STIX 2.0 Specification

Part 3a: Cyber Observable Core Concepts - Version 2.0-rc3

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

Additional Artifacts

This prose specification is one component of a Work Product, which consists of:

- STIX Version 2.0 Part 1: STIX Core Concepts
- STIX Version 2.0 Part 2: STIX Objects
- STIX Version 2.0 Part 3a: Cyber Observable Core Concepts (this document)
- STIX Version 2.0 Part 3b: Cyber Observable Objects
- STIX Version 2.0 Part 4: STIX Patterning

Table of Contents

1. Introduction

Parts 3a and 3b of the STIX specification define structured representations for observable objects and their properties in the cyber domain. These 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

STIX Cyber Observables document the facts concerning **what** happened on a network or host, but not necessarily the who or when, and never the why. For example, information about a file that existed, a process that was observed running, or that network traffic occurred between two IPs can all be captured as Cyber Observable data.

STIX Cyber Observables are used by various STIX SDOs, which provide additional context to the data. The Observed Data SDO, for example, indicates that the raw data was observed at a particular time and by a particular party. Outside of STIX, Observable Data can be used similarly to describe data collected from a malware or forensic analysis.

Previously, this part of the STIX specification was a separate specification known as Cyber Observable eXpression (CybOX). In response to lessons learned in implementing previous versions, the specification has been significantly redesigned and, as a result, omits many of the objects, types, and properties defined in CybOX 2.1.1. It has also been merged into the STIX specification, as the Cyber Observable layer, to improve integration and ease the burden of implementing both. The Cyber Observable Objects chosen for inclusion in STIX 2.0 represent a minimally viable product (MVP) that fulfills basic consumer and producer requirements. Objects and properties not included in STIX 2.0, but deemed necessary by the community, will be included in future releases.

This document (Part 3a) in the STIX specification describes Cyber Observable Core Concepts. CybOX Objects (Part 3b) contains the definitions for the Cyber Observable Objects.

1.1. Overview

1.1.1. Cyber Observable Objects

STIX 2.0 defines a set of Cyber Observable Objects for characterizing host-based, network, and related entities. Each of these objects correspond to a data point commonly represented in CTI and forensics. Using the building blocks of Cyber Observable Objects alongside relationships between these objects, individuals can create, document, and share broad and comprehensive information about computer systems and their state.

Throughout STIX 2.0 Part 3a and 3b, Cyber Observable Objects are referred to simply as "Observable Objects". These should not be confused with STIX Domain Objects (SDOs), as defined in STIX 2.0, Part 1 and 2.

1.1.2. Cyber Observable Relationships

A Cyber Observable Relationship is a reference linking two (or more) related Cyber Observable Objects. Cyber Observable Relationships are only resolvable within the same Observable Objects container. References are a property on Cyber Observable Objects that contain the ID of a different Cyber Observable Object.

Throughout STIX 2.0 Part 3a and 3b, Cyber Observable Relationships are referred to simply as "Relationships". These should not be confused with STIX Relationship Objects (SROs), as defined in STIX 2.0, Part 1 and 2.

1.1.3. Cyber Observable Extensions

Each Observable 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 Objects. To enable this, STIX permits the specification of such additional properties through the set of default Object Extensions. Where applicable, extensions are included in the definitions of Objects. Producers may extend the Cyber Observable Objects by defining their own custom extensions or properties. (For further information, refer to §5 (Customizing Cyber Observable Objects.)

1.1.4. Vocabularies & Enumerations

Many Objects contain properties whose values are constrained by a predefined enumeration or open vocabulary. In the case of enumerations, this is a requirement that producers must use the values in the enumeration and cannot use any outside values. 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, as an example, 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. Conventions

1.2.1. 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 Cyber Observable Object 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.

1.2.2. 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 - **`protocols`**
  - 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. Cyber Observable Specific Data Types

The Cyber Observable specification within STIX makes use of many common types that are defined in STIX 2.0 Part 1, STIX Core, §2. In addition, data types specific to the representation of Cyber Observables are defined in this section. The table below lists common data types from

STIX Core with a gray background and the Cyber Observable specific types with a white background.

Type	Description
`boolean`	A value of `true` or `false`.
`float`	An IEEE 754 double-precision number.
`integer`	A whole number.
`list`	An ordered sequence of values. The phrasing "`list` of type `<type>`" is used to indicate that all values within the list must conform to a specific type.
`open-vocab`	A value from a STIX open (`open-vocab`) or suggested vocabulary.
`string`	A series of Unicode characters.
`timestamp`	A time value (date and time).
`binary`	A sequence of bytes.
`hex`	An array of octets as hexadecimal.
`dictionary`	A set of key/value pairs.
`object-ref`	A local reference to a Cyber Observable Object.
`hashes-type`	One or more cryptographic hashes.
`observable-objects`	One or more Cyber Observable Objects.

