Android memory corruption debugger

原创 2017年11月29日 14:10:59 标签：android / memory 75

kongxinsun
+ 关注

原创	粉丝	喜欢	码云
7	0	0	未开通

他的最新文章 更多文章

binder call fail caused by no address spa ce

Decay timer of Jemalloc on android

Android show memory info cmd

android java process stack OOM

ion orphaned memory

roduction

memory corruption是最难搞的问题之一，这是因为：
坏内存的地方和内存破坏的结果常常是分离的，所以很难定位。
2. 有些破坏在特定case下才会出现，不好复现。

ory corruption errors大概可以分成以下几类：
用未初始化的内存
用非法内存(不是自己申请的内存，使用空指针，野指针)
3. buffer overflow，内存越界读写
4. 错误的heap管理问题：申请的内存没有free，free后又使用，两次free
5. Kernel driver 冲应用内存，特别是cma这种方式

fatal signal

memory corruption 会引起进程crash，触发的signal有以下几种：

1. SIGSEGV（segment fault）
   程序读写它没有分配的内存，或者写只读内存，会抛出 SIGSEGV
   非法内存访问（段错误），常见的case:
   1. 空指针解引用
   2. 引用非法地址
   3. 防问无权限的地址
   4. 写只读的内存（比如code段）

2. SIGBUS (bus error)

SIGSEGV indicates an invalid access to valid memory,
SIGBUS indicates an access to an invalid address.
SIGBUS通常是解引用未对齐的pointer，比如解引用一个four-word integer但是地址不能被4整除。也就是未对齐的数据访问导致

3. SIGILL（illegal instruction）
非法程序映像，例如非法指令。SIGILL通常表明，可执行文件被破坏了，或者程序在执行data。后一种情况，比较常见的是对函数指针非法赋值；或者是stack被破坏；或者是stack overflow。

0

bugger

://en.wikipedia.org/wiki/Memory_debugger

的wiki里，列举了各种debug方法。一种是在编译时检查，另一种是运行时检测。

对于android memory corruption的debugger方法大概有：

- Asan: address sanitizer
- UbSan: undefined Behavior Sanitizer
- Valgrind
- Libc malloc debug (setprop libc.debug.malloc 10)

这些方法只对native 层有用，可以检测null pointer, buffer overflow, free后又使用等。但是对于java heap 则无能为力。java heap出问题的概率非常小，android 官网上推荐可以gc verify 和 jni check。

https://source.android.com/devices/tech/dalvik/gc-debug

对于比较复杂的case，可以打开coredump分析。

asan

http://clang.llvm.org/docs/AddressSanitizer.html
https://source.android.com/devices/tech/debug/asan.html

LLVM是用于构建Android的编译器，其中包含多个组件可以进行静态和动态分析。Sanitizers就是其中之一。sanitizers通常指addressSanitizer和UndefinedBehaviorSanitizer。

Ascan是用来检测C/C++程序memory问题的编译工具，是在运行时检测的。由一个编译器 (external/clang) 和一个运行时 library (external/compile-rt/lib/asan)组成

可检测的memory问题包括：

- Out-of-bounds memory access
- Double free
- use after free

于valgrind，它具有以下特点：

检测stack和global objects的overflow问题

2. 内存泄漏无能为力
3. 比valgrind快（慢两三倍，valgrind要慢20~100倍）
4. 内存消耗更少

usage

1 native process build with Ascan
将下面这两行加到Android.mk文件里

```
1 LOCAL_CLANG:=true
2 LOCAL_SANITIZE:=address
```

2 Share library build with Ascan
将下面这三行加到Android.mk文件里

```
1 LOCAL_CLANG:=true
2 LOCAL_SANITIZE:=address
3 LOCAL_MODULE_RELATIVE_PATH := asan
```

