CybOX 3.0 Specification

Version 3.0-draft1

Specifications Cover Page

Technical Committee

OASIS Cyber Threat Intelligence (CTI) TC

Chair

Richard Struse (Richard.Struse@hq.dhs.gov), DHS Office of Cybersecurity and Communications (CS&C)

Editors

Ivan Kirillov, (ikirillov@mitre.org), MITRE Corporation
Trey Darley (trey@kingfisherops.com), Kingfisher Operations, sprl

Table of Contents

1. Introduction

CybOX is a language and serialization for the structured representation of observed objects and their properties in the cyber domain. CybOX can be used to describe data in many different functional domains, including but not limited to:

- Malware characterization

- Intrusion detection
- Incident response & management
- Digital forensics

CybOX describes one or more observed data points, for example, information about a file that existed, a process that was observed running, or that a network connection existed between two IPs. It describes the facts concerning **what** happened, but not necessarily the who or when, and never the why.

In response to lessons learned in implementing previous versions, CybOX has been significantly redesigned (TODO: add reference to CybOX 2.1.1) and, as a result, omits some of the objects, types, and properties defined in CybOX 2.1.1. The objects chosen for inclusion in CybOX 3.0 represent a minimally viable product (MVP) that fulfills basic consumer and producer requirements for cyber observed data characterization. Objects and properties not included in CybOX 3.0, but deemed necessary by the community, will be included in future releases.

1.1. Terminology

The keywords "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in RFC2119 (REF:RFC2119).

Consumer - Any entity that receives CybOX content.
CTI - Cyber Threat Intelligence
CybOX - Cyber Observable Expression
Entity - Anything that has a separately identifiable existence (e.g., organization, person, group, etc.).
MTI - Mandatory to Implement
MVP - Minimally Viable Product
Object Creator - The entity that created a CybOX object
Producer - Any entity that distributes CybOX content, including object creators as well as those passing along existing content.
STIX - Structured Threat Information Expression

1.2. Overview

1.2.1. CybOX Objects & Actions

CybOX 3.0 defines a set of CybOX Objects for characterizing host-based, network, and similar entities. Each of these objects correspond to a data point commonly represented in CTI and forensics. Using the building blocks of CybOX Objects alongside CybOX relationships,

individuals can create, document, and share broad and comprehensive information about computer systems and their state.

Actions in CybOX are intended to represent an cyber causally correlated data point that results in a change or effect on a CybOX Object, and are particularly useful in the CTI sub-domains of malware analysis and forensics. For example, an Action can represent the creation of a particular file on a file system. Unlike CybOX Objects, Actions are not defined in CybOX 3.0, and will be implemented in a future minor release.

1.2.2. CybOX Containers

CybOX Containers group CybOX Objects and Actions into a single collection of data. They enable the specification of relationships (via local identifiers) so that objects like Network Connection may be related to IP Address, while ensuring simplicity of finding and discovering information. A Container is a complete data point used for analysis and investigation.

1.2.3. CybOX Relationships

A relationship is a reference linking two (or more) related CybOX Objects. CybOX relationships are only resolvable within the same CybOX Container. References are a property on CybOX Objects that contain the ID of a different CybOX Object.

1.2.4. CybOX Extensions

Each CybOX Object defines a set of base properties that are generally applicable across any use case for the Object. There are a number of specific use cases that require additional specialized properties on CybOX Objects. To enable this, CybOX permits the specification of such additional properties through the set of default Object Extensions. Where applicable, extensions are included in the definitions of CybOX Objects. Producers may extend CybOX by defining their own custom extensions. For further information, refer to section **<TODO: add reference>**

1.2.5. CybOX Vocabularies

Many CybOX Objects contain properties whose values are constrained by a predefined enumeration. These sets of values are called vocabularies and are defined in CybOX in order to enhance interoperability by increasing the likelihood that different entities use the same exact string to represent the same concept. In the case of closed vocabularies, this is a hard constraint; in the case of open vocabularies, this is a suggestion for producers that permits the use of values outside of the suggested vocabulary. If used consistently, vocabularies make it less likely that one entity refers to the md5 hashing algorithm as "md5" and another as "md-5-hash", thereby making comparison and correlation easier.

