Project information

Created on
November 29, 2024

- Adds testing examples
Pedro Adão authored 2 weeks ago

Name	Last commit	Last update
slices	- Adds testing examples	2 weeks ago
README.md	- Adds testing examples	2 weeks ago

README.md

Discovering vulnerabilities in Javascript web applications

Change Logs

- 07Dec2024: Example slices and expected outputs and validating script are published.
- 07Dec2024: Outputs are Python dictionaries and were wrongly presented in previous version of `README.md`. Use the `.toDict()` method to obtain the AST as a Python dictionary. Details here.

1. Aims of this project

- To achieve an in-depth understanding of a security problem.
- To tackle the problem with a hands-on approach, by implementing a tool.
- To analyse a tool's underlying security mechanism according to the guarantees that it offers, and to its intrinsic limitations.
- To understand how the proposed solution relates to the state of the art of research on the security problem.
- To develop collaboration skills.

Components

The Project is presented in Section 2 as a problem, and its solution consists in the development and evaluation of a tool in Python, according to the Specification of the Tool, in groups of 3 students. Group members may be registered in different lab sessions, but they need to be registered in some lab.

A critical analysis of the experimental component is to be performed individually via a practical test Test.

Important dates and instructions

- Groups of 3 students should register in Fénix by **9 December 2024**.
- The submission deadline for the **code is 10 January 2025, 5pm**.
  - Please submit your code via your group's private repository at GitLab@RNL, under the appropriate Group number `https://gitlab.rnl.tecnico.ulisboa.pt/ssof2425/project/project-groups/GroupXX` (*to be created after registration in Fenix*).
  - The submissions should include all the necessary code, with all and any configuration in place for executing the tool according to the instructions in Specification of the Tool.
  - All tests that you would like to be considered for the evaluation of your tool should be made available in a common repository `https://gitlab.rnl.tecnico.ulisboa.pt/ssof2425/project/common-tests`. More info here.
- The submission deadline for the **5 Javascript patterns is 10 January 2025, 11:59pm**, via the same repository.
- The project will have a **practical assessment** on **15 January 2025**. For this test, students should be able to perform a critical analysis of their solution and answer questions regarding the experimental part of their project, as well as extend or adapt the solution to new requirements.

Authorship

Projects are to be solved in groups of 3 students. All members of the group are expected to be equally involved in solving, writing and presenting the project, and share full responsibility for all aspects of all components of the evaluation.

All sources should be adequately cited. Plagiarism will be punished according to the rules of the School.

2. Problem

A large class of vulnerabilities in applications originates in programs that enable user input information to affect the values of certain parameters of security sensitive functions. In other words, these programs encode a potentially dangerous information flow, in the sense that low integrity -- tainted -- information (user input) may interfere with high integrity parameters of sensitive functions **or variables** (so called sensitive sinks). This means that users are given the power to alter the behavior of sensitive functions **or variables**, and in the worst case may be able to induce the program to perform security violations. For this reason, such flows can be deemed illegal for their potential to encode vulnerabilities.

It is often desirable to accept certain illegal information flows, so we do not want to reject such flows entirely. For instance, it is useful to be able to use the inputted user name for building SQL queries. It is thus necessary to differentiate illegal flows that can be exploited, where a vulnerability exists, from those that are inoffensive and can be deemed secure, or endorsed, where there is no vulnerability. One approach is to only accept programs that properly sanitize the user input, and by so restricting the power of the user to acceptable limits, in effect neutralizing the potential vulnerability.

JavaScript has been the primary language for application development in browsers, but it is increasingly becoming popular on server side development as well. However, JavaScript suffers from vulnerabilities, such as cross-site scripting and malicious advertisement code on the client side, and SQL injection on the server side.

The aim of this project is to study how web vulnerabilities can be detected statically by means of taint and input sanitization analysis. We choose as a target web server and client side programs encoded in the JavaScript language.

