Graded lab assignment: Memory safety

1 Overview

In this exercise, you will analyze `miniunz`, a small utility that is part of of the compression library `zlib`.

Description: zlib 1.2.11 is a general purpose data compression library. All the code is thread safe. The data format used by the zlib library is described by RFCs (Request for Comments) 1950 to 1952 in the files http://tools.ietf.org/html/rfc1950 (zlib format), rfc1951 (deflate format) and rfc1952 (gzip format).

In the version you are given, a bug that has been introduced into the source code of `miniunz`. Your job is to diagnose the defect and to fix it. If you try `miniunz` on some examples, you will find out that the buggy version does not produce the correct output.

Note: Please do not download the source from the official repository, as it will not have the injected fault, so there would be not much to see.

1.1 Installation, running tests

1. Unpack the sources:
   ```
   tar -xzf zlib.tar.gz
   cd zlib/
   ```
2. Compile the sources with -g (debug information):
   ```
   ./configure && make
   ```
   It is better to re-compile `miniunz` with debug information after this:
   ```
   cd contrib/minizip/
   ```
   Inside that directory, add "-g" to "CFLAGS" in the Makefile, so it looks as follows:
   ```
   CFLAGS=-O -I../..  -g
   ```
   Re-compile miniunz:
   ```
   make
   ```
3. All your work will be done in that directory (`contrib/minizip/`). You can try to run the tool:
   ```
   ./miniunz -h
   ```

1.2 Valgrind usage

Note: If your code does not get re-compiled when it should be, use
```
make clean && make
```

Run valgrind on the binary with the example input:

```
valgrind ./miniunz -l mzm-bug.zip
```

You will see an error trace similar to this one:

```
==24814== Memcheck, a memory error detector
==24814== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==24814== Using Valgrind-3.14.0.GIT and LibVEX; rerun with -h for copyright info
==24814== Command: ./miniunz -l mzm-bug.zip
==24814==
--24814-- run: /usr/bin/dsymutil "./miniunz"
MiniUnz 1.01b, demo of zLib + Unz package written by Gilles Vollant
more info at http://www.winimage.com/zLibDll/unzip.html
mzm-bug.zip opened
  Length  Method     Size Ratio   Date    Time   CRC-32     Name
  ------  ------     ---- -----   ----    ----   ------     ----
error -103 with zipfile in unzGetCurrentFileInfo
==24814== Conditional jump or move depends on uninitialised value(s)
...
==24814== For counts of detected and suppressed errors, rerun with: -v
==24814== Use --track-origins=yes to see where uninitialised values come from
==24814== ERROR SUMMARY: 53 errors from 16 contexts (suppressed: 4 from 4)
```

2 Analyzing the error

When you encounter an error as shown above, you can see that valgrind suggests using an extra option to see where the error comes from: `--track-origins=yes`. Enable that option and run it again to see in which function the uninitialized memory was allocated. Use that information and the following hints to diagnose the error.

Use the provided example to reproduce the bug:

```
valgrind --track-origins=yes ./miniunz -l mzm-bug.zip
```

Deliverables: For the mandatory part, please carry out the following steps:

1. Defect report: Write a short report that describes what the valgrind trace means. Which data is uninitialized? Look at the definition of the data type (struct), and how it is being used. What happens if the data is not in the expected format?
2. Bug fix:
   (a) Create a copy of the source file you are going to modify.
   (b) Fix the program so valgrind reports less than 10 errors.
   (c) Upload a patch that fixes the program to Canvas.
   The output of valgrind should have `ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 4 from 4)` at the end, or any number < 10. The number of suppressed messages may vary. Memory leaks will still be reported in any case.

3 Optional assignment: perfect fix, fuzzing

3.1 Perfect fix

Fix the memory problem completely; you may have taken a route that fixes the bug partially but leaves some warnings. The "perfect" fix is very simple, so the easiest way to find it may be to revert the first attempt and look at what you can do if an entry in the ZIP archive is corrupted.

To get the extra point, your fix has to produce `ERROR SUMMARY: 0 errors from 0 contexts` by valgrind.

3.2 Fuzzing

Tasks:

1. Install radamsa from its home page: `https://github.com/aoh/radamsa`
   You do not need to run `make install`.
2. Set the environment variable RADAMSA so it points to the executable. Example:
   `export RADAMSA=${HOME}/dd2460/radamsa/bin/radamsa`
3. Create an example (a valid ZIP file) of how `miniunz` can be used. Let radamsa perturb it.

Hints:

1. Radamsa can fuzz an existing file with the following command:
   `${RADAMSA} example.zip > fuzzed.zip`
2. Run `valgrind` with `miniunz` on `fuzzed.zip`, until you see an error.

Deliverables:

For the extra point, you also have to submit a description of fuzzing:

1. How did you create the original example data?
2. How many attempts did it take to reproduce the problem?
3. Why do you think reproducing the problem with fuzzed data is easy or hard?

4 Grading criteria

Pass (G): The first part (mandatory assignment) is solved correctly.

VG (extra point): In addition to a correct and timely solution of the mandatory part, all optional parts solved correctly and submitted on time.