LibFuzzer Tutorial

K. Serebryany. libfuzzer a library for coverage-guided fuzz testing.
“https://llvm.org/docs/LibFuzzer.html”, 2018.

Overview : LibFuzzer In Fuzzing

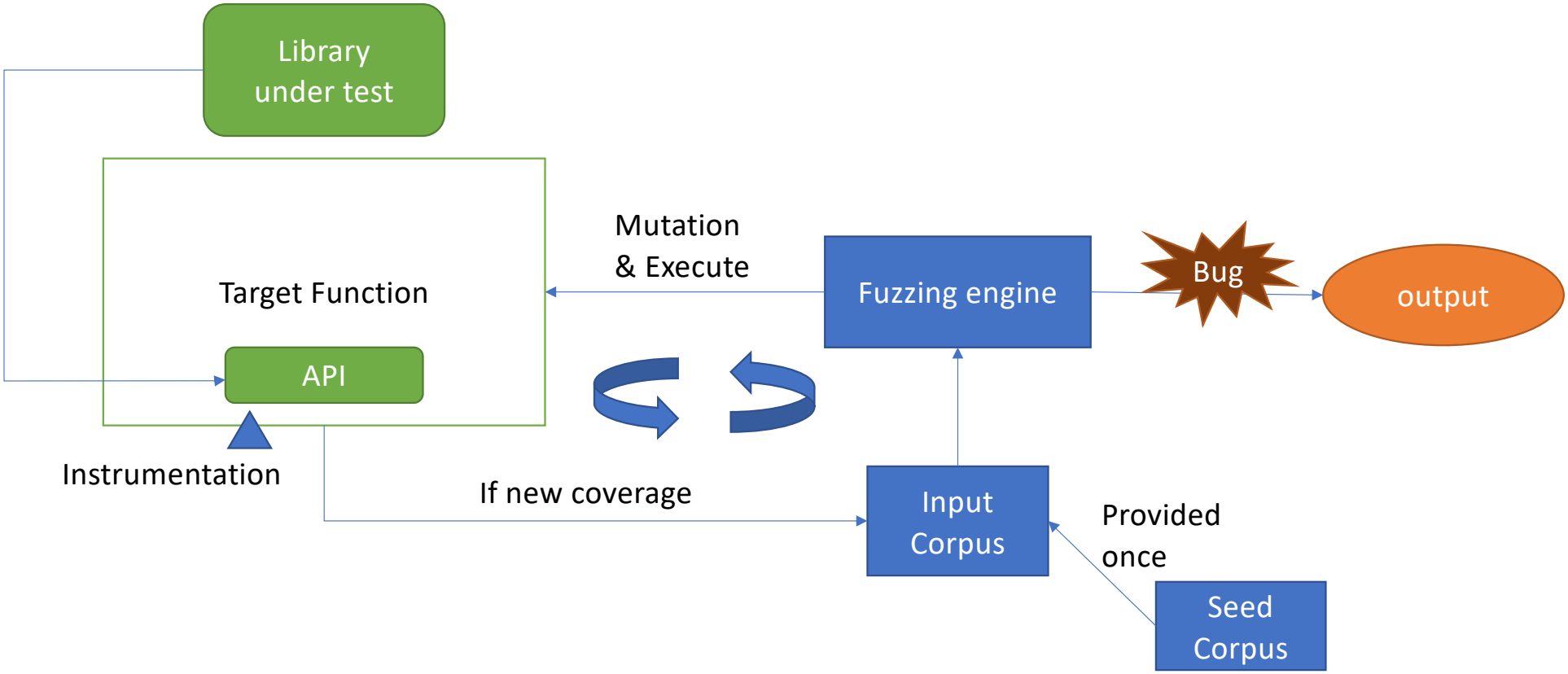
- Fuzzing : An automated testing technique that covers numerous boundary cases using anomalous data (from files, network protocols, API calls, and other targets) as input to discover vulnerabilities
- Mutation based fuzzing
  - Introducing small changes to existing inputs that may exercise new behavior
- Grey-box fuzzer
  - Libfuzzer uses compile-time instrumentation of the source code to to reach deeper into the program.

What is LibFuzzer?

LibFuzzer is in-process, coverage-guided, evolutionary fuzzing engine

- **In-process**
  - LibFuzzer do not launch a new process for every test case. Libfuzzer do fuzzing in a single process. It does the mutations and feeds the inputs into the API and test it.
- **Coverage-guided**
  - LibFuzzer measure code coverage (SanitizerCoverage instrumentation) for every input, and accumulate test cases that increase overall coverage
- **Evolutionary fuzzing**
  - LibFuzzer generates mutations on the corpus of input data in order to maximize the code coverage

LibFuzzer Flow Diagram



* Any kind of abort, exit, crash, assert, timeout is considered as a bug

Where do we use Libfuzzer?

