STIX™ Version 2.0. Part 5: STIX Patterning

Committee Specification Draft 02 / Public Review Draft 02

03 May 2017

Specification URIs

This version:

http://docs.oasis-open.org/cti/stix/v2.0/csprd02/part5-stix-patterning/stix-v2.0-csprd02-part5-stix-patterning.docx (Authoritative)
http://docs.oasis-open.org/cti/stix/v2.0/csprd02/part5-stix-patterning/stix-v2.0-csprd02-part5-stix-patterning.html
http://docs.oasis-open.org/cti/stix/v2.0/csprd02/part5-stix-patterning/stix-v2.0-csprd02-part5-stix-patterning.pdf

Previous version:

http://docs.oasis-open.org/cti/stix/v2.0/csprd01/part5-stix-patterning/stix-v2.0-csprd01-part5-stix-patterning.docx (Authoritative)
http://docs.oasis-open.org/cti/stix/v2.0/csprd01/part5-stix-patterning/stix-v2.0-csprd01-part5-stix-patterning.html
http://docs.oasis-open.org/cti/stix/v2.0/csprd01/part5-stix-patterning/stix-v2.0-csprd01-part5-stix-patterning.pdf

Latest version:

http://docs.oasis-open.org/cti/stix/v2.0/stix-v2.0-part5-stix-patterning.docx (Authoritative)
http://docs.oasis-open.org/cti/stix/v2.0/stix-v2.0-part5-stix-patterning.html
http://docs.oasis-open.org/cti/stix/v2.0/stix-v2.0-part5-stix-patterning.pdf

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:

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

Additional artifacts:

This prose specification is one component of a Work Product that also includes:

- *STIX™ Version 2.0. Part 1: STIX Core Concepts*. http://docs.oasis-open.org/cti/stix/v2.0/csprd02/part1-stix-core/stix-v2.0-csprd02-part1-stix-core.html.
- *STIX™ Version 2.0. Part 2: STIX Objects.* http://docs.oasis-open.org/cti/stix/v2.0/csprd02/part2-stix-objects/stix-v2.0-csprd02-part2-stix-objects.html.
- *STIX™ Version 2.0. Part 3: Cyber Observable Core Concepts.* http://docs.oasis-open.org/cti/stix/v2.0/csprd02/part3-cyber-observable-core/stix-v2.0-csprd02-part3-cyber-observable-core.html.

- *STIX™ Version 2.0. Part 4: Cyber Observable Objects.* http://docs.oasis-open.org/cti/stix/v2.0/csprd02/part4-cyber-observable-objects/stix-v2.0-csprd02-part4-cyber-observable-objects.html.
- (this document) *STIX™ Version 2.0. Part 5: STIX Patterning.* http://docs.oasis-open.org/cti/stix/v2.0/csprd02/part5-stix-patterning/stix-v2.0-csprd02-part5-stix-patterning.html.

Related work:

This specification replaces or supersedes:

- *STIX™ Version 1.2.1. Part 1: Overview.* Edited by Sean Barnum, Desiree Beck, Aharon Chernin, and Rich Piazza. Latest version: http://docs.oasis-open.org/cti/stix/v1.2.1/stix-v1.2.1-part1-overview.html.
- *CybOX™ Version 2.1.1. Part 01: Overview.* Edited by Trey Darley, Ivan Kirillov, Rich Piazza, and Desiree Beck. Latest version: http://docs.oasis-open.org/cti/cybox/v2.1.1/cybox-v2.1.1-part01-overview.html.

This specification is related to:

- *TAXII™ Version 2.0.* Edited by John Wunder, Mark Davidson, and Bret Jordan. Latest version: http://docs.oasis-open.org/cti/taxii/v2.0/taxii-v2.0.html.

Abstract:

Structured Threat Information Expression (STIX™) is a language for expressing cyber threat and observable information. This document defines a patterning language to enable the detection of possibly malicious activity on networks and endpoints.

Status:

This document was last revised or approved by the OASIS Cyber Threat Intelligence (CTI) TC on the above date. The level of approval is also listed above. Check the "Latest version" location noted above for possible later revisions of this document. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=cti#technical.

TC members should send comments on this specification to the TC's email list. Others should send comments to the TC's public comment list, after subscribing to it by following the instructions at the "Send A Comment" button on the TC's web page at https://www.oasis-open.org/committees/cti/.

This Committee Specification Public Review Draft is provided under the Non-Assertion Mode of the OASIS IPR Policy, the mode chosen when the Technical Committee was established. For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the TC's web page (https://www.oasis-open.org/committees/cti/ipr.php).