3 Native process used asan so lib
可执行程序要使用asan库，指定下面这个环境变量：

```
1  LD_LIBRARY_PATH=/system/lib/asan
```

例如，对于rc文件定义启动的native程序，可以在rc文件里加上。
确认进程使用ascan的lib呢？可以查看/proc/$PID/maps里面，是否使用的是/system/lib/asan/**的库。
另外，需要关闭selinux

an with app
address Sanitizer java 代码并不能使用，但是可以用来检测JNI 代码。需要编译的是
em/bin/app_process(32|64)
1) app_process mk里打开asan
eworks/base/cmds/app_process/Android.mk

```
1  LOCAL_CLANG:=true
2  LOCAL_SANITIZE:=address
```

2) 设置环境变量
init.zygote(32|64).rc

```
1  setenv LD_LIBRARY_PATH /system/lib/asan:/system/lib
2  setenv ASAN_OPTIONS
3  allow_user_segv_handler=true
```

3) Using the wrap property
上面的方法，所有的java程序都会使用Asan. 这样的话，比较费内存。
可以指定一个或几个apk使用Asan，下面的例子，gmail under asan

```
1  $ adb root
2  $ adb shell setenforce 0  # disable SELinux
3  $ adb shell setprop wrap.com.google.android.gm "asanwrapper"
```

Symbolization

有两种方法，可以将asan的输出定位到源文件和代码行：

- 将llvm-symbolizer 编译到平台里/system/bin/Llvm-symbolizer，这个bin文件源码在:third_party/llvm/tools/llvm-symbolizer

用这脚本分析 external/compiler-rt/lib/asan/scripts/symbolize.py

第二种方法能产生更多的信息，因为可以访问主机上的 symbolized libraries.

```
external/compiler-rt/lib/asan/scripts/symbolize.py  < log | c++filt
```

efile里加上这两行，可以得到更好的backtrace

```
LOCAL_CFLAGS:=-fno-omit-frame-pointer
LOCAL_ARM_MODE:=arm
```

另外，kernel也支持asan，但是kernel要是4.9以后的。

http://stackoverflow.com/questions/24566416/how-do-i-get-line-numbers-in-the-debug-output-with-clangs-fsanitize-address
http://clang.llvm.org/docs/AddressSanitizer.html#symbolizing-the-reports

UBSan

introduction

UBSan在编译时，检测变量的未定义操作。在android上，支持的检测有以下几种：

alignment, bool, bounds, enum, float-cast-overflow, float-divide-by-zero, integer-divide-by-zero, nonnull-attribute, null, return, returns-nonnull-attribute, shift-base, shift-exponent, signed-integer-overflow, unreachable, unsigned-integer-overflow, and vla-bound.

Usage

```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)

LOCAL_CFLAGS := -std=c11 -Wall -Werror -O0
LOCAL_SRC_FILES:= sanitizer-status.c

LOCAL_MODULE:= sanitizer-status
LOCAL_MODULE_TAGS := debug

LOCAL_SANITIZE := alignment bounds null unreachable integer LOCAL_SANITIZE_DIAG :

include $(BUILD_EXECUTABLE)
```

San shortcuts

有两个shortcuts： integer 和 default-ub,可以同时enable多个sanitizers。
integer enables integer-divide-by-zero, signed-integer-overflow and unsigned-integer-overflow.

default-ub enables the checks that have minimal compiler performance issues: bool, integer-divide-by-zero, return, returns-nonnull-attribute, shift-exponent, unreachable and vla-bound.

integer sanitizer 可以用在 SANITIZE_TARGET 和 LOCAL_SANITIZE 中；而 default-ub 只能用在 SANITIZE_TARGET中。

Better error reporting

Android.mk文件里：

```
LOCAL_SANITIZE:=integer
LOCAL_SANITIZE_DIAG:=integer
```

其中LOCAL_SANITIZE 在build时，enable UBSan。LOCAL_SANITIZE_DIAG 打开特定sanitizer的diagnostic mode。这两个可以设成不同的值，但是此时只有 LOCAL_SANITIZE 是 enabled。如果某种sanitizer没有在LOCAL_SANITIZE中指定，但是在 LOCAL_SANITIZE_DIAG中指定，那么检查不会

