maec-to-stix Documentation

Release 1.0.0-alpha1

The MITRE Corporation

October 19, 2016

Contents

The **maec-to-stix** library provides scripts and APIs for wrapping MAEC content in STIX, and also extracting STIX Indicators from MAEC dynamic analysis data.

For more information about MAEC, please visit the MAEC website. For more information about STIX, please visit the STIX website.

CHAPTER 1

Contents

1.1 Installation

The installation of maec-to-stix can be accomplished through a few different workflows.

1.1.1 Recommended Installation

Use PyPI and pip:

```
$ pip install maec-to-stix [--pre] [--upgrade]
```

Note: **maec-to-stix** is currently in **alpha** status. To install an alpha or beta release via `pip`, you must specify the version number or use `--pre`.

```
$ pip install maec-to-stix --pre
```

You might also want to consider using a virtualenv. Please refer to the pip installation instructions for details regarding the installation of pip.

1.1.2 Dependencies

The maec-to-stix package relies on some non-standard Python libraries for the processing of XML content. Revisions of maec-to-stix may depend on particular versions of dependencies to function correctly. These versions are detailed within the distutils setup.py installation script.

The following libraries are required to use maec-to-stix:

- python-maec - A python library for parsing and creating MAEC content.
- python-stix - A python library for parsing and creating STIX content.

Each of these can be installed with `pip` or by manually downloading packages from PyPI.

1.1.3 Manual Installation

If you are unable to use pip, you can also install maec-to-stix with setuptools. If you don't already have setuptools installed, please install it before continuing.

1. Download and install the dependencies above. Although setuptools will generally install dependencies automatically, installing the dependencies manually beforehand helps distinguish errors in dependency installation from errors in maec-to-stix installation. Make sure you check to ensure the versions you install are compatible with the version of maec-to-stix you plan to install.
2. Download the desired version of maec-to-stix from PyPI or the GitHub releases page. The steps below assume you are using the 1.0.0-alpha1 release.
3. Extract the downloaded file. This will leave you with a directory named maec-to-stix-1.0.0-alpha1.

```
$ tar -zxf maec-to-stix-1.0.0-alpha1.tar.gz
$ ls
maec-to-stix-1.0.0-alpha1 maec-to-stix-1.0.0-alpha1.tar.gz
```

OR

```
$ unzip maec-to-stix-1.0.0-alpha1.zip
$ ls
maec-to-stix-1.0.0-alpha1 maec-to-stix-1.0.0-alpha1.zip
```

4. Run the installation script.

```
$ cd maec-to-stix-1.0.0-alpha1
$ python setup.py install
```

5. Test the installation.

```
$ python
Python 2.7.8 (default, Mar 22 2015, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import maec_to_stix
>>> print maec_to_stix.__version__
1.0.0-alpha1
```

If you don't see an `ImportError`, the installation was successful.

1.1.4 Further Information

If you're new to installing Python packages, you can learn more at the Python Packaging User Guide, specifically the Installing Python Packages section.

1.2 Getting Started

This page gives an introduction to **maec-to-stix** and how to use it. Please note that this page is being actively worked on and feedback is welcome! If you have a suggestion or something doesn't look right, let us know: (maec@mitre.org).

Note that the GitHub repository is named `maec-to-stix`, but once installed, the library is imported using the `import maec_to_stix` statement.

1.2.1 Installation

To install **maec-to-stix** just run `pip install maec-to-stix`. If you have any issues, please refer to the instructions found on the *Installation* page.

1.2.2 Scripts

These instructions tell you how to wrap MAEC content in STIX or extract STIX Indicators from MAEC content using the scripts bundled with **maec-to-stix**.

Also discussed is the copying over of the JSON indicator extraction configuration files to a user specified directory.

maec_wrap.py

Bundled with **maec-to-stix** is `maec_wrap.py`, which is used for wrapping MAEC Package documents in STIX. It can be found on your `PATH` after installing **maec-to-stix**.