Note that any machine-readable content (Computer Language Definitions) declared Normative for this Work Product is provided in separate plain text files. In the event of a discrepancy between any such plain text file and display content in the Work Product's prose narrative document(s), the content in the separate plain text file prevails.

Citation format:

When referencing this specification the following citation format should be used:

[STIX-v2.0-Pt5-Patterning]

STIX™ Version 2.0. Part 5: STIX Patterning. Edited by Trey Darley and Ivan Kirillov. 03 May 2017. OASIS Committee Specification Draft 02 / Public Review Draft 02. http://docs.oasis-open.org/cti/stix/v2.0/csprd02/part5-stix-patterning/stix-v2.0-csprd02-part5-stix-patterning.html. Latest version: http://docs.oasis-open.org/cti/stix/v2.0/stix-v2.0-part5-stix-patterning.html.

Notices

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The name "OASIS" is a trademark of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see https://www.oasis-open.org/policies-guidelines/trademark for above guidance.

STIX™, CYBOX™, AND TAXII™ (STANDARD OR STANDARDS) AND THEIR COMPONENT PARTS ARE PROVIDED "AS IS" WITHOUT ANY WARRANTY OF ANY KIND, EITHER EXPRESSED, IMPLIED, OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, ANY WARRANTY THAT THESE STANDARDS OR ANY OF THEIR COMPONENT PARTS WILL CONFORM TO SPECIFICATIONS, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR FREEDOM FROM INFRINGEMENT, ANY WARRANTY THAT THE STANDARDS OR THEIR COMPONENT PARTS

Table of Contents

1 Introduction

In order to detect a large proportion of malicious behavior 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 a cyber environment. The STIX 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.

1.0.1 IPR Policy

This Committee Specification Public Review Draft is provided under the Non-Assertion Mode of the OASIS IPR Policy, the mode chosen when the Technical Committee was established.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the TC's web page (https://www.oasis-open.org/committees/cti/ipr.php).

1.1 Terminology

The key words "**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].

All text is normative except for examples and any text marked non-normative.

1.2 Normative References

[Davis] M. Davis and K. Whistler, "UNICODE NORMALIZATION FORMS", Unicode® Standard Annex #15, February 2016. [Online] Available: http://unicode.org/reports/tr15/

[RFC2119] Bradner, S., “"Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, http://www.rfc-editor.org/info/rfc4648.

1.3 Non-Normative Reference

[Pattern Grammar] OASIS Cyber Threat Intelligence (CTI) TC, "STIX Pattern Grammar", OASIS. [Online]. Available: https://github.com/oasis-open/cti-stix2-json-schemas/tree/master/pattern_grammar

1.4 ANTLR Grammar

The latest ANTLR grammar for the patterning specification can be found on Github in the Pattern Grammar repository [Pattern Grammar]. Note that this grammar is non-normative and is intended solely as an aid to implementers.

1.5 Naming Requirements

1.5.1 Property Names and String Literals

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 SDO common property **`created_by_ref`** must result in the JSON key name "created_by_ref". Properties marked required in the property tables **MUST** be present in the JSON serialization.

1.5.2 Reserved Names

Reserved property names are marked with a type called `RESERVED` and a description text of “RESERVED FOR FUTURE USE”. Any property name that is marked as `RESERVED` **MUST NOT** be present in STIX content conforming to this version of the specification.

1.6 Document Conventions

1.6.1 Naming Conventions

All type names, property names, and literals are in lowercase, except when referencing canonical names defined in another standard (e.g., literal values from an IANA registry). 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.

1.6.2 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`
  - property names are in bold style - **`protocols`**
  - literals (values) are in blue with a blue 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-point font, with straight quotes, black text and a `light grey background`, and 2-space indentation.
- Parts of the example may be omitted for conciseness and clarity. These omitted parts are denoted with the ellipses (...).

2 Definitions

The terms defined below are used throughout this document.

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 particular 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, with their **`first_observed`** timestamp defining 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 as part of a Comparison Expression. Cyber Observable Objects and their properties are defined in *STIX™ Version 2.0. Part 4: Cyber Observable Objects*.	`ipv6-addr:value`
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 an Observation Operator. This may be applied	`[ipv4-addr:value = '203.0.113.1' OR ipv4-addr:value = '203.0.113.2']` or (with Observation Operator):

	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. Qualifiers may be applied to all of the Observation Expressions joined with Observation Operators; in this case, parentheses should be used to group the set of Observation Expressions, with the Qualifier following the closing parenthesis.
		`([ipv4-addr:value = '198.51.100.5'] FOLLOWEDBY [ipv4-addr:value = '198.51.100.10'])`
		
		or (with Observation Operator and Qualifier):
		
		`([ipv4-addr:value = '198.51.100.5' ] AND [ipv4-addr:value = '198.51.100.10']) WITHIN 300 SECONDS`

Boolean Operators	Boolean Operators are used to combine 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'] AND [ ipv4-addr:value = '198.51.100.10']`
Pattern Expression	A Pattern Expression represents a valid instance of a Cyber Observable pattern. The most basic Pattern Expression consists of a single Observation Expression containing a single Comparison 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 *STIX™ Version 2.0. Part 3: Cyber Observable Core Concepts*.

