



# Using valgrind with gdb

Thomas Oulevey

ITLT-8



# Tools

**Valgrind** is an instrumentation framework for building dynamic analysis tools. There are Valgrind tools that can automatically detect many memory management and threading bugs, and profile your programs in detail.

It runs on **many** platforms:

X86/Linux, AMD64/Linux, ARM/Linux, ARM64/Linux, PPC32/Linux, PPC64/Linux, PPC64LE/Linux, S390X/Linux, MIPS32/Linux, MIPS64/Linux, TILEGX/Linux, X86/Solaris, AMD64/Solaris, ARM/Android (2.3.x and later), ARM64/Android, X86/Android (4.0 and later), MIPS32/Android, X86/Darwin and AMD64/Darwin (Mac OS X 10.10, with initial support for 10.11)

**GDB**, the GNU Project debugger, allows you to see what is going on `inside' another program while it executes -- or what another program was doing at the moment it crashed.

It supports **C**, **C++**, **D**, **Go**, Objective-C, Fortran, **Java**, OpenCL C, Pascal, assembly, Modula-2, and Ada.

# Valgrind Output

```
$ valgrind --leak-check=full ./t
==9612== HEAP SUMMARY:
==9612==   in use at exit: 28 bytes in 2 blocks
==9612== total heap usage: 2 allocs, 0 frees, 28 bytes allocated
==9612==
==9612== 7 bytes in 1 blocks are definitely lost in loss record 1 of 2
==9612==   at 0x4C29BFD: malloc (in /usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)
==9612==   by 0x4EBB529: strdup (in /usr/lib64/libc-2.17.so)
==9612==   by 0x40055E: main (toto.c:7)
==9612==
==9612== 21 bytes in 1 blocks are definitely lost in loss record 2 of 2
==9612==   at 0x4C29BFD: malloc (in /usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)
==9612==   by 0x4EBB529: strdup (in /usr/lib64/libc-2.17.so)
==9612==   by 0x400548: main (toto.c:5)
==9612==
==9612== LEAK SUMMARY:
==9612==   definitely lost: 28 bytes in 2 blocks
==9612==   indirectly lost: 0 bytes in 0 blocks
==9612==   possibly lost: 0 bytes in 0 blocks
==9612==   still reachable: 0 bytes in 0 blocks
==9612==   suppressed: 0 bytes in 0 blocks
==9612==
```

# GDB

```
$ gdb ./araignee
```

```
Reading symbols from ./araignee...done.
```

```
>>> run -t
```

```
>>> break myfunc
```

```
>>> print myvar
```

Cheat sheet :

[https://www.sthu.org/code/codesnippets/files/gdb\\_cheatsheet.pdf](https://www.sthu.org/code/codesnippets/files/gdb_cheatsheet.pdf)

<http://users.ece.utexas.edu/~adnan/gdb-refcard.pdf>



# Mixing the tools together

**--vgdb**=<no | yes | full> [default: yes]

Valgrind will provide "gdbserver" functionality when **--vgdb=yes** or **--vgdb=full** is specified.

This allows an external GNU GDB debugger to control and debug your program when it runs on

**--vgdb-error**=<number> [default: 999999999]

Tools that report errors will wait for "number" errors to be reported before freezing the program and waiting for you to connect with GDB.

# buggy C code : araignee.c

Memory leak ;

Variable malloced but not freed (10 bytes)

```
$ gcc -g -O1 -o araignee araignee.c
```

```
$ valgrind --leak-check=full \  
--vgdb=yes --vgdb-error=0 ./araignee
```

```
==13245== TO DEBUG THIS PROCESS USING GDB: start GDB like this
```


```
==13245== /path/to/gdb ./araignee
```

```
==13245== and then give GDB the following command
```

```
==13245== target remote | /usr/lib64/valgrind/../../bin/vgdb --pid=13245
```

```
==13245== --pid is optional if only one valgrind process is running
```

```
1 #include <stdio.h>  
2 #include <stdlib.h>  
3  
4 int main(void)  
5 {  
6     char *p = malloc(10);  
7     *p = 'a';  
8     int i =42;  
9     char c = *p;  
10  
11     printf("\n [%i][%c]\n", i, c);  
12  
13     return 0;  
14 }
```

