

# **Detecting Memory Errors Using Address Sanitizer**

October 2022

Rudranshu Datta

This guide will take you through running your code on the ECE Linux server and explain how to check for memory errors.

## Table of Contents

Set-Up .....	3
VPN.....	3
SSH into the servers .....	3
Moving your files over .....	4
Running your project with leak checker .....	5
Running your code on the Linux server .....	5
Notes.....	7

## Set-Up

As part of your project requirements, you are required to resolve memory errors, such as memory leaks and referencing something that has previously been freed (dangling pointer). You can check for memory errors by using an open-source tool called [Address Sanitizer](#).

This tool is available for you to run on the ECE Linux servers. These servers are pre-configured with most of the tools you will need for all of your classes at Georgia Tech.

For this class, we will be using the remote linux server linlabsrv01. This guide was written using a Windows machine running WSL2 with Ubuntu and remotely accessing this ECE Linux server.

## VPN

You will need the VPN to connect to the ECE servers.

Download the GT VPN if you don't have it already. You can use this [website](#) to download it.

## SSH into the servers

Before we get started, everyone in this class should have access to the servers. To check if you have access and can log in to the servers, go to your terminal application (e.g., the one that opens when running WSL2 with Ubuntu) and do the following:

```
>> ssh your\_username@ece-linlabsrv01.ece.gatech.edu
```

It is normal for it to ask if you want to add this host to your list of trusted servers, and assuming you typed the host name correctly, you should answer yes. When your password is requested, use your normal password for GT single-signon. There may be a delay before the server replies with a screenful of text.

If you cannot ssh into the server or get a permission issues, contact ECE CSG Help to activate your account. Their email is [help@ece.gatech.edu](mailto:help@ece.gatech.edu). But if you have a more basic problem (e.g., your terminal application says that there's no ssh command, etc.), ask a ECE 2035 TA.

After you successfully ssh into your linux machine you should have something like this image below. But depending on user preferences, the command line prompt at the bottom may differ.

```
(this is NOT the same as Debian/Ubuntu)
```

**NOTICE:**

```
/usr/scratch on this machine is intended as temporary  
space and is not backed up; do not store anything  
critical here as we will not attempt to restore it  
in case of a crash or reinstall. Files in  
/usr/scratch and /tmp that are older than 2 weeks  
will be automatically purged on all publicly  
available systems  
(https://help.ece.gatech.edu/labs/names).
```

```
Jobs are limited to one interactive or background  
process per login on each public computation server.  
Accounts running more than one job per machine will  
lose login privileges for all ecelinsrv systems for  
two weeks. Jobs running for longer than 5 days on the  
publically available computation systems will be  
terminated at the discretion of the CSG. If you need  
to run substantially long jobs, email help@ece  
requesting a PACE account.
```

```
If you are having problems running applications, see  
https://help.ece.gatech.edu/linux/shells for  
information on configuring your login environment.
```

```
[rdatta34@ece-linlabsrv01 ~]$ |
```

## Moving your files over

If you have never used this server before, you may have no files in your home directory. In order to proceed, you need to transfer some source code files, which could either be a current project which needs debugged, or an example project, such as the one that we will use here.

Go ahead and create a working directory on the server, calling it Sanitizer if you want your setup to resemble the example that follows.

Download the example source files here [add link to whatever code we provide], which are based on the examples given in the “Common Memory Errors” lecture. Since you will probably be downloading to your local computer, you will need to transfer them to the ECE Linux server that

To move files over, I recommend using [Filezilla](#). Get the simplest, free version, **but be careful when installing, because it will default to installing AVG or other add-ons.**

This official [documentation](#) is pretty good. But TL;DR, in the top line of Filezilla, enter these values and press the Quick Connect button:

Host: ece-linlabsrv01.ece.gatech.edu

Username: Your GT Username

Password: Your Password

Port: 22

You will be presented with a window interface showing your local files (Windows) on the left, and those on the ECE Linux server on the right. (You will see hidden Linux files and folders whose names begin with a period. This is normal.) Navigate to where you downloaded the example files on the left, and navigate to your Sanitizer directory on the right. Those two folders should be highlighted in the top left and top right menus, and the result will be that you see the files in each folder in the window panes just below that.