Options

Running `maec_wrap.py -h` displays the following:

```
$ maec_wrap.py -h
  usage: maec_wrap.py [-h] [--outfile OUTFILE] infile

  MAEC to STIX Wrapper Script v1.0.0-alpha1

  positional arguments:
    infile                the name of the input MAEC Package XML file to wrap in
                          STIX.

  optional arguments:
    -h, --help            show this help message and exit
    --outfile OUTFILE, -o OUTFILE
                          the name of the output STIX Package XML file. If not
                          specified, defaults to sys.stdout.
```

Basics To wrap a MAEC Package in STIX, just provide the input filename and optionally the output filename, respectively. If no output filename is specified, the script will print the output STIX Package to `sys.stdout`.

```
$ maec_wrap.py maec_doc.xml --outfile stix_doc.xml
```

maec_extract_indicators.py

Also bundled with **maec-to-stix** is `maec_extract_indicators.py`, which is used for extracting indicators from MAEC documents and outputting them in a STIX Package. It can likewise be found on your `PATH` after installing **maec-to-stix**.

Options

Running `maec_extract_indicators.py -h` displays the following:

```
$ maec_extract_indicators.py -h
  usage: maec_extract_indicators.py [-h] [--outfile OUTFILE]
                                    [--config_directory CONFIG_DIRECTORY]
                                    [--print_options]
                                    infile
```

```
MAEC to STIX Indicator Extraction Script v1.0.0-alpha1

positional arguments:
  infile                the name of the input MAEC Package XML file to extract
                        indicators from.

optional arguments:
  -h, --help            show this help message and exit
  --outfile OUTFILE, -o OUTFILE
                        the name of the output STIX Package XML file. If not
                        specified, defaults to sys.stdout.
  --config_directory CONFIG_DIRECTORY, -c CONFIG_DIRECTORY
                        the path to the directory housing the Indicator
                        extraction JSON configuration files.
  --print_options, -p   print out the current set of indicator extraction
                        options, including the supported Actions and Objects.
```

Basics To extract STIX Indicators from a MAEC `MAEC Package`, just provide the input filename and optionally the output filename, respectively. If no output filename is specified, the script will print the output STIX Package to `sys.stdout`. Note that the behavior of the Indicator extraction is driven by a set of JSON configuration files, covered in *Indicator Extraction Configuration*. For more information on the indicator extraction process itself, please refer to *Indicator Extraction Process*.

```
$ maec_extract_indicators.py maec_doc.xml --outfile stix_doc.xml
```

copy_maec_to_stix_config.py

The other script bundled with **maec-to-stix** is `copy_maec_to_stix_config.py`, which is simply intended to copy over the installed JSON indicator extraction configuration files to a user specified directory. For more information on the indicator extraction configuration files, please refer to *Indicator Extraction Configuration*.

Options

Running `copy_maec_to_stix_config.py -h` displays the following:

```
$ maec_to_stix.py -h
  usage: copy_maec_to_stix_config.py [-h] outpath

  MAEC to STIX configuration copying script

  positional arguments:
    outpath     the output directory into which the MAEC to STIX Indicator
                extraction configuration files will be copied. If the directory
                does not already exist, it will be created by the script.

  optional arguments:
    -h, --help  show this help message and exit
```

Basics

The only argument to the script is `outpath`, which should point to a directory into which the JSON indicator extraction configuration files will be copied. Note that if this directory does not exist, it will be created by the script.

```
$ copy_maec_to_stix_config.py "temp\json_config"
```

CHAPTER 2

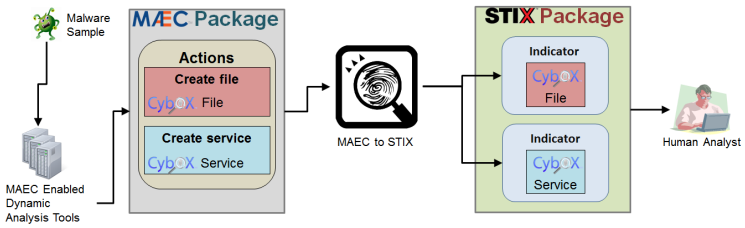
Indicator Extraction

2.1 Indicator Extraction

This page describes the premise behind the indicator extraction process used in **maec-to-stix**, for extracting STIX Indicators from MAEC Packages.

2.1.1 Overview

The diagram below highlights the overall process of extracting Indicators from MAEC Packages, starting with the generation of the MAEC output from some dynamic analysis tool (such as Cuckoo Sandbox) and ending with inspection of the resulting STIX Indicators by a human analyst. More details on the actual process of extracting Indicators from MAEC is provided in the sections below.



