cuckoo-unittest Documentation

Release 1.0.0

RicoVZ

Feb 14, 2018

Contents

This piece of documentation will teach you how to write a unit test for Cuckoo-unittest.

After reading the pages in the documentation you will be able to:

- Create a test file
- What to put into each test
- Automatically verify that your test file is free of errors

The unit test files are used to tell Cuckoo-unittest what to look for in the Cuckoo log. A unit test file is written in JSON. Each test file consists of two sections; a section that contains information about the test, and a section that contains the actual tests.

Contents:

CHAPTER 1

Creating tests

This part of the documentation will explain the required fields and the tests that are possible.

Each test file needs to have **exactly two keys at its root**; the '**test_info**' and the '**tests**' key.

The 'test_info' key contains a dictionary of information about this test and information needed to run this test. The 'tests' key is a dictionary that contains the actual tests.

Ok! Let's get started:

1. Open a new file.
2. Add the JSON dictionaries stated in the example.
3. Think of a name for your test. **For what malware sample is this test?** Use that as a name for the test.
4. Save your new file in the **tests directory** of Cuckoo-unittest **with a .json extension**.

Example: the filename is Cryptolocker-test.json

```
    {
            "test_info": {
            },
            "tests": {
            }
    }
```

1.1 test_info keys

At this time, there are six keys that you can use in test_info, **of which five are required**.

The possible and needed keys are:

- test_name (Required)
- sample_filename (Required)
- sample_sha1 (Required)

- sample_sha256 (Required)
- sample_url (Required)
- cuckoo_options (Recommended)

All of these keys should have a string as a value.

A simple description of each key is provided below

1.1.1 test_name

This key should contain the name of the test.

Value type: String

Best practise is to **keep this equal to the file name of the test**.

Example:

```
    {
            "test_info": {
                    "test_name":"SomeTestName"
            },
            "tests": {
            }
    }
```

1.1.2 sample_filename

This key is very important. It tells Cuckoo-unittest what the name of the malware sample in the sample directory is. So this is the name of the file that is submitted to Cuckoo for analysis. It should contain a string name of the file.

–> Do not upload the malware sample! <–

Value type: String

Best practise is using **the sha256 hash of the malware sample as a filename for it**.

Example:

```
    {
            "test_info": {
                    "test_name":"SomeTestName",
                    "sample_filename":
↪"5db1fee4b5703808c48078a76768b155b421b210c0761cd6a5d223f4d99f1eaa"
            },
            "tests": {
            }
    }
```

1.1.3 sample_sha1

This key should have the sha1 hash of the malware sample.

Value type: String

Example:

```
    {
            "test_info": {
                    "test_name":"SomeTestName",
                    "sample_filename":
↪"5db1fee4b5703808c48078a76768b155b421b210c0761cd6a5d223f4d99f1eaa",
                    "sample_sha1":"77ba9cd915c8e359d9733edcfe9c61e5aca92afb"
            },
            "tests": {
            }
    }
```

1.1.4 sample_sha256

This key should have the sha256 hash of the malware sample.

Value type: String

Example:

```
    {
            "test_info": {
                    "test_name":"SomeTestName",
                    "sample_filename":
↪"5db1fee4b5703808c48078a76768b155b421b210c0761cd6a5d223f4d99f1eaa",
                    "sample_sha1":"77ba9cd915c8e359d9733edcfe9c61e5aca92afb",
                    "sample_sha256":
↪"5db1fee4b5703808c48078a76768b155b421b210c0761cd6a5d223f4d99f1eaa"
            },
            "tests": {
            }
    }
```

1.1.5 sample_url

This key is important, it is a key that is used to store a URL of a sample. The Cuckoo webpanel gives the option to download the malware sample. You can use the link for that download in the sample_url.

This key can be used to automatically download the needed samples for tests that have this key.

Only add sample URLs from trusted sources!

Value type: String

Example:

```
    {
            "test_info": {
                    "test_name":"SomeTestName",
                    "sample_filename":
↪"5db1fee4b5703808c48078a76768b155b421b210c0761cd6a5d223f4d99f1eaa",
                    "sample_sha1":"77ba9cd915c8e359d9733edcfe9c61e5aca92afb",
                    "sample_sha256":
↪"5db1fee4b5703808c48078a76768b155b421b210c0761cd6a5d223f4d99f1eaa",
                    "sample_url":"http://yourtrustedcuckooserver.nl/file/
↪sample/560febba1a35004e8d1064b5"
            },
```

(continues on next page)

(continued from previous page)

```
        "tests": {
        }
}
```

1.1.6 cuckoo_options

This key can contain any options that Cuckoo might have and is like using the command line "–options" parameter. The options will be send to Cuckoo specifically for the samply belonging to this test. Seperate each option by a comma.

