Fuzzing Filesystems on NetBSD via AFL+KCOV

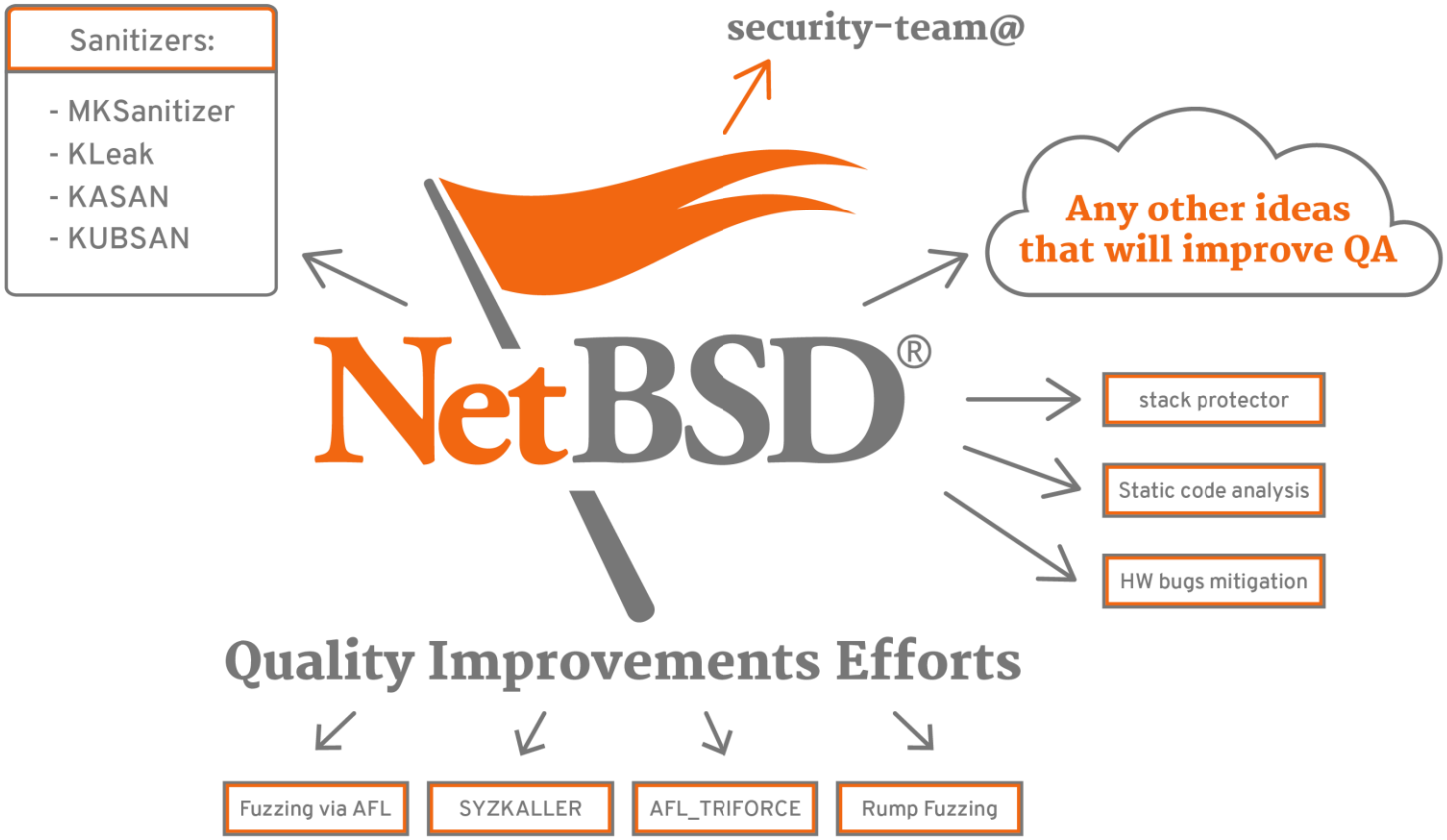
Maciej Grochowski
Maciej.Grochowski[at]protonmail.com

EuroBSDcon 2019
Lillehammer, Norway
Lillehammer — September 19-22, 2019

Outline

- NetBSD-qa
- What the fuzzing is?
- Porting AFL to fuzz NetBSD kernel
- Coverage and generic **kcov**
- Basic fuzzing setup
- Fuzzing the FS
- Conclusions