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 Cyber Observable Comparable Data Type(s) column in the table below 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 [RFC4648]. The base64 string **MUST** be surrounded by apostrophes ('' 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 apostrophes ('' 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 plus sign ('+' U+002b). Negative integers **MUST** be represented by prepending a hyphen-minus ('-' U+002d). When compared against a Cyber Observable `float`, the full value must be compared and must not be truncated. For example, the result

		of comparing a STIX Patterning constant integer value of 1 to a Cyber Observable `float` value of 1.5 is not equal. The valid range of values is defined in *STIX™ Version 2.0. Part 3: Cyber Observable Core Concepts.*
`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 plus sign' ('+' U+002b). Negative floating point numbers **MUST** be represented by prepending a hyphen-minus ('-' U+002d). The valid range of values is defined in *STIX™ Version 2.0. Part 3: Cyber Observable Core Concepts*.

`string`	`string`
	`binary`
	`hex`
		A constant of `string` type encodes a string as a list of Unicode code points surrounded by apostrophes (''' U+0027).
		
		The escape character is the backslash ('\' U+005c). Only the apostrophe 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 section 2.10 of *STIX™ Version 2.0 Part 1: STIX Core Concepts*) as a string. The timestamp string **MUST** be surrounded by apostrophes (''' U+0027) and 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, context-sensitive 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 given constant using a Comparison Operator. As a 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']
```

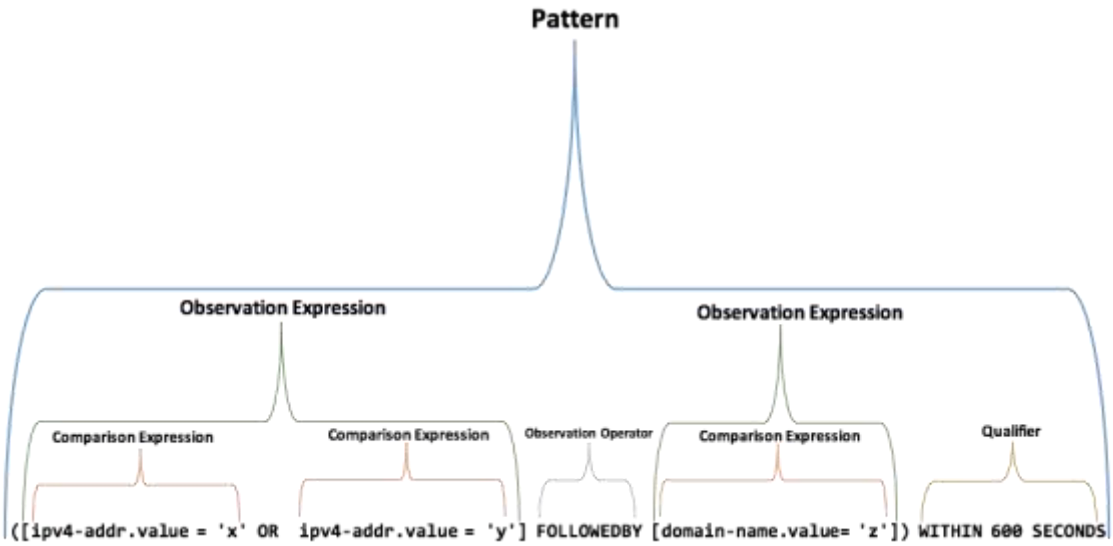
Observation Expressions may be followed by one or more Qualifiers, which allow for the expression of further restrictions on the set of data matching the pattern. Continuing with the above example, one might use a Qualifier to state that the IP addresses must be observed several times in repetition:

```
[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 previous 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 various STIX Patterning components in 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. 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 resulting from parsing error or invalid constants (e.g., an invalid hex or binary constant) **MUST NOT** match any Observations.

4.1 Observation Expressions

Observation Expressions comprise one or more Comparison Expressions, joined via Boolean Operators.

Observation Expressions **MUST** be delimited by square brackets left square bracket ('[' U+005b) and right square bracket (']' 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 Observation Operators.