Drag the source files (dll.c and Makefile) from the left to the right.

You can view files in Linux by using the 'ls' command and navigate to them using the cd command. At this point, you probably only need to do

```
>> cd Sanitizer
```

## Running your project with leak checker

For your project we will be utilizing [Address Sanitizer](#). It is a great built in tool in gcc for checking for memory and stack/heap errors. Address sanitizer libraries should already be preinstalled in gcc.

## Running your code on the Linux server

Once you have moved your project folder over using your preferred method, we can continue with the rest of this process.

Step 1: If you followed the steps above, you are already in the correct folder on your ECE Linux server terminal window (where your .c and Makefile are).

Step 2: You can use 'ls' to list files and cd to navigate. For me the file path looks like this:

```
tc3@ece-linlabsrv01.ece.gatech.edu>ls
dll.c  Makefile
tc3@ece-linlabsrv01.ece.gatech.edu>_
```

Your path may be different, depending on your file structure.

Step 3: Do a 'make clean'. This will remove all the extra files that you may had and all the object files. It is a good practice to do 'make clean' after you change something before you run make.

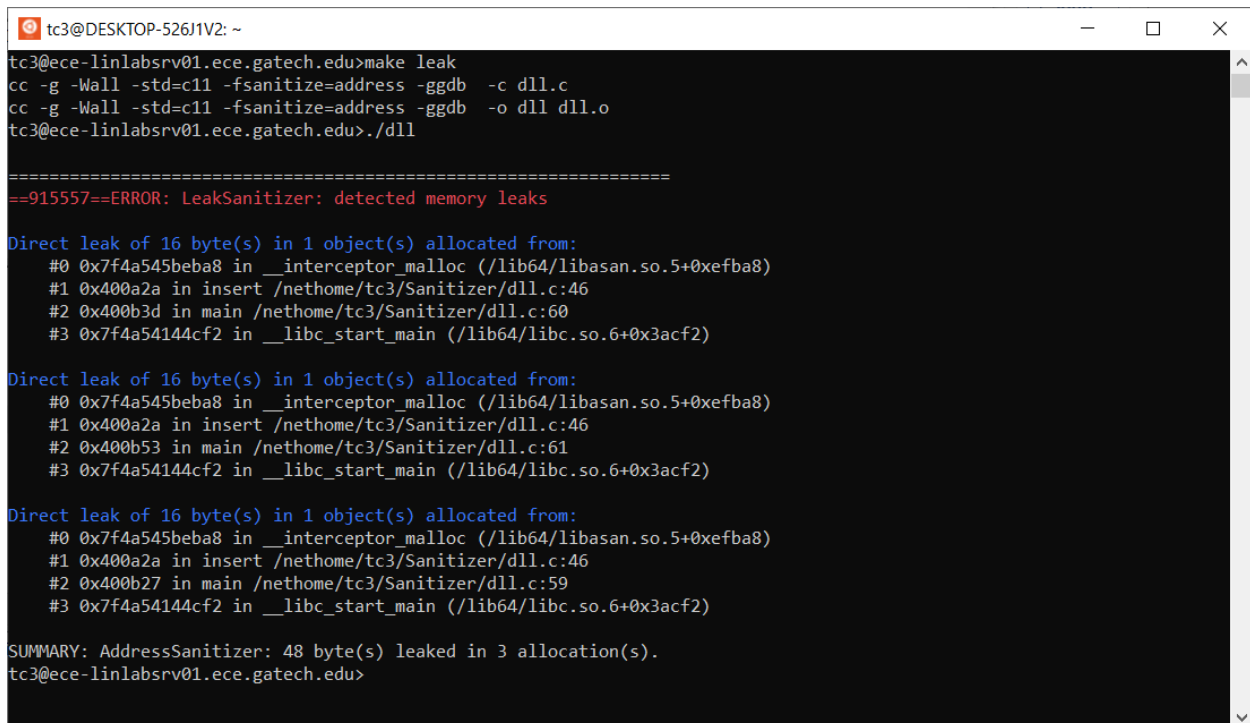
Step 4: Once make clean is done. We can test our code.

