

دانشگاه علم و صنعت

دانشکده مهندسی کامپیوتر

درجه تحصیلی: کارشناسی

تکلیف6 OS

گردآورنده:

پرنیان شاکریان - 99400064

استاد:

دکتر انتظاری

سال تحصیلی: خرداد 1402

Question 1: Segmentation Fault

هنگامی که برنامه را با روشی که در داک توضیح داده شد اجرا میکنیم، عملا اشاره‌گر null ptr را ارجاع نداده‌ایم. در اصل ما سعی میکنیم به مقدار آدرس حافظه که توسط pointer اشاره شده دسترسی پیدا کنیم اما از آنجایی که اشاره گر NULL است، به یک مکان حافظه معتبر اشاره نمیکند و همین عدم ارجاع منجر به رفتار undefined میشود. دلیلی که این اتفاق می افتد این است که سیستم عامل دسترسی غیرقانونی به حافظه را شناسایی کرده و برنامه را خاتمه می دهد تا از ایجاد مشکلات بیشتر و یا از دسترسی به حافظه محافظت شده جلوگیری کند. بنابراین، برنامه با خطایsegmentation fault از کار می افتد.

با گنجاندن گزینه g- در دستور (gcc -g -o null null.c)، به کامپایلر دستور می دهیم تا یک فایل اجرایی با نمادهای debugging تولید و اطلاعات debugging را در برنامه کامپایل شده اضافه کند. هنگامی که برنامه‌ را با g- کامپایل میکنیم، به کامپایلر این دستور را میدهد تا نمادهای debugging را در فایل اجرایی تولید شده قرار دهد. وجود این نمادها به ویژه هنگام debugging برنامه مانند gdb (GNU Debugger) بسیار مفید است. با آنها میتوانیم کد را بررسی کنیم، breakpoint تعیین کنیم، متغیرها را بررسی کنیم و سایر کارها را به طور موثرتری انجام دهیم. این نمادها حاوی اطلاعاتی هستند که میتواند در debugging برنامه بسیار موثر واقع میشود. برخی از این اطلاعات عبارتند از:

1. Symbol table: شامل نام و آدرس متغیرها، توابع و سایر نمادها در برنامه است. این اطلاعات در حین debugging به ارتباط آدرس های حافظه با متغیرها یا توابع مربوطه کمک میکند.
2. اطلاعات Source file: شامل اطلاعاتی از منبع اصلی مانند نام فایل، شماره خطوط و حتی خود کد اصلی است که به mapping آن در حین debugging کمک میکند.
3. نام توابع و متغیرها: نمادها نام اصلی توابع و متغیرها را حفظ کرده و به دیباگرها اجازه می دهند نام های معنی دار را به جای آدرس های حافظه نمایش دهند.

وقتی برنامه را با gdbاجرا و run را در خط فرمان gdb (GNU Debugger) تایپ میکنیم، gdb، اطلاعات مختلف debugging را در اختیار ما قرار میدهد. در برنامه به دلیل عدم ارجاع به یک اشاره‌گر NULL با یک خطای segmentation مواجه می شویم. gdb سیگنال را گرفته و اطلاعاتی در مورد خطا، از جمله نام سیگنال (SIGSEGV)، آدرس حافظه (0x0000000000400537) و همچنین خط کد (null.c:6) را که در آن خطا رخ داده است ارائه میدهد. توجه شود که خروجی خاص ممکن است بر اساس پیکربندی سیستم، کامپایلر و دیباگر متفاوت باشد.

GNU gdb (GNU/Linux 5.15.90.1-microsoft-standard-WSL2 x86\_64)

Copyright (C) 2020 Free Software Foundation, Inc.

License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software: you are free to change and redistribute it.

There is NO WARRANTY, to the extent permitted by law.

Type "show copying" and "show warranty" for details.

This GDB was configured as "x86\_64-gnu-linux".

Type "show configuration" for configuration details.

For help, type "help".

For bug reporting instructions, please see:

Type "apropos word" to search for commands related to "word"...

Reading symbols from null...

(gdb) run

Starting program: /home/parniansh/null

Program received signal SIGSEGV, Segmentation fault.

0x0000000000400537 in main () at null.c:6

6 printf("%d\n", \*ptr);

(gdb) quit

A debugging session is active.

Inferior 1 [process 6122] will be killed.

