TÉCNICO LISBOA (http://tecnico.ulisboa.pt)

Segurança em Software (https://fenix.tecnico.ulisboa.pt/disciplinas/SSof/2021-2022/1-semestre)

Discovering vulnerabilities in Python web applications

1. Aims

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 should have the following two parts:

1. An experimental component, consisting in the development and evaluation of a tool in your language of choice, according to the Specification of the Tool.
2. An analysis component, where the strengths and limitations of the tool are critically discussed and presented in a Report.

Submissions

Important dates and instructions:

- Groups (2 or 3 students) should register via Fenix by ~~17 December~~ **23 December**.
- The submission deadline for the code is ~~21 January 23:59~~ **28 January 23:59**. Please submit ~~a zip file containing~~ your code via your group's private repository at Git, under the appropriate Group number (https://git.rnl.tecnico.ulisboa.pt/SSof2122/Project-GroupXX). The submissions should include all the necessary code and a Readme.txt that specifies how the tool should be used. All tests that you would like to be considered for the evaluation of your tool should be made available in a common repository (https://git.rnl.tecnico.ulisboa.pt/SSof2122/Project-Tests), according to a forthcoming announcement.
- The submission deadline for the report is ~~28 January 23:59~~ **4 February 23:59**. The report should be submitted as a pdf.
- Demonstration and a discussion regarding the tool and report will take place ~~between 31 January and 2 February~~ **between 7 and 9 February**.

Authorship

Projects are to be solved in groups of 2 or 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. Presence at the discussion and tool demonstration is mandatory for all group members.

All sources should be adequately cited. Plagiarism (https://en.wikipedia.org/wiki/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 (so called sensitive sinks). This means that users are given the power to alter the behavior of sensitive functions, 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.

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 side programs encoded in the Python language. There exist a range of Web frameworks (https://wiki.python.org/moin/WebFrameworks) for Python, of which Django is the most widely used. While examples in this project specification often refer to Django views (https://docs.djangoproject.com/en/2.2/topics/http/views/), the problem is to be understood as generic to the Python language.

The following references are mandatory reading about the problem:

- S. Micheelsen and B. Thalmann, "PyT - A Static Analysis Tool for Detecting Security Vulnerabilities in Python Web Applications", Master's Thesis, Aalborg University 2016 (https://projekter.aau.dk/projekter/files/239563289/final.pdf)
- V. Chibotaru et. al, "Scalable Taint Specification Inference with Big Code", PLDI 2019 (https://files.sri.inf.ethz.ch/website/papers/scalable-taint-specification-inference-pldi2019.pdf) Note: This paper contains a large component of machine learning that is not within the scope of this course, and which you may skip through.
- L. Giannopoulos et. al, "Pythia: Identifying Dangerous Data-flows in Django-based Applications", EuroSec 2019 (https://dimitro.gr/assets/papers/GDTM19.pdf)

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 Python instructions that are considered to be relevant to our analysis.

The following code slice, which is written in Python, contains code lines which may impact a data flow between a certain entry point and a sensitive sink. The variable request (which for intuition can be seen as the request parameter of a Django view), is uninstantiated, and can be understood as an entry point. It uses the `MySQLCursor.execute()` method, which executes the given database operation query.

```
uname = retrieve_uname(request)
q = cursor.execute("SELECT pass FROM users WHERE user='%s'" % uname)
```

Inspecting this slice it is clear that the program from which the slice was extracted can potentially encode an SQL injection vulnerability. An attacker can inject a malicious username like `' OR 1 = 1 --`, modifying the structure of the query and obtaining all users' passwords.

The aim of the tool is to search in 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:

- name of vulnerability (e.g., SQL injection)
- a set of entry points (e.g., request parameter),
- a set of sanitization functions (e.g., escape_string),
- a set of sensitive sinks (e.g., execute),
- and a flag indicating whether implicit flows are to be considered.

The tool should signal potential vulnerabilities and sanitization efforts: If it identifies a possible data flow from an entry point to a sensitive sink (according to the inputted patterns), it should report a potential vulnerability; if the data flow passes through a sanitization 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 in testing the tool. It is however your responsibility to perform more extensive testing for ensuring the correctness and robustness of the tool. While the examples that we provide are mostly within the Django framework, your tool should be generic in order to function for other slices and vulnerabilities that can be expressed with patterns of the format above. You can find data for forming vulnerability patterns in the appendix of Chibotaru et. al (https://files.sri.inf.ethz.ch/website/papers/scalable-taint-specification-inference-pldi2019.pdf). (Note however that for the purpose of testing, the names of vulnerabilities, sources, sanitizers and sinks are irrelevant. 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.)