2.2. 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 serializations **SHOULD** use a native binary type, if available.

2.3. 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

```
...
   "src_flags_hex": "00000002"
...
```

2.4. Dictionary

Type Name: `dictionary`

A `dictionary` captures a set of key/value pairs. `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 (_). `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.

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

2.5. Object Reference

Type Name: `object-ref`

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

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

```
{
  "0": {
    "type": "ipv4addr",
    "value": "1.2.3.4"
```

```
  },
  "1":{
    "type": "network-traffic",
    "dst_ref": "0"
  }
}
```

2.6. Hashes Type

Type Name: `hashes-type`

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

Examples
MD5 and Custom Hash

```
    {
        "MD5": "3773a88f65a5e780c8dff9cdc3a056f3",
        "x_foo_hash": "aaaabbbbccccddddeeeeffff0123457890"
    }
```

2.7. Observable Objects

Type Name: `observable-objects`

The Observable Objects type represents 1 or more Observable Objects as a `dictionary`. The keys in the dictionary are references used to refer to the values, which are objects. Each key in the dictionary **SHOULD** be a non-negative monotonically increasing integer, 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 `dictionary` data type for keys.

Examples

```
{
    "0": {
      "type": "email-addr",
      "value": "jdoe@machine.example",
      "display_name": "John Doe"
    },
```

```
    "1": {
      "type": "email-addr",
      "value": "mary@example.net",
      "display_name": "Mary Smith"
    },
    "2": {
      "type": "email-message",
      "from_ref": "0",
      "to_refs": ["1"],
      "date": "1997-11-21T15:55:06Z",
      "subject": "Saying Hello"
    }
  }
}
```

3. Cyber Observable Objects

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

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

3.1. Common Properties

Property Name	Type	Description
type (required)	`string`	Indicates that this object is an Observable Object. The value of this property **MUST** be a valid Observable Object type name.
description (optional)	`string`	Specifies a textual description of the Object.
extended_properties (optional)	`dictionary`	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. Object References

Identifiers on Observable Objects are specified as keys in the `observable-objects` type. For more information on how such keys may be defined, see Section 2.14.

The `object-ref` type is used to define Observable Object properties that are *references* to other Observable Objects (such as the **`src_ref`** property on the Network Traffic Object). *Resolving* a reference is the process of identifying and obtaining the actual Observable Object referred to by the reference field. References resolve to an object when the value of the property (e.g., **`src_ref`**) is an exact match with the `key` of another Observable Object that resides in the same parent container as the Observable Object that specifies the reference. This specification does not address the implementation of reference resolution.

3.3. Object Property Metadata

3.3.1. String Encoding

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

Every string property in an Observable 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., **`name`** in the File Object) or a base64-encoded representation of the string contained in a field with a suffix of “_b64” (e.g., **`name_b64`** in the File Object). 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 Object, the **`name`** property in the `file-system-properties` type of the File Object has two sibling properties: **`name_enc`**, for capturing the observed encoding of the file name string and also **`name_b64`**, for capturing the native representation of the string. Note that as described above, **`name`** and **`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

```
{
  "0": {
    "type": "file",
    "hashes": {
      "MD5": "66e2ea40dc71d5ba701574ea215a81f1"
    },
    "name": "quêry.dll",
    "name_enc": "windows-1252"
  }
}
```

File with base64 encoded filename and corresponding encoding specification

```
{
  "0": {
    "type": "file",
    "hashes": {
      "MD5": "66e2ea40dc71d5ba701574ea215a81f1"
    },
    "name_bin": "cXXqcnkuZGxs",
    "name_enc": "windows-1252"
    }
  }
}
```

3.4. Object Relationships

A Cyber Observable Relationship is a connection between two or more Cyber Observable Objects within the scope of a given Observable Objects dictionary. Cyber Observable relationships are references that are represented as properties of a Cyber Observable Object, containing the keys of the target Cyber Observable Object(s).