It is recommended to use the "json.calls=0" option. This option tells Cuckoo not to store all function calls in a report. This saves a lot of time and a lot of disk space. Using this key does not affect the tests in any way.

Value type: String

Example:

```
{
        "test_info": {
                "test_name":"SomeTestName",
                "sample_filename":
↪"5db1fee4b5703808c48078a76768b155b421b210c0761cd6a5d223f4d99f1eaa",
                "sample_sha1":"77ba9cd915c8e359d9733edcfe9c61e5aca92afb",
                "sample_sha256":
↪"5db1fee4b5703808c48078a76768b155b421b210c0761cd6a5d223f4d99f1eaa",
                "sample_url":"http://yourtrustedcuckooserver.nl/file/
↪sample/560febba1a35004e8d1064b5"
                "cuckoo_options":"json.calls=0,free=yes"
        },
        "tests": {
        }
}
```

1.2 Adding tests

This section explains how to use the various types of tests that are available. You can add as many tests to one test file as you want.

The type of data type differs per test. The data type that you should use for the test is documented at each test.

The available tests are:

- check_md5 - *Type: String*
- check_sha1 - *Type: String*
- check_sha256 - *Type: String*
- check_sha512 - *Type: String*
- check_expected_api_calls - *Type: Dictionary: String:Integer*
- check_expected_regkeys_opened - *Type: List: String*
- check_expected_regkeys_read - *Type: List: String*
- check_expected_regkeys_written - *Type: List: String*

- check_expected_regkeys_deleted - *Type: List: String*
- check_expected_files_opened - *Type: List: String*
- check_expected_files_read - *Type: List: String*
- check_expected_files_written - *Type: List: String*
- check_expected_files_deleted - *Type: List: String*
- check_expected_mutexes_created - *Type: Dictionary: String:String*
- check_expected_ips_connected - *Type: List: String*
- check_expected_hosts_connected - *Type: List: String*
- check_expected_processes - *Type: Dictionary: String:Integer*
- check_expected_dll_loaded - *Type: List: String*
- check_expected_monitor_log_lines - *Type: Dictionary: String:dictionary(String:integer)*

Each test key in the 'tests' key should be seperated by a comma. Like so:

```
{
        "test_info": {
        },
        "tests": {
                "check_md5":"",
                "check_sha256":"",
                "check_expected_api_calls": {
                },
                "check_expected_ips_connected": [
                ]
        }
}
```

1.2.1 tests

This section explains what each test key does and how to use it.

Some tests have special features like; using wilcards (? and *) and using regular expressions.

'*' is a wildcard for any length of characters.

'?' is a wildcard for a single character

At each test it is stated what features the tests supports.

As stated before, each test needs a specific type of test data format. The formats that are currently used by the tests are: Strings, lists, and dictionaries.

Each test on this page states what type of data it needs.

check_md5

This test checks if the given md5 hash matches the md5 hash calculated by Cuckoo.

Data type: String

Support information:

- Wilcards: No

- Regular expressions: No

Example:

```
    {
            "test_info": {
            },
            "tests": {
                    "check_md5":"e48e13207341b6bffb7fb1622282247b"
            }
    }
```

check_sha1

This test checks if the given sha1 hash matches the sha1 hash calculated by Cuckoo.

Data type: String

Support information:

- Wilcards: No
- Regular expressions: No

Example:

```
    {
            "test_info": {
            },
            "tests": {
                    "check_sha1":"77ba9cd915c8e359d9733edcfe9c61e5aca92afb"
            }
    }
```

check_sha256

This test checks if the given sha256 hash matches the sha256 hash calculated by Cuckoo.

Data type: String

Support information:

- Wilcards: No
- Regular expressions: No

Example:

```
    {
            "test_info": {
            },
            "tests": {
                    "check_sha256":
↪"5db1fee4b5703808c48078a76768b155b421b210c0761cd6a5d223f4d99f1eaa"
            }
    }
```

check_sha512

This test checks if the given sha512 hash matches the sha512 hash calculated by Cuckoo.