- Libfuzzer performs guided-mutation fuzzing for libraries or APIs ("unit test-like fuzzing"). LibFuzzer makes it possible to fuzz individual components of application.
- Libfuzzer is a good choice for testing libraries that have relatively small inputs, each input takes < 10ms to run.
  (e.g. regular expression matchers, text or binary format parsers, compression, network, crypto)

LibFuzzer steps

1. Install Fuzzing engine and library under test
2. Provide Target function
3. Build Fuzzer
4. Run Fuzzer
5. Get output of Fuzzer

Fuzzing engine

- Fuzzing engine is the tool that finds inputs that are interesting enough and feeds them into fuzz target
- Libfuzzer installation
  - Clang is a compiler front end for the C language family(C, C++, Objective-C/C++, OpenCL, CUDA, and RenderScript).
  - Clang contains LibFuzzer. Install Clang. Then, no extra installation is necessary.

```
CLANG_VERSION=#Starting from version 6.0.0
CLANG_DIR=clang+llvm-$CLANG_VERSION-#depend on OS
curl https://releases.llvm.org/$CLANG_VERSION/$CLANG_DIR.tar.xz
#Copy Clang file to user local
cp -rf  $CLANG_DIR/bin/* /usr/local/bin
cp -rf $CLANG_DIR/lib/clang /usr/local/lib
```

Definition of Target function

- Definition: a function that has the fixed signature and does something interesting with its arguments.

```
// target.cc
extern "C" int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size) {
    DoSomethingInterestingWithMyAPI(Data, Size);
    return 0; // Non-zero return values are reserved for future use.
}
```

- Target function accepts array of bytes.
- Target function feed accepted data into the API under test

Role of Target function in Libfuzzer

- **Where do we use Target function?**
  - The first step in using libFuzzer on a library is to implement Target function
  - The fuzzing engine will execute the Target function many times with different inputs in the same process until a bug is found

  - This target function does not depend on LibFuzzer in any way and so it is possible to use it with other fuzzing engines(e.g. AFL and/ or Radamsa)

What API should be targeted?

- **What API should be targeted?**

  - Testing API that have relatively small inputs, each input takes < 10ms to run.
  - The library code is not expected to crash on inputs.
  - It must be as deterministic as possible. Non-determinism (e.g. random decisions not based on the input bytes) will make fuzzing inefficient
  - Ideally, it should not modify any global state
  - Usually, the narrower the target the better.

Recommendation on Target function

- **What should be done on Target function?**

  - Typecast or adjust generated random data to fit on format of target API.

  - Set prerequisite environment to run target API(i.e. initiation in network APIs, limitation on maximum size of data)

  - Customize fuzzing engine

Tested libraries using Libfuzzer

4 Tested libraries using Libfuzzer (Google test suite)

- **Json(json-2017-02-12) - Text or binary format parser**
- **Libpng(libpng-1.2.56) - Compression** : The free reference library available as standard (ANSI) C source code and used by many PNG-supporting applications.
- **SQLite(sqlite-2016-11-14) - Query execution** : SQLite is a C library that implemnts a SQL database engine which is the most used database engine in the world
- **Libssh(libssh-2017-1272)- Network / crypto** : C library that enables to write a program that uses the SSH protocol.

Tested libraries using Libfuzzer

- **Test suite analysis**

1. What library has tested?
   - Which API was tested?

2. How to make Target function?
   - In what way does the library have been tested?
   - How can we improve efficiency?

3. How to build and run the fuzzer
   - Configuration

4. What kind of error was founded?
   - The result of testing

JSON library (json-2017-02-12)

- **What does JSON library for?**
  - JSON Library is JSON parser which has design goals of intuitive syntax, trivial integration. This library is seriously unit-tested
- **Which API was tested?**

1. JSON Serialization / Deserialization
   - dump(), parse()
2. Binary format (CBOR & MessagePack) Serialization / Deserialization
   - to_cbor(), from_cbor()
   - to_msgpack(), from_msgpack()

API Under test : Serialization / Deserialization

- parse(), dump() function

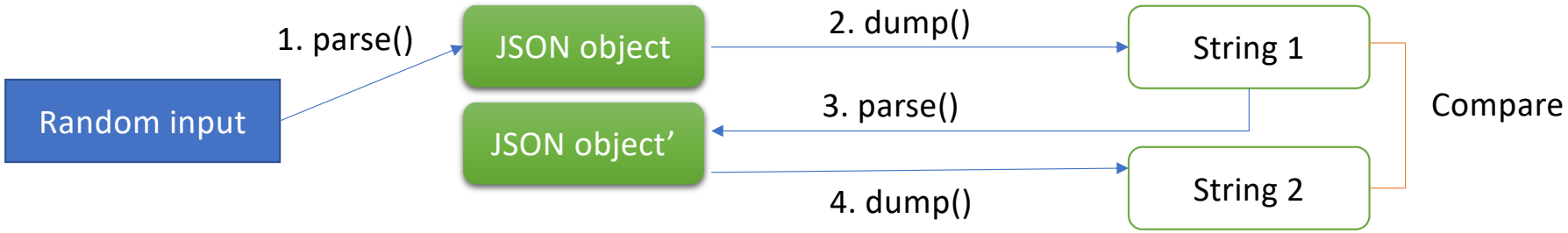
```
01 // create object from string literal
02 json j = "{ \"happy\": true, \"pi\": 3.141 }"_json;
03 // or parse explicitly
04 auto j2 = json::parse("{ \"happy\": true, \"pi\": 3.141 }");
05 // explicit conversion to string
06 std::string s = j.dump(); // {\"happy\":true,\"pi\":3.141}
```

API Under test : Binary format Serialization / Deserialization

```
01 json j = R"({"compact": true, "schema": 0})"_json; // create a JSON value
02 // CBOR format
03 std::vector<uint8_t> v_cbor = json::to_cbor(j); // serialize to CBOR
04 json j_from_cbor = json::from_cbor(v_cbor); // roundtrip to json
05 // MessagePack format
06 std::vector<uint8_t> v_msgpack = json::to_msgpack(j); // serialize to MessagePack
07 json j_from_msgpack = json::from_msgpack(v_msgpack); // roundtrip json
```

In what way does the library have been tested?

