LibFuzzer

* libpng

1. 克隆项目到本地

git clone https://github.com/glennrp/libpng.git

1. 编译项目：

./configure --disable-shared

make

1. 编写target.cc文件：

*#include <stddef.h>*

*#include <string.h>*

*#include <stdint.h>*

*#include <assert.h>*

*#define PNG\_INTERNAL*

*#include "png.h"*

struct BufState {

const uint8\_t\* data;

size\_t bytes\_left;

};

void user\_read\_data(png\_structp png\_ptr, png\_bytep data, png\_size\_t length) {

BufState\* buf\_state = static\_cast<BufState\*>(png\_get\_io\_ptr(png\_ptr));

**if** (length > buf\_state->bytes\_left) {

png\_error(png\_ptr, "read error");

}

memcpy(data, buf\_state->data, length);

buf\_state->bytes\_left -= length;

buf\_state->data += length;

}

static const int kPngHeaderSize = 8;

struct ScopedPngObject {

~ScopedPngObject() {

**if** (row && png\_ptr) {

png\_free(png\_ptr, row);

}

**if** (png\_ptr && info\_ptr) {

png\_destroy\_read\_struct(&png\_ptr, &info\_ptr, nullptr);

}

delete buf\_state;

}

png\_infop info\_ptr = nullptr;

png\_voidp row = 0;

png\_structp png\_ptr = nullptr;

BufState \*buf\_state = nullptr;

};

bool DetectLargeSize(const uint8\_t \*data, size\_t size) {

uint8\_t \*ihdr = reinterpret\_cast<uint8\_t \*>(memmem(data, size, "IHDR", 4));

**if** (!ihdr) **return** false;

**if** (ihdr + 12 > data + size) **return** false;

uint32\_t W = \*(uint32\_t\*)(ihdr + 4);

uint32\_t H = \*(uint32\_t\*)(ihdr + 8);

W = \_\_builtin\_bswap32(W);

H = \_\_builtin\_bswap32(H);

uint64\_t WxH = static\_cast<uint64\_t>(W) \* H;

**if** (WxH > 100000ULL) {

// fprintf(stderr, "ZZZ %zu %u %u\n", WxH, W, H);

**return** true;

}

**return** false;

}

// Fuzzing entry point

extern "C" int LLVMFuzzerTestOneInput(const uint8\_t\* data, size\_t size) {

**if** (size < kPngHeaderSize) {

**return** 0;

}

ScopedPngObject O;

**if** (png\_sig\_cmp(const\_cast<uint8\_t\*>(data), 0, kPngHeaderSize)) {

// not a PNG.

**return** 0;

}

// **if** (DetectLargeSize(data, size)) **return** 0;

auto &png\_ptr = O.png\_ptr;

png\_ptr = png\_create\_read\_struct

(PNG\_LIBPNG\_VER\_STRING, nullptr, nullptr, nullptr);

assert(png\_ptr);

png\_ptr->flags &= ~PNG\_FLAG\_CRC\_CRITICAL\_MASK;

png\_ptr->flags |= PNG\_FLAG\_CRC\_CRITICAL\_IGNORE;

png\_ptr->flags &= ~PNG\_FLAG\_CRC\_ANCILLARY\_MASK;

png\_ptr->flags |= PNG\_FLAG\_CRC\_ANCILLARY\_NOWARN;

auto &info\_ptr = O.info\_ptr;

info\_ptr = png\_create\_info\_struct(png\_ptr);

assert(info\_ptr);

// Setting up reading from buffer.

auto &buf\_state = O.buf\_state;

buf\_state = new BufState();

buf\_state->data = data + kPngHeaderSize;

buf\_state->bytes\_left = size - kPngHeaderSize;

png\_set\_read\_fn(png\_ptr, buf\_state, user\_read\_data);

png\_set\_sig\_bytes(png\_ptr, kPngHeaderSize);

int passes = 0;

// libpng error handling.