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

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, consider the following 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')]
```

This expression 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. Comparison Expressions that are intended to match a single object type can be joined by either AND or OR. For example:

```
[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 delineate which Observation Expressions the Qualifiers apply to. For example:

```
([ a ] AND [ b ] REPEATS 5 TIMES) WITHIN 5 MINUTES
```

The preceding example results in one *a* and 5 *b's* that all match in a 5 minute period. As another example:

```
([ a ] AND [ b ]) REPEATS 5 TIMES WITHIN 5 MINUTES
```

The preceding example results in 5 *a*'s and 5 *b's* (10 Observations) that all match 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 `REPEATS` 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 "AND *a",* x-1 times. 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.
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...'] AND [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 section 2.10 of *STIX™ Version 2.0. Part 1: STIX Core Concepts*.

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

For example, consider the following Pattern Expression:

```
[ a = 'b' ] FOLLOWEDBY [ c = 'd' ] REPEATS 5 TIMES
```

The preceding expression 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, we get the following example:

```
([ a = 'b' ] FOLLOWEDBY [ c = 'd' ]) REPEATS 5 TIMES
```

The preceding expression 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 below 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. Each Comparison Expression is a singleton, and so 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 a composite expression (which may be composed recursively) consisting of two or more Comparison Expressions joined with Boolean Operators and enclosed by parentheses.

Boolean Operator	Description	Associativity
a `AND` *b*	*a* and *b* **MUST** both be Comparison Expressions or a composite expression (which may be composed recursively) consisting of two or more Comparison Expressions joined with Boolean Operators and enclosed by parentheses. *a* and *b* **MUST** both evaluate to true on the same Observation.	Left to right
a `OR` *b*	*a* and *b* **MUST** both be Comparison Expressions or a composite expression (which may be composed recursively) consisting of two or more Comparison Expressions joined with Boolean Operators and enclosed by parentheses. 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 STIX Patterning constants and Cyber Observable data types may be comparable in a Comparison Expression. 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 STIX Patterning and Cyber Observable types.

A Comparison Operator **MAY** be preceded by the modifier `NOT`, in which case the resultant Comparison Expression is 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 constant of the same data type as the Object property specified by *a*.	`file:size < 1024`
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 the Comparison Expression evaluates to 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 [Davis] 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 compliant regular expression. *a* **MUST** be NFC normalized [Davis] 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 (http://www.pcre.org/original/doc/html/pcrepattern.html). 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., the 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 Expression 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 *STIX™ Version 2.0. Part 4: Cyber Observable Objects*). For example, if `ipv4-addr:value` was 198.51.100.0/27, `ISSUBSET '198.51.100.0/24'` would evaluate to true. In the case that both *a* and *b* evaluate to an identical single IP address or an identical IP subnet, the Comparison Expression evaluates to true.	`ipv4-addr:value ISSUBSET '198.51.100.0/24'`
a `ISSUPERSET` *b*	When *a* is a set that wholly contains the set specified by *b*, the Comparison Expression evaluates to true. *a* **MUST** be an Object Path referring either an `ipv4-addr` or `ipv6-addr` Object.	`ipv4-addr:value ISSUPERSET '198.51.100.0/24'`

	b **MUST** be a valid `string` representation of the corresponding Object type (as defined in *STIX™ Version 2.0. Part 4: Cyber Observable Objects*). For example, if `ipv4-addr:value` was 198.51.100.0/24, `ISSUPERSET '198.51.100.0/27'` would evaluate to true. In the case that both *a* and *b* evaluate to an identical single IP address or an identical IP subnet, the Comparison Expression evaluates 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 than, but not equal to, the shorter string. Unicode normalization **MUST NOT** be performed on the string. This means that combining marks [Davis] 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. Although 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, however, required for other Comparison 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 its native format when doing the comparison. For example, Cyber Observable Object properties that use the `binary` type (defined in section 2.2 of *STIX™ Version 2.0. Part 3: Cyber Observable Core Concepts*) must have their value decoded into its constituent bytes prior to comparison. This also means that Object Properties which use the `hex` type must be decoded into raw octets prior to being evaluated.

In cases where a binary Cyber Observable Object property (i.e., one ending with **_bin** or **_hex**) is evaluated against a string constant, the string constant **MUST** be converted into a binary constant when all of the constituent string code points are less than U+0100. If this conversion is not possible, the comparison **MUST** evaluate to false, unless the comparison operator is `!=`, in which case it **MUST** evaluate to 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 following patterns would all evaluate to true:

"`artifact:payload_bin = 'this is a test'`", "`artifact:payload_bin = b'dGhpcyBpcyBhIHRlc3Q='`", and "`artifact:payload_bin = h'746869732069732061207465737374'`".