- **Test serialization and deserialization is required.**
  - If parse errors occur, they throw std::invalid_argument
  - Since Any kind of abort, exit, crash, assert, timeout is considered as a bug, if random input is provided, parsing error will occur.
- **In what way does the library have been tested?**
  - Testing has done in round trip way : dump() and parse() function were executed twice with given input. Then, compare two serialized strings



Target function generation for JSON

Required source

1. The single required source, file json.hpp.

```
#include "json.hpp"

using json = nlohmann::json;
```

2. Target function source code

Fuzzing procedure

When the test driver is set, the next step is to build the driver and run fuzzer. Therefore, we are going look through the steps below in details.

1. How do we build?
   - Build library
   - Build target function?
2. How do we run fuzzer?
3. What does the result output looks like?

Build

```
clang -g -O1 -fsanitize=fuzzer                          mytarget.c # Builds the fuzz target w/o sanitizers
clang -g -O1 -fsanitize=fuzzer,address                  mytarget.c # Builds the fuzz target with ASAN
clang -g -O1 -fsanitize=fuzzer,signed-integer-overflow mytarget.c # Builds the fuzz target with a part of UBSAN
clang -g -O1 -fsanitize=fuzzer,memory                   mytarget.c # Builds the fuzz target with MSAN
```

Flags

- **-fsanitize=fuzzer(required)**
  provides in-process coverage information to libFuzzer and links with the libFuzzer
- **-g (optional)** : Generate debug information
- **-O0, -O1, -O2 … (optional)** : Specify which optimization level to use
  O0 no optimization
  O1 Somewhere between -00 and -02
  O2 Moderate level of optimization which enables most optimizations

Build

```
clang -g -O1 -fsanitize=fuzzer                      mytarget.c # Builds the fuzz target w/o sanitizers
clang -g -O1 -fsanitize=fuzzer,address              mytarget.c # Builds the fuzz target with ASAN
clang -g -O1 -fsanitize=fuzzer,signed-integer-overflow mytarget.c # Builds the fuzz target with a part of UBSAN
clang -g -O1 -fsanitize=fuzzer,memory               mytarget.c # Builds the fuzz target with MSAN
```

- AddressSanitizer is a memory error detector.

- Combine LibFuzzer with sanitizer.
  - AddressSanitizer(ASAN)
  - UndefinedBehaviorSanitizer(UBSAN)
  - MemorySanitizer(MSAN) //experimental for now, not available with ASAN

Build

- **Compile source code using clang flags**
  [CFLAGS] : -g -O2 -fsanitize=fuzzer–no-link -fno-omit-frame-pointer -gline-tables-only –fsanitize=address –fsanitize-address-use-after-scope
- **-fsanitize=fuzzer-no-link**
  request just the instrumentation without linking
  -fno-omit-frame-pointer
  Get nicer stack straces in error messages while using address sanitizer
  -gline-tables-only
  Enables debug info, makes the error messages easier to read
  -fsanitize=address
  Enables address sanitizer
  -fsanitize-address-use-after-scope
  Enables detecting memory usage after scope
- **Target function build**
  clang++ [CFLAGS] –ldl –pthread target.cc sqlite3.o ossfuzz.o –fsanitize=fuzzer

Running fuzzer

- Run the fuzzer
  ./my_fuzzer [Options] Input corpus/ Seed corpus/
- By default, the fuzzing process will continue indefinitely – at least until a bug is found.

  - What is input & seed corpus?
  - What options are available?

Input Corpus & Seed Corpus

- **Input Corpus**
  - As the fuzzer discovers new interesting test cases(e.g. test cases that trigger coverage of new paths through the code under test), those test cases will be added to the corpus directory
  - Generated corpus can also act as a sanity/regression check
- **Seed Corpus**
  - LibFuzzer will work without any initial seed but will be less efficient. With seeds, user can provide the input which has higher coverage compare to empty string. (e.g. variety of different small PNG/JPG/GIF/HTML files)
  - The size of the inputs that libFuzzer tries is limited by the size of the largest file in the seed corpus. It can be change with –max_len flag

Running Options : Limit the fuzzer

- -runs
  : Number of individual test runs, -1(the default) to run indefinitely
- -max_len
  : Maximum length of a test input. If 0(the default), libFuzzer tries to guess a good value based on the corpus (and reports it).
- -timeout
  : Timeout in seconds, default 1200. If an input takes longer than this timeout, the process is treated as a failure case
- -max_total_time
  : If positive, indicates the maximum total time in seconds to run the fuzzer

Running Options : Improve efficiency

- -jobs=[JOBS]
  Run fuzzer for the number of [JOBS]
  (e.g. If 8 is assigned, run fuzzer until 8 errors are found or reach limitation of fuzzer)
- -workers=[WORKERS]
  Run fuzzer with multiple processes

  for example, if flag is provided like –jobs=30 –workers=5 , each process will report approximately 6 errors.

