STIX™ 2.0 Specification

Part 5: STIX Patterning - Version 2.0-rc4

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 3: Cyber Observable Core Concepts
- STIX Version 2.0 Part 4: Cyber Observable Objects
- STIX Version 2.0 Part 5: STIX Patterning (this document)

Table of Contents

1. Introduction

In order to detect a large proportion of malicious behaviour in the course of defending our networks it is necessary to correlate telemetry from both host-based and network-based tools. Before undertaking work on STIX Patterning, as a technical subcommittee we made a thorough effort to evaluate whether there was already an existing patterning language that would support our use cases available as an open standard. In particular, we considered whether it would be possible to extend the syntax of Snort or Yara rather than create an entirely new language. This was eventually ruled out as unfeasible, both from a technical perspective as well as taking into consideration that from a licensing/IPR perspective, extending either of those languages under the auspices of OASIS would have been problematic.

Given that STIX Patterning exists to support STIX Indicators, consider what value Indicator-sharing provides: a mechanism for communicating how to find malicious code and/or threat actors active within a given network. Among the essential tools widely deployed by defenders are SIEMs (or similar data processing platforms capable of consuming, correlating, and interrogating large volumes of network and host-based telemetry.) These data processing platforms utilize proprietary query languages. As development began on STIX Patterning, one of the principal design goals was to create an abstraction layer capable of serializing these proprietary correlation rules so as to enhance the overall value proposition of indicator-sharing.

In order to enhance detection of possibly malicious activity on networks and endpoints, a standard language is needed to describe what to look for in an cyber environment. The patterning language allows matching against timestamped Cyber Observable data (such as STIX Observed Data Objects) collected by a threat intelligence platform or other similar system so that other analytical tools and systems can be configured to react and handle incidents that might arise.

This first language release is focused on supporting a common set of use cases and therefore allows for the expression of an initial set of patterns that producers and consumers of STIX can utilize. As more complex patterns are deemed necessary, the STIX patterning language will be extended in future releases to improve its effectiveness as an automated detection/remediation method.

2. Definitions

The terms defined below are used throughout this document. (Related terms are grouped by color.)

Terms	Definitions	Example
whitespace	Any Unicode code point that has WSpace set as a property, for example, line feeds, carriage returns, tabs, and spaces.	n/a
Observation	Observations represent data about systems or networks that is observed at a single point in time - for example, information about a file that existed, a process that was observed running, or network traffic that was transmitted between two IPs. In STIX, Observations are represented by Observed Data SDOs, and the **first_observed** timestamp defines the observation time.	n/a
Comparison Expression	Comparison Expressions are the basic components of Observation Expressions. They consist of an Object Path and a constant joined by a Comparison Operator (listed in Section 4.2.1, Comparison Operators).	`user-account:value = 'Peter'`
Comparison Operators	Comparison Operators are used within Comparison Expressions to compare an Object Path against a constant or set of constants.	`MATCHES`
Object Path	Object Paths define which properties of Cyber Observable Objects should be evaluated against as part of a Comparison Expression. Cyber	`ipv6-addr:value`

	Observable Objects and their properties are defined in STIX 2.0, Part 4.	

Observation Expression	Observation Expressions consist of one or more Comparison Expressions joined with Boolean Operators and surrounded by square brackets.
	
	An Observation Expression may consist of two Observation Expressions joined by a Boolean Operator or Observation Operator. This can be applied recursively to compose multiple Observation Expressions into a single Observation Expression.
	
	Observation Expressions may optionally be followed by one or more Qualifiers further constraining the result set. When a Qualifier is needed to be applied to all of the Observation Expressions joined with Observation Operators, parentheses may be used around the Observation Expressions, but before the Qualifier.
		`[ipv4-addr:value = '203.0.113.1' OR ipv4-addr:value = '203.0.113.2']`
		
		or (with Boolean Operator):
		
		`([ipv4-addr:value = '198.51.100.1'] AND [ipv4-addr:value = '198.51.100.2'])`
		
		or (with Observation Operator):
		
		`([ipv4-addr:value = '198.51.100.5'] FOLLOWEDBY [ipv4-addr:value = '198.51.100.10'])`
		
		or (with Boolean Operator and Qualifier):
		
		`([ipv4-addr:value = '198.51.100.5' ] AND [ipv4-addr:value = '198.51.100.10']) WITHIN 300 SECONDS`

Boolean Operators	Operators including AND or OR used to combine Observation Expressions or Comparison Expressions within an Observation Expression.	(Comparison Expressions) `user-account:value = 'Peter' OR user-account:value = 'Mary'`
Qualifier	Qualifiers provide a restriction on the Observations that are considered valid for matching the preceding Observation Expression.	`[file:name = 'foo.dll'] START '2016-06-01T00:00:00Z' STOP '2016-07-01T00:00:00Z'`

Observation Operators	Observation Operators are used to combine two Observation Expressions operating on two different Observed Data instances into a single pattern.	`[ipv4-addr:value = '198.51.100.5'] ALONGWITH [ ipv4-addr:value = '198.51.100.10']`
Pattern Expression	A Pattern Expression represents a valid instance of a Cyber Observable pattern. A basic Pattern Expression consists of a single Observation Expression.	`[file:size = 25536]`

2.1. Constants

The data types enumerated below are supported as operands within Comparison Expressions. This table is included here as a handy reference for implementers.

