stix2-validator Documentation

Release 1.1.2

OASIS Open

Dec 19, 2018

Contents:

The STIX Validator checks that STIX JSON content conforms to the requirements specified in the STIX 2.0 specification. In addition to checking conformance with the JSON schemas, the validator checks conformance with requirements that cannot be specified in JSON schema, as well as with established "best practices." This validator is non-normative; in cases of conflict with the STIX 2.0 specification, the specification takes precedence.

The STIX 2.0 specification contains two types of requirements: mandatory "MUST" requirements, and recommended "SHOULD" best practice requirements. The validator checks documents against the "MUST" requirements using JSON schemas. Some of these mandatory requirements cannot be implemented in JSON Schema, however, so the validator uses Python functions to check them. The "SHOULD" requirements are all checked by Python functions, and options may be used to ignore some or all of these recommended "best practices."

The only exception to this is the mandatory requirement that an object's 'type' be one of those defined by a STIX Object in the specification. This rules out custom objects, so this check was made optional.

The STIX Validator uses the stix2-patterns validator to check that Indicator patterns conform to the STIX Patterning language and only reference properties valid for the objects in the pattern.

The validator also color-codes its output to make it easier to tell at a glance whether validation passed.

CHAPTER 1

Installation

Note: The STIX 2 validator requires Python 2.7 or 3.4+.

The easiest way to install the STIX 2 validator is with pip:

```
$ pip install stix2-validator
```

Note that if you instead install it by cloning or downloading the repository, you will need to set up the submodules before you install it:

```
$ git clone https://github.com/oasis-open/cti-stix-validator.git
$ cd cti-stix-validator/
$ git submodule update --init --recursive
$ python setup.py install
```

CHAPTER 2

Usage

2.1 As A Script

The validator comes with a bundled script which you can use to validate a JSON file containing STIX content:

```
$ stix2_validator <stix_file.json>
```

2.2 As A Library

You can also use this library to integrate STIX validation into your own tools. You can validate a JSON file:

```
from stix2validator import validate_file, print_results

results = validate_file("stix_file.json")
print_results(results)
```

You can also validate a JSON string, and check if the input passed validation:

```
from stix2validator import validate_string, print_results

stix_json_string = "..."
results = validate_string(stix_json_string)
if results.is_valid:
    print_results(results)
```

If your STIX is already in a Python dictionary (for example if you have already run `json.loads()`), use `validate_instance()` instead:

```
import json
from stix2validator import validate_instance, print_results

```

(continues on next page)

(continued from previous page)

```
stix_json_string = "..."
stix_obj = json.loads(stix_json_string)
results = validate_instance(stix_obj)
if results.is_valid:
    print_results(results)
```

You can pass a ValidationOptions object into `validate_file()`, `validate_string()`, or `validate_instance()` if you want behavior other than the default:

```
from stix2validator import ValidationOptions

options = ValidationOptions(strict=True)
results = validate_string(stix_json_string, options)
```

2.3 Additional Schemas

The validator uses the STIX 2 JSON schemas as the basis for its validation, but you can also validate with your own additional schemas. This can help if you want to validate STIX content using custom objects, properties, or observables.

To do this use the `--schema-dir` argument:

```
$ stix2_validator --schema-dir /path/to/my/schemas <stix_file.json>
```

CHAPTER 3

Options

These are the different options that can be set, whether the validator is used as a command-line script or as a Python library. When using the validator as a library, these options can be passed as parameters to the `ValidationOptions` constructor.

Script	Library	Description
`FILES`	`files`	A whitespace separated list of STIX files or directories of STIX files to validate.
`-r, --recursive`	`recursive`	Recursively descend into input directories.
`-s SCHEMA_DIR, --schemas SCHEMA_DIR`	`schema_dir`	Custom schema directory. If provided, input will be validated against these schemas in addition to the STIX schemas bundled with this script.
`-v, --verbose`	`verbose`	Print informational notes and more verbose error messages.
`-q, --silent`	`silent`	Silence all output to stdout.
`-d DISABLED, --disable DISABLED, --ignore DISABLED`	`disabled`	A comma-separated list of recommended best practice checks to skip. By default, no checks are disabled. Example: –disable 202,210
`-e ENABLED, --enable ENABLED, --select ENABLED`	`enabled`	A comma-separated list of recommended best practice checks to enable. If the –disable option is not used, no other checks will be run. By default, all checks are enabled. Example: –enable 218
`--strict`	`strict`	Treat warnings as errors and fail validation if any are found.
`--strict-types`	`strict_types`	Ensure that no custom object types are used, only those defined in the STIX specification.
`--strict-properties`	`strict_properties`	Ensure that no custom properties are used, only those defined in the STIX specification.
`--no-cache`	`no_cache`	Disable the caching of external source values.
`--refresh-cache`	`refresh_cache`	Clears the cache of external source values, then during validation downloads them again.
`--clear-cache`	`clear_cache`	Clear the cache of external source values after validation.