Information of execution of fuzzer

- During operation, the fuzzer prints information to stderr
- The output has below part
  1. Early parts of output show information about the fuzzer options and configuration.
  2. Further output lines have the form of an event code and statistics for generated inputs.
  3. Debug message / Error message
  4. The information on input that leads bug

Fuzzer options and configuration (1/3)

- Information of Fuzzer options and configuration is printed with heading
  INFO: #
  INFO : Seed: *random#*
  - The fuzzer start with random seed. Rerun it with –seed= *random#* to get the same result
  INFO : -max_len info
  - libfuzzer will not generate inputs larger than 4096(default) bytes
  INFO : input corpus info
  - If a corpus is not provided, starting from an empty corpus
  INFO : seed corpus info
  - information of provided seeds and input corpus(if inputs exist in the directory)

Fuzzer options and configuration (1/3)

Example: Early part of Fuzzer output

INFO: Seed: 3271227540
INFO: Loaded 1 modules (8 inline 8-bit counters): 8 [0x7a6e90, 0x7a6e98),
INFO: Loaded 1 PC tables (8 PCs): 8 [0x56c628,0x56c6a8),
INFO: 0 files found in CORPUS_DIR/
INFO: 1 files found in seeds/
INFO: -max_len is not provided; libFuzzer will not generate inputs larger than 4096 bytes
INFO: seed corpus: files: 1 min: 3b max: 3b total: 3b rss: 26Mb

Event code and statistics for generated inputs(2/3)

- Information of generated inputs is printed with the form of

  [#the number of input] [Event code] [cov: #] [ft : #] [corp : #/size in bytes] [lim:#] [exec/s:#] [rss:#] ([L:] [MS:], ***for NEW and REDUCE event*** **)**

  For example,
  #2 INITED cov: 3 ft: 4 corp: 1/3b exec/s: 0 rss: 26Mb
  #3 NEW cov: 4 ft: 5 corp: 2/5b lim: 4 exec/s: 0 rss: 26Mb L: 2/3 MS: 1 EraseBytes-
  …

Event code and statistics for generated inputs(2/3)

- Event code

  - READ
  : The fuzzer has read in all of the provided input samples from the corpus directories
  - INITED
  : The fuzzer has completed initialization, which includes running each of the initial input samples through the code under test.
  - NEW
  : The fuzzer has created a test input that covers new areas of the code under test. This input will be saved to the input corpus directory.
  - REDUCE
  : The fuzzer has found a better(smaller) inputs that triggers previously discovered features

Event code and statistics for generated inputs(2/3)

- Event code

  - pulse
  : The fuzzer has generated 2^n inputs (generated periodically to reassure the user that the fuzzer is still working)
  - DONE
  : The fuzzer has completed operation (iteration limit *–runs* or time limit *–max_total_time*)
  - RELOAD
  : The fuzzer is performing a periodic reload of inputs from the corpus directory (for parallel Fuzzing)

Event code and statistics for generated inputs(2/3)

#32 NEW cov: 4 ft: 5 corp: 2/5b lim: 4 exec/s: 0 rss: 26Mb L: 2/3 MS: 1 EraseBytes-

- cov:
  - Total number of code blocks or edges covered by executing the current corpus
- ft:
  - LibFuzzer use different signals to evaluate the code coverage: specific coverage can be enabled with build / running options.
  edge coverage, edge counters, value profiles, indirect caller/callee pairs, etc.
  These signals combined are called features.
- corp:
  - Number of entries in the current in-memory test corpus and its size in bytes

Libfuzzer has tried at least 32 inputs, discovered 2 inputs of 5bytes total that together cover 4 coverage points(basic blocks in the code).

Event code and statistics for generated inputs(2/3)

#32 NEW cov: 4 ft: 5 corp: 2/5b lim: 4 exec/s: 0 rss: 26Mb L: 2/3 MS: 1 EraseBytes-

- lim:
  - Current limit on the length of new entries in the corpus. Increases over time until the max length(-max_len)is reached
- exec/s:
  - Number of fuzzer iterations per second.
- rss:
  - Current memory consumption

Event code and statistics for generated inputs(2/3)

For NEW, REDUCE events

- L : size of the new input in bytes
- MS : <n> <operations>
  Count and list of the mutation operations used to generate the inputs

TABLE I
SET OF LIBFUZZER MUTATION OPERATORS

Mutator	Description
EraseBytes	Reduce size by removing a random byte
InsertByte	Increase size by one random byte
InsertRepeated Bytes	Increase size by adding at least 3 random bytes
ChangeBit	Flip a Random bit
ChangeByte	Replace byte with random one
ShuffleBytes	Randomly rearrange input bytes
ChangeASCII Integer	Find ASCII integer in data, perform random math ops and overwrite into input.
ChangeBinary Integer	Find Binary integer in data, perform random math ops and overwrite into input
CopyPart	Return part of the input
CrossOver	Recombine with random part of corpus/self
AddWordPersist AutoDict	Replace part of input with one that previously increased coverage (entire run)
AddWordTemp AutoDict	Replace part of the input with one that recently increased coverage
AddWord FromTORC	Replace part of input with a recently performed comparison

The information on input that leads bug (3/3)

- The last input triggers error while running fuzzer. The information of the input and result of running fuzzer is printed
  - Information of mutation of input (mutation operation, base unit)
  - Data of the input
  - Result of running fuzzer
- The test unit is written in the file ./crash-*

JSON Build & Running Options

- **Build options**
  clang++ [CFLAGS] -std=c++11 -I BUILD/src json_target.cpp
  fsanitize=fuzzer -o json_fuzzer
- **Running options**
  ./json_fuzzer -jobs=8 -workers=8 $CORPUS seeds
- **One seed file is provided**
  - long number is provided as seed
  1000000010E5

Error in JSON

- The test suite found assertion failure

  json-2017-02-12-libfuzzer:
  BUILD/test/src/fuzzer-parse_json.cpp:50: int
  LLVMFuzzerTestOneInput(const uint8_t *, size_t):
  Assertion `s1 == s2' failed.
  ==...==
  ERROR: libFuzzer: deadly signal

Libpng library (libpng-1.2.56)

- **What does Libpng library for?**
  - Libpng is the official PNG reference library
  - PNG(portable network graphics) is a raster-graphics(dot matrix data structure) file format that supports lossless data compression.
- **What API was under test?**
  Reading image data : Libpng will read in all the image data and put it in the memory area supplied.

