

# JAST: Application Examples of our Modules

A purely syntactic AST-based analysis of JavaScript samples is performed. This study is based on a frequency analysis of the n-grams present in the considered files to classify them in two categories: benign or malicious. The default n value is 4, but it can be changed while giving command line inputs (`--n` option), provided it stays consistent during the whole analysis process. Each n-gram is mapped to a consistent dimension of a vector space either using a mapping dictionary or hashes (as defined by the *scikit-learn* `HashingVectorizer` function). The default value is to use a dictionary, but it can be changed while giving command line inputs (`--dnh` option), provided it stays consistent during the whole analysis process. Furthermore, a tolerant mode can be activated so that the *Esprima* parser accepts a few cases of syntactic errors (`--t` option).

We will consider here the following samples:

- the benign files `benign1`, `benign2` and `benign3`;
- the following malicious files `malicious1`, `malicious2` and `malicious3`;
- the benign folders `Dir-Benign1`, `Dir-Benign2`, `Dir-Benign3`;
- the malicious folders `Dir-Malicious1`, `Dir-Malicious2` and `Dir-Malicious3`;
- the unknown files `unknown1`, `unknown2`;
- and the unknown folders `Dir-Unknown1` and `Dir-Unknown2`.

Each name representing an hypothetical path to the files/folders you wish to analyze.

## Testing the Installation of the Modules

To test if the modules are correctly installed, you can try the following command:

```
$ python3 <path-of-js/is_js.py> --d <path-of-features/parsing/>
```

The two files contained in this *parsing/* folder should be recognized as valid JavaScript. If this is not the case, have a look at the log file *is\_js.log* at the repository's root. If the error is due to *Esprima* not being installed, check the commented section of *install.sh* to install it.

## JavaScript Detection Tool

*Detection of JavaScript samples respecting the grammar defined by ECMA-International, detection of broken JavaScript, and files not written in JavaScript.*

### Help

```
$ python3 <path-of-js/is_js.py> --help
```

Mandatory attributes are `--f <list-of-files>` or `--d <list-of-directories>`. The two options can be combined.

```
usage: is_js.py [-h] [--f FILE [FILE ...]] [--d DIR [DIR ...]] [--v VERBOSITY]
```

```
Given a list of directory, or of file paths, indicates whether the files are
either valid ('<fileName>: valid JavaScript'), malformed ('<fileName>:
malformed JavaScript'), or no JavaScript ('<fileName>: not JavaScript').
```

```
optional arguments:
```

```
-h, --help            show this help message and exit
--f FILE [FILE ...]  files to be tested
--d DIR [DIR ...]    directories to be tested
--v VERBOSITY         controls the verbosity of the output, from 0 (verbose)
                      to 5 (less verbose)
```

### To analyze files

```
$ python3 <path-of-js/is_js.py> --f benign1 malicious1
```

### To analyze directories

```
$ python3 <path-of-js/is_js.py> --d Dir-Malicious1 Dir-Unknown1 Dir-Unknown2
```

### To analyze files and directories

```
$ python3 <path-of-js/is_js.py> --f benign1 malicious1 --d Dir-Malicious1 Dir-Unknown1 Dir-Unknown2
```

## Clustering of JavaScript Samples

*Clustering of JavaScript samples into  $k$  (configurable) families.*

### Help

```
$ python3 <path-of-clustering/cluster.py> --help
```

Mandatory attributes are: `--f <list-of-files>` (or `--d <list-of-directories>`, the two options can be combined);  
and `--c <number-of-clusters>`.

```
usage: cluster.py [-h] [--d DIR [DIR ...]] [--f FILE [FILE ...]] [--c INTEGER]
                  [--g BOOL] [--t TOLERANT] [--n INTEGER] [--dnh BOOL]
                  [--v VERBOSITY]
```

Given a list of repository or file paths, clusters the JS inputs into several families.

