UROP: Evaluating SnapFuzz with ProFuzzBench

adotinthevoid.github.io/talks/snapfuzz.pdf

Alona Enraght-Moony

2022-09-09

Snapfuzz [1]

- Greybox Fuzzer for statefull network application
- Fork of AFLNet [4], itself a fork of AFL [5]
- Uses SaBRe [2] to intercept syscalls to dramaticly increase speed.
  - Avoid syncronization delays
  - In memory filesystem
  - Optimize forkserver
  - Avoiding Sleep

ProFuzzBench [3]

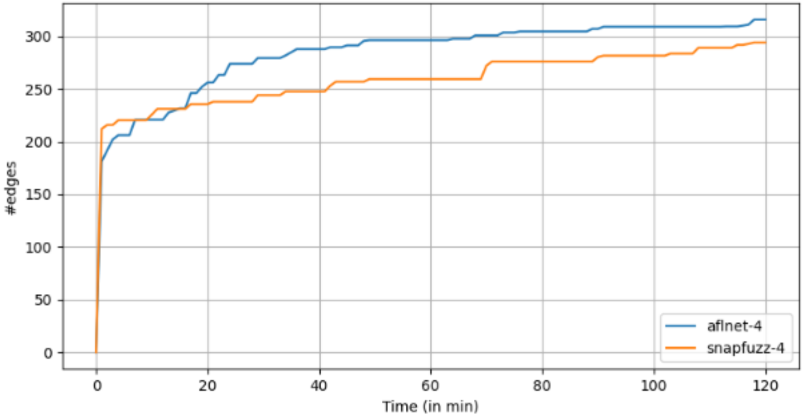
- Benchmark suite for statefull fuzzers.
- 10 Protocols, 13 Implementations.
- Patches for Derandomization, Initial seeds, etc.
- Fuzzing effectiveness measured by coverage.

Various Problems Starting Running

- Building snapfuzz
- Memory Limit
- AFL `ar`
- Finding `sabre`
- Ubuntu 20.04 Glibc Debug symbols. [1]
- Profuzzbench Graphing Hard Coded
- Typoing 50 to 50/ silently gives wrong output
- Analysis/plotting script hardcoded to existing fuzzers

[1] https://bugs.launchpad.net/ubuntu/+source/glibc/+bug/1918035

Perf Problems

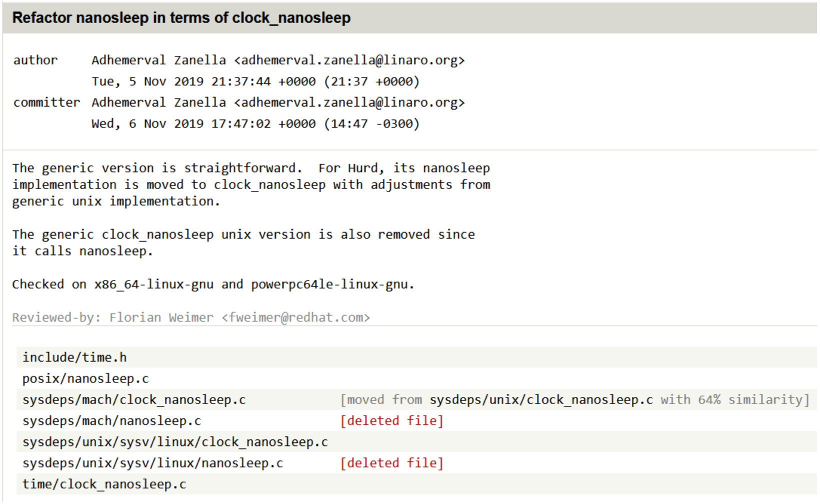


Fuzzer	Execs	Execs/Sec	Edge Cov %	Line Cov %
aflnet	47139.25	6.55	40.35	59.35
snapfuzz	12765.75	1.77	37.60	55.7