  - png_read_rows(png_ptr, row_pointers, NULL, number_of_rows);

In what way does the library have been tested?

There are several features that should be considered to test Libpng

1. There are two main structures important to libpng

png_struct and png_info need to be allocated and initialized by png_create_read_struct() and png_create_info_struct()

png_struct: It holds all of the basic information associated with the PNG image.

png_info: It is used to indicate what its state will be after all of the user-requested transformations are performed.

In what way does the library have been tested?

There are several features that should be considered to test Libpng

2. Libpng reads PNG image data from file using fread() internally.

- To use libfuzzer for testing, data should be read from memory rather than directly from a file. The function below is provided.

```
png_set_read_fn(png_structp read_ptr,
                voidp       read_io_ptr,
                png_rw_ptr  read_data_fn);
```

read_ptr : png_ptr
read_io_ptr (defined by user) : void pointer to the data source object
read_data_fn (defined by user) : a pointer to a function that will handle copying the data from the data source object into a given byte buffer

In what way does the library have been tested?

3. PNG file has a particular format

PNG file is PNG datastream stored as a file. PNG datastream is a result of encoding a PNG image. A PNG datastream consist of a PNG signature followed by a sequence of chunks

- PNG signature : A PNG file starts with an 8-byte signature.
(89 50 4E 47 0D 0A 1A 0A hex).

A. function png_sig_cmp(), will return 0 if the bytes match the corresponding bytes of the PNG signature.

B. Providing seeds that have PNG format improve efficiency.

In what way does the library have been tested?

- Chunk :PNG datastream consist of PNG signature followed by a sequence of chunks.

```
{  // chunk structure
        Length (4 byte),
        Chunk Type (4 byte), //18 chunk types are defined
        Chunk Data (length byte),
        CRC (4byte)
}
```

Length : Every chunk has a length, and thus a mutation that increases the size of a chunk also needs to change the stored length

Chunk Type : Each chunk has a chunk type which specifies its function

Chunk data : The data bytes appropriate to the chunk type, if any.

CRC : Every chunk contains a CRC checksum. CRC can be disabled for mutation

Target function generation for Libpng

Required source

1. **Set required header file and install libpng(INSTALL file)**
   #include “png.h” : defines all of the libpng datatypes
   png.c : location for general purpose libpng functions
2. **Structure-Aware fuzzing can be done with Custom Mutator**
3. **Target function source code**
4. **Custom mutator header file**

Is there any way to improve efficiency?

1. Provide PNG file as seed.
2. Structure-aware fuzzing

Structure-Aware Fuzzing with libFuzzer

- Generation-based fuzzers(Such as csmith, Peach) usually target a single input type, generating inputs according to a pre-defined grammar.
- However, Coverage-guided mutation-based fuzzers, such as libFuzzer or AFL, are not restricted to a single input type and do not require grammar definitions. Thus, mutation-based fuzzers are generally easier to set up and use. But the lack of an input grammar can also result result in inefficient fuzzing.
- With additional effort, libfuzzer can be turned into a grammar-aware(i.e. structure-aware) fuzzing engine for a specific input type.

Structure-Aware Fuzzing with libFuzzer

- **Custom mutators is a user-defined function with a fixed signature that does the following**

  1. Parses the input data according to the specified language grammar
  2. Mutates the parsed representation of the input. The custom mutator may request LibFuzzer to mutate some part of the raw data via the function LLVMFuzzerMutate.
  3. Serializes the mutated representation

- In Libpng, Custom mutator parses the PNG file into an in-memory data structure, mutates it, and serializes the mutant back to PNG

Custom Mutator

- Custom Mutator is combined while building fuzzer.
  Add flag -include png_mutator.h

```
extern "C" size_t LLVMFuzzerCustomMutator(uint8_t *Data, size_t Size, size_t
MaxSize, unsigned int Seed);
```

1. give data to PngMutator class as a stringstream
2. class PngMutator is defined.
   ```
   01 PngMutator p(in); // Parse the input stream as a PNG file
   02 p.Mutate(LLVMFuzzerMutate, Seed);
   // Mutate the in-memory representation of a PNG file
   03 p.Serialize(out); // Write back the PNG file
   ```
3. return mutated data and its size. If size exceed MaxSize, discard it.

