FuzzBuilder: Automated Building Greybox Fuzzing Environment for C/C++ Library

2019. 12. 12.

Graduate School of Information Security
Korea University

Joonun Jang,
Huy Kang Kim

KOREA UNIVERSITY

HCRL
Hacking and Countermeasure Research Lab

CONTENTS

Introduction

■Greybox Fuzzing for Development Process

- □ Greybox fuzzing is a famous test to find security vulnerabilities in software
- □ Good performance regardless of source code
- □ It is necessary to apply greybox fuzzing to development process to prevent security vulnerabilities at an early stage

greybox fuzzer	# of bugs	greybox fuzzer	# of bugs
AFL	more than 370	T-Fuzz	3
AFLFast	12	Angora	175
AFLGo	39	CollAFL	157
VUzzer	8		

The number of bugs found by recent greybox fuzzers
(Most of them are from open source programs)

Introduction

■Challenges of Greybox Fuzzing for Libraries

- □ Greybox fuzzing requires an execution of program
- □ Security vulnerabilities in libraries are critical
- □ Executable should be generated for library fuzzing

project	unittest	fuzzing
libphonenumber	✔	✗
rapidjson	✔	✗
http-parser	✔	✗
redcarpet	✔	✗
grbl	✔	✗
tinyxml2	✔	✗
cJSON	✔	✔
yajl	✔	✗
libpostal	✔	✗
pugixml	✔	✔

Status of support for unit test and fuzzing by developers in case of famous top 10 libraries in GitHub

Introduction

■Contributions

- □ Automated executable generation for library fuzzing
  - Practical challenges and solutions
  - Not researched enough so far
- □ Artifact is available
  - Available at https://github.com/hksecurity/FuzzBuilder
    - Source code of FuzzBuilder
    - User manuals
    - Detailed steps for regeneration of experiment results

Motivations & Background

■Libfuzzer

- □ Famous fuzzing framework for libraries
- □ Fuzzer, interface and user manual are provided
- □ Motivations about how to generate executables for library fuzzing

```
extern "C" int LLVMFuzzerTestOneInput(const uint8_t* data, size_t size)
{
  std::string input(reinterpret_cast<const char*>(data), size);
  auto hash_salt = std::hash<std::string>()(input);

  for (int use_ns = 0; use_ns <= 1; ++use_ns) {
    XML_Parser parser =
        use_ns ? XML_ParserCreateNS(kEncoding, '\n') :
                 XML_ParserCreate(kEncoding);

    XML_SetHashSalt(parser, hash_salt);
    XML_Parse(parser, input.c_str(), input.size(), true);
    XML_ParserFree(parser);
  }
  return 0;
}
```

How to use LibFuzzer (example of expat project)

Motivations & Background

■Fuzzable API(FA)

- □ FA is a library API function that get input values
- □ Fuzzer can feed generated inputs to library by calling FA
- □ Generated executable should include at least one FA

```
1  extern "C" int LLVMFuzzerTestOneInput(const uint8_t* data, size_t size)
       {
2    std::string input(reinterpret_cast<const char*>(data), size);
3    auto hash_salt = std::hash<std::string>()(input);
4
5    for (int use_ns = 0; use_ns <= 1; ++use_ns) {
6      XML_Parser parser =
7          use_ns ? XML_ParserCreateNS(kEncoding, '\n') :
8                   XML_ParserCreate(kEncoding);
9
10     // Set a hash salt to prevent MSan from crashing on random bytes
           generation.
11     XML_SetHashSalt(parser, hash_salt);
12     XML_Parse(parser, input.c_str(), input.size(), true);
13     XML_ParserFree(parser);
14   }
15   return 0;
16 }
```

