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


Joonun Jang,
Huy Kang Kim
CONTENTS

1. Introduction
2. Motivation & Background
3. Methodology
4. Evaluation
5. Future Works
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

<table>
<thead>
<tr>
<th>greybox fuzzer</th>
<th># of bugs</th>
<th>greybox fuzzer</th>
<th># of bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFL</td>
<td>more than 370</td>
<td>T-Fuzz</td>
<td>3</td>
</tr>
<tr>
<td>AFLFast</td>
<td>12</td>
<td>Angora</td>
<td>175</td>
</tr>
<tr>
<td>AFLGo</td>
<td>39</td>
<td>CollAFL</td>
<td>157</td>
</tr>
<tr>
<td>VUzzer</td>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>project</th>
<th>unittest</th>
<th>fuzzing</th>
</tr>
</thead>
<tbody>
<tr>
<td>libphonenumber</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td>rapidjson</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td>http-parser</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td>redcarpet</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td>grbl</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td>tinyxml2</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td>cJSON</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>yajl</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td>libpostal</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td>pugixml</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>

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

```c
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 gets input values
- Fuzzer can feed generated inputs to library by calling FA
- Generated executable should include at least one FA

```c
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);

        // Set a hash salt to prevent NSam from crashing on random bytes
        // generation.
        XML_SetHashSalt(parser, hash_salt);

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

FA: XML_Parse gets input through 2\textsuperscript{nd} and 3\textsuperscript{rd} 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`

```c
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

Example to show why seed is required

```c
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  }
```
Motivations & Background

**Unit Test**
- A set of test code to test unit functionalities of a program
- Ideally, every functionality 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 fuzzable 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

```plaintext
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"));
    ...
}
```

```
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

```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 removing unnecessary test functions
Methodology

Automated Executable Generation

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

```c
char* FUZZ_INPUT;
int FUZZ_INPUT_LEN;

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

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

After changing parameters for calling FA
Methodology

- Automated Seed Generation
  - Modification of library source code
  - 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

<table>
<thead>
<tr>
<th>program</th>
<th># of seed files</th>
<th>OSS-Fuzz</th>
<th>FuzzBuilder</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>5</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>c2</td>
<td>25</td>
<td>57</td>
<td></td>
</tr>
<tr>
<td>e1, e2</td>
<td>1</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>e3, e4</td>
<td>1</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>e5</td>
<td>1</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>e6</td>
<td>1</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>b1</td>
<td>42</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>b2</td>
<td>57</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>program</th>
<th># of seed files</th>
<th>OSS-Fuzz</th>
<th>FuzzBuilder</th>
</tr>
</thead>
<tbody>
<tr>
<td>b3</td>
<td>281</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>b4</td>
<td>85</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>b5</td>
<td>84</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>b6</td>
<td>4</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>b7</td>
<td>14</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td>b8</td>
<td>41</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>b9</td>
<td>260</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>b10</td>
<td>2</td>
<td>13</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>program</th>
<th># of seed files</th>
<th>OSS-Fuzz</th>
<th>FuzzBuilder</th>
</tr>
</thead>
<tbody>
<tr>
<td>b11</td>
<td>8</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>b12</td>
<td>87</td>
<td>153</td>
<td></td>
</tr>
<tr>
<td>y1</td>
<td>6</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>y2</td>
<td>3</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>y3</td>
<td>15</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>y4</td>
<td>8</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>y5</td>
<td>3</td>
<td>216</td>
<td></td>
</tr>
</tbody>
</table>

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

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

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

<table>
<thead>
<tr>
<th>project</th>
<th>bug type</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>expat</td>
<td>heap overflow</td>
<td>CVE-2019-15903</td>
</tr>
<tr>
<td>cJSON</td>
<td>NULL dereference</td>
<td>CVE-2019-1010239</td>
</tr>
<tr>
<td>mpc</td>
<td>stack overflow</td>
<td>reported to developer</td>
</tr>
<tr>
<td>mpc</td>
<td>heap overflow</td>
<td>reported to developer</td>
</tr>
</tbody>
</table>
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
## Executables in OSS-Fuzz

