|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| D:\02-资料分类\04-文档模板\扉页图1-new.jpg | | | | |
|  | AndroidR\_Asan使用说明 | | |  |
| 文档版本 V2.0 |  | |
| 发布日期 2020-06-27 |  | |
|  | | | | |
|  | 紫光展锐科技有限公司 | |  |  |

|  |
| --- |
| 版权所有 © 紫光展锐科技有限公司。保留一切权利。  本文件所含数据和信息都属于紫光展锐所有的机密信息，紫光展锐保留所有相关权利。本文件仅为信息参考之目的提供，不包含任何明示或默示的知识产权许可，也不表示有任何明示或默示的保证，包括但不限于满足任何特殊目的、不侵权或性能。当您接受这份文件时，即表示您同意本文件中内容和信息属于紫光展锐机密信息，且同意在未获得紫光展锐书面同意前，不使用或复制本文件的整体或部分，也不向任何其他方披露本文件内容。紫光展锐有权在未经事先通知的情况下，在任何时候对本文件做任何修改。紫光展锐对本文件所含数据和信息不做任何保证，在任何情况下，紫光展锐均不负责任何与本文件相关的直接或间接的、任何伤害或损失。  请参照交付物中说明文档对紫光展锐交付物进行使用，任何人对紫光展锐交付物的修改、定制化或违反说明文档的指引对紫光展锐交付物进行使用造成的任何损失由其自行承担。紫光展锐交付物中的性能指标、测试结果和参数等，均为在紫光展锐内部研发和测试系统中获得的，仅供参考，若任何人需要对交付物进行商用或量产，需要结合自身的软硬件测试环境进行全面的测试和调试。非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。 |



|  |
| --- |
| 紫光展锐科技有限公司 |

前 言

概述

AddressSanitizer用于检测android native层的内存覆盖，本文档讲述其使用方法。

读者对象

本文档适用于需要在android native层调试的内存覆盖的同事，及其他对此感兴趣的同事。

缩略语

|  |  |  |
| --- | --- | --- |
| 缩略语 | 英文全名 | 中文解释 |
| ASan | AddressSanitizer | 地址清理程序 |
|  |  |  |

变更信息

| 文档版本 | 发布日期 | 作者 | 修改说明 |
| --- | --- | --- | --- |
| V1.0 | 2019-08-25 | Jasson.zhang | 初稿 |
| V1.1 | 2019-09-27 | Jasson.zhang | Review修改 |
| V2.0 | 2020-08-27 | Jasson.zhang | 针对ASan在AndroidR上使用修订 |
|  |  |  |  |

关键字

Asan、AndroidQ、内存、AddressSanitizer。

目 录

[1 AddressSanitizer简介 6](#_Toc49427794)

[1.1 简介 6](#_Toc49427795)

[1.1.1 AddressSanitizer 6](#_Toc49427796)

[1.1.2 实际功能验证 7](#_Toc49427797)

[1.2 编译 7](#_Toc49427798)

[1.2.1 使用 Clang 编译 7](#_Toc49427799)

[1.2.2 限制 8](#_Toc49427800)

[1.3 其他 8](#_Toc49427801)

[1.3.1 使用分支 8](#_Toc49427802)

[1.3.2 更多信息 8](#_Toc49427803)

[2 在可执行文件中使用AddressSanitizer 9](#_Toc49427804)

[2.1 使用方法 9](#_Toc49427805)

[2.2 以surfaceflinger为例 9](#_Toc49427806)

[2.2.1 打开开关 9](#_Toc49427807)

[2.2.2 模拟一个问题 10](#_Toc49427808)

[2.2.3 发现问题 11](#_Toc49427809)

[2.2.4 解析后的log 13](#_Toc49427810)

[2.2.5 对应代码 15](#_Toc49427811)

[2.3 以ylog为例演示 16](#_Toc49427812)

[2.3.1 打开开关 16](#_Toc49427813)

[2.3.2 模拟一个问题 17](#_Toc49427814)

[2.3.3 发现问题 17](#_Toc49427815)

[2.3.4 解析后的log 18](#_Toc49427816)

[2.3.5 对应代码 19](#_Toc49427817)

[3 在动态库中使用AddressSanitizer 21](#_Toc49427818)

[3.1 使用方法 21](#_Toc49427819)

[3.1.1 ASan自身的限制 21](#_Toc49427820)

[3.1.2 在编译文件中打开ASan 22](#_Toc49427821)

[3.1.3 文件执行 23](#_Toc49427822)

[3.1.4 检查 23](#_Toc49427823)

[3.1.5 注意 23](#_Toc49427824)

[3.1.6 简单的办法 24](#_Toc49427825)

[3.2 简单调用调试举例-libsurfaceflinger.so 25](#_Toc49427826)

[3.2.1 配置文件的修改 25](#_Toc49427827)

[3.2.2 模拟一个问题 26](#_Toc49427828)

[3.2.3 编译替换生成库 27](#_Toc49427829)

[3.2.4 检查到问题Log如下 27](#_Toc49427830)

[3.2.5 栈解析如下 28](#_Toc49427831)

[3.2.6 对应代码 29](#_Toc49427832)

[3.3 一般动态库调试举例 30](#_Toc49427833)

[3.3.1 配置文件的修改 31](#_Toc49427834)

[3.3.2 模拟一个问题 31](#_Toc49427835)

[3.3.3 编译替换生成库 32](#_Toc49427836)

[3.3.4 修改服务启动文件 33](#_Toc49427837)

[3.3.5 修改selinux权限 33](#_Toc49427838)

[3.3 检测出问题Log如下 34](#_Toc49427839)

[3.3.6 栈解析如下 36](#_Toc49427840)

[3.3.7 对应代码 37](#_Toc49427841)

[4 在应用程序中使用ASan 38](#_Toc49427842)

[4.1 打开Asan开关 38](#_Toc49427843)

[4.2 修改启动配置 39](#_Toc49427844)

[4.3 关于wrap.属性 40](#_Toc49427845)

[4.4 其他 40](#_Toc49427846)

[5 调用栈解析 41](#_Toc49427847)

[5.1 解析脚本 41](#_Toc49427848)

[5.2 脚本使用 41](#_Toc49427849)

[5.3 脚本使用举例 41](#_Toc49427850)

[6 常见内存错误ASan提示 44](#_Toc49427851)

[6.1 double-free 44](#_Toc49427852)

[6.1.1 问题代码 44](#_Toc49427853)

[6.1.2 AddressSanitizer检测到问题 45](#_Toc49427854)

[6.1.3 调用栈解析 45](#_Toc49427855)

[6.2 wild pointer free 46](#_Toc49427856)

[6.2.1 问题代码 46](#_Toc49427857)

[6.2.2 AddressSanitizer检测到问题 47](#_Toc49427858)

[6.2.3 解析后的调用栈 47](#_Toc49427859)

[6.3 stack-buffer-overflow 47](#_Toc49427860)

[6.3.1 stack overflow代码 48](#_Toc49427861)

[6.3.2 stack overflow AddressSanitizer检测到问题 49](#_Toc49427862)

[6.3.3 stack overflow栈解析 50](#_Toc49427863)

[6.4 heap-buffer-overflow 52](#_Toc49427864)

[6.4.1 AddressSanitizer检测到问题 52](#_Toc49427865)

[6.4.2 解析后的log 54](#_Toc49427866)

[6.4.3 问题代码分析 54](#_Toc49427867)

[6.4.4 检查验证 55](#_Toc49427868)

[6.5 heap-use-after-free 56](#_Toc49427869)

[6.5.1 问题代码 56](#_Toc49427870)

[6.5.2 AddressSanitizer检测到问题 57](#_Toc49427871)

[6.5.3 栈解析如下 58](#_Toc49427872)

[6.6 stack-use-after-scope 61](#_Toc49427873)

[6.6.1 问题代码 61](#_Toc49427874)

[6.6.2 AddressSanitizer检测到问题 61](#_Toc49427875)

[6.6.3 栈解析 62](#_Toc49427876)

[6.7 unknown address 63](#_Toc49427877)

[6.7.1 问题代码 63](#_Toc49427878)

[6.7.2 AddressSanitizer检测到问题 64](#_Toc49427879)

[6.7.3 栈解析 65](#_Toc49427880)

[7 ASan误报 66](#_Toc49427881)

[7.1 调试libandroid\_runtime.so 遇到container-overflow 66](#_Toc49427882)

[7.1.1 栈信息 66](#_Toc49427883)

[7.1.2 分析 69](#_Toc49427884)

[7.1.3 解决 72](#_Toc49427885)

# AddressSanitizer简介

## 简介

AddressSanitizer用于检测android native层的内存覆盖，本文档适用于需要在android native层调试的内存覆盖的同事，及其他对此感兴趣的同事。

### AddressSanitizer

AddressSanitizer (ASan) 是一种基于编译器的快速检测工具，用于检测代码中的内存错误。



* AddressSanitizer 可以检测以下问题：
* 堆栈和堆缓冲区上溢/下溢
* 释放之后的堆使用情况
* 超出范围的堆栈使用情况
* 重复释放/错误释放
* AddressSanitizer 的限制：
* ASan 不能检测未初始化的读取和内存泄漏。
* AddressSanitizer 无法检查 Java 代码，但可以检测 JNI 库中的错误。

ASan 可同时在 32 位和 64 位 ARM 以及 x86 和 x86-64 上运行。

ASan 的 CPU 开销约为 2 倍，代码大小开销在一半到 2 倍之间，并且内存开销很大（具体取决于分配模式，但约为 2 倍）。

ASan 的原理简单来说是申请映射内存，并在映射内存中通过对申请内存的前后地址中放置不同的值用于标记边界，当访问相关地址时先监测映射内存中的标记，用于判断是否覆盖。

### 实际功能验证

我们对对常见的AddressSanitizer支持的类型进行了验证（详细内容见最后一章）：

* double-free （重复释放）
* wild pointer free （野指针）
* stack-buffer-overflow （栈溢出）
* heap-buffer-overflow （堆溢出）
* heap-use-after-free （释放后使用）
* stack-use-after-scope （范围外使用）



## 编译

### 使用 Clang 编译

要编译可通过 ASan 进行测试的二进制文件，第一步是要确保您的代码是使用 Clang 编译的。

ASan 的 master 分支会默认执行这一步骤，因此您无需执行任何操作。

如果您确认自己要测试的模块是使用 GCC 编译的，则需要向编译规则中添加以下行从而切换至 Clang。

|  |
| --- |
| LOCAL\_CLANG:=true |

Clang 也可以发现 GCC 遗漏的代码错误。

### 限制

以下目标不能使用 ASan 进行编译：

* 静态关联的可执行文件。
* LOCAL\_CLANG:=false 目标
* 不会针对 SANITIZE\_TARGET=address 进行 ASan 操作的 LOCAL\_SANITIZE:=false

在 SANITIZE\_TARGET 编译中，系统会跳过此类可执行文件，且会将第一次 make 调用中编译的版本留在 /system/bin 中。

## 其他

### 使用分支

本文档以 紫光展锐 AndroidR 平台为例，介绍如何使用 AddressSanitizer 来编译和运行 Android 平台的各个组成部分。

本文档主要基于 sprdroidr\_trunk 分支

### 更多信息

关于Asan的信息可参考gihub 及 android asan介绍

<https://github.com/google/sanitizers/wiki/AddressSanitizer>

<https://source.android.google.cn/devices/tech/debug/asan>

<https://developer.android.google.cn/ndk/guides/asan>

# 在可执行文件中使用AddressSanitizer

## 使用方法

使用 AddressSanitizer 编译可执行文件

将 打开ASan的编译规则 添加到可执行文件的编译规则文件中：

* 对于使用Android.mk文件的模块，在Android.mk中添加以下行：

|  |
| --- |
| LOCAL\_SANITIZE:=address |

* 对于使用Android.bp文件的模块，在Android.mk中添加以下行：

|  |
| --- |
| sanitize: {  address: true  }, |

之后重新编译替换可执行程序。并将可执行程序替换到手机中。

检测到错误时，ASan 会向标准输出和 logcat 发送一份详细报告，然后让相应进程崩溃。

## 以surfaceflinger为例

我们以surfaceflinger模块为例，看一下如何使用

### 打开开关

修改surfaceflinger的编译配置文件打开ASan开关

<http://review.source.unisoc.com/gerrit/#/c/694117/>

|  |
| --- |
| sprdroidr\_trunk/frameworks/native/services/surfaceflinger$ git diff  diff --git a/services/surfaceflinger/Android.bp b/services/surfaceflinger/Android.bp  index 4cd0a13..2d23146 100644  --- a/services/surfaceflinger/Android.bp  +++ b/services/surfaceflinger/Android.bp  @@ -218,6 +218,7 @@ filegroup {    cc\_binary {  name: "surfaceflinger",  + sanitize: { address: true },  defaults: ["libsurfaceflinger\_binary"],  init\_rc: ["surfaceflinger.rc"],  srcs: [":surfaceflinger\_binary\_sources"], |

之后我们将编译出的 system/bin/ surfaceflinger 替换到手机即可。

### 模拟一个问题

为验证ASan是否生效，我们修改surfaceflinger代码，制造一个问题，看ASan 是否可以检测到，我们模拟一个范围外使用的问题：

<http://review.source.unisoc.com/gerrit/#/c/694117/>

|  |
| --- |
| sprdroidr\_trunk/frameworks/native/services/surfaceflinger$ git diff  diff --git a/services/surfaceflinger/main\_surfaceflinger.cpp b/services/surfaceflinger/main\_surfaceflinger.cpp  index 2b8424c..51f8503 100644  --- a/services/surfaceflinger/main\_surfaceflinger.cpp  +++ b/services/surfaceflinger/main\_surfaceflinger.cpp  @@ -77,6 +77,16 @@ static status\_t startDisplayService() {  return err;  }    +volatile int \*p = 0;  +int test\_test() {  + {  + int x = 0;  + p = &x;  + }  + \*p = 5;  + return 0;  +}  +  int main(int, char\*\*) {  signal(SIGPIPE, SIG\_IGN);    @@ -105,6 +115,8 @@ int main(int, char\*\*) {  // Do this after the binder thread pool init  if (cpusets\_enabled()) set\_cpuset\_policy(0, SP\_SYSTEM);    + test\_test();  +  // initialize before clients can connect  flinger->init(); |

之后我们将编译出的 system/bin/ surfaceflinger 替换到手机即可。

### 发现问题

我们将编译出的 system/bin/ surfaceflinger 替换到手机，并重启手机，之后在log中我们可以看会报如下错误：

|  |
| --- |
| 07-17 15:23:29.109 5695 5695 I surfaceflinger: =================================================================  07-17 15:23:29.110 5695 5695 I surfaceflinger: ==5695==ERROR: AddressSanitizer: stack-use-after-scope on address 0xbeb1e2a0 at pc 0x0a5b7eb5 bp 0xbeb1e258 sp 0xbeb1e254  07-17 15:23:29.111 5695 5695 I surfaceflinger: WRITE of size 4 at 0xbeb1e2a0 thread T0  07-17 15:23:29.141 5695 5695 I surfaceflinger: #0 0xa5b7eb2 (/system/bin/surfaceflinger+0x2eb2)  07-17 15:23:29.141 5695 5695 I surfaceflinger: #1 0xb04fbe88 (/apex/com.android.runtime/lib/bionic/libc.so+0x5ce88)  07-17 15:23:29.142 5695 5695 I surfaceflinger: #2 0xa5b74fc (/system/bin/surfaceflinger+0x24fc)  07-17 15:23:29.143 5695 5695 I surfaceflinger:  07-17 15:23:29.143 5695 5695 I surfaceflinger: Address 0xbeb1e2a0 is located in stack of thread T0  07-17 15:23:29.144 5695 5695 I surfaceflinger: at offset 64 in frame  07-17 15:23:29.145 5695 5695 I surfaceflinger: #0 0xa5b7626 (/system/bin/surfaceflinger+0x2626)  07-17 15:23:29.145 5695 5695 I surfaceflinger:  07-17 15:23:29.146 5695 5695 I surfaceflinger: This frame has 11 object(s):  07-17 15:23:29.147 5695 5695 I surfaceflinger: [16, 20) 'displayservice.i' (line 70)  07-17 15:23:29.147 5695 5695 I surfaceflinger: [32, 44) 'ref.tmp.i62' (line 71)  07-17 15:23:29.148 5695 5695 I surfaceflinger: [64, 68) 'x.i' (line 83) <== Memory access at offset 64 is inside this variable  07-17 15:23:29.148 5695 5695 I surfaceflinger: [80, 92) 'ref.tmp.i' (line 50)  07-17 15:23:29.149 5695 5695 I surfaceflinger: [112, 124) 'ref.tmp6.i' (line 56)  07-17 15:23:29.149 5695 5695 I surfaceflinger: [144, 148) 'ref.tmp' (line 100)  07-17 15:23:29.150 5695 5695 I surfaceflinger: [160, 164) 'ps' (line 103)  07-17 15:23:29.150 5695 5695 I surfaceflinger: [176, 180) 'flinger' (line 107)  07-17 15:23:29.151 5695 5695 I surfaceflinger: [192, 196) 'sm' (line 124)  07-17 15:23:29.151 5695 5695 I surfaceflinger: [208, 212) 'ref.tmp14' (line 125)  07-17 15:23:29.152 5695 5695 I surfaceflinger: [224, 228) 'ref.tmp17' (line 125)  07-17 15:23:29.152 5695 5695 I surfaceflinger: HINT: this may be a false positive if your program uses some custom stack unwind mechanism, swapcontext or vfork  07-17 15:23:29.152 5695 5695 I surfaceflinger: (longjmp and C++ exceptions \*are\* supported)  07-17 15:23:29.153 5695 5695 I surfaceflinger: SUMMARY: AddressSanitizer: stack-use-after-scope (/system/bin/surfaceflinger+0x2eb2)  07-17 15:23:29.154 5695 5695 I surfaceflinger: Shadow bytes around the buggy address:  07-17 15:23:29.154 5695 5695 I surfaceflinger: 0xa819bc00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 15:23:29.154 5695 5695 I surfaceflinger: 0xa819bc10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 15:23:29.154 5695 5695 I surfaceflinger: 0xa819bc20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 15:23:29.155 5695 5695 I surfaceflinger: 0xa819bc30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 15:23:29.155 5695 5695 I surfaceflinger: 0xa819bc40: 00 00 00 00 00 00 00 00 00 00 00 00 f1 f1 f8 f2  07-17 15:23:29.155 5695 5695 I surfaceflinger: =>0xa819bc50: f8 f8 f2 f2[f8]f2 f8 f8 f2 f2 f8 f8 f2 f2 f8 f2  07-17 15:23:29.155 5695 5695 I surfaceflinger: 0xa819bc60: 04 f2 04 f2 f8 f2 f8 f2 f8 f3 00 00 00 00 00 00  07-17 15:23:29.155 5695 5695 I surfaceflinger: 0xa819bc70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 15:23:29.155 5695 5695 I surfaceflinger: 0xa819bc80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 15:23:29.155 5695 5695 I surfaceflinger: 0xa819bc90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 15:23:29.155 5695 5695 I surfaceflinger: 0xa819bca0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 15:23:29.155 5695 5695 I surfaceflinger: Shadow byte legend (one shadow byte represents 8 application bytes):  07-17 15:23:29.155 5695 5695 I surfaceflinger: Addressable: 00  07-17 15:23:29.155 5695 5695 I surfaceflinger: Partially addressable: 01 02 03 04 05 06 07  07-17 15:23:29.155 5695 5695 I surfaceflinger: Heap left redzone: fa  07-17 15:23:29.155 5695 5695 I surfaceflinger: Freed heap region: fd  07-17 15:23:29.155 5695 5695 I surfaceflinger: Stack left redzone: f1  07-17 15:23:29.155 5695 5695 I surfaceflinger: Stack mid redzone: f2  07-17 15:23:29.155 5695 5695 I surfaceflinger: Stack right redzone: f3  07-17 15:23:29.155 5695 5695 I surfaceflinger: Stack after return: f5  07-17 15:23:29.156 5695 5695 I surfaceflinger: Stack use after scope: f8  07-17 15:23:29.156 5695 5695 I surfaceflinger: Global redzone: f9  07-17 15:23:29.156 5695 5695 I surfaceflinger: Global init order: f6  07-17 15:23:29.156 5695 5695 I surfaceflinger: Poisoned by user: f7  07-17 15:23:29.156 5695 5695 I surfaceflinger: Container overflow: fc  07-17 15:23:29.156 5695 5695 I surfaceflinger: Array cookie: ac  07-17 15:23:29.156 5695 5695 I surfaceflinger: Intra object redzone: bb  07-17 15:23:29.156 5695 5695 I surfaceflinger: ASan internal: fe  07-17 15:23:29.156 5695 5695 I surfaceflinger: Left alloca redzone: ca  07-17 15:23:29.156 5695 5695 I surfaceflinger: Right alloca redzone: cb  07-17 15:23:29.157 5695 5695 I surfaceflinger: Shadow gap: cc  07-17 15:23:29.157 5695 5695 I surfaceflinger: ==5695==ABORTING |

我们看到错误类型为： stack-use-after-scope，

调用栈为：

|  |
| --- |
| 07-17 15:23:29.111 5695 5695 I surfaceflinger: WRITE of size 4 at 0xbeb1e2a0 thread T0  07-17 15:23:29.141 5695 5695 I surfaceflinger: #0 0xa5b7eb2 (/system/bin/surfaceflinger+0x2eb2)  07-17 15:23:29.141 5695 5695 I surfaceflinger: #1 0xb04fbe88 (/apex/com.android.runtime/lib/bionic/libc.so+0x5ce88)  07-17 15:23:29.142 5695 5695 I surfaceflinger: #2 0xa5b74fc (/system/bin/surfaceflinger+0x24fc)  07-17 15:23:29.143 5695 5695 I surfaceflinger:  07-17 15:23:29.143 5695 5695 I surfaceflinger: Address 0xbeb1e2a0 is located in stack of thread T0  07-17 15:23:29.144 5695 5695 I surfaceflinger: at offset 64 in frame  07-17 15:23:29.145 5695 5695 I surfaceflinger: #0 0xa5b7626 (/system/bin/surfaceflinger+0x2626)  07-17 15:23:29.145 5695 5695 I surfaceflinger: |

同时AddressSanitizer在检查到错误之后，主动生成一个SIGABRT 的NativeCrash

|  |
| --- |
| 07-17 15:23:29.158 5695 5695 F libc : Fatal signal 6 (SIGABRT), code -1 (SI\_QUEUE) in tid 5695 (surfaceflinger), pid 5695 (surfaceflinger)  07-17 15:23:29.222 5728 5728 F DEBUG : pid: 5695, tid: 5695, name: surfaceflinger >>> /system/bin/surfaceflinger <<<  07-17 15:23:29.222 5728 5728 F DEBUG : #0 0xa5b7eb2 (/system/bin/surfaceflinger+0x2eb2)  07-17 15:23:29.222 5728 5728 F DEBUG : #2 0xa5b74fc (/system/bin/surfaceflinger+0x24fc)  07-17 15:23:29.222 5728 5728 F DEBUG : #0 0xa5b7626 (/system/bin/surfaceflinger+0x2626)  07-17 15:23:29.222 5728 5728 F DEBUG : SUMMARY: AddressSanitizer: stack-use-after-scope (/system/bin/surfaceflinger+0x2eb2)  07-17 15:23:29.254 5728 5728 F DEBUG : #06 pc 00002eb1 /system/bin/surfaceflinger (main+2200) (BuildId: b5df860eb2d8583706c319c06fa6d768)  07-17 15:23:34.109 5849 5849 I surfaceflinger: =================================================================  07-17 15:23:34.110 5849 5849 I surfaceflinger: ==5849==ERROR: AddressSanitizer: stack-use-after-scope on address 0xbefda2a0 at pc 0x0944eeb5 bp 0xbefda258 sp 0xbefda254  07-17 15:23:34.110 5849 5849 I surfaceflinger: WRITE of size 4 at 0xbefda2a0 thread T0  07-17 15:23:34.139 5849 5849 I surfaceflinger: #0 0x944eeb2 (/system/bin/surfaceflinger+0x2eb2)  07-17 15:23:34.140 5849 5849 I surfaceflinger: #1 0xa97fee88 (/apex/com.android.runtime/lib/bionic/libc.so+0x5ce88)  07-17 15:23:34.140 5849 5849 I surfaceflinger: #2 0x944e4fc (/system/bin/surfaceflinger+0x24fc)  07-17 15:23:34.141 5849 5849 I surfaceflinger:  07-17 15:23:34.141 5849 5849 I surfaceflinger: Address 0xbefda2a0 is located in stack of thread T0  07-17 15:23:34.142 5849 5849 I surfaceflinger: at offset 64 in frame  07-17 15:23:34.143 5849 5849 I surfaceflinger: #0 0x944e626 (/system/bin/surfaceflinger+0x2626)  07-17 15:23:34.143 5849 5849 I surfaceflinger:  07-17 15:23:34.144 5849 5849 I surfaceflinger: This frame has 11 object(s):  07-17 15:23:34.144 5849 5849 I surfaceflinger: [16, 20) 'displayservice.i' (line 70)  07-17 15:23:34.144 5849 5849 I surfaceflinger: [32, 44) 'ref.tmp.i62' (line 71)  07-17 15:23:34.145 5849 5849 I surfaceflinger: [64, 68) 'x.i' (line 83) <== Memory access at offset 64 is inside this variable  07-17 15:23:34.145 5849 5849 I surfaceflinger: [80, 92) 'ref.tmp.i' (line 50)  07-17 15:23:34.146 5849 5849 I surfaceflinger: [112, 124) 'ref.tmp6.i' (line 56)  07-17 15:23:34.146 5849 5849 I surfaceflinger: [144, 148) 'ref.tmp' (line 100)  07-17 15:23:34.147 5849 5849 I surfaceflinger: [160, 164) 'ps' (line 103)  07-17 15:23:34.147 5849 5849 I surfaceflinger: [176, 180) 'flinger' (line 107)  07-17 15:23:34.148 5849 5849 I surfaceflinger: [192, 196) 'sm' (line 124)  07-17 15:23:34.148 5849 5849 I surfaceflinger: [208, 212) 'ref.tmp14' (line 125)  07-17 15:23:34.149 5849 5849 I surfaceflinger: [224, 228) 'ref.tmp17' (line 125)  07-17 15:23:34.149 5849 5849 I surfaceflinger: HINT: this may be a false positive if your program uses some custom stack unwind mechanism, swapcontext or vfork  07-17 15:23:34.150 5849 5849 I surfaceflinger: (longjmp and C++ exceptions \*are\* supported)  07-17 15:23:34.150 5849 5849 I surfaceflinger: SUMMARY: AddressSanitizer: stack-use-after-scope (/system/bin/surfaceflinger+0x2eb2)  07-17 15:23:34.151 5849 5849 I surfaceflinger: Shadow bytes around the buggy address:  07-17 15:23:34.152 5849 5849 I surfaceflinger: 0x9ffc3400: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 15:23:34.152 5849 5849 I surfaceflinger: 0x9ffc3410: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 15:23:34.152 5849 5849 I surfaceflinger: 0x9ffc3420: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 15:23:34.152 5849 5849 I surfaceflinger: 0x9ffc3430: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 15:23:34.152 5849 5849 I surfaceflinger: 0x9ffc3440: 00 00 00 00 00 00 00 00 00 00 00 00 f1 f1 f8 f2  07-17 15:23:34.152 5849 5849 I surfaceflinger: =>0x9ffc3450: f8 f8 f2 f2[f8]f2 f8 f8 f2 f2 f8 f8 f2 f2 f8 f2  07-17 15:23:34.152 5849 5849 I surfaceflinger: 0x9ffc3460: 04 f2 04 f2 f8 f2 f8 f2 f8 f3 00 00 00 00 00 00  07-17 15:23:34.152 5849 5849 I surfaceflinger: 0x9ffc3470: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 15:23:34.152 5849 5849 I surfaceflinger: 0x9ffc3480: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 15:23:34.152 5849 5849 I surfaceflinger: 0x9ffc3490: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 15:23:34.152 5849 5849 I surfaceflinger: 0x9ffc34a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 15:23:34.152 5849 5849 I surfaceflinger: Shadow byte legend (one shadow byte represents 8 application bytes):  07-17 15:23:34.152 5849 5849 I surfaceflinger: Addressable: 00  07-17 15:23:34.152 5849 5849 I surfaceflinger: Partially addressable: 01 02 03 04 05 06 07  07-17 15:23:34.152 5849 5849 I surfaceflinger: Heap left redzone: fa  07-17 15:23:34.153 5849 5849 I surfaceflinger: Freed heap region: fd  07-17 15:23:34.153 5849 5849 I surfaceflinger: Stack left redzone: f1  07-17 15:23:34.153 5849 5849 I surfaceflinger: Stack mid redzone: f2  07-17 15:23:34.153 5849 5849 I surfaceflinger: Stack right redzone: f3  07-17 15:23:34.153 5849 5849 I surfaceflinger: Stack after return: f5  07-17 15:23:34.153 5849 5849 I surfaceflinger: Stack use after scope: f8  07-17 15:23:34.153 5849 5849 I surfaceflinger: Global redzone: f9  07-17 15:23:34.153 5849 5849 I surfaceflinger: Global init order: f6  07-17 15:23:34.153 5849 5849 I surfaceflinger: Poisoned by user: f7  07-17 15:23:34.153 5849 5849 I surfaceflinger: Container overflow: fc  07-17 15:23:34.153 5849 5849 I surfaceflinger: Array cookie: ac  07-17 15:23:34.153 5849 5849 I surfaceflinger: Intra object redzone: bb  07-17 15:23:34.153 5849 5849 I surfaceflinger: ASan internal: fe  07-17 15:23:34.153 5849 5849 I surfaceflinger: Left alloca redzone: ca  07-17 15:23:34.154 5849 5849 I surfaceflinger: Right alloca redzone: cb  07-17 15:23:34.154 5849 5849 I surfaceflinger: Shadow gap: cc  07-17 15:23:34.154 5849 5849 I surfaceflinger: ==5849==ABORTING |