NetBSD-QA

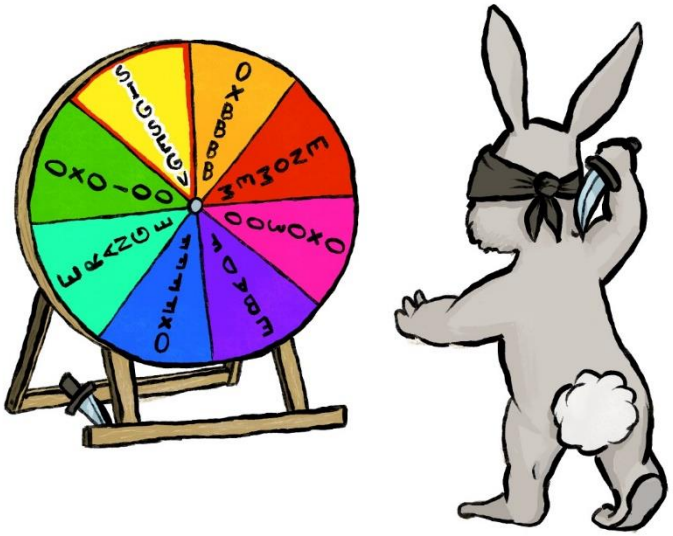


Join us on: [Freenode] #NetBSD-qa

Fuzzing: *The dumb fuzzer*

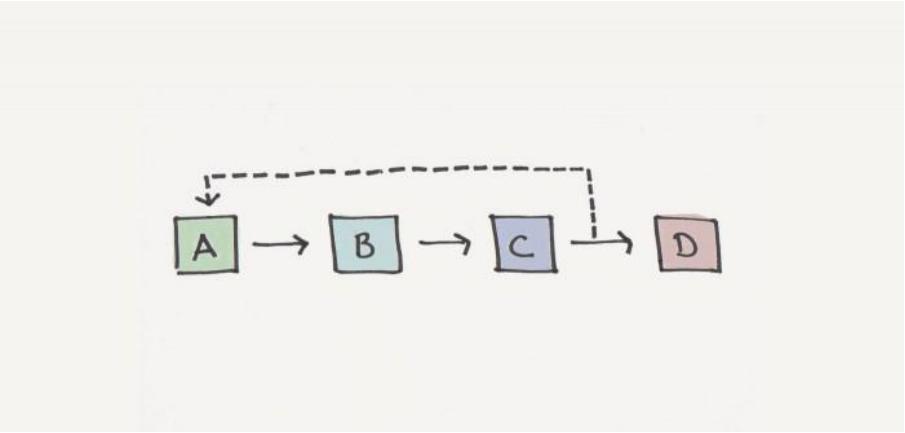
- The simplest fuzzer can be written in a few lines:

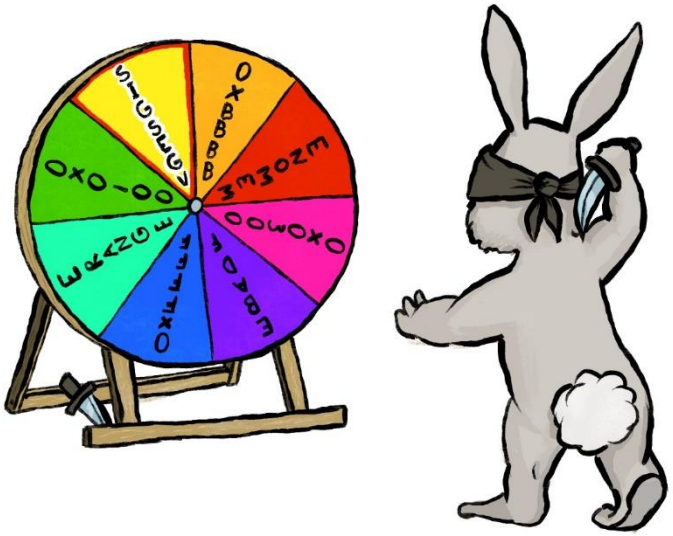
```
while true;
do;
dd if=/dev/random bs=1 count=1000 > testcase;
./fuzz_bin ./testcase;
done
```



Smarter Fuzzing

- More advanced fuzzers
  - Mutational
  - Evolutionary
- Changes based on the feedback loop





