CS412 Fuzzing Lab Demo Report

TAs

April 2025

Abstract

This is a simplified example report to give you an idea of what we expect you to do in this project. Note that we chose a very simple project along with simple improvements and shortened discussions. From your final report we expect more sophisticated changes and technical discussions. We will explicitly mark all places where we omit something (that you will need to include in your report).

1 Introduction

Students:

- TA1, 123456

For the fuzzing lab we chose the cJSON project `https://github.com/DaveGamble/cJSON`, a widely used JSON parsing library.

MISSING REPO: Please strictly follow the **Submission Format** we provided in assignment, else you will get **0 POINTS**. ◁

2 Part1

CJSON only has one harness (cjson_read_fuzzer). To run without seeds we modify the build script (fuzzing/ossfuzz.sh) and comment out the line that generates the seed corpus.

To build and run the cjson_read_fuzzer with seeds we run the following commands (note that all commands are run from the oss-fuzz root directory):

```
python3 infra/helper.py build_image  cjson
python3 infra/helper.py build_fuzzers cjson ../cJSON --clean
mkdir cjson_with_corpus
python3 infra/helper.py run_fuzzer cjson cjson_read_fuzzer --corpus
    -dir cjson_with_corpus
```

Then to run without:

```
cd ../cJSON && git apply cjson/fuzzing/cjson_reads_fuzzer_no_corpus
    .diff
python3 infra/helper.py build_fuzzers cjson ../cJSON --clean
```

```
mkdir cjson_without_corpus
python3 infra/helper.py run_fuzzer cjson cjson_read_fuzzer --corpus
    -dir cjson_without_corpus
```

The diff is located at ‘cjson/fuzzing/cjson_reads_fuzzer_no_corpus.diff‘. The diff:

```
    diff --git a/fuzzing/ossfuzz.sh b/fuzzing/ossfuzz.sh
index a2da64b..f5c1eae 100755
--- a/fuzzing/ossfuzz.sh
+++ b/fuzzing/ossfuzz.sh
@@ -3,7 +3,7 @@
-find $SRC/cjson/fuzzing/inputs -name "*" | \
-     xargs zip $OUT/cjson_read_fuzzer_seed_corpus.zip
+#find $SRC/cjson/fuzzing/inputs -name "*" | \
+#      xargs zip $OUT/cjson_read_fuzzer_seed_corpus.zip
```

Line coverage difference is only 15 lines of code between the run with and without seeds. The main difference is in the function ‘utf16_literal_to_utf8‘, where the fuzzer without initial seeds was unable to generate an input that passed the following check:

```
    if ((first_code >= 0xD800) && (first_code <= 0xDBFF))
```

The reason for the small difference in coverage is due to the fact that the harness uses a dictionary of JSON tokens, making it possible for the not-seeded fuzzer to generate valid json.

NOTE We only fuzzed for around a minute so this difference is likely spurious. ◁

MISSING REPO: We expect to be able to run the chosen harness with and without your changes out of the box with your provided run script. ◁

MISSING REPO: We expect to find the generated coverage html files for your fuzzing runs with and without seeds in your repository and the path to the html files here in the report. ◁

MISSING REPORT: We expect **AT LEAST one page double-column, or two page single-column** (including code listing and figures) detailed justification on the coverage report. You can explain 1) why both setups can cover some codes and 2) discuss the diff coverage, e.g., why one setup could cover them while another setup cannot. ◁

3 Part2

NOTE This example report only points out one uncovered region (but you'll need to identify two of them, uncovered by **oss-fuzz introspector**). ◁

3.1 Uncovered Region 1

The cJSON_Duplicate function and all it's dependencies (cJSON_strdup) are not covered by the existing fuzzer. The reason being is that the cjson_reads_fuzzer harness does not invoke the cJSON_Duplicate function. The function allows

the duplication of json objects and is an integral part of the cJSON api. The function and it's callees have about 100 lines of uncovered code (around 5% of the entire library).

MISSING REPORT: For **EACH** uncovered region, we expect **AT LEAST one-page single-column** explanation. You can talk about 1) the functionality of this region 2) whether this region is being used by other software 3) the reason that the region is not covered ... ◁

3.2 Uncovered Region 2

...