Note that unlike Cyber Observable Objects (which are defined in terms of the MTI JSON serialization), STIX Patterns are Unicode strings, regardless of the underlying serialization, hence the data types defined in the table below in some cases differ from the definitions contained in Cyber Observable Core.

Each constant defined in Patterning has a limited set of Cyber Observable Data types that they are allowed to be compared against. In some cases, there are multiple Cyber Observable Data Types that could be compared against a STIX Patterning Constant; this is due to the fact that certain Cyber Observable Data Types are semantically indistinguishable because of their JSON serialization. The column, Cyber Observable Comparable Data Type(s) defines these limitations.

STIX Patterning Constant	Cyber Observable Comparable Data Type(s)	Description
`boolean`	`boolean`	A constant of `boolean` type encodes truth or falsehood. Boolean truth is denoted by the literal `true` and falsehood by the literal `false`.

`binary`	`binary`
	`hex`
	`string`
		A constant of `binary` type is a base64 encoded array of octets (8-bit bytes) per RFC 4648. The base64 string **MUST** be surrounded by single quotes (U+0027) and prefixed by a 'b' (U+0062). Line feeds in the base64 encoded data **MUST** be supported and ignored, but are not required to be inserted.

		Example: `b'ABI='`

`hex`	`binary`
	`hex`
	`string`
		A constant of `hex` 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'. The hex string **MUST** be surrounded by single quotes (U+0027) and prefixed by an 'h' (U+0068).
		Example:
		`h'0012'`

`integer`	`integer` `float`	A constant of `integer` type encodes a signed decimal number in the usual fashion (e.g., 123). In the case of positive integers, the integer **MUST** be represented as-is, omitting the ‘+' (U+002b). Negative integers **MUST** be represented by prepending a ‘-' (U+002d). When compared against a Cyber Observable `float`, the full value must be compared. The `float` must not be truncated. For example, the STIX Patterning constant integer 1 compared to a Cyber Observable `float` 1.5 is not equal. The valid range of values is defined by the STIX 2.0 specification, Part 3.
`float`	`integer` `float`	A constant of `float` type encodes a floating point number in the usual fashion (e.g., 123.456). In the case of positive floating point number, the floating point number **MUST** be represented as-is, omitting the ‘+' (U+002b). Negative floating point numbers **MUST** be represented by prepending a ‘-' (U+002d). The valid range of values is defined by the STIX 2.0 specification, Part 3.

`string`	`string`
	`binary`
	`hex`
		A constant of `string` type encodes a string as a list of Unicode code points surrounded by single quotes (U+0027).
		
		The escape character is the backslash (U+005c). Only the single quote or the backslash may follow, and in that case, the respective character is used for the sequence.
		
		If a string only contains codepoints less than (U+0100), then the string **MAY** be converted to a binary type value (if needed for comparison). The mapping is code point U+0000 to 00 through U+00ff to ff.

`timestamp`	`timestamp`	A constant of `timestamp` type encodes a STIX timestamp (as specified in STIX Version 2.0 Part 1: STIX Core Concepts, Section 2.10) as a string. The timestamp string **MUST** be surrounded by single quotes (U+0027) prefixed with a 't' (U+0074). Example: `t'2014-01-13T07:03:17Z'`

3. STIX Patterns

STIX Patterns are composed of multiple building blocks, ranging from simple key-value comparisons to more complex, contextual expressions. The most fundamental building block is the Comparison Expression, which is a comparison between a single property of a Cyber Observable Object and a provided constant using a Comparison Operator. As a very simple example, one might use the following Comparison Expression (contained within an Observation Expression) to match against an IPv4 address:

```
[ipv4-addr:value = '198.51.100.1/32']
```

Moving up a level of complexity, the next building block of a STIX Pattern is the Observation Expression, which consists of one or more Comparison Expressions joined by Boolean Operators and bounded by square brackets. An Observation Expression refines which set of Cyber Observable data (i.e., as part of an Observation) will match the pattern, by selecting the set that has the Cyber Observable Objects specified by the Comparison Expressions. An Observation Expression consisting of a single Comparison

Expression is the most basic valid STIX Pattern. Building upon the previous example, one might construct an Observation Expression to match against multiple IPv4 addresses and an IPv6 address:

```
[ipv4-addr:value = '198.51.100.1/32' OR ipv4-addr:value = '203.0.113.33/32' OR
ipv6-addr:value = '2001:0db8:dead:beef:dead:beef:dead:0001/128']
```