**if** (setjmp(png\_ptr->jmpbuf)) {

**return** 0;

}

// png\_ptr->mode & PNG\_HAVE\_IDAT

// Reading

png\_read\_info(png\_ptr, info\_ptr);

png\_uint\_32 width, height;

int bit\_depth, color\_type, interlace\_type, compression\_type;

int filter\_type;

**if** (!png\_get\_IHDR(png\_ptr, info\_ptr, &width, &height,

&bit\_depth, &color\_type, &interlace\_type,

&compression\_type, &filter\_type)) {

**return** 0;

}

**if** (height \* width > 2000000) **return** 0;

passes = png\_set\_interlace\_handling(png\_ptr);

png\_start\_read\_image(png\_ptr);

O.row = png\_malloc(png\_ptr, png\_get\_rowbytes(png\_ptr, info\_ptr));

**for** (int pass = 0; pass < passes; ++pass) {

**for** (png\_uint\_32 y = 0; y < height; ++y) {

png\_read\_row(png\_ptr, static\_cast<png\_bytep>(O.row), NULL);

}

}

**return** 0;

}

1. 编译测试驱动文件并链接

clang++ -DFUZZING\_BUILD\_MODE\_UNSAFE\_FOR\_PRODUCTION -std=c++11 target.cc BUILD/.libs/libpng12.a libFuzzingEngine-fsanitize\_fuzzer.a -I BUILD/ -I BUILD -lz -o

1. 运行测试并记录到文件中

test\_source\_location() {

SRC\_LOC="$1"

echo "test\_source\_location: $SRC\_LOC"

./.-fsanitize\_fuzzer exit\_on\_src\_pos=$SRC\_LOC src\_pos=./ -close\_fd\_mask=3 -runs=100000000 -jobs=8 -workers=8 ./seeds -rss\_limit\_mb=3000

grep "INFO: found line matching '$SRC\_LOC'" fuzz-\*.log || exit 1

}

test\_source\_location png.c:1035

test\_source\_location png\_handle\_sRGB

test\_source\_location pngread.c:757

* Libjpeg-turbo

1. 克隆项目到本地

git clone https://github.com/libjpeg-turbo/libjpeg-turbo.git

1. 编译项目

cd BUILD

autoreconf -fiv

./configure --disable-shared

1. 测试驱动文件target.cc：

*#include <stdint.h>*

*#include <stdlib.h>*

*#include <memory>*

*#include <turbojpeg.h>*

extern "C" int LLVMFuzzerTestOneInput(const uint8\_t \*data, size\_t size) {

tjhandle jpegDecompressor = tjInitDecompress();

int width, height, subsamp, colorspace;

int res = tjDecompressHeader3(

jpegDecompressor, data, size, &width, &height, &subsamp, &colorspace);

**if** (res != 0 || width == 0 || height == 0 || ((size\_t)width \* height > (1024 \* 1024))) {

tjDestroy(jpegDecompressor);

**return** 0;

}

std::unique\_ptr<unsigned char[]> buf(new unsigned char[width \* height \* 3]);

tjDecompress2(jpegDecompressor, data, size, buf.get(), width, 0, height, TJPF\_RGB, 0);

tjDestroy(jpegDecompressor);

**return** 0;

}

1. 编译测试驱动文件并链接

clang++ -fsanitize=address,fuzzer-no-link -fsanitize-address-use-after-scope -std=c++11 ./libjpeg\_turbo\_fuzzer.cc -I BUILD BUILD/.libs/libturbojpeg.a -fsanitize=fuzzer -o .-fsanitize\_fuzzer