Wrong Trees Barked Up

- CPU Cores
- AFL Flags

18.04 $\rightarrow$ 20.04 Glibc changes



Refactor nanosleep in terms of clock_nanosleep

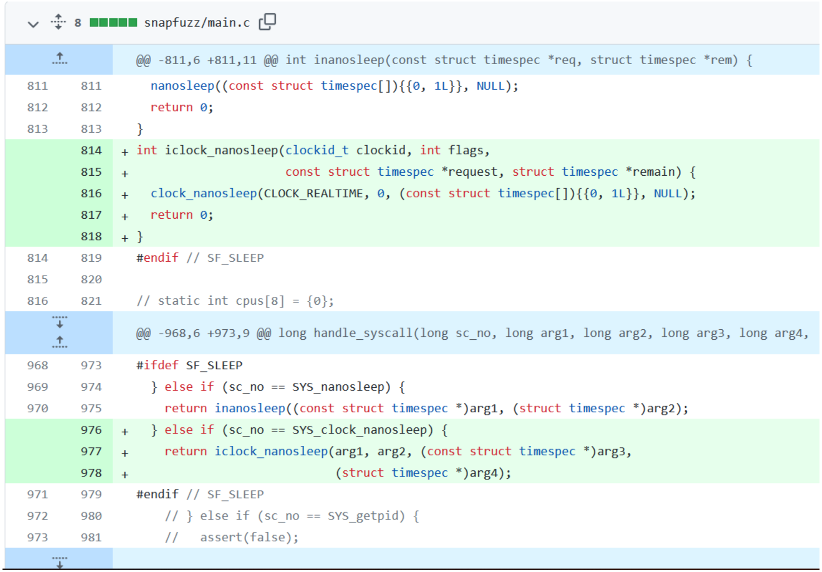
```
author     Adhemerval Zanella <adhemerval.zanella@linaro.org>
           Tue, 5 Nov 2019 21:37:44 +0000 (21:37 +0000)
committer  Adhemerval Zanella <adhemerval.zanella@linaro.org>
           Wed, 6 Nov 2019 17:47:02 +0000 (14:47 -0300)
```

```
The generic version is straightforward.  For Hurd, its nanosleep
implementation is moved to clock_nanosleep with adjustments from
generic unix implementation.

The generic clock_nanosleep unix version is also removed since
it calls nanosleep.

Checked on x86_64-linux-gnu and powerpc64le-linux-gnu.

Reviewed-by: Florian Weimer <fweimer@redhat.com>
```

```
include/time.h
posix/nanosleep.c
sysdeps/mach/clock_nanosleep.c            [moved from sysdeps/unix/clock_nanosleep.c with 64% similarity]
sysdeps/mach/nanosleep.c                  [deleted file]
sysdeps/unix/sysv/linux/clock_nanosleep.c
sysdeps/unix/sysv/linux/nanosleep.c       [deleted file]
time/clock_nanosleep.c
```