2.1.2 MAEC Actions

One of the fundamental MAEC entities captured in the MAEC Package is the MAEC Action. MAEC Actions represent discrete abstractions of system-level API calls, and thus are the types of activity recorded by dynamic analysis tools (sandboxes) such as Cuckoo, ThreatExpert, Anubis, etc.

Actions provide context as to the changes the malware made on the system, by specifying the particular **type** of action that was performed, such as *create file*, along with the entities that they operated on. In MAEC Actions, such entities are represented with CybOX Objects, such as the File Object, Windows Registry Key Object, etc.

2.1.3 Actions → Indicators

Certain types of Actions can leave detectable artifacts, whether they are discoverable on an endpoint (such as files on the system where the malware executed) or on some enterprise-level appliance (such as connection requests to particular IP addresses on a network gateway). Accordingly, it follows that such Actions make a good basis for the

creation of indicators, as having the knowledge of those types of Actions that leave detectable artifacts means that their resulting artifacts can be used as the basis for detection, i.e. as indicators.

Thus, the high-level indicator extraction process flow in maec-to-stix is:

1. **Parse Input MAEC Document**
2. **Extract MAEC Actions**
3. **Look for Actions with Detectable Artifacts**
4. **Perform Artifact Sanity/Consistency checking**
5. **Create STIX Indicators for acceptable Artifacts (from 4.)**
6. **Output STIX Indicators in new STIX Package**

For more information on this process, particularly with regards to step 4, please refer to *Indicator Extraction Process*.

It is important to note that this process is not fool-proof. Automatically constructing indicators from such Actions is at best a *starting point* for malware-oriented detection. However, it is **HIGHLY** recommended that such indicators still be vetted by a human analyst in order to ensure that they do not lead to false negatives or false positives.

2.1.4 Example

The following basic example demonstrates this premise with a notional MAEC Action and the STIX Indicator that results from it after automatic extraction.

Input MAEC Action

```
<maecBundle:Action>
  <cybox:Name xsi:type="maecVocabs:FileActionNameVocab-1.0">create file</cybox:Name>
  <cybox:Associated_Objects>
    <cybox:Associated_Object>
      <cybox:Properties xsi:type="FileObj:FileObjectType">
        <FileObj:File_Path>C:\T3MP\lbsec.dll</FileObj:File_Path>
        <FileObj:Size_In_Bytes>619</FileObj:Size_In_Bytes>
      </cybox:Properties>
      <cybox:Association_Type>output</cybox:Association_Type>
    </cybox:Associated_Object>
  </cybox:Associated_Objects>
</maecBundle:Action>
```

Output STIX Indicator

```
<stix:Indicator xsi:type='indicator:IndicatorType'>
    <indicator:Title>Malware Artifact Extracted from MAEC Document</indicator:Title>
    <indicator:Observable>
      <cybox:Object>
        <cybox:Properties xsi:type="FileObj:FileObjectType">
          <FileObj:File_Path condition="Equals">C:\T3MP\lbsec.dll</FileObj:File_Path>
          <FileObj:Size_In_Bytes conditions="Equals">619</FileObj:Size_In_Bytes>
        </cybox:Properties>
      </cybox:Object>
    </indicator:Observable>
</stix:Indicator>
```

Again, this is a very simplistic example, but it demonstrates that the context provided by the MAEC Action - that the file was created as the output of the action - allows us to make the determination that it could be suitable as a STIX Indicator.

2.2 Indicator Extraction Process

This page details the Indicator extraction process used in **maec-to-stix**.

2.2.1 Configuration Parsing

The first step involves parsing the JSON configuration files in order to build up the list of supported MAEC Actions and CybOX Objects (along with their properties). For more information on the configuration files, including how they can be edited and used, please refer to *Indicator Extraction Configuration Files*.

2.2.2 MAEC Package Parsing

The next step is the parsing of the MAEC Package, including its child Malware Subjects and their embedded Findings Bundles (which may contain MAEC Actions). Accordingly, a STIX TTP is created for each Malware Subject, and then referenced in the `Indicated_TTP` field of each STIX Indicator that gets extracted from the Malware Subject.