Cyber Observable Object 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 Cyber Observable relationships may be restricted to a subset of Cyber Observable 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 Traffic with Source/Destination IPv4 Addresses and AS

```
{
  "0": {
    "type": "ipv4-addr",
    "value": "1.2.3.4",
    "belongs_to_refs": ["3"]
  },
  "1": {
    "type": "ipv4-addr",
    "value": "2.3.4.5"
  },
  "2": {
    "type": "network-traffic",
    "src_ref": "0",
    "dst_ref": "1",
  }
  "3": {
    "type": "as"
    "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 NTFS extension would be specified using the key value of `ntfs-ext`.

Example

```
{
  "0": {
    "type": "file",
```

```
    "hashes": {
      "MD5": "3773a88f65a5e780c8dff9cdc3a056f3"
    },
    "size": 25537,
    "extended_properties": {
      "ntfs-ext": {
        "sid": "1234567"
      }
    }
  }
}
```

4. Common Vocabularies

4.1. 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`, wherever an authoritative name for a hashing algorithm name is defined, it should be used as the value. In cases where an there is variance in the name of a hashing algorithm, producers should exercise their best judgement.

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.

4.2. Encryption Algorithm Vocabulary

Type Name: `encryption-algo-ov`

An open vocabulary of encryption algorithms.

When specifying an encryption algorithm not already defined within the `hash-algo-ov`, wherever an authoritative name for an encryption algorithm name is defined, it should be used as the value. In cases where an there is variance in the name of a encryption algorithm, producers should exercise their best judgement.

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 8I00-38D.
`Salsa20`	Specifies the Salsa20 stream cipher, as defined in the Salsa 20 specification.
`Salsa12`	Specifies the Salsa20/12 stream cipher as defined in the Salsa20/8 and Salsa20/12 specification.
`Salsa8`	Specifies the Salsa20/8 stream cipher as defined in the Salsa20/8 and Salsa20/12 specification.
`ChaCha20-Poly1305`	Specifies the ChaCha20-Poly1305 stream cipher, as defined in RFC 7539.

`ChaCha20`	Specifies the ChaCha20 stream cipher (without poly1305 authentication), as defined in RFC 7539.
`DES-CBC`	Specifies the Data Encryption Standard algorithm with Cipher Block Chaining (CBC) mode, as defined in FIPS PUB 81.
`3DES-CBC`	Specifies the Triple Data Encryption Standard algorithm with Cipher Block Chaining (CBC) mode, as defined in TBD.
`DES-EBC`	Specifies the Data Encryption Standard algorithm with Electronic Codebook (ECB) mode, as defined in FIPS PUB 81.
`3DES-EBC`	Specifies the Triple Data Encryption Standard algorithm with Electronic Codebook (ECB) mode, as defined in NIST SP 800-67.
`CAST128-CBC`	Specifies the CAST-128 algorithm with Cipher Block Chaining (CBC) mode, as defined in RFC 2144.
`CAST256-CBC`	Specifies the CAST-256 algorithm with Cipher Block Chaining (CBC) mode, as defined in RFC 2612.
`RSA`	Specifies the RSA symmetric encryption algorithm, as defined by RFC 2313.
`DSA`	Specifies the Digital Signature Algorithm, as defined by FIPS 186-4.

5. Customizing Cyber Observable Objects

There are three means to customize Cyber Observable Objects: custom object extensions, custom observable objects, and custom properties. Custom object extensions provide a mechanism and requirements for the specification of extensions not defined by this specification (including relationships) on Observable Objects. Custom observable objects provides a mechanism and requirements to create custom Observable Objects (objects not defined by this specification). Custom properties, as in the rest of STIX, provide a mechanism to add individual properties anywhere in the data model.

Custom properties **SHOULD** be used for cases where it is necessary to add one or more simple additional properties on an Observable Object, with the expectation that these properties are unlikely to be standardized in the future. On the other hand, custom Object extensions **SHOULD** be used for cases where it is necessary to describe more complex additional properties (i.e., those with multiple potential levels of hierarchy) that may end up being standardized in the future. As an example, a property that expresses some custom threat score for a File Object should be added directly to the Observable Object as a custom property, whereas a set of

properties that represent metadata necessary to add support for a new file system to the File Object should be done as a custom extension.

A consumer that receives a STIX document containing Custom Cyber Observable Properties or Objects it does not understand **MAY** refuse to process the document or **MAY** ignore those properties or objects and continue processing the document.

5.1. Custom Observable 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 Observable Objects. This section provides guidance and requirements for how producers can use Custom Observable Objects and how consumers should interpret them in order to extend STIX in an interoperable manner.

5.1.1. Requirements

- Producers **MAY** include any number of Custom Observable Objects in an Observable Objects dictionary.
- The type field in a Custom Observable 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 Observable Object names **MUST** have a minimum length of 3 ASCII characters.
- Custom Observable Object names **MUST** be no longer than 250 ASCII characters in length.
- The value of the `type` field in a Custom Observable 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 Observable 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.
- A Custom Observable Object **MUST** have one or more Custom Properties:
    - Custom Property names **MUST** be in ASCII and **MUST** only contain the characters a–z (lowercase ASCII), 0–9, and underscore (_).
    - Custom Property names **MUST** have a minimum length of 3 ASCII characters.
    - Custom Property names **MUST** be no longer than 250 ASCII characters in length.
- Custom Observable Objects **SHOULD** only be used when there is no existing Observable Object defined by the STIX specification that fulfills that need.
- Custom Observable Object property values **MUST** be a valid primitive, type, or a homogenous list of types.

Examples

Simple Custom Observable Object