1.2.6. Serialization

CybOX is defined independent of any specific storage or serialization. However, the mandatory-to-implement (MTI) serialization for CybOX 3 is JSON (todo add ref). Therefore, all CybOX 3-compatible tools **MUST** support JSON as a serialization format. CybOX 3-compatible tools **MAY** support serializations other than JSON.

As JSON is the MTI serialization, all examples in this document are expressed in JSON.

1.2.7. Transporting CybOX

CybOX 3 is transport-agnostic, i.e. the structures and serializations do not rely on any specific transport mechanism. However, two companion CTI specifications, STIX **<TODO: add ref>** and TAXII **<TODO: add ref>**, are designed specifically to transport CybOX Objects in JSON.

1.3. Conventions

1.3.1. Naming Conventions

All type names, property names, and literals are in lowercase. Words in property names are separated with an underscore(_), while words in type names and string enumerations are separated with a dash (-). All type names, property names, object names, and vocabulary terms are between three and 250 characters long.

In the JSON serialization all property names and string literals **MUST** be exactly the same, including case, as the names listed in the property tables in this specification. For example, the CybOX property **extended_properties** must result in the JSON key name "extended_properties". Properties marked required in the property tables **MUST** be present in the JSON serialization.

1.3.2. Font Colors and Style

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

- The `Consolas` font is used for all type names, property names and literals.
  - type names are in red with a light red background - `hashes-type`
  - property names are in bold style - **`file_name`**
  - literals (values) are in green with a green background - `sha-256`

- In an object's property table, if a common property is being redefined in some way, then the background is dark gray.
- All examples in this document are expressed in JSON. They are in `Consolas` 9 pt found, with straight quotes, black text and a `light blue background`. All examples have a 2 space indentation. Parts of the example may be omitted for conciseness and clarity. These omitted parts are denoted with the ellipses (...).

2. Common Data Types

CybOX *properties* are named fields whose values are composed of single values or a homogenous list from the set of common data types (i.e., `boolean`, `float`, etc.)

Note that in certain cases the CybOX specification makes finer semantic distinctions than the JSON Mandatory to Implement (MTI) serialization natively allows. For example, CybOX distinguishes between an `integer` type and a `float` type, but at the JSON serialization level these types coalesce into a single representation. Similarly, at the MTI serialization level both the `binary` and `string` types are represented by JSON strings, but at the level of the CybOX data model a distinction is made.

2.1. Integer

Type Name: `integer`

The integer data type represents a whole number. Unless otherwise specified, all integers **MUST** be capable of being represented as a signed 64-bit value. Additional restrictions **MAY** be placed on the type as described where it is used.

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

2.2. Float

Type Name: `float`

The float data type represents an IEEE 754 double-precision number (e.g., a number with a fractional part). However, because the values ±Infinity and NaN are not representable in JSON, they are not valid values in CybOX.

In the JSON MTI serialization, floating point values are represented by the JSON number type.

2.3. Boolean

Type Name: `boolean`

A `boolean` is a value of either true or false. Properties with this type **MUST** have a value of `true` or `false`.

In the case of optional `boolean` fields, where they are omitted, they **MUST** be interpreted as implicitly `false`.

The JSON MTI serialization uses the JSON boolean type **<TODO: add reference>**, which is a literal (unquoted) `true` or `false`.

2.4. String

Type Name: `string`

The `string` data type represents a finite-length string of valid characters from the Unicode coded character set [REF:ISO.10646]. Unicode incorporates ASCII [REF: RFC20] and the characters of many other international character sets.