2.2.3 Indicator Object Selection & Filtering

The process of selecting and filtering the CybOX Objects suitable for use in Indicators itself contains several sub-steps, detailed below. This is done on a per-Bundle basis.

Candidate Object Selection

The initial sub-step with regards to constructing Indicators is to create the candidate list of CybOX Objects that may potentially be used as Indicators. This is accomplished by creating an ObjectHistory instance for the Bundle, which contains a list of the Objects found in the Bundle along with the Actions that operated on them. This latter aspect is important, as the candidate Objects are selected on the basis of having *at least one* supported MAEC Action (as parsed in from the configuration files) that operates on them.

For example, suppose the following Actions and Objects are defined as supported:

- **Supported Actions**: create file
- **Supported Objects**: File Object

Thus, only the second Object History entry would be considered a candidate Object, as it contains a supported Action.

Object	Actions	Candidate Object
File Object	modify file, move file	No
File Object	create file, write to file	**Yes**

Candidate Object Filtering

After creating the list of candidate CybOX Objects, the next step is to further filter this list based on the requirements dictated by the configuration files as well as some further sanity checking.

Contra-indicator Testing

The first step in the candidate CybOX Object filtering process is the testing of the Object History entries for contra-indicators. By this, we mean testing for the existence of specific Actions performed on the Object that modify its state and thus may render it unusable for detection. For example, deleting a file that was created would mean that it may not be detectable and thus unsuitable for use as an Indicator.

This logic operates by checking for specific terms in the names of the Actions that operate on the Object, including for direct contra-indicators (such as "delete"), and also for modifiers (such as "move") where the Object may be used as an input to the Action. Both of these sets of terms are captured as lists in the main indicator extraction configuration file; for more information please refer to *main_parameters*.

For example, suppose the following list of contra-indicators and modifiers is defined:

- **Contra-indicators**: delete
- **Modifiers**: move

Thus, the first two Object History inputs below would not pass the filter, as they contain Actions that serve as contra-indicators for the presence of the Object.

Object	Actions	Contra-indicator
File Object	create file, move file	**Yes**
File Object	create file, delete file	**Yes**
File Object	create file, write to file	No

Required Property Testing

If an Object History entry passes the contra-indicator tests, the next step in the filtering process is to test whether it contains the required set of properties, as specified in the `granular_config`. For example, a file Object would not be very useful without a file path that states where it can be found, or more generally an MD5 (or other) hash value. Thus, this logic checks for the existence of any required (or mutually exclusive required) properties that are defined for a particular Object type.

Also checked here is whether the value of an Object property matches against any of the whitelist entries specified in the configuration parameters for the property. Such whitelist entries are intended to specify values that are *whitelisted* from being searched for and therefore used in indicators. For example, internal IP addresses would be good candidates for additions to such a whitelist, as they would not make useful indicators. If an Object property value matches against a whitelist entry, the property will not be included in the corresponding Indicator. If such a property is required (or mutually exclusive required), this means that its parent Object will be discarded and not used in a STIX Indicator. For more information on how Object properties may be configured, including the use of the whitelist, please refer to *object_parameters*.

Extraneous Property Pruning

If a CybOX Object passes the required property testing, the final step in the Object filtering process is to prune from it any extraneous properties, that is those that aren't specified as required or optional in the `granular_config`. With this step complete, the resulting list of CybOX Objects represents the final Objects that will end up being used in the construction of the STIX Indicators.

Final Object Preparation

With the list of final (filtered and pruned) CybOX Objects constructed, there's one more step that must be done before these Objects can be used in STIX Indicators. Because these Objects came from *instance* data as reported by a

dynamic analysis tool (i.e. sandbox), we need to modify them so that they now represent *patterns* capable of being used in detection. This is achieved by setting the **condition** attribute on each property of the Object; by default, this is set to a value of **Equals**.

2.2.4 STIX Indicator Creation

The final step is the creation of the STIX Indicators themselves, one per each of the final CybOX Objects described above. Besides using the CybOX Object in the Observable of each Indicator, the following fields are populated:

- Title: states that the Indicator represents a malware artifact extracted from a MAEC document
- Type: set to "Malware Artifacts" from the `IndicatorTypeVocab`
- Description: includes the set of Actions that operated on the Object, e.g. "create file"
- Indicated_TTP: references the TTP that corresponds to the Malware Subject from which the Indicator was extracted
- Confidence/Value: set to a value of "Low" to denote that the Indicator was tool-generated

2.3 Indicator Extraction Configuration

This page describes the configuration structures employed by **maec-to-stix** for indicator extraction and how they may be modified by users in order to customize the behavior of the utility.