Data type: String

Support information:

- Wilcards: No
- Regular expressions: No

Example:

```
    {
            "test_info": {
            },
            "tests": {
                    "check_sha512":
↪"6f0ac65fe01188660aad900bfe16c566ebf0e56c0a7d4a15bd831049108de80bd3a2fbf1a8b91662433a40458ec20
↪"
            }
    }
```

check_expected_api_calls

This test checks if the expected api calls were used. You can specify multiple api calls and for each call how many times you expect it to be called.

Data type: Dictionary with string key and integer value

- Wilcards: No
- Regular expressions: No

Example:

```
    {
            "test_info": {
            },
            "tests": {
                    "check_expected_api_calls": {
                            "LdrLoadDll": 7,
                            "ReadProcessMemory": 21,
                            "NtCreateMutant": 2,
                            "LdrUnloadDll": 7
                    }
            }
    }
```

check_expected_regkeys_opened

This test checks if the specified registry keys were opened. You can specify multiple registry keys.

Data type: List with strings

- Wilcards: Yes
- Regular expressions: No

Example:

```
    {
            "test_info": {
            },
            "tests": {
                    "check_expected_regkeys_opened": [
                            "HKEY_CURRENT_USER\\Software\\*",
                            "HKEY_LOCAL_
↪MACHINE\\SOFTWARE\\dd20ce73d7\\30e5b11d",
                            "*\\SomeKey",
                            "HKEY_LOCAL_MACHINE\\SOFTWARE\\Number?"
                    ]
            }
    }
```

check_expected_regkeys_read

This test checks if the specified registry keys were read. You can specify multiple registry keys.

Data type: List with strings

- Wilcards: Yes
- Regular expressions: No

Example:

```
    {
            "test_info": {
            },
            "tests": {
                    "check_expected_regkeys_read": [
                            "HKEY_CURRENT_USER\\Software\\*",
                            "HKEY_LOCAL_
↪MACHINE\\SOFTWARE\\dd20ce73d7\\30e5b11d",
                            "*\\SomeKey",
                            "HKEY_LOCAL_MACHINE\\SOFTWARE\\Number?"
                    ]
            }
    }
```

check_expected_regkeys_written

This test checks if the specified registry keys were written. You can specify multiple registry keys.

Data type: List with strings

- Wilcards: Yes
- Regular expressions: No

Example:

```
    {
            "test_info": {
            },
            "tests": {
```

(continues on next page)

(continued from previous page)

```
                "check_expected_regkeys_written": [
                        "HKEY_CURRENT_USER\\Software\\*",
                        "HKEY_LOCAL_
↪MACHINE\\SOFTWARE\\dd20ce73d7\\30e5b11d",
                        "*\\SomeKey",
                        "HKEY_LOCAL_MACHINE\\SOFTWARE\\Number?"
                ]
        }
    }
```

check_expected_regkeys_deleted

This test checks if the specified registry keys were deleted. You can specify multiple registry keys.

Data type: List with strings

- Wilcards: Yes
- Regular expressions: No

Example:

```
    {
        "test_info": {
        },
        "tests": {
                "check_expected_regkeys_deleted": [
                        "HKEY_CURRENT_USER\\Software\\*",
                        "HKEY_LOCAL_
↪MACHINE\\SOFTWARE\\dd20ce73d7\\30e5b11d",
                        "*\\SomeKey",
                        "HKEY_LOCAL_MACHINE\\SOFTWARE\\Number?"
                ]
        }
    }
```

check_expected_mutexes_created

This test checks if the specified mutex or the specified pattern is found in the created mutexes.

Data type: Dictionary with string key and string value

- Wilcards: No
- Regular expressions: Yes

In this test, the key is the name or pattern of the mutex you expect to find. The value is a setting, this setting can be the value “regex_on” or “regex_off”. When using “regex_on”, the key will be read as a regular expression. If “regex_off” is used, it will be read as a normal string.

You can use the regular expressions in cases where a random mutex name is generated. But you might know that it always has a length of 16 characters.

For a guide on regular expressions, look here: python regular expressions

You can use pythex.org to test your regular expression. Make sure it only matches the values you expect.