The following references are mandatory reading about the problem:

- C. Staicu et. al, An Empirical Study of Information Flows in Real-World JavaScript, PLAS'2019.
- C. Staicu et. al, Extracting Taint Specifications for JavaScript Libraries, CSE'20.
- S. Guarnieri et. al, Saving the World Wide Web from Vulnerable JavaScript, ISSTA'11 and the companion IBM JavaScript Security Test Suite.

3. Specification of the Tool

The experimental part consists in the development of a static analysis tool for identifying data and information flow violations that are not protected in the program. In order to focus on the flow analysis, the aim is not to implement a complete tool. Instead, it will be assumed that the code to be analyzed has undergone a pre-processing stage to isolate, in the form of a program slice, a sequence of JavaScript instructions that are considered to be relevant to our analysis.

The following code slice, which is written in JavaScript, contains code lines which may impact a data flow between a certain source and a sensitive sink. The `URL` property of the `document` object (which can be accessed in a client side script) can be understood as an entry point. It uses the `document.write` to change the html page where it is embedded.

```
var pos = document.URL.indexOf("name=");
var name = document.URL.substring(pos + 5);
document.write(name);
```

Inspecting this slice it is clear that the program from which the slice was extracted could encode a DOM based XSS vulnerability. A victim can visit the page `http://www.vulnerable.com/welcome.html?name=<script>alert(document.cookie)</script>`, and the behavior of the webpage is modified and prints the cookies for the current site. However, sanitization of the untrusted input can remove the vulnerability:

```
var pos = document.URL.indexOf("name=");
var name = document.URL.substring(pos + 5);
var sanitizedName = DOMPurify.sanitize(name);
document.write(sanitizedName);
```

The aim of the tool is to search the slices for vulnerabilities according to inputted patterns, which specify for a given type of vulnerability its possible sources (a.k.a. entry points), sanitizers and sinks. A *pattern* is thus a 5-tuple with:

- name of vulnerability (e.g., DOM XSS)
- a list of sources (e.g., `document.URL`),
- a list of sanitization functions (e.g., `DOMPurify.sanitize`),
- a list of sensitive sinks (e.g., `document.write`),
- and a flag indicating whether implicit flows are to be considered.

it is returned by the function), *it should still report the vulnerability*, but also acknowledge the fact that its sanitization is possibly being addressed.

We provide program slices and patterns to assist you in testing the tool. It is however each group's responsibility to perform more extensive testing for ensuring the correctness and robustness of the tool. Note however that for the purpose of testing, the names of vulnerabilities, sources, sanitizers and sinks are irrelevant and do not need to be real vulnerabilities. In this context, you can produce your own patterns without specific knowledge of vulnerabilities, as this will not affect the ability of the tool to manage meaningful patterns. See examples in Section Input Vulnerability Patterns.

Running the tool

The tool should be called in the command line, and receive the following two arguments, and only the following two arguments:

- a path to a Javascript file containing the program slice to analyse;
- a path to a JSON file containing the list of vulnerability patterns to consider.

You can assume that the parsing of the JavaScript slices has been done, and that the input files are well-formed. The analysis should be fully customizable to the inputted vulnerability patterns described below. In addition to the entry points specified in the patterns, **by default any non-instantiated variable that appears in the slice should be considered as an entry point to all vulnerabilities being considered**.

The output should list the potential vulnerabilities encoded in the slice, and an indication of which sanitization functions(s) (if any) have been applied. The format of the output is specified below.

Your tool should be implemented in **Python, version >= 3.9.2**, and work in the following way:

1. be named `js_analyser.py`
2. be called in the command line with two arguments `<path_to_slice>/<slice>.js` and `<path_to_pattern>/<patterns>.json`
3. produce the output referred below and no other to a file named `<slice>.output.json` in the `./output/` folder.

For example

```
$ python ./js_analyser.py foo/slice_1.js bar/my_patterns.json
```

should analyse `slice_1.js` slice in folder `foo`, according to patterns in file `my_patterns.json` in folder `bar`, and output the result in file `./output/slice_1.output.json`.

