Filesystem Fuzzing with American Fuzzy Lop

Vegard Nossum <vegard.nossum@oracle.com>
Quentin Casasnovas <quentin.casasnovas@oracle.com>

Oracle Linux and VM Development – Ksplice team
April 21, 2016

ORACLE

Program

1. What is American Fuzzy Lop (AFL)
2. Porting AFL to the kernel
3. Applying AFL to filesystem fuzzing
4. Questions / Demo

ORACLE

Time to first bug

Filesystem	Time	Type
Btrfs	5s	BUG()
Ext4	2h	BUG()
F2fs	10s	BUG()
Gfs2	8m	Double free
Hfs	30s	Page Fault
Hfsplus	25s	Page Fault
Nilfs2	1m	Page Fault
Ntfs	4m	Soft lockup
Ocfs2	15s	BUG()
Reiserfs	25s	BUG()
Xfs	1h45m	Soft lockup

- Linux 4.3 or newer
- 3 AFL instances on my laptop
- Don’t believe us?
  - Live crash demo

ORACLE

What is American Fuzzy Lop?

(This is not related)

What is American Fuzzy Lop?

- Fuzzing
  - Software testing technique (black box)
  - Generate malformed input
  - Find interesting behaviours
- AFL is unique
  - Genetic fuzzer: uses branch instrumentation
  - Amazingly good to find deep/unusual paths
  - Found hundreds of security vulnerabilities
- Open source, developed by Michal Zalewski (lcamtuf)
  http://lcamtuf.coredump.cx/afl/

ORACLE

What is American Fuzzy Lop?

The power of coverage based fuzzing

```
while true; do ./lottery < /dev/urandom && break ; done
```

```
/* lottery.c */
int main(void)
{
   if (getchar() != 0x42 || getchar() != 'K' || getchar() != 's' ||
       getchar() != 'p'  || getchar() != 'l' || getchar() != 'i' ||
       getchar() != 'c'  || getchar() != 'e' || getchar() != '\n')
       return 1;
   return win_lottery();
}
```

- One chance / $2^{\text{BITS\_PER\_BYTE} * 9}$ = 4722366482869645213696 to win the lottery…

What is American Fuzzy Lop?

The power of coverage based fuzzing

- Instrument branches
- Use coverage as feedback loop
  - Keep inputs that generates new paths
  - Mutate those inputs
- Win the lottery in at most
  (1 << BITS_PER_BYTE) * 9 = 2034 iterations

- Think of very complex parsers with hundred of branches

ORACLE

What is American Fuzzy Lop?

The feedback loop

- Shared memory between afl-fuzz and target
- Branch edge increments a byte in the shm
- Allows to differentiate A > B from B > A

```
/* afl-fuzz.c */
while (1) {
        run_target(input);
        cov = gather_coverage(shared_mem);
        input = mutate_input(cov)
        memzero(shared_mem);
}
```

ORACLE

Porting AFL to the kernel

Porting AFL to the kernel

Instrumenting the branches, how?

- AFL in userland
  - GNU as wrapper
    - search conditonal jmp instructions
    - instrument each edge with some AFL stub:
      - embeds a fork server
      - configures shared memory
      - writes branch taken into shared memory

```
/* AFL 101 */
$ CC=afl-gcc ./configure
$ make lottery
$ afl-fuzz -i [input_dir] -o [output_dir] -- ./lottery @@
```

Porting AFL to the kernel

Instrumenting the branches, how?

- First approach
  - Take the GNU as wrapper approach
    - Remove userland AFL stub
    - Add a call to C function at every edge
    - Implement the C function in the kernel
    - Works with any GCC version
  - Not ideal:
    - Need to use afl-as for every compilation unit
    - Save all callee clobbered registers

ORACLE

Porting AFL to the kernel

Instrumenting the branches, how?