5 Object Path Syntax

Defined below is the syntax for addressing properties of Cyber Observable Objects within 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 given 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 hyphen-minus ('-' U+002d) or a full stop ('.' U+002e), the `<property_name>` **MUST** be enclosed in apostrophes (''' U+0027). 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 function as if they are 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 IPv4 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 the first `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 '`*`' indicates that if any of the items contained within a list matches against the Comparison Expression, the Comparison Expression 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.ssdeep
```

```
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 a valid property of the dereferenced Object (i.e., the Object in an Observation that is referenced via `<property_name>`).

For cases where `<property_name>` is a `list` of type `object-ref`, the corresponding syntax applies:

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

Accordingly, the same semantics for list indices as defined in section 5.2 apply in this case.

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.

Matching a File with a SHA-256 hash

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

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

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']
```

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']
```

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
```

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

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']
```

Matching a File Object with a Windows file path

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

Matching on a Windows PE File with high section entropy

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

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']
```

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']
```

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
```

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']
```

Matching on a URL

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

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']
```

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']
```

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']
```

Matching on an Email Message with specific Sender and Subject

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

Matching on a Custom USB Device

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

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

```
[process:command_line MATCHES '^.+>-add GlobalSign.cer -c -s -r localMachine Root$']
FOLLOWEDBY [process:command_line MATCHES'^.+>-add GlobalSign.cer -c -s -r
localMachineTrustedPublisher$'] WITHIN 300 SECONDS
```

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']
```


Matching on several different combinations of Malware Artifacts. Note 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 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 instances 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; however, the language purposefully does not specify anything 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.

7.1 Pattern Producer

Software that creates STIX patterns is known as a "Pattern Producer". Such software **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 7.3.

7.2 Pattern Consumer

Software that consumes STIX patterns is known as a "Pattern Consumer". Such software 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 7.3.

7.3 Conformance Levels

7.3.1 Level 1: Basic Conformance

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

Such software **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 software that is deployed at endpoints or network boundaries and which is architecturally unable to maintain state, as would be required in order to support Qualifiers such as `WITHIN`.

7.3.2 Level 2: Basic Conformance plus Observation Operators

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

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

- Single and 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 software such as HIDS that can detect patterns across separate Observations but may not support temporal-based patterning.

7.3.3 Level 3: Full Conformance

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

Such software **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 software such as SIEMs that support temporal-based patterning and can also aggregate and detect patterns across multiple and disparate sources of Observations.

Appendix A. Glossary

CAPEC - Common Attack Pattern Enumeration and Classification

Consumer - Any entity that receives STIX content

CTI - Cyber Threat Intelligence

Embedded Relationship - A link (an "edge" in a graph) between one STIX Object and another represented as a property on one object containing the ID of another object

Entity - Anything that has a separately identifiable existence (e.g., organization, person, group, etc.)

IEP - FIRST (Forum of Incident Response and Security Teams) Information Exchange Policy

Instance - A single occurrence of a STIX object version

MTI - Mandatory To Implement

MVP - Minimally Viable Product

Object Creator - The entity that created or updated a STIX object (see section 3.3 of *STIX™ Version 2.0. Part 1: STIX Core Concepts*).

Object Representation - An instance of an object version that is serialized as STIX

Producer - Any entity that distributes STIX content, including object creators as well as those passing along existing content

SDO - STIX Domain Object (a "node" in a graph)

SRO - STIX Relationship Object (one mechanism to represent an "edge" in a graph)

STIX - Structured Threat Information Expression

STIX Content - STIX documents, including STIX Objects, STIX Objects grouped as bundles, etc.

STIX Object - A STIX Domain Object (SDO) or STIX Relationship Object (SRO)

STIX Relationship - A link (an "edge" in a graph) between two STIX Objects represented by either an SRO or an embedded relationship

TAXII - An application layer protocol for the communication of cyber threat information

TLP - Traffic Light Protocol

TTP - Tactic, technique, or procedure; behaviors and resources that attackers use to carry out their attacks

Appendix B. Acknowledgments

The contributions of the OASIS Cyber Threat Intelligence (CTI) Technical Committee members, enumerated in STIX™ Version 2.0. Part 1: STIX Core Concepts, are gratefully acknowledged.

Appendix C. Revision History

Revision	Date	Editor	Changes Made
01	2017-01-20	Bret Jordan, John Wunder, Rich Piazza, Ivan Kirillov, Trey Darley	Initial Version
02	2017-04-24	Bret Jordan, John Wunder, Rich Piazza, Ivan Kirillov, Trey Darley	Changes made from first public review