4 Part3

NOTE just as in the previous section, we only discuss one uncovered region but we expect you to discuss two and improve these regions in **two DIFFERENT WAYS**. ◁

4.1 Uncovered Region 1

We simply add a call to cJSON_Duplicate after the fuzzing json has been parsed succesfully, making sure to delete the duped json to avoid memory leaks.

```
    index aa9c7ba..9bc3dc8 100644
--- a/fuzzing/cjson_read_fuzzer.c
+++ b/fuzzing/cjson_read_fuzzer.c
@@ -35,6 +35,11 @@ int LLVMFuzzerTestOneInput(const uint8_t* data,
    size_t size)

     if(json == NULL) return 0;

+
+    cJSON* dup = cJSON_Duplicate(json, 1);
+
+    cJSON_Delete(dup);
+
     if(buffered)
     {
         printed_json = cJSON_PrintBuffered(json, 1, formatted);
```

As a result the overall line coverage jumps from 42 to 44 %. In the coverage report cJSON_Duplicate and cJSON_strdup are almost fully covered (except for some error handling cases).

MISSING REPO: We expect your improved fuzzer to be buildable and runnable out of the box via your provided run script! If not you will get 0 points for the (Repository) deliverables for this part. ◁

MISSING REPO: We expect the generated coverage html files in your repository and the path to the coverage html files in the report here. ◁

MISSING REPORT: We expect **AT LEAST one single-column page** justify the improvement and uncovered region, which is exactly the region you

mentioned in Part 2. Consider the following angles: 1) What's the key challenge for an unmodified setup/harness to cover this region? e.g., magic byte, wrong dependencies, API unreachable... 2) Justify how your modification addresses these challenges. ◁

4.2 Uncovered Region 2

...

5 Part4

We did not find a bug with our newly updated fuzzer so we will triage CVE-2024-31755. To triage we build the vulnerable version of CJSON (v1.7.18) and write a small poc that triggers the crash:

```
#include "cJSON.h"
int main(){
cJSON *item = cJSON_CreateString("apple");
cJSON_SetValuestring(item, NULL);
}
```

NOTE Being able to build a POC can be challenging and we hope to encourage you to fuzz for new bugs since that way you have your POC for free (running your harness with your crashing seed). ◁

MISSING REPO: We expect all files necessary to build and run the poc in your submission along with a script that allows us to build/run the POC ourselves to reproduce the crash. ◁

MISSING REPORT: consider using crbug.com/360265320 as an example on how to write Root Cause Analysis and Security Implication. We expect **AT LEAST one and half single-column pages** for this section ◁

5.1 Root Cause Analysis

The cJSON_SetValueString function does not check if the value string argument is null. This leads to a null pointer dereference in the call to strlen.

```
CJSON_PUBLIC(char*) cJSON_SetValuestring(cJSON *object, const char
    *valuestring)
{
    if (object->valuestring == NULL)
    {
        return NULL;
    }
    if (strlen(valuestring) <= strlen(object->valuestring)) // null
     pointer dereference happens here
    {
        strcpy(object->valuestring, valuestring);
        return object->valuestring;
    }
```

To fix the vulnerability, we propose the following fix (adding a NULL check to valuestring argument):

```
CJSON_PUBLIC(char*) cJSON_SetValuestring(cJSON *object, const char
    *valuestring)
{
    if (object->valuestring == NULL)
    {
        return NULL;
    }
    if (valuestring == NULL)
    {
        return NULL;
    }
    if (strlen(valuestring) <= strlen(object->valuestring)) // null
     pointer dereference happens here
    {
        strcpy(object->valuestring, valuestring);
        return object->valuestring;
    }
```

NOTE Root Cause analysis is **NOT** always equal to Crash Site Analysis. Take crbug.com/360265320 as an example, the crash site is the place where heap-buffer-overflow occurs, but the root cause is the integer overflow at line `fIndexCount += that->fIndexCount;`. If you do Crash Site Analysis, you will **NOT** get full points! ◁

5.2 Security Implication

The bug is a null pointer dereference and thus can only lead to a denial of service. It is questionable if the bug is really a bug in the library or simply incorrect usage of the library API.

NOTE This is an unfortunate example of a "bug", when choosing an old bug to triage make sure to choose a more interesting one. ◁