<table>
<thead>
<tr>
<th>Project</th>
<th>ID</th>
<th>Executable</th>
<th>FA</th>
</tr>
</thead>
<tbody>
<tr>
<td>c-ares</td>
<td>c1</td>
<td>ares_create_query_fuzzer</td>
<td>ares_create_query</td>
</tr>
<tr>
<td></td>
<td>c2</td>
<td>ares_parse_reply_fuzzer</td>
<td>ares_parse_1</td>
</tr>
<tr>
<td>expat</td>
<td>e1</td>
<td>parse_ISO_8859_1_fuzzer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>e2</td>
<td>parse_US_ASCII_fuzzer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>e3</td>
<td>parse_UTF_16BE_fuzzer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>e4</td>
<td>parse_UTF_16_fuzzer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>e5</td>
<td>parse_UTF_16LE_fuzzer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>e6</td>
<td>parse_UTF_8_fuzzer</td>
<td></td>
</tr>
<tr>
<td>boringssl</td>
<td>b1</td>
<td>bn_div</td>
<td></td>
</tr>
<tr>
<td></td>
<td>b2</td>
<td>bn_mod_exp</td>
<td></td>
</tr>
<tr>
<td></td>
<td>b3</td>
<td>client</td>
<td></td>
</tr>
<tr>
<td></td>
<td>b4</td>
<td>dtls_client</td>
<td></td>
</tr>
<tr>
<td></td>
<td>b5</td>
<td>dtls_server</td>
<td></td>
</tr>
<tr>
<td></td>
<td>b6</td>
<td>pkcs12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>b7</td>
<td>pkcs8</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Project</th>
<th>ID</th>
<th>Executable</th>
<th>FA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>b8</td>
<td>ares_create_query_fuzzer</td>
<td>BIO_new_mem_buf</td>
</tr>
<tr>
<td></td>
<td>b9</td>
<td>ares_parse_reply_fuzzer</td>
<td>CBS_init</td>
</tr>
<tr>
<td></td>
<td>b10</td>
<td>parse_ISO_8859_1_fuzzer</td>
<td>SSL_SESSION_from_bytes</td>
</tr>
<tr>
<td></td>
<td>b11</td>
<td>parse_US_ASCII_fuzzer</td>
<td>CBS_init</td>
</tr>
<tr>
<td></td>
<td>b12</td>
<td>parse_UTF_16BE_fuzzer</td>
<td></td>
</tr>
<tr>
<td>yara</td>
<td>y1</td>
<td>dex_fuzzer</td>
<td>yr_rules_scan_mem</td>
</tr>
<tr>
<td></td>
<td>y2</td>
<td>dotnet_fuzzer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>y3</td>
<td>elf_fuzzer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>y4</td>
<td>pe_fuzzer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>y5</td>
<td>rules_fuzzer</td>
<td>yr_compiler_add_string</td>
</tr>
</tbody>
</table>
# Executables by FuzzBuilder

<table>
<thead>
<tr>
<th>Project</th>
<th>ID</th>
<th>Unittest Executable</th>
<th>FA</th>
</tr>
</thead>
<tbody>
<tr>
<td>c-ares</td>
<td>fc1</td>
<td>ares_test</td>
<td>ares_create_query</td>
</tr>
<tr>
<td></td>
<td>fc2</td>
<td>ares_test</td>
<td>ares_parse.*</td>
</tr>
<tr>
<td>expat</td>
<td>fe1</td>
<td>runtests</td>
<td>XML_Parse</td>
</tr>
<tr>
<td>fb1</td>
<td></td>
<td>crypto_test</td>
<td>CBS_init</td>
</tr>
<tr>
<td>fb2</td>
<td></td>
<td>ssl_test</td>
<td>BIO_new_mem_buf</td>
</tr>
<tr>
<td>fb3</td>
<td></td>
<td>crypto_test</td>
<td>BIO_new_mem_buf</td>
</tr>
<tr>
<td>fb4</td>
<td></td>
<td>ssl_test</td>
<td>SSL_session_from_bytes</td>
</tr>
<tr>
<td>yara</td>
<td>fy1</td>
<td>test-api</td>
<td>yr_rules_scan_mem</td>
</tr>
<tr>
<td></td>
<td>fy2</td>
<td>test-api</td>
<td>yr_rules_scan_mem</td>
</tr>
</tbody>
</table>