Running the tool

The tool should be called in the command line. All input and output is to be encoded in JSON (http://www.json.org/), according to the specifications that follow.

Your program should take two arguments, which are the only input that it should consider:

- the name of the JSON file containing the program slice to analyse, represented in the form of an Abstract Syntax Tree;
- the name of the JSON file containing the list of vulnerability patterns to consider.

You can assume that the parsing of the Python slices has been done, and that the input files are well-formed. The analysis should be fully customizable to the inputted vulnerability patterns. In addition to the entry points specified in the patterns, **by default any uninstantiated variable that appears in the slice is to 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 instruction(s) (if any) have been applied. The format of the output is specified below.

The way to call your tool depends on the language in which you choose to implement it, but **it should be called in the command line with two arguments `<program>.json` `<patterns>.json` and produce the output referred below and no other to a file `<program>.output.json`**.

```
$ ./bo-analyser program.json patterns.json
```

```
$ python ./bo-analyser.py program.json patterns.json
$ java bo-analyser program.json patterns.json
```

You are free to choose the programming language for the implementation, as long as your tool is runnable in the labs or VM (in case of doubt please ask).

Input

Program slices

Your program should read from a text file (given as first argument in the command line) the representation of a Python slice in the form of an Abstract Syntax Tree (AST). The AST is represented in JSON, using the same structure as in Python's AST module (https://docs.python.org/3.10/library/ast.html). To explore the structure of the AST that corresponds to a given program you can use this Python AST explorer (https://python-ast-explorer.com/). This tutorial (https://greentreesnakes.readthedocs.io/en/latest/) is a helpful resource.

(OBS: The structure of Python's ASTs varies slightly with different Python versions. The project should accept Python 3.9 ASTs -- as in th labs, similar to 3.8 and 3.10. The examples below have been corrected in order to illustrate the right version.)

For instance, the program

```
print("Hello World!")
```

is represented as

```
{
    "ast_type": "Module",
    "body": [
        {
            "ast_type": "Expr",
            "col_offset": 0,
            "end_col_offset": 21,
            "end_lineno": 1,
            "lineno": 1,
            "value": {
                "args": [
                    {
                        "ast_type": "Constant",
                        "col_offset": 6,
                        "end_col_offset": 20,
                        "end_lineno": 1,
                        "kind": null,
                        "lineno": 1,
                        "value": "Hello World!"
                    }
                ],
                "ast_type": "Call",
                "col_offset": 0,
                "end_col_offset": 21,
                "end_lineno": 1,
                "func": {
                    "ast_type": "Name",
                    "col_offset": 0,
                    "ctx": {
                        "ast_type": "Load"
                    },
                    "end_col_offset": 5,
                    "end_lineno": 1,
                    "id": "print",
                    "lineno": 1
                },
                "keywords": [],
                "lineno": 1
            }
        }
    ],
    "type_ignores": []
}
```

and the slice

```
uname = retrieve_uname(request)
q = cursor.execute("SELECT pass FROM users WHERE user='%s'" % uname)
```