Coverage based fuzzing

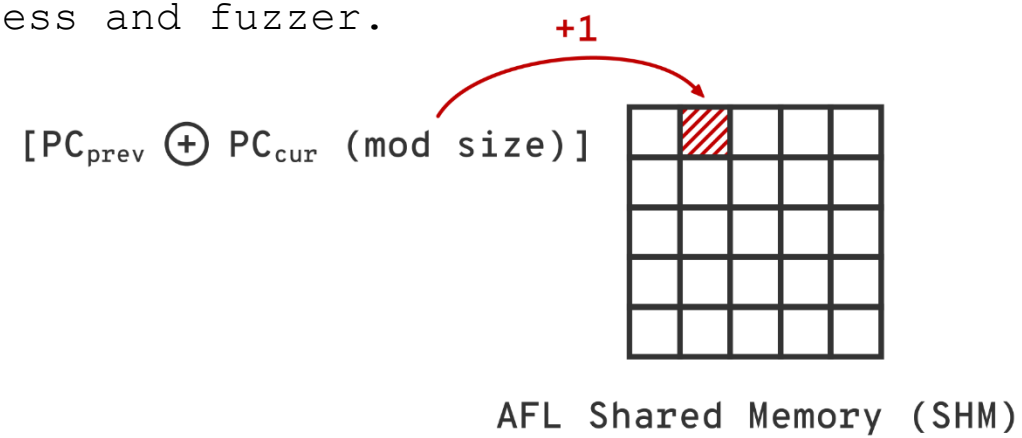
- AFL uses its own format of data:
  - Map/Array of pairs: {src, dst} execution branch
  - Every byte set in the map can be thought of as a hit for a particular [*branch_src*, *branch_dst*] tuple in the instrumented code.
  - Historically that was done by custom binary instrumentation
- NetBSD has kcov(4) which provides:
  - PC Trace
  - CMP trace

Porting AFL

NetBSD KCOV exposes raw coverage: PC, CMP
Plans to brings other formats like DIV, GEP

AFL uses 64kB buffer of the custom format
Shared between running process and fuzzer.

```
0xffffffffff
0xffffffffff
0xffffffffff
0xffffffffff
0xffffffffff
0xffffffffff
0xffffffffff
0xffffffffff
```



Not much modification on the AFL Fuzzer side
Replacing SHM_GET with MMAP for KCOV device

Porting AFL to kcov

- Modification to original KCOV
  - Plugin architecture
  - AFL works as module that register into kcov