NOTE: Examples of slices, patterns, outputs, and scripts that validate their correct format will be made available.

Input

Program slices

Your program should read from a text file (given as first argument in the command line) the representation of a JavaScript slice. See below how you can easily convert it into an Abstract Syntax Tree (AST).

Vulnerability patterns

The patterns are to be loaded from a file, whose name is given as the second argument in the command line. You can assume that pattern names are unique within a file.

An example JSON file with two patterns:

```
[
  {
    "vulnerability": "Command Injection",
    "sources": ["req.headers", "readFile"],
    "sanitizers": ["shell-escape"],
    "sinks": ["exec", "execSync", "spawnSync", "execFileSync"],
    "implicit": "no"
  },
  {
    "vulnerability": "DOM XSS",
    "sources": ["document.referrer", "document.URL", "document.location"],
    "sanitizers": ["DOMPurify.sanitize"],
    "sinks": ["document.write", "innerHTML", "setAttribute"],
    "implicit": "yes"
  }
]
```

The JavaScript file (given as first argument in the command line) containing the JavaScript slice should be converted into an Abstract Syntax Tree (AST).

You can use Python's `esprima` module to obtain the AST of the slice in the form of a dictionary.

```python
import esprima
import sys

with open(sys.argv[1], 'r') as f:
    program = f.read().strip()

ast_dict = esprima.parseScript(program, loc = True).toDict()
print(ast_dict)
```

The script above receives a filename of a javascript slice, and `ast_dict` is a Python dictionary encoding the AST that represents the code. The AST is represented in JSON, with further details available here. Below we provide 2 examples:

```javascript
alert("Hello World!");
```

is represented as

```json
{
    "type": "Program",
    "sourceType": "script",
    "body": [
        {
            "type": "ExpressionStatement",
            "expression": {
                "type": "CallExpression",
                "callee": {
                    "type": "Identifier",
                    "name": "alert",
                    "loc": {
                        "start": {
                            "line": 1,
                            "column": 0
                        },
                        "end": {
                            "line": 1,
                            "column": 5
                        }
                    }
                },
                "arguments": [
                    {
                        "type": "Literal",
                        "value": "Hello World!",
                        "raw": "\"Hello World!\"",
                        "loc": {
                            "start": {
                                "line": 1,
                                "column": 6
                            },
                            "end": {
                                "line": 1,
                                "column": 20
                            }
                        }
                    }
                ],
                "loc": {
                    "start": {
                        "line": 1,
                        "column": 0
                    },
                        "column": 21
                    }
                }
            },
            "loc": {
                "start": {
                    "line": 1,
                    "column": 0
                },
                "end": {
                    "line": 1,
                    "column": 22
                }
            }
        }
    ],
    "loc": {
        "start": {
            "line": 1,
            "column": 0
        },
        "end": {
            "line": 1,
            "column": 22
        }
    }
}
```

and the slice

```
var pos = document.URL.indexOf("name=");
var name = document.URL.substring(pos + 5);
var sanitizedName = DOMPurify.sanitize(name);
document.write(sanitizedName);
```