2.3.1 Overview

Extracting indicators from malware is a process that requires fine tuning based on a number of internal or external factors. As such, **maec-to-stix** supports the customization of its behavior in terms of extracting indicators from MAEC data. This customization can be performed at multiple levels, both high-level and granular.

In terms of high-level customization options, **maec-to-stix** offers the ability to specify:

- Whether to extract indicators for some predefined system activity OR based on a user-specified granular configuration.
- For extracting indicators based on predefined system activity, the particular type of activity to extract indicators for.
  - E.g., file system, Windows registry, network, etc.
- Whether to normalize the indicator output to make it relatively system independent.

With regards to granular customization options, **maec-to-stix** offers the ability to specify:

- The particular MAEC action types to attempt to extract indicators from.
  - E.g., create file, create registry key, etc.
- The particular CybOX object types to attempt to extract indicators from, as well as the specific properties of each object type that are allowable for usage in an indicator.

2.3.2 Configuration Structures

The **maec-to-stix** configuration structures are stored in JSON files and have two distinct levels of granularity. Details of how to edit and use the high-level and granular configuration files, as well as information about the structures of the files themselves can be found at:

2.4 Indicator Extraction Configuration Files

This page describes the location and usage of the indicator extraction configuration files. For details on the structures of the files and their parameters please refer to the high_level_config or granular_config pages.

2.4.1 Overview

There are multiple configuration files - a main configuration file, one each for the different types of system activity included by default, and one granular configuration file that contains the full list of MAEC Actions and CybOX Objects:

File	Description	Reference
extractor_config.json	The main configuration file.	high_level_config
driver_activity_config.json	System activity configuration file.	granular_config
file_system_activity_config.json	System activity configuration file.	granular_config
mutex_activity_config.json	System activity configuration file.	granular_config
network_activity_config.json	System activity configuration file.	granular_config
process_activity_config.json	System activity configuration file.	granular_config
registry_activity_config.json	System activity configuration file.	granular_config
service_activity_config.json	System activity configuration file.	granular_config
granular_config.json	Full granular configuration file.	granular_config

Main Configuration File

The main configuration file is the driver of indicator extraction behavior and is the first file parsed by the utility for this purpose. As such, it is either automatically parsed by the utility from the **maec-to-stix** installation directory, or passed in by the user. More information on this can be found in the section below.

System Activity Configuration Files

Each of the system activity configuration files contains only the set of MAEC Actions and CybOX Objects that are relevant in the context of the particular type of system activity that it refers to. Note that not all of these Actions and CybOX Objects and their properties are enabled in each activity-level configuration file by default; please click on the file name above or refer to granular_config_defaults for the list of default Actions and CybOX Objects in each. Thus, each of these files may be edited for more granular control of a particular system activity for which to extract indicators for.

Full Granular Configuration File

If one wishes to have even more control, there is a single "full" granular configuration file that represents the FULL list of possible MAEC Actions and CybOX Objects that may be configured for use in indicator extraction. This file is only used by the utility if the use_granular_options parameter in the high_level_config is set to **true**. Note that usage of this file is mutually exclusive with usage of the system-level activity configuration files.

2.4.2 Installation and Usage

By default, the configuration files are installed in the **maec-to-stix** installation directory in `python/lib/site-packages`. However, instead of editing them in place there, we recommend copying them over to another directory and making any changes as needed to these copies. To that end, we've provided a script, `copy_maec_to_stix_config.py`, that will copy all of the configuration files to a user-specified directory. For more information on this script, please refer to *copy_maec_to_stix_config.py*.

Accordingly, in order to use any user-edited files, the utility needs to be told where to find them. Luckily, this is a very simple process, for both the `maec_extract_indicators.py` script, as well as the API.

maec_extract_indicators.py

`maec_extract_indicators.py` includes a *–config_directory* (or *-c*) command-line parameter for specifying the directory where the configuration files are located.