is represented as:

```
{
    "ast_type": "Module",
    "body": [
        {
            "ast_type": "Assign",
            "col_offset": 0,
            "end_col_offset": 31,
            "end_lineno": 1,
            "lineno": 1,
            "targets": [
                {
                    "ast_type": "Name",
                    "col_offset": 0,
                    "ctx": {
                        "ast_type": "Store"
                    },
                    "end_col_offset": 5,
                    "end_lineno": 1,
                    "id": "uname",
                    "lineno": 1
                }
            ],
            "type_comment": null,
            "value": {
                "args": [
                    {
                        "ast_type": "Name",
                        "col_offset": 23,
                        "ctx": {
                            "ast_type": "Load"
                        },
                        "end_col_offset": 30,
                        "end_lineno": 1,
                        "id": "request",
                        "lineno": 1
                    }
                ],
                "ast_type": "Call",
                "col_offset": 8,
                "end_col_offset": 31,
                "end_lineno": 1,
                "func": {
                    "ast_type": "Name",
                    "col_offset": 8,
                    "ctx": {
                        "ast_type": "Load"
                    },
                    "end_col_offset": 22,
                    "end_lineno": 1,
                    "id": "retrieve_uname",
                    "lineno": 1
                },
                "keywords": [],
                "lineno": 1
            }
        },
        {
            "ast_type": "Assign",
            "col_offset": 0,
            "end_col_offset": 68,
            "end_lineno": 2,
            "lineno": 2,
            "targets": [
                {
                    "ast_type": "Name",
                    "col_offset": 0,
                    "ctx": {
                        "ast_type": "Store"
                    },
                    "end_col_offset": 1,
                    "end_lineno": 2,
                    "id": "q",
                    "lineno": 2
                }
            ],
            "type_comment": null,
            "value": {
                "args": [
                    {
                        "ast_type": "BinOp",
                        "col_offset": 19,
                        "end_col_offset": 67,
                        "end_lineno": 2,
                        "left": {
                            "ast_type": "Constant",
                            "col_offset": 19,
                            "end_col_offset": 59,
                            "end_lineno": 2,
                            "kind": null,
                            "lineno": 2,
                            "value": "SELECT pass FROM users WHERE user='%s'"
                        },
                        "lineno": 2,
                        "op": {
                            "ast_type": "Mod"
                        },
                        "right": {
                            "ast_type": "Name",
                            "col_offset": 62,
                            "ctx": {
                                "ast_type": "Load"
                            },
                            "end_col_offset": 67,
                            "end_lineno": 2,
                            "id": "uname",
                            "lineno": 2
                        }
                    }
                ],
                "ast_type": "Call",
                "col_offset": 4,
                "end_col_offset": 68,
                "end_lineno": 2,
                "func": {
                    "ast_type": "Attribute",
                    "attr": "execute",
                    "col_offset": 4,
                    "ctx": {
                        "ast_type": "Load"
                    },
                    "end_col_offset": 18,
                    "end_lineno": 2,
                    "lineno": 2,
                    "value": {
                        "ast_type": "Name",
                        "col_offset": 4,
                        "ctx": {
                            "ast_type": "Load"
                        },
                        "end_col_offset": 10,
                        "end_lineno": 2,
                        "id": "cursor",
                        "lineno": 2
                    }
                },
                "keywords": [],
                "lineno": 2
            }
        }
    ],
    "type_ignores": []
}
```

In order to parse the ASTs, you can use an off-the-shelf parser for JSON. Note that not all of the information that is available in the AST needs necessarily to be used and stored by your program.

Besides the slices that are made available, you can produce your own ASTs for testing your program by using a python-to-json parser (https://pypi.org/project/astexport/). You can visualize the JSON outputs as a tree using this online tool (http://jsonviewer.stack.hu/). The parser can also be used for building a stand-alone tool -- though this is **not** valued within the context of this project.

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.

An example JSON file with three patterns:

```
[
  {"vulnerability": "SQL injection A",
  "sources": ["get", "get_object_or_404", "QueryDict", "ContactMailForm", "ChatMessa
geForm"],
  "sanitizers": ["mogrify", "escape_string"],
  "sinks": ["execute"],
  "implicit": "no"},

  {"vulnerability":"SQL injection B",
  "sources":["QueryDict", "ContactMailForm", "ChatMessageForm", "copy", "get_query_s
tring"],
  "sanitizers":["mogrify", "escape_string"],
  "sinks":["raw", "RawSQL"],
  "implicit":"yes"},

  {"vulnerability":"XSS",
  "sources":["get", "get_object_or_404", "QueryDict", "ContactMailForm", "ChatMessag
eForm"],
  "sanitizers":["clean","escape","flatatt","render_template","render","render_to_res
ponse"],
  "sinks":["send_mail_jinja","mark_safe","Response","Markup","send_mail_jinja","Ra
w"],
  "implicit":"no"}
]
```

Output

The output of the program is a `JSON` list of vulnerability objects that should be written to a file `analysethis.output.json` when the program under analysis is `analysethis.json` . The structure of the objects should include 5 pairs, with the following meaning:

- "name": name of vulnerability (string, according to inputted pattern)
- "source": input source (string, according to inputted pattern)
- "sink": sensitive sink (string, according to inputted pattern)
- "unsanitized flows": whether there are unsanitized flows (string)
- "sanitized flows": sanitizing functions if present, otherwise empty (list of lists of strings)