### 解析后的log

调用栈解析结果如下：（调用栈解析方法见后面章节）

|  |
| --- |
| =================================================================  ==5695==ERROR: AddressSanitizer: stack-use-after-scope on address 0xbeb1e2a0 at pc 0x0a5b7eb5 bp 0xbeb1e258 sp 0xbeb1e254  WRITE of size 4 at 0xbeb1e2a0 thread T0  #0 0xa5b7eb2 in \_Z9test\_testv frameworks/native/services/surfaceflinger/main\_surfaceflinger.cpp:86  #1 0xa5b7eb2 in main frameworks/native/services/surfaceflinger/main\_surfaceflinger.cpp:118  #2 0xb04fbe88 in \_\_libc\_init bionic/libc/bionic/libc\_init\_dynamic.cpp:152  #3 0xa5b74fc in \_start\_main bionic/libc/arch-common/bionic/crtbegin.c:45  Address 0xbeb1e2a0 is located in stack of thread T0  at offset 64 in frame  #4 0xa5b7626 in main frameworks/native/services/surfaceflinger/main\_surfaceflinger.cpp:90  This frame has 11 object(s):  [16, 20) 'displayservice.i' (line 70)  [32, 44) 'ref.tmp.i62' (line 71)  [64, 68) 'x.i' (line 83) <== Memory access at offset 64 is inside this variable  [80, 92) 'ref.tmp.i' (line 50)  [112, 124) 'ref.tmp6.i' (line 56)  [144, 148) 'ref.tmp' (line 100)  [160, 164) 'ps' (line 103)  [176, 180) 'flinger' (line 107)  [192, 196) 'sm' (line 124)  [208, 212) 'ref.tmp14' (line 125)  [224, 228) 'ref.tmp17' (line 125)  HINT: this may be a false positive if your program uses some custom stack unwind mechanism, swapcontext or vfork  (longjmp and C++ exceptions \*are\* supported)  SUMMARY: AddressSanitizer: stack-use-after-scope (/system/bin/surfaceflinger+0x2eb2)  Shadow bytes around the buggy address:  0xa819bc00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0xa819bc10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0xa819bc20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0xa819bc30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0xa819bc40: 00 00 00 00 00 00 00 00 00 00 00 00 f1 f1 f8 f2  =>0xa819bc50: f8 f8 f2 f2[f8]f2 f8 f8 f2 f2 f8 f8 f2 f2 f8 f2  0xa819bc60: 04 f2 04 f2 f8 f2 f8 f2 f8 f3 00 00 00 00 00 00  0xa819bc70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0xa819bc80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0xa819bc90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0xa819bca0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  Shadow byte legend (one shadow byte represents 8 application bytes):  Addressable: 00  Partially addressable: 01 02 03 04 05 06 07  Heap left redzone: fa  Freed heap region: fd  Stack left redzone: f1  Stack mid redzone: f2  Stack right redzone: f3  Stack after return: f5  Stack use after scope: f8  Global redzone: f9  Global init order: f6  Poisoned by user: f7  Container overflow: fc  Array cookie: ac  Intra object redzone: bb  ASan internal: fe  Left alloca redzone: ca  Right alloca redzone: cb  Shadow gap: cc  ==5695==ABORTING |

我们可以看到，可以解析出文件名，以及函数所在行

|  |
| --- |
| WRITE of size 4 at 0xbeb1e2a0 thread T0  #0 0xa5b7eb2 in \_Z9test\_testv frameworks/native/services/surfaceflinger/main\_surfaceflinger.cpp:86  #1 0xa5b7eb2 in main frameworks/native/services/surfaceflinger/main\_surfaceflinger.cpp:118  #2 0xb04fbe88 in \_\_libc\_init bionic/libc/bionic/libc\_init\_dynamic.cpp:152  #3 0xa5b74fc in \_start\_main bionic/libc/arch-common/bionic/crtbegin.c:45  Address 0xbeb1e2a0 is located in stack of thread T0  at offset 64 in frame  #4 0xa5b7626 in main frameworks/native/services/surfaceflinger/main\_surfaceflinger.cpp:90 |

### 对应代码

在 86 行存在范围外使用的情况

|  |
| --- |
| 78 }  79  80 volatile int \*p = 0;  81 int test\_test() {  82 {  83 int x = 0;  84 p = &x;  85 }  86 \*p = 5;  87 return 0;  88 }  89  90 int main(int, char\*\*) {  91 signal(SIGPIPE, SIG\_IGN);  92  93 hardware::configureRpcThreadpool(1 /\* maxThreads \*/,  94 false /\* callerWillJoin \*/);  95  96 startGraphicsAllocatorService();  97  98 // When SF is launched in its own process, limit the number of  99 // binder threads to 4.  100 ProcessState::self()->setThreadPoolMaxThreadCount(4);  101  102 // start the thread pool  103 sp<ProcessState> ps(ProcessState::self());  104 ps->startThreadPool();  105  106 // instantiate surfaceflinger  107 sp<SurfaceFlinger> flinger = surfaceflinger::createSurfaceFlinger();  108  109 setpriority(PRIO\_PROCESS, 0, PRIORITY\_URGENT\_DISPLAY);  110  111 set\_sched\_policy(0, SP\_FOREGROUND);  112  113 // Put most SurfaceFlinger threads in the system-background cpuset  114 // Keeps us from unnecessarily using big cores  115 // Do this after the binder thread pool init  116 if (cpusets\_enabled()) set\_cpuset\_policy(0, SP\_SYSTEM);  117  118 test\_test();  119  120 // initialize before clients can connect  121 flinger->init(); |

## 以ylog为例演示

我们以ylog模块为例，看一下如何使用

### 打开开关

编译配置文件打开ASan开关

<http://review.source.unisoc.com/gerrit/#/c/693789/>

|  |
| --- |
| sprdroidr\_trunk/vendor/sprd/tools/ylog/ylog/plugin/sgm.toolkits$ git diff  diff --git a/plugin/sgm.toolkits/Android.mk b/plugin/sgm.toolkits/Android.mk  old mode 100755  new mode 100644  index 1a7b22f..1385c10  --- a/plugin/sgm.toolkits/Android.mk  +++ b/plugin/sgm.toolkits/Android.mk  @@ -4,5 +4,6 @@ include $(CLEAR\_VARS)  LOCAL\_SRC\_FILES := sgm.cpu\_memory.c  LOCAL\_MODULE := sgm.cpu\_memory  LOCAL\_MODULE\_TAGS := optional  +LOCAL\_SANITIZE := address  include $(BUILD\_EXECUTABLE)  CUSTOM\_MODULES += sgm.cpu\_memory # no need appending this module in PRODUCT\_PACKAGES |

### 模拟一个问题

<http://review.source.unisoc.com/gerrit/#/c/693789/>

|  |
| --- |
| sprdroidr\_trunk/vendor/sprd/tools/ylog/ylog/plugin/sgm.toolkits$ git diff  diff --git a/plugin/sgm.toolkits/sgm.cpu\_memory.c b/plugin/sgm.toolkits/sgm.cpu\_memory.c  old mode 100755  new mode 100644  index 84f2eef..a577d9b  --- a/plugin/sgm.toolkits/sgm.cpu\_memory.c  +++ b/plugin/sgm.toolkits/sgm.cpu\_memory.c  @@ -539,7 +539,15 @@ static int parse\_cmdline(int argc, char \*\*argv) {  }  return ret;  }  +int \*p = 0;  +int test\_test() {  + {  + int x = 0;  + p = &x;  + }  + \*p = 5;  + return 0;  +}  int main(int argc, char \*argv[]) {  int ret;  int FD;  @@ -559,6 +567,7 @@ int main(int argc, char \*argv[]) {  signal(SIGTERM, cpu\_sig\_handler);  signal(SIGCHLD, cpu\_sig\_killchld);  + test\_test();  while (!is\_cpu\_basic\_quit) {  gettimeofday(&tv, NULL);  cur = localtime(&tv.tv\_sec); |

合入后编译，生成/system/bin/sgm.cpu\_memory文件将其替换到手机。

在adb shell中执行 sgm.cpu\_memory -t 1000运行程序

### 发现问题

捕获异常的log

|  |
| --- |
| ==17149==ERROR: AddressSanitizer: stack-use-after-scope on address 0xbeb11110 at pc 0x010ad090 bp 0xbeb110f8 sp 0xbeb110f4  WRITE of size 4 at 0xbeb11110 thread T0  #0 0x10ad08c (/system/bin/sgm.cpu\_memory+0x308c)  #1 0xa974be88 (/apex/com.android.runtime/lib/bionic/libc.so+0x5ce88)  #2 0x10ac440 (/system/bin/sgm.cpu\_memory+0x2440)  Address 0xbeb11110 is located in stack of thread T0 at offset 16 in frame  #0 0x10ac460 (/system/bin/sgm.cpu\_memory+0x2460)  This frame has 4 object(s):  [16, 20) 'x.i' (line 531) <== Memory access at offset 16 is inside this variable  [32, 40) 'tp' (line 542)  [64, 72) 'tv' (line 543)  [96, 352) 'cur\_time' (line 544)  HINT: this may be a false positive if your program uses some custom stack unwind mechanism, swapcontext or vfork  (longjmp and C++ exceptions \*are\* supported)  SUMMARY: AddressSanitizer: stack-use-after-scope (/system/bin/sgm.cpu\_memory+0x308c)  Shadow bytes around the buggy address:  0xa10ba1d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0xa10ba1e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0xa10ba1f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0xa10ba200: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0xa10ba210: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  =>0xa10ba220: f1 f1[f8]f2 00 f2 f2 f2 00 f2 f2 f2 00 00 00 00  0xa10ba230: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0xa10ba240: 00 00 00 00 00 00 00 00 00 00 00 00 f3 f3 f3 f3  0xa10ba250: f3 f3 f3 f3 00 00 00 00 00 00 00 00 00 00 00 00  0xa10ba260: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0xa10ba270: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  Shadow byte legend (one shadow byte represents 8 application bytes):  Addressable: 00  Partially addressable: 01 02 03 04 05 06 07  Heap left redzone: fa  Freed heap region: fd  Stack left redzone: f1  Stack mid redzone: f2  Stack right redzone: f3  Stack after return: f5  Stack use after scope: f8  Global redzone: f9  Global init order: f6  Poisoned by user: f7  Container overflow: fc  Array cookie: ac  Intra object redzone: bb  ASan internal: fe  Left alloca redzone: ca  Right alloca redzone: cb  Shadow gap: cc  ==17149==ABORTING |

### 解析后的log

|  |
| --- |
| AddressSanitizer: stack-use-after-scope on address 0xbeb11110 at pc 0x010ad090 bp 0xbeb110f8 sp 0xbeb110f4  WRITE of size 4 at 0xbeb11110 thread T0  #0 0x10ad08c in test\_test vendor/sprd/tools/ylog/ylog/plugin/sgm.toolkits/sgm.cpu\_memory.c:534  #1 0x10ad08c in main vendor/sprd/tools/ylog/ylog/plugin/sgm.toolkits/sgm.cpu\_memory.c:557  #2 0xa974be88 in \_\_libc\_init bionic/libc/bionic/libc\_init\_dynamic.cpp:152  #3 0x10ac440 in \_start\_main bionic/libc/arch-common/bionic/crtbegin.c:45  Address 0xbeb11110 is located in stack of thread T0 at offset 16 in frame  #4 0x10ac460 in main vendor/sprd/tools/ylog/ylog/plugin/sgm.toolkits/sgm.cpu\_memory.c:538  This frame has 4 object(s):  [16, 20) 'x.i' (line 531) <== Memory access at offset 16 is inside this variable  [32, 40) 'tp' (line 542)  [64, 72) 'tv' (line 543)  [96, 352) 'cur\_time' (line 544)  this may be a false positive if your program uses some custom stack unwind mechanism, swapcontext or vfork  (longjmp and C++ exceptions \*are\* supported)  AddressSanitizer: stack-use-after-scope (/system/bin/sgm.cpu\_memory+0x308c)  Shadow bytes around the buggy address:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  f1 f1[f8]f2 00 f2 f2 f2 00 f2 f2 f2 00 00 00 00  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 00 00 00 00 f3 f3 f3 f3  f3 f3 f3 f3 00 00 00 00 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  Shadow byte legend (one shadow byte represents 8 application bytes):  00  01 02 03 04 05 06 07  fa  fd  f1  f2  f3  f5  f8  f9  f6  f7  fc  ac  bb  fe  ca  cb  cc  ==17149==ABORTING |

### 对应代码

|  |
| --- |
| 526 }  527 int \*p = 0;  528 int test\_test() {  529 printf("+sgm.cpu\_memory test\_test ");  530 {  531 int x = 0;  532 p = &x;  533 }  534 \*p = 5;  535 printf("-sgm.cpu\_memory test\_test ");  536 return 0;  537 }  538 int main(int argc, char \*argv[]) {  539 int ret;  540 int FD;  541 struct tm \*cur;  542 struct timespec tp;  543 struct timeval tv;  544 char cur\_time[256];  545  546 ret = parse\_cmdline(argc, argv);  547 if (ret < 0)  548 return -1;  549 setbuf(stdout, NULL);  550 setbuf(stderr, NULL);  551  552 if (no\_header == false)  553 logs\_print(0, PRINT\_HEAD, output);  554 signal(SIGTERM, cpu\_sig\_handler);  555 signal(SIGCHLD, cpu\_sig\_killchld);  556  557 test\_test();  558 while (!is\_cpu\_basic\_quit) {  559 gettimeofday(&tv, NULL);  560 cur = localtime(&tv.tv\_sec);  561 clock\_gettime(CLOCK\_MONOTONIC\_RAW, &tp);  562 sprintf(cur\_time, "%d-%d %d:%d:%d.%03ld<%5lu.%06lu>", cur->tm\_mon + 1, cur->tm\_mday, cur->tm\_hour, cur->tm\_min, cur->tm\_sec, tv.tv\_usec / 1000, tp.tv\_sec, tp.tv\_nsec / 1000);  563  564 logs\_print(cur\_time, PRINT\_BODY, output);  565 msleep(interval);  566 }  567 return 0; |

# 在动态库中使用AddressSanitizer

## 使用方法

### ASan自身的限制

基于 ASan 的工作原理，不是通过 ASan 编译的可执行文件将无法使用通过 ASan 编译的库。

当我们调试共享库的时候，共享库有可能会被多个可执行文件调用，如果这些可执行文件中没有使用ASan 编译的情况，就会遇到错误：在运行时，如果将 ASan 库加载到错误的进程中，系统将显示以 \_asan 或 \_sanitizer 开头的消息，提示您有无法解析的符号。



上图中：

* 蓝色框：使用asan编译的库和可执行文件
* 黄色框：未用asan编译的库和可执行文件
* 绿色实线：两端模块均使用asan编译，相关路径调用时出现问题可以被asan检测到
* 绿色虚线：箭头指向模块未用asan编译，箭头指向模块中内存问题不能被asan检测到
* 桔色虚线：未用asan编译的库，调用使用asan编译的库，会报错

为了避免不是通过 ASan 编译的可执行文件将无法使用通过 ASan 编译的库的情况，Google提供了， 2 个副本的方法，来解决这种问题，

* 使用Asan编译的可执行文件调用使用Asan编译的的库
* 未使用Asan编译的可执行文件调用使用未用Asan编译的的库



### 在编译文件中打开ASan

可以针对相应的模块向 Android.mk 中添加以下内容：

* 对于使用Android.mk文件的模块，在Android.mk中添加以下行：

|  |
| --- |
| LOCAL\_SANITIZE:=address  LOCAL\_MODULE\_RELATIVE\_PATH := asan |

这样一来，系统会将库放置到 /system/lib/asan（而非 /system/lib）中。

然后，将生成的的out目录中的/system/lib/asan中的动态库push到手机上的/system/lib/asan中。

* 对于使用Android.bp文件的模块，在Android.mk中添加以下行：

|  |
| --- |
| sanitize: {  address: true  }, |

对于Android.bp文件的模块，无需指定，系统编译完成后自动将文件放到/data/asan下。然后，将生成的的out目录中的/data/asan/system中的动态库push到手机上的/data/asan/system中。

### 文件执行

#### 待测可执行文件的执行：

如果待测可执行文件不是系统服务是普通可执行文件，使用以下方法运行您的可执行文件：

|  |
| --- |
| LD\_LIBRARY\_PATH=/system/lib/asan |

#### 系统服务的执行

如果待测文件是系统服务，还需要将以下内容添加到 /init.rc 或 /init.$device$.rc 的相应部分。

|  |
| --- |
| setenv LD\_LIBRARY\_PATH /system/lib/asan |

### 检查

启动之后我们还需要验证一下被测可执行文件使用的库是否来自/system/lib/asan。我们可以通过读取 /proc/$PID/maps，验证相应进程使用的库是否来自 /system/lib/asan（如果此库存在）。

如果不是，可能需要停用 SELinux，如下所示：

|  |
| --- |
| adb root  adb shell setenforce 0  # restart the process with adb shell kill $PID  # if it is a system service, or may be adb shell stop; adb shell start. |

如上所述，关闭SELinux后，需要将被测进程杀了重启，如果是系统服务的话，需要使用：adb shell stop; 和adb shell start来重启服务。

之后再次验证调用库是否为/system/lib/asan中库

### 注意

警告： LOCAL\_MODULE\_RELATIVE\_PATH 会将您的库移至 /system/lib/asan，也就是说该设置并不是编两套库出来，而是只编1套库放在在 /system/lib/asan里，所以如果删了out目录重新全编，就只在/system/lib/asan里有这个库，而/system/lib/下没有这个库，其他需要调用/system/lib/中该库的程序会找不到库而报错。

严重情况下造成编出来的映像可能会无法启动。

这是当前编译系统存在的一个令人遗憾的限制。

也就是说，如果非要这样用的话，需要先确保Android.mk中没有这个设置编一套库出来，然后修改Android.mk将这一句加上，再编一套库出来，并修改可执行文件的启动环境。期间不要删除out目录。

### 简单的办法

基于 ASan 的工作原理，不是通过 ASan 编译的可执行文件将无法使用通过 ASan 编译的库。也就是说，ASan 编译的可执行文件可以调用ASan 编译的库和非ASan 编译的库，前面官方给出的方法是两套库。我们在实际使用中，对于被较少可执行文件调用的共享库，可以采用的比较简单的办法：

* 将所有调用该库的可执行文件都使用 ASan 来编译，
* 或者确保在测试过程中不会用到那些没用ASan 编译可可执行文件。



这样我们就可以像前面一章修改可执行文件的方法一样，将打开ASan的编译规则添加到动态库的编译规则文件和可执行文件的编译规则文件中。

* 对于使用Android.mk文件的模块，在可执行文件或动态库的Android.mk中添加以下行：

|  |
| --- |
| LOCAL\_SANITIZE:=address |

之后重新编译动态库及可执行程序，然后，将 system/lib 或 system/lib64 下的生成的动态库或system/bin下可执行程序替换到手机。

* 对于使用Android.bp文件的模块，在可执行文件或动态库的Android.bp文件中添加以下行：

|  |
| --- |
| sanitize: {  address: true  }, |

之后重新编译动态库及可执行程序。并将/data/asan/system/lib64和/data/asan/system/lib动态以及system/bin下可执行程序替换到手机中。

## 简单调用调试举例-libsurfaceflinger.so

与之前版本不同的是，如果一个库只有一个可执行文件调用，该库直接放在/system/lib下，无需再放在放在asan目录下。

我们以调试libcameraservice.so库为例



### 配置文件的修改

调试libcameraservice.so库，需要修改调试libcameraservice.so库的编译配置及调用它的surfaceflinger的编译配置。

<http://review.source.unisoc.com/gerrit/#/c/694140/>

|  |
| --- |
| sprdroidr\_trunk/frameworks/native/services/surfaceflinger$ git diff  diff --git a/services/surfaceflinger/Android.bp b/services/surfaceflinger/Android.bp  index 4cd0a13..8126b1e 100644  --- a/services/surfaceflinger/Android.bp  +++ b/services/surfaceflinger/Android.bp  @@ -165,6 +165,7 @@ cc\_library\_shared {  // Please use libsurfaceflinger\_defaults to configure how the sources are  // built, so the same settings can be used elsewhere.  name: "libsurfaceflinger",  + sanitize: { address: true }, // libsurfaceflinger.so动态库  defaults: ["libsurfaceflinger\_production\_defaults"],  srcs: [  ":libsurfaceflinger\_sources",  @@ -218,6 +219,7 @@ filegroup {    cc\_binary {  name: "surfaceflinger",  + sanitize: { address: true }, // surfaceflinger 可执行文件  defaults: ["libsurfaceflinger\_binary"],  init\_rc: ["surfaceflinger.rc"],  srcs: [":surfaceflinger\_binary\_sources"], |

### 模拟一个问题

为验证ASan是否生效，我们修改surfaceflinger代码，制造一个问题，看ASan 是否可以检测到，我们制造一个范围外使用的问题：

<http://review.source.unisoc.com/gerrit/#/c/694140/>

|  |
| --- |
| sprdroidr\_trunk/frameworks/native/services/surfaceflinger$ git diff  diff --git a/services/surfaceflinger/SurfaceFlingerFactory.cpp b/services/surfaceflinger/SurfaceFlingerFactory.cpp  index e425b2a..83e3a63 100644  --- a/services/surfaceflinger/SurfaceFlingerFactory.cpp  +++ b/services/surfaceflinger/SurfaceFlingerFactory.cpp  @@ -39,6 +39,16 @@    namespace android::surfaceflinger {    +volatile int \*p = 0;  +int test\_test() {  + {  + int x = 0;  + p = &x;  + }  + \*p = 5;  + return 0;  +}  +  sp<SurfaceFlinger> createSurfaceFlinger() {  class Factory final : public surfaceflinger::Factory {  public:  @@ -136,6 +146,8 @@ sp<SurfaceFlinger> createSurfaceFlinger() {  };  static Factory factory;    + test\_test();  return new SurfaceFlinger(factory);  } |

### 编译替换生成库

系统编译完成后替换生成库

|  |
| --- |
| sprdroidr\_trunk/out/ $ find -name "libsurfaceflinger.so"  ./target/product/s9863a1h10\_go\_32b/system/lib/libsurfaceflinger.so  /target/product/s9863a1h10\_go\_32b/symbols/system/lib/libsurfaceflinger.so |