- Second approach
  - Use a GCC plugin
    - Dmitry Vyukov wrote a GCC patch for syzkaller [1]
    - Port the patch to its own plugin
      - No need to recompile GCC :)
  - Dmitry’s plugin
    - Run at GIMPLE level after all generic optimizations
    - Call a function (our stub) at each “Basic Block”
    - GCC knows register allocations :)

[1] https://github.com/google/syzkaller

ORACLE

Porting AFL to the kernel

Instrumenting the branches, visual example

```c
/* example.c */
void foo(int x) {
    if (x)
        do_stuff(x);
    else
        do_other_stuff(x);
}
```

instrument →

```c
/* example.c */
void foo(int x) {
    __afl_stub();
    if (x) {
        __afl_stub();
        do_stuff(x);
    }
    else {
        __afl_stub();
        do_other_stuff(x);
    }
    __afl_stub();
}
```

- __afl_stub() uses RET_IP as an index to the shared memory

Porting AFL to the kernel

Shared memory, how?

- Need for shared memory between afl-fuzz/kernel
  - per task
- `/dev/afl` driver

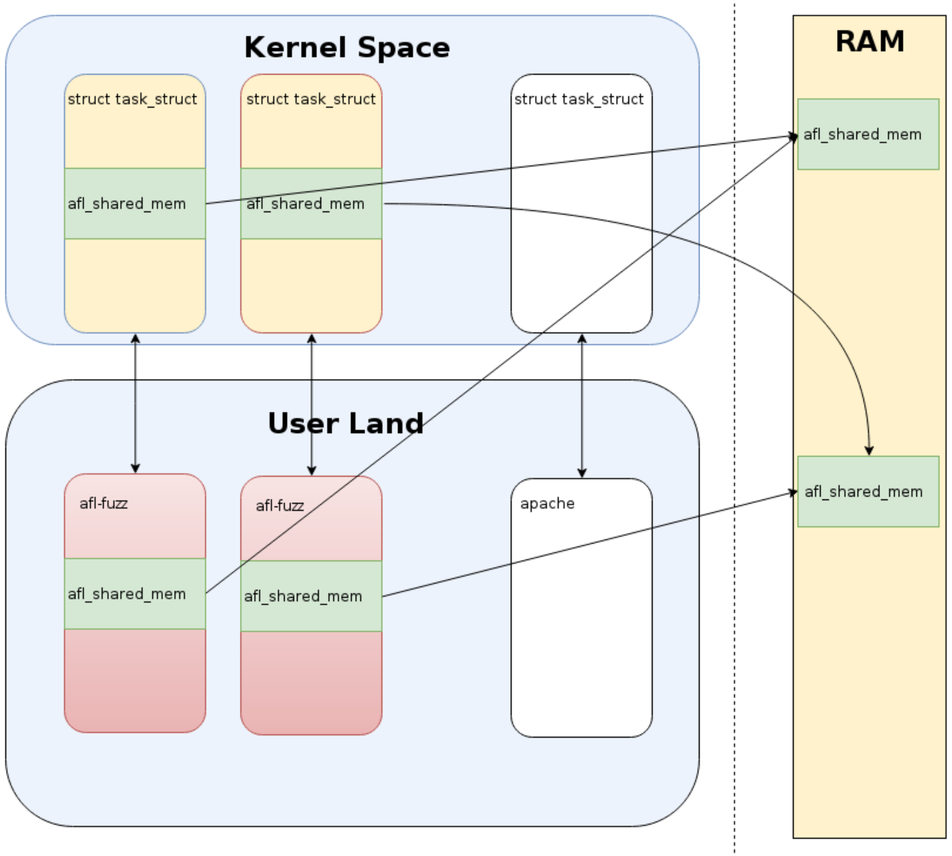
```
/* afl-fuzz.c - USERSPACE */
int afl_fd = open("/dev/afl");
shared_mem = mmap(afl_fd);
while (1) {
    ...
}
```

```
/* drivers/afl.c - Kernel */
int afl_mmap(...) {
   current->afl_shm = vmalloc();
   /* Magic here to map afl_shm
    * in the userspace mm */
}

void __afl_stub() {
   current->afl_shm[RET_IP]++;
}
```

Porting AFL to the kernel

Shared memory, visual



Applying AFL to filesystem fuzzing

Filesystem fuzzing: Overview

- Writing a filesystem-specific fuzzer
- Starting a fuzzer instance
- Challenges:
  - Dealing with large filesystem images
  - Dealing with filesystem checksums
  - Virtualisation overhead
  - Execution indeterminism
- Next steps/where to go from here