FA: XML_Parse gets input through 2nd and 3rd parameter

Motivations & Background

■Function Sequence

- □ A set of library API functions that are tested jointly
- □ Order of calling library API functions should be considered
  - Valid and efficient fuzzing
- □ Various function sequences should be considered
  - High test coverage
- □ Valid function sequences in example
  - init → insert → parse_A
  - init → insert → parse_B

```
static bool initflag = false;
static char* buffer = NULL;
static int buffer_len = 0;

init() {
    ... // omitted logic about an initialization
    initflag = true;
}

insert(char* input, int len) {
    if(initflag == false) return;

    buffer = calloc(len + 1, 1);
    memcpy(buffer, input, len);
    buffer_len = len;
}

parse_A() {
    ... // omitted logic about a parsing
}

parse_B() {
    ... // omitted logic about a parsing
}
```

Example of Function Sequence

Motivations & Background

■Seed

- □ Sample inputs
- □ General solution to overcome low coverage problem of greybox fuzzing
- □ Preparation of seed is necessary for each target binaries
  - Automated Seed Generation also required for practical fuzzing

```
1 read(0, buf, 1024);
2 if(strncmp(buf, "FUZZ", 4) == 0) {
3     process(); // requires roughly 2^32 input generation
4 } else {
5     exit(0);
6 }
```

Example to show why seed is required

Motivations & Background

■Unit Test

- □A set of test code to test unit funtualities of a program
- □Ideally, every functuality should be tested
  - Various combination of functions should be tested
  - Various test inputs are required to test each function

■FuzzBuilder

- □Executable Generation using various function sets in unit tests
- □Seed Generation using various test inputs in unit tests

Methodology

■Automated Executable Generation

- □ Modification of unit test source code
  - Transforming unit test executables to fuzzble executables
- □ Assumptions
  - Each unit test is a function in unit tests
  - Each test function does not affect other test function
- □ Overall Process
  - select FA
  - preprocess: collecting test functions
  - insert_interface: getting input from fuzzers
  - remove_test: removing unnecessary test functions
  - insert_operands: replacing operands of FA

```
procedure FuzzBuilder(functions)
    tests, entry ← preprocess(functions)
    entry ← insert_interface(entry)
    for all test ∈tests do
        if is_FA_exist(test) then
            test ← insert_operands(test)
        else
            test ← remove_test(test)
        end if
    end for
    modify(entry, tests)
```

Overall process to generate executable

Methodology

■Automated Executable Generation

□ insert_interface

- Inserting instructions in main function to get input from a fuzzer

```c
int main(int argc, char* argv[]) {
    test_A();
    test_B();
}

void test_A() {
    ...
    LIBRARY_API_1("test_input_1", strlen("test_input_1"));
    ...
}

void test_B() {
    ...
    LIBRARY_API_2("test_input_2", strlen("test_input_2"));
    ...
}
```

```c
char* FUZZ_INPUT;
int FUZZ_INPUT_LEN;

int main(int argc, char* argv[]) {
    read_from_fuzzer(&FUZZ_INPUT, &FUZZ_INPUT_LEN);
    test_A();
    test_B();
}

void test_A() {
    ...
    LIBRARY_API_1("test_input_1", strlen("test_input_1"));
    ...
}

void test_B() {
    ...
    LIBRARY_API_2("test_input_2", strlen("test_input_2"));
    ...
}
```

After inserting instructions to get input from fuzzers

Methodology

■Automated Executable Generation

□ remove_test

- Removing test functions that do not include calling FAs

```
1  char* FUZZ_INPUT;
2  int FUZZ_INPUT_LEN;
3
4  int main(int argc, char* argv[]) {
5      read_from_fuzzer(&FUZZ_INPUT, &FUZZ_INPUT_LEN);
6      test_A();
7      test_B();
8  }
9
10 void test_A() {
11     ...
12     LIBRARY_API_1("test_input_1", strlen("test_input_1"));
13     ...
14 }
15
16 void test_B() {
17     ...
18     LIBRARY_API_2("test_input_2", strlen("test_input_2"));
19     ...
20 }
```

```
1  char* FUZZ_INPUT;
2  int FUZZ_INPUT_LEN;
3
4  int main(int argc, char* argv[]) {
5      read_from_fuzzer(&FUZZ_INPUT, &FUZZ_INPUT_LEN);
6      test_A();
7  }
8
9  void test_A() {
10     ...
11     LIBRARY_API_1(FUZZ_INPUT, FUZZ_INPUT_LEN);
12     ...
13 }
```

