



## Software security, secure programming

Tools for code security analysis

Master M2 Cybersecurity

Academic Year 2023 - 2024

# Motivation

**Most** software (are likely to) contain **security vulnerabilities** ...

There is a strong need for **tools** allowing to:

- ▶ detect potential vulnerabilities
- ▶ help to evaluate their exploitability/dangerousness

→ Useful for:

developers, users of 3rd-party libraries/applications, code auditors, etc.

Other possible applications :

- ▶ malware (behavioral) analysis
- ▶ reverse engineering
- ▶ code (de-)obfuscation
- ▶ exploit generation
- ▶ variant analysis
- ▶ etc.

# Several classes of tools

## Syntactic vs Semantic

- ▶ syntactic: check compliance w.r.t. to **coding rules/standarts**
- ▶ **semantic**: check for **behavioral** inconsistencies

## Static vs Dynamic

- ▶ static: check are performed at **“compile time”**  
(no concrete code execution)
- ▶ dynamic: on-line and/or offline checks require **code execution** steps

## Black vs Grey vs White Box

- ▶ black box: **no access** required to the target code
- ▶ white box: **full access** required to the (source ?) target code
- ▶ grey box: **partial access** required to the target code

etc.

## Taking into account the limits of computability . . .

. . . no hope to get a **fully automated** (powerful) tool:

*all non-trivial semantic properties of programs are undecidable*  
*[Rice theorem]*

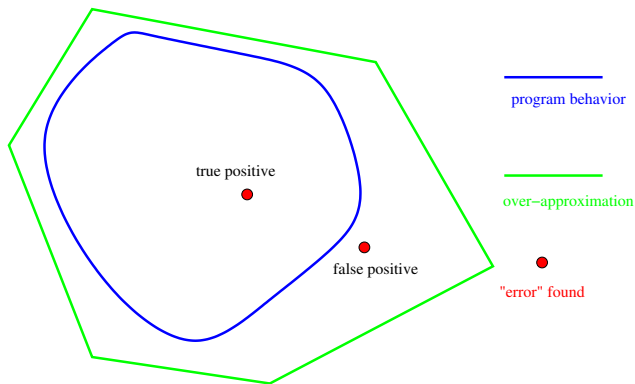
Possible **work-arounds**:

- ▶ **Approximate** enough the program behavior to make the analysis **decidable**  $\Rightarrow$  the results may be no longer **sound** no **complete**
- ▶ use a **semi-algorithm**  
**if the analysis terminates** then it gives **sound** and **complete** results . . .

**In practice:**

re-use & extend existing code analysis techniques used for compilation, test, verification

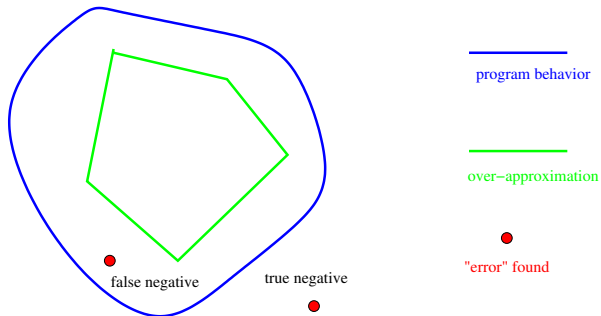
## Over-approximation of the program behavior



**Sound:** “correct” verdicts are always reliable ...  
(never miss an “incorrect” execution)

**Not Complete:** may reject correct programs ...  
( $\exists$  “false positives”)

## Under-approximation of the program behavior



**Unsound:** “correct” verdicts are not reliable ...  
(may miss “incorrect” executions)

**Complete:** never reject correct programs ...  
(“incorrect” execution reported are real ones)

## In the following . . .

An overview of:

- ▶ dynamic approaches:

  - ▶ **fuzzing**

  - ▶ **dynamic-symbolic execution**

- ▶ static approaches:

  - ▶ **value-set analysis**

  - ▶ **code-pattern** based vulnerability detection