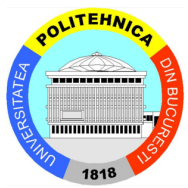


Session 0x0C
Fuzzing

Security Summer School

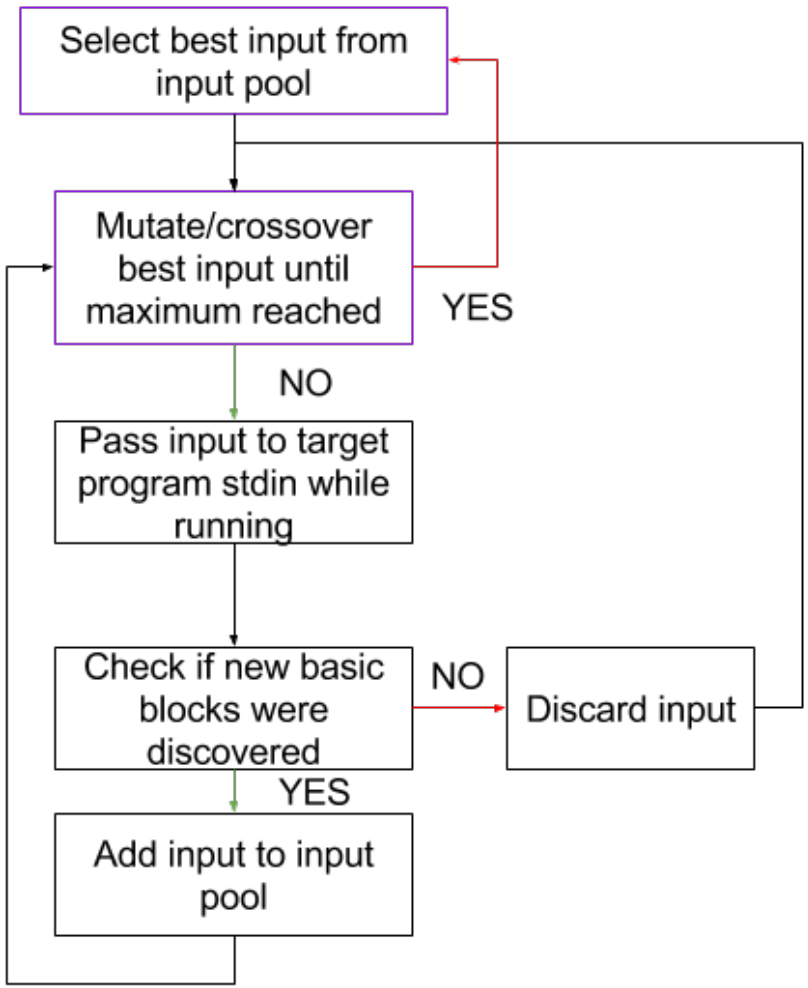
ACS/Ixia/Hexcellents

Fuzzing

- Generate large number of possible inputs
- Use heuristics in order to guide the execution
- Run the target program and collect execution feedback
- eg. libFuzzer(requires source code), afl(source code or binary)

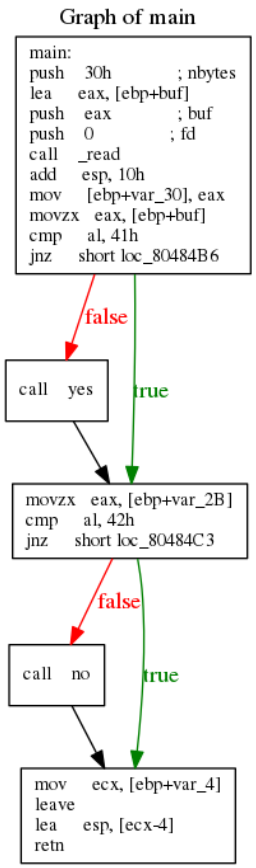
Fuzzing

- The main components of a fuzzing engine are shown below:



Flow Graph

- The flow graph is the target program layout representing all the paths that can be traversed during program execution
- It consists of nodes, representing basic blocks, and edges, representing whether or not there is a direct jump from a basic block to another



Heuristics Based on Coverage

- Basic block - set of instructions that doesn't contain branches - excepting the branch in and the branch out
- Code coverage - the number of basic blocks exercised during target executions in certain conditions (stdin, env, command line parameters, network, disk, etc.)
- Fuzzers run the target in different conditions in order to execute as many basic blocks as possible
- After each execution, if a new basic block was exercised, then the current conditions are **interesting**
- Only interesting inputs are mutated

Assisted Fuzzing

- Fuzzers get stuck often
- Branches can be simple (bytes are compared with precise values)

```
804852c:      0f b6 45 d8           movzx  eax,BYTE PTR [ebp-0x28]
8048530:      3c 41                 cmp    al,0x41
8048532:      75 0a                 jne    804853e <run+0x56>
```

- Or complex - string comparisons, simple encryption algorithms, etc.
- Here is an example where 4 bytes are compared using string functions

```
8048509:      6a 04                 push   0x4
804850b:      68 10 86 04 08        push   0x8048610
8048510:      8d 45 d0              lea    eax,[ebp-0x30]
8048513:      50                    push   eax
8048514:      e8 97 fe ff ff        call   80483b0 <strncmp@plt>
```

Symbolic Execution

- One solution to generate input that can satisfy a complex comparison is to use symbolic execution
- Disadvantage: cannot be used on complex software. SE will explore each path in the program, thus resulting in **path explosion**