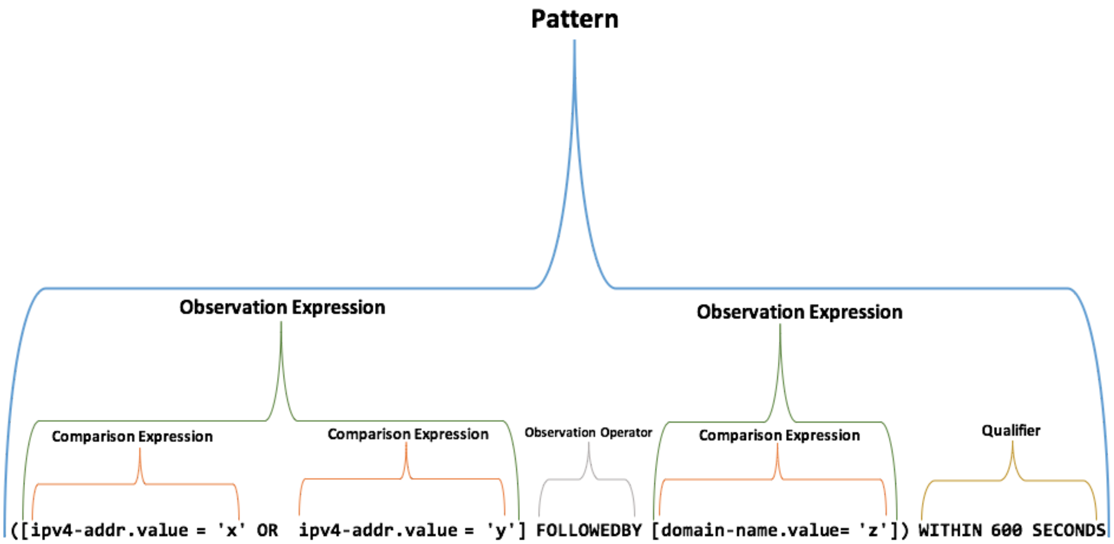
Observation Expressions may be followed by one or more Qualifiers, which allow for further restrictions on the set of data matching the pattern. Continuing with the above example, one might use a Qualifier to state that the IP addresses must be observed several times in repetition:

```
[ipv4-addr:value = '198.51.100.1/32' OR ipv4-addr:value = '203.0.113.33/32' OR
ipv6-addr:value = '2001:0db8:dead:beef:dead:beef:dead:0001/128'] REPEATS 5 TIMES
```

The final, highest level building block of STIX Patterning combines two or more Object Expressions via Observation Operators, yielding a STIX Pattern capable of matching across multiple STIX Observed Data SDOs. Building further upon our example, one might use an Observation Operator to specify that an observation of a particular domain name must follow the observation of the IP addresses (note the use of parentheses to encapsulate the two Observation Expressions), along with a different Qualifier to state that both the IP address and domain name must be observed within a specific time window:

```
([ipv4-addr:value = '198.51.100.1/32' OR ipv4-addr:value = '203.0.113.33/32' OR
ipv6-addr:value = '2001:0db8:dead:beef:dead:beef:dead:0001/128'] FOLLOWEDBY
[domain-name:value = 'example.com']) WITHIN 600 SECONDS
```

The diagram below depicts a truncated version of the above example.



4. Pattern Expressions

Pattern Expressions evaluate to true or false. They comprise one or more Observation Expressions joined by Observation Operators. Pattern Expressions are evaluated against a set of specific Observations. If one or more of those Observations match the Pattern Expression, then it evaluates to true. Otherwise, if no Observations match, the Pattern Expression evaluates to false.

Pattern Expressions **MUST** be encoded as Unicode strings.

Whitespace (i.e., Unicode code points where WSpace=Y) in the pattern string is used to delimit parts of the pattern, including keywords, constants, and field objects. Whitespace characters between operators, including line feeds and carriage returns, **MUST** be allowed. Multiple whitespace characters in a row **MUST** be treated as a single whitespace character.

An invalid pattern, due to parse error, invalid constants (e.g. invalid hex or binary constant) **MUST** not match any Observations. It is up to each implementation if this error is raised to the user.

4.1. Observation Expressions

Observation Expressions are comprised of one or more Comparison Expressions, joined via Boolean Operators.

Observation Expressions **MUST** be surrounded by square brackets ‘[‘ (U+005b) and ‘]' (U+005d). One or more Observation Expression Qualifiers **MAY** be provided after the closing square bracket or closing parenthesis of an Observation Expression. Observation Expressions **MAY** be joined by an Observation Operator.

Individual Observation Expressions (e.g., `[a = b]`) match against a single Observation, i.e., a single STIX Observed Data entity. In cases where matching against *multiple* Observations is required, two or more Observation Expressions may be combined via Observation Operators, indicating that two or more distinct Observations must be matched.

When matching an Observation against an Observation Expression, all Comparison Expressions contained within the Observation Expression **MUST** match against the same Cyber Observable Object including referenced objects. An Observation Expression **MAY** contain Comparison Expressions with Object Paths that are based on different **object-types**, but such Comparison Expressions **MUST** be joined by **OR**. The Comparison Expressions of an Observation Expression that use **AND** **MUST** use the same base Object Path, e.g., `file:`.

For example, the Pattern Expression `[(type-a:property-j = 'W' AND type-a:property-k = 'X') OR (type-b:property-m = 'Y' AND type-b:property-n = 'Z')]` can match an Observable with an object of either `type-a` or `type-b`, but both Comparison Expressions for that specific type must evaluate to True for the same object. The Comparison Expressions that are intended to match a single object can be joined by either **`AND`** or **`OR`**, e.g., `[type-a:property-j = 'W' AND type-a:property-k = 'X' OR type-a:property-l = 'Z']`. As **`AND`** has higher precedence than **`OR`**, the preceding example requires an Observation to have either both property-j = 'W' and property-k = 'X' or just property-l = 'Z'.