is represented as:

```
{
    "type": "Program",
    "sourceType": "script",
    "body": [
        {
            "type": "VariableDeclaration",
            "declarations": [
                {
                    "type": "VariableDeclarator",
                    "id": {
                        "type": "Identifier",
                        "name": "pos",
                        "loc": {
                            "start": {
                                "line": 1,
                                "column": 4
                            },
                            "end": {
                                "line": 1,
                                "column": 7
                            }
                        }
                    },
                    "init": {
                        "type": "CallExpression",
                        "callee": {
                            "type": "MemberExpression",
                            "computed": false,
        "computed": false,
        "object": {
            "type": "Identifier",
            "name": "document",
            "loc": {
                "start": {
                    "line": 1,
                    "column": 10
                },
                "end": {
                    "line": 1,
                    "column": 18
                }
            }
        },
        "property": {
            "type": "Identifier",
            "name": "URL",
            "loc": {
                "start": {
                    "line": 1,
                    "column": 19
                },
                "end": {
                    "line": 1,
                    "column": 22
                }
            }
        },
        "loc": {
            "start": {
                "line": 1,
                "column": 10
            },
            "end": {
                "line": 1,
                "column": 22
            }
        }
    },
    "property": {
        "type": "Identifier",
        "name": "indexOf",
        "loc": {
            "start": {
                "line": 1,
                "column": 23
            },
            "end": {
                "line": 1,
                "column": 30
            }
        }
    },
    "loc": {
        "start": {
            "line": 1,
            "column": 10
        },
        "end": {
            "line": 1,
            "column": 30
        }
    }
},
                                "type": "Literal",
                                "value": "name=",
                                "raw": "\"name=\"",
                                "loc": {
                                    "start": {
                                        "line": 1,
                                        "column": 31
                                    },
                                    "end": {
                                        "line": 1,
                                        "column": 38
                                    }
                                }
                            }
                        ],
                        "loc": {
                            "start": {
                                "line": 1,
                                "column": 10
                            },
                            "end": {
                                "line": 1,
                                "column": 39
                            }
                        }
                    },
                    "loc": {
                        "start": {
                            "line": 1,
                            "column": 4
                        },
                        "end": {
                            "line": 1,
                            "column": 39
                        }
                    }
                }
            ],
            "kind": "var",
            "loc": {
                "start": {
                    "line": 1,
                    "column": 0
                },
                "end": {
                    "line": 1,
                    "column": 40
                }
            }
        },
        {
            "type": "VariableDeclaration",
            "declarations": [
                {
                    "type": "VariableDeclarator",
                    "id": {
                        "type": "Identifier",
                        "name": "name",
                        "loc": {
                            "start": {
                                "line": 2,
                                "column": 4
                            },
                            "end": {
                                "line": 2,
                        }
                    },
                    "init": {
                        "type": "CallExpression",
                        "callee": {
                            "type": "MemberExpression",
                            "computed": false,
                            "object": {
                                "type": "MemberExpression",
                                "computed": false,
                                "object": {
                                    "type": "Identifier",
                                    "name": "document",
                                    "loc": {
                                        "start": {
                                            "line": 2,
                                            "column": 11
                                        },
                                        "end": {
                                            "line": 2,
                                            "column": 19
                                        }
                                    }
                                },
                                "property": {
                                    "type": "Identifier",
                                    "name": "URL",
                                    "loc": {
                                        "start": {
                                            "line": 2,
                                            "column": 20
                                        },
                                        "end": {
                                            "line": 2,
                                            "column": 23
                                        }
                                    }
                                },
                                "loc": {
                                    "start": {
                                        "line": 2,
                                        "column": 11
                                    },
                                    "end": {
                                        "line": 2,
                                        "column": 23
                                    }
                                }
                            },
                            "property": {
                                "type": "Identifier",
                                "name": "substring",
                                "loc": {
                                    "start": {
                                        "line": 2,
                                        "column": 24
                                    },
                                    "end": {
                                        "line": 2,
                                        "column": 33
                                    }
                                }
                            },
                            "loc": {
                                "start": {
                },
                "end": {
                    "line": 2,
                    "column": 33
                }
            }
        },
        "arguments": [
            {
                "type": "BinaryExpression",
                "operator": "+",
                "left": {
                    "type": "Identifier",
                    "name": "pos",
                    "loc": {
                        "start": {
                            "line": 2,
                            "column": 34
                        },
                        "end": {
                            "line": 2,
                            "column": 37
                        }
                    }
                },
                "right": {
                    "type": "Literal",
                    "value": 5,
                    "raw": "5",
                    "loc": {