3537ecb49cf7177274607004c562d6f9ecc99474

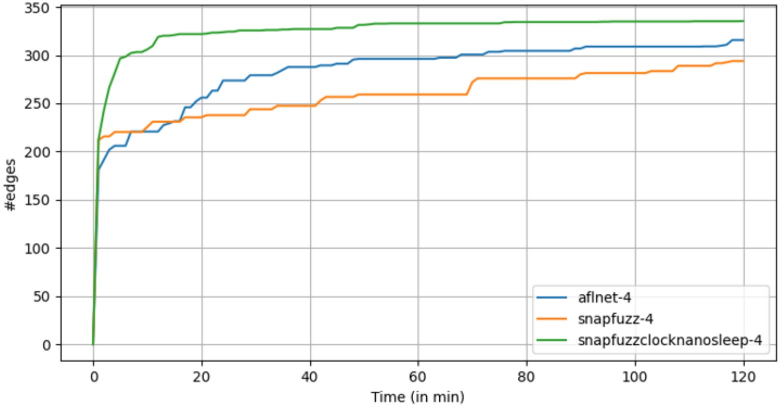
Fix



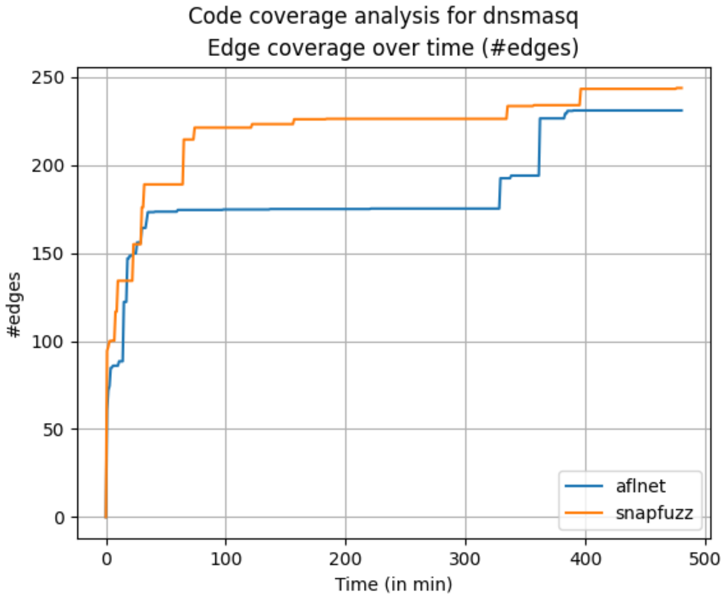
```
8 snapfuzz/main.c
@@ -811,6 +811,11 @@ int inanosleep(const struct timespec *req, struct timespec *rem) {
811 811     nanosleep((const struct timespec[]){{0, 1L}}, NULL);
812 812     return 0;
813 813   }
    814 + int iclock_nanosleep(clockid_t clockid, int flags,
    815 +                      const struct timespec *request, struct timespec *remain) {
    816 +   clock_nanosleep(CLOCK_REALTIME, 0, (const struct timespec[]){{0, 1L}}, NULL);
    817 +   return 0;
    818 + }
814 819   #endif // SF_SLEEP
815 820
816 821   // static int cpus[8] = {0};
@@ -968,6 +973,9 @@ long handle_syscall(long sc_no, long arg1, long arg2, long arg3, long arg4,
968 973   #ifdef SF_SLEEP
969 974     } else if (sc_no == SYS_nanosleep) {
970 975       return inanosleep((const struct timespec *)arg1, (struct timespec *)arg2);
    976 +   } else if (sc_no == SYS_clock_nanosleep) {
    977 +     return iclock_nanosleep(arg1, arg2, (const struct timespec *)arg3,
    978 +                             (struct timespec *)arg4);
971 979   #endif // SF_SLEEP
972 980       // } else if (sc_no == SYS_getpid) {
973 981       //   assert(false);
```

ef005157cd97c9d3242b6d0a17908165fe1b74a6

Perf once fixed

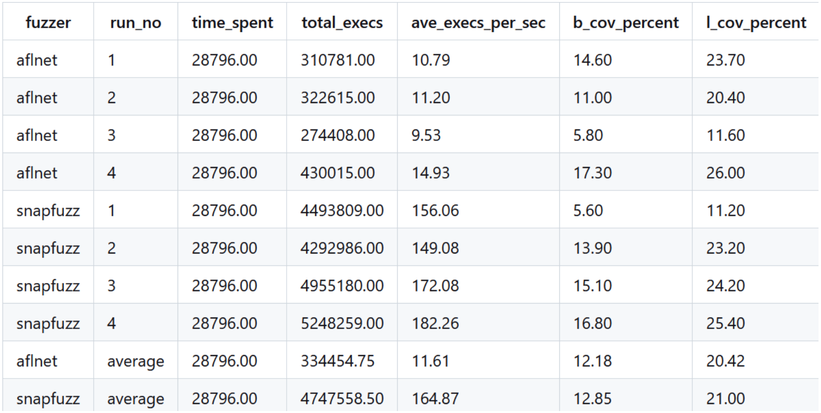


Dnsmasq



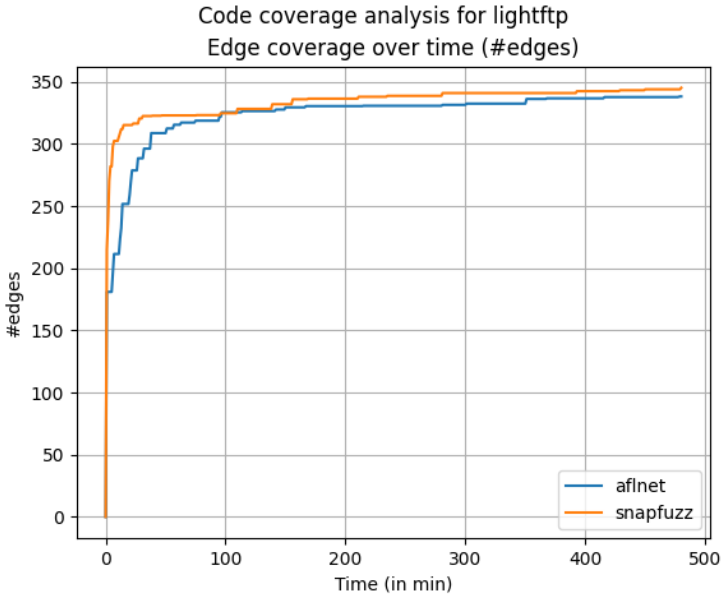
Code coverage analysis for dnsmasq
Edge coverage over time (#edges)
#edges
Time (in min)
0
50
100
150
200
250
0
100
200
300
400
500
aflnet
snapfuzz