Observation Expressions along with their Observation Operators and optional Qualifiers **MAY** be surrounded with parenthesis to restrict which Observation Expressions the Qualifiers apply to. For example: `([ a ] ALONGWITH [ b ] REPEATS 5 TIMES) WITHIN 5 MINUTES` results in one `a` and 5 `b`'s that all match in a 5 minute period. As another example: `([ a ] ALONGWITH [ b ]) REPEATS 5 TIMES WITHIN 5 MINUTES` results in 5 `a`'s and 5 `b`'s all 10 Observations matching in a 5 minute period.

4.1.1. Observation Expression Qualifiers

Each Observation Expression **MAY** have additional temporal or repetition restrictions using the respective **`WITHIN`**, **`START`**/**`STOP`**, and **`REPEATED`** keywords.

Qualifiers	**Description**
a **REPEATS** *x* **TIMES**	*a* **MUST** be an Observation Expression or a preceding Qualifier. *a* **MUST** match exactly *x* times, where each match is a different Observation. *x* **MUST** be a positive integer. This is purely a shorthand way of writing *a* followed by x - 1 times ALONGWITH a. Example: `[ b ] FOLLOWEDBY [ c ] REPEATS 5 TIMES` In this example, the REPEATS applies to `c`, and it does not apply to `b`. The results will be `b` plus 5 `c`'s where all 5 `c`'s were observed after the `b`. (Note that there is only a single Qualifier in this example; more complex patterns may use more than one.) (Note that whether the counter gets reset upon a successful match is not addressed in this specification.)
a **WITHIN** *x* **SECONDS**	*a* **MUST** be an Observation Expression or a preceding Qualifier. All Observations matched by *a* **MUST** occur, or have been observed, within the specified time window. *x* **MUST** be a positive floating point value. If there is a set of two or more Observations matched

	by *a*, the most recent Observation timestamp contained within that set **MUST NOT** be equal to or later than the delta of the earliest Observation timestamp within the set plus the specified time window. Example: `([file:hashes."SHA-256" = '13987239847'] ALONGWITH [win-registry-key:key = 'hkey']) WITHIN 120 SECONDS` The above Pattern Expression looks for a file hash and a registry key that were observed within 120 seconds of each other. The parentheses are needed to apply the WITHIN Qualifier to both Observation Expressions.
a **START** *x* **STOP** *y*	*a* **MUST** be an Observation Expression or a preceding Qualifier. All Observations that match *a* **MUST** have an observation time >= *x* and < *y*. *x* and *y* **MUST** be a timestamp as defined in the STIX 2.0 specification.

4.1.2. Observation Operators

Two or more Observation Expressions **MAY** be combined using an Observation Operator in order to further constrain the set of Observations that match against the Pattern Expression.

Observation Operators	Description	Associativity
[*a*] **AND** [*b*]	Both *a* and *b* **MUST** be Observation Expressions and **MUST** evaluate to true on *different* Observations.	Left to right
[*a*] **OR** [*b*]	*a* and *b* **MUST** be Observation Expressions and one of *a* or *b* **MUST** evaluate to true on *different* Observations.	Left to right
[*a*] **FOLLOWEDBY** [*b*]	Both *a* and *b* **MUST** be Observation Expressions. Both *a* and *b* **MUST** evaluate to true, where the observation timestamp associated with *b* is greater than or equal to the observation timestamp associated with *a* and **MUST**	Left to right

	evaluate to true on *different* Observations.	

For example: `[ a = 'b' ] FOLLOWEDBY [ c = 'd' ] REPEATS 5 TIMES` says to match an Observation with a equal to 'b' that precedes 5 occurrences of Observations that have `c` equal to `d`, for a total of 6 Observations matched. This interpretation is due to qualifiers not being greedy, and is equivalent to `[ a = 'b' ] FOLLOWEDBY ( [ c = 'd' ] REPEATS 5 TIMES)`.

Alternatively, using parenthesis to group the initial portion, the example `([ a = 'b' ] FOLLOWEDBY [ c = 'd' ]) REPEATS 5 TIMES` will match 5 pairs of Observations where `a` equals 'b' followed by an Observation where `c` is equal to 'd', for a total of 10 Observations matched.

4.1.3. Operator Precedence

Operator associativity and precedence may be overridden by the use of parentheses. Unless otherwise specified, operator associativity (including for parentheses) is left-to-right. Precedence in the table is from highest to lowest.

Operators	Associativity	Valid Scope
()	left to right	Observation Expression or Pattern Expression, Observation Expression and Qualifier
AND	left to right	Observation Expression, Pattern Expression
OR	left to right	Observation Expression, Pattern Expression
FOLLOWEDBY (Observation Operator)	left to right	Pattern Expression

4.2. Comparison Expression

Comparison Expressions are the most basic components of STIX Patterning, comprising an Object Path and a constant joined by a Comparison Operator. As each Comparison Expression is a singleton, they are evaluated from left to right.

