Automated Race Detection On Linux Driver

Using Symbolic Execution

Index

Index

Advantages of Symbolic Execution

- In general, debugging drivers on run-time is difficult because most of them require specific hardware devices to invoke their functionalities.

- Symbolic execution:
  - provide a symbolic hardware device to invoke a driver's functions .
  - explore various execution paths of driver using symbolic inputs.

Symbolic Hardware Device

- Symbolic hardware device:
  - All reads from such devices are symbolic and writes are discarded. Symbolic devices can also generate interrupts and handle DMA.
- In order to set up symbolic hardware device, we have to:
  - provide the configuration in lua:
    - Example
  - provide following flags/arguments for QEMU(the virtual machine):
    - Example

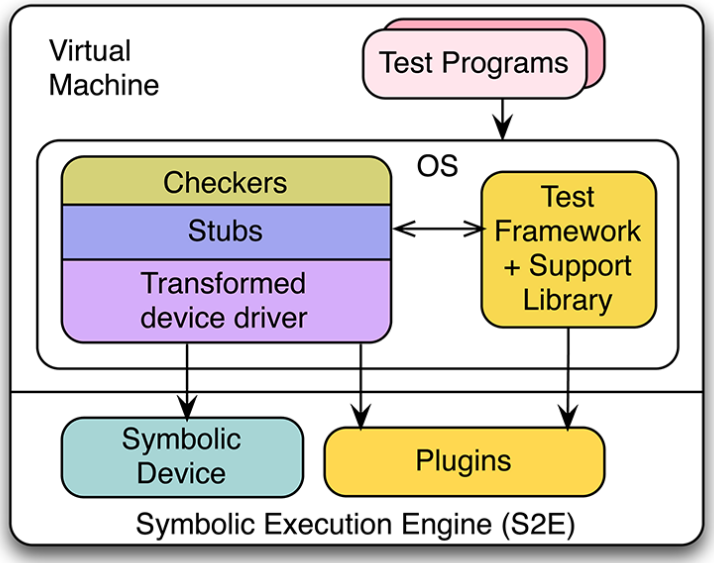
Symbolic Inputs

- All functions of s2e framework can be found in s2e.h.
- Function for making an input symbolic:
  - s2e_make_symbolic (void* buf, int size, const char* name).
  - Variation:
    - s2e_make_dma_symbolic (void* buf, int size, const char* name).
      - For direct memory access(dma)
        - Do not leave memory as symbolic afterward
          - Revert to underlying concrete value with

            s2e_free_dma_symbolic (void* buf, int size, const char* name).

Architecture of Symdrive

- Inputs:
  - Symbolic Device
  - Transformed device driver
    - Stubs
  - Test Framework
    - Checkers
  - Plugins
  - Test Programs
- Output:
  - Logs



Inputs Part 1

- Symbolic Device
  - See the slide: Symbolic Hardware Device.
- Transformed device driver
  - transformed by SymGen, a code transformation tool based on CIL.
    - Stubs
      - Interposed to every function call into and out of the driver by SymGen to call checkers.
- Test Framework
  - A kernel module that assists symbolic execution and executes checkers.
    - Checkers
      - Are been called before and after every function call to validate the driver behavior and check common bugs within the current driver function.

Input Part 2

- Plugins
  - Customizable tool provided by s2e to oversee the execution of paths in symbolic execution.
- Test Programs
  - User-mode C programs that invoke driver functions that deal with files such as read, write, and ioctl.
  - Is able to use functions of s2e framework such as s2e_make_symbolic.
    - Example:
      - int main() {

        int test = 0;

        s2e_make_symbolic(&test, sizeof(int), “test_number”);

        … }

Output

- Logs
  - Display on-screen.
  - Generated into txt file.
- Function for printing out messages
  - int uprintk (const char *fmt, ...)
    - The function implemented by SymDrive in test framework to print out messages on-screen
  - printf is still supported by the framework.

Analysis of leds-lp5523

- Export multiple dev files.
  - automatically created by using the function sysf_create_group.
    - this function creates this group of files in sysf file system.
- Each file has driver functions mapped to its read and write call.
  - Example: File name: engine1_mode
    - the driver function that is mapped to its read operation: show_engine1_mode
    - the driver function that is mapped to its write operation: store_engine1_mode
    - The mapping is done by the macro DEVICE_ATTR(name, mode, read, write)

Dev File

- Represent the device in sysfs filesystem.
- Created by typing the command mknod in the terminal.
  - need to provide unique major and minor numbers in order to represent the device.
- Use to perform file operations such as read, write, and ioctl.
- Can be automatically generated by the device driver in file sytem without using the command mknod.
  - Example: leds-lp5523.

Analyzing Procedure

- Insmod test framework.
- Insmod transformed device driver.
  - invoke init function.
  - probe function will be invoked if the symbolic device is been recognized by the transformed device driver module.
- run the test program.
  - test file operations and invoke related driver functions.
- rmmod transformed device driver.
  - invoke exit function.
- This procedure is able to invoke all the driver functions in leds-lp5523.

Analysis of Shared Memory

- SymGen creates common patterns in every driver function during transformation that ease the difficult of analysis.
  - All the variables are declared on top of the function.
  - every driver function’s parameters are passed to prefn and postfn functions of this driver function.
- An example is provided in next few slides.

Shared Memory Analysis Example Part 1

- leds-lp5523 driver function:
  - static ssize_t show_max_current(struct device *dev, struct device_attribute *attr, char * buf)

    {

        struct led_classdev *dev_cdec = dev_get_drvdata(dev);

        struct lp5523_led *led = cdev_to_led(led_cdev);

        ...

    }

Shared Memory Analysis Example Part 2

- After SymGen transformation:
  - static ssize_t show_current(struct device *dev, struct device_attribute *attr, char * buf)

    {

    struct led_classdev *dev_cdec; #All the pointers are listed on top of the function

    struct lp5523_led *led;

    int _call_kernel_fn_;

    _call_kernel_fn_=prefn_show_current(&dev, &attr, &buf);

    dev_cdec = dev_get_drvdata(dev);

    led = cdev_to_led(led_cdev);

    ...

    _call_kernel_fn_=postfn_show_current(&dev, &attr, &buf);

    return …;

    }

Explanation and Logging of Shared Memory

- Explanation:
  - prefn_show_current and postfn_show_current can be thought as stubs that are being called before and after the driver function to find checkers to check driver behavior and find common bugs.
- Logging the variables (or pointers) and parameters.
  - Variables (or pointers)
    - printf statement on every variable (or pointer) is inserted before return call.
  - Parameters
    - printf statement is inserted in prefn_show_current and postfn_show_current to print out their parameters.