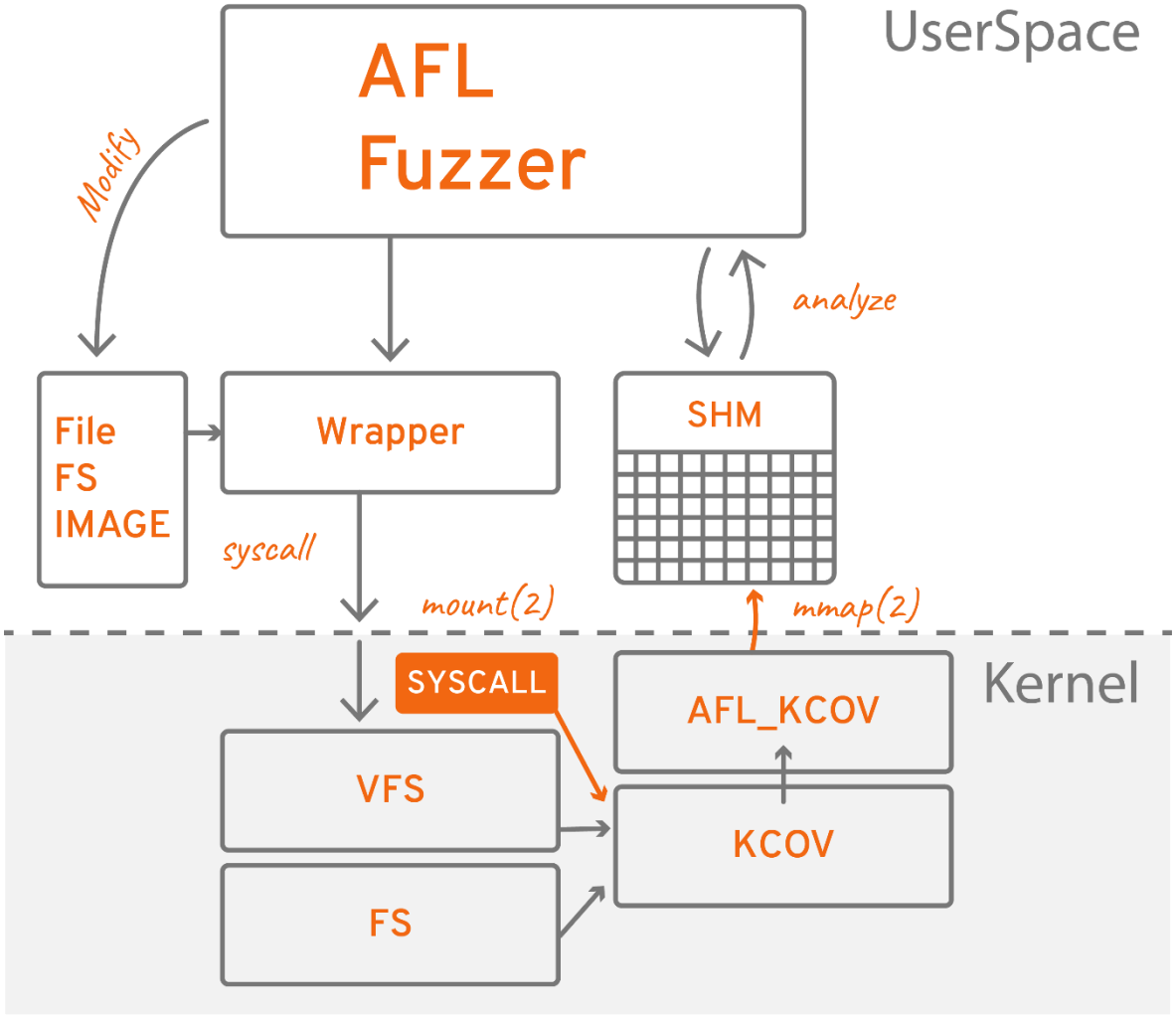
The registration require to fill such structure

```
struct kcov_ops kcov_mod_ops = {
    .open = kcov_afl_open,
    .free = kcov_afl_free,
    .setbufsize = kcov_afl_setbufsize,
    .enable = kcov_afl_enable,
    .disable = kcov_afl_disable,
    .mmap = kcov_afl_mmap,
    .cov_trace_pc = kcov_afl_cov_trace_pc,
    .cov_trace_cmp = kcov_afl_cov_trace_cmp
};
```

Generic kcov(4)

- Raw traces of PC, CMP (and potentially other in the future) inside kcov
- Data accessible via registered callbacks
- Can support multiple fuzzers using separate modules for each of them without unnecessary complexity
- Coverage data can be transformed or filtered

Fuzzing setup

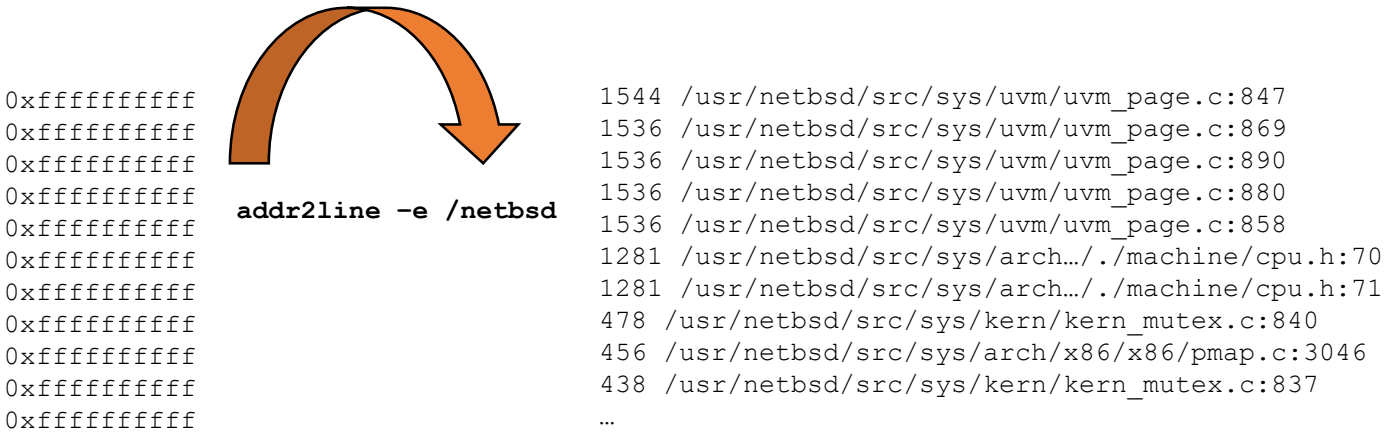


Coverage Data

- How to inspect what data is seen by fuzzer?