The JSON MTI serialization uses the JSON string type **<TODO: add reference>**, which mandates the UTF-8 encoding to support Unicode.

2.5. Binary

Type Name: `binary`

The `binary` data type represents a sequence of bytes.

The JSON MTI serialization represents this as a base64-encoded string as specified in RFC4648. Other CybOX serializations SHOULD use a native binary type, where available.

2.6. Hexadecimal

Type Name: `hex`

The `hex` data type encodes an array of octets (8-bit bytes) as hexadecimal. The string **MUST** consist of an even number of hexadecimal characters, which are the digits '0' through '9' and the letters 'a' through 'f'.

Example

```
...
        "tcp":{
          "src_port":"3372",
          "dst_port":"80",
          "src_flags":"00000002"
        }
...
```

2.7. List

Type Name: `list`

A `list` contains an ordered sequence of values. When the phrasing "`list` of type `<type>`" is used, all values in the list **MUST** be of the specified type. For instance, `list` of type `number` means that all values of the list must be of the number type. Upper and lower bounds of the list – the minimum and maximum number of elements – may be specified where the list is used. This section does not specify the upper and lower bounds of `list`.

The JSON MTI serialization uses the JSON array type, which is an ordered list of zero or more values.

2.8. Dictionary

Type Name: `dictionary`

A CybOX `dictionary` captures a set of key/value pairs. CybOX `dictionary` keys **MUST** be unique in that dictionary, **MUST** be in ASCII, and are limited to the characters a-z (lowercase ASCII), A-Z (uppercase ASCII), numerals 0-9, hyphen (-), and underscore (_). CybOX `dictionary` keys **SHOULD** be no longer than 30 ASCII characters in length, **MUST** have a minimum length of 3 ASCII characters, **MUST** be no longer than 256 ASCII characters in length, and **SHOULD** be lowercase.

CybOX `dictionary` values **MUST** be valid CybOX property base types.

2.9. Timestamp

Type Name: `timestamp`

The timestamp type represents a discrete date/time stamp. The timestamp value **MUST** be a valid RFC 3339-formatted timestamp using the format `YYYY-MM-DDTHH:mm:ss[.s+]Z` where

the "s+" represents 0 or more sub second values. The timestamp **MUST** be represented in the UTC timezone and **MUST** use the 'Z' designation to indicate this. Unless otherwise specified, CybOX timestamps **MUST** have millisecond precision.

Example
Both the **start_time** and **end_time** properties on the Network Connection Object use the `timestamp` type.

```
{
  "type":"cybox-container",
  "spec_version":"3.0",
  "objects":{
    "0":{
      "type":"ipv4-address-object",
      "value":"1.2.3.4"
    },
    "1":{
      "type":"ipv4-address-object",
      "value":"2.3.4.5"
    },
    "2":{
      "type":"network-connection-object",
      "start":"2016-01-20T12:31:12.12345Z",
      "end":"2016-01-20T14:11:25.55Z",
      "src_ref":"0",
      "dst_ref":"1",
      "protocols":{
        "layer4":"tcp"
      }
    }
  }
}
```

2.10. Controlled Vocabulary

Type Name: `controlled-vocab`

A controlled vocabulary is a string property that defines a list of allowable values; these allowable values are said to be the specified vocabulary. The value of a `controlled-vocab` property **MUST** be a value from the specified vocabulary.

Controlled vocabulary properties will also have an optional companion “extension” property, of type `vocab-ext`, that can be used to provide an additional value that is not in the specified vocabulary. The key name for the extension property will be **`[vocabulary_property_name]_ext`**. The **`[vocabulary_property_name]_ext`** extension

property is only intended to provide further specification for the `controlled-vocab` property in a custom vocabulary and therefore **MUST NOT** be present unless the `controlled-vocab` property is also present.