Custom Mutator

- Class PngMutator

Class PngMutator
public :
PngMutator(std::istream & in){}

1. Parse the input stream as a PNG file.
2. Put every chunk into its own vector.
3. Uncompress chunk data when the format is compressed.
4. Then, merge the IDAT(data) chunks into one vector

void Serialize(std::ostream &out){}

1. Add PNG signature and IEND chunk at the end
2. Write back the PNG file.

Custom Mutator

- Class PngMutator

```
using Mutator = size_t (*)(uint8_t *Data, size_t Size, size_t MaxSize);
// Raw byte array mutator
void Mutate(Mutator m, unsigned int seed){}
```

1. Mutator contains chunks
2. Given the same seed, the same mutation is performed
3. Provide custom mutation. That might increase efficiency. For example,
   - Shuffle chunks
   - Delete a random chunks
   - Insert a random chunk with one of the known types, or a random type
   - give new chunk type

Custom Mutator - CrossOver

- LLVMFuzzerCustomCrossOver(uint8_t *data1, size_t size1, uint8_t *data2, size_t size2, uint8_t *Out, size_t MaxOutSize, unsinged int Seed);

1. PngMutation is done with two Data
   PngMutator p1(in1);
   PngMutator p2(in2);
2. Takes a random chunk from data2 and inserts into data1
   p1.CrossOver(p2, Seed);
   void CrossOver(const PngMutator &p, unsigned int seed);
   //Takes a random chunk from p and inserts into *this

Libpng Build & Running Options

- **Build options**
  -Without custom mutator
  clang++ [CFLAGS] -std=c++11 libpng_target.cc libpng12.a
  –fsanitize=fuzzer -I BUILD -lz -o libpng_fuzzer

  -With Custom mutator add below flags
  -include png_mutator.h -DPNG_MUTATOR_DEFINE_LIBFUZZER_CUSTOM_MUTATOR
  -DSTANDALONE_TARGET=$STANDALONE_TARGET
- **Running options**
  ./libpng_fuzzer -close_fd_mask=3 -exit_on_src_pos=**[source location]**
  -runs=100000000 -jobs=8 -workers=8 CORPUS/ seeds/
  -rss_limit_mb=3000
- **One seed file is provided**
  - A small png file is provided to offer format of png file

Error in Libpng

- The test suite may call malloc(2147483648)

  This test suite is used to verify that the fuzzer can reach a set of known source location

  source location should be provided with a flag exit_on_src_pos -exit_on_src_pos=[source location]

Library under test : SQLite

- **What does library for?**
  - SQLite is a C-language library that implements a small, fast, few dependencies, high-reliability, full-featured, SQL database engine
- **What API was under test?**
  - Query execution

```
int sqlite3_exec(sqlite3*,
                 const char *,
                 int(*callback)(void*, int, char**, char**),
                 void *,
                 char ** );
```

In what way does the library have been tested?

There is configuration that should be done to test SQLite

1. **Database connection handle**

    typedef struct sqlite3 sqlite3;

Each open SQLite database is represented by a pointer to an instance of structure named “sqlite3”.

Constructor : sqlite3_open(), sqlite_open_v2()

Destructor : sqlite3_close(), sqlite3_close_v2()

In what way does the library have been tested?

2. Database connection should be opened and use an in-memory database

```
int sqlite3_open_v2(
        const char *filename,       /* Database filename */
        sqlite3 **ppDb,             /* database connection handle */
        int flags,                  /* Flags */
        const char *zVfs            /* Name of VFS module(using default) */
);
```

- Open an SQLite database file as specified by the filename argument.
- Flags

required flag : **SQLITE_OPEN_READWRITE | SQLITE_OPEN_CREATE**
 -> The database is opened for reading and writing and is created if it does not already exist.

optional flag : **SQLITE_OPEN_MEMORY**
 -> The database will be opened as an in-memory database.

In what way does the library have been tested?

3. Using first byte of data as a user defined selector that enable or disable below options.

- progress handler
- foreign key constraints
- limit the number of result rows coming out of the evaluated SQL statements

uint8_t uSelector; /* First byte of input data[] */

uSelector =

		7	6	5	4	3	2	1	0
0	b	1	1	1	1	1	1	0	1

Bit 0 enable or disable progress handler

Bit 1 indicates progress handler return value

Bit 2 enable or disable foreign key constraints

Bit 3~7 indicates the number of result rows

In what way does the library have been tested?

3. Using first byte of data as a user defined selector that enable or disable below options.

- progress handler :

sqlite3_progress_handler() interface invoke a callback periodically as each SQL statement runs, and have that callback return non-zero to abort the statement if the statement runs for too long.

- foreign key constraints : SQLITE_DBCONFIG_ENABLE_FKEY

sqlite3_db_config() interface is used to make configuration changes to a database connection

- limit on the number of result rows :

The value from selector is passed to sqlite3_exec() as argument.

In what way does the library have been tested?

4. sqlite_exec() interface expects a string that ends with null

sqlite3_mprintf() work alike with printf().
However, sqlite3_mprintf() write their results into memory obtained from sqlite3_malloc64()

```
Char *zSql;
zSql = sqlite3_mprintf("%.*s", (int)size, data);
```

In what way does the library have been tested?

4. sqlite_exec() interface expects a string that ends with null

```
sqlite3_exec(db, zSql, exec_handler, (void*)&execCnt, &zErrMsg);


// db : sqlite3 variable
// zSql : random input data
// exec_handler : This argument is invoked for each result of row coming out of evaluated SQL statements. (Could be NULL)
// execCnt : first argument of exec_handler
//zErrMsg : Error message is written in here
```

Target function generation for SQLite

1. #include “sqlite3.h”

sqlite3.h and sqlite3.c file is in same directory with target function.

-Through this, we can build even if libsqlite3-dev is not on the system.

2.Target function source code

3.SQlite dictionary file

Running Options : Improve efficiency

- -dict
  : Provide a dictionary of input keywords. Libfuzzer supports user-supplied dictionaries with input language keywords or other interesting byte sequences.