A Boolean Operator joins two Comparison Expressions together. In the following table, *a* or *b* is either a Comparison Expression, or another expression that contains a Boolean Operator.

Boolean Operator	**Description**	**Associativity**
a **AND** *b*	Both *a* and *b* **MUST** be Comparison Expressions and **MUST** evaluate to true on the same Observation.	Left to right
a **OR** *b*	Both *a* and *b* **MUST** be Comparison Expressions. Either *a* or *b* **MUST** evaluate to true.	Left to right

4.2.1. Comparison Operators

The table below describes the available Comparison Operators for use in Comparison Expressions; in the table, *a* **MUST** be an Object Path and *b* **MUST** be a constant. If the arguments to the comparison operators are of incompatible types (e.g. the Object Path is an integer and the constant is a string), the results are false, the sole exception is the **!=** operator in which case the result is true. Some constants and Cyber Observable data types may be compatible, for example, the `hex` and `binary` types both represent binary data, and their representative binary data is that which **MUST** be compared for equality. See Section 2.1 for type compatibility between Pattern and Cyber Observable types.

A Comparison Operator **MAY** be preceded by the modifier **NOT**, in which case the resultant Comparison Expression **MUST** be logically negated.

Comparison Operator	**Description**	**Example**
a **=** *b*	*a* and *b* **MUST** be equal (transitive), where *a* **MUST** be an Object Path and *b* **MUST** be a constant of the same data type as the Object property specified by *a*.	`file:name = 'foo.dll'`
a **!=** *b*	*a* and *b* **MUST NOT** be equal (transitive), where *a* **MUST** be an Object Path and *b* **MUST** be a constant of the same data type as the Object property specified by *a*.	`file:size != 4112`
a **>** *b*	*a* is numerically or lexically greater than *b*, where *a* **MUST** be an Object Path and *b* **MUST** be a constant of the same data type as the Object property specified by *a*.	`file:size > 256`
a **<** *b*	*a* is numerically or lexically less than *b*, where *a* **MUST** be an Object Path and *b* **MUST** be a	`file:size < 1024`

	constant of the same data type as the Object property specified by *a*.	
a **<=** *b*	*a* is numerically or lexically less than or equal to *b*, where *a* **MUST** be an Object Path and *b* **MUST** be a constant of the same data type as the Object property specified by *a*.	`file:size <= 25145`
a **>=** *b*	*a* is numerically or lexically greater than or equal to *b*, where *a* **MUST** be an Object Path and *b* **MUST** be a constant of the same data type as the Object property specified by *a*.	`file:size >= 33312`
a **IN** (*x*,*y*,...)	*a* **MUST** be an Object Path and **MUST** evaluate to one of the values enumerated in the set of x,y,... (transitive). The set values in *b* **MUST** be constants of homogenous data type and **MUST** be valid data types for the Object Property specified by ***a.*** The return value is true if a is equal to one of the values in the list. If a is not equal to any of the items in the list, then it is false.	`process:name IN ('proccy', 'proximus', 'badproc')`
a **LIKE** *b*	*a* **MUST** be an Object Path and **MUST** match the pattern specified in *b* where any '%' is 0 or more characters and '_' is any one character. This operator is based upon the SQL LIKE clause and makes use of the same wildcards. The string constant *b* **MUST** be NFC normalized prior to evaluation.	`directory:path LIKE 'C:\\Windows\\%\\foo'`
a **MATCHES** *b*	*a* **MUST** be an Object Path and **MUST** be matched by the pattern specified in *b,* where *b* is a string constant containing a PCRE or PCRE2 compliant regular expression. *a* **MUST** be NFC normalized before comparison if the property is of string type. Regular expressions **MUST** be conformant to the syntax defined by the Perl-compatible Regular Expression (PCRE) library. The search function **MUST** be used. The DOTALL option **MUST** be specified. The standard beginning and end anchors may be used in the pattern to obtain match behavior. In the case that the property is binary (e.g. property name ends in **_bin** or **_hex**), then the UNICODE flag **MUST NOT** be specified.	`directory:path MATCHES '^C:\\Windows\\w+$'`

Set Operator	Description	Example
a **ISSUBSET** *b*	When *a* is a set that is wholly contained by the set *b*, the comparison evaluates to true. *a* **MUST** be an Object Path referring to the **value** property of an Object of type `ipv4-addr` or `ipv6-addr`. *b* **MUST** be a valid `string` representation of the corresponding Object type (as defined in **<add reference to Network Objects here>**). In the case that both *a* and *b* evaluate to an identical single IP address or an identical IP subnet, the comparison evaluates to true.	`ipv4-addr:value ISSUBSET '198.51.100.0/24'` For example, if `ipv4-addr:value` was `198.51.100.0/27`, `ISSUBSET '198.51.100.0/24'` would evaluate to true.
a **ISSUPERSET** *b*	When *a* is a set that wholly contains the set specified by *b*, the comparison evaluates to true. *a* **MUST** be an Object Path referring either an `ipv4-addr` or `ipv6-addr` Object. *b* **MUST** be a valid `string` representation of the corresponding Object type (as defined in **<add reference to Network Objects here>**). In the case that both *a* and *b* evaluate to an identical single IP address or an identical IP subnet, the comparison evaluates to true.	`ipv4-addr:value ISSUPERSET '198.51.100.0/24'` For example, if `ipv4-addr:value` was `198.51.100.0/24`, `ISSUPERSET '198.51.100.0/27'` would evaluate to true.