In cases where the `controlled-vocab` property is a list, the **`[vocabulary_property_name]_ext`** extension property will also be a list. There is no correspondence between items in these lists: all items in the `controlled-vocab` property are considered independent from all items in the **`[vocabulary_property_name]_ext`** property.

Example
In this example the property **`service_type`** is a controlled vocabulary, which means that only values present in the specified vocabulary are valid.

```
{
  "type":"cybox-container",
  "spec_version":"3.0",
  "objects":{
    "0":{
      "type":"process-object",
      "pid":266,
      "extended_properties":{
        "windows-service":{
          "service_type":"service_file_system_driver"
        }
      }
    }
  }
}
```

2.11. Open Vocabulary

Type Name: `open-vocab`

An open vocabulary is a string property that provides a list of suggested values, without constraining producers from extending those values. The list of suggested values is known as the suggested vocabulary. The value of an `open-vocab` property **SHOULD** be a value from the suggested vocabulary and **MAY** be or any other `string` value. Values that are not from the value list **SHOULD** be all lowercase (where lowercase is defined by the locality conventions) and **SHOULD** use dashes instead of spaces or underscores.

Example
In this example the property *encryption_algorithm* is an open vocabulary, which means that both values present in the specified vocabulary and any custom vocabulary values are valid.

```
{
  "type":"cybox-container",
  "spec_version":"3.0",
  "objects":{
    "0":{
      "type":"file-object",
      "hashes":{
        "md5":"66e2ea40dc71d5ba701574ea215a81f1"
      },
      "size":641028,
      "extended_properties":{
        "archive":{
          "encryption_algorithm":"aes"
        }
      }
    }
  }
}
```

2.12. Object Reference

Type Name: `object-ref`

The Object Reference data type specifies a local reference to a CybOX Object, that is, one which **MUST** be valid within the local scope of the CybOX Container that holds both the Object itself and the referenced Object via its `id`.

Example
The following example demonstrates how a Network Connection Object specifies its destination via a reference to an IPv4 Address Object.

```
{
  "type":"cybox-container",
  "spec_version":"3.0",
  "objects":{
    "0":{
      "type":"ipv4addr-object",
      "value":"1.2.3.4"
    },
    "1":{
      "type":"network-connection-object",
      "dst_ref":"0"
    }
  }
}
```

2.13. Hashes Type

Type Name: `hashes-type`

The Hashes class represents 1 or more hashes, as a dictionary. Accordingly, the name of each hashing algorithm **MUST** be specified as a key in the dictionary. This name **MUST** either be one of the values defined in the `hash-algo-ov` OR a custom value prepended with "x_" (e.g., "x_custom_hash"). The key **MUST** identify the hashing algorithm used to generate the value. The corresponding value for each key **MUST** be the hash value.

Examples
MD5 and Custom Hash

```
    {
        "md5": "3773a88f65a5e780c8dff9cdc3a056f3",
        "x_foo_hash": "aaaabbbbccccddddeeeeffff0123457890"
    }
```

3. CybOX Objects

This section outlines the common properties and behavior across all CybOX Objects.

The JSON MTI serialization uses the JSON object type <todo add reference> when representing all CybOX Objects.

3.1. Common Properties

Property Name	Type	Description
`type` (required)	`string`	Indicates that this object is a CybOX Object. The value of this property **MUST** be a valid CybOX object type name.
`description` (optional)	`string`	Specifies a textual description of the Object.
`extended_properties` (optional)	`dictionary`	The set of extended properties specified for the Object, such as those defined for a specific scope.

		Specifies any extended properties of the object, as a dictionary. Dictionary keys **MUST** identify the extension type by name. The corresponding dictionary values **MUST** contain the contents of the extension instance.

3.2. IDs and References

<todo need a section that talks about these>

3.3. Object Property Metadata

3.3.1. String Encoding

Capturing the observed encoding of a particular Object string is useful for attribution, the creation of indicators, and related use cases.