1. 运行测试并记录日志到文件中

test\_source\_location() {

SRC\_LOC="$1"

echo "test\_source\_location: $SRC\_LOC"

rm -f \*.log

./.-fsanitize\_fuzzer -artifact\_prefix=CORPUS-.-fsanitize\_fuzzer/ -exit\_on\_src\_pos=$SRC\_LOC -jobs=8 -workers=8 -print\_pcs=1 CORPUS-.-fsanitize\_fuzzer ./seeds

grep "INFO: found line matching '$SRC\_LOC'" fuzz-\*.log || exit 1

}

test\_source\_location jdmarker.c:659

* Libssh

1. 克隆项目到本地

git clone git://git.libssh.org/projects/libssh.git

1. 编译项目

cmake -DCMAKE\_C\_COMPILER="clang" \

-DCMAKE\_CXX\_COMPILER="clang++" \

-DCMAKE\_C\_FLAGS="-O2 -fno-omit-frame-pointer -gline-tables-only -fsanitize=address,fuzzer-no-link -fsanitize-address-use-after-scope -fcommon" \

-DCMAKE\_CXX\_FLAGS="-O2 -fno-omit-frame-pointer -gline-tables-only -fsanitize=address,fuzzer-no-link -fsanitize-address-use-after-scope -fcommon" \

-DWITH\_STATIC\_LIB=ON ..

1. 测试驱动文件：

*#include <assert.h>*

*#include <fcntl.h>*

*#include <stdint.h>*

*#include <string.h>*

*#include <sys/socket.h>*

*#include <unistd.h>*

*#define LIBSSH\_STATIC 1*

*#include <libssh/libssh.h>*

*#include <libssh/server.h>*

static const char kRSAPrivateKeyPEM[] =

"-----BEGIN RSA PRIVATE KEY-----\n"

"MIIEowIBAAKCAQEArAOREUWlBXJAKZ5hABYyxnRayDZP1bJeLbPVK+npxemrhHyZ\n"

"gjdbY3ADot+JRyWjvll2w2GI+3blt0j+x/ZWwjMKu/QYcycYp5HL01goxOxuusZb\n"

"i+KiHRGB6z0EMdXM7U82U7lA/j//HyZppyDjUDniWabXQJge8ksGXGTiFeAJ/687\n"

"uV+JJcjGPxAGFQxzyjitf/FrL9S0WGKZbyqeGDzyeBZ1NLIuaiOORyLGSW4duHLD\n"

"N78EmsJnwqg2gJQmRSaD4BNZMjtbfiFcSL9Uw4XQFTsWugUDEY1AU4c5g11nhzHz\n"

"Bi9qMOt5DzrZQpD4j0gA2LOHpHhoOdg1ZuHrGQIDAQABAoIBAFJTaqy/jllq8vZ4\n"

"TKiD900wBvrns5HtSlHJTe80hqQoT+Sa1cWSxPR0eekL32Hjy9igbMzZ83uWzh7I\n"

"mtgNODy9vRdznfgO8CfTCaBfAzQsjFpr8QikMT6EUI/LpiRL1UaGsNOlSEvnSS0Z\n"

"b1uDzAdrjL+nsEHEDJud+K9jwSkCRifVMy7fLfaum+YKpdeEz7K2Mgm5pJ/Vg+9s\n"

"vI2V1q7HAOI4eUVTgJNHXy5ediRJlajQHf/lNUzHKqn7iH+JRl01gt62X8roG62b\n"

"TbFylbheqMm9awuSF2ucOcx+guuwhkPir8BEMb08j3hiK+TfwPdY0F6QH4OhiKK7\n"

"MTqTVgECgYEA0vmmu5GOBtwRmq6gVNCHhdLDQWaxAZqQRmRbzxVhFpbv0GjbQEF7\n"

"tttq3fjDrzDf6CE9RtZWw2BUSXVq+IXB/bXb1kgWU2xWywm+OFDk9OXQs8ui+MY7\n"

"FiP3yuq3YJob2g5CCsVQWl2CHvWGmTLhE1ODll39t7Y1uwdcDobJN+ECgYEA0LlR\n"

"hfMjydWmwqooU9TDjXNBmwufyYlNFTH351amYgFUDpNf35SMCP4hDosUw/zCTDpc\n"

