Virtualization Techniques: Lab 1

Valgrind, a Process-VM for Observation of Process Execution

Functionality, Inner Workings, Writing your own Tool

Winter Term 18/19, November 14, 2018

Dai Yang, Josef Weidendorfer
CAPS I-10, TUM

Prerequisites

- Laptop with Linux (native or in VMWare/VirtualBox).
- Developer/Build tools installed: gcc (C compiler), automake
- Valgrind source from http://valgrind.org/downloads/valgrind-3.14.0.tar.bz2
- Patch for the skeleton of the Observer tool, "vg314ob.patch" from the lecture website

- Unpack Valgrind sources, patch in the Observer tool skeleton, compile and install:

```
tar xvf valgrind-3.14.0.tar.bz2
cd valgrind-3.14.0
patch -p1 < vg314-ob.patch
./autogen.sh
./configure  (use --prefix to specify non-default install directory, see below)
make -j4
```

  You can do "sudo make install", installing Valgrind into
  `/usr/local/`
  Remove the installation later with
  `sudo make uninstall`

  Alternatively, provide a path for installation (e.g. within your home directory) by using the configure option `--prefix=`<path> (in configure step above)

- Test this installation with

`/usr/local/bin/valgrind --tool=observer /bin/ls`

For more comfort, check that `/usr/local/bin` is in your $PATH before `/usr/bin` !

Task 1: Usage of existing „Valgrind tools"

Use the manual/online help of Valgrind.

- What is a „Valgrind tool"? What is "memcheck"? What is "helgrind"?

- Run „callgrind" to get the call graph and "executed instructions" for running "ls /usr/bin". Check the results with „callgrind_annotate" (or „kcachegrind", a visualization GUI).

What is the function where most instructions have been executed?

Task 2: Inner workings of Valgrind

- How does the tool „none“ work?
  - Run „valgrind --tool=none ls“, and then again with „`-v -v --stats=yes`“. Try to understand the additional output information.

- „valgrind --help-debug“ shows options for debugging/development of Valgrind tools.
  - What does „`--trace-flags=10100001`“ mean?
    Hint: As output will get huge, redirect the output into a file:
    („valgrind --tool=none --trace-flags=10100001 /bin/true &> log“)
    Hint 2: use „--trace-notbelow=0“ (Why?)
  - Try to explain the output.

- Identify the instrumentation (code inserted by the tool) of Callgrind (with/without cache simulator) for the first BBs. Try to understand the instrumentation.

Task 3: Writing your own tool: extend dummy „Observer“ tool

On the web site there is a tutorial from IISCW 2006. We go through the steps needed to write a tool. Important steps are from slide 53 onwards (the API changed since the tutorial was given: it uses „IRSB“ instead of „IRBB“). The skeleton for the new tool “observer” should be already compiled.

1. At the moment, the observer does the same as the none tool: it passes through all VEX instructions. Try to understand the code. Test the tool directly from the observer directory in the sources (the first line just creates an empty environment variable):

   ```
   export VALGRIND_LAUNCHER=
   ./observer-amd64-linux --tool=observer /bin/true
   ```

2. Add instrumentation into the observer sources to call “inc_guest_instr()” whenever a guest instruction is to be executed (this is marked by VEX as pseudo statement Ist_IMark. See also first “TODO” in “observer/ob_main.c”. For this, try to understand the comments for dirty helper calls, and check the API for generating VEX code in “VEX/pub/libvex_ir.h”.
3. Check your instrumentation with “--trace-flags=10100000”. How many instructions are executed in /bin/true? How much does the code size increase with the new instrumentation?
4. Add instrumentation to print out the address and size of every memory store access. See the second “TODO” remark. For this, a call to the helper “trace_store()” should be added whenever a store is about to be executed.
5. Again, use „--trace-flags“ to check your instrumentation. To test, better redirect the output into a file: “./observer-amd64-linux --tool=observer /bin/true &> log”
6. How can counters/statistics be collected separately for each source line?

Hints for enhancements:

- Trace/count the execution of FLOPs / memory allocations