جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

كلية العلوم

قسم الأمن السيبراني

Lecture: 10

Sanitizers

Subject: software security

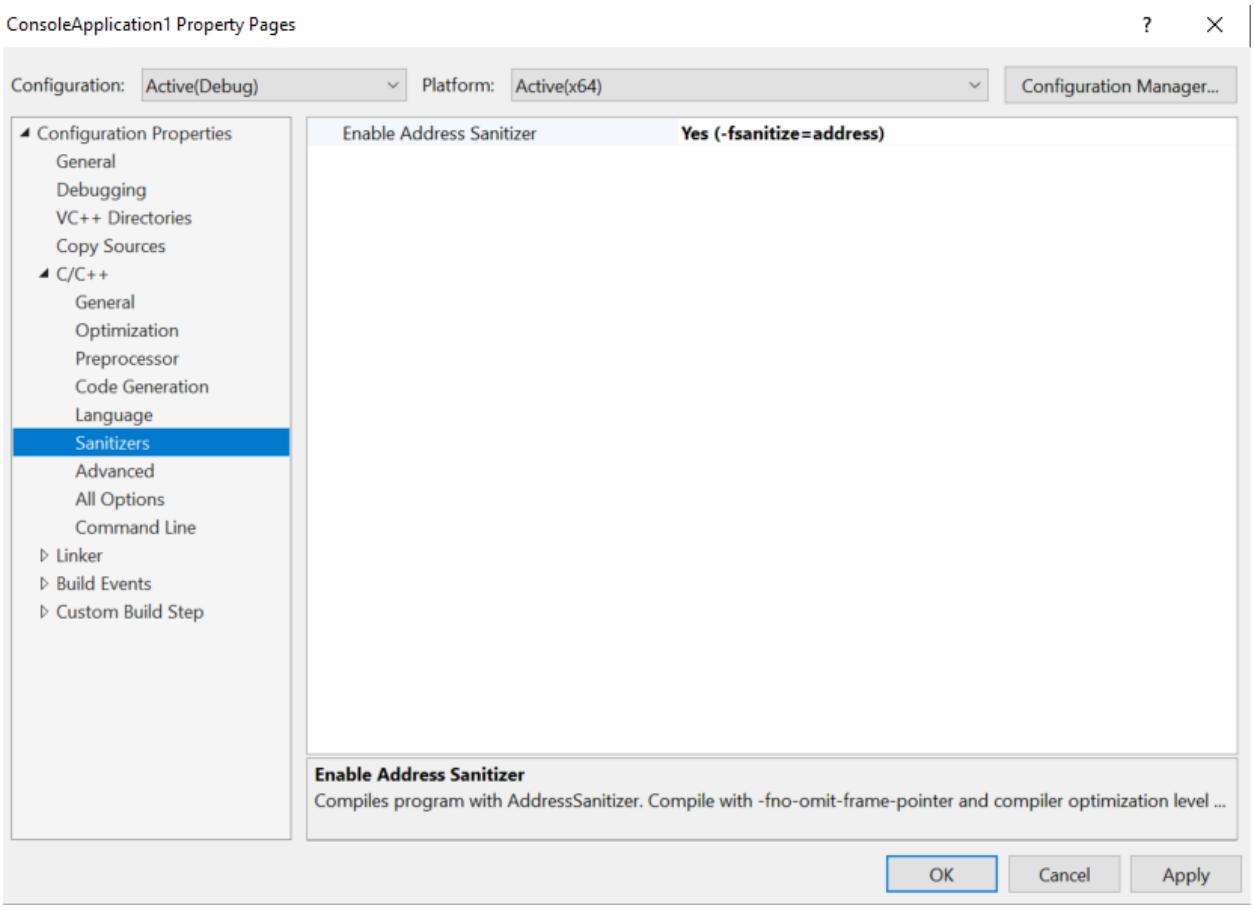
second Stage

Lecturer: Asst. Lecturer. Suha Alhussieny

Sanitizers

Sanitizers are tools designed to detect various types of bugs in programs, often related to memory safety and undefined behavior. They work by instrumenting the code to check for common issues during runtime. Some common types of sanitizers include:

