AFL-based fuzzing for Java with Kelinci

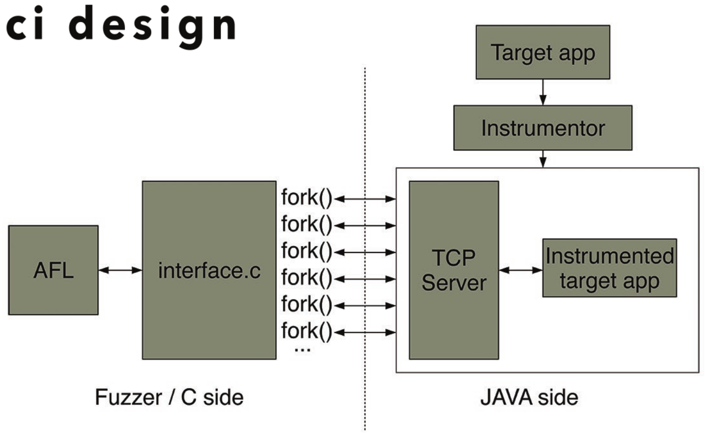
Rody Kersten[1], Kasper Luckow[1], Corina Păsăreanu[1,2]
[1]Carnegie Mellon University Silicon Valley, [2]NASA Ames Research Center

Grey-box fuzzing with AFL

- Light-weight instrumentation to differentiate program behaviors:
  ```
  Mem.mem[id^Mem.prev_location]++;
  Mem.prev_location = id >> 1;
  ```
- Prioritize inputs that trigger new behaviors
- AFL has found numerous vulnerabilities (several of the Stage fright vulnerabilities, Shellshock related vulnerabilities, DoS vulnerabilities in BIND), as well as numerous bugs in (security-critical) applications and libraries such as OpenSSL, OpenSSH, GnuTLS, GnuPG, PHP, Apache, IJG jpeg, libjpeg-turbo and many more.
- Unfortunately, no real support for Java

- Run AFL on Java programs
- No modifications to AFL
- Easily parallelizable
- Found bugs in JDK 6-9 and Apache Commons Imaging

Kelinci design



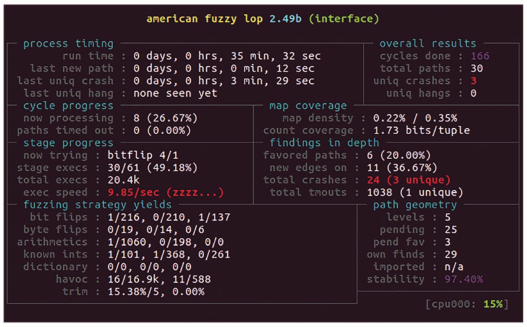
- C program that mimics AFL-instrumented binary
- Java instrumentor based on ASM
- Sides communicate over TCP connection
- AFL is unaware it is fuzzing a Java program

Interaction between C and Java side

For a given input file generated by the fuzzer, the interaction between the C and Java sides is as follows:

1. interface.c receives a fork request from AFL.
2. One of the forked interface.c processes loads the provided input file, the other keeps running the fork server.
3. The former interface.c process sends the provided input file over a TCP connection to the Kelinci server.
4. The Kelinci server receives the incoming request and enqueues it.
5. The Kelinci server processes the request by writing the incoming input file to disk, starting a new thread in which the main method of the target application is called on the provided input and monitoring it. If the thread throws an exception that escapes main, it is considered a bug. If the thread does not terminate within a given time-out, it is considered a hang.
6. The Kelinci server communicates the results back to the C side. The shared memory bitmap is sent over the TCP connection, as well as the status (OK, ERROR or TIMEOUT).
7. On the C side, the received bitmap is written to shared memory. Depending on the received status, the program exits normally, aborts or keeps looping until AFL hits its time-out.

Found bugs in JDK 6-9 and Apache Commons Imaging



```
                    american fuzzy lop 2.49b (interface)

┌─ process timing ─────────────────────────────────────┬─ overall results ────┐
│        run time : 0 days, 0 hrs, 35 min, 32 sec      │  cycles done : 166   │
│   last new path : 0 days, 0 hrs, 0 min, 12 sec       │  total paths : 30    │
│ last uniq crash : 0 days, 0 hrs, 3 min, 29 sec       │ uniq crashes : 3     │
│  last uniq hang : none seen yet                      │   uniq hangs : 0     │
├─ cycle progress ────────────────────┬─ map coverage ─┴──────────────────────┤
│  now processing : 8 (26.67%)        │    map density : 0.22% / 0.35%        │
│ paths timed out : 0 (0.00%)         │ count coverage : 1.73 bits/tuple      │
├─ stage progress ────────────────────┼─ findings in depth ───────────────────┤
│  now trying : bitflip 4/1           │ favored paths : 6 (20.00%)            │
│ stage execs : 30/61 (49.18%)        │  new edges on : 11 (36.67%)           │
│ total execs : 20.4k                 │ total crashes : 24 (3 unique)         │
│  exec speed : 9.85/sec (zzzz...)    │  total tmouts : 1038 (1 unique)       │
├─ fuzzing strategy yields ───────────┴───────────────┬─ path geometry ────────┤
│   bit flips : 1/216, 0/210, 1/137                   │    levels : 5          │
│  byte flips : 0/19, 0/14, 0/6                       │   pending : 25         │
│ arithmetics : 1/1060, 0/198, 0/0                    │  pend fav : 3          │
│  known ints : 1/101, 1/368, 0/261                   │ own finds : 29         │
│  dictionary : 0/0, 0/0, 0/0                         │  imported : n/a        │
│       havoc : 16/16.9k, 11/588                      │ stability : 97.40%     │
│        trim : 15.38%/5, 0.00%                       ├────────────────────────┘
└─────────────────────────────────────────────────────┘          [cpu000: 15%]
```

- Used on JPEG parsers in JDK 8 and Apache Commons Imaging
- Behavior similar to AFL: identifies JPEG header
- ~500 times slower than AFL
- Finds a similar issue in both
  - -If specified segment size is 0 or 1, throws RuntimeException
  - -Found after about half an hour

https://github.com/isstac/kelinci