Valgrind

introduction

http://valgrind.org/docs/manual/mc-manual.html

valgrind是用来检查内存问题的工具，memcheck是默认打开的，可以检测 C/C++程序以下几种内存问题：

- Accessing memory you shouldn't, e.g. overrunning and underrunning heap blocks, overrunning the top of the stack, and accessing memory after it has been freed.
- Using undefined values, i.e. values that have not been initialised, or that have been derived from other undefined values.
- Incorrect freeing of heap memory, such as double-freeing heap blocks, or mismatched use of malloc/new/new[] versus free/delete/delete[]
- Overlapping src and dst pointers in memcpy and related functions.
- Passing a fishy (presumably negative) value to the size parameter of a memory allocation function.
- Memory leaks.

当然使用valgrind会带来以下问题：

- 程序的内存使用量大大增加
- 程序的运行速度大大减慢（通常会慢20~100倍）

use valgrind in android

https://source.android.com/devices/tech/debug/valgrind

有两点需要注意，一个建议使用eng版本，另一个是valgrind下运程程序会超级慢。这实际上也就限制了它的使用场景，只有比较单纯的环境下才能用。

1 Build valgrind

```
1  $ mmm -j6 external/valgrind
```

```
$ adb root
$ adb remount
$ export ANDROID_PRODUCT_OUT=$ANDROID_PRODUCT_OUT
$ adb sync
```

3 Set up the temporary directory:

```
$ adb shell mkdir /data/local/tmp
$ adb shell chmod 777 /data/local/tmp
```

4 disable selinux

```
$ adb shell setenforce 0
```

5 push symbols

```
$ adb shell mkdir /data/local/symbols
$ adb push $OUT/symbols /data/local/symbols
$ adb push $ANDROID_PRODUCT_OUT/symbols /data/local/symbols
```

6 使用valgrind 运行需要debug的程序

对于native 程序，开机启动的services，需要修改init.*.rc文件。例如下面的

```
service example /system/bin/foo --arg1 --arg2
change to:
service example /system/bin/logwrapper /system/bin/valgrind  /system/bin/foo  --a
```

对与java程序，使用wrap property，例如calculator：

```
$ adb shell setprop wrap.com.android.calculator2 "TMPDIR=/data/data/com.android.c

kill calculator and restart it.
```

下面的例子是system server使用valgrind:

```
adb root
adb shell setenforce 0
adb shell chmod 777 /data/local/tmp
adb shell setprop wrap.system_server "logwrapper valgrind"
adb shell stop && adb shell start
```

valgrind other option

ory check选项，valgrind是默认打开。valgrind还有其它的一些选项，可以在选择打开，方法是在
ind命令后面，增加这些选项和参数。

```
valgrind --error-limit=no --trace-children=yes --sigill-diagnostics=no --extra-de
```

https://source.android.com/devices/tech/dalvik/gc-debug.html#valgrind

Problems

在有些android版本上， system_server在valgrind下跑不起来，有两个问题。

- 有些指令识别不了
- 找不到system server这个class， path不对

其它apk在valgrind下跑，会出现anr，watchdog kill system server等问题。总之valgrind并不太好用。

Libc malloc debugger

Libc malloc debugger是android libc里自带的，也是最简单易用的corruption debugger了。使用方法，编个userdebug或eng版本，同时设上这两上property:

```
setprop libc.debug.malloc 10
setprop libc.debug.malloc.program nativeprocess
```

虽然简单易用，但是这种方法也有自身的限制。检测方法是在申请buffer的头部和尾部都写入了特殊字符，但是只有当buffer被free，或者是realloc时，才会检查是否发生了变化。很多情况下，当某块buffer被破坏时，但是crash发生是检查前，就无能为力了。

Coredump

Introduction

Coredump 是操作系统在进程收到某些信号而终止运行时，将此时进程地址空间的内容以及有关进程状态的其他信息写出的一个磁盘文件。这种信息往往用于调试。

1 查看系统是否打开了coredump

```
1  ulimit -c
```

如果返回0，则表明功能关闭，不会生成core文件

2 打开coredump的方法\

- 命令行方式 ulimit -c 1024，限制core文件大小1024K。ulimit -c unlimited 不加限制
- linux下，修改配置文件， /etc/profile 加上 ulimit -S -c unlimited > /dev/null 2>&1
- 修改init.rc文件