Dnsmasq

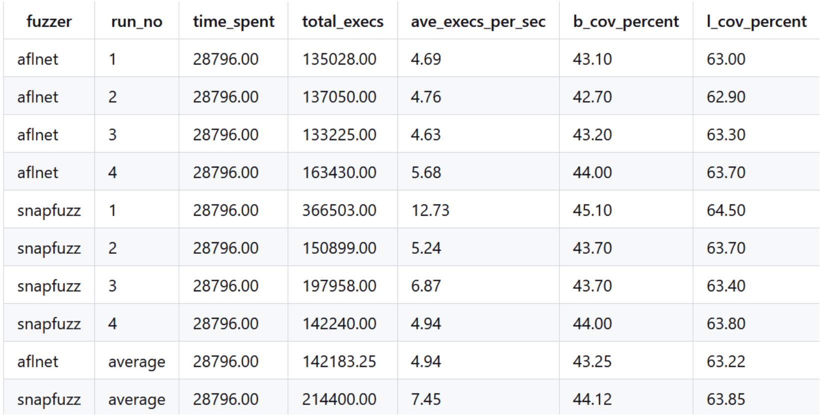


fuzzer	run_no	time_spent	total_execs	ave_execs_per_sec	b_cov_percent	l_cov_percent
aflnet	1	28796.00	310781.00	10.79	14.60	23.70
aflnet	2	28796.00	322615.00	11.20	11.00	20.40
aflnet	3	28796.00	274408.00	9.53	5.80	11.60
aflnet	4	28796.00	430015.00	14.93	17.30	26.00
snapfuzz	1	28796.00	4493809.00	156.06	5.60	11.20
snapfuzz	2	28796.00	4292986.00	149.08	13.90	23.20
snapfuzz	3	28796.00	4955180.00	172.08	15.10	24.20
snapfuzz	4	28796.00	5248259.00	182.26	16.80	25.40
aflnet	average	28796.00	334454.75	11.61	12.18	20.42
snapfuzz	average	28796.00	4747558.50	164.87	12.85	21.00

LightFTP

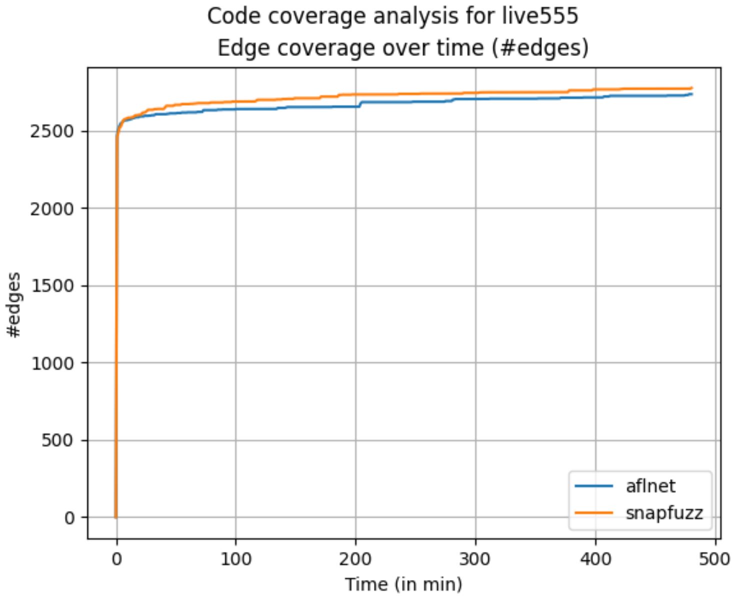

Code coverage analysis for lightftp
Edge coverage over time (#edges)
#edges
Time (in min)
0
50
100
150
200
250
300
350
0
100
200
300
400
500
aflnet
snapfuzz

