OPGen

OPGen is a tool that generates object property graph (OPG) and analyze vulnerabilities for JavaScript.

Installation

OPGen requires Python 3.7+ and Node.js 12+. To set up the environment, simply run `install.sh`.

Command line arguments

Use the following arugments to run the tool:

```
generate_graph.py [-h] [-p] [-t VUL_TYPE] [-P] [-m] [-q] [-s] [-a]
                  [-f FUNCTION_TIMEOUT] [-c CALL_LIMIT]
                  [-e ENTRY_FUNC] [input_file]
```

Argument	Description
`input_file`	See subsection Input.
`-p, --print`	Print logs to console, instead of files.
`-t VUL_TYPE, --vul-type VUL_TYPE`	Set the vulnerability type to be checked. (See the Vulterability Types section.)
`-P, --prototype-pollution, --pp`	Shortcut for checking prototype pollution.
`-m, --module`	Module mode. Indicate the input is a module, instead of a script. This implies -a.
`--export LEVEL`	export the graph to Neo4J. The value can be 'light' or 'all'. Run import2neo4j.sh after the generation.
`-a, --run-all`	Run all exported functions in module.exports. By default, only main functions will be run.
`-q, --exit`	Exit the analysis immediately when vulnerability is found. Do not use this if you need a complete graph.
`-s, --single-branch`	Single branch mode (or single execution). Do not execute multiple branches in parallel.
`-f SEC, --function-timeout SEC`	Set the time limit when running all exported function, in seconds. (Defaults to no limit.) Do NOT use this parameter as it is very unstable.
`--run-env ENV_DIR`	set the running env location.
`--babel CONVERT_DIR`	set the dir to convert using babel.
`-c CALL_LIMIT, --call-limit CALL_LIMIT`	Set the how many times at most the same call statement can appear in the caller stack. (Defaults to 3.)
`-e ENTRY_FUNC, --entry-func ENTRY_FUNC`	Mannualy set the entry point function. Use this parameter only if you know which function to start the analysis with. This only affects the input module, i.e., dependent packages will not be affected.

Attention

Currently, for the packages that use CLASS, we need to use babel to convert them into ES5 format. To use babel, the prefix of babel path should be same to the prefix of input file. For example, if the babel path is /a/b/c, the input file should be under /a/b/c/. As for the input file, /a/b/c/index.js works for the input file, but ~/c/index.js does not.

For example:

```
$ ./generate_opg.py -t os_command ./chas_class/main.js --babel ./chas_class/
```

Examples

```
$ ./generate_opg.py ./tests/test.js -m -t os_command
```

For the modified version of challenge example, you can simply run

```
$ ./generate_opg.py -t os_command ./chas/main.js
```

For the original version of challenge example, you can simply run

```
$ ./generate_opg.py -t os_command ./chas_class/main.js --babel ./chas_class/
```