### 检查到问题Log如下

|  |
| --- |
| 07-17 12:34:17.900 11196 11196 I surfaceflinger: ==11196==ERROR: AddressSanitizer: stack-use-after-scope on address 0xbef83210 at pc 0xb160aae3 bp 0xbef831f8 sp 0xbef831f4  07-17 12:34:17.901 11196 11196 I surfaceflinger: WRITE of size 4 at 0xbef83210 thread T0  07-17 12:34:17.919 11196 11196 I surfaceflinger: #0 0xb160aae0 (/system/lib/libsurfaceflinger.so+0x27fae0)  07-17 12:34:17.920 11196 11196 I surfaceflinger: #1 0xec0d71a (/system/bin/surfaceflinger+0x271a)  07-17 12:34:17.920 11196 11196 I surfaceflinger: #2 0xb19ece88 (/apex/com.android.runtime/lib/bionic/libc.so+0x5ce88)  07-17 12:34:17.921 11196 11196 I surfaceflinger: #3 0xec0d434 (/system/bin/surfaceflinger+0x2434)  07-17 12:34:17.921 11196 11196 I surfaceflinger:  07-17 12:34:17.922 11196 11196 I surfaceflinger: Address 0xbef83210 is located in stack of thread T0  07-17 12:34:17.922 11196 11196 I surfaceflinger: at offset 16 in frame  07-17 12:34:17.922 11196 11196 I surfaceflinger: #0 0xb160a9a2 (/system/lib/libsurfaceflinger.so+0x27f9a2)  07-17 12:34:17.923 11196 11196 I surfaceflinger:  07-17 12:34:17.923 11196 11196 I surfaceflinger: This frame has 1 object(s):  07-17 12:34:17.924 11196 11196 I surfaceflinger: [16, 20) 'x.i' (line 29) <== Memory access at offset 16 is inside this variable  07-17 12:34:17.924 11196 11196 I surfaceflinger: HINT: this may be a false positive if your program uses some custom stack unwind mechanism, swapcontext or vfork  07-17 12:34:17.924 11196 11196 I surfaceflinger: (longjmp and C++ exceptions \*are\* supported)  07-17 12:34:17.925 11196 11196 I surfaceflinger: SUMMARY: AddressSanitizer: stack-use-after-scope (/system/lib/libsurfaceflinger.so+0x27fae0)  07-17 12:34:17.926 11196 11196 I surfaceflinger: Shadow bytes around the buggy address:  07-17 12:34:17.926 11196 11196 I surfaceflinger: 0xa7c585f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 12:34:17.926 11196 11196 I surfaceflinger: 0xa7c58600: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 12:34:17.926 11196 11196 I surfaceflinger: 0xa7c58610: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 12:34:17.926 11196 11196 I surfaceflinger: 0xa7c58620: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 12:34:17.926 11196 11196 I surfaceflinger: 0xa7c58630: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 12:34:17.926 11196 11196 I surfaceflinger: =>0xa7c58640: f1 f1[f8]f3 00 00 00 00 00 00 00 00 00 00 00 00  07-17 12:34:17.926 11196 11196 I surfaceflinger: 0xa7c58650: f1 f1 f8 f2 f8 f8 f2 f2 f8 f8 f2 f2 f8 f8 f2 f2  07-17 12:34:17.926 11196 11196 I surfaceflinger: 0xa7c58660: f8 f2 04 f2 04 f2 f8 f2 f8 f2 f8 f3 00 00 00 00  07-17 12:34:17.926 11196 11196 I surfaceflinger: 0xa7c58670: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 12:34:17.927 11196 11196 I surfaceflinger: 0xa7c58680: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 12:34:17.927 11196 11196 I surfaceflinger: 0xa7c58690: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 12:34:17.927 11196 11196 I surfaceflinger: Shadow byte legend (one shadow byte represents 8 application bytes):  07-17 12:34:17.927 11196 11196 I surfaceflinger: Addressable: 00  07-17 12:34:17.927 11196 11196 I surfaceflinger: Partially addressable: 01 02 03 04 05 06 07  07-17 12:34:17.927 11196 11196 I surfaceflinger: Heap left redzone: fa  07-17 12:34:17.927 11196 11196 I surfaceflinger: Freed heap region: fd  07-17 12:34:17.927 11196 11196 I surfaceflinger: Stack left redzone: f1  07-17 12:34:17.927 11196 11196 I surfaceflinger: Stack mid redzone: f2  07-17 12:34:17.927 11196 11196 I surfaceflinger: Stack right redzone: f3  07-17 12:34:17.927 11196 11196 I surfaceflinger: Stack after return: f5  07-17 12:34:17.927 11196 11196 I surfaceflinger: Stack use after scope: f8  07-17 12:34:17.927 11196 11196 I surfaceflinger: Global redzone: f9  07-17 12:34:17.927 11196 11196 I surfaceflinger: Global init order: f6  07-17 12:34:17.927 11196 11196 I surfaceflinger: Poisoned by user: f7  07-17 12:34:17.927 11196 11196 I surfaceflinger: Container overflow: fc  07-17 12:34:17.927 11196 11196 I surfaceflinger: Array cookie: ac  07-17 12:34:17.927 11196 11196 I surfaceflinger: Intra object redzone: bb  07-17 12:34:17.927 11196 11196 I surfaceflinger: ASan internal: fe  07-17 12:34:17.927 11196 11196 I surfaceflinger: Left alloca redzone: ca  07-17 12:34:17.927 11196 11196 I surfaceflinger: Right alloca redzone: cb  07-17 12:34:17.927 11196 11196 I surfaceflinger: Shadow gap: cc  07-17 12:34:17.928 11196 11196 I surfaceflinger: ==11196==ABORTING |

### 栈解析如下

调用栈解析结果如下：（调用栈解析方法见后面章节）

|  |
| --- |
| ==11196==ERROR: AddressSanitizer: stack-use-after-scope on address 0xbef83210 at pc 0xb160aae3 bp 0xbef831f8 sp 0xbef831f4  WRITE of size 4 at 0xbef83210 thread T0  #0 0xb160aae0 in \_ZN7android14surfaceflinger9test\_testEv frameworks/native/services/surfaceflinger/SurfaceFlingerFactory.cpp:32  #1 0xb160aae0 in \_ZN7android14surfaceflinger20createSurfaceFlingerEv frameworks/native/services/surfaceflinger/SurfaceFlingerFactory.cpp:39  #2 0xec0d71a in main frameworks/native/services/surfaceflinger/main\_surfaceflinger.cpp:97  #3 0xb19ece88 in \_\_libc\_init bionic/libc/bionic/libc\_init\_dynamic.cpp:152  #4 0xec0d434 in \_start\_main bionic/libc/arch-common/bionic/crtbegin.c:45  Address 0xbef83210 is located in stack of thread T0  at offset 16 in frame  #5 0xb160a9a2 in \_ZN7android14surfaceflinger20createSurfaceFlingerEv frameworks/native/services/surfaceflinger/SurfaceFlingerFactory.cpp:36  This frame has 1 object(s):  [16, 20) 'x.i' (line 29) <== Memory access at offset 16 is inside this variable  HINT: this may be a false positive if your program uses some custom stack unwind mechanism, swapcontext or vfork  (longjmp and C++ exceptions \*are\* supported)  SUMMARY: AddressSanitizer: stack-use-after-scope (/system/lib/libsurfaceflinger.so+0x27fae0)  Shadow bytes around the buggy address:  0xa7c585f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0xa7c58600: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0xa7c58610: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0xa7c58620: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0xa7c58630: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  =>0xa7c58640: f1 f1[f8]f3 00 00 00 00 00 00 00 00 00 00 00 00  0xa7c58650: f1 f1 f8 f2 f8 f8 f2 f2 f8 f8 f2 f2 f8 f8 f2 f2  0xa7c58660: f8 f2 04 f2 04 f2 f8 f2 f8 f2 f8 f3 00 00 00 00  0xa7c58670: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0xa7c58680: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0xa7c58690: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  Shadow byte legend (one shadow byte represents 8 application bytes):  Addressable: 00  Partially addressable: 01 02 03 04 05 06 07  Heap left redzone: fa  Freed heap region: fd  Stack left redzone: f1  Stack mid redzone: f2  Stack right redzone: f3  Stack after return: f5  Stack use after scope: f8  Global redzone: f9  Global init order: f6  Poisoned by user: f7  Container overflow: fc  Array cookie: ac  Intra object redzone: bb  ASan internal: fe  Left alloca redzone: ca  Right alloca redzone: cb  Shadow gap: cc  ==11196==ABORTING |

### 对应代码

|  |
| --- |
| 22 #include "SurfaceFlingerDefaultFactory.h"  23  24 namespace android::surfaceflinger {  25  26 volatile int \*p = 0;  27 int test\_test() {  28 {  29 int x = 0;  30 p = &x;  31 }  32 \*p = 5;  33 return 0;  34 }  35  36 sp<SurfaceFlinger> createSurfaceFlinger() {  37 static DefaultFactory factory;  38  39 test\_test();  40 return new SurfaceFlinger(factory);  41 }  42  43 } // namespace android::surfaceflinger |

## 一般动态库调试举例

我们以调试libSurfaceFlingerProp.so库为例



正常情况下我们需要使用上图中左面的方案，将所有使用asan编译的相关库放在/data/asan目录下

但我们知道，实际上libsurfaceflinger.so 只有surfaceflinger调用，没有其他调用，因此只需将libSurfaceFlingerProp.so放到asan下即可，所以我们使用下面的方案



### 配置文件的修改

调试libSurfaceFlingerProp.so库，需要修改调试libSurfaceFlingerProp.so库的编译配置及调用它的surfaceflinger的编译配置。

<http://review.source.unisoc.com/gerrit/#/c/694140/>

|  |
| --- |
| sprdroidr\_trunk/frameworks/native/services/surfaceflinger$ git diff .  diff --git a/services/surfaceflinger/Android.bp b/services/surfaceflinger/Android.bp  index a790d0b..4435b31 100644  --- a/services/surfaceflinger/Android.bp  +++ b/services/surfaceflinger/Android.bp  @@ -190,6 +190,7 @@ cc\_library\_shared {  // Please use libsurfaceflinger\_defaults to configure how the sources are  // built, so the same settings can be used elsewhere.  name: "libsurfaceflinger",  + sanitize:{ address:true },  defaults: ["libsurfaceflinger\_production\_defaults"],  srcs: [  ":libsurfaceflinger\_sources",  @@ -249,6 +250,7 @@ filegroup {    cc\_binary {  name: "surfaceflinger",  + sanitize:{ address:true },  defaults: ["libsurfaceflinger\_binary"],  init\_rc: ["surfaceflinger.rc"],  srcs: [":surfaceflinger\_binary\_sources"],  @@ -265,6 +267,7 @@ subdirs = [    cc\_library\_shared {  name: "libSurfaceFlingerProp",  + sanitize:{ address:true },  srcs: [  "SurfaceFlingerProperties.cpp",  ], |

### 模拟一个问题

为验证ASan是否生效，我们修改surfaceflinger代码，制造一个问题，看ASan 是否可以检测到，我们制造一个范围外使用的问题：

<http://review.source.unisoc.com/gerrit/#/c/694140/>

|  |
| --- |
| diff --git a/services/surfaceflinger/SurfaceFlingerProperties.cpp b/services/surfaceflinger/SurfaceFlingerProperties.cpp  index 9d78702..b0fef3f 100644  --- a/services/surfaceflinger/SurfaceFlingerProperties.cpp  +++ b/services/surfaceflinger/SurfaceFlingerProperties.cpp  @@ -154,10 +154,20 @@ bool start\_graphics\_allocator\_service(bool defaultValue) {  return getBool<ISurfaceFlingerConfigs, &ISurfaceFlingerConfigs::startGraphicsAllocatorService>(  defaultValue);  }  +volatile int \*p = 0;  +int test\_test() {  + {  + int x = 0;  + p = &x;  + }  + \*p = 5;  + return 0;  +}  SurfaceFlingerProperties::primary\_display\_orientation\_values primary\_display\_orientation(  SurfaceFlingerProperties::primary\_display\_orientation\_values defaultValue) {  auto temp = SurfaceFlingerProperties::primary\_display\_orientation();  + test\_test();  if (temp.has\_value()) {  return \*temp;  } |

### 编译替换生成库

系统编译完成后自动生成两套库和符号表

这里AndroidR与AndroidQ不同，之前AndroidQ版本会将自动将asan编译的文件放到/data/asan下，而AndroidR是下面这样：

|  |
| --- |
| sprdroidr\_trunk/out$ find -name libSurfaceFlingerProp.so  ./soong/.intermediates/frameworks/native/services/surfaceflinger/libSurfaceFlingerProp/android\_arm\_armv7-a-neon\_shared/libSurfaceFlingerProp.so  ./soong/.intermediates/frameworks/native/services/surfaceflinger/libSurfaceFlingerProp/android\_arm\_armv7-a-neon\_shared/unstripped/libSurfaceFlingerProp.so  ./soong/.intermediates/frameworks/native/services/surfaceflinger/libSurfaceFlingerProp/android\_arm\_armv7-a-neon\_shared\_asan/libSurfaceFlingerProp.so  ./soong/.intermediates/frameworks/native/services/surfaceflinger/libSurfaceFlingerProp/android\_arm\_armv7-a-neon\_shared\_asan/unstripped/libSurfaceFlingerProp.so  ./target/product/s9863a1h10\_go\_32b/obj/SHARED\_LIBRARIES/libSurfaceFlingerProp\_intermediates/libSurfaceFlingerProp.so  ./target/product/s9863a1h10\_go\_32b/system/lib/libSurfaceFlingerProp.so  ./target/product/s9863a1h10\_go\_32b/symbols/system/lib/libSurfaceFlingerProp.so |

上面绿色的库是不含asan的编译，黄色的是含asan的编译，蓝色的与黄色的相同

我们只需要将蓝色的库放到手机里

|  |
| --- |
| sprdroidr\_trunk/out/target/product/s9863a1h10\_go\_32b$ find -name libSurfaceFlingerProp.so  ./obj/SHARED\_LIBRARIES/libSurfaceFlingerProp\_intermediates/libSurfaceFlingerProp.so  ./system/lib/libSurfaceFlingerProp.so  ./symbols/system/lib/libSurfaceFlingerProp.so |

然后，将生成的的out目录中的/data/asan/system中的动态库push到手机上的/data/asan/system中。

### 修改服务启动文件

Surfaceflinger是系统服务，因此需要修改surfaceflinger.rc

修改工程目录 out目录中 /system/etc/init/surfaceflinger.rc 文件

|  |
| --- |
| service surfaceflinger /system/bin/surfaceflinger  class core animation  user system  group graphics drmrpc readproc  capabilities SYS\_NICE  onrestart restart zygote  setenv LD\_LIBRARY\_PATH /data/asan/system/lib/  task\_profiles HighPerformance  socket pdx/system/vr/display/client stream 0666 system graphics u:object\_r:pdx\_display\_client\_endpoint\_socket:s0  socket pdx/system/vr/display/manager stream 0666 system graphics u:object\_r:pdx\_display\_manager\_endpoint\_socket:s0  socket pdx/system/vr/display/vsync stream 0666 system graphics u:object\_r:pdx\_display\_vsync\_endpoint\_socket:s0 |

将该文件替换到手机上/ system/etc/init/surfaceflinger.rc文件

### 修改selinux权限

停用 SELinux，如下所示：

|  |
| --- |
| $sudo adb root  $adb shell setenforce 0 |

关闭SELinux后，将被测进程surfaceflinger杀了重启。

### 3.3 检测出问题Log如下

我们看到 调用栈显示 /data/asan 下的库，说明调用到的是我们的库

|  |
| --- |
| 07-17 19:12:43.051 30144 30144 I surfaceflinger: ==30144==ERROR: AddressSanitizer: stack-use-after-scope on address 0xbeac8020 at pc 0xb291df81 bp 0xbeac7ff8 sp 0xbeac7ff4  07-17 19:12:43.051 30144 30144 I surfaceflinger: WRITE of size 4 at 0xbeac8020 thread T0  07-17 19:12:43.071 30144 30144 I surfaceflinger: #0 0xb291df7e (/data/asan/system/lib/libSurfaceFlingerProp.so+0xff7e)  07-17 19:12:43.071 30144 30144 I surfaceflinger: #1 0xb18e98ea (/data/asan/system/lib/libsurfaceflinger.so+0x1cd8ea)  07-17 19:12:43.071 30144 30144 I surfaceflinger: #2 0xb199b9a8 (/data/asan/system/lib/libsurfaceflinger.so+0x27f9a8)  07-17 19:12:43.072 30144 30144 I surfaceflinger: #3 0x971d71a (/system/bin/surfaceflinger+0x271a)  07-17 19:12:43.072 30144 30144 I surfaceflinger: #4 0xb1ccce88 (/apex/com.android.runtime/lib/bionic/libc.so+0x5ce88)  07-17 19:12:43.072 30144 30144 I surfaceflinger: #5 0x971d434 (/system/bin/surfaceflinger+0x2434)  07-17 19:12:43.073 30144 30144 I surfaceflinger:  07-17 19:12:43.073 30144 30144 I surfaceflinger: Address 0xbeac8020 is located in stack of thread T0  07-17 19:12:43.074 30144 30144 I surfaceflinger: at offset 32 in frame  07-17 19:12:43.074 30144 30144 I surfaceflinger: #0 0xb291ddee (/data/asan/system/lib/libSurfaceFlingerProp.so+0xfdee)  07-17 19:12:43.075 30144 30144 I surfaceflinger:  07-17 19:12:43.075 30144 30144 I surfaceflinger: This frame has 3 object(s):  07-17 19:12:43.075 30144 30144 I surfaceflinger: [16, 17) 'defValue.addr.i'  07-17 19:12:43.076 30144 30144 I surfaceflinger: [32, 36) 'x.i' (line 160) <== Memory access at offset 32 is inside this variable  07-17 19:12:43.076 30144 30144 I surfaceflinger: [48, 56) 'temp' (line 169)  07-17 19:12:43.076 30144 30144 I surfaceflinger: HINT: this may be a false positive if your program uses some custom stack unwind mechanism, swapcontext or vfork  07-17 19:12:43.077 30144 30144 I surfaceflinger: (longjmp and C++ exceptions \*are\* supported)  07-17 19:12:43.077 30144 30144 I surfaceflinger: SUMMARY: AddressSanitizer: stack-use-after-scope (/data/asan/system/lib/libSurfaceFlingerProp.so+0xff7e)  07-17 19:12:43.078 30144 30144 I surfaceflinger: Shadow bytes around the buggy address:  07-17 19:12:43.078 30144 30144 I surfaceflinger: 0xa8e78fb0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 19:12:43.078 30144 30144 I surfaceflinger: 0xa8e78fc0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 19:12:43.078 30144 30144 I surfaceflinger: 0xa8e78fd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 19:12:43.078 30144 30144 I surfaceflinger: 0xa8e78fe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 19:12:43.078 30144 30144 I surfaceflinger: 0xa8e78ff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 19:12:43.078 30144 30144 I surfaceflinger: =>0xa8e79000: f1 f1 f8 f2[f8]f2 00 f3 f3 f3 00 00 00 00 00 00  07-17 19:12:43.078 30144 30144 I surfaceflinger: 0xa8e79010: 00 00 00 00 00 00 00 00 00 00 00 00 f1 f1 f8 f8  07-17 19:12:43.078 30144 30144 I surfaceflinger: 0xa8e79020: f8 f8 f8 f8 f2 f2 f2 f2 f8 f8 f8 f8 f8 f8 f8 f8  07-17 19:12:43.078 30144 30144 I surfaceflinger: 0xa8e79030: f8 f8 f8 f8 f2 f2 f2 f2 f8 f8 f2 f2 f8 f8 f3 f3  07-17 19:12:43.078 30144 30144 I surfaceflinger: 0xa8e79040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 19:12:43.078 30144 30144 I surfaceflinger: 0xa8e79050: f1 f1 f8 f2 f8 f8 f2 f2 f8 f8 f2 f2 f8 f8 f2 f2  07-17 19:12:43.078 30144 30144 I surfaceflinger: Shadow byte legend (one shadow byte represents 8 application bytes):  07-17 19:12:43.078 30144 30144 I surfaceflinger: Addressable: 00  07-17 19:12:43.078 30144 30144 I surfaceflinger: Partially addressable: 01 02 03 04 05 06 07  07-17 19:12:43.078 30144 30144 I surfaceflinger: Heap left redzone: fa  07-17 19:12:43.078 30144 30144 I surfaceflinger: Freed heap region: fd  07-17 19:12:43.079 30144 30144 I surfaceflinger: Stack left redzone: f1  07-17 19:12:43.079 30144 30144 I surfaceflinger: Stack mid redzone: f2  07-17 19:12:43.079 30144 30144 I surfaceflinger: Stack right redzone: f3  07-17 19:12:43.079 30144 30144 I surfaceflinger: Stack after return: f5  07-17 19:12:43.079 30144 30144 I surfaceflinger: Stack use after scope: f8  07-17 19:12:43.079 30144 30144 I surfaceflinger: Global redzone: f9  07-17 19:12:43.079 30144 30144 I surfaceflinger: Global init order: f6  07-17 19:12:43.079 30144 30144 I surfaceflinger: Poisoned by user: f7  07-17 19:12:43.079 30144 30144 I surfaceflinger: Container overflow: fc  07-17 19:12:43.079 30144 30144 I surfaceflinger: Array cookie: ac  07-17 19:12:43.079 30144 30144 I surfaceflinger: Intra object redzone: bb  07-17 19:12:43.079 30144 30144 I surfaceflinger: ASan internal: fe  07-17 19:12:43.079 30144 30144 I surfaceflinger: Left alloca redzone: ca  07-17 19:12:43.079 30144 30144 I surfaceflinger: Right alloca redzone: cb  07-17 19:12:43.079 30144 30144 I surfaceflinger: Shadow gap: cc  07-17 19:12:43.080 30144 30144 I surfaceflinger: ==30144==ABORTING |

由于out下asan编译库的位置就在/system/lib下，因此需要将其手工删除/data/asan，改为out下asan库的路径，如下：

|  |
| --- |
| 07-17 19:12:43.051 30144 30144 I surfaceflinger: ==30144==ERROR: AddressSanitizer: stack-use-after-scope on address 0xbeac8020 at pc 0xb291df81 bp 0xbeac7ff8 sp 0xbeac7ff4  07-17 19:12:43.051 30144 30144 I surfaceflinger: WRITE of size 4 at 0xbeac8020 thread T0  07-17 19:12:43.071 30144 30144 I surfaceflinger: #0 0xb291df7e (/system/lib/libSurfaceFlingerProp.so+0xff7e)  07-17 19:12:43.071 30144 30144 I surfaceflinger: #1 0xb18e98ea (/system/lib/libsurfaceflinger.so+0x1cd8ea)  07-17 19:12:43.071 30144 30144 I surfaceflinger: #2 0xb199b9a8 (/system/lib/libsurfaceflinger.so+0x27f9a8)  07-17 19:12:43.072 30144 30144 I surfaceflinger: #3 0x971d71a (/system/bin/surfaceflinger+0x271a)  07-17 19:12:43.072 30144 30144 I surfaceflinger: #4 0xb1ccce88 (/apex/com.android.runtime/lib/bionic/libc.so+0x5ce88)  07-17 19:12:43.072 30144 30144 I surfaceflinger: #5 0x971d434 (/system/bin/surfaceflinger+0x2434)  07-17 19:12:43.073 30144 30144 I surfaceflinger:  07-17 19:12:43.073 30144 30144 I surfaceflinger: Address 0xbeac8020 is located in stack of thread T0  07-17 19:12:43.074 30144 30144 I surfaceflinger: at offset 32 in frame  07-17 19:12:43.074 30144 30144 I surfaceflinger: #0 0xb291ddee (/system/lib/libSurfaceFlingerProp.so+0xfdee)  07-17 19:12:43.075 30144 30144 I surfaceflinger:  07-17 19:12:43.075 30144 30144 I surfaceflinger: This frame has 3 object(s):  07-17 19:12:43.075 30144 30144 I surfaceflinger: [16, 17) 'defValue.addr.i'  07-17 19:12:43.076 30144 30144 I surfaceflinger: [32, 36) 'x.i' (line 160) <== Memory access at offset 32 is inside this variable  07-17 19:12:43.076 30144 30144 I surfaceflinger: [48, 56) 'temp' (line 169)  07-17 19:12:43.076 30144 30144 I surfaceflinger: HINT: this may be a false positive if your program uses some custom stack unwind mechanism, swapcontext or vfork  07-17 19:12:43.077 30144 30144 I surfaceflinger: (longjmp and C++ exceptions \*are\* supported)  07-17 19:12:43.077 30144 30144 I surfaceflinger: SUMMARY: AddressSanitizer: stack-use-after-scope (/system/lib/libSurfaceFlingerProp.so+0xff7e)  07-17 19:12:43.078 30144 30144 I surfaceflinger: Shadow bytes around the buggy address:  07-17 19:12:43.078 30144 30144 I surfaceflinger: 0xa8e78fb0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 19:12:43.078 30144 30144 I surfaceflinger: 0xa8e78fc0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 19:12:43.078 30144 30144 I surfaceflinger: 0xa8e78fd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 19:12:43.078 30144 30144 I surfaceflinger: 0xa8e78fe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 19:12:43.078 30144 30144 I surfaceflinger: 0xa8e78ff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 19:12:43.078 30144 30144 I surfaceflinger: =>0xa8e79000: f1 f1 f8 f2[f8]f2 00 f3 f3 f3 00 00 00 00 00 00  07-17 19:12:43.078 30144 30144 I surfaceflinger: 0xa8e79010: 00 00 00 00 00 00 00 00 00 00 00 00 f1 f1 f8 f8  07-17 19:12:43.078 30144 30144 I surfaceflinger: 0xa8e79020: f8 f8 f8 f8 f2 f2 f2 f2 f8 f8 f8 f8 f8 f8 f8 f8  07-17 19:12:43.078 30144 30144 I surfaceflinger: 0xa8e79030: f8 f8 f8 f8 f2 f2 f2 f2 f8 f8 f2 f2 f8 f8 f3 f3  07-17 19:12:43.078 30144 30144 I surfaceflinger: 0xa8e79040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07-17 19:12:43.078 30144 30144 I surfaceflinger: 0xa8e79050: f1 f1 f8 f2 f8 f8 f2 f2 f8 f8 f2 f2 f8 f8 f2 f2  07-17 19:12:43.078 30144 30144 I surfaceflinger: Shadow byte legend (one shadow byte represents 8 application bytes):  07-17 19:12:43.078 30144 30144 I surfaceflinger: Addressable: 00  07-17 19:12:43.078 30144 30144 I surfaceflinger: Partially addressable: 01 02 03 04 05 06 07  07-17 19:12:43.078 30144 30144 I surfaceflinger: Heap left redzone: fa  07-17 19:12:43.078 30144 30144 I surfaceflinger: Freed heap region: fd  07-17 19:12:43.079 30144 30144 I surfaceflinger: Stack left redzone: f1  07-17 19:12:43.079 30144 30144 I surfaceflinger: Stack mid redzone: f2  07-17 19:12:43.079 30144 30144 I surfaceflinger: Stack right redzone: f3  07-17 19:12:43.079 30144 30144 I surfaceflinger: Stack after return: f5  07-17 19:12:43.079 30144 30144 I surfaceflinger: Stack use after scope: f8  07-17 19:12:43.079 30144 30144 I surfaceflinger: Global redzone: f9  07-17 19:12:43.079 30144 30144 I surfaceflinger: Global init order: f6  07-17 19:12:43.079 30144 30144 I surfaceflinger: Poisoned by user: f7  07-17 19:12:43.079 30144 30144 I surfaceflinger: Container overflow: fc  07-17 19:12:43.079 30144 30144 I surfaceflinger: Array cookie: ac  07-17 19:12:43.079 30144 30144 I surfaceflinger: Intra object redzone: bb  07-17 19:12:43.079 30144 30144 I surfaceflinger: ASan internal: fe  07-17 19:12:43.079 30144 30144 I surfaceflinger: Left alloca redzone: ca  07-17 19:12:43.079 30144 30144 I surfaceflinger: Right alloca redzone: cb  07-17 19:12:43.079 30144 30144 I surfaceflinger: Shadow gap: cc  07-17 19:12:43.080 30144 30144 I surfaceflinger: ==30144==ABORTING |