                        "start": {
                            "line": 2,
                            "column": 40
                        },
                        "end": {
                            "line": 2,
                            "column": 41
                        }
                    }
                },
                "loc": {
                    "start": {
                        "line": 2,
                        "column": 34
                    },
                    "end": {
                        "line": 2,
                        "column": 41
                    }
                }
            }
        ],
        "loc": {
            "start": {
                "line": 2,
                "column": 11
            },
            "end": {
                "line": 2,
                "column": 42
            }
        }
    },
    "loc": {
        "start": {
                        },
                        "end": {
                            "line": 2,
                            "column": 42
                        }
                    }
                }
            ],
            "kind": "var",
            "loc": {
                "start": {
                    "line": 2,
                    "column": 0
                },
                "end": {
                    "line": 2,
                    "column": 43
                }
            }
        },
        {
            "type": "VariableDeclaration",
            "declarations": [
                {
                    "type": "VariableDeclarator",
                    "id": {
                        "type": "Identifier",
                        "name": "sanitizedName",
                        "loc": {
                            "start": {
                                "line": 3,
                                "column": 4
                            },
                            "end": {
                                "line": 3,
                                "column": 17
                            }
                        }
                    },
                    "init": {
                        "type": "CallExpression",
                        "callee": {
                            "type": "MemberExpression",
                            "computed": false,
                            "object": {
                                "type": "Identifier",
                                "name": "DOMPurify",
                                "loc": {
                                    "start": {
                                        "line": 3,
                                        "column": 20
                                    },
                                    "end": {
                                        "line": 3,
                                        "column": 29
                                    }
                                }
                            },
                            "property": {
                                "type": "Identifier",
                                "name": "sanitize",
                                "loc": {
                                    "start": {
                                        "line": 3,
                                        "column": 30
                                        "line": 3,
                                        "column": 38
                                    }
                                }
                            },
                            "loc": {
                                "start": {
                                    "line": 3,
                                    "column": 20
                                },
                                "end": {
                                    "line": 3,
                                    "column": 38
                                }
                            }
                        },
                        "arguments": [
                            {
                                "type": "Identifier",
                                "name": "name",
                                "loc": {
                                    "start": {
                                        "line": 3,
                                        "column": 39
                                    },
                                    "end": {
                                        "line": 3,
                                        "column": 43
                                    }
                                }
                            }
                        ],
                        "loc": {
                            "start": {
                                "line": 3,
                                "column": 20
                            },
                            "end": {
                                "line": 3,
                                "column": 44
                            }
                        }
                    },
                    "loc": {
                        "start": {
                            "line": 3,
                            "column": 4
                        },
                        "end": {
                            "line": 3,
                            "column": 44
                        }
                    }
                }
            ],
            "kind": "var",
            "loc": {
                "start": {
                    "line": 3,
                    "column": 0
                },
                "end": {
                    "line": 3,
                    "column": 45
                }
```

```
        {
            "type": "ExpressionStatement",
            "expression": {
                "type": "CallExpression",
                "callee": {
                    "type": "MemberExpression",
                    "computed": false,
                    "object": {
                        "type": "Identifier",
                        "name": "document",
                        "loc": {
                            "start": {
                                "line": 4,
                                "column": 0
                            },
                            "end": {
                                "line": 4,
                                "column": 8
                            }
                        }
                    },
                    "property": {
                        "type": "Identifier",
                        "name": "write",
                        "loc": {
                            "start": {
                                "line": 4,
                                "column": 9
                            },
                            "end": {
                                "line": 4,
                                "column": 14
                            }
                        }
                    },
                    "loc": {
                        "start": {
                            "line": 4,
                            "column": 0
                        },
                        "end": {
                            "line": 4,
                            "column": 14
                        }
                    }
                },
                "arguments": [
                    {
                        "type": "Identifier",
                        "name": "sanitizedName",
                        "loc": {
                            "start": {
                                "line": 4,
                                "column": 15
                            },
                            "end": {
                                "line": 4,
                                "column": 28
                            }
                        }
                    }
                ],
                "loc": {
                    "start": {
                        "line": 4,
                    "end": {
                        "line": 4,
                        "column": 29
                    }
                }
            },
            "loc": {
                "start": {
                    "line": 4,
                    "column": 0
                },
                "end": {
                    "line": 4,
                    "column": 30
                }
            }
        }
    ],
    "loc": {
        "start": {
            "line": 1,
            "column": 0
        },
        "end": {
            "line": 4,
            "column": 30
        }
    }
}
```

