# plyara Documentation

**plyara**

**Apr 24, 2021**

# Contents

Parse YARA rules into a dictionary representation.

Plyara is a script and library that lexes and parses a file consisting of one more YARA rules into a python dictionary representation. The goal of this tool is to make it easier to perform bulk operations or transformations of large sets of YARA rules, such as extracting indicators, updating attributes, and analyzing a corpus. Other applications include linters and dependency checkers.

Plyara leverages the Python module PLY for lexing YARA rules.

This is a community-maintained fork of the original plyara by 8u1a. The "plyara" trademark is used with permission.

CHAPTER 1

# Installation

Plyara requires Python 3.6+.

Install with pip:

```
pip3 install plyara
```

CHAPTER 2

# Usage

Use the plyara Python library in your own applications:

```
>>> import plyara
>>> parser = plyara.Plyara()
>>> mylist = parser.parse_string('rule MyRule { strings: $a="1" \n condition: false }
↪')
>>>
>>> import pprint
>>> pprint.pprint(mylist)
[{'condition_terms': ['false'],
  'raw_condition': 'condition: false ',
  'raw_strings': 'strings: $a="1" \n ',
  'rule_name': 'MyRule',
  'start_line': 1,
  'stop_line': 2,
  'strings': [{'name': '$a', 'type': 'text', 'value': '1'}]}]
>>>
```

Or, use the included `plyara` script from the command line:

```
$ plyara -h
usage: plyara [-h] [--log] FILE

Parse YARA rules into a dictionary representation.

positional arguments:
  FILE        File containing YARA rules to parse.

optional arguments:
  -h, --help  show this help message and exit
  --log       Enable debug logging to the console.
```

The command-line tool will print valid JSON output when parsing rules:

```
$ cat example.yar
rule silent_banker : banker
{
    meta:
        description = "This is just an example"
        thread_level = 3
        in_the_wild = true
    strings:
        $a = {6A 40 68 00 30 00 00 6A 14 8D 91}
        $b = {8D 4D B0 2B C1 83 C0 27 99 6A 4E 59 F7 F9}
        $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"
    condition:
        $a or $b or $c
}

$ plyara example.yar
[
    {
        "condition_terms": [
            "$a",
            "or",
            "$b",
            "or",
            "$c"
        ],
        "metadata": [
            {
                "description": "This is just an example"
            },
            {
                "thread_level": 3
            },
            {
                "in_the_wild": true
            }
        ],
        "raw_condition": "condition:\n        $a or $b or $c\n",
        "raw_meta": "meta:\n        description = \"This is just an example\"\n
↪ thread_level = 3\n        in_the_wild = true\n    ",
        "raw_strings": "strings:\n        $a = {6A 40 68 00 30 00 00 6A 14 8D 91}\n
↪     $b = {8D 4D B0 2B C1 83 C0 27 99 6A 4E 59 F7 F9}\n        $c = \
↪"UVODFRYSIHLNWPEJXQZAKCBGMT\"\n    ",
        "rule_name": "silent_banker",
        "start_line": 1,
        "stop_line": 13,
        "strings": [
            {
                "name": "$a",
                "type": "byte",
                "value": "{6A 40 68 00 30 00 00 6A 14 8D 91}"
            },
            {
                "name": "$b",
                "type": "byte",
                "value": "{8D 4D B0 2B C1 83 C0 27 99 6A 4E 59 F7 F9}"
            },
            {
```

(continues on next page)

(continued from previous page)

```
                "name": "$c",
                "type": "text",
                "value": "UVODFRYSIHLNWPEJXQZAKCBGMT"
            }
        ],
        "tags": [
            "banker"
        ]
    }
]
```

# CHAPTER 3

## Reusing The Parser

If you want to reuse a single instance of the parser object for efficiency when parsing large quantities of rule or rulesets, the new clear() method must be used.

```
rules = list()
parser = plyara.Plyara()

for file in files:
    with open(file, 'r') as fh:
        yararules = parser.parse_string(fh.read())
        rules += yararules
    parser.clear()
```

CHAPTER 4

# Migration

If you used an older version of plyara, and want to migrate to this version, there will be some changes required. Most importantly, the parser object instantiation has changed. It was:

```
# Old style - don't do this!
import plyara.interp as interp
rules_list = interp.parseString(open('myfile.yar').read())
```

But is now:

```
# New style - do this instead!
import plyara
parser = plyara.Plyara()
rules_list = parser.parse_string(open('myfile.yar').read())
```

The existing parsed keys have stayed the same, and new ones have been added.

When reusing a `parser` for multiple rules and/or files, be aware that imports are now shared across all rules - if one rule has an import, that import will be added to all rules in your parser object.

CHAPTER 5

# Contributing

- If you find a bug, or would like to see a new feature, Pull Requests and Issues are always welcome.
- By submitting changes, you agree to release those changes under the terms of the LICENSE.
- Writing passing unit tests for your changes, while not required, is highly encouraged and appreciated.
- Please run all code contributions through each of the linters that we use for this project: pycodestyle, pydocstyle, and pyflakes. See the .travis.yml file for exact use. For more information on these linters, please refer to the Python Code Quality Authority: http://meta.pycqa.org/en/latest/

CHAPTER 6

# Discussion

- You may join our IRC channel on irc.freenode.net #plyara

# CHAPTER 7

# Module Documentation

**class** plyara.**Plyara**(*console_logging=False*, *store_raw_sections=True*, *meta_as_kv=False*)
Bases: *plyara.core.Parser*

Define the lexer and the parser rules.

**class** plyara.core.**Parser**(*console_logging=False*, *store_raw_sections=True*, *meta_as_kv=False*)
Bases: object

Interpret the output of the parser and produce an alternative representation of YARA rules.

plyara utility functions.

This module contains various utility functions for working with plyara output.

plyara.utils.**is_valid_rule_name**(*entry*)
Check to see if entry is a valid rule name.

**Args:** entry: String containing rule name.

**Returns:** bool

plyara.utils.**is_valid_rule_tag**(*entry*)
Check to see if entry is a valid rule tag.

**Args:** entry: String containing tag.

**Returns:** bool

plyara.utils.**detect_imports**(*rule*)
Take a parsed yararule and provide a list of required imports based on condition.

**Args:** rule: Dict output from a parsed rule.

**Returns:** list: Imports that are required.

plyara.utils.**detect_dependencies**(*rule*)
Take a parsed yararule and provide a list of external rule dependencies.

**Args:** rule: Dict output from a parsed rule.

**Returns:** list: External rule dependencies.

plyara.utils.**generate_logic_hash**(*rule*)
Calculate hash value of rule strings and condition.

**Args:** rule: Dict output from a parsed rule.

**Returns:** str: Hexdigest SHA-256.

plyara.utils.**generate_hash**(*rule*, *secure_hash=None*)
Calculate a secure hash of the logic in the rule strings and condition.

If the resultant hashes are identical for two YARA rules, the rules will match on identical content. The reverse it not true, so two rules that match the same content may not generate the same hash. For example, if a rule only contains one string, the logic for 'any of' and 'all of' generate different hashes, but the rules contain the same logic.

**Args:** rule: Dict output from a parsed rule. secure_hash: Alternate hash function, defaults to SHA-256

**Returns:** str: hexdigest

plyara.utils.**rebuild_yara_rule**(*rule*, *condition_indents=False*)
Take a parsed yararule and rebuild it into a usable one.

**Args:** rule: Dict output from a parsed rule. condition_indents: Use nested indentation for condition

**Returns:** str: Formatted text string of YARA rule.

CHAPTER 8

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## p

# Index