3 core 文件的保存路径和文件名格式
core文件路径和格式在：/proc/sys/kernel/core_pattern
core 文件格式如下：

```
1  %p – insert pid into filename
2  %u – insert current uid into filename
3  %g – insert current gid into filename
4  %s – insert signal that caused the coredump into the filename
5  %t – insert UNIX time that the coredump occurred into filename
6  %h – insert hostname where the coredump happened into filename
7  %e – insert coredumping executable name into filename
```

Open coredump in android

Coredump文件都比较大，android系统上默认是关掉的。

1 native 程序 init.rc 修改如下

```
diff --git a/rootdir/init.rc b/rootdir/init.rc
index 179316c..d32cb85 100755
--- a/rootdir/init.rc
+++ b/rootdir/init.rc
@@ -267,6 +267,9 @@ on boot

 # set RLIMIT_NICE to allow priorities from 19 to -20
     setrlimit 13 40 40
+    setrlimit 4 -1 -1
+    write /proc/sys/kernel/core_pattern "/data/core-%e.%p.%u"
+    write /proc/sys/fs/suid_dumpable 1
```

2 java程序
Android zygote和system server里关掉了core dump。
TODO find the open way.

coredump debug

得到coredump文件后，可以使用gdb 进行debug，查看当时的内存信息。

```
arm-eabi-gdb --core=corefile  （arm-eabi-gdb 在ndk里有）
```

方法如下：

1 将core文件和symbols放到同一目录下，symbols包括可执行程序和so库文件。在out/target/product/*/symbols/目录下
2 运行gdb，要用arm-eabi-gdb

```
arm-eabi-gdb
```

```
set solib-search-path system/lib
```

4 file 命令载入主执行文件

```
file system/bin/execfile
```

e-file 载入Coredump文件

```
core-file  coredumpfile
```

查看so库的加载路径是否正确可使用

```
info sharedlibrary
```

7 使用bt 命令，查看crash时的backtrace. backtrace中，每个函数都有一个frame number。可以使用frame [num] 查看特定的stack frame. 使用list查看源码，info locals 查看局部变量。

在设置了搜索路路径后，最好先用file命令载入主执行文件，再用core命令载入Coredump文件，这样才能保证正确载入库的符号表。

problem

Coredump不是特别有用，由于编译优化，局部变量都看不到了。只能看一些全局变量。而且java层，coredump默认是关闭的。对于，特别复杂的case，可以使用coredump。

如何关掉android 编译优化呢?
可以将下面这一行，加到make file里，可以关闭掉编译优化。

```
LOCAL_CFLAGS += -O0
```

Summary

另外，例版本是终极大招

但是如果是driver冲了上层内存，就比较麻烦。如果被踩踏的位置比较固定，位置固定是指物理地址固定或者在特定范围；可以将分配这块物理地址的callstack dump出来，找到怀疑点。也可以将这个固定的物理地址，设成只读的，如果有人写它的话，就可以抓到。

但是如果破坏位置是随机的，这种情况是最难定位。一种可行的，比较暴力的debug方法是随机地申请内存，将这申请到的内存设成只读的。那么如果踩踏到这些只读内存时，就可以抓到破坏者。当申请的只读比较大的时候，抓到的概率也是比较高的。

如何是硬件DMA冲掉应用内存造成crash的问题，没有好的手段可以debug，很多时候问题的解决得靠猜测运气，并且需要反反复复各种测试。

malloc的实现原理

qq_27114221 2016年12月02日 23:53 74

malloc（ ）是C语言中动态存储管理 的一组标准库函数之一。其作用是在内存的动态存储区中分配一个长度为size的连续空间。其参数是一个无符号整形数，返回值 是一个指向所分配的连续存储域的起始地址...