اجرای برنامه با gdb از این جهات به ما کمک میکند:

1. Debugging errors: gdb اطلاعاتی در مورد محل خطا و وضعیت برنامه در آن نقطه ارائه میدهد که تشخیص و رفع آن را آسان تر می کند و به ما اجازه میدهد تا خطاهای برنامه را debug کنیم. مانند segmentation faults، نقض دسترسی به حافظه و غیره.
2. بررسی وضعیت برنامه: با gdb میتوانیم مقادیر متغیرها، data structures و محتویات حافظه را در طول اجرای برنامه بررسی کنیم که به درک نحوه رفتار برنامه و تعامل متغیرهای مختلف با یکدیگر کمک می کند.
3. آزمایش فرضیه ها: با دنبال کردن برنامه و بررسی متغیرها، می توانیم فرضیه های خود را در مورد رفتار برنامه آزمایش کرده و صحت آنها را بررسی کنیم که به درک و عیب‌یابی خروجی غیرمنتظره کمک می کند.
4. پروفایل سازی و تجزیه و تحلیل عملکرد: gdb به ما این امکان را میدهد تا زمان اجرا را اندازه گیری، استفاده از منابع را تجزیه و تحلیل، و نقطه کور عملکرد را شناسایی کنیم.

وقتی دستور (valgrind--tool=memcheck--leak-check=yes./null) را اجرا میکنیم،Valgrind Memcheck ، میزان استفاده از حافظه برنامه را تحلیل کرده و خطاها و leaks حافظه را بررسی میکند. هنگام اجرای این دستور، خروجی را میتوان به صورت زیر تفسیر کرد:

1. Valgrind: مقداردهی اولیه میکند و اطلاعاتی در مورد نسخه مورد استفاده ارائه می دهد.
2. Invalid read of size 4: نشان میدهد که یک حافظه نامعتبر در اندازه 4 خوانده شده است.
3. Address 0x0 is not stack'd, malloc'd or (recently) free'd: به این معنی که تلاشی صورت گرفته تا از آدرس حافظه 0x0 که یک اشاره گر NULL است خوانده شود.
4. HEAP SUMMARY: نشان میدهد که هیچ حافظه ای در طول اجرای برنامه تخصیص داده نشده، لو رفته یا نشت حافظه وجود نداشته است.
5. All heap blocks were freed: تایید میکند که هیچ نشت حافظه ای در برنامه وجود ندارد.
6. ERROR SUMMARY: نشان میدهد که 1 خطا حافظه توسط Valgrind شناسایی شده.

==11118== Memcheck, a memory error detector

==11118== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.

==11118== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info

==11118== Command: ./null

==11118==

==11118== Invalid read of size 4

==11118== at 0x114093: main (null.c:6)

==11118== Address 0x0 is not stack'd, malloc'd or (recently) free'd

==11118==

==11118==

==11118== Process terminating with default action of signal 11 (SIGSEGV)

==11118== Access not within mapped region at address 0x0

==11118== at 0x114093: main (null.c:6)

==11118== If you believe this happened as a result of a stack

==11118== overflow in your program's main thread (unlikely but

==11118== possible), you can try to increase the size of the

==11118== main thread stack using the --main-stacksize= flag.

==11118== The main thread stack size used in this run was 8103118.

==11118==

==11118== HEAP SUMMARY:

==11118== in use at exit: 0 bytes in 0 blocks

==11118== total heap usage: 0 allocs, 0 frees, 0 bytes allocated

==11118==

==11118== All heap blocks were freed -- no leaks are possible

==11118==

==11118== For lists of detected and suppressed errors, rerun with: -s

==11118== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)

Question 2: Memory Leak

طبق خواسته داک ابتدا باید کد مورد نظر را با توجه به مراحل زده و کامپایل کنیم. بعد از آن باید به سه روش مختلف آن را run کرده و در هر روش بررسی کنیم که خروجی ها و نحوه نشان دادن آنها به چه صورت میباشد. به طور خلاصه میتوانیم متوجه شویم که برنامه ما مشکل memory leak دارد. هر روش run این مسئله را بیان میکند اما میزان دقت و جزئیاتی که نشان میدهند با هم فرق دارند.

وقتی برنامه را نرمال run میکنیم، با استفاده از تابع malloc حافظه را به صورت پویا اختصاص میدهد اما قبل از خروج آن حافظه را آزاد نمیکند. این به عنوان نشت حافظه یا memory leak شناخته میشود. نشت حافظه یعنی حافظه اختصاص داده شده توسط سیستم پس از عدم نیاز به درستی آزاد نشده و دوباره به سیستم بازگردانده نمیشود. برنامه‌های به این شکل میتواند به اتلاف قابل توجهی از منابع حافظه در طول زمان منجر شود. در اینجا ترمینال بلافاصله پس از اجرای برنامه بدون نمایش اطلاعات اضافی به خط فرمان باز میگردد.