Every string property in a CybOX Object must be represented by one of two allowable field names - the field name without any suffix, known as the “base name property” (e.g., **file_name**) or a base64-encoded representation of the string contained in a field with a suffix of “_b64” (e.g., **file_name_b64**). The base name property **MUST** be a unicode representation of the string. If the base name property is not specified, the base64-encoded property **MUST** be specified. If the property is required, either the base name property or the “_b64” version **MUST** be specified, but both **MUST NOT** be specified. If required, an additional field with the same base name and a suffix of “_enc” can be specified that captures the observed encoding of the property value. All “_enc” properties MUST specify their encoding using the corresponding name from the 2013-12-20 revision of the IANA character set registry. If the preferred MIME name for a character set is defined, this value **MUST** be used; if it is not defined, then the Name value from the registry **MUST** be used instead.

As an example of how this is defined in a CybOX Object, the **file_name** property in the `file-system-properties` type of the File Object has two sibling properties: **file_name_enc**,

for capturing the observed encoding of the file name string and also **file_name_b64**, for capturing the native representation of the string. Note that as described above, **file_name** and **file_name_b64** are mutually exclusive and cannot be used together in a File Object instance.

Examples

File with unicode representation of the filename and a corresponding encoding specification

```
{
  "type":"cybox-container",
  "spec_version":"3.0",
  "objects":{
    "0":{
      "objects":{
        "type":"file-object",
        "hashes":{
          "md5":"66e2ea40dc71d5ba701574ea215a81f1"
        },
        "file_system_properties":{
          "type":"file-system-properties",
          "is_directory":false,
          "file_name":"quêry.dll",
          "file_name_enc":"windows-1252"
        }
      }
    }
  }
}
```

File with base64 encoded filename and corresponding encoding specification

```
{
  "type":"cybox-container",
  "spec_version":"3.0",
  "objects":{
    "0":{
      "type":"file-object",
      "hashes":{
        "md5":"66e2ea40dc71d5ba701574ea215a81f1"
      },
      "file_system_properties":{
        "type":"file-system-properties",
        "is_directory":false,
        "file_name_b64":"cXXqcnkuZGxs",
        "file_name_enc":"windows-1252"
      }
    }
```

```
  }
}
```

3.4. Object Relationships

A CybOX Object relationship is a connection between two or more CybOX Objects within the scope of a given CybOX Container. CybOX relationships are references that are represented as properties of a CybOX Object, containing the IDs of the target CybOX Object(s).

CybOX relationships may be either singletons or lists. In the case of singleton relationships, the name of their Object property ends in **_ref**, whereas for list relationships the name of their Object property ends in **_refs**.

The target(s) of CybOX relationships may be restricted to a subset of CybOX Object types, as specified in the description of the property that defines the relationship. For example, the **belongs_to_refs** property on the IPv4 Address Object specifies that the *only* valid target of the relationship is an AS Object.

Example

Network Connection with Source/Destination IPv4 Addresses and AS

```
{
  "type":"cybox-container",
  "spec_version":"3.0",
  "objects":{
    "0":{
      "type":"ipv4-address-object",
      "value":"1.2.3.4",
      "belongs_to_refs":["3"]
    },
    "1":{
      "type":"ipv4-address-object",
      "value":"2.3.4.5"
    },
    "2":{
      "type":"network-connection-object",
      "src_refs":["0"],
      "dst_refs":["1"],
    }
    "3":{
      "type":"as-object",
      "number":42
    }
  }
}
```

3.5. Object Extensions

Each CybOX Object may have one or more extensions defined for it. (As previously stated, all CybOX extensions are structs with one or more named properties.)

3.5.1. Predefined Extensions

Each predefined extension can be defined at most once on a given Object. In an Object instance, each extension is specified under the **`extended_properties`** property, which is of type `dictionary`. Note that this means that each extension is specified through a corresponding key in the `extended_properties` property. For example, when specified in a File Object instance, the file metadata extension would be specified using the key value of `metadata`.