Example

As an example, let's assume that we've edited the main configuration file and some of the granular configuration files and placed them in `/usr/tmp`. The following command-line would force `maec_extract_indicators.py` to use these modified configuration files:

```
$ maec_extract_indicators.py --config_directory /usr/tmp maec_doc.xml --outfile stix_doc.xml
```

API

The **maec-to-stix** API supports passing in the path to the directory where the configuration files are stored through the **config_directory** parameter in `maec_to_stix.extract_indicators()`.

Example

As an example, let's assume that we've edited the main configuration file and some of the granular configuration files and placed them in `/usr/tmp`. The following **maec-to-stix** API usage demonstrates how these modified configuration files would be passed in:

```
import maec_to_stix

# Extract STIX Indicators from the 'sample_maec_package.xml' MAEC document
# Pass in the modified configuration file
stix_package = maec_to_stix.extract_indicators('sample_maec_package.xml', config_directory="/usr/tmp'
```

CHAPTER 3

API Reference

3.1 API Documentation

The **maec-to-stix** APIs provide methods for wrapping MAEC data in STIX and also extracting STIX Indicators from MAEC content. Listed below are the modules and packages provided by the **maec-to-stix** library.

For examples of how make use of all of this, check out the *Example Code* page.

Note: The **maec-to-stix** APIs are currently under heavy development. Feel free to check out our issue tracker to see what we're working on!

3.1.1 `maec_to_stix` Module

`maec_to_stix.`**`extract_indicators`**(*package*, *config_directory=None*)

Extract STIX Indicators from a MAEC Package file.

Parameters

- **package** – The MAEC Package file or file-like object to wrap.
- **config_directory** – (optional) The path to the directory housing the indicator extraction configuration files.

Returns If indicators were extracted, a `stix.STIXPackage` instance with the extracted STIX Indicators. Otherwise, if no indicators were extracted, `None`.

`maec_to_stix.`**`wrap_maec_package`**(*package*)

Wrap a MAEC Package file in a STIX Package/TTP.

Parameters **package** – The MAEC Package file or file-like object to wrap.

Returns A `stix.STIXPackage` instance with the wrapped MAEC data.

3.1.2 `maec_to_stix.stix_wrapper` Module

`maec_to_stix.stix_wrapper.`**`wrap_maec`**(*maec_package*, *file_name=None*)

Wrap a MAEC Package in a STIX TTP/Package. Return the newly created STIX Package.

Parameters

- **maec_package** – the `maec.package.package.Package` instance to wrap in STIX.

- **file_name** – the name of the input file from which the MAEC Package originated, to be used in the Title of the STIX TTP that wraps the MAEC Package. Optional.

Returns A `stix.STIXPackage` instance with a single TTP that wraps the input MAEC Package.

3.1.3 maec_to_stix.indicator_extractor Module

class maec_to_stix.indicator_extractor.**IndicatorExtractor**(*maec_package*, *file_name=None*, *config_directory=None*)

Bases: object

Used to extract STIX Indicators from a MAEC Package.

stix_package
the output STIX Package (with Indicators). An instance of the `stix.STIXPackage` class.

Parameters

- **maec_package** – the input MAEC Package, an instance of the `maec.package.package.Package` class.
- **file_name** – the name of the file that contained the MAEC Package. Optional.
- **config_directory** – the path to the directory where the JSON configuration files can be found. Optional.

extract()
Attempt to extract STIX Indicators from the provided MAEC Package using the specified configuration.

Returns If indicators were extracted, a `stix.STIXPackage` instance with the extracted STIX Indicators. Otherwise, if no indicators were extracted, `None`.

exception maec_to_stix.indicator_extractor.**UnsupportedMAECEntityException**

Bases: exceptions.Exception

Basic exception for throwing when an unsupported MAEC document type is encountered.

3.1.4 maec_to_stix.indicator_extractor.config_parser Module

class maec_to_stix.indicator_extractor.config_parser.**ConfigParser**(*config_directory=None*)

Bases: object

Used to parse the JSON indicator extraction configuration files.

config_dict
the parsed dictionary representation of the main configuration file.

supported_actions
the list of supported Actions (names).

supported_objects
a dictionary of supported Objects and their properties.

Parameters **config_directory** – the path to the directory where the configuration files can be found.

static flatten_dict(*d*, *parent_key=''*, *sep='/'*)
Flatten a nested dictionary into one with a single set of key/value pairs.

