The functionality of the code depends on a compiled linux kernel, an intraid fuzzcan, and a .raw image, and the afl fuzz. The confs section of the github will consist of a series of configuration tools which will set what is enabled or disabled when the linux kernel is compiling, the things in which it is enabled will be processing tools, memory management, I/O scheduling, and network configurations. The actual fuzzing tool used is afl-fuzz, which according to Marek Majkowski's *A gentle introduction to Linux Kernel fuzzing,* a "fuzzer picks the most promising test case" then "mutates the test into a large number of new test cases", next it "runs the mutated test cases, and reports back code coverage", and finally it "computes a score from the reported coverage, and uses it to prioritize the interesting mutated tests and remove the redundant ones." There is also a set of C code most of which is to enable kcov. Kcov enables the fuzzer to perform fuzzing against the kernel and save any unexpected output. The other main purpose being to construct socketcans which are to be fuzzed. There also exists a makefile, which will create a binary file called fuzzcan which will consist of the set of C code mentioned earlier. Afterwards the file will then be copied over to a directory called rootfs/bin/, which then executes a shell script called mkrootfs.sh. The shell script will create a bunch of directories to rootfs, which will then be compressed to a.gz file. The rootfs path is a template for a initrd which accouriding to M. Jones's *Linux initial RAM disk (initrd) overview* a "initial RAM disk (initrd) is an initial root file system that is mounted prior to when the real root file system is available." The .raw image is where if a crash or hang happens a file would be created in the filesystem that would save the output of the crash or hang.

https://developer.ibm.com/articles/l-initrd/
https://blog.cloudflare.com/a-gentle-introduction-to-linux-kernel-fuzzing/