Run kcov process and print data

(see man kcov(4))



Removing the Noise

- Statically removing the noise `no_instrument_function`
- Dynamic removal based on blacklist

```
0xffffff... -> /usr/netbsd/src/sys/uvm/uvm_page.c:847
0xffffff... -> /usr/netbsd/src/sys/uvm/uvm_page.c:869
0xffffff... -> /usr/netbsd/src/sys/uvm/uvm_page.c:890
0xffffff... -> /usr/netbsd/src/sys/uvm/uvm_page.c:880
0xffffff... -> /usr/netbsd/src/sys/uvm/uvm_page.c:858
0xffffff... -> /usr/netbsd/src/sys/arch/amd64/compile/obj/GENERIC/./machine/cpu.h:70
0xffffff... -> /usr/netbsd/src/sys/arch/amd64/compile/obj/GENERIC/./machine/cpu.h:71
0xffffff... -> /usr/netbsd/src/sys/kern/kern_mutex.c:840
0xffffff... -> /usr/netbsd/src/sys/arch/x86/x86/pmap.c:3046
0xffffff... -> /usr/netbsd/src/sys/kern/kern_mutex.c:837
```

```
for address in address_list; do
        ioctl(kcov, address);
done;
```

Coverage benchmark

How fast can our fuzzer win the lottery?

```
if (buffer[0] == 'L' && buffer[1] == '0' &&
    buffer[2] == 't' && buffer[3] == 'T' &&
    buffer[4] == 'T' && buffer[5] == '\r' )
       printf("You Won the Panic Lottery!\n");
```

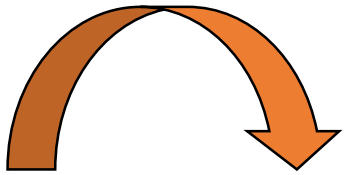
- Before removing noise
  - No progress after 2 weeks*
- After applying filtering
  - Less than 24h* (although some improvements still possible)

For the reference same code in Userspace takes few hours (between 1-4h*)

* These are just approximate values and are highly dependent on other variables!

FFS Mount Wrapper

```
# Expose tmpfs file as block device
vndconfig vnd0 /tmp/rand.tmp

# Create a new FS image on the blkdev
newfs /dev/vnd0

# Mount our fresh FS
mount /dev/vnd0 /mnt

# Check if FS works fine
echo "Mounted!" > /mnt/test

# Undo mount
umount /mnt

# Last undo step
vndconfig -u vnd0
```



[Raw syscalls]

mount(2)

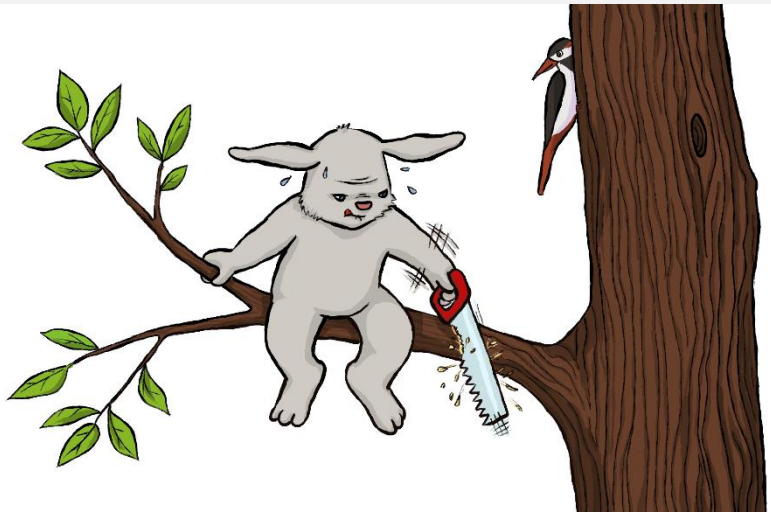
Reminder:

Performance is the key!