$ ./leak

$

وقتی برنامه را با gdb run میکنیم، شروع به اجرا و تخصیص حافظه با استفاده از malloc میکند، سپس به عبارت (0) exit رسیده و خاتمه می‌یابد. پیام آخر خروجی نشان میدهد که برنامه با موفقیت اجرا و کد با وضعیت 0 که رفتار مورد انتظار است، خارج میشود. با این حال، لازم به ذکر است، اجرای برنامه تنها با gdb به طور مستقیم مشکل نشت حافظه را برجسته نمیکند. خروجی ای که به دست آوردیم نشان میدهد برنامه بدون خطا اجرا شده، اما اطلاعات خاصی در مورد نشت حافظه نمیدهد.

GNU gdb (GNU/Linux 5.15.90.1-microsoft-standard-WSL2 x86\_64)

Copyright (C) 2020 Free Software Foundation, Inc.

License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software: you are free to change and redistribute it.

There is NO WARRANTY, to the extent permitted by law.

Type "show copying" and "show warranty" for details.

This GDB was configured as "x86\_64-gnu-linux".

Type "show configuration" for configuration details.

For help, type "help".

For bug reporting instructions, please see:

Type "apropos word" to search for commands related to "word"...

Reading symbols from leak...

(gdb) run

Starting program: /home/parniansh/leak

[Inferior 1 (process 10913) exited normally]

وقتی برنامه را Valgrind run میکنیم درخروجی، کل استفاده از heap را گزارش می‌کند که نشان می‌دهد 2 بایت تخصیص داده شده اما آزاد نشده است. همچنین جزئیات مربوط به memory leak، از جمله مکانی که در کد حافظه به آن تخصیص داده شده است (leak.c:4) ارائه میکند. در بخش «LEAK SUMMARY» تعداد بایت‌های قطعی از دست رفته، بایت‌های گمشده غیرمستقیم، بایت‌های احتمالاً از دست رفته و بایت‌های هنوز قابل دسترسی مشخص میشود. در این حالت، قسمت " definitely lost" میزان حافظه leak شده را نشان می دهد.

==9706== Memcheck, a memory error detector

==9706== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.

==9706== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info

==9706== Command: ./ leak

==9706==

==9706==

==9706== HEAP SUMMARY:

==9706== in use at exit: 2 bytes in 1 blocks

==9706== total heap usage: 1 allocs, 0 frees, 2 bytes allocated

==9706==

==9706== 2 bytes in 1 blocks are definitely lost in loss record 1 of 1

==9706== at 0x441B1F2: malloc (in /usr/lib/x86\_64-linux gnu/valgrind/vgpreload\_memcheck-amd64-linux.so)

==9706== by 0x16322E: main (leak.c:4)

==9706==

==9706== LEAK SUMMARY:

==9706== definitely lost: 2 bytes in 1 blocks

==9706== indirectly lost: 0 bytes in 0 blocks

==9706== possibly lost: 0 bytes in 0 blocks

==9706== still reachable: 0 bytes in 0 blocks

==9706== suppressed: 0 bytes in 0 blocks

==9706==

==9706== For lists of detected and suppressed errors, rerun with: -s

==9706== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)

Question 3: Index Out of Range

در ابتدا با توجه به خواسته داک، کد ساده برنامه رو میزنیم. وقتی که برنامه را اجرا میکنیم، رفتار undefined از خودش نشان میدهد زیرا ما به عنصر آرایه، خارج از محدوده آن دسترسی میدهیم. در زبان C، index های آرایه از 0 شروع میشود، بنابراین وقتی آرایه data را روی اندازه 100 اعلام میکنیم، به این معنی است که میتواند اعداد صحیح را از داده [0] تا [99] ذخیره کند. با این حال، در data[100] = 99400064 ما در حال تلاش برای دسترسی و اختصاص مقداری به [100] data هستیم که خارج از محدوده آرایه است. این نوع دسترسی نامعتبر بوده و میتواند منجر به crash یا خطای تقسیم‌بندی شود. در اینجور مواقع رفتار برنامه تضمین نشده و نتیجه ممکن است بسته به کامپایلر و سیستم خاص متفاوت باشد. در کیس ما خروجی نشان میدهد که برنامه اجرا شده. لطفا توجه شود که عدم وجود پیام خطا یا crash به معنای صحیح بودن برنامه یا اجرای همیشه بدون مشکل آن نیست بلکه بسته به عوامل مختلفی میتواند متفاوت باشد. ما دیباگر (GDB) را در سیستم اوبونتو خود راه اندازی کرده‌ایم. برای استفاده از GDB، دستورات را در اعلان GDB وارد میکنیم تا مراحل debug کردن را کنترل کنیم.