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

```
[{"vulnerability":"SQL injection A",
"source":"request",
"sink":"execute",
"unsanitized flows":"yes"},
"sanitized flows":[]]
```

The output list must include a vulnerability object for every pair source-sink between which there is at least one flow of information. If at least one of the flows is not sanitized, it must be signaled. Since it is possible that there are more than one flow paths for a given pair 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> ::= { "name":"<STRING>",
                      "source":"<STRING>",
                      "sink":"<STRING>",
                      "unsanitized flows": <YESNO>,
                      "sanitized flows": [ <FLOWS> ] }
<YESNO> ::= "yes" | "no"
<FLOWS> ::= "none" | <FLOW> | <FLOW>,<FLOWS>
<FLOW> ::= [ <SANITIZERS> ]
<SANITIZERS> ::= <STRING> | <STRING>,<SANITIZERS>
```

*NEW** **(OBS: A flow is said to be sanitized if it goes "through" the 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, i.e. successful detection of illegal taint flows. In particular, treatment of implicit taint flows will be valued.
- Precision, i.e. avoiding signalling programs that do not encode illegal taint flows. In particular, sensitivity to the order of execution will be valued.
- Scope, i.e. treatment of a larger subset of the language. The mandatory language constructs are those that appear in the slices provided, and include: assignments, binary operations, function calls, condition test and while loop.

Obs: Using the terms in the definition of the Python's AST module (https://greentreesnakes.readthedocs.io/en/latest/nodes.html#meet-the-nodes), the mandatory constructs are Literals (Constant), Variables (Name), Expressions (Expr, BinOp, Compare, Call, Attribute), Statements (Assign), Control flow (If and While).

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

4. Report

Critical Analysis

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.
- A tool (developed by you), that receives a configuration file containing vulnerability patterns, and signals potential vulnerabilities in given slices according to those patterns, as well as possible

sanitization efforts, as implemented by you in the experimental part.

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

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 not properly sanitized the input? (false negatives) Are all possible sanitization procedures detected by the tool? (false positives) **Make sure that you give at least one example to the 4 questions above.**
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? **Make sure that you answer the questions at least for each of the 4 examples given in point 4.**
3. Propose one way of making the tool more precise, and predict what would be the trade-offs (efficiency, precision) involved in this change. You can use the papers suggested above for ideas of other techniques that could be used.

Structure

Report your work and your conclusions in a written document, using no more than 4 pages, single column, (excluding references and appendices), according to the following guidelines:

- Use a structure that helps to read and find the relevant information.
- Assume a reader acquainted with the context of the Project (in other words go precisely and straight to the point!).
- Briefly describe the experimental part, presenting the architecture of the tool and the main design options (maximum 2 pages).
- Define and discuss what your tool is able to achieve, making sure that you answer all the above questions.

We suggest the following structure for the report:

1. Introduce the context of your work (1 paragraph) -- What is the problem that you want to solve? Why is it important?
2. Explain the approach (1 page, possible diagram) -- What is the underlying technique behind the tool?
3. Demonstrate the behavior of the tool (1 paragraph) -- How do you use it? What does it do?
4. Present the evaluation of the tool (2 paragraphs and link to performed tests) -- How did you test the correctness and robustness of the tool?
5. Critically analyse and discuss your solution (1-2 pages) -- What are the strengths and limitations of the tool?
6. Refer related work (2-3 paragraphs) - What other tools address the same problem for the considered language? What other tools use the same technique for solving similar problems? **Make sure that you refer at least to the 3 papers given above, and try to include 2 other that you find via these papers' references or via eg. Google Scholar.**
