

# Lab 4: Delta Debugging

## Synopsis

Building a delta debugger for minimizing inputs that cause a program to crash — making it easier for the user to understand the bug.

## Objective

In this lab, you will build a delta debugger that implements an efficient algorithm for finding a 1-minimal crashing input given a large crashing input. You will combine this tool with a fuzzer like the one you built in **lab3** to minimize the crashing inputs found by the fuzzer.

## Setup

The code for Lab 4 is located under **cis547vm/lab4/**. We will frequently refer to this directory **lab4**. Open the **lab4** directory in VSCode following the Instructions from **Course VM document**.

This lab builds on top of the previous labs. We have provided you with pre-compiled binaries for the **runtime** library, **InstrumentPass** for **coverage** and **sanitize**, and a **fuzzer** executable; you can find them under **lab4/lib**. Their implementations are identical to the implementations in **lab3**.

### Step 1.

This lab uses python to implement delta debugger. We do so by building a python package called **delta\_debugger**.

To build and install the package, run:

```
/Lab4$ make install
```

Unlike with **c++**, you *won't* need to re-run this command after making changes to your code. Further, you will be able to use your delta debugger using the **delta-debugger** command from the terminal.

The **delta-debugger** tool performs delta debugging to shrink a crashing input to a program.

### Step 2.

To use **delta-debugger** with a program you first need to find some input that will crash the program. To find such an input we will use a fuzzer.

Just like **lab3**, to run the **fuzzer** you will first need to instrument the program and setup appropriate output directories where fuzzer will store its results.

```
/lab4/test$ make sanity1           # Instrument and build sanity1
/lab4/test$ mkdir fuzz_output_sanity1 # Create output directory
# Run the fuzzer on sanity1 with a timeout of 6 seconds.
/lab4/test$ timeout 6s fuzzer ./sanity1 fuzz_input fuzz_output_sanity1
```

You can also use the Makefile to instrument, build, setup output directory and run the fuzzer for you:

```
/lab4/test$ make sanity1           # Instrument and build sanity1
/lab4/test$ make fuzz-sanity1      # Run the fuzzer on sanity1
```

### Step 3.

Once you have run the fuzzer you will find inputs that couse the program to crash under **test/fuzz\_output\_sanity1/failure**.

```
fuzz_output_sanity1
├─ success
├─ randomSeed.txt
└─ failure # Inputs that cause a crash.
    ├── input0
    ├── input1
    ├── ...
    └─ inputN
```

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorocs

You can now use **delta-debugger** to minimize the crashing inputs found by the fuzzer.

```
/lab4/test$ delta-debugger ./sanity1 fuzz_output_sanity1/failure/input1
```

The last argument is path to the crashing input and depends on which input you want to minimize. In this example the reduced input is stored in **fuzz\_output/failure/input1.delta**. Additionally, before running another invocation of **delta-debugger**, make sure to clean up the **fuzz\_output** directory.

You can do this by running:

```
/lab4/test$ rm -rf fuzz_output_sanity1 && mkdir fuzz_output_sanity1
```

## Lab Instructions

You will need to edit the **lab4/delta\_debugger/delta.py** file to build a delta debugging tool. We have provided a template function — **delta\_debug** — for you to implement your minimization logic. The **delta\_debug** function takes a **target** program, and **input** that causes **target** to crash, and is supposed to return a 1-minimal input that still crashes the **target** program.

To perform delta debugging, you will have to repeatedly run **target** with various **input** strings. We provide a **run\_target** function to help you run **target** program with an **input**. It returns a value of 0 if the target didn't crash.

```
def run_target(target: str, input: Union[str, bytes]) -> int:
    """
    Run the target program with input on its stdin.
    :param target: The target program to run.
    :param input: The input to pass to the target program.
    :return: The return code of the target program.
    """
    ...
```

For this lab you will modify the **delta\_debug** function to implement the algorithm to you learn in class to find a 1-minimal crashing input.

You likely want to add a helper function for example called **\_delta\_debug** that takes a **target**, an **input** and a parameter **n** that correspond to search granularity, and performs one iteration of delta debugging algorithm to return the next **input** and **n**.

## Example Input and Output

Your delta debugger should run on any executable that accepts input from **stdin**.

You run the delta debugger on a test program by passing in the following arguments:

```
delta-debugger ./test crashing-input
```

And the delta debugger will store its result in **crashing-input.delta** file.

As a specific example consider the string: "abckdanmvelcbaghcajbtckxmntplwqsrakstuvbxyz", which causes **test3** to fail:

```
/lab4/test$ echo -n "abckdanmvelcbaghcajbtckxmntplwqsrakstuvbxyz" > tmp
/lab4/test$ delta-debugger ./test3 tmp
/lab4/test$ cat tmp.delta
abckdanmvel
```

## Items to Submit

Once you are done with the lab, you can create a **submission.zip** file by using the following command:

```
lab4$ make submit
...
submission.zip created successfully.
```

Then upload the **submission.zip** file to Gradescope.