GNU gdb (GNU/Linux 5.15.90.1-microsoft-standard-WSL2 x86\_64)

Copyright (C) 2020 Free Software Foundation, Inc.

License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software: you are free to change and redistribute it.

There is NO WARRANTY, to the extent permitted by law.

Type "show copying" and "show warranty" for details.

This GDB was configured as "x86\_64-gnu-linux".

Type "show configuration" for configuration details.

For help, type "help".

For bug reporting instructions, please see:

Type "apropos word" to search for commands related to "word"...

Reading symbols from index...

(gdb) run

Starting program: /home/parniansh/index

data[100] = 99400064

[Inferior 1 (process 12208) exited normally]

وقتی برنامه را با Valgrind اجرا میکنیم، میتواند دسترسی خارج از محدوده حافظه را تشخیص داده و آن را به عنوان خطا گزارش کند. پیام خطای Valgrind به ما کمک می‌کند تا مشکل را شناسایی کنیم و درک کنیم که چرا دسترسی به index مورد نظر نادرست است. این پیام‌های خطا به وضوح مشکلات برنامه ما را نشان داده و اطلاعات ارزشمندی را برای رفع خطاها ارائه میدهد. در خروجی داریم:

Invalid write of size: نشان میدهد که در حال نوشتن در یک مکان حافظه نامعتبر هستیم.

bytes are definitely lost: این خطا نشان دهنده نشت حافظه است و بیان میکند که 2 بایت را در خط 6 index.c اختصاص داده ایم، اما قبل از پایان برنامه، حافظه را آزاد نکرده ای.

==12228== Memcheck, a memory error detector

==12228== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.

==12228== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info

==12228== Command: ./ index

==12228==

==12228== Invalid write of size 4

==12228== at 0x10919C: main (index.c:6)

==12228== Address 0x4b2d07c is 0 bytes after a block of size 2 alloc'd

==12228== at 0x4839C67: malloc (in /usr/lib/x86\_64-linux gnu/valgrind/vgpreload\_memcheck-amd64-linux.so)

==12228== by 0x109171: main (index.c:6)

==12228==

==12228==

==12228== HEAP SUMMARY:

==12228==     in use at exit: 2 bytes in 1 blocks

==12228==   total heap usage: 1 allocs, 0 frees, 2 bytes allocated

==12228==

==12228== 2 bytes in 1 blocks are definitely lost in loss record 1 of 1

==12228==    at 0x4839C67: malloc (in /usr/lib/x86\_64-linux-gnu/valgrind/vgpreload\_memcheck-amd64-linux.so)

==12228==    by 0x109171: main (index.c:6)

==12228==

==12228== LEAK SUMMARY:

==12228==    definitely lost: 2 bytes in 1 blocks

==12228==    indirectly lost: 0 bytes in 0 blocks

==12228==      possibly lost: 0 bytes in 0 blocks

==12228==    still reachable: 0 bytes in 0 blocks

==12228==         suppressed: 0 bytes in 0 blocks

==12228==

==12228== For counts of detected and suppressed errors, rerun with: -s

==12228== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)

اگر داده را 0 = [1000000000] تنظیم کنیم، رفتار undefined ایجاد میکند. این بار، به index دسترسی داده‌ایم که بسیار فراتر از حافظه اختصاص داده شده برای آرایه data (0تا 99) است. برنامه هنگام تلاش برای دسترسی خارج از محدوده چنین بزرگی crash شده و خطای تقسیم بندی میدهد. در C، باید اطمینان حاصل کرد که به عناصر آرایه، در محدوده اختصاص داده شده آنها دسترسی بدهیم.

Question 4: Access

پیغام خطایی که با آن مواجه میشویم نشان میدهد که برنامه با مشکل آزادسازی حافظه مواجه شده است. خطای "free invalid pointer" نشان میدهد که برنامه سعی در آزاد کردن حافظه با استفاده از یک اشاره‌گر نامعتبر داشته. این مشکل به دلایل زیر رخ میدهد:

1. Double Free: برنامه سعی دارد یک بلوک حافظه مشابه را بیش از یک بار آزاد کند. این وضعیت می تواند باعث رفتار undefined شود.
2. Invalid Pointer: برنامه از یک اشاره گر نامعتبر به عنوان آرگومان برای تابع استفاده میکند. اگر به pointer آدرس معتبری اختصاص داده نشده باشد یا مقداری متفاوت از آنچه توسط malloc برگردانده شده باشد این اتفاق می‌افتد.