 \$ gcc -c -Q -O1 --help=optimizers

# Workflow

```
$ gdb ./araignee
Reading symbols from ./araignee...done.
>>> target remote | vgdb
Remote debugging using | vgdb
relaying data between gdb and process 13850
>>> break 7
>>> break 14
Breakpoint 1 at 0x400596: file araignee.c, line 7.
Undefined command: "". Try "help".
>>> continue
>>> monitor leak_check
==13850== All heap blocks were freed -- no leaks are possible
>>> continue
>>> monitor leak_check
==13850== LEAK SUMMARY:
==13850==   definitely lost: 10 bytes in 1 blocks
==13850==   indirectly lost: 0 bytes in 0 blocks
==13850==   possibly lost: 0 bytes in 0 blocks
==13850==   still reachable: 0 bytes in 0 blocks
==13850==   suppressed: 0 bytes in 0 blocks
```



```
>>> break araignee.c:8 if i == 42
```

1 . Run your program under GDB and Valgrind

2 . Put a break at where you think the memory is lost

```
break 7
break main
```

3. Continue there

```
continue
```

4. Check for memory leak

```
monitor leak_check
```

5. reiterate until you find the leak

```
next / step / continue / print
monitor leak_check
```

# Cool gdb tuning

## gdb-dashboard

```
Assembly
0x000000000400544 main+20 callq 0x400430 <strdup@plt>
0x000000000400549 main+25 mov %rax,%rax
0x00000000040054d main+29 movq $0x0,%rax
0x000000000400555 main+37 mov $0x400615,%edi
0x00000000040055a main+42 callq 0x400430 <strdup@plt>
0x00000000040055f main+47 mov $0x0,%eax
0x000000000400564 main+52 leaveq

Expressions
History
History
$$_0 = 0x602010 "catch me if you can!"
Memory
Registers
rax 0x0000000000002010 rbx 0x0000000000000000
rcx 0x7920666920656d20 rdx 0x0000000000000000
rsi 0x000000000400615 rdi 0x0000000000002025
rbp 0x00007ffffffd440 rsp 0x00007ffffffd420
r8 0x00215e616320756f r9 0x0000000000000002
r10 0x000000000000001 r11 0x0000000000000245
r12 0x000000000400440 r13 0x00007ffffffd520
r14 0x000000000000000 r15 0x0000000000000000
rip 0x000000000400555 eflags [ PF ZF IF ]
cs 0x00000033 ss 0x0000002b
ds 0x00000000 es 0x00000000
fs 0x00000000 gs 0x00000000

Source
2 #include <string.h>
3
4 int main(int argc, char *argv[]) {
5     char *lost = strdup("catch me if you can!");
6     lost = NULL;
7     strdup("foobar");
8     return 0;
9 }

Stack
[0] from 0x000000000400555 in main+37 at toto.c:7
arg argc = 1
arg argv = 0x7ffffffd528

Threads
[1] id 10635 name t from 0x000000000400555 in main+37 at toto.c:7

>>> dashboard -output /dev/pts/2
>>> b strdup
Note: breakpoint 1 also set at pc 0x7ffff7aa0510.
Breakpoint 2 at 0x7ffff7aa0510
>>> r
Starting program: /afs/cern.ch/user/a/alphacc/t

Breakpoint 1, 0x00007ffff7aa0510 in strdup () from /lib64/libc.so.6
>>> n
Single stepping until exit from function strdup,
which has no line number information.
0x00007ffff7aa0510 in __memcpy_sse2 () from /lib64/libc.so.6
>>> n
Single stepping until exit from function __memcpy_sse2,
which has no line number information.
main (argc=1, argv=0x7ffffffd528) at toto.c:6
6 lost = NULL;
>>> p lost
$1 = 0x602010 "catch me if you can!"
>>> n
7 strdup("foobar");
>>> □

[0] 1:[tmux]- 2:a1phacc@craba:~ "craba.cern.ch" 16:40 03-Dec-11
```

<https://github.com/cyrus-and/gdb-dashboard>



# Thank you !

SSSD bug : <https://fedorahosted.org/sssd/ticket/2803>

Thanks, Sebastien Ponce for the debug session