Example

```
{
  "type":"cybox-container",
  "spec_version":"3.0",
  "objects":{
    "0":{
      "type":"file-object",
      "hashes":{
        "md5":"3773a88f65a5e780c8dff9cdc3a056f3"
      },
      "size":25537,
      "extended_properties":{
        "metadata":{
          "mime_type":"vnd.microsoft.portable-executable"
        },
        "ntfs":{
          "sid":"1234567"
        }
      }
    }
  }
}
```

4. CybOX Container

Type Name: `cybox-container`

CybOX Containers capture a set of related CybOX Objects or Actions in their corresponding **objects** or **actions** properties, respectively.

The key in each **objects** or **actions** property specifies the identifier for a given CybOX Object or Action within the local scope of its parent `cybox-container`. Accordingly, each key **SHOULD** be a non-negative integer monotonically increasing, incrementing by 1 from a starting value of 0, and represented as a string within the JSON MTI serialization. However, implementers **MAY** elect to use an alternate key format, provided that it complies with the constraints as stipulated by the CybOX `dictionary` data type for keys.

A `cybox-container` must not be empty - that is, one of either the **objects** or **actions** properties **MUST** be included. (Note that the **actions** property is currently marked as reserved for future use in the `cybox-container` definition.)

Property Name	Type	Description
type (required)	`string`	Indicates that this object is a CybOX Container. The value of this property **MUST** be `cybox-container`.
spec_version (required)	`string`	Indicates the version of the CybOX specification that the entities captured in the container conform to.
objects (optional)	`dictionary`	A dictionary of CybOX Objects captured in the container.
actions (optional)	`dictionary`	`RESERVED FOR FUTURE USE`

Examples

```
{
  "type":"cybox-container",
  "spec_version":"3.0",
  "objects":{
    "0":{
      "type":"email-address-object",
      "value":"jdoe@machine.example",
      "display_name":"John Doe"
    },
    "1":{
      "type":"email-address-object",
      "value":"mary@example.net",
      "display_name":"Mary Smith"
```

```
    },
    "2":{
      "type":"email-message-object",
      "header":{
        "from_ref":"0",
        "to_refs":["1"],
        "date":"1997-11-21T15:55:06Z",
        "subject":"Saying Hello"
      }
    }
  }
}
```

5. Common Vocabularies

5.1. Hashing Algorithm Vocabulary

Type Name: `hash-algo-ov`

An open vocabulary of hashing algorithms.

When specifying a hashing algorithm not already defined within the `hash-algo-ov`, the hashing algorithm name **SHOULD** be downcased, with dashes substituted for whitespace.

Vocabulary Value	Description
`md5`	Specifies the MD5 message digest algorithm. The corresponding hash string for this value **MUST** be a valid MD5 message digest as defined in RFC 1321.
`md6`	Specifies the MD6 message digest algorithm. The corresponding hash string for this value **MUST** be a valid MD6 message digest as defined in the MD6 proposal.
`ripemd-160`	Specifies the RIPEMD-160 (RACE Integrity Primitives Evaluation Message Digest) cryptographic hash function. The corresponding hash string for this value **MUST** be a valid RIPEMD-160 message digest as defined in the RIPEMD-160 specification.
`sha-1`	Specifies the SHA-1 (secure-hash algorithm 1) cryptographic hash function. The corresponding hash string for this value **MUST** be a valid SHA-1 message digest as defined in RFC 3174.