### 栈解析如下

调用栈解析结果如下：（调用栈解析方法见后面章节）

|  |
| --- |
| ==30144==ERROR: AddressSanitizer: stack-use-after-scope on address 0xbeac8020 at pc 0xb291df81 bp 0xbeac7ff8 sp 0xbeac7ff4  WRITE of size 4 at 0xbeac8020 thread T0  #0 0xb291df7e in \_ZN7android7sysprop9test\_testEv frameworks/native/services/surfaceflinger/SurfaceFlingerProperties.cpp:163  #1 0xb291df7e in \_ZN7android7sysprop27primary\_display\_orientationENS0\_24SurfaceFlingerProperties34primary\_display\_orientation\_valuesE frameworks/native/services/surfaceflinger/SurfaceFlingerProperties.cpp:170  #2 0xb18e98ea in SurfaceFlinger frameworks/native/services/surfaceflinger/SurfaceFlinger.cpp:364  #3 0xb199b9a8 in \_ZN7android14surfaceflinger20createSurfaceFlingerEv frameworks/native/services/surfaceflinger/SurfaceFlingerFactory.cpp:29  #4 0x971d71a in main frameworks/native/services/surfaceflinger/main\_surfaceflinger.cpp:97  #5 0xb1ccce88 in \_\_libc\_init bionic/libc/bionic/libc\_init\_dynamic.cpp:152  #6 0x971d434 in \_start\_main bionic/libc/arch-common/bionic/crtbegin.c:45  Address 0xbeac8020 is located in stack of thread T0  at offset 32 in frame  #7 0xb291ddee in \_ZN7android7sysprop27primary\_display\_orientationENS0\_24SurfaceFlingerProperties34primary\_display\_orientation\_valuesE frameworks/native/services/surfaceflinger/SurfaceFlingerProperties.cpp:168  This frame has 3 object(s):  [16, 17) 'defValue.addr.i'  [32, 36) 'x.i' (line 160) <== Memory access at offset 32 is inside this variable  [48, 56) 'temp' (line 169)  HINT: this may be a false positive if your program uses some custom stack unwind mechanism, swapcontext or vfork  (longjmp and C++ exceptions \*are\* supported)  SUMMARY: AddressSanitizer: stack-use-after-scope (/system/lib/libSurfaceFlingerProp.so+0xff7e)  Shadow bytes around the buggy address:  0xa8e78fb0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0xa8e78fc0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0xa8e78fd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0xa8e78fe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0xa8e78ff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  =>0xa8e79000: f1 f1 f8 f2[f8]f2 00 f3 f3 f3 00 00 00 00 00 00  0xa8e79010: 00 00 00 00 00 00 00 00 00 00 00 00 f1 f1 f8 f8  0xa8e79020: f8 f8 f8 f8 f2 f2 f2 f2 f8 f8 f8 f8 f8 f8 f8 f8  0xa8e79030: f8 f8 f8 f8 f2 f2 f2 f2 f8 f8 f2 f2 f8 f8 f3 f3  0xa8e79040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0xa8e79050: f1 f1 f8 f2 f8 f8 f2 f2 f8 f8 f2 f2 f8 f8 f2 f2  Shadow byte legend (one shadow byte represents 8 application bytes):  Addressable: 00  Partially addressable: 01 02 03 04 05 06 07  Heap left redzone: fa  Freed heap region: fd  Stack left redzone: f1  Stack mid redzone: f2  Stack right redzone: f3  Stack after return: f5  Stack use after scope: f8  Global redzone: f9  Global init order: f6  Poisoned by user: f7  Container overflow: fc  Array cookie: ac  Intra object redzone: bb  ASan internal: fe  Left alloca redzone: ca  Right alloca redzone: cb  Shadow gap: cc  ==30144==ABORTING |

### 对应代码

|  |
| --- |
| 156 }  157 volatile int \*p = 0;  158 int test\_test() {  159 {  160 int x = 0;  161 p = &x;  162 }  163 \*p = 5;  164 return 0;  165 }  166  167 SurfaceFlingerProperties::primary\_display\_orientation\_values primary\_display\_orientation(  168 SurfaceFlingerProperties::primary\_display\_orientation\_values defaultValue) {  169 auto temp = SurfaceFlingerProperties::primary\_display\_orientation();  170 test\_test();  171 if (temp.has\_value()) {  172 return \*temp;  173 } |

# 在应用程序中使用ASan

AddressSanitizer无法检查Java代码，它只能检测JNI库中的错误。

## 打开Asan开关

因为apk都是通过app\_process来执行的，因此，我们可以用ASan构建app\_process可执行文件， /system/bin/app\_process（32 | 64）。

与之前版本 使用amdroid.mk不同，AndroidR 使用 Android.bp

将sanitize: {address: true} 添加到frameworks/base/cmds/app\_process中的amdroid.mk文件。

|  |
| --- |
| sprdroidr\_trunk/frameworks/base/cmds/app\_process$ git diff .  diff --git a/cmds/app\_process/Android.bp b/cmds/app\_process/Android.bp  index 8be95e4..3eda87d 100644  --- a/cmds/app\_process/Android.bp  +++ b/cmds/app\_process/Android.bp  @@ -1,6 +1,6 @@  cc\_binary {  name: "app\_process",  -  + sanitize: { address: true },  srcs: ["app\_main.cpp"],    multilib: { |

并将编译出的app\_process32及app\_process64替换到手机中即可。

## 修改启动配置

在 system/core/rootdir/init.zygote(32|64).rc 中修改 Zygote 记录，以添加以下行：

|  |
| --- |
| setenv LD\_LIBRARY\_PATH /system/lib/asan:/system/lib  setenv ASAN\_OPTIONS allow\_user\_segv\_handler=true:detect\_container\_overflow=0 |

其中/system/lib/asan 是 使用asan编译的库的位置，根据实际情况填写，如果你的库放在 /data/asan/system/lib下，这里就填/data/asan/system/lib

修改sprdroidr\_trunk/system/core/rootdir下相关的init.zygote.rc 文件

|  |
| --- |
| sprdroidr\_trunk/system/core/rootdir$  init.zygote32\_64.rc  init.zygote32.rc  init.zygote64\_32.rc  init.zygote64.rc |

添加相关选项：

|  |
| --- |
| diff --git a/rootdir/init.zygote32.rc b/rootdir/init.zygote32.rc  index bf3fb42..3dbd6dd 100644 (file)  --- a/rootdir/init.zygote32.rc  +++ b/rootdir/init.zygote32.rc  @@ -3,6 +3,9 @@ service zygote /system/bin/app\_process -Xzygote /system/bin --zygote --start-sys  priority -20  user root  group root readproc reserved\_disk  + setenv LD\_LIBRARY\_PATH /system/lib/asan:/system/lib  + setenv ASAN\_OPTIONS allow\_user\_segv\_handler=true  socket zygote stream 660 root system  socket usap\_pool\_primary stream 660 root system  onrestart write /sys/android\_power/request\_state wake |

**system/etc/init/hw/init.zygote32.rc**

## 关于wrap.属性

上一部分中的方法将 AddressSanitizer 放置到了系统的每个应用中（实际上是放置到了 Zygote 进程的每个子项中）。

您可以只通过 ASan 运行一个或少数几个应用，从而节省一些内存空间，但是应用启动速度会变慢。

我们可以通过“wrap.”属性（与在 Valgrind 下运行应用时所用的属性相同）启动特定应用。

下面是在 ASan 下运行 Gmail 应用的示例：

|  |
| --- |
| adb root  adb shell setenforce 0 # disable SELinux  adb shell setprop wrap.com.google.android.gm "asanwrapper" |

在此情况下， asanwrapper 会将 /system/bin/app\_process 重写至（使用 AddressSanitizer 编译的）/system/bin/asan/app\_process。此外，它还会在动态库搜索路径的开头处添加 /system/lib/asan。

这样一来，通过 asanwrapper 运行应用时，系统会优先使用 /system/lib/asan 中可通过 ASan 进行测试的库，而非 /system/lib 中的普通库。

同样，如果发现错误，应用会崩溃，且系统会将报告记录到日志中。

实际使用中2G内存，所有应用，运行不卡顿，无需使用wrap.属性。

如使用wrap.属性，需注意，需要手工将含有asan的库复制到/system/bin/asan/目录下。

## 其他

所有应用修改都是针对app\_process，按照相关步骤操作即可，这个不再模拟内存问题举例说明。

# 调用栈解析

## 解析脚本

Google原生代码中提供了调用栈解析的脚本symbolize.py，脚本位于external/compiler-rt/lib/asan/scripts/目录中。

## 脚本使用

使用方法如下：

1.首先在工程目录启环境

|  |
| --- |
| source build/envsetup.sh  lunch 指定工程 |

2.运行脚本

|  |
| --- |
| python external/compiler-rt/lib/asan/scripts/symbolize.py |

脚本运行后窗口处于等待输入的状态，此时，我们可以将调用栈贴入窗口，然后 Ctrl+D 表示输出完成，之后会显示解析的结果。

## 脚本使用举例

问题log

|  |
| --- |
| 08-02 08:11:20.706 986 986 I surfaceflinger: =================================================================  08-02 08:11:20.707 986 986 I surfaceflinger: ==986==ERROR: AddressSanitizer: stack-use-after-scope on address 0x007fe896bde0 at pc 0x005f05bdfbc0 bp 0x007fe896bdb0 sp 0x007fe896bda8  08-02 08:11:20.708 986 986 I surfaceflinger: WRITE of size 4 at 0x007fe896bde0 thread T0  08-02 08:11:20.737 986 986 I surfaceflinger: #0 0x5f05bdfbbc (/system/bin/surfaceflinger+0x4bbc)  08-02 08:11:20.738 986 986 I surfaceflinger: #1 0x733ed4b798 (/apex/com.android.runtime/lib64/bionic/libc.so+0x7d798)  08-02 08:11:20.738 986 986 I surfaceflinger:  08-02 08:11:20.739 986 986 I surfaceflinger: Address 0x007fe896bde0 is located in stack of thread T0  08-02 08:11:20.739 986 986 I surfaceflinger: at offset 32 in frame  08-02 08:11:20.740 986 986 I surfaceflinger: #0 0x5f05bdf1a8 (/system/bin/surfaceflinger+0x41a8)  08-02 08:11:20.741 986 986 I surfaceflinger:  08-02 08:11:20.741 986 986 I surfaceflinger: This frame has 12 object(s):  08-02 08:11:20.742 986 986 I surfaceflinger: [32, 36) 'x.i' (line 78) <== Memory access at offset 32 is inside this variable  08-02 08:11:20.742 986 986 I surfaceflinger: [48, 56) 'displayservice.i' (line 65)  08-02 08:11:20.743 986 986 I surfaceflinger: [80, 104) 'ref.tmp.i43' (line 66)  08-02 08:11:20.743 986 986 I surfaceflinger: [144, 168) 'ref.tmp.i' (line 45)  08-02 08:11:20.744 986 986 I surfaceflinger: [208, 232) 'ref.tmp4.i' (line 51)  08-02 08:11:20.744 986 986 I surfaceflinger: [272, 280) 'ref.tmp' (line 94)  08-02 08:11:20.745 986 986 I surfaceflinger: [304, 312) 'ps' (line 97)  08-02 08:11:20.745 986 986 I surfaceflinger: [336, 344) 'flinger' (line 101)  08-02 08:11:20.746 986 986 I surfaceflinger: [368, 376) 'sm' (line 116)  08-02 08:11:20.746 986 986 I surfaceflinger: [400, 408) 'ref.tmp12' (line 117)  08-02 08:11:20.747 986 986 I surfaceflinger: [432, 440) 'ref.tmp14' (line 117)  08-02 08:11:20.747 986 986 I surfaceflinger: [464, 468) 'param' (line 122)  08-02 08:11:20.747 986 986 I surfaceflinger: HINT: this may be a false positive if your program uses some custom stack unwind mechanism, swapcontext or vfork  08-02 08:11:20.748 986 986 I surfaceflinger: (longjmp and C++ exceptions \*are\* supported)  08-02 08:11:20.748 986 986 I surfaceflinger: SUMMARY: AddressSanitizer: stack-use-after-scope (/system/bin/surfaceflinger+0x4bbc)  08-02 08:11:20.749 986 986 I surfaceflinger: Shadow bytes around the buggy address:  08-02 08:11:20.750 986 986 I surfaceflinger: 0x001ffd12d760: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  08-02 08:11:20.750 986 986 I surfaceflinger: 0x001ffd12d770: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  08-02 08:11:20.750 986 986 I surfaceflinger: 0x001ffd12d780: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  08-02 08:11:20.750 986 986 I surfaceflinger: 0x001ffd12d790: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  08-02 08:11:20.750 986 986 I surfaceflinger: 0x001ffd12d7a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  08-02 08:11:20.750 986 986 I surfaceflinger: =>0x001ffd12d7b0: 00 00 00 00 00 00 00 00 f1 f1 f1 f1[f8]f2 f8 f2  08-02 08:11:20.750 986 986 I surfaceflinger: 0x001ffd12d7c0: f2 f2 f8 f8 f8 f2 f2 f2 f2 f2 f8 f8 f8 f2 f2 f2  08-02 08:11:20.751 986 986 I surfaceflinger: 0x001ffd12d7d0: f2 f2 f8 f8 f8 f2 f2 f2 f2 f2 f8 f2 f2 f2 00 f2  08-02 08:11:20.751 986 986 I surfaceflinger: 0x001ffd12d7e0: f2 f2 00 f2 f2 f2 00 f2 f2 f2 f8 f2 f2 f2 f8 f2  08-02 08:11:20.751 986 986 I surfaceflinger: 0x001ffd12d7f0: f2 f2 04 f3 00 00 00 00 00 00 00 00 00 00 00 00  08-02 08:11:20.751 986 986 I surfaceflinger: 0x001ffd12d800: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  08-02 08:11:20.751 986 986 I surfaceflinger: Shadow byte legend (one shadow byte represents 8 application bytes):  08-02 08:11:20.751 986 986 I surfaceflinger: Addressable: 00  08-02 08:11:20.751 986 986 I surfaceflinger: Partially addressable: 01 02 03 04 05 06 07  08-02 08:11:20.751 986 986 I surfaceflinger: Heap left redzone: fa  08-02 08:11:20.752 986 986 I surfaceflinger: Freed heap region: fd  08-02 08:11:20.752 986 986 I surfaceflinger: Stack left redzone: f1  08-02 08:11:20.752 986 986 I surfaceflinger: Stack mid redzone: f2  08-02 08:11:20.752 986 986 I surfaceflinger: Stack right redzone: f3  08-02 08:11:20.752 986 986 I surfaceflinger: Stack after return: f5  08-02 08:11:20.752 986 986 I surfaceflinger: Stack use after scope: f8  08-02 08:11:20.752 986 986 I surfaceflinger: Global redzone: f9  08-02 08:11:20.753 986 986 I surfaceflinger: Global init order: f6  08-02 08:11:20.753 986 986 I surfaceflinger: Poisoned by user: f7  08-02 08:11:20.753 986 986 I surfaceflinger: Container overflow: fc  08-02 08:11:20.753 986 986 I surfaceflinger: Array cookie: ac  08-02 08:11:20.753 986 986 I surfaceflinger: Intra object redzone: bb  08-02 08:11:20.753 986 986 I surfaceflinger: ASan internal: fe  08-02 08:11:20.753 986 986 I surfaceflinger: Left alloca redzone: ca  08-02 08:11:20.754 986 986 I surfaceflinger: Right alloca redzone: cb  08-02 08:11:20.754 986 986 I surfaceflinger: Shadow gap: cc  08-02 08:11:20.757 986 986 I surfaceflinger: ==986==ABORTING |

解析结果如下：

|  |
| --- |
| ==986==ERROR: AddressSanitizer: stack-use-after-scope on address 0x007fe896bde0 at pc 0x005f05bdfbc0 bp 0x007fe896bdb0 sp 0x007fe896bda8  WRITE of size 4 at 0x007fe896bde0 thread T0  #0 0x5f05bdfbbc in \_Z9test\_testv frameworks/native/services/surfaceflinger/main\_surfaceflinger.cpp:81  #1 0x5f05bdfbbc in main frameworks/native/services/surfaceflinger/main\_surfaceflinger.cpp:129  #2 0x733ed4b798 in \_\_libc\_init bionic/libc/bionic/libc\_init\_dynamic.cpp:136  Address 0x007fe896bde0 is located in stack of thread T0  at offset 32 in frame  #3 0x5f05bdf1a8 in main frameworks/native/services/surfaceflinger/main\_surfaceflinger.cpp:84  This frame has 12 object(s):  [32, 36) 'x.i' (line 78) <== Memory access at offset 32 is inside this variable  [48, 56) 'displayservice.i' (line 65)  [80, 104) 'ref.tmp.i43' (line 66)  [144, 168) 'ref.tmp.i' (line 45)  [208, 232) 'ref.tmp4.i' (line 51)  [272, 280) 'ref.tmp' (line 94)  [304, 312) 'ps' (line 97)  [336, 344) 'flinger' (line 101)  [368, 376) 'sm' (line 116)  [400, 408) 'ref.tmp12' (line 117)  [432, 440) 'ref.tmp14' (line 117)  [464, 468) 'param' (line 122)  HINT: this may be a false positive if your program uses some custom stack unwind mechanism, swapcontext or vfork  (longjmp and C++ exceptions \*are\* supported)  SUMMARY: AddressSanitizer: stack-use-after-scope (/system/bin/surfaceflinger+0x4bbc)  Shadow bytes around the buggy address:  0x001ffd12d760: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0x001ffd12d770: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0x001ffd12d780: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0x001ffd12d790: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0x001ffd12d7a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  =>0x001ffd12d7b0: 00 00 00 00 00 00 00 00 f1 f1 f1 f1[f8]f2 f8 f2  0x001ffd12d7c0: f2 f2 f8 f8 f8 f2 f2 f2 f2 f2 f8 f8 f8 f2 f2 f2  0x001ffd12d7d0: f2 f2 f8 f8 f8 f2 f2 f2 f2 f2 f8 f2 f2 f2 00 f2  0x001ffd12d7e0: f2 f2 00 f2 f2 f2 00 f2 f2 f2 f8 f2 f2 f2 f8 f2  0x001ffd12d7f0: f2 f2 04 f3 00 00 00 00 00 00 00 00 00 00 00 00  0x001ffd12d800: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  Shadow byte legend (one shadow byte represents 8 application bytes):  Addressable: 00  Partially addressable: 01 02 03 04 05 06 07  Heap left redzone: fa  Freed heap region: fd  Stack left redzone: f1  Stack mid redzone: f2  Stack right redzone: f3  Stack after return: f5  Stack use after scope: f8  Global redzone: f9  Global init order: f6  Poisoned by user: f7  Container overflow: fc  Array cookie: ac  Intra object redzone: bb  ASan internal: fe  Left alloca redzone: ca  Right alloca redzone: cb  Shadow gap: cc  ==986==ABORTING |

我们可以看到，可以解析出文件名，以及函数所在行。

结合代码我们可以快速定位问题

# 常见内存错误ASan提示

我们对常见的AddressSanitizer支持的类型进行了验证：

## double-free

double-free 问题是我们遇到最多的问题，

libc原生设计，对于同一地址，申请时分配使用该地址，该地址释放后可以再次申请该地址。如果该地址释放后，没有申请再次释放该地址，此时是不会上报错误的。只有再一次申请时分配使用该地址时，才会上报错误。

由于默认出问题时是在再次申请内存时才会出现问题，只有发生问题时的调用栈，我们无法知道首次在哪里发生问题，再次在哪里进行释放，因此是比较难以分析定的问题。

AddressSanitizer可以检测定位到double-free时的调用栈并及时报错，并可以给出首次free时的调用栈，以及内存申请时的调用栈，对我的问题调试非常有帮助。

### 问题代码

为验证AddressSanitizer是否可以检测到该类型的问题，我们加入下面的代码：

|  |
| --- |
| int test\_test(int ii) {  char \*e = malloc(sizeof(char) \* 10);  if (e == NULL) {  return 0;  }  char \*e2 = e;  free(e);  ylog\_info("ylog zzg == [%d]", ii);  free(e2); //再次释放    return 0;  }  int main(int argc, char \*argv[]) {  ...  int ret = start\_cmd\_srv();  if(-1 == ret) {  return -1;  }  int ii=0;  ii=test\_test(25);  ylog\_info("ylog zzg == [%d]", ii);  start\_active\_log(); |

### AddressSanitizer检测到问题

|  |
| --- |
| 04-11 22:12:49.389 306 306 I ylog : =================================================================  04-11 22:12:49.390 306 306 I ylog : ==306==ERROR: AddressSanitizer: attempting double-free on 0x003100000050 in thread T0:  04-11 22:12:49.407 306 306 I ylog : #0 0x79faaa9e1b (/system/lib64/libclang\_rt.asan-aarch64-android.so+0x9ce1b)  04-11 22:12:49.408 306 306 I ylog : #1 0x558406c48b (/system/bin/ylog+0x6f48b)  04-11 22:12:49.409 306 306 I ylog : #2 0x558406cc37 (/system/bin/ylog+0x6fc37)  04-11 22:12:49.409 306 306 I ylog : #3 0x79fba2fbd3 (/system/lib64/libc.so+0xacbd3)  04-11 22:12:49.410 306 306 I ylog :  04-11 22:12:49.411 306 306 I ylog : 0x003100000050 is located 0 bytes inside of 10-byte region [0x003100000050,0x00310000005a)  04-11 22:12:49.411 306 306 I ylog :  04-11 22:12:49.411 306 306 I ylog : freed by thread T0 here:  04-11 22:12:49.412 306 306 I ylog : #0 0x79faaa9e1b (/system/lib64/libclang\_rt.asan-aarch64-android.so+0x9ce1b)  04-11 22:12:49.413 306 306 I ylog : #1 0x558406c463 (/system/bin/ylog+0x6f463)  04-11 22:12:49.413 306 306 I ylog : #2 0x558406cc37 (/system/bin/ylog+0x6fc37)  04-11 22:12:49.414 306 306 I ylog : #3 0x79fba2fbd3 (/system/lib64/libc.so+0xacbd3)  04-11 22:12:49.415 306 306 I ylog : #4 0x5584003df7 (/system/bin/ylog+0x6df7)  04-11 22:12:49.421 306 306 I ylog : #5 0x79fbe725db (/system/bin/linker64+0x2e5db)  04-11 22:12:49.422 306 306 I ylog :  04-11 22:12:49.422 306 306 I ylog : previously allocated by thread T0 here:  04-11 22:12:49.423 306 306 I ylog : #0 0x79faaaa133 (/system/lib64/libclang\_rt.asan-aarch64-android.so+0x9d133)  04-11 22:12:49.423 306 306 I ylog : #1 0x558406c437 (/system/bin/ylog+0x6f437)  04-11 22:12:49.424 306 306 I ylog : #2 0x558406cc37 (/system/bin/ylog+0x6fc37)  04-11 22:12:49.425 306 306 I ylog : #3 0x79fba2fbd3 (/system/lib64/libc.so+0xacbd3)  04-11 22:12:49.427 306 306 I ylog : #4 0x5584003df7 (/system/bin/ylog+0x6df7)  04-11 22:12:49.432 306 306 I ylog : #5 0x79fbe725db (/system/bin/linker64+0x2e5db)  04-11 22:12:49.433 306 306 I ylog :  04-11 22:12:49.434 306 306 I ylog : SUMMARY: AddressSanitizer: double-free (/system/lib64/libclang\_rt.asan-aarch64-android.so+0x9ce1b)  04-11 22:12:49.434 306 306 I ylog :  04-11 22:12:49.434 306 306 I ylog : ==306==ABORTING |

### 调用栈解析

|  |
| --- |
| =================================================================  ==306==ERROR: AddressSanitizer: attempting double-free on 0x003100000050 in thread T0:  #0 0x79faaa9e1b (/system/lib64/libclang\_rt.asan-aarch64-android.so+0x9ce1b)  #0 0x558406c48b in test\_test vendor/sprd/proprietories-source/ylog/server/main.c:68  #1 0x558406cc37 in main vendor/sprd/proprietories-source/ylog/server/main.c:167  #2 0x79fba2fbd3 in \_\_libc\_init bionic/libc/bionic/libc\_init\_dynamic.cpp:129  0x003100000050 is located 0 bytes inside of 10-byte region [0x003100000050,0x00310000005a)  freed by thread T0 here:  #0 0x79faaa9e1b (/system/lib64/libclang\_rt.asan-aarch64-android.so+0x9ce1b)  #3 0x558406c463 in test\_test vendor/sprd/proprietories-source/ylog/server/main.c:66  #4 0x558406cc37 in main vendor/sprd/proprietories-source/ylog/server/main.c:167  #5 0x79fba2fbd3 in \_\_libc\_init bionic/libc/bionic/libc\_init\_dynamic.cpp:129  #6 0x5584003df7 in \_start\_main bionic/libc/arch-common/bionic/crtbegin.c:45  #7 0x79fbe725db in \_ZL11out\_vformatI18BufferOutputStreamEvRT\_PKcSt9\_\_va\_list bionic/libc/async\_safe/async\_safe\_log.cpp:226  previously allocated by thread T0 here:  #0 0x79faaaa133 (/system/lib64/libclang\_rt.asan-aarch64-android.so+0x9d133)  #8 0x558406c437 in test\_test vendor/sprd/proprietories-source/ylog/server/main.c:60  #9 0x558406cc37 in main vendor/sprd/proprietories-source/ylog/server/main.c:167  #10 0x79fba2fbd3 in \_\_libc\_init bionic/libc/bionic/libc\_init\_dynamic.cpp:129  #11 0x5584003df7 in \_start\_main bionic/libc/arch-common/bionic/crtbegin.c:45  #12 0x79fbe725db in \_ZL11out\_vformatI18BufferOutputStreamEvRT\_PKcSt9\_\_va\_list bionic/libc/async\_safe/async\_safe\_log.cpp:226  SUMMARY: AddressSanitizer: double-free (/system/lib64/libclang\_rt.asan-aarch64-android.so+0x9ce1b)  ==306==ABORTING |