For the list of checks that can be used with the “enabled” or “disabled” options, see the **:doc:‘Best Practices page**

<best-practices>‘_.

CHAPTER 4

Checking Best Practices

The validator will always validate input against all of the mandatory "MUST" requirements from the spec. By default it will issue warnings if the input fails any of the "SHOULD" recommendations, but validation will still pass. To turn these "best practice" warnings into errors and cause validation to fail, use the `--strict` option with the command-line script, or create a ValidationOptions object with `strict=True` when using the library.

You cannot select which of the "MUST" requirement checks will be performed; all of them will always be performed. However, you may select which of the "SHOULD" checks to perform. Use the codes from the table below to enable or disable these checks. For example, to disable the checks for the report label and tool label vocabularies, use `--disable 218,222` or `disabled="218,222"`. All the other checks will still be performed. Conversely, to only check that custom property names adhere to the recommended format but not run any of the other "best practice" checks, use `--enable 103` or `enabled="103"`.

Enabling supersedes disabling. Simultaneously enabling and disabling the same check will result in the validator performing that check.

Some checks access Internet resources to determine valid values for certain properties. For instance, the 'mime-type' check accesses the IANA's list of registered MIME types. These web requests are cached to conserve bandwidth, will expire after one week, and are stored in a file called 'cache.sqlite' in the same directory the script is run from. The cache can be refreshed manually with the `--refresh-cache` or `refresh_cache=True`, or cleared with `--clear-cache` or `clear_cache=True`. This caching can be disabled entirely with `--no-cache` or `no_cache=True`.

Recommended Best Practice Check Codes

Code	Name	Ensures...
1	format-checks	all 1xx checks are run
101	custom-prefix	names of custom object types, properties, observable objects, observable object properties
102	custom-prefix-lax	same as 101 but more lenient; no source identifier needed in prefix
111	open-vocab-format	values of open vocabularies follow the correct format
121	kill-chain-names	kill-chain-phase name and phase follow the correct format
141	observable-object-keys	observable object keys follow the correct format
142	observable-dictionary-keys	dictionaries in cyber observable objects follow the correct format
149	windows-process-priority-format	windows-process-ext's 'priority' follows the correct format

Table 1 – continued from previous page

150	hash-length	keys in 'hashes'-type properties are not too long
2	approved-values	all 2xx checks are run
201	marking-definition-type	marking definitions use a valid definition_type
202	relationship-types	relationships are among those defined in the specification
203	duplicate-ids	objects in a bundle with duplicate IDs have different *modified* timestamps
210	all-vocabs	all of the following open vocabulary checks are run
211	attack-motivation	certain property values are from the attack_motivation vocabulary
212	attack-resource-level	certain property values are from the attack_resource_level vocabulary
213	identity-class	certain property values are from the identity_class vocabulary
214	indicator-label	certain property values are from the indicator_label vocabulary
215	industry-sector	certain property values are from the industry_sector vocabulary
216	malware-label	certain property values are from the malware_label vocabulary
218	report-label	certain property values are from the report_label vocabulary
219	threat-actor-label	certain property values are from the threat_actor_label vocabulary
220	threat-actor-role	certain property values are from the threat_actor_role vocabulary
221	threat-actor-sophistication	certain property values are from the threat_actor_sophistication vocabulary
222	tool-label	certain property values are from the tool_label vocabulary
241	hash-algo	certain property values are from the hash-algo vocabulary
242	encryption-algo	certain property values are from the encryption-algo vocabulary
243	windows-pebinary-type	certain property values are from the windows-pebinary-type vocabulary
244	account-type	certain property values are from the account-type vocabulary
270	all-external-sources	all of the following external source checks are run
271	mime-type	file.mime_type is a valid IANA MIME type
272	protocols	certain property values are valid IANA Service and Protocol names
273	ipfix	certain property values are valid IANA IP Flow Information Export (IPFIX) Entities
274	http-request-headers	certain property values are valid HTTP request header names
275	socket-options	certain property values are valid socket options
276	pdf-doc-info	certain property values are valid PDF Document Information Dictionary keys
301	network-traffic-ports	network-traffic objects contain both src_port and dst_port
302	extref-hashes	external references SHOULD have hashes if they have a url

CHAPTER 5

Indices and tables

- genindex
- modindex
- search