7. Conclusion (1 paragraph) -- What have you achieved? What is the tool good for? What could you improve in the tool?

5. Grading

Discussion

The baseline grade for the group will be determined based on the experimental part and report, acording to the rules below. During the discussion, all students in the groups are expected to be able to demonstrate knowledge of all details of these two components. In order to be graded for the project, each student must participate in the discussion, and his/her grade might be adjusted accordingly.

Experimental part

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
- Sanitization recognition (20%) - signals potential sanitization of vulnerabilities
- Distinguish patterns vulnerability data (5%) - respects which sources correspond to which sinks for different vulnerabilities, and which function may sanitize which vulnerability
- Bonus (5%) - treats other program constructs beyond the mandatory ones, extra effort for avoiding false positives

This part corresponds to 20% of the final grade (2/3 of what is referred to as "Project" in Métodos de Avaliação (https://fenix.tecnico.ulisboa.pt/disciplinas/SSof/2021-2022/1-semestre/metodos-de-avaliacao)).

Report

The maximum grade of the report does not depend on the complexity of the tool. It will of course reflect whether the analysis of the imprecisions matches the precision of the tool that was developed (which in turn depends on the complexity of the tool). The components of the grading are worth 20% each, and are the following:

- Quality of writing - structure of the report, clarity of the ideas
- Related work - depth of understanding of the related work, detachment from words used in cited papers. See mandatory references in Problem above.
- Identification of imprecisions - connection with experimental work. See questions 1.a) and 1.b) in Section Requirements for the discussion - Report above.
- Understanding of imprecisions - connection with experimental work. See questions 2.a) and 2.b) in Section Requirements for the discussion - Report above.
- Improving precision - originality, own ideas. See question 3 in Section Requirements for the discussion - Report above.
- Bonus (5%) - cites other references beyond the mandatory ones. See references in Problem above.

This part corresponds to 10% of the final grade (1/3 of what is referred to as "Project" in Métodos de Avaliação (https://fenix.tecnico.ulisboa.pt/disciplinas/SSof/2021-2022/1-semestre/metodos-de-avaliacao)).

Attachments

- slices-14Jan.zip (https://fenix.tecnico.ulisboa.pt/downloadFile/845043405565957/slices-14Jan.zip)

Página Inicial (https://fenix.tecnico.ulisboa.pt/disciplinas/SSof/2021-2022/1-semestre/pagina-inicial)

(https://fenix.tecnico.ulisboa.pt/disciplinas/SSof/2021-2022/1-semestre/rss/announcement)

Grupos (https://fenix.tecnico.ulisboa.pt/disciplinas/SSof/2021-2022/1-semestre/grupos)

Avaliação (https://fenix.tecnico.ulisboa.pt/disciplinas/SSof/2021-2022/1-semestre/avaliacao)

Bibliografia (https://fenix.tecnico.ulisboa.pt/disciplinas/SSof/2021-2022/1-semestre/bibliografia)

Horário (https://fenix.tecnico.ulisboa.pt/disciplinas/SSof/2021-2022/1-semestre/horario)

Métodos de Avaliação (https://fenix.tecnico.ulisboa.pt/disciplinas/SSof/2021-2022/1-semestre/metodos-de-avaliacao)

Objectivos (https://fenix.tecnico.ulisboa.pt/disciplinas/SSof/2021-2022/1-semestre/objectivos)

Planeamento (https://fenix.tecnico.ulisboa.pt/disciplinas/SSof/2021-2022/1-semestre/planeamento)

Programa (https://fenix.tecnico.ulisboa.pt/disciplinas/SSof/2021-2022/1-semestre/programa)

Turnos (https://fenix.tecnico.ulisboa.pt/disciplinas/SSof/2021-2022/1-semestre/turnos)

Anúncios (https://fenix.tecnico.ulisboa.pt/disciplinas/SSof/2021-2022/1-semestre/anuncios) (https://fenix.tecnico.ulisboa.pt/disciplinas/SSof/2021-2022/1-semestre/rss/announcement)

Sumários (https://fenix.tecnico.ulisboa.pt/disciplinas/SSof/2021-2022/1-semestre/sumarios) (https://fenix.tecnico.ulisboa.pt/disciplinas/SSof/2021-2022/1-semestre/rss/summary)

Notas (https://fenix.tecnico.ulisboa.pt/disciplinas/SSof/2021-2022/1-semestre/notas)

Course Logistics (https://fenix.tecnico.ulisboa.pt/disciplinas/SSof/2021-2022/1-semestre/course-logistics)

Study materials (https://fenix.tecnico.ulisboa.pt/disciplinas/SSof/2021-2022/1-semestre/study-materials)

Office hours (https://fenix.tecnico.ulisboa.pt/disciplinas/SSof/2021-2022/1-semestre/office-hours)

Project (https://fenix.tecnico.ulisboa.pt/disciplinas/SSof/2021-2022/1-semestre/project-e8f)

VSSD Labs (https://fenix.tecnico.ulisboa.pt/disciplinas/SSof/2021-2022/1-semestre/vssd-labs-new)