`sha-224`	Specifies the SHA-224 cryptographic hash function (part of the SHA2 family). The corresponding hash string for this value **MUST** be a valid SHA-224 message digest as defined in RFC 6234.
`sha-256`	Specifies the SHA-256 cryptographic hash function (part of the SHA2 family). The corresponding hash string for this value **MUST** be a valid SHA-256 message digest as defined in RFC 6234.
`sha-384`	Specifies the SHA-384 cryptographic hash function (part of the SHA2 family). The corresponding hash string for this value **MUST** be a valid SHA-384 message digest as defined in RFC 6234.
`sha-512`	Specifies the SHA-512 cryptographic hash function (part of the SHA2 family). The corresponding hash string for this value **MUST** be a valid SHA-512 message digest as defined in RFC 6234.
`sha3-224`	Specifies the SHA3-224 cryptographic hash function. The corresponding hash string for this value **MUST** be a valid SHA3-224 message digest as defined in FIPS PUB 202.
`sha3-256`	Specifies the SHA3-256 cryptographic hash function. The corresponding hash string for this value **MUST** be a valid SHA3-256 message digest as defined in FIPS PUB 202.
`sha3-384`	Specifies the SHA3-384 cryptographic hash function. The corresponding hash string for this value **MUST** be a valid SHA3-384 message digest as defined in FIPS PUB 202.
`sha3-512`	Specifies the SHA3-512 cryptographic hash function. The corresponding hash string for this value **MUST** be a valid SHA3-512 message digest as defined in FIPS PUB 202.
`ssdeep`	Specifies the ssdeep fuzzy hashing algorithm. The corresponding hash string for this value **MUST** be a valid piecewise hash as defined in the SSDEEP specification.
`whirlpool`	Specifies the whirlpool cryptographic hash function. The corresponding hash string for this value **MUST** be a valid WHIRLPOOL message digest as defined in ISO/IEC 10118-3:2004.

5.2. Encryption Algorithm Vocabulary

Type Name: `encryption-algo-ov`

An open vocabulary of encryption algorithms.

When specifying an encryption algorithm not already defined within the `encryption-algo-ov`, the encryption algorithm name **SHOULD** be downcased, with dashes substituted for whitespace.

Vocabulary Value	Description
`aes128-ecb`	Specifies the Advanced Encryption Standard (AES), as a defined in NIST SP 800-38A.
`aes128-cbc`	Specifies the Advanced Encryption Standard (AES), as a defined in NIST SP 800-38A.
`aes128-cfb`	Specifies the Advanced Encryption Standard (AES), as a defined in NIST SP 800-38A.
`aes128-cofb`	Specifies the Advanced Encryption Standard (AES), as a defined in NIST SP 800-38A.
`aes128-ctr`	Specifies the Advanced Encryption Standard (AES), as a defined in NIST SP 800-38A.
`aes128-xts`	Specifies the Advanced Encryption Standard (AES), as a defined in NIST SP 800-38E.
`aes128-gcm`	Specifies the Advanced Encryption Standard (AES), as a defined in NIST SP 800-38D.
`salsa20`	Specified in Salsa 20 specification published by Daniel J. Bernstein
`salsa12`	Specified in Salsa20/8 and Salsa20/12 published by Daniel J. Bernstein.
`salsa8`	Specified in Salsa20/8 and Salsa20/12 published by Daniel J. Bernstein.
`chacha20-poly1305`	Specified in RFC 7539.
`chacha20`	Specified in RFC 7539, but without the poly1305 authentication.
`des-cbc`	
`3des-cbc`	
`des-ebc`	
`3des-ebc`	

`cast128-cbc`	https://en.wikipedia.org/wiki/CAST-128
`cast256-cbc`	https://en.wikipedia.org/wiki/CAST-256

6. Customizing CybOX

There are two means to customize CybOX: custom object extensions, and custom objects. Custom object extensions provide a mechanism and requirements for the specification of properties not defined by this specification (including relationships) on CybOX Objects. Custom objects, on the other hand, provides a mechanism and requirements to create custom CybOX Objects (objects not defined by this specification).

6.1. Custom Objects

There will be cases where certain information exchanges can be improved by adding objects that are not specified nor reserved in this document; these objects are called Custom Objects. This section provides guidance and requirements for how producers can use Custom Objects and how consumers should interpret them in order to extend CybOX in an interoperable manner.