"+1w04BJJfkH1SNvXSOilpdaYRTYuryDvGmWC66K2KX1nLErhlhs17CwzV997nYgD\n"

"H3OOU4HfqIKmdGbjvWlkmY+mLHyG10bbpOTbujkCgYAc68xHejSWDCT9p2KjPdLW\n"

"LYZGuOUa6y1L+QX85Vlh118Ymsczj8Z90qZbt3Zb1b9b+vKDe255agMj7syzNOLa\n"

"/MseHNOyq+9Z9gP1hGFekQKDIy88GzCOYG/fiT2KKJYY1kuHXnUdbiQgSlghODBS\n"

"jehD/K6DOJ80/FVKSH/dAQKBgQDJ+apTzpZhJ2f5k6L2jDq3VEK2ACedZEm9Kt9T\n"

"c1wKFnL6r83kkuB3i0L9ycRMavixvwBfFDjuY4POs5Dh8ip/mPFCa0hqISZHvbzi\n"

"dDyePJO9zmXaTJPDJ42kfpkofVAnfohXFQEy+cguTk848J+MmMIKfyE0h0QMabr9\n"

"86BUsQKBgEVgoi4RXwmtGovtMew01ORPV9MOX3v+VnsCgD4/56URKOAngiS70xEP\n"

"ONwNbTCWuuv43HGzJoVFiAMGnQP1BAJ7gkHkjSegOGKkiw12EPUWhFcMg+GkgPhc\n"

"pOqNt/VMBPjJ/ysHJqmLfQK9A35JV6Cmdphe+OIl28bcKhAOz8Dw\n"

"-----END RSA PRIVATE KEY-----\n";

extern "C" int LLVMFuzzerTestOneInput(const uint8\_t \*data, size\_t size) {

int socket\_fds[2];

int res = socketpair(AF\_UNIX, SOCK\_STREAM, 0, socket\_fds);

assert(res >= 0);

ssize\_t send\_res = send(socket\_fds[1], data, size, 0);

assert(send\_res == size);

res = shutdown(socket\_fds[1], SHUT\_WR);

assert(res == 0);

int fd = open("/tmp/libssh\_fuzzer\_private\_key", O\_WRONLY | O\_CREAT, S\_IRWXU);

assert(fd >= 0);

ssize\_t write\_res = write(fd, kRSAPrivateKeyPEM, strlen(kRSAPrivateKeyPEM));

assert(write\_res == strlen(kRSAPrivateKeyPEM));

close(fd);

ssh\_bind sshbind = ssh\_bind\_new();

ssh\_session session = ssh\_new();

ssh\_bind\_options\_set(sshbind, SSH\_BIND\_OPTIONS\_RSAKEY, "/tmp/libssh\_fuzzer\_private\_key");

res = ssh\_bind\_accept\_fd(sshbind, session, socket\_fds[0]);

assert(res == SSH\_OK);

**if** (ssh\_handle\_key\_exchange(session) == SSH\_OK) {

**while** (true) {

ssh\_message message = ssh\_message\_get(session);

**if** (!message) {

**break**;

}

ssh\_message\_free(message);

}

}

close(socket\_fds[0]);

close(socket\_fds[1]);

ssh\_disconnect(session);

ssh\_free(session);

ssh\_bind\_free(sshbind);

**return** 0;

}

1. 编译测试驱动文件并链接

clang++ -O2 -fno-omit-frame-pointer -gline-tables-only -fsanitize=address,fuzzer-no-link -fsanitize-address-use-after-scope -std=c++11 ./libssh\_server\_fuzzer.cc -I BUILD/include/ BUILD/build/src/libssh.a -fsanitize=fuzzer -lcrypto -lgss -lz -o .-fsanitize\_fuzzer

1. 运行测试并记录日志到文件中

./.-fsanitize\_fuzzer -max\_len=60 -artifact\_prefix=CORPUS-.-fsanitize\_fuzzer/ -jobs=8 -workers=8 CORPUS-.-fsanitize\_fuzzer