Google的[tcmalloc](http://gperftools.googlecode.com/files/gperftools-2.0.tar.gz)可以做内存越界检查，也就是查野指针。

野指针是应用程序最难查的崩溃的问题。google真的很强大，赞！

基本原理就是在分配时分配到页的底部，这样越界时就会报错了。也就是PAGE\_FENCE，这个选项是可以通过环境变量设置的，代码在：src/debugallocation.cc: 101

DEFINE\_bool(malloc\_page\_fence,

EnvToBool("TCMALLOC\_PAGE\_FENCE", false),

"Enables putting of memory allocations at page boundaries "

"with a guard page following the allocation (to catch buffer "

"overruns right when they happen).");

* 1
* 2
* 3
* 4
* 5

可以直接将代码改掉：

将

EnvToBool("TCMALLOC\_PAGE\_FENCE", false)

改成了

EnvToBool("TCMALLOC\_PAGE\_FENCE", true)

脚本：

sed -i "s/EnvToBool(\"TCMALLOC\_PAGE\_FENCE\", false)/EnvToBool(\"TCMALLOC\_PAGE\_FENCE\", true)/g" src/debugallocation.cc

* 1
* 2
* 3
* 4
* 5
* 6

或者设置环境变量：

env TCMALLOC\_PAGE\_FENCE=1 ./your\_application

* 1

编译出静态库(若需要使用so库需要安装)：

cd gperftools-2.0 && ./configure --enable-frame-pointers && make

* 1

编译选项加上：

LibTcMalloc="-fno-builtin-malloc -fno-builtin-calloc -fno-builtin-realloc -fno-builtin-free ${SMT\_OBJS}/gperftools-2.0/.libs/libtcmalloc\_debug.a"

* 1

使用gdb调试，在越界的地方就会停下来。

下面的代码有越界，但是执行是没有问题的：

/\*\*

g++ memory.error.notcmalloc.cpp -g -O0 -o memory.error.notcmalloc

\*/

#include <unistd.h>

#include <string.h>

#include <stdio.h>

void foo(char\* p){

memcpy(p, "01234567890abcdef", 16);

}

int main(int argc, char\*\* argv){

char\* p = new char[10];

foo(p);

printf("p=%s\n", p);

return 0;

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16

执行是没有问题，一般linux会多分配，而且越界的地方并非只读：

[winlin@dev6 code]$ ./memory.error.notcmalloc

p=01234567890

* 1
* 2

加上tcmalloc的debug库之后，就可以看到越界的地方了：

/\*\*

(unzip -q ../../3rdparty/gperftools-2.1.zip &&

cd gperftools-2.1 && ./configure --enable-frame-pointers && make)

g++ memory.error.tcmalloc.cpp -g -O0 \

-fno-builtin-malloc -fno-builtin-calloc -fno-builtin-realloc \

-fno-builtin-free ./gperftools-2.1/.libs/libtcmalloc\_debug.a \

-o memory.error.tcmalloc -lpthread

\*/

#include <unistd.h>

#include <string.h>

#include <stdio.h>

void foo(char\* p){

memcpy(p, "01234567890abcdef", 16);

}

int main(int argc, char\*\* argv){

char\* p = new char[10];

foo(p);

printf("p=%s\n", p);

return 0;

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21

[winlin@dev6 code]$ env TCMALLOC\_PAGE\_FENCE=1 gdb memory.error.tcmalloc

(gdb) r

Program received signal SIGSEGV, Segmentation fault.

(gdb) bt

#0 memcpy () at ../sysdeps/x86\_64/memcpy.S:120

#1 0x0000000000405436 in foo (p=0x7ffff7ff9ff6 "01234567\253\253") at memory.error.tcmalloc.cpp:14

#2 0x0000000000405461 in main (argc=1, argv=0x7fffffffe388) at memory.error.tcmalloc.cpp:18

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8

真的很牛逼：

(gdb) f 1

#1 0x0000000000405436 in foo (p=0x7ffff7ff9ff6 "01234567\253\253") at memory.error.tcmalloc.cpp:14

14 memcpy(p, "01234567890abcdef", 16);

(gdb) l

9 #include <unistd.h>

10 #include <string.h>

11 #include <stdio.h>

12

13 void foo(char\* p){

14 memcpy(p, "01234567890abcdef", 16);

15 }

16 int main(int argc, char\*\* argv){

17 char\* p = new char[10];

18 foo(p);

(gdb)

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15

靠人来找这种问题，找死了都找不到。