Fault Simulator Tutorial

The Basics

- Fault injection requires hardware
  - target and something to manipulate it

- We emulate injection with a simulator
  - emulate the effects of an injection
  - some typical faults: skip instructions, corrupt some memory, …
  - during the execution of any binary

- Configured via a file
  - each line: 1 fault specified

Triggering: When to inject the fault?

- Instruction Pointer
  - inject fault when instruction pointer (RIP) has certain value
  - not realistic, but great for testing
  - syntax: `@<RIP>` → `@0x401bd1`

- Instruction Counter
  - counts number of assembly instructions since start of program
  - similar to a cycle counter, much more realistic
  - (but a bit unreliable here)
  - syntax: `#<count>` → `#300`

Fault Spec: What fault to inject?

- `skip <bytes>`
  - moves instruction pointer by <bytes>
  - limited to +-15

- `zero <address>`
  - sets 4 bytes (int) starting at <address> to zero
- `havoc <address>`
  - sets 4 bytes (int) starting at <address> to a random value
- `bitflip <bit_index> <address>`
  - flips a bit (indexed with <bit_index>) at byte <address>
  - memory[address] ^= (1 << bit_index)

Demos

- Examine source and determine exploitation path
  - check what faults are allowed!
- Examine binary (disassembly) to find error positions
- If needed: use debugger to find addresses
- Insert your faults into the script
- …and do some trial and error

Some Notes

- **Target the precompiled binaries!**
  - for hacklets and for faults
  - recompilation on your system:
    different library and compiler versions → different addresses and cycle counts
  - compile yourself only for debugging, revert back afterwards
- **Use the Newsgroup!**
  - we are more than happy to help you