Parameters

- **d** – an input dictionary to flatten.
- **parent_key** – the parent_key, for use in building the root key name when handling nested dictionaries.
- **sep** – the separator to use between the concatenated keys in the root key.

Returns The flattened representation of the input dictionary.

parse_config()
Parse the JSON configuration structure and build the appropriate data structures.

print_config()
Print the current set of configuration parameters to stdout.

Note: This method prints detailed information about the parsed Indicator extraction configuration, including:

1. The general Indicator extraction parameters (from config/extractor_config.json)
2. The supported Actions (derived from all of the parsed JSON configuration files)
3. The supported Objects and their properties (derived from all of the parsed JSON configuration files)
4. The contra-indicators and modifiers to use in candidate Object filtering

---

3.1.5 maec_to_stix.indicator_extractor.indicator_filter Module

class maec_to_stix.indicator_extractor.indicator_filter.**IndicatorFilter**(*config*)
Bases: object

Used to filter Object History entries through contraindicator checking and required property checking. Also, used to prune any extraneous properties from an Object.

Parameters **config** – The configuration structure. An instance of maec_to_stix.indicator_extractor.config_parser.ConfigParser.

prune_objects(*candidate_indicator_objects*)
Perform contraindicator and required property checking and prune un-wanted properties from the input list of candidate Indicator CybOX Objects.

Parameters **candidate_indicator_objects** – a list of maec.bundle.object_history.ObjectHistoryEntr objects representing the initial list of CybOX Objects that may be used in the STIX Indicators.

Returns A list of maec.bundle.object_history.ObjectHistoryEntry objects representing the final list of checked and pruned CybOX Objects that will be used for the STIX Indicators.

3.2 Example Code

The following sections demonstrate how to use the **maec-to-stix** library to wrap MAEC content in STIX and also extract STIX Indicators from MAEC. For more details about the **maec-to-stix** API, see the *API Documentation* page.

3.2.1 Import maec-to-stix

To use **maec-to-stix** for wrapping MAEC in STIX and extracting STIX Indicators, you must import the `maec-to-stix` module. There are lots of functions, classes, and submodules under `maec-to-stix`, but the top-level module is all you need for most usage.

```
import maec_to_stix  # That's it!
```

3.2.2 Wrapping MAEC Content in STIX

Wrapping MAEC content with **maec-to-stix** is simple - once the imports are taken care of, you only need to call the `maec_to_stix.wrap_maec_package()` method, which parses the input MAEC Package, wraps it in STIX, and returns an instance of a *stix.STIXPackage* class (from the **python-stix** API) with the wrapped MAEC content.

```
import maec_to_stix

# Wrap the 'sample_maec_package.xml' MAEC document in a STIX Package
stix_package = maec_to_stix.wrap_maec_package('sample_maec_package.xml')
```

Note:

The `maec_to_stix.wrap_maec_package()` method expects a filename to be passed in. For passing in *maec.Package* objects directly, please see the *maec_to_stix.stix_wrapper Module* documentation.

3.2.3 Extracting STIX Indicators from MAEC Content

Extracting STIX Indicators from MAEC content with **maec-to-stix** is equally simple - once the imports are taken care of, you only need to call the `maec_to_stix.extract_indicators()` method, which parses the input MAEC Package, attempts to extract STIX Indicators from it, and returns an instance of a *stix.STIXPackage* class (from the **python-stix** API) with the extracted Indicators.

```
import maec_to_stix

# Extract STIX Indicators from the 'sample_maec_package.xml' MAEC document
stix_package = maec_to_stix.extract_indicators('sample_maec_package.xml')
```

Note:

The `maec_to_stix.extract_indicators()` method expects a filename to be passed in. For passing in *maec.Package* objects directly, please see the *maec_to_stix.indicator_extractor Module* documentation.

CHAPTER 4

Contributing

If a bug is found, a feature is missing, or something just isn't behaving the way you'd expect it to, please submit an issue to our tracker. If you'd like to contribute code to our repository, you can do so by issuing a pull request and we will work with you to try and integrate that code into our repository.

CHAPTER 5

Indices and tables

- *genindex*
- *modindex*
- *search*

m

C

E

F

I

M

P

S

U

W