## wild pointer free

野指针问题也是我们常遇到的问题。

### 问题代码

为验证AddressSanitizer是否可以检测到该类型的问题，我们加入下面的代码：

|  |
| --- |
| int test\_test(int ii) {  int \*e = malloc(sizeof(char) \* 10);  if (e == NULL) {  return 0;  }  int \*e2 = e;  for (ii=0;ii<25;ii++){  e2++;  }  ylog\_info("ylog zzg == [%d]", ii);  free(e2);  return 0;  }  int main(int argc, char \*argv[]) {  ...  int ret = start\_cmd\_srv();  if(-1 == ret) {  return -1;  }  int ii=0;  ii=test\_test(25);  ylog\_info("ylog zzg == [%d]", ii);  start\_active\_log(); |

### AddressSanitizer检测到问题

我们将前面的问题代码合入后，将编译生成库替换到手机中后重启运行，AddressSanitizer检测出相关问题，并给出以下问题栈。

|  |
| --- |
| 04-12 02:07:22.022 301 301 I ylog : =================================================================  04-12 02:07:22.023 301 301 I ylog : ==301==ERROR: AddressSanitizer: attempting free on address which was not malloc()-ed: 0x0031000000b4 in thread T0  04-12 02:07:22.042 301 301 I ylog : #0 0x7b95b42e1b (/system/lib64/libclang\_rt.asan-aarch64-android.so+0x9ce1b)  04-12 02:07:22.043 301 301 I ylog : #1 0x56c0a5c4af (/system/bin/ylog+0x6f4af)  04-12 02:07:22.043 301 301 I ylog : #2 0x56c0a5cc5b (/system/bin/ylog+0x6fc5b)  04-12 02:07:22.044 301 301 I ylog : #3 0x7b95a2fbd3 (/system/lib64/libc.so+0xacbd3)  04-12 02:07:22.045 301 301 I ylog :  04-12 02:07:22.046 301 301 I ylog : Address 0x0031000000b4 is a wild pointer.  04-12 02:07:22.046 301 301 I ylog :  04-12 02:07:22.047 301 301 I ylog : SUMMARY: AddressSanitizer: bad-free (/system/lib64/libclang\_rt.asan-aarch64-android.so+0x9ce1b)  04-12 02:07:22.047 301 301 I ylog :  04-12 02:07:22.048 301 301 I ylog : ==301==ABORTING |

### 解析后的调用栈

|  |
| --- |
| =================================================================  ==301==ERROR: AddressSanitizer: attempting free on address which was not malloc()-ed: 0x0031000000b4 in thread T0  #0 0x7b95b42e1b (/system/lib64/libclang\_rt.asan-aarch64-android.so+0x9ce1b)  #0 0x56c0a5c4af in test\_test vendor/sprd/proprietories-source/ylog/server/main.c:71  #1 0x56c0a5cc5b in main vendor/sprd/proprietories-source/ylog/server/main.c:170  #2 0x7b95a2fbd3 in \_\_libc\_init bionic/libc/bionic/libc\_init\_dynamic.cpp:129  Address 0x0031000000b4 is a wild pointer.  SUMMARY: AddressSanitizer: bad-free (/system/lib64/libclang\_rt.asan-aarch64-android.so+0x9ce1b)  ==301==ABORTING |

## stack-buffer-overflow

异常类型描述

栈溢出是我们会遇到的内存错误类型

对于一些简单的栈溢出，编译器就可以检测到，如下面的例子

|  |
| --- |
| int stack\_array[100];  stack\_array[-1]=321; |

我们在编译的时候，编译器会提示如下错误

|  |
| --- |
| frameworks/av/services/camera/libcameraservice/CameraService.cpp:358:3: error: array index -1 is before the beginning of the array [-Werror,-Warray-bounds]  stack\_array[-1]=321;  ^ ~~  frameworks/av/services/camera/libcameraservice/CameraService.cpp:356:3: note: array 'stack\_array' declared here  int stack\_array[100];  ^  1 error generated. |

对于一些动态的问题，编译器不能捕获，但AddressSanitizer可以检测到这种错误，比如下面这种错误。

|  |
| --- |
| int stack\_array[100];  stack\_array[0-ii]=321; |

### stack overflow代码

为验证AddressSanitizer是否可以检测到该类型的问题，我们加入下面的代码：

sprdroid9.0\_trunk\_18c/frameworks/av/services/camera/libcameraservice/CameraService.cpp

|  |
| --- |
| +int CameraService::test\_test(int ii) {  + int i=0;  + int j=0;  + int stack\_array[100];  +  + for (i=0;i<100;i++){  + j = ii+i ;  + ALOGI("zzg j=[%d] ",j );  + stack\_array[j]=321; // BOOM  + }  +  + return stack\_array[0];  +}  void CameraService::onTorchStatusChangedLocked(const String8& cameraId,  TorchModeStatus newStatus) {  ALOGI("%s: Torch status changed for cameraId=%s, newStatus=%d",  \_\_FUNCTION\_\_, cameraId.string(), newStatus);  + int ii=70;  + int ret=0;  + ret=test\_test(ii);  TorchModeStatus status;  status\_t res = getTorchStatusLocked(cameraId, &status); |

### stack overflow AddressSanitizer检测到问题

我们将前面的问题代码合入后，将编译生成库替换到手机中后重启运行，AddressSanitizer检测出相关问题，并给出以下问题栈。

|  |
| --- |
| 04-12 00:36:31.315 412 412 I CameraService: zzg j=[70]  …  04-12 00:36:31.322 412 412 I CameraService: zzg j=[98]  04-12 00:36:31.322 412 412 I CameraService: zzg j=[99]  04-12 00:36:31.322 412 412 I CameraService: zzg j=[100] |

|  |
| --- |
| 04-12 00:36:31.323 412 412 I : =================================================================  04-12 00:36:31.324 412 412 I : ==412==ERROR: AddressSanitizer: stack-buffer-overflow on address 0xff8eafc0 at pc 0xe5d0b2d8 bp 0xff8eadd8 sp 0xff8eadd4  04-12 00:36:31.324 412 412 I : WRITE of size 4 at 0xff8eafc0 thread T0  04-12 00:36:31.537 412 412 I : #0 0xe5d0b2d7 (/system/lib/libcameraservice.so+0x772d7)  04-12 00:36:31.538 412 412 I : #1 0xe5d3715f (/system/lib/libcameraservice.so+0xa315f)  04-12 00:36:31.538 412 412 I : #2 0xe5d36383 (/system/lib/libcameraservice.so+0xa2383)  04-12 00:36:31.538 412 412 I : #3 0xe5d6604b (/system/lib/libcameraservice.so+0xd204b)  04-12 00:36:31.539 412 412 I : #4 0xe5d65d87 (/system/lib/libcameraservice.so+0xd1d87)  04-12 00:36:31.539 412 412 I : #5 0xe5e1b6cf (/system/lib/libcameraservice.so+0x1876cf)  04-12 00:36:31.540 412 412 I : #6 0xe5e1b367 (/system/lib/libcameraservice.so+0x187367)  04-12 00:36:31.540 412 412 I : #7 0xe5d23fd7 (/system/lib/libcameraservice.so+0x8ffd7)  04-12 00:36:31.540 412 412 I : #8 0xe5d22263 (/system/lib/libcameraservice.so+0x8e263)  04-12 00:36:31.541 412 412 I : #9 0xe5d2584f (/system/lib/libcameraservice.so+0x9184f)  04-12 00:36:31.541 412 412 I : #10 0xe6e60021 (/system/lib/libcamera\_client.so+0x21021)  04-12 00:36:31.542 412 412 I : #11 0xe5d30a9f (/system/lib/libcameraservice.so+0x9ca9f)  04-12 00:36:31.542 412 412 I : #12 0xe603a161 (/system/lib/libbinder.so+0x36161)  04-12 00:36:31.542 412 412 I : #13 0xe60419a5 (/system/lib/libbinder.so+0x3d9a5)  04-12 00:36:31.543 412 412 I : #14 0xe6041735 (/system/lib/libbinder.so+0x3d735)  04-12 00:36:31.543 412 412 I : #15 0xe6041c5d (/system/lib/libbinder.so+0x3dc5d)  04-12 00:36:31.543 412 412 I : #16 0xb8f18e83 (/system/bin/cameraserver+0xe83)  04-12 00:36:31.544 412 412 I : #17 0xe719578d (/system/lib/libc.so+0x8c78d)  04-12 00:36:31.544 412 412 I : #18 0xb8f18c9f (/system/bin/cameraserver+0xc9f)  04-12 00:36:31.555 412 412 I :  04-12 00:36:31.558 412 412 I : Address 0xff8eafc0 is located in stack of thread T0  04-12 00:36:31.559 412 412 I : at offset 480 in frame  04-12 00:36:31.559 412 412 I : #0 0xe5d0b077 (/system/lib/libcameraservice.so+0x77077)  04-12 00:36:31.560 412 412 I :  04-12 00:36:31.560 412 412 I : This frame has 5 object(s):  04-12 00:36:31.561 412 412 I : [16, 24) 'ref.tmp.i.i91'  04-12 00:36:31.563 412 412 I : [48, 56) 'ref.tmp.i.i'  04-12 00:36:31.565 412 412 I : [80, 480) 'stack\_array.i' (line 357) <== Memory access at offset 480 overflows this variable  04-12 00:36:31.565 412 412 I : [544, 556) 'agg.tmp.ensured'  04-12 00:36:31.567 412 412 I : [576, 580) 'ref.tmp48' (line 425)  04-12 00:36:31.569 412 412 I :  04-12 00:36:31.569 412 412 I : HINT: this may be a false positive if your program uses some custom stack unwind mechanism or swapcontext  04-12 00:36:31.570 412 412 I : (longjmp and C++ exceptions \*are\* supported)  04-12 00:36:31.570 412 412 I :  04-12 00:36:31.571 412 412 I : SUMMARY: AddressSanitizer: stack-buffer-overflow (/system/lib/libcameraservice.so+0x772d7)  04-12 00:36:31.571 412 412 I :  04-12 00:36:31.571 412 412 I : Shadow bytes around the buggy address:  04-12 00:36:31.577 412 412 I : 0xe53f55a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  04-12 00:36:31.577 412 412 I : 0xe53f55b0: 00 00 00 00 00 00 00 00 00 00 00 00 f1 f1 f8 f2  04-12 00:36:31.577 412 412 I : 0xe53f55c0: f2 f2 f8 f2 f2 f2 00 00 00 00 00 00 00 00 00 00  04-12 00:36:31.577 412 412 I : 0xe53f55d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  04-12 00:36:31.577 412 412 I : 0xe53f55e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  04-12 00:36:31.577 412 412 I : =>0xe53f55f0: 00 00 00 00 00 00 00 00[f2]f2 f2 f2 f2 f2 f2 f2  04-12 00:36:31.577 412 412 I : 0xe53f5600: 00 04 f2 f2 f8 f3 00 00 00 00 00 00 00 00 00 00  04-12 00:36:31.577 412 412 I : 0xe53f5610: 00 00 00 00 00 00 00 00 00 00 00 00 f1 f1 f8 f2  04-12 00:36:31.577 412 412 I : 0xe53f5620: f2 f2 00 04 f2 f2 f8 f2 00 f3 f3 f3 00 00 00 00  04-12 00:36:31.578 412 412 I : 0xe53f5630: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  04-12 00:36:31.578 412 412 I : 0xe53f5640: 00 00 00 00 00 00 00 00 f1 f1 00 f2 f2 f2 f8 f2  04-12 00:36:31.578 412 412 I : Shadow byte legend (one shadow byte represents 8 application bytes):  04-12 00:36:31.578 412 412 I : Addressable: 00  04-12 00:36:31.578 412 412 I : Partially addressable: 01 02 03 04 05 06 07  04-12 00:36:31.578 412 412 I : Heap left redzone: fa  04-12 00:36:31.578 412 412 I : Freed heap region: fd  04-12 00:36:31.578 412 412 I : Stack left redzone: f1  04-12 00:36:31.578 412 412 I : Stack mid redzone: f2  04-12 00:36:31.578 412 412 I : Stack right redzone: f3  04-12 00:36:31.578 412 412 I : Stack after return: f5  04-12 00:36:31.578 412 412 I : Stack use after scope: f8  04-12 00:36:31.578 412 412 I : Global redzone: f9  04-12 00:36:31.579 412 412 I : Global init order: f6  04-12 00:36:31.580 412 412 I : Poisoned by user: f7  04-12 00:36:31.580 412 412 I : Container overflow: fc  04-12 00:36:31.580 412 412 I : Array cookie: ac  04-12 00:36:31.580 412 412 I : Intra object redzone: bb  04-12 00:36:31.580 412 412 I : ASan internal: fe  04-12 00:36:31.580 412 412 I : Left alloca redzone: ca  04-12 00:36:31.580 412 412 I : Right alloca redzone: cb  04-12 00:36:31.580 412 412 I :  04-12 00:36:31.581 412 412 I : ==412==ABORTING  04-12 00:36:31.581 412 412 I : |

### stack overflow栈解析

|  |
| --- |
| =================================================================  ==412==ERROR: AddressSanitizer: stack-buffer-overflow on address 0xff8eafc0 at pc 0xe5d0b2d8 bp 0xff8eadd8 sp 0xff8eadd4  WRITE of size 4 at 0xff8eafc0 thread T0  #0 0xe5d0b2d7 in \_ZN7android13CameraService9test\_testEi frameworks/av/services/camera/libcameraservice/CameraService.cpp:362  #1 0xe5d0b2d7 in \_ZN7android13CameraService26onTorchStatusChangedLockedERKNS\_7String8ENS\_8hardware6camera6common4V1\_015TorchModeStatusE frameworks/av/services/camera/libcameraservice/CameraService.cpp:375  #2 0xe5d3715f in operator() frameworks/av/services/camera/libcameraservice/CameraService.cpp:2950  #3 0xe5d3715f in \_ZN7android13CameraService11CameraState12updateStatusIZNS0\_12updateStatusENS0\_14StatusInternalERKNS\_7String8ESt16initializer\_listIS3\_EE3$\_0EEvS3\_S6\_S8\_T\_ frameworks/av/services/camera/libcameraservice/CameraService.cpp:3004  #4 0xe5d3715f in \_ZN7android13CameraService12updateStatusENS0\_14StatusInternalERKNS\_7String8ESt16initializer\_listIS1\_E frameworks/av/services/camera/libcameraservice/CameraService.cpp:2936  #5 0xe5d36383 in \_ZN7android13CameraService12updateStatusENS0\_14StatusInternalERKNS\_7String8E frameworks/av/services/camera/libcameraservice/CameraService.cpp:2918  #6 0xe5d36383 in \_ZN7android13CameraService11BasicClient14startCameraOpsEv frameworks/av/services/camera/libcameraservice/CameraService.cpp:2291  #7 0xe5d6604b in \_ZN7android17Camera2ClientBaseINS\_22CameraDeviceClientBaseEE14initializeImplINS\_2spINS\_21CameraProviderManagerEEEEEiT\_RKNS\_7String8E frameworks/av/services/camera/libcameraservice/common/Camera2ClientBase.cpp:98  #8 0xe5d65d87 in \_ZN7android17Camera2ClientBaseINS\_22CameraDeviceClientBaseEE10initializeENS\_2spINS\_21CameraProviderManagerEEERKNS\_7String8E frameworks/av/services/camera/libcameraservice/common/Camera2ClientBase.cpp:85  #9 0xe5e1b6cf in \_ZN7android18CameraDeviceClient14initializeImplINS\_2spINS\_21CameraProviderManagerEEEEEiT\_RKNS\_7String8E frameworks/av/services/camera/libcameraservice/api2/CameraDeviceClient.cpp:101  #10 0xe5e1b367 in \_ZN7android18CameraDeviceClient10initializeENS\_2spINS\_21CameraProviderManagerEEERKNS\_7String8E frameworks/av/services/camera/libcameraservice/api2/CameraDeviceClient.cpp:93  #11 0xe5d23fd7 in \_ZN7android13CameraService13connectHelperINS\_8hardware7camera222ICameraDeviceCallbacksENS\_18CameraDeviceClientEEENS\_6binder6StatusERKNS\_2spIT\_EERKNS\_7String8EiiRKNS\_8String16EiiNS0\_8apiLevelEbbRNS8\_IT0\_EE frameworks/av/services/camera/libcameraservice/CameraService.cpp:1400  #12 0xe5d22263 in \_ZN7android13CameraService13connectDeviceERKNS\_2spINS\_8hardware7camera222ICameraDeviceCallbacksEEERKNS\_8String16ESA\_iPNS1\_INS3\_17ICameraDeviceUserEEE frameworks/av/services/camera/libcameraservice/CameraService.cpp:1285  #13 0xe5d2584f in \_ZTv0\_n40\_N7android13CameraService13connectDeviceERKNS\_2spINS\_8hardware7camera222ICameraDeviceCallbacksEEERKNS\_8String16ESA\_iPNS1\_INS3\_17ICameraDeviceUserEEE frameworks/av/services/camera/libcameraservice/CameraService.cpp:?  #14 0xe6e60021 in \_ZN7android8hardware15BnCameraService10onTransactEjRKNS\_6ParcelEPS2\_j out/soong/.intermediates/frameworks/av/camera/libcamera\_client/android\_arm\_armv7-a-neon\_cortex-a15\_core\_shared/gen/aidl/frameworks/av/camera/aidl/android/hardware/ICameraService.cpp:657  #15 0xe5d30a9f in \_ZN7android13CameraService10onTransactEjRKNS\_6ParcelEPS1\_j frameworks/av/services/camera/libcameraservice/CameraService.cpp:2040  #16 0xe603a161 in \_ZN7android7BBinder8transactEjRKNS\_6ParcelEPS1\_j frameworks/native/libs/binder/Binder.cpp:129  #17 0xe60419a5 in \_ZN7android14IPCThreadState14executeCommandEi frameworks/native/libs/binder/IPCThreadState.cpp:1161  #18 0xe6041735 in \_ZN7android14IPCThreadState20getAndExecuteCommandEv frameworks/native/libs/binder/IPCThreadState.cpp:464  #19 0xe6041c5d in \_ZN7android14IPCThreadState14joinThreadPoolEb frameworks/native/libs/binder/IPCThreadState.cpp:544  #20 0xb8f18e83 in main frameworks/av/camera/cameraserver/main\_cameraserver.cpp:37  #21 0xe719578d in \_\_libc\_init bionic/libc/bionic/libc\_init\_dynamic.cpp:129  #22 0xb8f18c9f in \_start\_main bionic/libc/arch-common/bionic/crtbegin.c:45  Address 0xff8eafc0 is located in stack of thread T0  at offset 480 in frame  #23 0xe5d0b077 in \_ZN7android13CameraService26onTorchStatusChangedLockedERKNS\_7String8ENS\_8hardware6camera6common4V1\_015TorchModeStatusE frameworks/av/services/camera/libcameraservice/CameraService.cpp:369  This frame has 5 object(s):  [16, 24) 'ref.tmp.i.i91'  [48, 56) 'ref.tmp.i.i'  [80, 480) 'stack\_array.i' (line 357) <== Memory access at offset 480 overflows this variable  [544, 556) 'agg.tmp.ensured'  [576, 580) 'ref.tmp48' (line 425)  HINT: this may be a false positive if your program uses some custom stack unwind mechanism or swapcontext  (longjmp and C++ exceptions \*are\* supported)  SUMMARY: AddressSanitizer: stack-buffer-overflow (/system/lib/libcameraservice.so+0x772d7)  Shadow bytes around the buggy address:  0xe53f55a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0xe53f55b0: 00 00 00 00 00 00 00 00 00 00 00 00 f1 f1 f8 f2  0xe53f55c0: f2 f2 f8 f2 f2 f2 00 00 00 00 00 00 00 00 00 00  0xe53f55d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0xe53f55e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  =>0xe53f55f0: 00 00 00 00 00 00 00 00[f2]f2 f2 f2 f2 f2 f2 f2  0xe53f5600: 00 04 f2 f2 f8 f3 00 00 00 00 00 00 00 00 00 00  0xe53f5610: 00 00 00 00 00 00 00 00 00 00 00 00 f1 f1 f8 f2  0xe53f5620: f2 f2 00 04 f2 f2 f8 f2 00 f3 f3 f3 00 00 00 00  0xe53f5630: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0xe53f5640: 00 00 00 00 00 00 00 00 f1 f1 00 f2 f2 f2 f8 f2  Shadow byte legend (one shadow byte represents 8 application bytes):  Addressable: 00  Partially addressable: 01 02 03 04 05 06 07  Heap left redzone: fa  Freed heap region: fd  Stack left redzone: f1  Stack mid redzone: f2  Stack right redzone: f3  Stack after return: f5  Stack use after scope: f8  Global redzone: f9  Global init order: f6  Poisoned by user: f7  Container overflow: fc  Array cookie: ac  Intra object redzone: bb  ASan internal: fe  Left alloca redzone: ca  Right alloca redzone: cb  ==412==ABORTING |

## heap-buffer-overflow

### AddressSanitizer检测到问题

Ylog代码之前没有使用AddressSanitizer工具检查过，因此，我们的运气很好恰巧发现了ylog的一个问题。

在log中我们可以看会报如下错误：

|  |
| --- |
| 02-15 02:01:43.466 3007 3007 I ylog : =================================================================  02-15 02:01:43.468 3007 3007 I ylog : ==3007==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x003c0000017f at pc 0x00617109e4dc bp 0x007fd2ccad40 sp 0x007fd2ccad38  02-15 02:01:43.469 3007 3007 I ylog : READ of size 1 at 0x003c0000017f thread T0  02-15 02:01:43.492 3007 3007 I ylog : #0 0x617109e4db (/system/bin/ylog+0x3a4db)  02-15 02:01:43.493 3007 3007 I ylog : #1 0x617109e96f (/system/bin/ylog+0x3a96f)  02-15 02:01:43.494 3007 3007 I ylog : #2 0x6171077463 (/system/bin/ylog+0x13463)  02-15 02:01:43.495 3007 3007 I ylog : #3 0x6171095863 (/system/bin/ylog+0x31863)  02-15 02:01:43.496 3007 3007 I ylog : #4 0x79dc3acbd3 (/system/lib64/libc.so+0xacbd3)  02-15 02:01:43.497 3007 3007 I ylog :  02-15 02:01:43.497 3007 3007 I ylog : 0x003c0000017f is located 0 bytes to the right of 319-byte region [0x003c00000040,0x003c0000017f)  02-15 02:01:43.498 3007 3007 I ylog :  02-15 02:01:43.498 3007 3007 I ylog : allocated by thread T0 here:  02-15 02:01:43.501 3007 3007 I ylog : #0 0x79db446133 (/system/lib64/libclang\_rt.asan-aarch64-android.so+0x9d133)  02-15 02:01:43.503 3007 3007 I ylog : #1 0x617109df1b (/system/bin/ylog+0x39f1b)  02-15 02:01:43.504 3007 3007 I ylog : #2 0x617109e8f3 (/system/bin/ylog+0x3a8f3)  02-15 02:01:43.504 3007 3007 I ylog : #3 0x6171077463 (/system/bin/ylog+0x13463)  02-15 02:01:43.505 3007 3007 I ylog : #4 0x6171095863 (/system/bin/ylog+0x31863)  02-15 02:01:43.506 3007 3007 I ylog : #5 0x79dc3acbd3 (/system/lib64/libc.so+0xacbd3)  02-15 02:01:43.507 3007 3007 I ylog : #6 0x617106ab47 (/system/bin/ylog+0x6b47)  02-15 02:01:43.516 3007 3007 I ylog : #7 0x79dc52b54f (/system/bin/linker64+0x2e54f)  02-15 02:01:43.518 3007 3007 I ylog :  02-15 02:01:43.518 3007 3007 I ylog : SUMMARY: AddressSanitizer: heap-buffer-overflow (/system/bin/ylog+0x3a4db)  02-15 02:01:43.519 3007 3007 I ylog :  02-15 02:01:43.520 3007 3007 I ylog : Shadow bytes around the buggy address:  02-15 02:01:43.520 3007 3007 I ylog : 0x00177fffffd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  02-15 02:01:43.521 3007 3007 I ylog : 0x00177fffffe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  02-15 02:01:43.521 3007 3007 I ylog : 0x00177ffffff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  02-15 02:01:43.521 3007 3007 I ylog : 0x001780000000: fa fa fa fa fa fa fa fa 00 00 00 00 00 00 00 00  02-15 02:01:43.521 3007 3007 I ylog : 0x001780000010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  02-15 02:01:43.521 3007 3007 I ylog : =>0x001780000020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00[07]  02-15 02:01:43.522 3007 3007 I ylog : 0x001780000030: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa  02-15 02:01:43.522 3007 3007 I ylog : 0x001780000040: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa  02-15 02:01:43.522 3007 3007 I ylog : 0x001780000050: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa  02-15 02:01:43.522 3007 3007 I ylog : 0x001780000060: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa  02-15 02:01:43.523 3007 3007 I ylog : 0x001780000070: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa  02-15 02:01:43.523 3007 3007 I ylog : Shadow byte legend (one shadow byte represents 8 application bytes):  02-15 02:01:43.523 3007 3007 I ylog : Addressable: 00  02-15 02:01:43.523 3007 3007 I ylog : Partially addressable: 01 02 03 04 05 06 07  02-15 02:01:43.523 3007 3007 I ylog : Heap left redzone: fa  02-15 02:01:43.524 3007 3007 I ylog : Freed heap region: fd  02-15 02:01:43.524 3007 3007 I ylog : Stack left redzone: f1  02-15 02:01:43.524 3007 3007 I ylog : Stack mid redzone: f2  02-15 02:01:43.524 3007 3007 I ylog : Stack right redzone: f3  02-15 02:01:43.526 3007 3007 I ylog : Stack after return: f5  02-15 02:01:43.526 3007 3007 I ylog : Stack use after scope: f8  02-15 02:01:43.527 3007 3007 I ylog : Global redzone: f9  02-15 02:01:43.527 3007 3007 I ylog : Global init order: f6  02-15 02:01:43.527 3007 3007 I ylog : Poisoned by user: f7  02-15 02:01:43.528 3007 3007 I ylog : Container overflow: fc  02-15 02:01:43.528 3007 3007 I ylog : Array cookie: ac  02-15 02:01:43.528 3007 3007 I ylog : Intra object redzone: bb  02-15 02:01:43.528 3007 3007 I ylog : ASan internal: fe  02-15 02:01:43.529 3007 3007 I ylog : Left alloca redzone: ca  02-15 02:01:43.529 3007 3007 I ylog : Right alloca redzone: cb  02-15 02:01:43.529 3007 3007 I ylog :  02-15 02:01:43.530 3007 3007 I ylog : ==3007==ABORTING |