After removing unnecessary test functions

Methodology

■Automated Executable Generation

□ insert_operands

- Replacing parameter of FA to global variables
- Fuzzer can affect library code behavior

```
1  char* FUZZ_INPUT;
2  int FUZZ_INPUT_LEN;
3
4  int main(int argc, char* argv[]) {
5      read_from_fuzzer(&FUZZ_INPUT, &FUZZ_INPUT_LEN);
6      test_A();
7  }
8
9  void test_A() {
10     ...
11     LIBRARY_API_1("test_input_1", strlen("test_input_1"));
12     ...
13 }
```

```
1  char* FUZZ_INPUT;
2  int FUZZ_INPUT_LEN;
3
4  int main(int argc, char* argv[]) {
5      read_from_fuzzer(&FUZZ_INPUT, &FUZZ_INPUT_LEN);
6      test_A();
7  }
8
9  void test_A() {
10     ...
11     LIBRARY_API_1(FUZZ_INPUT, FUZZ_INPUT_LEN);
12     ...
13 }
```

After chainging parameters for calling FA

Methodology

■Automated Seed Generation

- □ Modification of library source coce
- □ Test inputs are stored into the specific file by executing unit test executable
- □ Store each test inputs in the specific file into a separate files

```
FA(char* input, size_t size, ...) {
    int fd = open("file.txt", ...);
    write(fd, input, size);
    close(fd);
    ...
}
```

Instructions for collecting test inputs

Evaluation

■Experiment Design

- □ The effectiveness of automatically generated seeds
- □ The effectiveness of automatically generated executables
- □ The effectiveness of FuzzBuilder as a bug finding tool