ORACLE

Writing a filesystem-specific fuzzer: ingredients

- A list of sources/source directories to instrument
  - e.g. `fs/ext4/`
- A list of config options to enable/disable
  - e.g. `CONFIG_EXT4_FS=y`
- The stub itself
  - set up loopback device/mount point/etc.
  - expand sparse image to real image
  - call `mount()`
  - filesystem activity
- A set of initial filesystem images

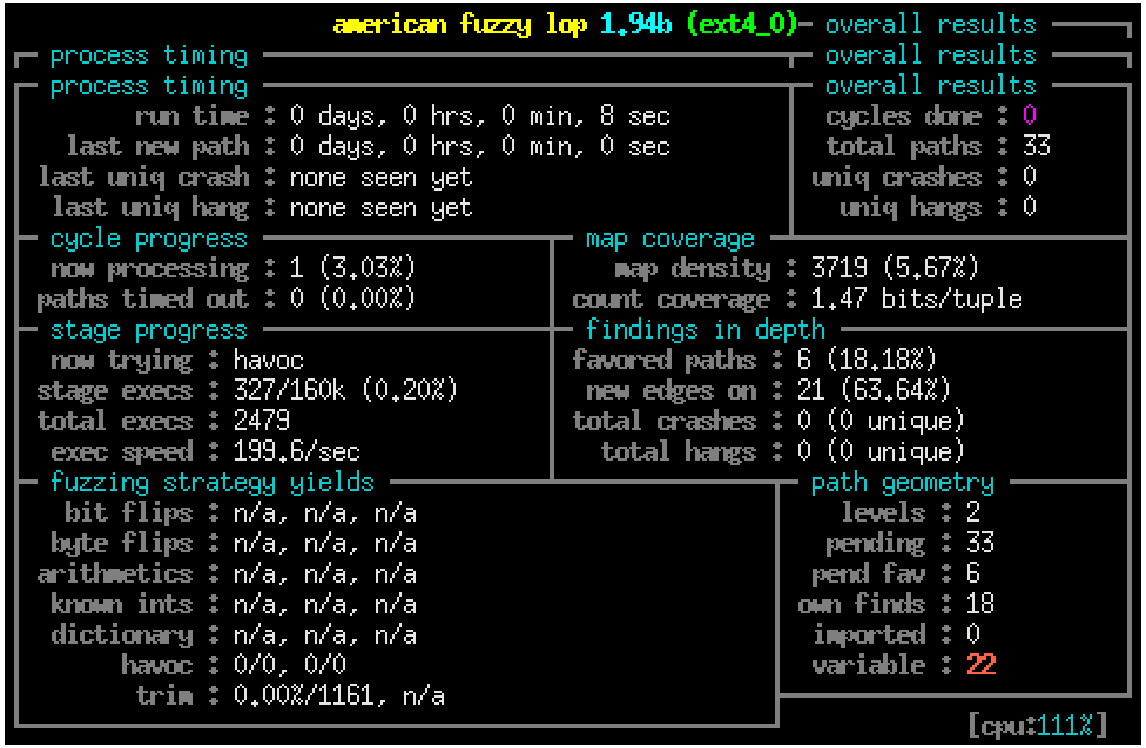
ORACLE

Starting a fuzzer instance – it's easy!

- Edit top-level config.yml:
  - Point it to afl.git, linux.git, and the branches to use
  - (Optionally) point it to a specific gcc binary to use
- Building AFL+kernel+fuzzer and running:

```
./start ext4 0
```