Example:

```
{
        "test_info": {
        },
        "tests": {
                 "check_expected_mutexes_created": {
                        "^[A-Z0-9]{16}$": "regex_on",
                        "SuperAwesomeMutexName": "regex_off"
                 }
        }
}
```

check_expected_ips_connected

This test checks if the specified IP addresses were connected to.

Data type: List with strings

- Wilcards: No
- Regular expressions: No

Example:

```
{
        "test_info": {
        },
        "tests": {
                "check_expected_ips_connected": [
                        "8.8.8.8",
                        "144.55.48.7"
                ]
        }
}
```

check_expected_hosts_connected

This test checks if the specified hostnames were connected to.

Data type: List with strings

- Wilcards: No
- Regular expressions: No

Example:

```
{
        "test_info": {
        },
        "tests": {
                "check_expected_hosts_connected": [
                        "malicious.something.com",
                        "virus.ru"
                ]
        }
}
```

check_expected_processes

This test checks if the processes and amount of them were started.

Data type: Dictionary with string key and integer value

- Wilcards: No
- Regular expressions: No

The key is the name of the process you expect to be started. The value is the amount of times you expect the process to be started.

Example:

```
    {
            "test_info": {
            },
            "tests": {
                     "check_expected_processes": {
                             "regsvr32.exe": 2,
                             "virus.exe": 1
                     }
            }
    }
```

check_expected_dll_loaded

This test checks if the specified DLLs were loaded

Data type: List with strings

- Wilcards: Yes
- Regular expressions: No

The value of this test should contain a path to a DLL. If you don't know the path, use the * wildcard.

Example:

```
    {
            "test_info": {
            },
            "tests": {
                    "check_expected_dll_loaded": [
                            "*SomeDLL.dll",
                            "C:\\WINDOWS\\system32\\wininet.dll"
                    ]
            }
    }
```

check_expected_monitor_log_lines

This test checks the output of the debug log that was generated by Cuckoo and can be used to say that a certain log line should exist or not, how many times etc. It is possible to use comparison operators to specify the amount you expect.

Data type: Dictionary with String:dictionary(String:integer) values

- Wilcards: Yes

- Regular expressions: No

The value of this test should contain some or part of a debug log line.

Possible comparison operators:

- ==
- >
- <
- >=
- <=

Example:

```
    {
            "test_info": {
            },
            "tests": {
                      "check_expected_monitor_log_lines": {
                                  "*INFO:Test passed:*": {">=": 4},
                                  "*CRITICAL:Test didn't pass:*": {"==": 0}
↪,
                                  "*Starting analyzer*": {"<": 4},
                                  "*lib.common.registry*": {">=": 5}
                      }
            }
    }
```

CHAPTER 2

Utilities

The **utils** folder contains a set of tools which are meant to make life a little bit easier.

The tools that are avaiable are:

- TestChecker.py - Checks tests for valid JSON, use of correct test keys and test data types.
- GetSample.py - Downloads the sample files for the specified test(s). Uses the "sample_url" key from "test_info"

2.1 TestChecker

This tool checks if a test file:

- Consists of valid JSON
- Contains the mandatory test_info keys
- Contains existing test keys
- Has the correct test data type for tests. (*Dictionary, String, List etc..*)

It reports all errors it can find. If the error is a JSON error, run the checker again after fixing it.

This tool can be given a test file or a directory containing test files.

Usage:

```
$ ./utils/TestChecker.py test_file/tests_dir
```

Checking a single test file:

```
$ ./utils/TestChecker.py tests/sometest.json
```

This will check the specified test file for errors and report them.

Check all test files in a directory:

```
$ ./utils/TestChecker.py tests
```

This will check all test files in the specified directory for errors and report them.

2.2 GetSample

This tool downloads the in "sample_url" specified sample for the specified test and stores it in the specified directory and **writes the filename to the sample_filename key in test_info**.

This tool can be given a test file or a directory containing test files.

Usage:

```
$ ./utils/GetSample.py sample_dir test_file/test_dir
```

Downloading a single sample:

```
$ ./utils/GetSample.py samples tests/sometest.json
```

This will download the sample for sometest.json to the samples directory.

Downloading samples for all tests in a directory:

```
$ ./utils/GetSample.py samples tests
```

This will download samples for all the tests in the tests directory and store them in the samples directory.