Fuzzing OpenSSL

OpenSSL can use either LibFuzzer or AFL to do fuzzing.

LibFuzzer

How to fuzz OpenSSL with libfuzzer, starting from a vanilla+OpenSSH server Ubuntu install.

With `clang` from a package manager

Install `clang`, which ships with `libfuzzer` since version 6.0:

```
sudo apt-get install clang
```

Configure `openssl` for fuzzing. For now, you'll still need to pass in the path to the `libFuzzer` library file while configuring; this is represented as `$PATH_TO_LIBFUZZER` below. A typical value would be `/usr/lib/llvm-7/lib/clang/7.0.1/lib/linux/libclang_rt.fuzzer-x86_64.a`.

```
CC=clang ./config enable-fuzz-libfuzzer \
        --with-fuzzer-lib=$PATH_TO_LIBFUZZER \
        -DPEDANTIC enable-asan enable-ubsan no-shared \
        -DFUZZING_BUILD_MODE_UNSAFE_FOR_PRODUCTION \
        -fsanitize=fuzzer-no-link \
        enable-ec_nistp_64_gcc_128 -fno-sanitize=alignment \
        enable-weak-ssl-ciphers enable-rc5 enable-md2 \
        enable-ssl3 enable-ssl3-method enable-nextprotoneg \
        --debug
```

Compile:

```
sudo apt-get install make
make clean
LDCMD=clang++ make -j4
```

Finally, perform the actual fuzzing:

```
fuzz/helper.py $FUZZER
```

where $FUZZER is one of the executables in `fuzz/`. It will run until you stop it.

If you get a crash, you should find a corresponding input file in `fuzz/corpora/$FUZZER-crash/`.

With `clang` from source/pre-built binaries

You may also wish to use a pre-built binary from the LLVM Download site, or to build `clang` from source. After adding `clang` to your path and locating the `libfuzzer` library file, the procedure for configuring fuzzing is the same, except that you also need to specify a `--with-fuzzer-include` option, which should be the parent directory of the prebuilt fuzzer library. This is represented as `$PATH_TO_LIBFUZZER_DIR` below.

```
CC=clang ./config enable-fuzz-libfuzzer \
        --with-fuzzer-include=$PATH_TO_LIBFUZZER_DIR \
        --with-fuzzer-lib=$PATH_TO_LIBFUZZER \
        -DPEDANTIC enable-asan enable-ubsan no-shared \
        -DFUZZING_BUILD_MODE_UNSAFE_FOR_PRODUCTION \
        -fsanitize=fuzzer-no-link \
        enable-ec_nistp_64_gcc_128 -fno-sanitize=alignment \
        enable-weak-ssl-ciphers enable-rc5 enable-md2 \
        enable-ssl3 enable-ssl3-method enable-nextprotoneg \
        --debug
```

AFL

This is an alternative to using LibFuzzer.

Configure for fuzzing:

```
sudo apt-get install afl-clang
CC=afl-clang-fast ./config enable-fuzz-afl no-shared no-module \
    -DPEDANTIC enable-tls1_3 enable-weak-ssl-ciphers enable-rc5 \
    enable-md2 enable-ssl3 enable-ssl3-method enable-nextprotoneg \
    enable-ec_nistp_64_gcc_128 -fno-sanitize=alignment \
    --debug
make clean
make
```

The following options can also be enabled: enable-asan, enable-ubsan, enable-msan

Run one of the fuzzers:

```
afl-fuzz -i fuzz/corpora/$FUZZER -o fuzz/corpora/$FUZZER/out fuzz/$FUZZER
```

Where $FUZZER is one of the executables in `fuzz/`.

Reproducing issues

If a fuzzer generates a reproducible error, you can reproduce the problem using the fuzz/*-test binaries and the file generated by the fuzzer. They binaries don't need to be built for fuzzing, there is no need to set CC or the call config with enable-fuzz-* or -fsanitize-coverage, but some of the other options above might be needed. For instance the enable-asan or enable-ubsan option might be useful to show you when the problem happens. For the client and server fuzzer it might be needed to use -DFUZZING_BUILD_MODE_UNSAFE_FOR_PRODUCTION to reproduce the generated random numbers.

To reproduce the crash you can run:

```
fuzz/$FUZZER-test $file
```

To do all the tests of a specific fuzzer such as asn1 you can run

```
fuzz/asn1-test fuzz/corpora/asn1
```

or make test TESTS=fuzz_test_asn1

To run several fuzz tests you can use for instance:

```
make test TESTS='test_fuzz_cmp test_fuzz_cms'
```

To run all fuzz tests you can use:

```
make test TESTS='test_fuzz_*'
```

Random numbers

The client and server fuzzer normally generate random numbers as part of the TLS connection setup. This results in the coverage of the fuzzing corpus changing depending on the random numbers. This also has an effect for coverage of the rest of the test suite and you see the coverage change for each commit even when no code has been modified.

Since we want to maximize the coverage of the fuzzing corpus, the client and server fuzzer will use predictable numbers instead of the random numbers. This is controlled by the FUZZING_BUILD_MODE_UNSAFE_FOR_PRODUCTION define.

The coverage depends on the way the numbers are generated. We don't disable any check of hashes, but the corpus has the correct hash in it for the random numbers that were generated. For instance the client fuzzer will always generate the same client hello with the same random number in it, and so the server, as emulated by the file, can be generated for that client hello.

Coverage changes

Since the corpus depends on the default behaviour of the client and the server, changes in what they send by default will have an impact on the coverage. The corpus will need to be updated in that case.

Updating the corpus

The client and server corpus is generated with multiple config options:

- The options as documented above
- Without enable-ec_nistp_64_gcc_128 and without –debug
- With no-asm
- Using 32 bit
- A default config, plus options needed to generate the fuzzer.

The libfuzzer merge option is used to add the additional coverage from each config to the minimal set.

Minimizing the corpus

When you have gathered corpus data from more than one fuzzer run or for any other reason want to minimize the data in some corpus subdirectory `fuzz/corpora/DIR` this can be done as follows:

```
mkdir fuzz/corpora/NEWDIR
fuzz/$FUZZER -merge=1 fuzz/corpora/NEWDIR fuzz/corpora/DIR
```