■Metrics

- □ Line coverage, The number of discovered bugs

■Comparative Evaluation

- □ OSS-Fuzz

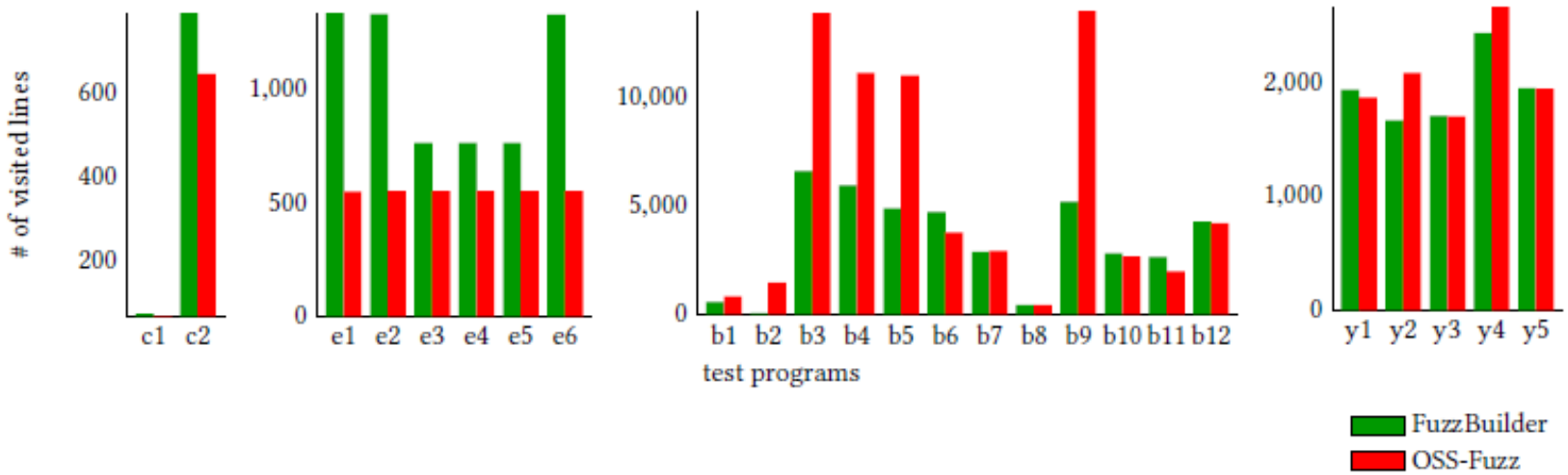
■Target library projects

- □ OSS-Fuzz: c-ares, expat, boringssl, yara
- □ GitHub: cJSON, mpc

Evaluation

■Automated Seed Generation

- □ Seed by FuzzBuilder could help visit more code lines than OSS-Fuzz in most cases



Line coverage comparison between OSS-Fuzz and FuzzBuilder

Evaluation

■Automated Seed Generation

- □ Best
  - The number of test inputs is more than the number of seed files
- □ Worst
  - Function sequences are not defined in unit tests
    - Test inputs are not defined in unit tests

program	# of seed files	
	OSS-Fuzz	FuzzBuilder
c1	6	12
c2	25	57
e1, e2	1	14
e3, e4	1	6
e5	1	7
e6	1	13
b1	42	12
b2	57	6

program	# of seed files	
	OSS-Fuzz	FuzzBuilder
b3	281	7
b4	85	6
b5	84	6
b6	4	60
b7	14	39
b8	41	13
b9	269	8
b10	2	13

program	# of seed files	
	OSS-Fuzz	FuzzBuilder
b11	8	65
b12	87	153
y1	6	15
y2	3	14
y3	15	15
y4	8	15
y5	3	216

The number of seed (after applying afl-cmin)

Evaluation

■Automated Executable Generation

- □ 8 Test sets based on FA
- □ Allocating same fuzzing time to set of executables in each test set
  - In case of T2, fc2 and c2 fuzzed for 6 hours.
  - In case of T3, from e1 to e6 fuzzed for 6 hours; fe1 fuzzed for 36 hours

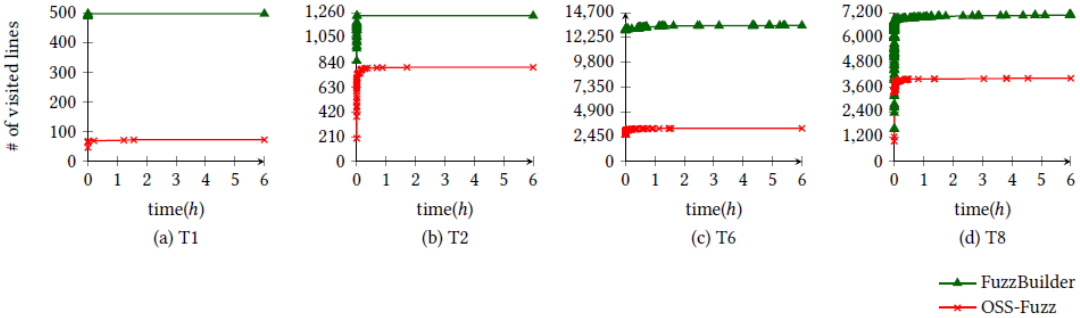
set	*FuzzBuilder*	*OSS-Fuzz*	FA
T1	fc1	c1	ares_create_query
T2	fc2	c2	ares_parse_*
T3	fe1 (36)	e1, e2, e3 e4, e5, e6	XML_Parse
T4	fb1 (60)	b1, b2, b3 b4, b5, b6 b7, b9, b11 b12	CBS_init
T5	fb2, fb3	b8 (12)	BIO_new_mem_buf
T6	fb4	b10	SSL_SESSION_from_bytes
T7	fy1 (24)	y1, y2, y3 y4	yr_rules_scan_mem
T8	fy2	y5	yr_compiler_add_string

Test set and allocated time for each executable(default: 6 hours)