4.2.2. String Comparison

For simple string operators, i.e., "**=**", "**!=**", "**<**", "**>**", "**<=**" and "**>=**", as collation languages and methods are unspecifiable, a simple code point (binary) comparison **MUST** be used. If one string is longer than the other, but otherwise equal, the longer string is greater, but not equal to, than the shorter string. Unicode normalization **MUST NOT** be performed on the string. This means that combining marks are sorted by their code point, not the NFC normalized value. E.g. ‘o' U+006f < ‘oz' U+006f U+007a < ‘ò' U+006f U+0300 < ‘z' U+007a < ‘ò' U+00f2. Though Unicode recommends normalizing strings for comparisons, the use of combining marks may be significant, and normalizing by default would remove this information. NFC normalization is required for other matching operators, e.g., **`LIKE`** and **`MATCHES`**.

4.2.3. Binary Type Comparison

When the value of two binary object properties are compared, they are compared as unsigned octets. That is `00` is less than `ff`. If one value is longer than the other, but they are otherwise equal, the longer value is considered greater than, but not equal to, the shorter value.

4.2.4. Native Format Comparison

The Cyber Observable Object's value **MUST** be in it's native format when doing the comparison. For example, Cyber Observable Object properties that use the `binary` type (defined in Part 3 Section 2.2) must have their value decoded into its constituent bytes before being compared. This also means Object Properties that use the `hex` type must be decoded to raw octets before being compared.

In cases that a binary Cyber Observable Object property (i.e., one ending with **_bin** or **_hex**) is compared against a string constant, the string constant **MUST** be converted to be a binary constant when all string code points are less than U+0100. If the conversion is not possible, the results of comparison is false, unless the comparison operator is !=, then it is true.

For example given the following object, where the **payload_bin** property is of `binary` type :

```
{
  "0":{
    "type": "artifact",
    "mime_type": "application/octet-stream",
    "payload_bin": "dGhpcyBpcyBhIHRlc3Q="
    }
}
```

The pattern "`artifact:payload_bin = 'dGhpcyBpcyBhIHRlc3Q='`" would evaluate to false, while the patterns "`artifact:payload_bin = 'this is a test'`", "`artifact:payload_bin = b'dGhpcyBpcyBhIHRlc3Q='`", and "`artifact:payload_bin = h'7468697320697320612074657374'`" would all evaluate to true.

5. Object Path Syntax

Defined below is the syntax for addressing values of Cyber Observable Objects in a STIX Pattern. The following notation is used throughout the definitions below:

Notation	Definition
<object-type>	The type of Cyber Observable Object to match against. This **MUST** be the value of the **type** field specified for a Cyber Observable Object in an Observation.
<property_name>	The name of a Cyber Observable Object property to match against. This **MUST** be a valid property name as specified in the definition of the Cyber Observable Object type referenced by the `<object-type>` notation.

	If the `<property_name>` contains a dash ('-' U+002d) or a dot (U+002e), the `<property_name>` **MUST** be enclosed in double quotes (U+0022). Properties that are nested, i.e. are children of other properties in a Cyber Observable Object, **MUST** be specified using the syntax `<property_name>`.`<property_name>`, where the `<property_name>` preceding the '.' is the name of the parent property and the one following is the name of the child property. If the property name is a reference to another Cyber Observable Object, the referenced Object **MUST** be dereferenced, so that its properties are effectively nested in the Object that it is referenced by. For example, if the **src_ref** property of the Network Traffic Object references an IPv4 Address Object, the value of this address would be specified by **network-traffic**:**src_ref.value**.

5.1. Basic Object Properties

Any non-`dictionary` and non-`list` property that is directly specified on a Cyber Observable Object.

Syntax

`<object-type>`:`<property_name>`

Example

`file:size`

5.2. List Object Properties

Any property on a Cyber Observable Object that uses the `list` data type.

Syntax

`<object-type>`:`<property_name>[list_index]`.`<property_name>`

Where `property_name` **MUST** be the name of an Object property of type `list` and `[list_index]` **MUST** be one of the following:

- An integer in the range of 0..N-1, where N is the length of the list. If list_index is out of range, the result of any operation is false.
- The literal `'*'`, which indicates that if any of the items contained within a list match the Comparison Expression, it evaluates to true.

Example

`file:extensions.windows-pebinary.sections[*].entropy > 7.0`

The above example will return true if any PE section has an entropy property whose value is greater than 7.0.

5.3. Dictionary Object Properties

Any property on a Cyber Observable Object that uses the `dictionary` data type.

Syntax
`<object-type>`:`<property_name>`.`<key_name>`

Where `<property_name>` **MUST** be the name of an Object property of type `dictionary` and `<key_name>` **MUST** be the name of key in the dictionary.