```
{
  "0": {
    "type": "x-example",
    "foo": "bar",
    "vals": ["this",
             "is",
             "an",
             "example"]
  }
}
```

5.2. Custom Object Extensions

In addition to the predefined Cyber Observable Object extensions specified in this document, STIX supports user-defined custom extensions. As with predefined extensions, custom extension data MUST be conveyed under the **`extended_properties`** property.

5.2.1. Requirements

- An Observable 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, and dash (-).
- Custom Extension names **SHOULD** start with “x-” followed by a source unique identifier (like a domain name), a dash 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.
- A Custom Extension **MUST** have one or more Custom Properties:
    - Custom Property names **MUST** be in ASCII and **MUST** only contain the characters a–z (lowercase ASCII), 0–9, and underscore (_).
    - Custom Property names **MUST** have a minimum length of 3 ASCII characters.
    - Custom Property names **MUST** be no longer than 250 ASCII characters in length.

Examples

Custom File Object Extension

```
{
  "0": {
    "type": "file",
    "hashes": {
      "MD5": "9B996B8785BFC7C857FF346931FC4B51"
    },
    "extended_properties": {
      "x-example-com-foo": {
        "foo_val": "foo",
        "bar_val": "bar"
      }
    }
  }
}
```

6. Custom Object Properties

There will be cases where certain information exchanges can be improved by adding properties that are neither specified nor reserved in this document; these properties are called Custom Properties. This section provides guidance and requirements for how producers can use Custom Properties and how consumers should interpret them in order to extend Cyber Observable Objects in an interoperable manner.

6.1. Requirements

- A Cyber Observable Object **MAY** have any number of Custom Properties.
- Custom Property names **MUST** be in ASCII and MUST only contain the characters a–z (lowercase ASCII), 0–9, and underscore (_).
- Custom Property names **SHOULD** start with “x_” followed by a source unique identifier (such as a domain name with dots replaced by underscores), an underscore and then the name. For example, **`x_example_com_customfield`**.
- Custom Property names **MUST** have a minimum length of 3 ASCII characters.
- Custom Property names **MUST** be no longer than 250 ASCII characters in length.
- Custom Property names that do not start 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 Properties **SHOULD** only be used when there is no existing properties defined by the Cyber Observable specification that fulfils that need.
- Custom Properties **SHOULD** only be used to define simple properties (e.g., those of string or integer type)
- For Custom Properties that use the `hex` type, the property name **MUST** end with '_hex'.

- For Custom Properties that use the `binary` type, the property name **MUST** end with '_bin'.

Examples

File Object with Custom Properties

```
{
  "0": {
    "type": "file",
    "hashes": {
      "MD5": "9B996B8785BFC7C857FF346931FC4B51"
    },
    "x_example_com_foo": "bar",
    "x_example_com_bar": 27
  }
}
```

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 Observable Object or Property.

The following object names are reserved:

- `action`

8. Conformance

8.1. Producers and Consumers

A "Cyber Observable Producer" is any software that creates Cyber Observable content and conforms to the following normative requirements:

1. It **MUST** be able to create content encoded as JSON.
2. All required properties **MUST** be present in the created content.
3. All properties **MUST** conform to the data type and normative requirements for that property.
4. It **MUST** support at least one Defined Cyber Observable Object per the Conformance section in Part 3b, Cyber Observable Objects.

A "Cyber Observable Consumer" is any software that consumes Cyber Observable content and conforms to the following normative requirements:

1. It **MUST** support parsing all required properties for the content that it consumes.

9. 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
- Dave Cridland, Surevine

Appendix B. Revision History