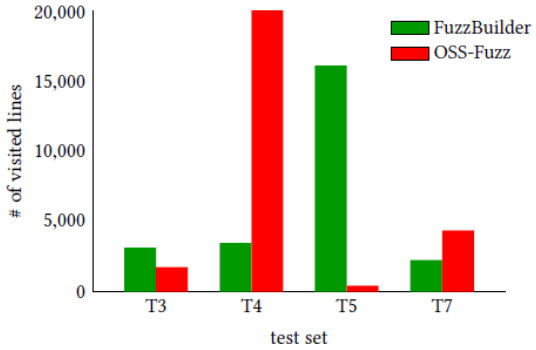
Evaluation

■Automated Executable Generation

- □ Executables by FuzzBuilder guarantee fuzzing based on high coverage of library code in most cases
- □ T4, T7 shows lower performance of FuzzBuilder



Line coverage comparison over time for test set that includes one executable



Line coverage comparison for test set that includes more than one executable

Evaluation

■ Discovered bugs

- □ 4 bugs from 3 projects
- □ A bug in expat
  - expat project have been fuzzed significantly by OSS-Fuzz
  - FuzzBuilder helps to find a bug by finding valid function sequence that can help to reach the state where the bug can be disclosed
- □ A bug in cJSON and mpc
  - FuzzBuilder helps to find a bug by applying greybox fuzzing into library project that is not applied enough so far

project	bug type	result
expat	heap overflow	CVE-2019-15903
cJSON	NULL dereference	CVE-2019-1010239
mpc	stack overflow	reported to developer
mpc	heap overflow	reported to developer

Discovered bugs

Future works

■FA automation

- □ Need to automated for full automation

■Optimization for generated executables

- □ Optimization is required to reduce execution time for efficient fuzzing

■Errors in unit test

- □ Need to be considered to save time of human resources

■Data type expansion

- □ Support of various data type can increase test coverage

Thank you

KOREA UNIVERSITY

HCRL
Hacking and Countermeasure Research Lab

Appendix

■ Executables in OSS-Fuzz

Project	ID	Executable	FA
c-ares	c1	ares_create_query_fuzzer	ares_create_query
	c2	ares_parse_reply_fuzzer	ares_parse_*[1]
expat	e1	parse_ISO_8859_1_fuzzer	XML_Parse
	e2	parse_US_ASCII_fuzzer	
	e3	parse_UTF_16BE_fuzzer	
	e4	parse_UTF_16_fuzzer	
	e5	parse_UTF_16LE_fuzzer	
	e6	parse_UTF_8_fuzzer	
boringssl	b1	bn_div	CBS_init
	b2	bn_mod_exp	
	b3	client	
	b4	dtls_client	
	b5	dtls_server	
	b6	pkcs12	
	b7	pkcs8	

Project	ID	Executable	FA
boringssl	b8	ares_create_query_fuzzer	BIO_new_mem_buf
	b9	ares_parse_reply_fuzzer	CBS_init
	b10	parse_ISO_8859_1_fuzzer	SSL_SESSION_from_bytes
	b11	parse_US_ASCII_fuzzer	CBS_init
	b12	parse_UTF_16BE_fuzzer	
yara	y1	dex_fuzzer	yr_rules_scan_mem
	y2	dotnet_fuzzer	
	y3	elf_fuzzer	
	y4	pe_fuzzer	
	y5	rules_fuzzer	yr_compiler_add_string

Appendix

■ Executables by FuzzBuilder

Project	ID	Unittest Executable	FA
c-ares	fc1	ares_test	ares_create_query
	fc2	ares_test	ares_parse_*
expat	fe1	runtests	XML_Parse
boringssl	fb1	crypto_test	CBS_init
	fb2	ssl_test	BIO_new_mem_buf
	fb3	crypto_test	BIO_new_mem_buf
	fb4	ssl_test	SSL_session_from_bytes
yara	fy1	test-api	yr_rules_scan_mem
	fy2	test-api	yr_rules_scan_mem