同时AddressSanitizer在检查到错误之后，主动生成一个SIGABRT 的NativeCrash

|  |
| --- |
| 02-15 02:01:43.532 3007 3007 F libc : Fatal signal 6 (SIGABRT), code -6 (SI\_TKILL) in tid 3007 (ylog), pid 3007 (ylog)  02-15 02:01:43.648 3019 3019 I crash\_dump64: obtaining output fd from tombstoned, type: kDebuggerdTombstone  02-15 02:01:43.650 456 456 I /system/bin/tombstoned: received crash request for pid 3007  02-15 02:01:43.665 3019 3019 I crash\_dump64: performing dump of process 3007 (target tid = 3007)  02-15 02:01:43.668 3019 3019 F DEBUG : \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\*  02-15 02:01:43.668 3019 3019 F DEBUG : Native Crash TIME: 70013  02-15 02:01:43.668 3019 3019 F DEBUG : \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\*  02-15 02:01:43.669 3019 3019 F DEBUG : Build fingerprint: 'SPRD/sp9832e\_1h10\_oversea/sp9832e\_1h10:9/PPR1.180610.011/07524:userdebug/test-keys'  02-15 02:01:43.669 3019 3019 F DEBUG : Revision: '0'  02-15 02:01:43.669 3019 3019 F DEBUG : ABI: 'arm64'  02-15 02:01:43.669 3019 3019 F DEBUG : pid: 3007, tid: 3007, name: ylog >>> /system/bin/ylog <<<  02-15 02:01:43.669 3019 3019 F DEBUG : signal 6 (SIGABRT), code -6 (SI\_TKILL), fault addr --------  02-15 02:01:43.669 3019 3019 F DEBUG : Abort message too long: claimed length = 2520  02-15 02:01:43.669 3019 3019 F DEBUG : x0 0000000000000000 x1 0000000000000bbf x2 0000000000000006 x3 0000000000000008  02-15 02:01:43.669 3019 3019 F DEBUG : x4 464d4853514e4140 x5 464d4853514e4140 x6 464d4853514e4140 x7 7f7f7f7f7f7f7f7f  02-15 02:01:43.669 3019 3019 F DEBUG : x8 0000000000000083 x9 0d3fbca6ce13de90 x10 0000000000000000 x11 fffffffc7fffffdf  02-15 02:01:43.669 3019 3019 F DEBUG : x12 0000000000000001 x13 0000000000000000 x14 0000000000000006 x15 0000000000000000  02-15 02:01:43.669 3019 3019 F DEBUG : x16 00000079dc3ef2c8 x17 00000079dc32d0d0 x18 0000000000000004 x19 0000000000000bbf  02-15 02:01:43.669 3019 3019 F DEBUG : x20 0000000000000bbf x21 00000079dc130108 x22 00000079db7263b8 x23 00000079dc1300f0  02-15 02:01:43.670 3019 3019 F DEBUG : x24 000000617109e4dc x25 0000000000000001 x26 0000007fd2ccad80 x27 0000003c00000128  02-15 02:01:43.670 3019 3019 F DEBUG : x28 0000000000000000 x29 0000007fd2cca060  02-15 02:01:43.670 3019 3019 F DEBUG : sp 0000007fd2cca020 lr 00000079dc321bfc pc 00000079dc321c24  02-15 02:01:43.681 3019 3019 F DEBUG :  02-15 02:01:43.681 3019 3019 F DEBUG : backtrace:  02-15 02:01:43.682 3019 3019 F DEBUG : #00 pc 0000000000021c24 /system/lib64/libc.so (abort+116)  02-15 02:01:43.682 3019 3019 F DEBUG : #01 pc 0000000000033690 /system/lib64/libclang\_rt.asan-aarch64-android.so (\_\_sanitizer::Abort()+56)  02-15 02:01:43.682 3019 3019 F DEBUG : #02 pc 0000000000031250 /system/lib64/libclang\_rt.asan-aarch64-android.so (\_\_sanitizer::Die()+164)  02-15 02:01:43.682 3019 3019 F DEBUG : #03 pc 00000000000a1fd0 /system/lib64/libclang\_rt.asan-aarch64-android.so (\_\_asan::ScopedInErrorReport::~ScopedInErrorReport()+316)  02-15 02:01:43.682 3019 3019 F DEBUG : #04 pc 00000000000a1768 /system/lib64/libclang\_rt.asan-aarch64-android.so (\_\_asan::ReportGenericError(unsigned long, unsigned long, unsigned long, unsigned long, bool, unsigned long, unsigned int, bool)+348)  02-15 02:01:43.682 3019 3019 F DEBUG : #05 pc 00000000000a21c0 /system/lib64/libclang\_rt.asan-aarch64-android.so (\_\_asan\_report\_load1+48)  02-15 02:01:43.682 3019 3019 F DEBUG : #06 pc 000000000003a4d8 /system/bin/ylog (next\_token+808)  02-15 02:01:43.682 3019 3019 F DEBUG : #07 pc 000000000003a96c /system/bin/ylog (load\_configs+488)  02-15 02:01:43.682 3019 3019 F DEBUG : #08 pc 0000000000013460 /system/bin/ylog (start\_ylog+1300)  02-15 02:01:43.682 3019 3019 F DEBUG : #09 pc 0000000000031860 /system/bin/ylog (main+1036)  02-15 02:01:43.682 3019 3019 F DEBUG : #10 pc 00000000000acbd0 /system/lib64/libc.so (\_\_libc\_init+88) |

### 解析后的log

可参照本文后面的章节，对调用栈进行解析

解析结果如下：

|  |
| --- |
| =================================================================  ==2847==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x0033000000c0 at pc 0x0059ed18f4dc bp 0x007ffe3ee560 sp 0x007ffe3ee558  READ of size 1 at 0x0033000000c0 thread T0  #0 0x59ed18f4db in next\_token vendor/sprd/proprietories-source/ylog/server/config\_parser.c:96  #1 0x59ed18f96f in load\_configs vendor/sprd/proprietories-source/ylog/server/config\_parser.c:392  #2 0x59ed168463 in init\_logger vendor/sprd/proprietories-source/ylog/server/ylogger.c:754  #3 0x59ed168463 in start\_ylog vendor/sprd/proprietories-source/ylog/server/ylogger.c:833  #4 0x59ed186863 in main vendor/sprd/proprietories-source/ylog/server/main.c:160  #5 0x74572b0bd3 in \_\_libc\_init bionic/libc/bionic/libc\_init\_dynamic.cpp:129  0x0033000000c0 is located 0 bytes to the right of 64-byte region [0x003300000080,0x0033000000c0)  allocated by thread T0 here:  #0 0x74575ad133 (/system/lib64/libclang\_rt.asan-aarch64-android.so+0x9d133)  #6 0x59ed18ef1b in ylog\_load\_config vendor/sprd/proprietories-source/ylog/server/config\_parser.c:196  #7 0x59ed18f8f3 in load\_configs vendor/sprd/proprietories-source/ylog/server/config\_parser.c:383  #8 0x59ed168463 in init\_logger vendor/sprd/proprietories-source/ylog/server/ylogger.c:754  #9 0x59ed168463 in start\_ylog vendor/sprd/proprietories-source/ylog/server/ylogger.c:833  #10 0x59ed186863 in main vendor/sprd/proprietories-source/ylog/server/main.c:160  #11 0x74572b0bd3 in \_\_libc\_init bionic/libc/bionic/libc\_init\_dynamic.cpp:129 |

### 问题代码分析

我们可以看到，可以解析出文件名，以及函数所在行

内存在config\_parser.c:196行申请，在config\_parser.c:96行越界

|  |
| --- |
| 41 static int next\_token(struct parse\_state \*state) {  …  88 textdone:  89 state->ptr = x;  90 \*s = 0;  91 return T\_TEXT;  92 text:  93 state->text = s = x;  94 textresume:  95 for(;;) {  96 switch(\*x) <--------------------------------- 此处越界  97 {  98 case 0:  99 goto textdone;  100 |

|  |
| --- |
| 181static char \*ylog\_load\_config(const char \*path) {  182 char \*data;  183 struct stat sb;  184 int fd = open(path, O\_RDONLY | O\_NOFOLLOW | O\_CLOEXEC);  185  186 if(fd == -1) {  187 return NULL;  188 }  189  190 if(fstat(fd, &sb) == -1) {  191 ylog\_error("fstat failed for '%s': %s\n", path, strerror(errno));  192 CLOSE(fd);  193 return NULL;  194 }  195  196 data = malloc(sb.st\_size + 64); <-------------------------- 此处申请  197 if(data) {  198 int ret;  199 char \*p = data;  200 char \*pmax = p + sb.st\_size;  201 do  202 { |

### 检查验证

为验证工具是否越界，已经分析其如何越界，我们在申请和越界的地方以及x++的地方加log验证。

复测抓取log，我们看到申请大小为318，对x操作的地址从0x3c00000040增长318个增长到到0x3c0000017e发生越界，工具确实有效，发现了代码中隐藏的问题。

|  |
| --- |
| 02-15 02:58:55.422 301 301 W YLOG : [02-15 02:58:55.422] ylog<warn> ylog : == ylog\_load\_config 7ff69b11a0 size= 318  02-15 02:58:55.422 301 301 W YLOG : [02-15 02:58:55.422] ylog<warn> ylog : + x= [0x3c00000040]  02-15 02:58:55.422 301 301 W YLOG : [02-15 02:58:55.422] ylog<warn> ylog :-2 x= [0x3c00000040] \*x = [0x79]  02-15 02:58:55.422 301 301 W YLOG : [02-15 02:58:55.422] ylog<warn> ylog : + x= [0x3c00000041]  02-15 02:58:55.422 301 301 W YLOG : [02-15 02:58:55.422] ylog<warn> ylog :-2 x= [0x3c00000041] \*x = [0x6c]  02-15 02:58:55.422 301 301 W YLOG : [02-15 02:58:55.422] ylog<warn> ylog : + x= [0x3c00000042]  02-15 02:58:55.423 301 301 W YLOG : [02-15 02:58:55.422] ylog<warn> ylog :-2 x= [0x3c00000042] \*x = [0x6f]  02-15 02:58:55.423 301 301 W YLOG : [02-15 02:58:55.423] ylog<warn> ylog : + x= [0x3c00000043]  02-15 02:58:55.423 301 301 W YLOG : [02-15 02:58:55.423] ylog<warn> ylog :-2 x= [0x3c00000043] \*x = [0x67]  02-15 02:58:55.423 301 301 W YLOG : [02-15 02:58:55.423] ylog<warn> ylog : + x= [0x3c00000044]  02-15 02:58:55.423 301 301 W YLOG : [02-15 02:58:55.423] ylog<warn> ylog : + x= [0x3c00000045]  02-15 02:58:55.423 301 301 W YLOG : [02-15 02:58:55.423] ylog<warn> ylog :-2 x= [0x3c00000045] \*x = [0x65]  02-15 02:58:55.423 301 301 W YLOG : [02-15 02:58:55.423] ylog<warn> ylog : + x= [0x3c00000046]  02-15 02:58:55.423 301 301 W YLOG : [02-15 02:58:55.423] ylog<warn> ylog :-2 x= [0x3c00000046] \*x = [0x6e]  02-15 02:58:55.423 301 301 W YLOG : [02-15 02:58:55.423] ylog<warn> ylog : + x= [0x3c00000047]  ...  02-15 02:58:55.486 301 301 W YLOG : [02-15 02:58:55.486] ylog<warn> ylog :-2 x= [0x3c00000179] \*x = [0xbe]  02-15 02:58:55.486 301 301 W YLOG : [02-15 02:58:55.486] ylog<warn> ylog : + x= [0x3c0000017a]  02-15 02:58:55.486 301 301 W YLOG : [02-15 02:58:55.486] ylog<warn> ylog :-2 x= [0x3c0000017a] \*x = [0xbe]  02-15 02:58:55.486 301 301 W YLOG : [02-15 02:58:55.486] ylog<warn> ylog : + x= [0x3c0000017b]  02-15 02:58:55.486 301 301 W YLOG : [02-15 02:58:55.486] ylog<warn> ylog :-2 x= [0x3c0000017b] \*x = [0xbe]  02-15 02:58:55.486 301 301 W YLOG : [02-15 02:58:55.486] ylog<warn> ylog : + x= [0x3c0000017c]  02-15 02:58:55.486 301 301 W YLOG : [02-15 02:58:55.486] ylog<warn> ylog :-2 x= [0x3c0000017c] \*x = [0xbe]  02-15 02:58:55.487 301 301 W YLOG : [02-15 02:58:55.487] ylog<warn> ylog : + x= [0x3c0000017d]  02-15 02:58:55.487 301 301 W YLOG : [02-15 02:58:55.487] ylog<warn> ylog :-2 x= [0x3c0000017d] \*x = [0xbe]  02-15 02:58:55.487 301 301 W YLOG : [02-15 02:58:55.487] ylog<warn> ylog : + x= [0x3c0000017e]  02-15 02:58:55.487 301 301 I ylog : =================================================================  02-15 02:58:55.489 301 301 I ylog : ==301==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x003c0000017e at pc 0x005e4dbc577c bp 0x007ff69b1280 sp 0x007ff69b1278  02-15 02:58:55.489 301 301 I ylog :  02-15 02:58:55.490 301 301 I ylog : READ of size 1 at 0x003c0000017e thread T0  02-15 02:58:55.511 301 301 I ylog : #0 0x5e4dbc577b (/system/bin/ylog+0x3a77b)  02-15 02:58:55.512 301 301 I ylog : #1 0x5e4dbc5b57 (/system/bin/ylog+0x3ab57)  02-15 02:58:55.513 301 301 I ylog : #2 0x5e4db9e463 (/system/bin/ylog+0x13463)  02-15 02:58:55.513 301 301 I ylog : #3 0x5e4dbbc863 (/system/bin/ylog+0x31863)  02-15 02:58:55.514 301 301 I ylog : #4 0x786696dbd3 (/system/lib64/libc.so+0xacbd3)  02-15 02:58:55.516 301 301 I ylog :  02-15 02:58:55.517 301 301 I ylog : 0x003c0000017e is located 0 bytes to the right of 318-byte region [0x003c00000040,0x003c0000017e)  02-15 02:58:55.517 301 301 I ylog :  02-15 02:58:55.518 301 301 I ylog : allocated by thread T0 here:  02-15 02:58:55.518 301 301 I ylog : #0 0x7865a50133 (/system/lib64/libclang\_rt.asan-aarch64-android.so+0x9d133)  02-15 02:58:55.519 301 301 I ylog : #1 0x5e4dbc4f1b (/system/bin/ylog+0x39f1b)  02-15 02:58:55.520 301 301 I ylog : #2 0x5e4dbc5adb (/system/bin/ylog+0x3aadb)  02-15 02:58:55.521 301 301 I ylog : #3 0x5e4db9e463 (/system/bin/ylog+0x13463)  02-15 02:58:55.522 301 301 I ylog : #4 0x5e4dbbc863 (/system/bin/ylog+0x31863)  02-15 02:58:55.522 301 301 I ylog : #5 0x786696dbd3 (/system/lib64/libc.so+0xacbd3)  02-15 02:58:55.523 301 301 I ylog : #6 0x5e4db91b47 (/system/bin/ylog+0x6b47)  02-15 02:58:55.536 301 301 I ylog : #7 0x7866b0854f (/system/bin/linker64+0x2e54f)  02-15 02:58:55.538 301 301 I ylog :  02-15 02:58:55.538 301 301 I ylog : SUMMARY: AddressSanitizer: heap-buffer-overflow (/system/bin/ylog+0x3a77b) |

## heap-use-after-free

### 问题代码

为验证AddressSanitizer是否可以检测到该类型的问题，我们加入下面的代码：

sprdroid9.0\_trunk\_18c/frameworks/av/services/camera/libcameraservice/CameraService.cpp

|  |
| --- |
| diff --git a/services/camera/libcameraservice/CameraService.cpp b/services/camera/libcameraservice/CameraService.cpp  index f96b60d..82159ce 100644  --- a/services/camera/libcameraservice/CameraService.cpp  +++ b/services/camera/libcameraservice/CameraService.cpp  @@ -351,11 +351,20 @@ void CameraService::onTorchStatusChanged(const String8& cameraId,  onTorchStatusChangedLocked(cameraId, newStatus);  }    +int CameraService::test\_test() {  + int \*array = new int[100];  + delete [] array;  + return array[7]; // BOOM  +}  +  void CameraService::onTorchStatusChangedLocked(const String8& cameraId,  TorchModeStatus newStatus) {  ALOGI("%s: Torch status changed for cameraId=%s, newStatus=%d",  \_\_FUNCTION\_\_, cameraId.string(), newStatus);    + int ret=test\_test();  +  TorchModeStatus status;  status\_t res = getTorchStatusLocked(cameraId, &status);  if (res) {  @@ -408,6 +417,7 @@ void CameraService::onTorchStatusChangedLocked(const String8& cameraId,  i->onTorchStatusChanged(mapToInterface(newStatus), String16{cameraId});  }  }  + ALOGI("ret=newStatus=%d", ret );  }    Status CameraService::getNumberOfCameras(int32\_t type, int32\_t\* numCameras) {  @@ -2077,6 +2087,7 @@ void CameraService::releaseSound() {  }  }    void CameraService::playSound(sound\_kind kind) {  ATRACE\_CALL();    diff --git a/services/camera/libcameraservice/CameraService.h b/services/camera/libcameraservice/CameraService.h  index 8d4bcdb..6a5736d 100644  --- a/services/camera/libcameraservice/CameraService.h  +++ b/services/camera/libcameraservice/CameraService.h  @@ -426,7 +426,7 @@ public:  private:    typedef hardware::camera::common::V1\_0::CameraDeviceStatus CameraDeviceStatus;  -  + int test\_test() ;  /\*\*  \* Typesafe version of device status, containing both the HAL-layer and the service interface-  \* layer values. |

### AddressSanitizer检测到问题

我们将前面的问题代码合入后，将编译生成库替换到手机中后重启运行，AddressSanitizer检测出相关问题，并给出以下问题栈。

|  |
| --- |
| 02-15 10:06:06.120 398 398 I CameraService: onTorchStatusChangedLocked: Torch status changed for cameraId=0, newStatus=0  02-15 10:06:06.124 398 398 I : =================================================================  02-15 10:06:06.126 398 398 I : ==398==ERROR: AddressSanitizer: heap-use-after-free on address 0xd0903c5c at pc 0xf53e0818 bp 0xffacd718 sp 0xffacd714  02-15 10:06:06.126 398 398 I : READ of size 4 at 0xd0903c5c thread T0  02-15 10:06:06.282 398 398 I : #0 0xf53e0817 (/system/lib/libcameraservice.so+0x77817)  02-15 10:06:06.283 398 398 I : #1 0xf540bd6f (/system/lib/libcameraservice.so+0xa2d6f)  02-15 10:06:06.283 398 398 I : #2 0xf540af93 (/system/lib/libcameraservice.so+0xa1f93)  02-15 10:06:06.283 398 398 I : #3 0xf543ac5b (/system/lib/libcameraservice.so+0xd1c5b)  02-15 10:06:06.284 398 398 I : #4 0xf543a997 (/system/lib/libcameraservice.so+0xd1997)  02-15 10:06:06.284 398 398 I : #5 0xf54f02df (/system/lib/libcameraservice.so+0x1872df)  02-15 10:06:06.285 398 398 I : #6 0xf54eff77 (/system/lib/libcameraservice.so+0x186f77)  02-15 10:06:06.285 398 398 I : #7 0xf53f8be7 (/system/lib/libcameraservice.so+0x8fbe7)  02-15 10:06:06.286 398 398 I : #8 0xf53f6e73 (/system/lib/libcameraservice.so+0x8de73)  02-15 10:06:06.286 398 398 I : #9 0xf53fa45f (/system/lib/libcameraservice.so+0x9145f)  02-15 10:06:06.286 398 398 I : #10 0xf591d021 (/system/lib/libcamera\_client.so+0x21021)  02-15 10:06:06.287 398 398 I : #11 0xf54056af (/system/lib/libcameraservice.so+0x9c6af)  02-15 10:06:06.287 398 398 I : #12 0xf5db7161 (/system/lib/libbinder.so+0x36161)  02-15 10:06:06.287 398 398 I : #13 0xf5dbe9a5 (/system/lib/libbinder.so+0x3d9a5)  02-15 10:06:06.288 398 398 I : #14 0xf5dbe735 (/system/lib/libbinder.so+0x3d735)  02-15 10:06:06.288 398 398 I : #15 0xf5dbec5d (/system/lib/libbinder.so+0x3dc5d)  02-15 10:06:06.288 398 398 I : #16 0xb2a41e83 (/system/bin/cameraserver+0xe83)  02-15 10:06:06.289 398 398 I : #17 0xf60d478d (/system/lib/libc.so+0x8c78d)  02-15 10:06:06.289 398 398 I : #18 0xb2a41c9f (/system/bin/cameraserver+0xc9f)  02-15 10:06:06.290 398 398 I :  02-15 10:06:06.290 398 398 I : 0xd0903c5c is located 28 bytes inside of 400-byte region [0xd0903c40,0xd0903dd0)  02-15 10:06:06.290 398 398 I :  02-15 10:06:06.290 398 398 I : freed by thread T0 here:  02-15 10:06:06.291 398 398 I : #0 0xf61c4d1b (/system/lib/libclang\_rt.asan-arm-android.so+0xbdd1b)  02-15 10:06:06.291 398 398 I : #1 0xf53e018b (/system/lib/libcameraservice.so+0x7718b)  02-15 10:06:06.292 398 398 I : #2 0xf540bd6f (/system/lib/libcameraservice.so+0xa2d6f)  02-15 10:06:06.292 398 398 I : #3 0xf540af93 (/system/lib/libcameraservice.so+0xa1f93)  02-15 10:06:06.292 398 398 I : #4 0xf543ac5b (/system/lib/libcameraservice.so+0xd1c5b)  02-15 10:06:06.293 398 398 I : #5 0xf543a997 (/system/lib/libcameraservice.so+0xd1997)  02-15 10:06:06.293 398 398 I : #6 0xf54f02df (/system/lib/libcameraservice.so+0x1872df)  02-15 10:06:06.293 398 398 I : #7 0xf54eff77 (/system/lib/libcameraservice.so+0x186f77)  02-15 10:06:06.294 398 398 I : #8 0xf53f8be7 (/system/lib/libcameraservice.so+0x8fbe7)  02-15 10:06:06.294 398 398 I : #9 0xf53f6e73 (/system/lib/libcameraservice.so+0x8de73)  02-15 10:06:06.295 398 398 I : #10 0xf53fa45f (/system/lib/libcameraservice.so+0x9145f)  02-15 10:06:06.297 398 398 I : #11 0xf591d021 (/system/lib/libcamera\_client.so+0x21021)  02-15 10:06:06.297 398 398 I : #12 0xf5db7161 (/system/lib/libbinder.so+0x36161)  02-15 10:06:06.298 398 398 I : #13 0xf60d478d (/system/lib/libc.so+0x8c78d)  02-15 10:06:06.298 398 398 I :  02-15 10:06:06.298 398 398 I : previously allocated by thread T0 here:  02-15 10:06:06.299 398 398 I : #0 0xf61c4177 (/system/lib/libclang\_rt.asan-arm-android.so+0xbd177)  02-15 10:06:06.299 398 398 I : #1 0xf53e0183 (/system/lib/libcameraservice.so+0x77183)  02-15 10:06:06.300 398 398 I : #2 0xf540bd6f (/system/lib/libcameraservice.so+0xa2d6f)  02-15 10:06:06.300 398 398 I : #3 0xf540af93 (/system/lib/libcameraservice.so+0xa1f93)  02-15 10:06:06.300 398 398 I : #4 0xf543ac5b (/system/lib/libcameraservice.so+0xd1c5b)  02-15 10:06:06.301 398 398 I : #5 0xf543a997 (/system/lib/libcameraservice.so+0xd1997)  02-15 10:06:06.301 398 398 I : #6 0xf54f02df (/system/lib/libcameraservice.so+0x1872df)  02-15 10:06:06.302 398 398 I : #7 0xf54eff77 (/system/lib/libcameraservice.so+0x186f77)  02-15 10:06:06.303 398 398 I : #8 0xf53f8be7 (/system/lib/libcameraservice.so+0x8fbe7)  02-15 10:06:06.303 398 398 I : #9 0xf53f6e73 (/system/lib/libcameraservice.so+0x8de73)  02-15 10:06:06.312 398 398 I : #10 0xf53fa45f (/system/lib/libcameraservice.so+0x9145f)  02-15 10:06:06.312 398 398 I : #11 0xf591d021 (/system/lib/libcamera\_client.so+0x21021)  02-15 10:06:06.313 398 398 I : #12 0xf5db7161 (/system/lib/libbinder.so+0x36161)  02-15 10:06:06.313 398 398 I : #13 0xf60d478d (/system/lib/libc.so+0x8c78d)  02-15 10:06:06.313 398 398 I :  02-15 10:06:06.314 398 398 I : SUMMARY: AddressSanitizer: heap-use-after-free (/system/lib/libcameraservice.so+0x77817)  02-15 10:06:06.314 398 398 I :  02-15 10:06:06.326 398 398 I : Shadow bytes around the buggy address:  02-15 10:06:06.326 398 398 I : 0xeecd0730: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa  02-15 10:06:06.326 398 398 I : 0xeecd0740: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa  02-15 10:06:06.326 398 398 I : 0xeecd0750: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa  02-15 10:06:06.326 398 398 I : 0xeecd0760: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa  02-15 10:06:06.327 398 398 I : 0xeecd0770: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa  02-15 10:06:06.327 398 398 I : =>0xeecd0780: fa fa fa fa fa fa fa fa fd fd fd[fd]fd fd fd fd  02-15 10:06:06.327 398 398 I : 0xeecd0790: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd  02-15 10:06:06.327 398 398 I : 0xeecd07a0: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd  02-15 10:06:06.327 398 398 I : 0xeecd07b0: fd fd fd fd fd fd fd fd fd fd fa fa fa fa fa fa  02-15 10:06:06.327 398 398 I : 0xeecd07c0: fa fa fa fa fa fa fa fa fd fd fd fd fd fd fd fd  02-15 10:06:06.327 398 398 I : 0xeecd07d0: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd  02-15 10:06:06.327 398 398 I : Shadow byte legend (one shadow byte represents 8 application bytes):  02-15 10:06:06.327 398 398 I : Addressable: 00  02-15 10:06:06.327 398 398 I : Partially addressable: 01 02 03 04 05 06 07  02-15 10:06:06.327 398 398 I : Heap left redzone: fa  02-15 10:06:06.327 398 398 I : Freed heap region: fd  02-15 10:06:06.327 398 398 I : Stack left redzone: f1  02-15 10:06:06.327 398 398 I : Stack mid redzone: f2  02-15 10:06:06.327 398 398 I : Stack right redzone: f3  02-15 10:06:06.328 398 398 I : Stack after return: f5  02-15 10:06:06.328 398 398 I : Stack use after scope: f8  02-15 10:06:06.328 398 398 I : Global redzone: f9  02-15 10:06:06.328 398 398 I : Global init order: f6  02-15 10:06:06.328 398 398 I : Poisoned by user: f7  02-15 10:06:06.328 398 398 I : Container overflow: fc  02-15 10:06:06.328 398 398 I : Array cookie: ac  02-15 10:06:06.328 398 398 I : Intra object redzone: bb  02-15 10:06:06.328 398 398 I : ASan internal: fe  02-15 10:06:06.328 398 398 I : Left alloca redzone: ca  02-15 10:06:06.328 398 398 I : Right alloca redzone: cb  02-15 10:06:06.328 398 398 I :  02-15 10:06:06.329 398 398 I : ==398==ABORTING  02-15 10:06:06.329 398 398 I : |