LightFTP



fuzzer	run_no	time_spent	total_execs	ave_execs_per_sec	b_cov_percent	l_cov_percent
aflnet	1	28796.00	135028.00	4.69	43.10	63.00
aflnet	2	28796.00	137050.00	4.76	42.70	62.90
aflnet	3	28796.00	133225.00	4.63	43.20	63.30
aflnet	4	28796.00	163430.00	5.68	44.00	63.70
snapfuzz	1	28796.00	366503.00	12.73	45.10	64.50
snapfuzz	2	28796.00	150899.00	5.24	43.70	63.70
snapfuzz	3	28796.00	197958.00	6.87	43.70	63.40
snapfuzz	4	28796.00	142240.00	4.94	44.00	63.80
aflnet	average	28796.00	142183.25	4.94	43.25	63.22
snapfuzz	average	28796.00	214400.00	7.45	44.12	63.85

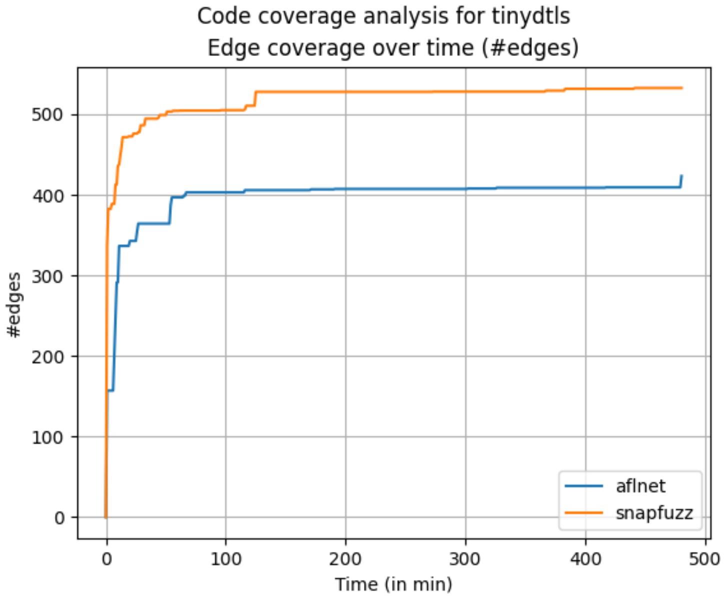
LIVE555


Code coverage analysis for live555
Edge coverage over time (#edges)
#edges
0
500
1000
1500
2000
2500
Time (in min)
0
100
200
300
400
500
aflnet
snapfuzz

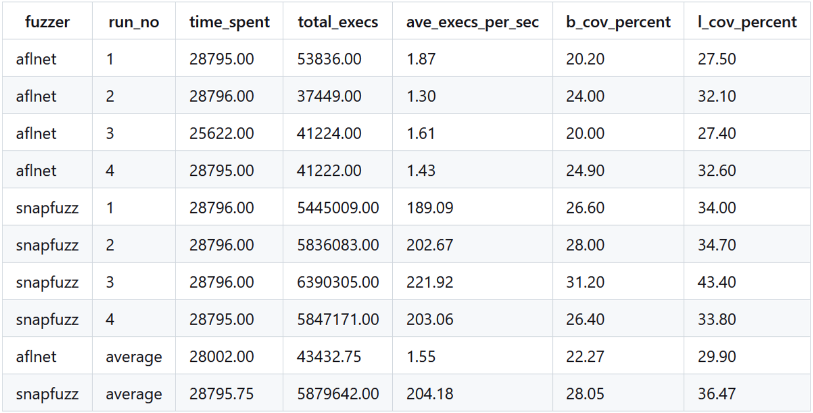
LIVE555

fuzzer	run_no	time_spent	total_execs	ave_execs_per_sec	b_cov_percent	l_cov_percent
aflnet	1	28796.00	302426.00	10.50	16.40	25.40
aflnet	2	28795.00	320816.00	11.14	16.60	25.90
aflnet	3	28796.00	332498.00	11.55	16.70	26.00
aflnet	4	28796.00	381030.00	13.23	16.70	26.00
snapfuzz	1	28796.00	1073514.00	37.28	16.90	26.10
snapfuzz	2	28796.00	1272257.00	44.18	17.20	26.30
snapfuzz	3	28795.00	1111905.00	38.61	16.70	25.50
snapfuzz	4	28796.00	1051298.00	36.51	16.70	25.90
aflnet	average	28795.75	334192.50	11.61	16.60	25.82
snapfuzz	average	28795.75	1127243.50	39.15	16.88	25.95