  - Example in SQLite
    function_abs=" abs(1)”
    // Adds abs(1) to the dictionary
    " application_id”
    // The name of the keyword may be omitted

SQLite Build & Running Options

- **Build options**
  clang++ [CFLAGS] -ldl -pthread sqlite3.o ossfuzz.o
  –fsanitize=fuzzer -o sqlite_fuzzer
- **Running options**
  ./sqlite_fuzzer –dict=[dictionary_file] –jobs=8 -workers=8
  CORPUS/
  -dict : Modeled base on SQLite documentation
- **seed is not provided**

Error in SQLite

- The test suite found two heap-buffer-overflows and a memory leak
- Among the bugs, for heap-buffer-overflow as example,

Crashed input saved in crash-* file
SELECT(2,5)IN(SELECT 2,9)""""""""WHERE""AND""""""""AND""AND""""""""AND""""""""AND""""""""AND""""""”

Crash Type: Heap-use-after-free READ 8
Crash Address: 0x619000000e30
Crash State:
exprAnalyze
sqlite3WhereExprAnalyze
sqlite3WhereBegin

Libssh library (libssh-2017-1272)

- **What does Libssh library for?**
  With Libssh, it is available to remotely execute programs, transfer files(SFTP), or use a secure and transparent tunnel for remote programs.

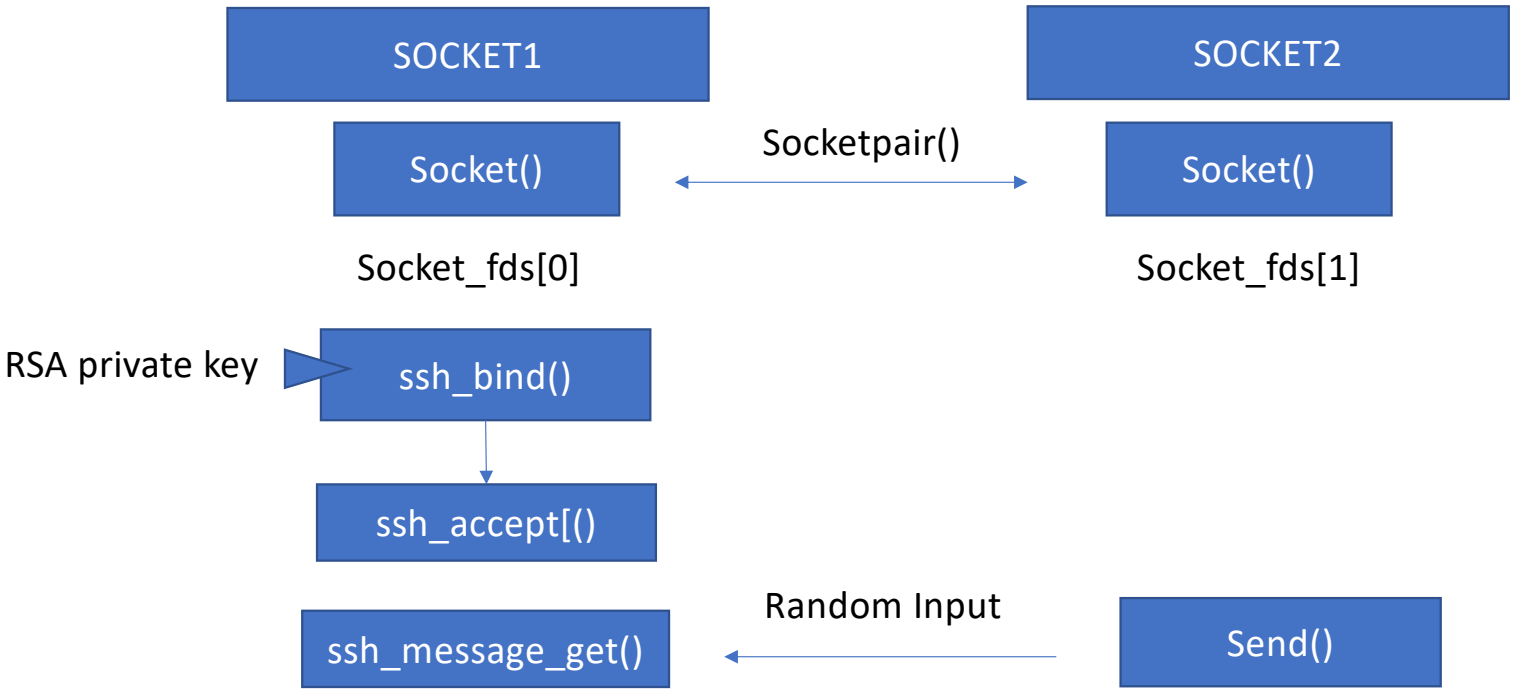
- **What API was under test?**
  ssh_message_get(ssh_session session);

  Retrieve SSH message from SSH Session

In what way does the library have been tested?

There are several features that should be considered to test Libssh

1. Two sockets should be set



In what way does the library have been tested?

There are several features that should be considered to test Libssh

1. Server and Client should be set

Using socketpair function, create a pair of connected sockets.
Using ssh functions in Libssh, transfer mutated inputs

- **socketpair() : create a pair of connected sockets**
  socketpair(int domain, int type, int protocol, int socket_fds[2]);
  socket_fds[2] : file descriptors used in referencing the new sockets are returned in socket_fds[0] and socket_fds[1]
- **Send input to sv[1] and shutdown**
  #include<sys/socket.h>
  send(socket_fds[1], data, size, 0);
  shutdown(socket_fds[1], SHUT_WR);

In what way does the library have been tested?