First, run 'make'. This code has a bug (as seen in the relevant part of the lecture), but it causes no compiler errors, and you can even run it successfully:

```
>> ./dll
```

We are also using the Google Test framework to detect errors such as the one in this code, and it would always be good practice to fix all tests before you do a memory check. But this code does not have any tests built in, so we will go straight to a memory leak test using Address Sanitizer.

To do so, first do a 'make clean', and then do a 'make leak', and finally run the new executable program by entering './dll'. This enables the address sanitizer and performs a memory check as the program executes.

A terminal window titled 'tc3@DESKTOP-526J1V2: ~' showing the execution of a program with AddressSanitizer. The user runs 'make leak' and then './dll'. The output shows three direct memory leaks of 16 bytes each, all originating from 'insert' in 'dll.c' at line 46. The stack traces for each leak show the path from the main function through the libc start to the program's main function, then to the 'insert' function. A summary at the bottom states that 48 bytes were leaked in 3 allocations.

```
tc3@ece-linlabsrv01.ece.gatech.edu>make leak
cc -g -Wall -std=c11 -fsanitize=address -ggdb -c dll.c
cc -g -Wall -std=c11 -fsanitize=address -ggdb -o dll dll.o
tc3@ece-linlabsrv01.ece.gatech.edu>./dll

=====
==915557==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 16 byte(s) in 1 object(s) allocated from:
#0 0x7f4a545beba8 in __interceptor_malloc (/lib64/libasan.so.5+0xefba8)
#1 0x400a2a in insert /nethome/tc3/Sanitizer/dll.c:46
#2 0x400b3d in main /nethome/tc3/Sanitizer/dll.c:60
#3 0x7f4a54144cf2 in __libc_start_main (/lib64/libc.so.6+0x3acf2)

Direct leak of 16 byte(s) in 1 object(s) allocated from:
#0 0x7f4a545beba8 in __interceptor_malloc (/lib64/libasan.so.5+0xefba8)
#1 0x400a2a in insert /nethome/tc3/Sanitizer/dll.c:46
#2 0x400b53 in main /nethome/tc3/Sanitizer/dll.c:61
#3 0x7f4a54144cf2 in __libc_start_main (/lib64/libc.so.6+0x3acf2)

Direct leak of 16 byte(s) in 1 object(s) allocated from:
#0 0x7f4a545beba8 in __interceptor_malloc (/lib64/libasan.so.5+0xefba8)
#1 0x400a2a in insert /nethome/tc3/Sanitizer/dll.c:46
#2 0x400b27 in main /nethome/tc3/Sanitizer/dll.c:59
#3 0x7f4a54144cf2 in __libc_start_main (/lib64/libc.so.6+0x3acf2)

SUMMARY: AddressSanitizer: 48 byte(s) leaked in 3 allocation(s).
tc3@ece-linlabsrv01.ece.gatech.edu>
```

You can see that three errors are detected, related to each insertion of a node, and resulting from the error discussed in the lecture.

#### Step 5: Resolve the memory leak

This step is the toughest bit. To find the cause behind your memory leak, you have to go through the stack trace. This becomes more complicated with larger programs, especially with the Gtest framework adding more information to the stack traces. But here, it is relatively simple.

In the stack trace, the last called function is at the top of the log. Reading the stack trace, we know that the leak was triggered by line 46 in dll.cc, where malloc was called by the insert function. That was where memory was allocated and never freed.

## Notes

1. If you are familiar with running the X windowing environment on Linux and are able to set up an X server on your system, you can edit directly on the linux machines using FastX or MobaXterm. If you navigate to your project directory and type 'code .' in the terminal, it should open up a vscode window for you to edit on.
2. Alternatively, you can run tests on your local machine as well. If you are using WSL2, it should work the same way as running it on the linux server. On MacOS, some changes may be required before you can run leak checker on your local machine. Everything else should still work on your local machines on all platforms if you do not run address sanitizer using the 'make leak' command.
3. I recommend that you test on your local machine first before uploading to the linux servers if possible. This will save you a lot of time and frustration.
4. Going through stack traces can be difficult and the line numbers may not always be totally accurate but they put you in the ballpark to start debugging your code.