### 栈解析如下

|  |
| --- |
| ==398==ERROR: AddressSanitizer: heap-use-after-free on address 0xd0903c5c at pc 0xf53e0818 bp 0xffacd718 sp 0xffacd714  READ of size 4 at 0xd0903c5c thread T0  #0 0xf53e0817 in \_ZN7android13CameraService9test\_testEv frameworks/av/services/camera/libcameraservice/CameraService.cpp:357  #1 0xf53e0817 in \_ZN7android13CameraService26onTorchStatusChangedLockedERKNS\_7String8ENS\_8hardware6camera6common4V1\_015TorchModeStatusE frameworks/av/services/camera/libcameraservice/CameraService.cpp:366  #2 0xf540bd6f in operator() frameworks/av/services/camera/libcameraservice/CameraService.cpp:2943  #3 0xf540bd6f in \_ZN7android13CameraService11CameraState12updateStatusIZNS0\_12updateStatusENS0\_14StatusInternalERKNS\_7String8ESt16initializer\_listIS3\_EE3$\_0EEvS3\_S6\_S8\_T\_ frameworks/av/services/camera/libcameraservice/CameraService.cpp:2997  #4 0xf540bd6f in \_ZN7android13CameraService12updateStatusENS0\_14StatusInternalERKNS\_7String8ESt16initializer\_listIS1\_E frameworks/av/services/camera/libcameraservice/CameraService.cpp:2929  #5 0xf540af93 in \_ZN7android13CameraService12updateStatusENS0\_14StatusInternalERKNS\_7String8E frameworks/av/services/camera/libcameraservice/CameraService.cpp:2911  #6 0xf540af93 in \_ZN7android13CameraService11BasicClient14startCameraOpsEv frameworks/av/services/camera/libcameraservice/CameraService.cpp:2284  #7 0xf543ac5b in \_ZN7android17Camera2ClientBaseINS\_22CameraDeviceClientBaseEE14initializeImplINS\_2spINS\_21CameraProviderManagerEEEEEiT\_RKNS\_7String8E frameworks/av/services/camera/libcameraservice/common/Camera2ClientBase.cpp:98  #8 0xf543a997 in \_ZN7android17Camera2ClientBaseINS\_22CameraDeviceClientBaseEE10initializeENS\_2spINS\_21CameraProviderManagerEEERKNS\_7String8E frameworks/av/services/camera/libcameraservice/common/Camera2ClientBase.cpp:85  #9 0xf54f02df in \_ZN7android18CameraDeviceClient14initializeImplINS\_2spINS\_21CameraProviderManagerEEEEEiT\_RKNS\_7String8E frameworks/av/services/camera/libcameraservice/api2/CameraDeviceClient.cpp:101  #10 0xf54eff77 in \_ZN7android18CameraDeviceClient10initializeENS\_2spINS\_21CameraProviderManagerEEERKNS\_7String8E frameworks/av/services/camera/libcameraservice/api2/CameraDeviceClient.cpp:93  #11 0xf53f8be7 in \_ZN7android13CameraService13connectHelperINS\_8hardware7camera222ICameraDeviceCallbacksENS\_18CameraDeviceClientEEENS\_6binder6StatusERKNS\_2spIT\_EERKNS\_7String8EiiRKNS\_8String16EiiNS0\_8apiLevelEbbRNS8\_IT0\_EE frameworks/av/services/camera/libcameraservice/CameraService.cpp:1392  #12 0xf53f6e73 in \_ZN7android13CameraService13connectDeviceERKNS\_2spINS\_8hardware7camera222ICameraDeviceCallbacksEEERKNS\_8String16ESA\_iPNS1\_INS3\_17ICameraDeviceUserEEE frameworks/av/services/camera/libcameraservice/CameraService.cpp:1277  #13 0xf53fa45f in \_ZTv0\_n40\_N7android13CameraService13connectDeviceERKNS\_2spINS\_8hardware7camera222ICameraDeviceCallbacksEEERKNS\_8String16ESA\_iPNS1\_INS3\_17ICameraDeviceUserEEE frameworks/av/services/camera/libcameraservice/CameraService.cpp:?  #14 0xf591d021 in \_ZN7android8hardware15BnCameraService10onTransactEjRKNS\_6ParcelEPS2\_j out/soong/.intermediates/frameworks/av/camera/libcamera\_client/android\_arm\_armv7-a-neon\_cortex-a15\_core\_shared/gen/aidl/frameworks/av/camera/aidl/android/hardware/ICameraService.cpp:657  #15 0xf54056af in \_ZN7android13CameraService10onTransactEjRKNS\_6ParcelEPS1\_j frameworks/av/services/camera/libcameraservice/CameraService.cpp:2032  #16 0xf5db7161 in \_ZN7android7BBinder8transactEjRKNS\_6ParcelEPS1\_j frameworks/native/libs/binder/Binder.cpp:129  #17 0xf5dbe9a5 in \_ZN7android14IPCThreadState14executeCommandEi frameworks/native/libs/binder/IPCThreadState.cpp:1161  #18 0xf5dbe735 in \_ZN7android14IPCThreadState20getAndExecuteCommandEv frameworks/native/libs/binder/IPCThreadState.cpp:464  #19 0xf5dbec5d in \_ZN7android14IPCThreadState14joinThreadPoolEb frameworks/native/libs/binder/IPCThreadState.cpp:544  #20 0xb2a41e83 in main frameworks/av/camera/cameraserver/main\_cameraserver.cpp:37  #21 0xf60d478d in \_\_libc\_init bionic/libc/bionic/libc\_init\_dynamic.cpp:129  #22 0xb2a41c9f in \_start\_main bionic/libc/arch-common/bionic/crtbegin.c:45  0xd0903c5c is located 28 bytes inside of 400-byte region [0xd0903c40,0xd0903dd0)  freed by thread T0 here:  #0 0xf61c4d1b (/system/lib/libclang\_rt.asan-arm-android.so+0xbdd1b)  #23 0xf53e018b in \_ZN7android13CameraService9test\_testEv frameworks/av/services/camera/libcameraservice/CameraService.cpp:356  #24 0xf53e018b in \_ZN7android13CameraService26onTorchStatusChangedLockedERKNS\_7String8ENS\_8hardware6camera6common4V1\_015TorchModeStatusE frameworks/av/services/camera/libcameraservice/CameraService.cpp:366  #25 0xf540bd6f in operator() frameworks/av/services/camera/libcameraservice/CameraService.cpp:2943  #26 0xf540bd6f in \_ZN7android13CameraService11CameraState12updateStatusIZNS0\_12updateStatusENS0\_14StatusInternalERKNS\_7String8ESt16initializer\_listIS3\_EE3$\_0EEvS3\_S6\_S8\_T\_ frameworks/av/services/camera/libcameraservice/CameraService.cpp:2997  #27 0xf540bd6f in \_ZN7android13CameraService12updateStatusENS0\_14StatusInternalERKNS\_7String8ESt16initializer\_listIS1\_E frameworks/av/services/camera/libcameraservice/CameraService.cpp:2929  #28 0xf540af93 in \_ZN7android13CameraService12updateStatusENS0\_14StatusInternalERKNS\_7String8E frameworks/av/services/camera/libcameraservice/CameraService.cpp:2911  #29 0xf540af93 in \_ZN7android13CameraService11BasicClient14startCameraOpsEv frameworks/av/services/camera/libcameraservice/CameraService.cpp:2284  #30 0xf543ac5b in \_ZN7android17Camera2ClientBaseINS\_22CameraDeviceClientBaseEE14initializeImplINS\_2spINS\_21CameraProviderManagerEEEEEiT\_RKNS\_7String8E frameworks/av/services/camera/libcameraservice/common/Camera2ClientBase.cpp:98  #31 0xf543a997 in \_ZN7android17Camera2ClientBaseINS\_22CameraDeviceClientBaseEE10initializeENS\_2spINS\_21CameraProviderManagerEEERKNS\_7String8E frameworks/av/services/camera/libcameraservice/common/Camera2ClientBase.cpp:85  #32 0xf54f02df in \_ZN7android18CameraDeviceClient14initializeImplINS\_2spINS\_21CameraProviderManagerEEEEEiT\_RKNS\_7String8E frameworks/av/services/camera/libcameraservice/api2/CameraDeviceClient.cpp:101  #33 0xf54eff77 in \_ZN7android18CameraDeviceClient10initializeENS\_2spINS\_21CameraProviderManagerEEERKNS\_7String8E frameworks/av/services/camera/libcameraservice/api2/CameraDeviceClient.cpp:93  #34 0xf53f8be7 in \_ZN7android13CameraService13connectHelperINS\_8hardware7camera222ICameraDeviceCallbacksENS\_18CameraDeviceClientEEENS\_6binder6StatusERKNS\_2spIT\_EERKNS\_7String8EiiRKNS\_8String16EiiNS0\_8apiLevelEbbRNS8\_IT0\_EE frameworks/av/services/camera/libcameraservice/CameraService.cpp:1392  #35 0xf53f6e73 in \_ZN7android13CameraService13connectDeviceERKNS\_2spINS\_8hardware7camera222ICameraDeviceCallbacksEEERKNS\_8String16ESA\_iPNS1\_INS3\_17ICameraDeviceUserEEE frameworks/av/services/camera/libcameraservice/CameraService.cpp:1277  #36 0xf53fa45f in \_ZTv0\_n40\_N7android13CameraService13connectDeviceERKNS\_2spINS\_8hardware7camera222ICameraDeviceCallbacksEEERKNS\_8String16ESA\_iPNS1\_INS3\_17ICameraDeviceUserEEE frameworks/av/services/camera/libcameraservice/CameraService.cpp:?  #37 0xf591d021 in \_ZN7android8hardware15BnCameraService10onTransactEjRKNS\_6ParcelEPS2\_j out/soong/.intermediates/frameworks/av/camera/libcamera\_client/android\_arm\_armv7-a-neon\_cortex-a15\_core\_shared/gen/aidl/frameworks/av/camera/aidl/android/hardware/ICameraService.cpp:657  #38 0xf5db7161 in \_ZN7android7BBinder8transactEjRKNS\_6ParcelEPS1\_j frameworks/native/libs/binder/Binder.cpp:129  #39 0xf60d478d in \_\_libc\_init bionic/libc/bionic/libc\_init\_dynamic.cpp:129  previously allocated by thread T0 here:  #0 0xf61c4177 (/system/lib/libclang\_rt.asan-arm-android.so+0xbd177)  #40 0xf53e0183 in \_ZN7android13CameraService9test\_testEv frameworks/av/services/camera/libcameraservice/CameraService.cpp:355  #41 0xf53e0183 in \_ZN7android13CameraService26onTorchStatusChangedLockedERKNS\_7String8ENS\_8hardware6camera6common4V1\_015TorchModeStatusE frameworks/av/services/camera/libcameraservice/CameraService.cpp:366  #42 0xf540bd6f in operator() frameworks/av/services/camera/libcameraservice/CameraService.cpp:2943  #43 0xf540bd6f in \_ZN7android13CameraService11CameraState12updateStatusIZNS0\_12updateStatusENS0\_14StatusInternalERKNS\_7String8ESt16initializer\_listIS3\_EE3$\_0EEvS3\_S6\_S8\_T\_ frameworks/av/services/camera/libcameraservice/CameraService.cpp:2997  #44 0xf540bd6f in \_ZN7android13CameraService12updateStatusENS0\_14StatusInternalERKNS\_7String8ESt16initializer\_listIS1\_E frameworks/av/services/camera/libcameraservice/CameraService.cpp:2929  #45 0xf540af93 in \_ZN7android13CameraService12updateStatusENS0\_14StatusInternalERKNS\_7String8E frameworks/av/services/camera/libcameraservice/CameraService.cpp:2911  #46 0xf540af93 in \_ZN7android13CameraService11BasicClient14startCameraOpsEv frameworks/av/services/camera/libcameraservice/CameraService.cpp:2284  #47 0xf543ac5b in \_ZN7android17Camera2ClientBaseINS\_22CameraDeviceClientBaseEE14initializeImplINS\_2spINS\_21CameraProviderManagerEEEEEiT\_RKNS\_7String8E frameworks/av/services/camera/libcameraservice/common/Camera2ClientBase.cpp:98  #48 0xf543a997 in \_ZN7android17Camera2ClientBaseINS\_22CameraDeviceClientBaseEE10initializeENS\_2spINS\_21CameraProviderManagerEEERKNS\_7String8E frameworks/av/services/camera/libcameraservice/common/Camera2ClientBase.cpp:85  #49 0xf54f02df in \_ZN7android18CameraDeviceClient14initializeImplINS\_2spINS\_21CameraProviderManagerEEEEEiT\_RKNS\_7String8E frameworks/av/services/camera/libcameraservice/api2/CameraDeviceClient.cpp:101  #50 0xf54eff77 in \_ZN7android18CameraDeviceClient10initializeENS\_2spINS\_21CameraProviderManagerEEERKNS\_7String8E frameworks/av/services/camera/libcameraservice/api2/CameraDeviceClient.cpp:93  #51 0xf53f8be7 in \_ZN7android13CameraService13connectHelperINS\_8hardware7camera222ICameraDeviceCallbacksENS\_18CameraDeviceClientEEENS\_6binder6StatusERKNS\_2spIT\_EERKNS\_7String8EiiRKNS\_8String16EiiNS0\_8apiLevelEbbRNS8\_IT0\_EE frameworks/av/services/camera/libcameraservice/CameraService.cpp:1392  #52 0xf53f6e73 in \_ZN7android13CameraService13connectDeviceERKNS\_2spINS\_8hardware7camera222ICameraDeviceCallbacksEEERKNS\_8String16ESA\_iPNS1\_INS3\_17ICameraDeviceUserEEE frameworks/av/services/camera/libcameraservice/CameraService.cpp:1277  #53 0xf53fa45f in \_ZTv0\_n40\_N7android13CameraService13connectDeviceERKNS\_2spINS\_8hardware7camera222ICameraDeviceCallbacksEEERKNS\_8String16ESA\_iPNS1\_INS3\_17ICameraDeviceUserEEE frameworks/av/services/camera/libcameraservice/CameraService.cpp:?  #54 0xf591d021 in \_ZN7android8hardware15BnCameraService10onTransactEjRKNS\_6ParcelEPS2\_j out/soong/.intermediates/frameworks/av/camera/libcamera\_client/android\_arm\_armv7-a-neon\_cortex-a15\_core\_shared/gen/aidl/frameworks/av/camera/aidl/android/hardware/ICameraService.cpp:657  #55 0xf5db7161 in \_ZN7android7BBinder8transactEjRKNS\_6ParcelEPS1\_j frameworks/native/libs/binder/Binder.cpp:129  #56 0xf60d478d in \_\_libc\_init bionic/libc/bionic/libc\_init\_dynamic.cpp:129 |

## stack-use-after-scope

stack-use-after-scope是栈在范围之外使用，AddressSanitizer可以检测到这种类型的错误

### 问题代码

为验证AddressSanitizer是否可以检测到该类型的问题，我们加入下面的代码：

sprdroid9.0\_trunk\_18c/vendor/sprd/proprietories-source/ylog/server/main.c

|  |
| --- |
| +volatile int \*p = 0;  +int test\_test() {  +  + {  + int x = 0;  + p = &x;  + }  + \*p = 5;  +  + return 0;  +  +}  int main(int argc, char \*argv[]) {  ...  int ret = start\_cmd\_srv();  if(-1 == ret) {  return -1;  }    + test\_test();  start\_active\_log(); |

### AddressSanitizer检测到问题

我们将前面的问题代码合入后，将编译生成库替换到手机中后重启运行，AddressSanitizer检测出相关问题，并给出以下问题栈。

|  |
| --- |
| 04-12 04:03:20.168 4642 4642 I ylog : =================================================================  04-12 04:03:20.168 4642 4642 I ylog :  04-12 04:03:20.169 4642 4642 I ylog :  04-12 04:03:20.169 4642 4642 I ylog : ==4642==ERROR: AddressSanitizer: stack-use-after-scope on address 0x007fc1898b80 at pc 0x0060d36059b8 bp 0x007fc1898b50 sp 0x007fc1898b48  04-12 04:03:20.169 4642 4642 I ylog :  04-12 04:03:20.170 4642 4642 I ylog :  04-12 04:03:20.170 4642 4642 I ylog : WRITE of size 4 at 0x007fc1898b80 thread T0  04-12 04:03:20.170 4642 4642 I ylog :  04-12 04:03:20.185 4642 4642 I ylog : #0 0x60d36059b7 (/system/bin/ylog+0x319b7)  04-12 04:03:20.185 4642 4642 I ylog :  04-12 04:03:20.186 4642 4642 I ylog : #1 0x7d5ccedbd3 (/system/lib64/libc.so+0xacbd3)  04-12 04:03:20.186 4642 4642 I ylog :  04-12 04:03:20.187 4642 4642 I ylog :  04-12 04:03:20.187 4642 4642 I ylog : Address 0x007fc1898b80 is located in stack of thread T0  04-12 04:03:20.188 4642 4642 I ylog : at offset 32 in frame  04-12 04:03:20.188 4642 4642 I ylog :  04-12 04:03:20.188 4642 4642 I ylog :  04-12 04:03:20.189 4642 4642 I ylog : #0 0x60d3605463 (/system/bin/ylog+0x31463)  04-12 04:03:20.189 4642 4642 I ylog :  04-12 04:03:20.189 4642 4642 I ylog :  04-12 04:03:20.190 4642 4642 I ylog : This frame has 5 object(s):  04-12 04:03:20.190 4642 4642 I ylog :  04-12 04:03:20.190 4642 4642 I ylog : [32, 36) 'x.i' (line 62) <== Memory access at offset 32 is inside this variable  04-12 04:03:20.190 4642 4642 I ylog :  04-12 04:03:20.191 4642 4642 I ylog : [48, 140) 'enable' (line 74)  04-12 04:03:20.191 4642 4642 I ylog :  04-12 04:03:20.191 4642 4642 I ylog : [176, 268) 'debuggable' (line 75)  04-12 04:03:20.192 4642 4642 I ylog :  04-12 04:03:20.192 4642 4642 I ylog : [304, 560) 'service\_para' (line 77)  04-12 04:03:20.192 4642 4642 I ylog :  04-12 04:03:20.193 4642 4642 I ylog : [624, 716) 'bootmode' (line 78)  04-12 04:03:20.193 4642 4642 I ylog :  04-12 04:03:20.193 4642 4642 I ylog : HINT: this may be a false positive if your program uses some custom stack unwind mechanism or swapcontext  04-12 04:03:20.193 4642 4642 I ylog :  04-12 04:03:20.194 4642 4642 I ylog : (longjmp and C++ exceptions \*are\* supported)  04-12 04:03:20.194 4642 4642 I ylog :  04-12 04:03:20.194 4642 4642 I ylog : SUMMARY: AddressSanitizer: stack-use-after-scope (/system/bin/ylog+0x319b7)  04-12 04:03:20.194 4642 4642 I ylog :  04-12 04:03:20.195 4642 4642 I ylog : Shadow bytes around the buggy address:  04-12 04:03:20.196 4642 4642 I ylog : 0x001ff8313120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  04-12 04:03:20.196 4642 4642 I ylog : 0x001ff8313130: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  04-12 04:03:20.196 4642 4642 I ylog : 0x001ff8313140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  04-12 04:03:20.196 4642 4642 I ylog : 0x001ff8313150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  04-12 04:03:20.196 4642 4642 I ylog : 0x001ff8313160: 00 00 00 00 00 00 00 00 00 00 00 00 f1 f1 f1 f1  04-12 04:03:20.197 4642 4642 I ylog : =>0x001ff8313170:[f8]f2 00 00 00 00 00 00 00 00 00 00 00 04 f2 f2  04-12 04:03:20.197 4642 4642 I ylog : 0x001ff8313180: f2 f2 00 00 00 00 00 00 00 00 00 00 00 04 f2 f2  04-12 04:03:20.197 4642 4642 I ylog : 0x001ff8313190: f2 f2 00 00 00 00 00 00 00 00 00 00 00 00 00 00  04-12 04:03:20.197 4642 4642 I ylog : 0x001ff83131a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  04-12 04:03:20.197 4642 4642 I ylog : 0x001ff83131b0: 00 00 f2 f2 f2 f2 f2 f2 f2 f2 00 00 00 00 00 00  04-12 04:03:20.198 4642 4642 I ylog : 0x001ff83131c0: 00 00 00 00 00 04 f3 f3 f3 f3 f3 f3 00 00 00 00  04-12 04:03:20.198 4642 4642 I ylog : Shadow byte legend (one shadow byte represents 8 application bytes):  04-12 04:03:20.198 4642 4642 I ylog : Addressable: 00  04-12 04:03:20.198 4642 4642 I ylog : Partially addressable: 01 02 03 04 05 06 07  04-12 04:03:20.198 4642 4642 I ylog : Heap left redzone: fa  04-12 04:03:20.198 4642 4642 I ylog : Freed heap region: fd  04-12 04:03:20.199 4642 4642 I ylog : Stack left redzone: f1  04-12 04:03:20.199 4642 4642 I ylog : Stack mid redzone: f2  04-12 04:03:20.199 4642 4642 I ylog : Stack right redzone: f3  04-12 04:03:20.199 4642 4642 I ylog : Stack after return: f5  04-12 04:03:20.199 4642 4642 I ylog : Stack use after scope: f8  04-12 04:03:20.199 4642 4642 I ylog : Global redzone: f9  04-12 04:03:20.199 4642 4642 I ylog : Global init order: f6  04-12 04:03:20.200 4642 4642 I ylog : Poisoned by user: f7  04-12 04:03:20.200 4642 4642 I ylog : Container overflow: fc  04-12 04:03:20.200 4642 4642 I ylog : Array cookie: ac  04-12 04:03:20.200 4642 4642 I ylog : Intra object redzone: bb  04-12 04:03:20.201 4642 4642 I ylog : ASan internal: fe  04-12 04:03:20.201 4642 4642 I ylog : Left alloca redzone: ca  04-12 04:03:20.201 4642 4642 I ylog : Right alloca redzone: cb  04-12 04:03:20.201 4642 4642 I ylog :  04-12 04:03:20.202 4642 4642 I ylog : ==4642==ABORTING |

### 栈解析

|  |
| --- |
| =================================================================  ==4642==ERROR: AddressSanitizer: stack-use-after-scope on address 0x007fc1898b80 at pc 0x0060d36059b8 bp 0x007fc1898b50 sp 0x007fc1898b48  WRITE of size 4 at 0x007fc1898b80 thread T0  #0 0x60d36059b7 in test\_test vendor/sprd/proprietories-source/ylog/server/main.c:65  #1 0x60d36059b7 in main vendor/sprd/proprietories-source/ylog/server/main.c:165  #2 0x7d5ccedbd3 in \_\_libc\_init bionic/libc/bionic/libc\_init\_dynamic.cpp:129  Address 0x007fc1898b80 is located in stack of thread T0  at offset 32 in frame  #3 0x60d3605463 in main vendor/sprd/proprietories-source/ylog/server/main.c:72  This frame has 5 object(s):  [32, 36) 'x.i' (line 62) <== Memory access at offset 32 is inside this variable  [48, 140) 'enable' (line 74)  [176, 268) 'debuggable' (line 75)  [304, 560) 'service\_para' (line 77)  [624, 716) 'bootmode' (line 78)  HINT: this may be a false positive if your program uses some custom stack unwind mechanism or swapcontext  (longjmp and C++ exceptions \*are\* supported)  SUMMARY: AddressSanitizer: stack-use-after-scope (/system/bin/ylog+0x319b7)  Shadow bytes around the buggy address:  0x001ff8313120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0x001ff8313130: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0x001ff8313140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0x001ff8313150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0x001ff8313160: 00 00 00 00 00 00 00 00 00 00 00 00 f1 f1 f1 f1  =>0x001ff8313170:[f8]f2 00 00 00 00 00 00 00 00 00 00 00 04 f2 f2  0x001ff8313180: f2 f2 00 00 00 00 00 00 00 00 00 00 00 04 f2 f2  0x001ff8313190: f2 f2 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0x001ff83131a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  0x001ff83131b0: 00 00 f2 f2 f2 f2 f2 f2 f2 f2 00 00 00 00 00 00  0x001ff83131c0: 00 00 00 00 00 04 f3 f3 f3 f3 f3 f3 00 00 00 00  Shadow byte legend (one shadow byte represents 8 application bytes):  Addressable: 00  Partially addressable: 01 02 03 04 05 06 07  Heap left redzone: fa  Freed heap region: fd  Stack left redzone: f1  Stack mid redzone: f2  Stack right redzone: f3  Stack after return: f5  Stack use after scope: f8  Global redzone: f9  Global init order: f6  Poisoned by user: f7  Container overflow: fc  Array cookie: ac  Intra object redzone: bb  ASan internal: fe  Left alloca redzone: ca  Right alloca redzone: cb  ==4642==ABORTING |