#### optional arguments:

<code>-h, --help</code>	show this help message and exit
<code>--d DIR [DIR ...]</code>	directories containing the JS files to be clustered
<code>--f FILE [FILE ...]</code>	files to be analyzed
<code>--c INTEGER</code>	number of clusters
<code>--g BOOL</code>	produces a 2D representation of the files from the JS corpus
<code>--t TOLERANT</code>	tolerates a few cases of syntax errors
<code>--n INTEGER</code>	stands for the size of the sliding-window which goes through the units contained in the files to be analyzed
<code>--dnh BOOL</code>	the $n$ -grams are mapped to integers using a dictionary and not hashes
<code>--v VERBOSITY</code>	controls the verbosity of the output, from 0 (verbose) to 5 (less verbose)

**Clustering** of the given files in 5 clusters with k-means++ algorithm

```
$ python3 <path-of-clustering/cluster.py> --f benign1 malicious1 --d Dir-Malicious1 Dir-Unknown1 Dir-Unknown2 --c 5
```

## Classification of JavaScript Samples

*Detection of malicious JavaScript documents.*

### ● Creating a Model

#### Help

```
$ python3 <path-of-clustering/learner.py> --help
```

Mandatory attributes are: `--d <list-of-directories>` (or `--f <list-of-files>`, the two options can be combined);  
and `--l <list-of-directory-labels>` (or `--lf <list-of-file-labels>` according to the input).

**To create a model using a directory containing malicious files, and another one with benign files**

```
$ python3 <path-of-clustering/learner.py> --d Dir-Malicious1 Dir-Benign1 --l 'malicious' 'benign'
```

**To specify a model path** (`--md` option) **and/or a model name** (`--mn` option)

```
$ python3 <path-of-clustering/learner.py> --d Dir-Malicious1 Dir-Benign1 --l 'malicious' 'benign' --md 'Test/' --mn 'model1'
```

```
usage: learner.py [-h] [--d DIR [DIR ...]] [--l LABEL [LABEL ...]]
                [--f FILE [FILE ...]] [--lf LABEL [LABEL ...]]
                [--md MODEL-DIR] [--mn MODEL-NAME] [--ps BOOL] [--pr BOOL]
                [--nt NB_TREES] [--t TOLERANT] [--n INTEGER] [--dnh BOOL]
                [--v VERBOSITY]
```

Given a list of directory or file paths, builds a model to classify future JS inputs.

```
optional arguments:
  -h, --help            show this help message and exit
  --d DIR [DIR ...]     directories to be used to build a model from
  --l LABEL [LABEL ...] labels of the JS directories used to build a model
                        from
  --f FILE [FILE ...]   files to be used to build a model from
  --lf LABEL [LABEL ...] labels of the JS files used to build a model from
                        path to store the model that will be produced
  --md MODEL-DIR        name of the model that will be produced
  --mn MODEL-NAME        indicates whether to print or not the classifier's
                        detection rate
  --ps BOOL             indicates whether to print or not the classifier's
                        predictions
  --pr BOOL             number of trees in the forest
  --nt NB_TREES         tolerates a few cases of syntax errors
  --t TOLERANT           stands for the size of the sliding-window which goes
                        through the units contained in the files to be
                        analyzed
  --n INTEGER           the n-grams are mapped to integers using a dictionary
                        and not hashes
  --dnh BOOL            controls the verbosity of the output, from 0 (verbose)
                        to 5 (less verbose)
```

## • Updating a Model

### Help

\$ python3 <path-of-clustering/updater.py> --help

Mandatory attributes are: --d <list-of-directories> (or --f <list-of-files>, the two options can be combined);

and --l <list-of-directory-labels> (or --lf <list-of-file-labels> according to the input);

and --m <old-model-path> (see the previous point to create a model).

```
usage: updater.py [-h] [--d DIR [DIR ...]] [--l LABEL [LABEL ...]]
                 [--f FILE [FILE ...]] [--lf LABEL [LABEL ...]]
                 [--m OLD-MODEL] [--md MODEL-DIR] [--mn MODEL-NAME]
                 [--at NB_TREES] [--t TOLERANT] [--n INTEGER] [--dnh BOOL]
                 [--v VERBOSITY]
```

Given a list of directory or file paths, updates a model to classify future JS inputs.

```
optional arguments:
  -h, --help            show this help message and exit
  --d DIR [DIR ...]     directories to be used to update a model with
  --l LABEL [LABEL ...] labels of the JS directories used to update a model
                        with
  --f FILE [FILE ...]   files to be used to update a model with
  --lf LABEL [LABEL ...] labels of the JS files used to update a model with
                        path of the old model you wish to update with new JS
                        inputs
  --m OLD-MODEL         path to store the model that will be produced
  --md MODEL-DIR        name of the model that will be produced
  --mn MODEL-NAME        number of trees to be added into the forest
  --at NB_TREES         tolerates a few cases of syntax errors
  --t TOLERANT           stands for the size of the sliding-window which goes
                        through the units contained in the files to be
                        analyzed
  --n INTEGER           the n-grams are mapped to integers using a dictionary
                        and not hashes
  --dnh BOOL            controls the verbosity of the output, from 0 (verbose)
                        to 5 (less verbose)
```

