

# Topics and ideas for master theses in cybersecurity

Emilio Coppa

coppa@diag.uniroma1.it

Sapienza University of Rome

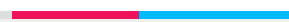


# Background: Fuzzing

Fuzzing [\[RIF\]](#) is a software testing technique that can identify vulnerabilities in real-world applications.

The ideas behind it is to take an input program, mutate it (change a few bytes), and run the program over the input. This process (pick input, mutate it, execute program) is repeated millions of time. The mutations applied over an input can be very simple, e.g., randomly flipping a bit, or more sophisticated, e.g., placing a specific value determined with a complex heuristics at specific byte. When running the program, the fuzzer checks for crashes (it has won the lottery! There is a bug) or “progress” in the analysis, e.g., the current input is making the program do new things during the execution, hence showings new behavior.

Fuzzing should not work (too simple on paper!), but it does! Google has discovered more than 40,000 bugs in OSS-Fuzz [\[RIF\]](#). **Fuzzing is “dumb” but quite “efficient” at being dumb :)**



# Fuzzing IoT Devices for vulnerability detection

- **Goal:** Grey-box Fuzzing is a software testing technique that can find bugs and vulnerabilities in real-world applications [RIF] (even when the source code is not available). This thesis will investigate how to effectively perform fuzzing of firmware images taken from IoT devices. Several IoT devices are likely running low level code that may contain several critical vulnerabilities, which we may want to discover.
- **Required skills:** Knowledge of assembly language and other low level aspects of a machine.
- **Thesis type:** This is a research project, a minimum of 6 months of work are expected.

# Background: Symbolic Execution

Symbolic execution [RIF] is a program analysis technique extensively used for different security purposes: identify vulnerabilities, exploit the vulnerabilities, perform automatic reverse engineering of malware.

For instance, given the program:

```
int foo(int inputA, int inputB) {  
    if (inputA * 37 == 119 * inputB) bug();  
    else not_interesting();  
}
```

Symbolic execution can automatically understand that finding an assignment for {**inputA**, **inputB**} such the **query** "**inputA** \* 37 == 119 \* **inputB**" is true makes the program execute the function **bug()**. This is important since in security we often want to compute the input conditions that make a program reach a specific portion of code. Given a **query**, symbolic execution uses an SMT solver to (possibly) obtain an assignment. Unfortunately, reasoning over a query can be very expensive for an SMT solver.

**Symbolic Execution is “smart” but quite “inefficient” at being smart :(**

# Symbolic Execution of floating-point computations

- **Background:** see “Background: Symbolic Execution”
- **Goal:** Most existing implementation of Symbolic Execution [RIF] builds queries only related to integer computations. This thesis will explore how to improve existing symbolic execution frameworks in order to support reasoning even on floating-point computations. Reasoning on floating-point computations may help find critical and unexpected vulnerabilities in several real-world applications.
- **Required skills:** Knowledge of assembly language and other low level aspects of a machine.
- **Thesis type:** This is a research project, a minimum of 6 months of work are expected.

# Query solving using Fuzzing Techniques

- **Background:** see “Background: Symbolic Execution”
- **Goal:** Symbolic Execution [RIF] builds queries that are usually solved exploiting an SMT solver. This thesis will instead explore how to use fuzzing techniques [RIF] to solve the queries generated by symbolic execution.
- **Required skills:** Knowledge of assembly language and other low level aspects of a machine.
- **Thesis type:** This is a research project, a minimum of 6 months of work are expected.

# Query complexity estimate with Deep Neural Networks

- **Background:** see “Background: Symbolic Execution”
- **Goal:** This thesis aims at developing a novel technique for estimating the complexity of solving a query (generated by symbolic execution) using an SMT solver. We aim to build a system that is able to use DNN to give an estimate of the time complexity needed to require a query (e.g., the times in seconds), and then integrate such system into a symbolic execution environment.
- **Required skills:** We are looking for students that have a familiarity with machine learning techniques and deep neural networks and low-level assembly code. For the part on machine learning, a knowledge of python is required.
- **Thesis type:** This is a research project, a minimum of 6 months of work are expected.
- **Main contact:** Emilio Coppa and Giuseppe Di Luna

# Combining Fuzzing and Symbolic Execution for Vulnerability Detection

- **Background:** see “Background: Symbolic Execution”
  - **Goal:** Symbolic Execution [RIF] and Fuzzing [RIF] are two powerful software testing techniques for finding bugs and vulnerabilities in real-world applications [RIF] (even when the source code is not available). This thesis will explore how to combine these two techniques to make them more effective, aiming at an improved cooperation.
  - **Required skills:** Knowledge of assembly language and other low level aspects of a machine.
  - **Thesis type:** This is a research project, a minimum of 6 months of work are expected.
-