acts on/contextual	Equivalent action.	n/a
contains	container	Embedded object.	There are lots of possible contains relationships between different Objects. Do we understand all of them? Also, this could lead to some massive nested objects - imagine an archive file that contains another archive that contains another archive file…
contained_within	container	Embedded object.	Same as for contains.
characterizes	contextual	? Probably should be deprecated.	n/a
dropped	contextual	? Probably should be deprecated.	n/a
resolved_to	contextual	Embedded object (domain | hostname | ip addr).	Do the semantics make sense at the Object-level?
sub_domain_of	contextual	Embedded object (domain).	n/a

supra_domain_of	contextual	Embedded object (domain).	n/a
fqdn_of	contextual	Embedded object (domain).	n/a
received_from	contextual	Embedded object (email?).	n/a
renamed_from	contextual	Embedded object (file).	Do the semantics make sense at the Object-level?
moved_from	contextual	Embedded object (file).	Do the semantics make sense at the Object-level?
copied_from	contextual	Embedded object (file).	Do the semantics make sense at the Object-level?
parent_of	contextual	Embedded object (process | thread).	n/a
child_of	contextual	Embedded object (process | thread).	n/a
downloaded_from	contextual	Embedded object (URL?).	Do the semantics make sense at the Object-level?
redirects_to	contextual	Embedded object (URL).	Do the semantics make sense at the Object-level?

Proposal

TLO Approach

Objects and relationships are defined as top-level entities, and used (nearly) identically in all languages that adopt CybOX.

This would entail that:

- CybOX Objects, Actions, and Object Relationships are defined as **TLOs** in CybOX Core
    - Notional properties:
        - id
        - created_timestamp
        - information_source

- STIX imports and uses CybOX Objects and Relationships as TLOs in a STIX Package
- STIX Observations would reference the “primary” CybOX Object involved
    - More implicit, and would require following the relationship chain to determine the full scope of the Observation
- MAEC, et. al would import and use CybOX Objects and perhaps Relationships as TLOs in a MAEC Package
- CybOX can be used as a standalone language
    - E.g., as a means for sensors to report what they’ve observed
- **Pros**
    - Consistent usage of Objects and Relationships across all CybOX users (STIX, MAEC, et. al)
        - Always defined as TLOs
    - CybOX data from different contexts can be easily associated
- **Cons**
    - This may make it more difficult to share individual CybOX Objects, since they would have to be shared with their corresponding Relationships and Related Objects
        - Not of great consequence if the STIX Package is the default shared entity, as it would encompass the needed Objects and Relationships

Container Approach

Objects and relationships are embedded directly inside of their parent container (e.g., a STIX Observation or a MAEC Package).

This would entail that:

- CybOX Objects, Actions, and Object Relationships are defined as **types** in CybOX Core
- STIX imports and uses CybOX Objects and Relationships **not** as TLOs, but only in key components such as Observation
    - More explicit, but in more complex cases (many objects and relationships), may need a special field (e.g., “observation_target”) to specify the “root” of was observed
- CybOX cannot be used as a standalone language (it is just a collection of types)
- **Pros**
    - This gives each language the flexibility to use the CybOX Components they want as needed
        - As TLOs, as embedded entities, etc.
    - May keep CybOX from having to take on additional STIX-centric components, such as data markings, versioning, etc.
- **Cons**
    - May make it more difficult to associate data from different contexts

- E.g., a MAEC report and STIX Observation that point to the same File Object
- In such cases, CybOX Objects would need to be duplicated, but could be de-duplicated by the consumer of the data (e.g., a tool)

Workflow Examples

The following are a set of notional CTI workflows to help illustrate how STIX/CybOX/MAEC instance data would look using either a TLO or container based approach, in conjunction with either the relationship based approach of specifying CybOX Objects or a notional one based on embedding ALL Objects (just as a strawman).

Relationship-based

Workflow	TLO Approach	Container Approach
Network Connection w/ AS Characterization	JSON Example	JSON Example
Domain → IP Address Resolution Characterization	JSON Example	JSON Example
Recursive Archive File Characterization	JSON Example	JSON Example
Malware Characterization w/ MAEC	JSON Example	JSON Example
Malware Characterization w/ MAEC + Observation	JSON Example	JSON Example

Embedded-object (notional) Based

Workflow	TLO Approach	Container Approach
Network Connection w/ AS Characterization	JSON Example	JSON Example
Domain → IP Address Resolution Characterization	JSON Example	JSON Example
Recursive Archive File Characterization	JSON Example	JSON Example

Moving Forward/Thoughts

TLO vs. Containers

- The container approach appears to lead to a cleaner, simpler CybOX, since the usage of CybOX is dictated by its *downstream* users
  - This results in a more explicit use of CybOX, with clearer semantics
  - It may result in some duplicate data when dealing with the same CybOX Objects in multiple contexts (e.g., MAEC + STIX), but this could be handled at the consumption level (e.g., by a tool)

- The TLO approach makes the most sense if there is real value in having CybOX as a standalone language
    - This is largely dependent on whether Observations are a STIX or CybOX construct
    - However, even for the sensor-language argument, one could envision sensors generating STIX Observations
- **Suggested path forward**
    - Move forward with container-based approach, unless strong arguments are made in favor of TLO-based approach

Relationships vs. Embedded Objects

- Embedded Objects should be easier to create and parse in most cases
    - Exceptions: anything with recursion, such as the archive file
- Contextual relationships may still be useful in some cases, as they provide clearer semantics than a field on an Object
    - Especially true for Objects where a relationship is applicable to more than one class of Object
        - E.g., parent/child for a Process or Thread
- **Suggested path forward**
    - Keep relationships, but consider simplifying certain Objects through use of embedded Objects
        - Case-by-case basis
    - Prune existing set of Object relationships
        - Consider deprecating Action-based relationships in favor of explicit Actions
        - Evaluate container and contextual relationships