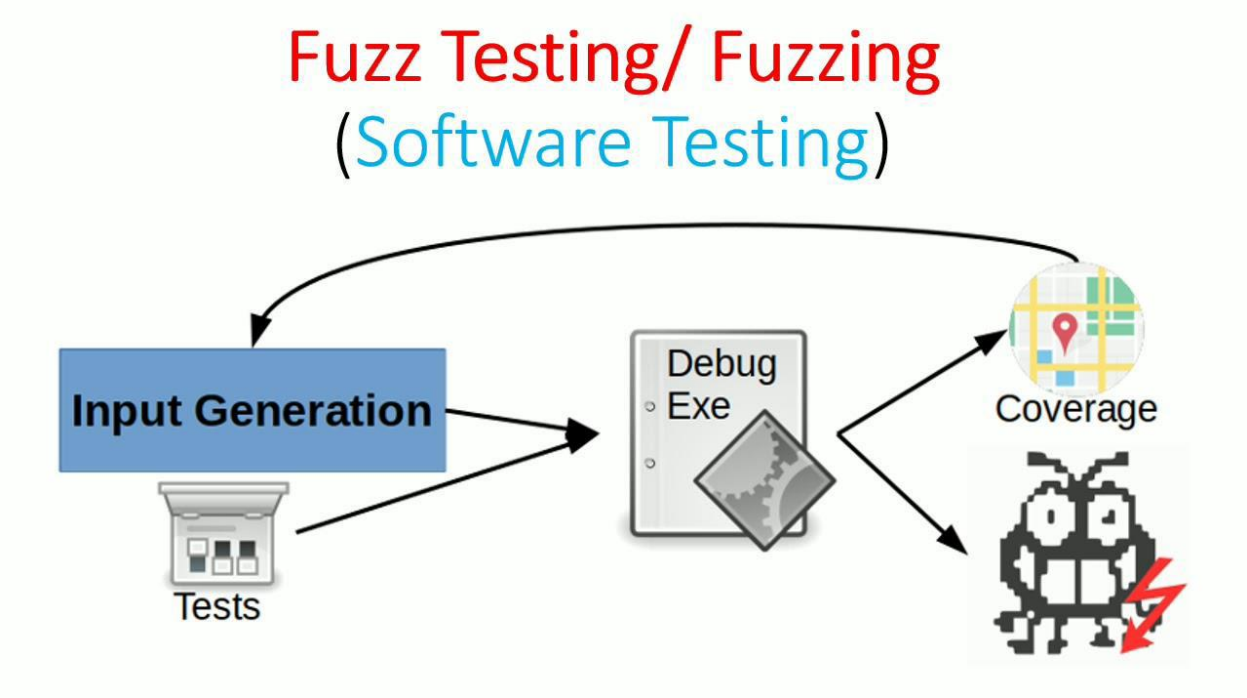
- **AddressSanitizer (ASan)**: Detects memory errors such as buffer overflows, use-after-free, and memory leaks.
- **MemorySanitizer (MSan)**: Identifies uninitialized memory reads.
- **ThreadSanitizer (TSan)**: Finds data races and threading issues.
- **UndefinedBehaviorSanitizer (UBSan)**: Catches undefined behaviors, such as integer overflows or invalid operations.



Fuzzing

Fuzzing is an automated testing technique that involves sending a large volume of random or semi-random inputs to a program to uncover vulnerabilities or crashes. It works by generating a wide range of inputs to test how the software handles unexpected or malformed data. Fuzzing can be categorized into different types:

- **Mutation-Based Fuzzing**: Modifies existing inputs or seeds to generate new test cases.
- **Generation-Based Fuzzing**: Creates inputs from scratch based on input formats or protocols.
- **Coverage-Guided Fuzzing**: Uses feedback from the program's execution to guide the generation of more effective test inputs.