Output

The output of the program is a `JSON` list of vulnerability objects that should be written to a file `./output/<slice>.output.json` where `<slice>.js` is the program slice under analysis. The structure of the objects should include 6 pairs, with the following meaning:

- `vulnerability`: name of vulnerability (string, according to the inputted pattern)
- `source`: input source (string, according to the inputted pattern, and line where it appears in the code)
- `sink`: sensitive sink (string, according to the inputted pattern, and line where it appears in the code)
- `implicit_flows`: whether there are implicit flows (string)
- `unsanitized_flows`: whether there are unsanitized flows (string)
- `sanitized_flows`: list of lists of the sanitizing functions (string, according to the inputted pattern, and line where it appears in the code) if present, otherwise empty (list of lists of pairs)

As an example, the output with respect to the program and patterns that appear in the examples in Specification of the Tool would be:

```
[
    {
        "vulnerability": "DOM XSS",
        "source": ["document.URL", 1],
        "sink": ["document.write", 4],
        "implicit_flows": "no",
        "unsanitized_flows": "no",
        "sanitized_flows": [[["DOMPurify.sanitize", 3]]]
    },
    {
        "vulnerability": "DOM XSS",
        "source": ["document.URL", 2],
        "sink": ["document.write", 4],
        "implicit_flows": "no",
        "unsanitized_flows": "no",
        "sanitized_flows": [[["DOMPurify.sanitize", 3]]]
    }
]
```

The output list must include a vulnerability object for every pair source-sink between which there is at least one flow of information:

source-sink, that could be sanitized in different ways, sanitized flows are represented as a list. Since each flow might be sanitized by more than one sanitizer, each flow is itself a list (with no particular order).

More precisely, the format of the output should be:

```
<OUTPUT> ::= [ <VULNERABILITIES> ]
<VULNERABILITIES> := "none" | <VULNERABILITY> | <VULNERABILITY>, <VULNERABILITIES>
<VULNERABILITY> ::= { "vulnerability": "<STRING>",
                    "source": [ "<STRING>", <INT> ]
                    "sink": [ "<STRING>", <INT> ],
                    "implicit_flows": <YESNO>,
                    "unsanitized_flows": <YESNO>,
                    "sanitized_flows": [ <FLOWS> ] }
<YESNO> ::= "yes" | "no"
<FLOWS> ::= "none" | <FLOW> | <FLOW>, <FLOWS>
<FLOW> ::= [ <SANITIZERS> ]
<SANITIZERS> ::= [ <STRING>, <INT> ] | [ <STRING>, <INT> ], <SANITIZERS>
```

Note: A flow is said to be sanitized if it goes "through" an appropriate sanitizer, i.e., if at some point the entire information is converted into the output of a sanitizer.

Precision and scope

The security property that underlies this project is the following:

Given a set of vulnerability patterns of the form (vulnerability name, a set of entry points, a set of sensitive sinks, a set of sanitizing functions), a program is secure if it does not encode, for any given vulnerability pattern, an information flow from an entry point to a sensitive sink, unless the information goes through a sanitizing function.

You will have to make decisions regarding whether your tool will signal, or not, illegal taint flows that are encoded by certain combinations of program constructs. You can opt for an approach that simplifies the analysis. This simplification may introduce or omit features that could influence the outcome, thus leading to wrong results.

