Product Installation & Usage Guide

API Specification Risk Assessment Tool

Introduction

This document aims to introduce the API Specification Risk Assessment Tool. The tool can be used to identify risks associated with one or more OpenAPI v2 or v3 specifications, and consequently be used for reducing the attack surface across applications.

API Specification Risks

OpenAPI specifications (hereby called spec(s)) define the properties associated with an application and its APIs. Such properties include but are not limited to: enablement of HTTPS for transactions, the type of parameters to be input, or data types used for API response. The API Specification Risk Assessment tool subjects each spec to a number of ***rules***. Each rule consists of one or more ***expressions*** that checks for specific violations. The rule also defines one or more applicable ***categories*** and associates the violation with a ***severity*** and ***score***. In addition, the tool also provides sufficient information related to each of the rules that has triggered such as the application, the API, the parameters etc. which can help identify and fix the violations.

Risk Categories

The tool checks for risks across three major categories:

Security

This category contains rules associated with the security aspects of the spec. An example of a security attribute would be to check whether the credentials for an API are transported over HTTP/HTTPS. The Security category is further categorized into 3 sub-categories:

- Authentication
- Authorization
- Transport

Data Validation

The data category rules check for properties associated with the data exchanged between the client and the server. As an example, we check whether array parameters have a maximum number of elements pre-defined. Absence of such a restriction on parameters can result in DoS attempts by a malicious actor.

Format

Finally, the rules in this category check for format related violations. As an example, properties associated with an invalid response code such as “805” would be a violation in this category.

Risk Severity & Scores

Each rule that is triggered results in a violation and is assigned a corresponding score. The score is assigned based on the research performed by CloudVector Security Research Team. These scores will be customizable in subsequent updates released to this tool. The score ranges between 0-10. The tool then maps every score to a severity as defined below:

Critical : 9-10
High : 6-8
Medium : 4-5
Low : 1-3
NoRisk : 0

Custom Rules

In addition to the predefined Security, Data, and Format rules that are shipped out of the box for evaluation against every OpenAPI spec, this tool allows users to define their own custom rules. This is useful when the enterprise defines their custom standards or guidelines for their development teams. An example of such a custom rule can be: “Every defined API endpoint URL should adhere to the format/reg-ex “^/api/v[0-9]+/”. Such custom rules can be defined using a BNF grammar. For instance, a rule that checks for all array parameters to have the maximum number of elements defined, would translate in BNF as:

parameters->->type eq array AND parameters->*->maxItems is-missing True*

The rules can also be expressed in JSON format as:

```
{"name": "CVSPD005a",
"description": "Parameters of type array should have maxItems defined.",
"score": 6,
"rule": [
                        {"identifier": "parameters->*->type",
                        "condition": "eq",
                        "value": "array"},
                "and",
                        {"identifier": "parameters->*->maxItems",
                        "condition": "is-missing",
                        "value": "True"}
            ]
}
```

Note that the identifier refers to the objects to be evaluated within the OpenAPI spec. The identifier can either be a *keyword* identifier or an *absolute* identifier:

- *Absolute identifiers* represent the absolute path to the specific field within the spec. Each absolute identifier starts with a # which is typically used to represent the root of the OpenAPI spec. E.g. "#->info->contact" refers to the *contact* field within the *info* object in the spec.
- If it is not an absolute identifier, it is assumed to be a *keyword identifier*. The first token in the keyword identifier is expected to represent all objects within the spec. For instance, the keyword identifier "*parameters->*->type*" refers to all parameters objects across all APIs within the analyzed spec.

Further notes about custom rules:

- This tool supports the following comparison operators based on data types:
  - Integers: *<, >, <=, >=, !=, ==*
  - Strings: *eq, ne, pattern-match*
  - *is-missing*, *is-empty*
- *__key__ suffix-operator:* A typical rule such as "*parameters->*->type eq array"* would compare the **value** of the *type* field, with the user-supplied value *"array"*. We support the *__key__* suffix-operator which forces the identifier field key itself to be used in comparison. E.g. an identifier such as *"responses->*__key__ >= 600"* checks for the response codes themselves to be compared against the user-supplied value *"600"*.
- While the first token can be used to refer to a keyword across the spec, we also support the *operation* token which can be used to refer to the *Operation item*s within the OpenAPI spec. The *operation* keyword helps retrieve all request methods for an API such as *"get", "post", "delete",* etc.
- Operators ("*and"/"or"*) joining different expressions are evaluated from left-to-right. Take an example of a rule definition:

```
{"name": "CVSPD001",
"description": "Parameters of type array should have maxItems defined.",
"score": 6,
"rule": [
                        {"identifier": "parameters->*->type",
                        "condition": "eq",
                        "value": "array"},
                    "and",
                        {"identifier": "parameters->*->maxItems",
                        "condition": "is-missing",
                        "value": "True"},
                    "or",
                        {"identifier": "parameters->*->format",
                        "condition": "is-missing",
                        "value": "True"},
```

```
        ]
    }
```

If each of the three expressions as above evaluated to, say: [*True*, *False, True*] respectively, then the final evaluation occurs as:

(True and False or True) == True.