GNU gdb (GNU/Linux 5.15.90.1-microsoft-standard-WSL2 x86\_64)

Copyright (C) 2020 Free Software Foundation, Inc.

License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software: you are free to change and redistribute it.

There is NO WARRANTY, to the extent permitted by law.

Type "show copying" and "show warranty" for details.

This GDB was configured as "x86\_64-gnu-linux".

Type "show configuration" for configuration details.

For help, type "help".

For bug reporting instructions, please see:

Type "apropos word" to search for commands related to "word"...

Reading symbols from access...

(gdb) run

Starting program: /home/parniansh/ access

free(): invalid pointer

هنگامی که کد را با Valgrind اجرا میکنیم، در شناسایی خطاهایی مانند leak memory و دسترسی‌های نامعتبر به حافظه کمک میکند. خروجی یک عملیات free نامعتبر را نشان میدهد، که با سناریویی که در آن یک اشاره‌گر نامعتبر آزاد میکنیم (invalidPtr) مطابقت دارد. Valgrind این خطا را تشخیص داده و اطلاعاتی در مورد خط مشکل دار در کد ارائه میدهد.

1. Invalid free: اشاره میکند که تابع در یک آدرس حافظه نامعتبر فراخوانی میشود. زیرا به invalidPtr مقدار data اختصاص داده میشود که یک اشاره گر معتبر است، اما داده ها آزاد میشوند و invalidPtr نیز نامعتبر میشود.
2. Memory leak: می‌گوید که قطعاً 4 بایت در 1 بلوک از دست رفته است. زیرا بلوک حافظه اختصاص داده شده توسط data ها، قبل از پایان برنامه آزاد نمیشود و منجر به leak میشود.

==11905== Memcheck, a memory error detector

==11905== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.

==11905== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info

==11905== Command: ./access

==11905==

==11905== Invalid free() / delete / delete[] / realloc()

==11905== at 0x462AE80: free (in /usr/lib/x86\_64-linux gnu/valgrind/vgpreload\_memcheck-amd64-linux.so)

==11905== by 0x1088C3: main (access.c:7)

==11905== Address 0x4a56108 is 2 bytes inside a block of size 4 alloc'd

==11905== at 0x475B7E2: malloc (in /usr/lib/x86\_64-linux-gnu/valgrind/vgpreload\_memcheck-amd64-linux.so)

==11905== by 0x10794E: main (access.c:6)

==11905==

==11905==

==11905== HEAP SUMMARY:

==11905==     in use at exit: 4 bytes in 1 blocks

==11905==   total heap usage: 1 allocs, 1 frees, 4 bytes allocated

==11905==

==11905== 4 bytes in 1 blocks are definitely lost in loss record 1 of 1

==11905== at 0x475B7E2: malloc (in /usr/lib/x86\_64-linux gnu/valgrind/vgpreload\_memcheck-amd64-linux.so)

==11905== by 0x10794E: main (access.c:6)

==11905==

==11905== LEAK SUMMARY:

==11905== definitely lost: 4 bytes in 1 blocks

==11905== indirectly lost: 0 bytes in 0 blocks

==11905== possibly lost: 0 bytes in 0 blocks

==11905== still reachable: 0 bytes in 0 blocks

==11905== suppressed: 0 bytes in 0 blocks

==11905==

==11905== For counts of detected and suppressed errors, rerun with: -s

==11905== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)

در ادامه داک میخواهیم نشانگر funnyPtr را که به وسط آرایه data اشاره میکند را آزاد کنیم. هنگام اجرا کد رفتار undefined نشان میدهد. تابع انتظار یک اشاره‌گر معتبر را دارد که قبلاً با استفاده از malloc، تخصیص داده شده. ارائه یک اشاره‌گر مانند funnyPtr که با این معیارها مطابقت ندارد، منجر به نتایج غیرقابل پیش بینی میشود. برای شناسایی و حل این مشکل، می‌توانیم از ابزارهایی مانند memory debugging (مثال Valgrind) یا ابزارهای static code analysis استفاده کنیم که میتوانند دسترسی‌ها یا تخصیص نامعتبر حافظه را شناسایی کنند.

(خروجی‌ها را برای داشتن شکل منظم یک دور در vscode کپی و از آنجا در داک ارسالی پیست کردم)