Note that the following criteria will be valued:

- *Soundness* - successful detection of illegal taint flows (i.e., true positives). In particular, treatment of implicit taint flows will be valued.
- *Precision* - avoiding signalling programs that do not encode illegal taint flows (i.e., false-positives). In particular, sensitivity to the order of execution will be valued.
- Scope - treatment of a larger subset of the language. Using the same terms as in the Esprima Syntax Tree Format the mandatory constructs are those associated with nodes of type
  - Expression
    - Literal
    - Identifier
    - UnaryExpression
    - BinaryExpression
    - CallExpression
    - MemberExpression (.)
    - AssignmentExpression
  - Statement
    - BlockStatement
    - ExpressionStatement
    - IfStatement
    - WhileStatement

When designing and implementing this component, you are expected to take into account and to incorporate precision and efficiency considerations, as presented in the critical analysis criteria below.

4. Practical Test

Critical Analysis

The test will contain questions that evaluate your ability to critically analyse the tool that you have submitted, from the point of view of its precision and scope.

You will be asked to consider the security property expressed in Precision and scope, and the security mechanism that is studied in this project, which comprises:

- A component (assume already available) that statically extracts the program slices that could encode potential vulnerabilities in a program.

Given the intrinsic limitations of the static analysis problem, the tool you developed in the experimental part is necessarily imprecise in determining which programs encode vulnerabilities or not. It can be unsound (produce false negatives), incomplete (produce false positives), or both. You should be able to:

1. Explain and give examples of what are the imprecisions that are built into the proposed mechanism. Have in mind that they can originate at different levels:
   - imprecise tracking of information flows
     - Are all illegal information flows captured by the adopted technique? (false negatives)
     - Are there flows that are unduly reported? (false positives)
   - imprecise endorsement of input sanitization
     - Are there sanitization functions that could be ill-used and do not properly sanitize the input? (false negatives)
     - Are all possible sanitization procedures detected by the tool? (false positives)
2. *For each* of the identified imprecisions that lead to:
   - undetected vulnerabilities (false negatives)
     - Can these vulnerabilities be exploited?
     - If yes, how (give concrete examples)?
   - reporting non-vulnerabilities (false positives)
     - Can you think of how they could be avoided?

Mastering your code

Additionally, you should be able to extend or adapt your tool in order to tackle information flows encoded for different language constructs, or render information about the illegal flows in a different manner.

5. Grading

The baseline grade for the group will be determined based on the experimental part and test, according to the rules below.

Experimental part

The tool must adhere to the input/output formats specified in this document. Failure to do so may jeopardize/add delays to grading the project.

Grading of the Tool and Patterns will reflect the level of complexity of the developed tool, according to the following:

- Basic vulnerability detection (50%) - signals potential vulnerability based solely on explicit flows in slices with mandatory constructs
- Advanced vulnerability detection (25%) - signals potential vulnerability that can include implicit flows in slices with mandatory constructs, including the correct identification of implicit flows
- Sanitization recognition (20%) - signals potential sanitization of vulnerabilities
- Definition of a minimum of 5 appropriate Javascript Vulnerability Patterns (5%) - consider the provided and other related work. To be submitted in the same repo as the Group's project under name `5_patterns.json`.
- Bonus (5%) - treats other program constructs beyond the mandatory ones, extra effort for avoiding false positives

This part corresponds to 2/3 of the project grade.

Test and Discussion

The test is to be performed individually and in person. It corresponds to 1/3 of the project grade.

Besides the project, students can be selected for a discussion about the project, as decided by the course instructors. For students who are called, the discussion is mandatory in order to be graded for the project. During the discussion, each student is expected to be able to demonstrate full knowledge of all details of the project. Each individual grade might be adjusted according to the student's performance during the discussion.

6. Other Materials

Folder slices/ contains examples of analysis. For each slice `slice.js` we provide the expected output `slice.output.json` according to vulnerability patterns `slice.patterns.json`.

- Slices