- The tool also allows the ability to specify nested rules in scenarios where the language may itself not be sufficient to express a relatively sophisticated rule. An example of a parent rule (*name: CVSPS003*) referring to a nested rule (*CVSP003a*) is:

```
    {"name": "CVSPS003a",
    "description": "Global security field is missing, is empty, or contains an empty security requirement.",
    "enabled": false,
    "rule": [
                            {"identifier": "#->security",
                            "condition": "is-missing",
                            "value": "True"},
                    "or",
                            {"identifier": "#->security",
                            "condition": "is-empty",
                            "value": "True"},
                    "or",
                            {"identifier": "#->security->*",
                            "condition": "is-empty",
                            "value": "True"}
            ]
    },

    {"name": "CVSPS003",
    "description": "Global security field is missing, is empty, or contains an empty security requirement.",
    "score": 9,
    "enabled": true,
    "rule": [
                            {"identifier": "operation->security",
                            "condition": "is-missing",
                            "value": "True"},
                    "and",
                            {"identifier": "__rule__CVSPS003a",
                            "condition": "==",
                            "value": "True"}
            ]
    }
```

Note here that the nested rule is prefixed with the string *“__rule__”*. The referenced nested rule refers to another disabled rule using the rule name (here *CVSPS003a*).

Installation

System Requirements

- Operating System: MacOS, Linux, Windows
- Software: Python 3+

Install

- First setup a Python3 virtual environment:
  ```
  $ virtualenv -p python3 cvapirisk_venv
  ```

- Activate the virtualenv
  ```
  $ cd cvapirisk_venv
  $ source cvapirisk_venv/bin/activate
  ```

- Use pip to install the tool:
  ```
  $ pip install --extra-index-url http://pypi.cloudvector.net:8182/
  --trusted-host pypi.cloudvector.net cvapirisk
  ```

  Username: `cvapirisk_user` Password: `ApiS3cur!ty`

- Validate the installation:
  ```
  $ cvapirisk -h
  ```

  The output should look like:
  ```
  usage: cvapirisk [-h] {eval_risk,trend_risk} ...

  positional arguments:
    {eval_risk,trend_risk}
                          sub-command help
      eval_risk           eval_risk help
      trend_risk          trend_risk help

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

Upgrade

- On command line:
  ```
  $ pip install -U --extra-index-url http://pypi.cloudvector.net:8182/
  --trusted-host pypi.cloudvector.net cvapirisk
  ```

Usage

The following section provides the usage details of the tool assuming that you have installed the tool successfully.

- **Show help:**
  cvapirisk -h

  Usage:

  cvapirisk [-h]
  cvapirisk eval_risk (-z spec_zip_file | -s spec_file)
  (-i cv_rules_file -r custom_rules_file) (-o output_file)

  cvapirisk trend_risk <original_spec_file> <updated_spec_file>
  (-i cv_rules_file -r custom_rules_file) (-o output_file)

eval_risk	Evaluate the risks within a collection of specs or a single spec
trend_risk	Observe the risk trend between two specs

  Options:

-h	Show Usage

  Options (eval_risk):

-z spec_zip_file	The zip file that contains a collection of specs
-s spec_file	A single spec file
-i cv_rules_file	CloudVector rules file. Note that this is optional.
-r custom_rules_file	The custom rules file. Note that this is optional.
-o output_file	The output file to capture the evaluation results

  Options (trend_risk):

<original_spec_file>	The original spec file
<updated_spec_file>	The updated spec file
-i cv_rules_file	CloudVector rules file. Note that this is optional.
-r custom_rules_file	The custom rules file. Note that this is optional.
-o output_file	The output file to capture the evaluation results

- **Identify violations in a spec:**

```
$ cvapirisk eval_risk -s orangebank.json -i cv_rules.json -o cvreport.json
...
```

- **Identify violations in a spec using ONLY custom rules:**

```
$ cvapirisk eval_risk -s orangebank.json -o cvreport.json -r custom_rules.json
...
```

- **Identify violations across a collection of OpenAPI specs, using CV rules AND custom rules:**

```
$ cvapirisk eval_risk -z orangebank_specs.zip -o cvreport.json -i cv_rules.json -r custom_rules.json
...
```

- **Compare two versions of the spec using ONLY CV rules:**

```
$ cvapirisk trend_risk orangebank_user_orig.json orangebank_user.json -i cv_rules.json -o cvreport.json
...
```

- **Run the tool with CICD triggers defined:**

```
$ cvapirisk eval_risk -z orangebank_specs.zip -o cvreport.json -i cv_rules.json -c cicd.cfg
$ echo $?
```

Note that the tool outputs violations on stdout and also writes them to the output json file (*cvreport.json* in above example). The JSON output data can be persisted in a backend database for richer analytics.

- **Using the cvapirisk API server:**

```
$ cvapiriskserver -c apisparc_server.cfg
```

Please find the config file `apisparc_server.cfg` in the installation folder. Typically, in a Python virtual environment installation of the tool, the file can be found at `lib/python3.x/site-packages/cvsvc_apirisk/score/spec_security/apisparc_server.cfg`

Note that the rules file(s) can be pre-loaded at a specific location and then referenced when making client requests.

Once the server is started, an API spec can be evaluated as:

```
$ curl -X POST http://localhost:8500/eval_risk -d '{"spec_url":
"file:///tmp/orangebank_stores.json",                "cv_rules_path":
"/tmp/cv_rules.json", "custom_rules_path": null}
```

The `spec_url` parameter can also refer to a specification residing over the web. That is, the URL starts with "`http(s)://`".

Support

For further troubleshooting and support, please reach out to your customer success manager or email us at support@cloudvector.com.