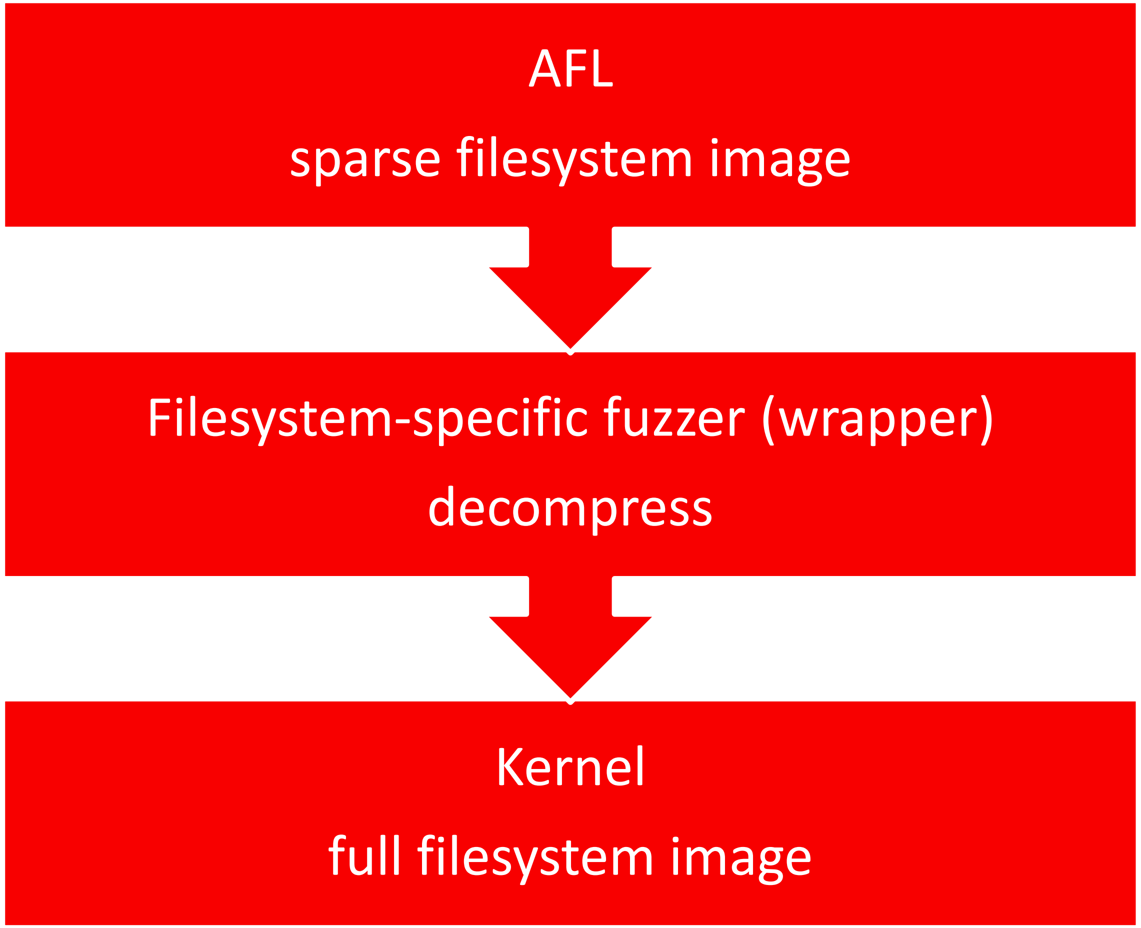
- That's it.



ORACLE

Challenge: Large filesystem images

- AFL works best with small images (smaller is better), 1 MiB max
- Many filesystems have minimum size requirements
- Idea: Only fuzz the “important” bits:
  - All-zero areas are excluded from the fuzzing process as they most likely represent empty/unused space
  - AFL only works with sparse filesystem images
  - Kernel only works with full filesystem images



ORACLE

Challenge: Large filesystem images

- Remember: we start with an initial set of filesystem images
- Split input file in chunks (e.g. of 64 bytes)
- Idea 1: Only include chunks which are non-zero
  - We often have long runs of non-zero chunks; combine
- Idea 2: Detect frequently repeating chunks
- Tool to “compress” and “decompress” filesystem images
- Some filesystems (e.g. GFS2) write out many non-repeating structures
  - Maybe block numbers or other bookkeeping
  - Needs filesystem-specific code to compress to reasonably small test-cases