Local setup

```
# We need a block, big enough to fit FS image
dd if=/dev/zero of=./in/test bs=10k count=8

# A block is already inside fuzzer ./in
vndconfig vnd0 ./in/test

# Create new FFS filesystem
newfs /dev/vnd0

vndconfig -u vnd0
```

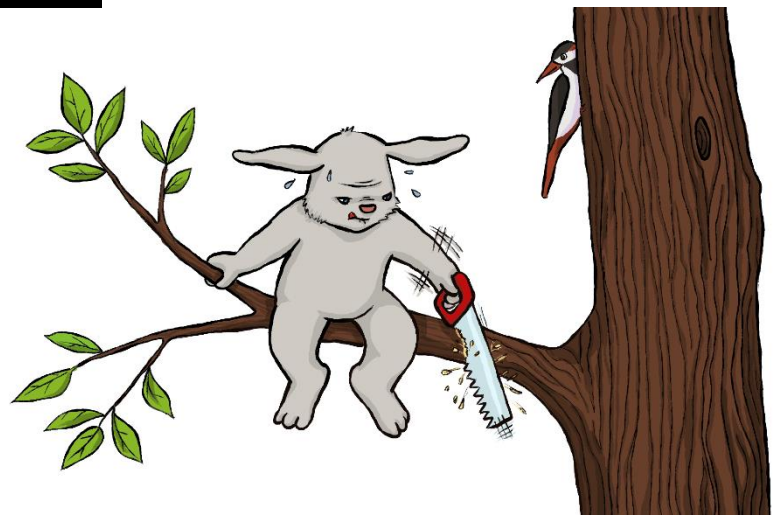
```
./afl-fuzz -k -i ./in -o ./out -- /path/wrapper_mnt.so @@
```



How many iteration to find a bug?

```
┌─ process timing ────────────────────────────────┐ ┌─ overall results ─┐
│        run time : 0 days, 0 hrs, 0 min, 17 sec  │ │  cycles done : 0  │
│   last new path : none seen yet                 │ │  total paths : 1  │
│ last uniq crash : none seen yet                 │ │ uniq crashes : 0  │
│  last uniq hang : none seen yet                 │ │   uniq hangs : 0  │
├─ cycle progress ────────────────┬─ map coverage ─┴───────────────────┤
│  now processing : 0 (0.00%)     │    map density : 17.16% / 17.18%   │
│ paths timed out : 0 (0.00%)     │ count coverage : 3.57 bits/tuple   │
├─ stage progress ────────────────┼─ findings in depth ────────────────┤
│  now trying : trim 512/512      │ favored paths : 1 (100.00%)        │
│ stage execs : 14/160 (8.75%)    │  new edges on : 1 (100.00%)        │
│ total execs : 201               │ total crashes : 0 (0 unique)       │
│  exec speed : 27.46/sec (slow!) │   total hangs : 0 (0 unique)       │
├─ fuzzing strategy yields ───────┴─ path geometry ────────────────────┤
│   bit flips : 0/0, 0/0, 0/0                │    levels : 1           │
│  byte flips : 0/0, 0/0, 0/0                │   pending : 1           │
│ arithmetics : 0/0, 0/0, 0/0                │  pend fav : 1           │
│  known ints : 0/0, 0/0, 0/0                │ own finds : 0           │
│  dictionary : 0/0, 0/0, 0/0                │  imported : n/a         │
│       havoc : 0/0, 0/0                     │ stability : 21.61%      │
│        trim : n/a, n/a                     ├─────────────────────────┘
└────────────────────────────────────────────┘           [cpu:  0%]
```

```
afl-fuzz: /dev/vnd0: opendisk: Device busy

# mount
/dev/wd0a on / type ffs (local)
...
tmpfs on /var/shm type tmpfs (local)
/dev/vnd0 on /mnt1 type ffs (local)

# ls /mnt*
ls: /mnt1: No such file or directory
/mnt:

# ls -alh /mnt1
ls: /mnt1: No such file or directory
```



Conclusions

- Coverage based tracking improved in NetBSD
- Very low entry bar to start fuzzing the NetBSD kernel code
- Filesystems are very important from OS quality perspective
- Run your fuzzing with different Sanitizers

Resources

- **blog.netbsd.org**
  - Write your own fuzzer for NetBSD kernel!
  - Fuzzing NetBSD Filesystems via AFL.
- Filesystem Fuzzing with American Fuzzy Lop Oracle 2016
- AFL project page
- Collection of mount wrappers

Questions?

Thank you!

Credit for graphics: @FableMode