Dynamic Analysis: Fuzzing

An automated and scalable approach to test software at runtime

- Bombard a program with random, corrupted, or unexpected data to identify how it behaves under unexpected conditions.
- Observe the program for crashes, memory issues or unexpected behaviors.
- Examine failures to determine if they represent exploitable vulnerabilities.

A lot of software testing tools based on fuzzing

- AFL: https://github.com/google/AFL
- FOT: https://sites.google.com/view/fot-the-fuzzer
- Peach: https://wiki.mozilla.org/Security/Fuzzing/Peach

Limitations

- Limited code coverage.
- Require expert analysis to assess whether system crashes are exploitable
- May miss logic flaws that do not result in crashes.

Different Types of Fuzzing Techniques

Mutation-based:

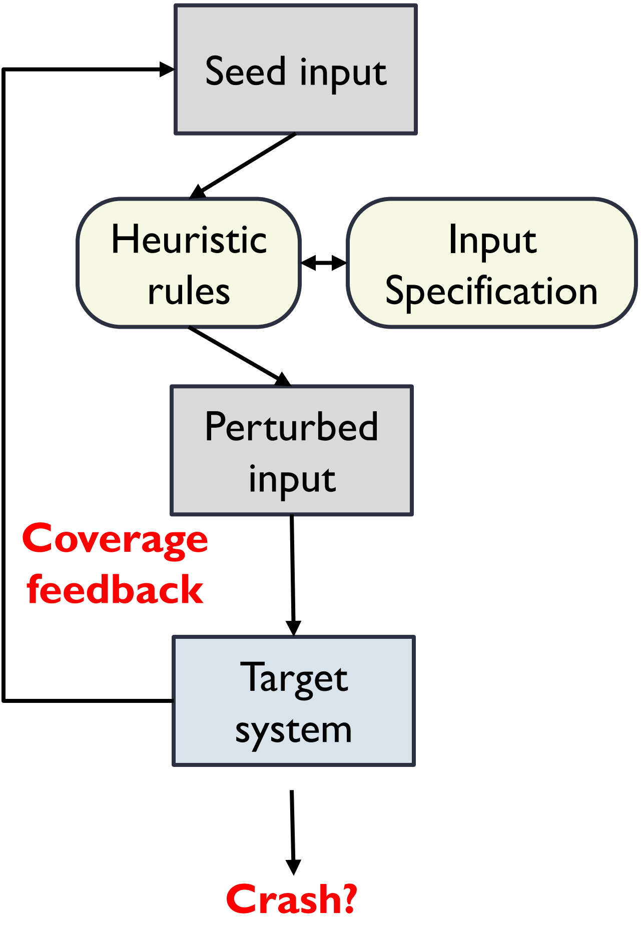
- Collect a corpus of inputs that explores as many states as possible
- Perturb inputs randomly, possibly guided by heuristics, e.g., bit flips, integer increments, substitute with small, large or negative integers.
- Simple to set up. Can be used for off-the-shelf software.

Generation-based

- Convert a specification of input format into a generative procedure
- Generate test cases according to procedure with perturbations
- Get higher coverage by leveraging knowledge of the input format
- Requires lots of effort to set up, and is domain-specific;

Coverage-guided

- Using traditional fuzzing strategies to create new test cases.
- Test the program and measure the code coverage.
- Using code coverage as a feedback to craft input for uncovered code
- Good at finding new states, and combine well with other solutions;



Outline

- Safe Programing
- Vulnerability Detection
- **Compiler and System Support**

Recall: Steps of Stack Smashing Attack

1. Find a buffer overflow vulnerability in the program
2. Inject shellcode into a known memory address
3. Exploit the buffer overflow vulnerability to overwrite EIP with the shellcode address.
4. Return from the vulnerable function.
5. Start to execute the shellcode.

Key insight of defense:

- Make some critical steps more difficult or even impossible to achieve.
- The attacker can only crash the system, but not hijack the control flow to execute arbitrary code.
- This is possibly denial-of-service attacks. Availability is not the main consideration of our threat model. Integrity is more important.

Recall: Steps of Stack Smashing Attack

1. Find a buffer overflow vulnerability in the program
2. Inject shellcode into a known memory address
3. Exploit the buffer overflow vulnerability to overwrite EIP with the shellcode address.
4. Return from the vulnerable function.
5. Start to execute the shellcode.

Solution:

- Address Space Layout Randomization (ASLR)