CS510'15 Project #1 Address Sanitizer
(Due: Feb-03, 2015)

January 22, 2015

1 Goal

In this project you will learn to use Address Sanitizer to find bugs in a program.

2 Introduction to Address Sanitizer

Address Sanitizer (ASan) is a tool for finding memory errors (bugs) in C/C++ programming. We're going to be using ASan as part of the clang compiler. There is a lot of information on ASan online (Google "Address Sanitizer"). This section is only the bare minimum you need to get up and running with ASan. If you can't reproduce this whole section on your machine after a day of trying, go see the TA.

First you need a newer version of clang install on your system. Any decent Linux distribution should have version 3.4, 3.5, or 3.6 available in its package manager. For Ubuntu you install clang by running:

```
$ sudo apt-get install clang llvm
```

You need a program with a memory error to test. Make a file called ex1.c with the following contents:

```
#include <stdlib.h>
int main(int argc, char **argv) {
  int *array = malloc(sizeof(int)*100);
  free(array);
  return array[argc];  // BOOM
}
```

Now compile the program by invoking clang:

```
clang -g -fsanitize=address ex1.c -o ex1
```

The argument -g tells clang to create debugging symbols, which you need to see that names of the variables in the ASan output. The argument -fsantizie=address turned on ASan. Without that argument clang just compiles the normal program.

To analyze the program, you need to run it:

```
./ex1
```

You should get something almost identical to the following output (except of course your path names should be different):

```
=================================================================
==4892==ERROR: AddressSanitizer: heap-use-after-free on address 0
    x61400000fe44 at pc 0x00000049aa1b bp 0x7fffd89da530 sp 0
    x7fffd89da528
READ of size 4 at 0x61400000fe44 thread T0
    #0 0x49aa1a in main /home/carr27/cs510class/projects/1/ex1.c
    :5:3
    #1 0x7fa201cbfec4 in __libc_start_main /build/buildd/eglibc
    -2.19/csu/libc-start.c:287
    #2 0x417a86 in _start (/home/carr27/cs510class/projects/1/a.out
    +0x417a86)

0x61400000fe44 is located 4 bytes inside of 400-byte region [0
    x61400000fe40,0x61400000ffd0)
freed by thread T0 here:
    #0 0x47b089 in free /home/carr27/llvm/projects/compiler-rt/lib/
    asan/asan_malloc_linux.cc:30:3
    #1 0x49a949 in main /home/carr27/cs510class/projects/1/ex1.c
    :4:3
    #2 0x7fa201cbfec4 in __libc_start_main /build/buildd/eglibc
    -2.19/csu/libc-start.c:287

previously allocated by thread T0 here:
    #0 0x47b349 in malloc /home/carr27/llvm/projects/compiler-rt/
    lib/asan/asan_malloc_linux.cc:40:3
    #1 0x49a90c in main /home/carr27/cs510class/projects/1/ex1.c
    :3:16
    #2 0x7fa201cbfec4 in __libc_start_main /build/buildd/eglibc
    -2.19/csu/libc-start.c:287

SUMMARY: AddressSanitizer: heap-use-after-free /home/carr27/
    cs510class/projects/1/ex1.c:5 main
Shadow bytes around the buggy address:
  0x0c287fff9f70: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c287fff9f80: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c287fff9f90: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c287fff9fa0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c287fff9fb0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x0c287fff9fc0: fa fa fa fa fa fa fa fa[fd]fd fd fd fd fd fd fd
  0x0c287fff9fd0: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c287fff9fe0: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c287fff9ff0: fd fd fd fd fd fd fd fd fd fd fa fa fa fa fa fa
  0x0c287fffa000: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c287fffa010: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes)
    :
   Addressable:           00
   Partially addressable: 01 02 03 04 05 06 07
   Heap left redzone:       fa
   Heap right redzone:      fb
   Freed heap region:       fd
   Stack left redzone:      f1
   Stack mid redzone:       f2
   Stack right redzone:     f3
```

```
  Stack partial redzone:   f4
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
==4892==ABORTING
```

If you can't get output like this after a reasonable effort, contact the TA.

3 ASan Limitations

Memory errors are fatal. ASan does not stop memory errors from crashing the program. Often the programmer must use ASan iteratively. The first time the programmer runs ASan it might return some bugs but when the programmer fixes those rerunning ASan might find even more bugs. Hint: you will need to repeat this process multiple times to complete the project.

4 What you will turn in

You will be turning in everything via a BitBucket git repository. If you didn't get an email from BitBucket with a link to a repository called csStudentXX, make a BitBucket account (using you're Purdue email address) and email your username to the TA. You can't do anything without cloning the repository first. If you can't figure out how to clone the repository contact the TA. After cloning the repository there should be three files in your directory under project1:

1. bzip2.c
2. project1.txt
3. big.txt

There is a template file in each of your repositories called project1.txt. You will fill in the answers to all the questions in that file. When you make changes to the file you should commit them using git, then push your changes to the bitbucket server (also using git). If you don't know how to use git this is a great time to learn! You will not need to be a git master, but only need the basic commands. See the TA if you're struggling.

Also in your repository is the file you'll be analyzing: bzip2.c. The goal of the project is to find all 10 bugs in bzip2.c using ASan. For extra credit you

may create a copy of bzip2.c called fixed.c and add it to your repository. To receive extra credit you must fix all the bugs ASan finds and explain how you fixed them. You may not simply delete the buggy code. The fixed version has to have all the functionality of the original version.

The file big.txt is just an example file you can input to bzip2.

You can double check that all your changes will be seen by the TA by going to the BitBucket website and browsing your repository's source. The TA cannot see anything that doesn't show up there.

5 Git and BitBucket

This section is not a substitute for reading a real git tutorial. This is the bare minimum you need for this project only.

To clone your repository (but change the XX part to the name of your repo):

```
git clone git@bitbucket.org:scottcarr/cs510studentXX.git
```

That should make a cs510studentXX directory on your local machine. Inside that directory should be another directory called project1. The rest of the commands must be run from inside your cs510studentXX/project1 directory.

To commit the changes you made:

```
git commit -am "fixed line 42"
```

To push (upload) your changes to the server

```
git push
```

If you're going for the extra credit you can Google how to add a file to a git repository.

You can commit/push as many times as you want. The TA will only grade the final version in BitBucket.

6 About bzip2.c

bzip2.c is an implementation of the Linux command bzip2. The project will be much easier if you have a basic understand of what it does and the command line options it excepts. You can find that information on Google or just "man bunzip2" if you're on a Linux machine.