libc --有关 How main() is executed on Linux

ivan240 2010年05月03日 17:53 659

http://hi.baidu.com/heidycat/blog/item/f8fe3ba1c1026b8e46106472.html在linux下，程序的运行过程比我想象中要复杂得多便一个程...

android native 内存泄露检查（libc.debug.malloc ）

http://www.csdn123.com/html/blogs/20131011/81402.htm haima1998 2016年05月26日 17:11 4213

1. Introduction Android对内存的使用包括内存泄漏和...

Android memory leak detect

kongxinsun 2017年11月29日 15:10 91

What is memory leak Native process memory leak detect Java process memory leak detect Kmemleak usage...

Malloc Debug & Native Memory Tracking using libc Callbacks

https://android.googlesource.com/platform/bionic/+/master/libc/malloc_debug/README_api.md ...

agwtpcbox 2016年11月30日 18:02 608

Redis安装与调试

21aspnet 2014年10月31日 17:38 9701

最新版Redis安装与调试Redis安装与调试linux版本：64位CentOS 6.5Redis版本：2.8.17（更新到2014年10月31日) Redis官网：http://redis.io...

Android中native进程内存泄露的调试技巧（一）

l_nan 2015年02月04日 18:03 8078

基于Android5.0版本 Android为Java程序提供了方便的内存泄露信息和工具（如MAT），便于查找。但是，对于纯粹C/C++ 编 tvie进程，却不那么容易查找内存泄露。传统的C/C...

0

nux上使用AFL对Stagefright进行模糊测试

omnispace 2017年06月29日 15:03 677

模糊测试是一种自动向程序传递输入数据并监控其输出的自动化测试技术。通过这种技术，安全人员可以测试程序的可 及识别潜在的安全漏洞。 我们（360成都安全响应中心）将对Stagefr...

linux调试工具glibc的演示分析

macky0668 2011年10月01日 19:54 5975

一)MALLOC_CHECK_ GNU的标准库(glibc)可以通过内置的调试特性对动态内存进行调试,它就是MALLOC_CHECK_环境变量, 它在默认情况下是不设定的,在老的版本默认这...

AddressSanitizer

sgzy001 2017年08月22日 23:39 257

Version:0.9StartHTML:0000000167EndHTML:0000040239StartFragment:0000000203EndFragment:0000040203Sourc...

malloc(): memory corruption (fast): 0x09a5e3e8错误的解决

1.现象描述： 程序在启动时，解析xml文件时出现malloc(): memory corr apn172 2012年06月09日 17:47 13298

uption (fast): 0x09a5e3e8错误。相同的代码在windows下运行时不会出现错误。 ...

*** glibc detected*** malloc(): memory corruption: 0x09eab988 ***发现是由于memset越界写引起的。在Linux Server上不好...

K346K346 2015年05月14日 09:22 2481

27.windbg-内存破坏实例分析

hgy413 2012年05月25日 15:00 3069

以下实例来自AWD 代码： /*++ Copyright (c) Advanced Windows Debugging (ISBN 0321374460) from Addison-Wesley ...

bg memory corruption

realduke2000 2011年02月22日 00:19 2788

http://www.informit.com/articles/article.aspx?p=1081496 While heap-based security attacks are much h...

内存写越界导致破环堆结构引起的崩溃问题定位经验[如报错malloc(): memory corr…

转载

zhang911025 2014年06月09日 18:46 2217

Yii2杂记

flyforfreedom2008 2016年06月02日 09:28 1768

最近组内Web开发用到了Yii2，我也花了一个多星期的时间来稍微研究一下这个框架。本来想把一些心得写成文章，可是时间过去了 很久，有些调用关系我自己现在已经模糊不清，加上长时间没有码字，要成文还需要考...

yii2 strace 追踪，本地文件

terry_water 2016年02月27日 16:23 762

13631 16:16:13 execve("/usr/local/php/bin/php", ["/usr/local/php/bin/php", "/www/web/develop/erp2/ba...

Unable to open debugger port (127.0.0.1:63777): java.net.BindException "Addr…