Examples
`file:hashes.MD5`

`file:extensions.raster-image.image_height`

5.4. Object Reference Properties

Any property on a Cyber Observable Object that uses the `object-ref` data type, either as a singleton or as a list (i.e., `list` of type `object-ref`).

Syntax
`<object-type>`:`<property_name>`.`<dereferenced_object_property>`

Where `<property_name>` **MUST** be the name of an Object property of type `object-ref` and `<dereferenced_object_property>` **MUST** be the name of the property of the dereferenced Object (i.e., one that in an Observation is specified via the `<property_name>` as a reference).

Examples
`email-message:from_ref.value = 'mary@example.com'`

`directory:contains_refs[*].name = 'foobar.dll'`

6. Examples

Note: the examples below are **NOT** JSON encoded. This means that some characters, like double quotes, are not escaped, though they will be when encoded in a JSON string.

6.1. Matching a File with a SHA-256 hash

```
[file:hashes."SHA-256" =
'aec070645fe53ee3b3763059376134f058cc337247c978add178b6ccdfb0019f']
```

6.2. Matching an Email Message with a particular From Email Address and Attachment File Name Using a Regular Expression

```
[email-message:from_ref.value MATCHES '.+\\@example\\.com$' AND
email-message:body_multipart[*].body_raw_ref.name MATCHES '^Final Report.+\\.exe$']
```

6.3. Matching a File with a SHA-256 hash and a PDF MIME type

```
[file:hashes."SHA-256" =
'aec070645fe53ee3b3763059376134f058cc337247c978add178b6ccdfb0019f' AND file:mime_type =
'application/x-pdf']
```

6.4. Matching a File with SHA-256 or a MD5 hash (e.g., for the case of two different end point tools generating either an MD5 or a SHA-256), and a different File that has a different SHA-256 hash, against two different Observations

```
[file:hashes."SHA-256" =
'bf07a7fbb825fc0aae7bf4a1177b2b31fcf8a3feeaf7092761e18c859ee52a9c' OR file:hashes.MD5 =
'cead3f77f6cda6ec00f57d76c9a6879f']
 AND [file:hashes."SHA-256" =
'aec070645fe53ee3b3763059376134f058cc337247c978add178b6ccdfb0019f']
```

6.5. Matching a File with a MD5 hash, followed by (temporally) a Registry Key Object that matches a value, within 5 minutes

```
[file:hashes.MD5 = '79054025255fb1a26e4bc422aef54eb4'] FOLLOWEDBY [win-registry-key:key
= 'HKEY_LOCAL_MACHINE\\foo\\bar'] WITHIN 300 SECONDS
```

6.6. Matching three different, but specific Unix User Accounts

```
[user-account:account_type = 'unix' AND user-account:user_id = '1007' AND
user-account:account_login = 'Peter'] AND [user-account:account_type = 'unix' AND
user-account:user_id = '1008' AND user-account:user_id = 'Paul'] AND
[user-account:account_type = 'unix' AND user-account:user_id = '1009' AND
user-account:user_id = 'Mary']
```

6.7. Matching an Artifact Object PCAP payload header

```
[artifact:mime_type = 'application/vnd.tcpdump.pcap' AND artifact:payload_bin MATCHES
'\\xd4\\xc3\\xb2\\xa1\\x02\\x00\\x04\\x00']
```

6.8. Matching a File Object with a Windows file path

```
[file:name = 'foo.dll' AND file:parent_directory_ref.path = 'C:\\Windows\\System32']
```

6.9. Matching on a Windows PE File with high section entropy

```
[file:extensions.windows-pebinary-ext.sections[*].entropy > 7.0]
```

6.10. Matching on a mismatch between a File Object magic number and mime type

```
[file:mime_type = 'image/bmp' AND file:magic_number_hex = h'ffd8']
```

6.11. Matching on Network Traffic with a particular destination

```
[network-traffic:dst_ref.type = 'ipv4-addr' AND network-traffic:dst_ref.value =
'203.0.113.33/32']
```

6.12. Matching on Malware Beaconing to a Domain Name

```
[network-traffic:dst_ref.type = 'domain-name' AND network-traffic:dst_ref.value =
'example.com'] REPEATS 5 TIMES WITHIN 1800 SECONDS
```

6.13. Matching on a Domain Name with IPv4 Resolution

```
[domain-name:value = 'www.5z8.info' AND domain-name:resolves_to_refs[*].value =
'198.51.100.1/32']
```

6.14. Matching on a URL

```
[url:value = 'http://example.com/foo' OR url:value = 'http://example.com/bar']
```

6.15. Matching on an X509 Certificate

```
[x509-certificate:issuer = 'CN=WEBMAIL' AND x509-certificate:serial_number =
'4c:0b:1d:19:74:86:a7:66:b4:1a:bf:40:27:21:76:28']
```

6.16. Matching on a Windows Registry Key

```
[windows-registry-key:key = 'HKEY_CURRENT_USER\\Software\\CryptoLocker\\Files' OR
windows-registry-key:key =
'HKEY_CURRENT_USER\\Software\\Microsoft\\CurrentVersion\\Run\\CryptoLocker_0388']
```

6.17. Matching on a File with a set of properties

```
[(file:name = 'pdf.exe' OR file:size = '371712') AND file:created =
t'2014-01-13T07:03:17Z']
```