There are several features that should be considered to test Libssh

2. SSH server setting

- Open temporal file which contains RSA private key

Temporal RSA private key should be stored in target function as well

- Create a new SSH server bind

ssh_bind ssh_bind_new(void);

- Create a new SSH session

ssh_session ssh_new(void);

In what way does the library have been tested?

There are several features that should be considered to test Libssh

2. SSH server setting
- Set SSH bind options : RSA private key setting

: ssh_bind_options_set(ssh_bind, SSH_BIND_OPTIONS_RSAKEY, “private key location”)
//ssh_bind from step ssh server bind

- Accept an incoming SSH connection on the given file descriptor

int ssh_bind_accept_fd (ssh_bind ssh_bind_o,
ssh_session session,
socket_t fd)

Accept an incoming ssh connection on the given file descriptor and initialize the session

Target function generation

3. Retrieve message through SSH

- Beforehand connection, RSA key should be exchanged. The below function will handle it

ssh_handle_key_exchange(ssh_session);
Handles the key exchange and set up encryption

Ssh_message_get(ssh_session)
Retrieve SSH message from a SSH session

Target function generation configuration

Required source

1. **Set required header file and install libssh(INSTALL file)**

   ```
   #include <libssh/libssh.h>
   #include <libssh/server.h>
   ```

2. **RSA private key should be stored.**

   ```
   static const char kRSAPrivateKeyPEM[] =
   "-----BEGIN RSA PRIVATE KEY-----\n
   "MIIEowIBAAKCAQEArAOREUWlBXJAKZ5hABYyxnRayDZP1bJeLbPVK+npxe\n"
   …
   ```

Libssh Build & Running Options

- **Building options**
  clang [CFLAGS] –std=c++11 –fsanitize=fuzzer target.cc
  -I BUILD/include libssh.a –lcrypto –lgss –lz -o libssh_fuzzer
- **Running options**
  ./libssh_fuzzer –max_len=[MAXLEN] –jobs=8 -workers=8
  CORPUS/
  -max_len=[MAXLEN] : maximum length of a test input (60)
- **Seed is not provided**

Error

- Error found in Libssh : Memory leak bug
- ERROR: LeakSanitizer: detected memory leaks Direct leak of 1 byte(s) in 1 object(s) allocated
  crash state :
  calloc
  ssh_packet_userauth_info_response src/messages.c:1001:30
  ssh_packet_process src/packet.c:451:5
  ssh_packet_socket_callback src/packet.c:332:13
  ssh_socket_pollcallback src/socket.c:298:25
  ssh_poll_ctx_dopoll src/poll.c:632:27
  ssh_handle_packets src/session.c:634:10
  ssh_handle_packets_termination src/session.c:696:15
  ssh_handle_key_exchange src/server.c:589:10
  LLVMFuzzerTestOneInput

LibFuzzer Evaluation

- When libFuzzer is not a good solution for a problem?
  - If the test inputs are validated by the target library and the validator asserts/crashes on invalid inputs.
  - Bugs in the target library may accumulate without being detected. (E.g. a memory corruption that goes undetected at first and then leads to a crash while testing another input)
  - The target library have significant change on global state that is not reset between the runs

Fuzzing Practice-Myhtml

- MyHTML is a fast HTML Parser using Threads implemented as a pure C99 library with no outside dependencies.
  https://github.com/lexborisov/myhtml.git

- Two functions for parsing and serialization will be tested

  myhtml_parse(myhtml_tree* , myencoding_t, const char*, size_t)
  myhtml_serialization(myhtml_tree_node_t*, mycore_string_raw_t)

Fuzzing steps

1. Get library from Github and build
   - build with fsanitize=fuzzer-no-link
2. Generate target function
   - Functions to test : myhtml_parse(), myhtml_serialization()
3. For efficiency, seed corpus and dictionary should be provided
   - Seeds contain html syntax may increase initial coverage
   - Provide dictionary that will be inserted during mutation.
     http://www.csun.edu/science/help/help_docs/html_tags.html
4. Build target function and run
   - To detect error, address sanitizer should be combined

Generate target function

1. **Initiation must be done.**
   Basic struct myhtml_t should be created and initialized.

   myhtml_create(void)

   mystatus_t myhtml_init(myhtml_t* myhtml,
   enum myhtml_options opt,
   size_t thread_count,
   size_t queue_size);

Generate target function

2. Tree struct should be defined to parse html file

```
myhtml_tree_t* tree = myhtml_tree_create();

myhtml_tree_init(myhtml_tree_t*,
                 myhtml_t* );
```

This tree contains information about token, attribute of token, reference for nodes and another options

Generate target function

3. For efficiency, below option is added

- parsing option : set parse flags

```
myhtml_tree_parse_flags_set(
                                myhtml_tree_t*,
                                myhtml_tree_parse_flags_t );
```

MyHTML_TREE_PARSE_FLAGS_SKIP_WHITESPACE_TOKEN

: skip white space token

Bug found in Myhtml

- Null pointer dereference

  AddressSanitizer: SEGV on unkown address 0x000000000000
  The signal is caused by a READ memory access.
  Address points to the zero page

- Fuzzing Environment
  (In GCE) Ubuntu 16.0.4 / n1-standard-4 / vCPU:4 / memory 15gb

  ./my_fuzzer CORPUS/ SEEDS/ -dict=html.dict –jobs=4 –workers=4

- The library contains null pointer dereference.