tinydtls


Code coverage analysis for tinydtls
Edge coverage over time (#edges)
#edges
Time (in min)
0
100
200
300
400
500
aflnet
snapfuzz

tinydtls



fuzzer	run_no	time_spent	total_execs	ave_execs_per_sec	b_cov_percent	l_cov_percent
aflnet	1	28795.00	53836.00	1.87	20.20	27.50
aflnet	2	28796.00	37449.00	1.30	24.00	32.10
aflnet	3	25622.00	41224.00	1.61	20.00	27.40
aflnet	4	28795.00	41222.00	1.43	24.90	32.60
snapfuzz	1	28796.00	5445009.00	189.09	26.60	34.00
snapfuzz	2	28796.00	5836083.00	202.67	28.00	34.70
snapfuzz	3	28796.00	6390305.00	221.92	31.20	43.40
snapfuzz	4	28795.00	5847171.00	203.06	26.40	33.80
aflnet	average	28002.00	43432.75	1.55	22.27	29.90
snapfuzz	average	28795.75	5879642.00	204.18	28.05	36.47

One's that didn't work

- Bftpd: `dup2(2)` suport missing
- forked-daapd: `statfs(2)` support missing
- DCMTK: Segfaults in startup code
- Kamailio: Deadlock between afl-fuzz and forkserver
- Exim: `wait4(-1, ...)` and `close(-1, ...)`, so forkserver exits.
- OpenSSH: Invalid pid sent to afl-fuzz
- OpenSSL: `close(3)` gives `EBADFD`, then forkserver exits.

Conclusions

- Snapfuzz isn't a drop in replacement for AFLNet.
- Snapfuzz is increadably fragile.
- It's not enough to be slightly faster.

Future work

- Add more syscalls
- Increase Debugabillity
    - Get PID of forkserver, and attach gdb before it crashes
- Fix snapfuzz bugs.

Source Code

- https://github.com/aDotInTheVoid/profuzzbench/tree/snapfuzz
- https://github.com/aDotInTheVoid/snapfuzz-omni
- https://github.com/aDotInTheVoid/pfb-analysis

Bibliography

[1]
Anastasios Andronidis and Cristian Cadar. 2022. SnapFuzz: High-throughput fuzzing of network applications. In *Proceedings of the 31st ACM SIGSOFT international symposium on software testing and analysis* (ISSTA 2022), Association for Computing Machinery, New York, NY, USA, 340–351. DOI:https://doi.org/10.1145/3533767.3534376

[2]
Paul-Antoine Arras, Anastasios Andronidis, Luís Pina, Karolis Mituzas, Qianyi Shu, Daniel Grumberg, and Cristian Cadar. 2022. SaBRe: Load-time selective binary rewriting. *International Journal on Software Tools for Technology Transfer* 24, 2 (April 2022), 205–223. DOI:https://doi.org/10.1007/s10009-021-00644-w

[3]
Roberto Natella and Van-Thuan Pham. 2021. ProFuzzBench: A benchmark for stateful protocol fuzzing. In *Proceedings of the 30th ACM SIGSOFT international symposium on software testing and analysis* (ISSTA 2021), Association for Computing Machinery, New York, NY, USA, 662–665. DOI:https://doi.org/10.1145/3460319.3469077

[4]
Van-Thuan Pham, Marcel Böhme, and Abhik Roychoudhury. 2020. AFLNET: A greybox fuzzer for network protocols. In *2020 IEEE 13th international conference on software testing, validation and verification (ICST)*, 460–465. DOI:https://doi.org/10.1109/ICST46399.2020.00062

[5]
Michal Zalewski. American fuzzy lop. Retrieved from https://lcamtuf.coredump.cx/afl/