6.1.1. Requirements

- Producers **MAY** include any number of Custom Objects in STIX documents.
- The type field in a Custom Object **MUST** be in ASCII and **MUST** only contain the characters a-z (lowercase ASCII), 0-9, and hyphen (-).
- The type field **MUST NOT** contain a hyphen (-) character immediately following another hyphen (-) character.
- Custom Object names **MUST** have a minimum length of 3 ASCII characters.
- Custom Object names **MUST** be no longer than 250 ASCII characters in length.
- The value of the `type` field in a Custom Object **SHOULD** start with “x-” followed by a source unique identifier (like a domain name with dots replaced by dashes), a dash and then the name. For example: `x-example-com-customobject`.
- A Custom Object whose name is not prefixed with “x-” **MAY** be used in a future version of the specification with a different meaning. Therefore, if compatibility with future versions of this specification is required, the “x-” prefix **MUST** be used.

Custom Objects **SHOULD** only be used when there is no existing CybOX Object defined by the CybOX specification that fulfills that need.

Example

6.2. Custom Object Extensions

In addition to the predefined Object extensions specified in this document, CybOX supports user-defined Custom extensions. As with predefined CybOX extensions, custom extension data is conveyed under the `extended_properties` property.

6.2.1. Requirements

- A CybOX Object **MAY** have any number of Custom Extensions.
- Custom Extension names **MUST** be in ASCII and are limited to characters a-z (lowercase ASCII), 0-9, underscore (_), and dash (-).
- Custom Extension names **SHOULD** start with "x_" followed by a source unique identifier (like a domain name), an underscore and then the name. For example: `x_examplecom_customextension`.
- Custom Extension names **MUST** have a minimum length of 3 ASCII characters.
- Custom Extension names **MUST** be no longer than 250 ASCII characters in length.
- Custom Extension names that are not prefixed with "x_" may be used in a future version of the specification for a different meaning. If compatibility with future versions of this specification is required, the "x_" prefix **MUST** be used.
- Custom Extensions **SHOULD** only be used when there is no existing extension defined by the CybOX specification that fulfills that need.

Examples

Custom File Object Relationship

```
{
  "type":"cybox-container",
  "spec_version":"3.0",
  "objects":{
    "0":{
      "type":"file-object",
      "hashes":{
        "md5":"66e2ea40dc71d5ba701574ea215a81f1"
      }
    }
  },
  "1":{
    "type":"file-object",
    "hashes":{
```

```
      "md5":"9B996B8785BFC7C857FF346931FC4B51"
    },
    "x_foo_ref":"0"
  }
}
```

7. Reserved Names

This section defines names that are reserved for future use in revisions of this document. The names defined in this section **MUST NOT** be used for the name of any Custom Object or Property.

The following object names are reserved:

- `action`

8. Appendix A. Acknowledgments

CybOX Subcommittee Chairs:
Trey Darley (trey@kingfisherops.com), Kingfisher Operations, sprl
Ivan Kirillov, (ikirillov@mitre.org), MITRE Corporation

Contributors
The following individuals made substantial contributions in the form of normative text and proofing of this specification, their contributions are gratefully acknowledged:

- Bret Jordan, Blue Coat Systems, Inc.
- Terry MacDonald, Cosive
- Jane Ginn, Cyber Threat Intelligence Network, Inc. (CTIN)
- Trey Darley, Kingfisher Operations, sprl
- Jason Keirstead, IBM
- Allan Thomson, LookingGlass Cyber
- John-Mark Gurney, New Context Services, Inc.
- Christian Hunt, New Context Services, Inc.
- Greg Back, MITRE Corporation
- Sean Barnum, MITRE Corporation
- Ivan Kirillov, MITRE Corporation

- John Wunder, MITRE Corporation