6.18. Matching on an Email Message with specific Sender and Subject

```
[email-message:sender_ref.value = 'jdoe@example.com' AND email-message:subject =
'Conference Info']
```

6.19. Matching on a Custom USB Device

```
[x-usb-device:usbdrive.serial_number = '575833314133343231313937']
```

6.20. Matching on Two Processes Launched with a Specific Set of Command Line Arguments Within a Certain Time Window

```
[process:arguments = '>-add GlobalSign.cer -c -s -r localMachine Root'] FOLLOWEDBY
[process:arguments = '>-add GlobalSign.cer -c -s -r localMachineTrustedPublisher']
WITHIN 300 SECONDS
```

6.21. Matching on a Network Traffic IP that is part of a particular Subnet

```
[network-traffic:dst_ref.value ISSUBSET '2001:0db8:dead:beef:0000:0000:0000:0000/64']
```

6.22. Matching on several different combinations of Malware Artifacts

The following pattern requires that both a file and registry key exist, or that one of two processes exist.

```
([file:name = 'foo.dll'] AND [win-registry-key:key = 'HKEY_LOCAL_MACHINE\\foo\\bar']) OR
[process:name = 'fooproc' OR process:name = 'procfoo']
```

7. ANTLR Grammar

The latest ANTLR grammar for the patterning specification can be found here. Note that this grammar is non-normative and is intended solely as an aid to implementers.

8. Conformance

Implementers of the STIX Patterning language are not required to support the full capabilities provided by the language. Rather, implementers are strongly encouraged to support as much of STIX Patterning as feasible, given the capabilities of their products, but only required to support the minimum conformance level (defined below) necessary for their particular use cases. For example, the vendor of a network intrusion detection system (NIDS) that looks for malicious network traffic may only need to implement the Comparison Operators and support basic Observation Expressions to explicitly match against network traffic and IP addresses.

While the STIX Patterning language specification is tightly coupled with the STIX Cyber Observable object data models, it is understood that in many (or even most) implementations STIX Patterns will be used as an abstraction layer for transcoding into other proprietary query formats. STIX Patterns may be evaluated directly against a corpus of STIX Observed Data entities but they may also, for example, be translated into some query syntax for a packet inspection device. In this second case, the STIX Patterns are in fact evaluated in the context of data passing on the wire, not in the form of STIX Cyber Observables.

The STIX Patterning language's Observation Operators allow for the creation of patterns that explicitly match across multiple Observations but says nothing about the source of the underlying data for each Observation. For example, depending on a particular patterning implementation, the data for a pattern that matches on network traffic could come from an endpoint or from a NIDS. It is incumbent upon implementers to ascertain the appropriate data sources (where applicable) for each Observation within a given pattern.

8.1. Pattern Producer

A product that creates STIX patterns is known as a "Pattern Producer". Such products **MUST** support the creation of patterns that conform to all normative statements and formatting rules in this document. Pattern Producers **MUST** specify their conformance in terms of the conformance levels defined in Section 8.3.

8.2. Pattern Consumer

A product that consumes STIX patterns is known as a "Pattern Consumer". Such products **MUST** support the consumption of patterns that conform to all normative statements and formatting rules in this document. Pattern Consumers **MUST** specify their conformance in terms of the conformance levels defined in Section 8.3.

8.3. Conformance Levels

8.3.1. Level 1: Basic Conformance

Products that conform to the minimum required aspects of the patterning specification, and is known as a “Level 1 STIX Patterning Implementation”.

Such products **MUST** support the following features by conforming to all normative statements and behaviors in the referenced sections:

- Single Observation Expressions (omitting Qualifiers), as described in Section 4.1
- All Comparison Operators, as described in Section 4.2.1

(This level of conformance is intended primarily for products that are deployed at end points or network boundaries and which are architecturally unable to maintain state as would be required in order to support Qualifiers such as `WITHIN`.)

8.3.2. Level 2: Basic Conformance plus Observation Operators

Products that support the minimum required aspects of the patterning specification but can operate on multiple Observations, and is known as a “Level 2 STIX Patterning Implementation”.

Such products **MUST** support the following features by conforming to all normative statements and behaviors in the referenced sections:

- Compound Observation Expressions (omitting Qualifiers) as described in Section 4.1
- All Comparison Operators, as described in Section 4.2.1
- The `AND` Observation Operator, as described in Section 4.1.2
- The `OR` Observation Operator, as described in Section 4.1.2

(This level of conformance is intended primarily for products such as HIDS that can detect patterns across separate Observations but may not support temporal-based patterning.)

8.3.3. Level 3: Full Conformance

A product that is fully conformant with the **all** of the capabilities of the patterning specification is known as a "Level 3 STIX Patterning Implementation".

Such products **MUST** support the following features by conforming to all normative statements and behaviors in the referenced sections:

- Section 2. Definitions
- Section 3. STIX Patterns
- Section 4. Pattern Expressions
- Section 5. Object Path Syntax

(This level of conformance is intended primarily for products such as SIEMs that support temporal-based patterning and can also aggregate and detect patterns across multiple and disparate sources of Observations.)

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

10. Appendix B. Revision History