ORACLE

Challenge: filesystem checksums

- Large obstacles for the fuzzer to get past
- Serve no purpose on a known-corrupt filesystem
- Solution 1:
  - comment out the filesystem code in the kernel
  - your test-cases no longer reproduce on a stock kernel ☹
  - possibility of introducing a bug of your own in the kernel
- Solution 2 (preferred):
  - calculate correct checksums before passing image to kernel
  - can require a lot of effort depending on filesystem
  - slightly slower, but hardly noticeable

Challenge: Virtualisation overhead (enter UML)

- Problem: KVM was really slow (~30 execs/sec)
- Solution: Compile the kernel as a regular userspace program (UML)
- To compile:

```
make ARCH=um
```

- To run:

```
./vmlinux rootfstype=hostfs rw init=/bin/bash
```

- `SMP=n; PREEMPT=n` lowers overhead *and* increases determinism
- Result: 60x speedup
- More info: http://user-mode-linux.sourceforge.net/

ORACLE

Challenge: execution indeterminism

- Goal: each execution should be deterministic and independent
- Asynchronous code/interrupts
  - Interrupts clobber the instrumentation feedback buffer
  - Solution: disable instrumentation in interrupts
- `printk()` ratelimiting
  - causes changes to persistent state that affect later testcases
  - Solution: either always filter or always allow message through
- Work offloading to kthreads (e.g. async/workqueues)
- Disabling SMP and preemption helps!

Next steps: Regression suites

  - Running AFL results in a set of filesystem images
  - These images trigger distinct code paths in the kernel
  - Idea: We can use the images as a regression suite
  - For every new commit, run all the tests
- If you are a filesystem developer, we challenge you to:
  - Keep track of all the images found by AFL in a git repo
  - Use these images as part of automated regression testing
  - Use these images to generate coverage reports for your filesystem
- Much of the work has already been done!

ORACLE

Example: btrfs coverage report

```
 1 | int btrfs_delayed_update_inode(struct btrfs_trans_handle *trans,
   |                                struct btrfs_root *root, struct inode *inode)
   | {
   |         struct btrfs_delayed_node *delayed_node;
   |         int ret = 0;
   |
 1 |         delayed_node = btrfs_get_or_create_delayed_node(inode);
 1 |         if (IS_ERR(delayed_node))
 0 |                 return PTR_ERR(delayed_node);
   |
 1 |         mutex_lock(&delayed_node->mutex);
 1 |         if (test_bit(BTRFS_DELAYED_NODE_INODE_DIRTY, &delayed_node->flags)) {
 0 |                 fill_stack_inode_item(trans, &delayed_node->inode_item, inode);
 0 |                 goto release_node;
   |         }
   |
 1 |         ret = btrfs_delayed_inode_reserve_metadata(trans, root, inode,
   |                                                    delayed_node);
 1 |         if (ret)
   |                 goto release_node;
   |
 1 |         fill_stack_inode_item(trans, &delayed_node->inode_item, inode);
 1 |         set_bit(BTRFS_DELAYED_NODE_INODE_DIRTY, &delayed_node->flags);
 1 |         delayed_node->count++;
 1 |         atomic_inc(&root->fs_info->delayed_root->items);
   | release_node:
 1 |         mutex_unlock(&delayed_node->mutex);
   |         btrfs_release_delayed_node(delayed_node);
 1 |         return ret;
   | }
```

- Start a kernel with gcov support
- Run *all* testcases sequentially
- Format output (here: Jenkins)

- Surface analysis shows a lot of error conditions (e.g. out of memory) are not covered by AFL
- We can use coverage information to nudge the fuzzer in the right direction
  - e.g.: xattr code was never run, we need to read/write xattrs on the mounted fs

ORACLE

Next steps: Finding concurrency issues

- AFL fundamentally relies on testcase determinism
    - The same testcase always results in the same code paths taken
    - Syzkaller is better suited for finding concurrency issues
- What about finding bugs due to race conditions?
  - For each filesystem image found by AFL, mount and run a parallel test suite
    - e.g. syzkaller or trinity
  - Results will be less precise and indeterministic
- Conjecture: If a particular filesystem image causes different paths to be taken for sequential operations, it will also cause different paths to be taken for parallel operations

ORACLE

Questions / Demo