### To update a model with benign and malicious files

\$ python3 <path-of-clustering/updater.py> --f malicious1 malicious2 benign1 --lf 'malicious' 'malicious' 'benign' --m <path-of-the-model-to-be-updated>

To specify a path for the new model (--md option) and/or a model name (--mn option)  
\$ python3 <path-of-clustering/updater.py> --d Dir-Malicious1 Dir-Benign1 --l 'malicious' 'benign' --m <path-of-the-model-to-be-updated> --md 'Test/' --mn 'model2'

## • Testing a Model / Classifying new Files

### Help

\$ python3 <path-of-clustering/classifier.py> --help

Mandatory attributes are: --d <list-of-directories> (or --f <list-of-files>, the two options can be combined);  
and --m <path-of-the-model-to-be-used> (see how to create a model, previously).

```
usage: classifier.py [-h] [--d DIR [DIR ...]] [--l LABEL [LABEL ...]]
                  [--f FILE [FILE ...]] [--lf LABEL [LABEL ...]]
                  [--m MODEL] [--th THRESHOLD] [--t TOLERANT] [--n INTEGER]
                  [--dnh BOOL] [--v VERBOSITY]

Given a list of directory or file paths, detects the malicious JS inputs.

optional arguments:
  -h, --help            show this help message and exit
  --d DIR [DIR ...]     directories containing the JS files to be analyzed
  --l LABEL [LABEL ...] labels of the JS directories to evaluate the model
                        from
  --f FILE [FILE ...]   files to be analyzed
  --lf LABEL [LABEL ...] labels of the JS files to evaluate the model from
                        path of the model used to classify the new JS inputs
                        (see >$ python3 <path-of-clustering/learner.py> -help)
                        to build a model)
  --m MODEL              threshold over which all samples are considered
                        malicious
  --th THRESHOLD         tolerates a few cases of syntax errors
  --t TOLERANT           stands for the size of the sliding-window which goes
                        through the units contained in the files to be
                        analyzed
  --n INTEGER            the n-grams are mapped to integers using a dictionary
                        and not hashes
  --dnh BOOL             controls the verbosity of the output, from 0 (verbose)
                        to 5 (less verbose)
```

### To detect malicious JS samples

\$ python3 <path-of-clustering/classifier.py> --d Dir-Unknown1 Dir-Unknown2 --m <path-of-the-model-to-be-used>

A list of the files tested (only those respecting the grammar defined by *ECMA-International*) will be returned.  
For each file, you will see whether it was classified as 'benign' or as 'malicious'.

## A Word about JAST used for the Detection of Malicious JavaScript Samples

The classifier used to determine if a sample is malicious or benign is Random Forest.

Random Forest is a combination of tree predictors which vote for the most popular class: an input is entered at the top of each tree and as it traverses down the trees, the data gets bucketed into smaller and smaller sets.

### Algorithm:

- Creation of  $N$  bootstrap samples from the original data;
- For each bootstrap sample, an unpruned classification tree is grown, such as at each node:

- a small subset of  $m$  variables is chosen at random;
- the predictor variable providing the best split is used to do a binary split on the node;
- at the next node, another predictor variable is chosen and the process iterated.

- Prediction of new data by aggregating the predictions of each tree.

The default parameters chosen for Random Forest are indicated on the [scikit-learn webpage](#), except for `n_estimators` (500 trees), `max_depth` (each tree has a maximum depth of 50 nodes), `random_state` (seed 0)

and `n_jobs` (to run in parallel as many jobs as cores). This optimal tuple of hyperparameters have been obtained using random and grid search with 5-fold cross-validation on an independent data set containing 17,500 unique JavaScript samples (half of which were benign, the other half being malicious).

Initially, a forest of 500 trees is created when the `learner` function is called. Each time the `updater` function is called, 100 trees are added to the previous forest. Therefore, we suggest the user to be **cautious** with the `updater` function and its default parameter, as it does not necessarily make sense to always add 100 trees to a previous forest, which could result in the model over- or under-fitting the datasets.

Last but not least, we consider that a sample is malicious if the probability of it being malicious, according to our Random Forest classifier, is bigger than a given threshold. The default value 0.29 has been chosen using Youden's J statistic and cross-validation on several data sets, and provides the best trade-off between false-positive rate and false-negative rate. This index can be increased to lower the false-positive rate (at the cost of the false-negative rate), or decreased to lower the false-negative rate (at the cost of the false-positive rate).

For more information, please refer to our paper.