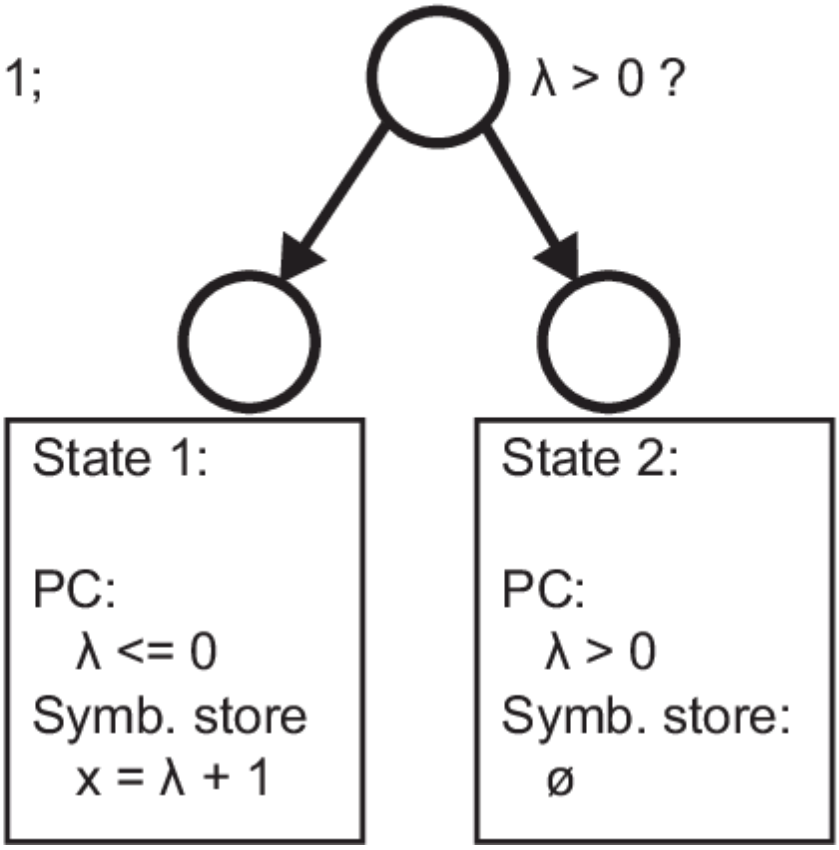


Symbolic Execution

Symbolic execution is a technique used to analyze a program by exploring different execution paths based on symbolic values rather than concrete inputs. It involves:

- **Path Exploration**: The symbolic execution engine explores possible execution paths of the program, which can lead to discovering hidden bugs and vulnerabilities.
- **Constraint Solving**: The system generates constraints based on the symbolic inputs and solves them to find possible values that would lead to different execution paths.
- **Error Detection**: Identifies potential issues by analyzing the constraints and the symbolic execution paths.

```
λ = makeSymbolic();
if (λ > 0)
    x = 14;
else
    x = λ + 1;
```



Q/

1. What is the primary purpose of sanitizers in software?

A. To compile code

B. To remove comments from code

C. To detect bugs during runtime

D. To manage databases

E. To encrypt code

2. Which sanitizer is used to detect memory errors like buffer overflows?

A. ThreadSanitizer

B. AddressSanitizer

C. MemorySanitizer

D. CodeSanitizer

E. UndefinedBehaviorSanitizer

3. What kind of issues does MemorySanitizer primarily detect?

A. Race conditions

B. Buffer overflows

C. Use-after-free errors

D. Memory leaks

E. Uninitialized memory reads

4. What kind of problems does ThreadSanitizer help find?

A. Memory leaks

B. Data races

C. Buffer overflows

D. Integer overflows

E. Input validation errors

5. Which sanitizer helps identify undefined behaviors like integer overflows?

A. AddressSanitizer

B. MemorySanitizer

C. ThreadSanitizer

D. UndefinedBehaviorSanitizer

E. ControlFlowSanitizer

6. What technique involves sending random or semi-random inputs to software to find vulnerabilities?

A. Sanitizing

B. Symbolic Execution

C. Fuzzing

D. Debugging

E. Compilation

7. What is mutation-based fuzzing?

A. It starts from scratch to build inputs

B. It changes protocols

C. It modifies existing inputs to generate test cases

D. It encrypts the code

E. It analyzes symbolic paths

8. What is the focus of generation-based fuzzing?

A. Modifying known inputs

B. Using code coverage tools

C. Generating inputs from scratch

D. Copying test cases

E. Monitoring runtime memory

9. How does coverage-guided fuzzing improve effectiveness?

A. By disabling sanitizers

B. By using symbolic execution

C. By using program feedback to guide input generation

D. By encrypting inputs

E. By skipping invalid inputs

10. What does symbolic execution use instead of real inputs?

A. Random strings

B. Manual test cases

C. Symbolic values

D. Code sanitizers

E. Machine learning

11. What is the purpose of path exploration in symbolic execution?

A. To delete unnecessary code

B. To find runtime errors

C. To explore all execution paths of a program

D. To compile efficiently

E. To find dead code

12. What does constraint solving in symbolic execution involve?

A. Minimizing code

B. Solving equations to guide execution paths

C. Encrypting variables

D. Detecting input types

E. Blocking invalid paths

13. Which testing method best uncovers hidden bugs via logical paths?

A. Manual testing

B. Symbolic execution

C. Code review

D. Debugging

E. Static analysis

14. Which of the following tools would best detect a use-after-free error?

A. MSan

B. UBSan

C. TSan

D. ASan

E. GDB

15. What category of fuzzing depends heavily on knowledge of input format or protocol?

A. Mutation-based

B. Black-box

C. White-box

D. Generation-based

E. Constraint-guided