nccgroup / **TriforceLinuxSyscallFuzzer**

A linux system call fuzzer using TriforceAFL

46 commits | **1** branch | **0** releases | **3** contributors

Branch: **master** | New pull request | Create new file | Upload files | Find file | Clone or download

jessemtso updated readme with dockerfile — Latest commit 86c9be5 on 5 Jan

crash_reports	adding crash reports	a year ago
docs	update docs and fix wrong numbers in testfile defn	a year ago
rootTemplate	write up local util for extract kallsyms	a year ago
.gitignore	add the various run scripts and the input generator and beef up docs.	a year ago
Makefile	minor tweaks	a year ago
README.md	updated readme with dockerfile	9 months ago
aflCall.c	remove unneeded clause	a year ago
argfd.c	fix IPPROTO_RAW sockets	a year ago
driver.c	fix bug that made bad parses run really slow, and fixed fflush (again)	a year ago
drv.h	initial import of driver source code	a year ago
gen.py	add vec32 arg type and fix bugs in child pid arg type	a year ago
gen2-shapes.txt	add generator for making syscalls of different shapes	a year ago
gen2.py	add generator for making syscalls of different shapes	a year ago
getSyms	write up local util for extract kallsyms	a year ago
getvmlinux	document the process for debugging	a year ago
heater.c	initial import of driver source code	a year ago
makeRoot	add the various run scripts and the input generator and beef up docs.	a year ago
parse.c	initial import of driver source code	a year ago
runCmd	lower RAM assigned to linux guests to 64M.	a year ago
runFuzz	lower RAM assigned to linux guests to 64M.	a year ago
runTest	lower RAM assigned to linux guests to 64M.	a year ago
sysc.c	add vec32 arg type and fix bugs in child pid arg type	a year ago
sysc.h	add pid and ref arg types to driver, add syscall filtering to driver,...	a year ago
testAfl.c	fix bug that made bad parses run really slow, and fixed fflush (again)	a year ago

README.md

TriforceLinuxSyscallFuzzer

- 20160613
- https://github.com/nccgroup/TriforceLinuxSyscallFuzzer
- Jesse Hertz jesse.hertz@nccgroup.trust
- Tim Newsham tim.newsham@nccgroup.trust

New: For those looking to play with TriforceAFL and TLSF, Richard Johnson created a Dockerfile which installs both (and even builds a Linux kernel for you). It's available here https://hub.docker.com/r/moflow/afl-triforce/tags/.

This is a collection of files used to perform system call fuzzing of Linux x86_64 kernels using AFL and QEMU. To use it you will need TriforceAFL from https://github.com/nccgroup/TriforceAFL and a kernel image to fuzz. Scripts assume that TriforceAFL is found in `$TAFL` or `../TriforceAFL/` (N.B. building `testAfl` requires that `../TriforceAFL/config.h` exist).

Building

To build:

```
make
```

Fuzzing

To run, first install a kernel into `./kern/bzImage` and extract `/proc/kallsyms` into `./kern/kallsyms`. Set `K=kern` environment variable to point to your kernel. Now run:

```
make inputs
./runFuzz -M M0
```

Note that the `runFuzz` script expects a master or slave name, as it always runs in master/slave mode. See the `runFuzz` script for more usage information.

Also Note that this only creates a small set of example inputs. To test a large number of important system calls, you will probably want to generate one example of each system call, or at least one example for every "shape" of system call. These should be placed in `inputs/`. See `gen2.py` for an example.

Reproducing

To reproduce test cases (such as crashes) run:

```
./runTest inputs/ex1
./runTest outputs/crashes/id*
```

You can also run the driver out of the emulated environment with the `-t` option, with verbose logging with `-vv` and without actually performing the system calls with `-x`:

```
./driver -tvvx < inputs/ex1
strace ./driver -t < inputs/ex1
```

It is sometimes useful to be able to boot the kernel and interactively run tests. To do so, edit the rootTemplate files as you see fit (for example, to add more test tools to the root filesystem), then run:

```
./runCmd
```

Other commands other than the shell can be invoked by specifying them as command line arguments to `runCmd`. Note: when done with the shell, use `^A-c` to get the QEMU prompt and type `quit`.

Debugging

Debugging is easiest with a kernel built with debugging symbols enabled. Use `runTest` to start the kernel and run a test through the driver, or use `runCmd` to manually run a test case from the shell. Edit your run script to include the `-s` option when starting `afl-qemu-system-trace`. This will enable `gdb` support on TCP port 1234. Use `getvmlinux` to extract the `vmlinux` kernel image from your `bzImage` kernel and run gdb after the system has booted:

```
cp kern/bzImage .
./getvmlinux
gdb ./vmlinux
target remote :1234
```

```
break somefunction
continue
```

You can attach the debugger after `runTest` has caused a crash or before you manually trigger then bug in `runCmd`.

Note that Linux sources are compiled with optimization turned on by default. This can make debugging confusing and difficult. You can disable optimization on a file-by-file basis by editing the Linux make file for the subdirectory a file is in and adding `CFLAGS_name.o = -O0` to the `Makefile`. For example editing `kernel/Makefile` and adding `CFLAGS_sys_ni.o = -O0` will disable optimization when building `kernel/sys_ni.o`.

Utility

The `getSyms` shell script uses `runCmd` to execute `cat /proc/kallsyms` and extract it to a local file named `kallsyms`. This is typically used to prep your kernel for fuzzing:

- run `K=yourKernDir ./getSyms` to get `kallsyms`
- run `mv kallsyms yourKernDir` to install it

Bugs

Note: When fuzzing a Linux 2.* kernel you will need to enable the CPU timer. When the timer is not enabled panic and logging detection do not seem to operate properly and panics result in hangs. To enable the timer, call `startForkserver(1)` in `driver.c` instead of `startForkserver(0)`. This issue does not seem to occur in Linux3.* and Linux4.* kernels.