## unknown address

相关支持类型的说明中，没有提到给未分配内存的指针赋值的情况，我们验证一下是否可以检测出来。

### 问题代码

为验证AddressSanitizer是否可以检测到该类型的问题，我们加入下面的代码：

sprdroid9.0\_trunk\_18c/vendor/sprd/proprietories-source/ylog/server/main.c

|  |
| --- |
| +int test\_test(int ii) {  +  + char \*p;  + char \*e = malloc(sizeof(char) \* 10);  + if (e == NULL) {  + return 0;  + }  + \*e = 'm';  + ylog\_info("ylog zzg == [%d]", ii);  +  + \*p = 'm'; // p 没有申请分配内存  +  + free(e);  +  + return 0;  +  +}  int main(int argc, char \*argv[]) {  ...  int ret = start\_cmd\_srv();  if(-1 == ret) {  return -1;  }  + int ii=0;  + ii=test\_test(25);  + ylog\_info("ylog zzg == [%d]", ii);  start\_active\_log(); |

### AddressSanitizer检测到问题

我们将前面的问题代码合入后，将编译生成库替换到手机中后重启运行，AddressSanitizer检测出相关问题，并给出以下问题栈。

|  |
| --- |
| 04-11 22:58:28.450 303 303 I ylog : =================================================================  04-11 22:58:28.451 303 303 I ylog : ==303==ERROR: AddressSanitizer: SEGV on unknown address 0x000000000000 (pc 0x00590e6b4524 bp 0x007feeea68b0 sp 0x007feeea6860 T0)  04-11 22:58:28.451 303 303 I ylog :  04-11 22:58:28.452 303 303 I ylog : ==303==The signal is caused by a WRITE memory access.  04-11 22:58:28.452 303 303 I ylog :  04-11 22:58:28.452 303 303 I ylog : ==303==Hint: address points to the zero page.  04-11 22:58:28.453 303 303 I ylog :  04-11 22:58:28.475 303 303 I ylog : #0 0x590e6b4523 (/system/bin/ylog+0x6f523)  04-11 22:58:28.475 303 303 I ylog :  04-11 22:58:28.475 303 303 I ylog : #1 0x590e6b4cdb (/system/bin/ylog+0x6fcdb)  04-11 22:58:28.476 303 303 I ylog :  04-11 22:58:28.476 303 303 I ylog : #2 0x72b656cbd3 (/system/lib64/libc.so+0xacbd3)  04-11 22:58:28.476 303 303 I ylog :  04-11 22:58:28.477 303 303 I ylog :  04-11 22:58:28.477 303 303 I ylog : AddressSanitizer can not provide additional info.  04-11 22:58:28.478 303 303 I ylog :  04-11 22:58:28.478 303 303 I ylog : SUMMARY: AddressSanitizer: SEGV (/system/bin/ylog+0x6f523)  04-11 22:58:28.478 303 303 I ylog :  04-11 22:58:28.479 303 303 I ylog : ==303==ABORTING |

### 栈解析

|  |
| --- |
| =================================================================  ==303==ERROR: AddressSanitizer: SEGV on unknown address 0x000000000000 (pc 0x00590e6b4524 bp 0x007feeea68b0 sp 0x007feeea6860 T0)  ==303==The signal is caused by a WRITE memory access.  ==303==Hint: address points to the zero page.  #0 0x590e6b4523 in test\_test vendor/sprd/proprietories-source/ylog/server/main.c:68  #1 0x590e6b4cdb in main vendor/sprd/proprietories-source/ylog/server/main.c:181  #2 0x72b656cbd3 in \_\_libc\_init bionic/libc/bionic/libc\_init\_dynamic.cpp:129  AddressSanitizer can not provide additional info.  SUMMARY: AddressSanitizer: SEGV (/system/bin/ylog+0x6f523)  ==303==ABORTING |

# ASan误报

本章记录我们遇见的AddressSanitizer的误报情况：

## 调试libandroid\_runtime.so 遇到container-overflow

我们在打开/system/lib/libandroid\_runtime.so调试的过程中遇到container-overflow误报

### 栈信息

#### AddressSanitizer 检测到ERROR

|  |
| --- |
| 08-06 03:33:55.228 24469 24469 I app\_process: type=1400 audit(0.0:5114): avc: denied { execute } for path="/data/asan/system/lib/libhwui.so" dev="dm-5" ino=6172 scontext=u:r:zygote:s0 tcontext=u:object\_r:system\_data\_file:s0 tclass=file permissive=1  08-06 03:33:55.586 24469 24469 I app\_process: =================================================================  08-06 03:33:55.587 24469 24469 I app\_process: ==24469==ERROR: AddressSanitizer: container-overflow on address 0xadc02874 at pc 0xb27ba070 bp 0xbe889ac0 sp 0xbe889698  08-06 03:33:55.588 24469 24469 I app\_process: READ of size 12 at 0xadc02874 thread T0  08-06 03:33:55.643 24469 24469 I app\_process: #0 0xb27ba06c (/system/lib/libclang\_rt.asan-arm-android.so+0x9f06c)  08-06 03:33:55.643 24469 24469 I app\_process: #1 0xb235e8fa (/data/asan/system/lib/libandroid\_runtime.so+0x18d8fa)  08-06 03:33:55.644 24469 24469 I app\_process: #2 0xb235e662 (/data/asan/system/lib/libandroid\_runtime.so+0x18d662)  08-06 03:33:55.645 24469 24469 I app\_process: #3 0xb176d81c (/data/asan/system/lib/libcodec2\_vndk.so+0x6581c)  08-06 03:33:55.646 24469 24469 I app\_process: #4 0xb173d4fe (/data/asan/system/lib/libcodec2\_vndk.so+0x354fe)  08-06 03:33:55.647 24469 24469 I app\_process: #5 0xb173d9d6 (/data/asan/system/lib/libcodec2\_vndk.so+0x359d6)  08-06 03:33:55.648 24469 24469 I app\_process: #6 0xb1740d70 (/data/asan/system/lib/libcodec2\_vndk.so+0x38d70)  08-06 03:33:55.648 24469 24469 I app\_process:  08-06 03:33:55.649 24469 24469 I app\_process: 0xadc02874 is located 36 bytes inside of 48-byte region [0xadc02850,0xadc02880)  08-06 03:33:55.650 24469 24469 I app\_process: allocated by thread T0 here:  08-06 03:33:55.651 24469 24469 I app\_process: #0 0xb27c7f04 (/system/lib/libclang\_rt.asan-arm-android.so+0xacf04)  08-06 03:33:55.651 24469 24469 I app\_process: #1 0xb235e5e2 (/data/asan/system/lib/libandroid\_runtime.so+0x18d5e2)  08-06 03:33:55.652 24469 24469 I app\_process:  08-06 03:33:55.653 24469 24469 I app\_process: HINT: if you don't care about these errors you may set ASAN\_OPTIONS=detect\_container\_overflow=0.  08-06 03:33:55.653 24469 24469 I app\_process: If you suspect a false positive see also: https://github.com/google/sanitizers/wiki/AddressSanitizerContainerOverflow.  08-06 03:33:55.654 24469 24469 I app\_process: SUMMARY: AddressSanitizer: container-overflow (/system/lib/libclang\_rt.asan-arm-android.so+0x9f06c)  08-06 03:33:55.656 24469 24469 I app\_process: Shadow bytes around the buggy address:  08-06 03:33:55.656 24469 24469 I app\_process: 0xa4d184b0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa  08-06 03:33:55.656 24469 24469 I app\_process: 0xa4d184c0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa  08-06 03:33:55.657 24469 24469 I app\_process: 0xa4d184d0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa  08-06 03:33:55.657 24469 24469 I app\_process: 0xa4d184e0: fa fa fa fa fa fa fa fa fa fa 00 00 00 00 00 00  08-06 03:33:55.657 24469 24469 I app\_process: 0xa4d184f0: fa fa 00 00 00 00 00 00 fa fa 00 00 00 00 00 00  08-06 03:33:55.658 24469 24469 I app\_process: =>0xa4d18500: fa fa fd fd fd fd fd fd fa fa 00 00 00 00[04]fc  08-06 03:33:55.658 24469 24469 I app\_process: 0xa4d18510: fa fa 00 00 00 00 00 00 fa fa 00 00 00 00 00 00  08-06 03:33:55.659 24469 24469 I app\_process: 0xa4d18520: fa fa fd fd fd fd fd fd fa fa fd fd fd fd fd fd  08-06 03:33:55.659 24469 24469 I app\_process: 0xa4d18530: fa fa fd fd fd fd fd fd fa fa fd fd fd fd fd fd  08-06 03:33:55.659 24469 24469 I app\_process: 0xa4d18540: fa fa 00 00 00 00 00 00 fa fa 00 00 00 00 00 00  08-06 03:33:55.659 24469 24469 I app\_process: 0xa4d18550: fa fa 00 00 00 00 00 00 fa fa 00 00 00 00 00 00  08-06 03:33:55.660 24469 24469 I app\_process: Shadow byte legend (one shadow byte represents 8 application bytes):  08-06 03:33:55.660 24469 24469 I app\_process: Addressable: 00  08-06 03:33:55.661 24469 24469 I app\_process: Partially addressable: 01 02 03 04 05 06 07  08-06 03:33:55.661 24469 24469 I app\_process: Heap left redzone: fa  08-06 03:33:55.661 24469 24469 I app\_process: Freed heap region: fd  08-06 03:33:55.662 24469 24469 I app\_process: Stack left redzone: f1  08-06 03:33:55.662 24469 24469 I app\_process: Stack mid redzone: f2  08-06 03:33:55.663 24469 24469 I app\_process: Stack right redzone: f3  08-06 03:33:55.663 24469 24469 I app\_process: Stack after return: f5  08-06 03:33:55.663 24469 24469 I app\_process: Stack use after scope: f8  08-06 03:33:55.664 24469 24469 I app\_process: Global redzone: f9  08-06 03:33:55.664 24469 24469 I app\_process: Global init order: f6  08-06 03:33:55.665 24469 24469 I app\_process: Poisoned by user: f7  08-06 03:33:55.665 24469 24469 I app\_process: Container overflow: fc  08-06 03:33:55.665 24469 24469 I app\_process: Array cookie: ac  08-06 03:33:55.666 24469 24469 I app\_process: Intra object redzone: bb  08-06 03:33:55.666 24469 24469 I app\_process: ASan internal: fe  08-06 03:33:55.666 24469 24469 I app\_process: Left alloca redzone: ca  08-06 03:33:55.667 24469 24469 I app\_process: Right alloca redzone: cb  08-06 03:33:55.667 24469 24469 I app\_process: Shadow gap: cc  08-06 03:33:55.668 24469 24469 I app\_process: ==24469==ABORTING |

#### Native crash 栈

|  |
| --- |
| 08-06 03:33:55.854 24486 24486 F DEBUG : \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\*  08-06 03:33:55.855 24486 24486 F DEBUG : Native Crash TIME: 3116463  08-06 03:33:55.855 24486 24486 F DEBUG : \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\*  08-06 03:33:55.855 24486 24486 F DEBUG : Build fingerprint: 'UNISOC/sp9832e\_1h10\_gofu/sp9832e\_1h10\_go:11/RP1A.200709.001/32426:userdebug/test-keys'  08-06 03:33:55.856 24486 24486 F DEBUG : Revision: '0'  08-06 03:33:55.856 24486 24486 F DEBUG : ABI: 'arm'  08-06 03:33:55.858 24486 24486 F DEBUG : Timestamp: 2020-08-06 03:33:55+0800  08-06 03:33:55.858 24486 24486 F DEBUG : pid: 24469, tid: 24469, name: app\_process >>> /system/bin/app\_process <<<  08-06 03:33:55.858 24486 24486 F DEBUG : uid: 0  08-06 03:33:55.859 24486 24486 F DEBUG : signal 6 (SIGABRT), code -1 (SI\_QUEUE), fault addr --------  08-06 03:33:55.859 24486 24486 F DEBUG : Abort message: '=================================================================  08-06 03:33:55.859 24486 24486 F DEBUG : ==24469==ERROR: AddressSanitizer: container-overflow on address 0xadc02874 at pc 0xb27ba070 bp 0xbe889ac0 sp 0xbe889698  …  08-06 03:33:55.859 24486 24486 F DEBUG : '  08-06 03:33:55.859 24486 24486 F DEBUG : r0 00000000 r1 00005f95 r2 00000006 r3 be888b38  08-06 03:33:55.860 24486 24486 F DEBUG : r4 be888b4c r5 be888b30 r6 00005f95 r7 0000016b  08-06 03:33:55.860 24486 24486 F DEBUG : r8 be888b38 r9 be888b48 r10 be888b68 r11 be888b58  08-06 03:33:55.860 24486 24486 F DEBUG : ip 00005f95 sp be888b08 lr b0b3664d pc b0b36660  08-06 03:33:55.933 24486 24486 F DEBUG : backtrace:  08-06 03:33:55.933 24486 24486 F DEBUG : #00 pc 00062660 /apex/com.android.runtime/lib/bionic/libc.so (abort+172) (BuildId: 839a29bd5b7284a5aa2054a464f499e4)  08-06 03:33:55.933 24486 24486 F DEBUG : #01 pc 0003956c /system/lib/libclang\_rt.asan-arm-android.so (\_\_sanitizer::Abort()+68) (BuildId: dc587666eacd829dfabfe29f89bfb506d0b28dc7)  08-06 03:33:55.934 24486 24486 F DEBUG : #02 pc 00038394 /system/lib/libclang\_rt.asan-arm-android.so (\_\_sanitizer::Die()+192) (BuildId: dc587666eacd829dfabfe29f89bfb506d0b28dc7)  08-06 03:33:55.935 24486 24486 F DEBUG : #03 pc 000a4388 /system/lib/libclang\_rt.asan-arm-android.so (\_\_asan::ScopedInErrorReport::~ScopedInErrorReport()+428) (BuildId: dc587666eacd829dfabfe29f89bfb506d0b28dc7)  08-06 03:33:55.935 24486 24486 F DEBUG : #04 pc 000a65a8 /system/lib/libclang\_rt.asan-arm-android.so (\_\_asan::ReportGenericError(unsigned long, unsigned long, unsigned long, unsigned long, bool, unsigned long, unsigned int, bool)+2868) (BuildId: dc587666eacd829dfabfe29f89bfb506d0b28dc7)  08-06 03:33:55.935 24486 24486 F DEBUG : #05 pc 0009f08c /system/lib/libclang\_rt.asan-arm-android.so (\_\_asan\_memcpy+328) (BuildId: dc587666eacd829dfabfe29f89bfb506d0b28dc7)  08-06 03:33:55.936 24486 24486 F DEBUG : #06 pc 0018d8fb /data/asan/system/lib/libandroid\_runtime.so (std::\_\_1::vector<std::\_\_1::basic\_string<char, std::\_\_1::char\_traits<char>, std::\_\_1::allocator<char> >, std::\_\_1::allocator<std::\_\_1::basic\_string<char, std::\_\_1::char\_traits<char>, std::\_\_1::allocator<char> > > >::\_\_swap\_out\_circular\_buffer(std::\_\_1::\_\_split\_buffer<std::\_\_1::basic\_string<char, std::\_\_1::char\_traits<char>, std::\_\_1::allocator<char> >, std::\_\_1::allocator<std::\_\_1::basic\_string<char, std::\_\_1::char\_traits<char>, std::\_\_1::allocator<char> > >&>&)+142) (BuildId: 59a20443e83bf5c3db715e6c7054ad22)  08-06 03:33:55.937 24486 24486 F DEBUG : #07 pc 0018d663 /data/asan/system/lib/libandroid\_runtime.so (void std::\_\_1::vector<std::\_\_1::basic\_string<char, std::\_\_1::char\_traits<char>, std::\_\_1::allocator<char> >, std::\_\_1::allocator<std::\_\_1::basic\_string<char, std::\_\_1::char\_traits<char>, std::\_\_1::allocator<char> > > >::\_\_push\_back\_slow\_path<std::\_\_1::basic\_string<char, std::\_\_1::char\_traits<char>, std::\_\_1::allocator<char> > >(std::\_\_1::basic\_string<char, std::\_\_1::char\_traits<char>, std::\_\_1::allocator<char> >&&)+486) (BuildId: 59a20443e83bf5c3db715e6c7054ad22)  08-06 03:33:55.937 24486 24486 F DEBUG : #08 pc 0006581d /data/asan/system/lib/libcodec2\_vndk.so (\_C2EnumUtils::sanitizeEnumValueNames(std::\_\_1::vector<char const\*, std::\_\_1::allocator<char const\*> >, char const\*)+344) (BuildId: 357a9cc91ffee3e17731a0f35aed7441)  08-06 03:33:55.938 24486 24486 F DEBUG : #09 pc 000354ff /data/asan/system/lib/libcodec2\_vndk.so (std::\_\_1::vector<std::\_\_1::pair<std::\_\_1::basic\_string<char, std::\_\_1::char\_traits<char>, std::\_\_1::allocator<char> >, C2Value::Primitive>, std::\_\_1::allocator<std::\_\_1::pair<std::\_\_1::basic\_string<char, std::\_\_1::char\_traits<char>, std::\_\_1::allocator<char> >, C2Value::Primitive> > > \_C2EnumUtils::sanitizeEnumValues<C2Value::Primitive>(std::\_\_1::vector<C2Value::Primitive, std::\_\_1::allocator<C2Value::Primitive> >, std::\_\_1::vector<char const\*, std::\_\_1::allocator<char const\*> >, char const\*)+106) (BuildId: 357a9cc91ffee3e17731a0f35aed7441)  08-06 03:33:55.939 24486 24486 F DEBUG : #10 pc 000359d7 /data/asan/system/lib/libcodec2\_vndk.so (std::\_\_1::vector<std::\_\_1::pair<std::\_\_1::basic\_string<char, std::\_\_1::char\_traits<char>, std::\_\_1::allocator<char> >, C2Value::Primitive>, std::\_\_1::allocator<std::\_\_1::pair<std::\_\_1::basic\_string<char, std::\_\_1::char\_traits<char>, std::\_\_1::allocator<char> >, C2Value::Primitive> > > C2FieldDescriptor::namedValuesFor<C2Config::supplemental\_info\_t>(C2Config::supplemental\_info\_t const&)+122) (BuildId: 357a9cc91ffee3e17731a0f35aed7441)  08-06 03:33:55.939 24486 24486 F DEBUG : #11 pc 00038d71 /data/asan/system/lib/libcodec2\_vndk.so (\_GLOBAL\_\_sub\_I\_C2Config.cpp+912) (BuildId: 357a9cc91ffee3e17731a0f35aed7441)  08-06 03:33:55.939 24486 24486 F DEBUG : #12 pc 0002cb45 /apex/com.android.runtime/bin/linker (\_\_dl\_\_ZL10call\_arrayIPFviPPcS1\_EEvPKcPT\_jbS5\_+204) (BuildId: 23965ba71b370e025e6b846c6325beb3)  08-06 03:33:55.939 24486 24486 F DEBUG : #13 pc 40000541 <unknown>  08-06 03:33:56.510 24469 24469 I DEBUG : Crash thread dumpable |

#### 调用栈解析

|  |
| --- |
| =================================================================  ==12504==ERROR: AddressSanitizer: container-overflow on address 0xa9002874 at pc 0xac3b6070 bp 0xbef23ac0 sp 0xbef23698  READ of size 12 at 0xa9002874 thread T0  #0 0xac3b606c in \_\_asan\_memcpy /out/llvm-project/compiler-rt/lib/asan/asan\_interceptors\_memintrinsics.cpp:22  #1 0xaf4d68fa in basic\_string external/libcxx/include/string:1877  #2 0xaf4d68fa in \_ZNSt3\_\_19allocatorINS\_12basic\_stringIcNS\_11char\_traitsIcEENS0\_IcEEEEE9constructIS5\_JS5\_EEEvPT\_DpOT0\_ external/libcxx/include/memory:1826  #3 0xaf4d68fa in \_ZNSt3\_\_116allocator\_traitsINS\_9allocatorINS\_12basic\_stringIcNS\_11char\_traitsIcEENS1\_IcEEEEEEE11\_\_constructIS6\_JS6\_EEEvNS\_17integral\_constantIbLb1EEERS7\_PT\_DpOT0\_ external/libcxx/include/memory:1718  #4 0xaf4d68fa in \_ZNSt3\_\_116allocator\_traitsINS\_9allocatorINS\_12basic\_stringIcNS\_11char\_traitsIcEENS1\_IcEEEEEEE9constructIS6\_JS6\_EEEvRS7\_PT\_DpOT0\_ external/libcxx/include/memory:1561  #5 0xaf4d68fa in \_ZNSt3\_\_116allocator\_traitsINS\_9allocatorINS\_12basic\_stringIcNS\_11char\_traitsIcEENS1\_IcEEEEEEE20\_\_construct\_backwardIPS6\_EEvRS7\_T\_SC\_RSC\_ external/libcxx/include/memory:1680    #6 0xaf4d68fa in \_ZNSt3\_\_16vectorINS\_12basic\_stringIcNS\_11char\_traitsIcEENS\_9allocatorIcEEEENS4\_IS6\_EEE26\_\_swap\_out\_circular\_bufferERNS\_14\_\_split\_bufferIS6\_RS7\_EE external/libcxx/include/vector:938  #7 0xaf4d6662 in \_ZNSt3\_\_16vectorINS\_12basic\_stringIcNS\_11char\_traitsIcEENS\_9allocatorIcEEEENS4\_IS6\_EEE21\_\_push\_back\_slow\_pathIS6\_EEvOT\_ external/libcxx/include/vector:1622  #8 0xaa8ed81c in \_ZNSt3\_\_16vectorINS\_12basic\_stringIcNS\_11char\_traitsIcEENS\_9allocatorIcEEEENS4\_IS6\_EEE9push\_backEOS6\_ external/libcxx/include/vector:1659    #9 0xaa8ed81c in \_ZN12\_C2EnumUtils22sanitizeEnumValueNamesENSt3\_\_16vectorIPKcNS0\_9allocatorIS3\_EEEES3\_ frameworks/av/media/codec2/vndk/util/C2ParamUtils.cpp:168    #10 0xaa8bd4fe in \_ZN12\_C2EnumUtils18sanitizeEnumValuesIN7C2Value9PrimitiveEEENSt3\_\_16vectorINS3\_4pairINS3\_12basic\_stringIcNS3\_11char\_traitsIcEENS3\_9allocatorIcEEEES2\_EENS9\_ISC\_EEEENS4\_IT\_NS9\_ISF\_EEEENS4\_IPKcNS9\_ISJ\_EEEESJ\_ frameworks/av/media/codec2/core/include/C2Enum.h:80  #11 0xaa8bd9d6 in \_ZN17C2FieldDescriptor14namedValuesForIN8C2Config19supplemental\_info\_tEEENSt3\_\_16vectorINS3\_4pairINS3\_12basic\_stringIcNS3\_11char\_traitsIcEENS3\_9allocatorIcEEEEN7C2Value9PrimitiveEEENS9\_ISE\_EEEERKT\_ frameworks/av/media/codec2/core/include/C2Config.h:722  #12 0xaa8c0d70 in \_ZN17C2FieldDescriptor18\_NamedValuesGetterIN8C2Config19supplemental\_info\_tELb1EE14getNamedValuesEv frameworks/av/media/codec2/core/include/C2Param.h:1117  #13 0xaa8c0d70 in C2FieldDescriptor<C2Config::supplemental\_info\_t, C2Config::supplemental\_info\_t> frameworks/av/media/codec2/core/include/C2Param.h:1026  #14 0xaa8c0d70 in \_\_cxx\_global\_var\_init.34 frameworks/av/media/codec2/core/include/C2Config.h:749  #15 0xaa8c0d70 in \_GLOBAL\_\_sub\_I\_C2Config.cpp frameworks/av/media/codec2/vndk/C2Config.cpp:?  0xa9002874 is located 36 bytes inside of 48-byte region [0xa9002850,0xa9002880)  allocated by thread T0 here:  #16 0xac3c3f04 in operator new /out/llvm-project/compiler-rt/lib/asan/asan\_new\_delete.cpp:99  #17 0xaf4d65e2 in \_ZNSt3\_\_117\_\_libcpp\_allocateEjj external/libcxx/include/new:239  #18 0xaf4d65e2 in \_ZNSt3\_\_19allocatorINS\_12basic\_stringIcNS\_11char\_traitsIcEENS0\_IcEEEEE8allocateEjPKv external/libcxx/include/memory:1814  #19 0xaf4d65e2 in \_ZNSt3\_\_116allocator\_traitsINS\_9allocatorINS\_12basic\_stringIcNS\_11char\_traitsIcEENS1\_IcEEEEEEE8allocateERS7\_j external/libcxx/include/memory:1547  #20 0xaf4d65e2 in \_\_split\_buffer external/libcxx/include/\_\_split\_buffer:311  #21 0xaf4d65e2 in \_ZNSt3\_\_16vectorINS\_12basic\_stringIcNS\_11char\_traitsIcEENS\_9allocatorIcEEEENS4\_IS6\_EEE21\_\_push\_back\_slow\_pathIS6\_EEvOT\_ external/libcxx/include/vector:1618  HINT: if you don't care about these errors you may set ASAN\_OPTIONS=detect\_container\_overflow=0.  If you suspect a false positive see also: https://github.com/google/sanitizers/wiki/AddressSanitizerContainerOverflow.  SUMMARY: AddressSanitizer: container-overflow (/system/lib/libclang\_rt.asan-arm-android.so+0x9f06c)  Shadow bytes around the buggy address:  0x9f8484b0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa  0x9f8484c0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa  0x9f8484d0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa  0x9f8484e0: fa fa fa fa fa fa fa fa fa fa 00 00 00 00 00 00  0x9f8484f0: fa fa 00 00 00 00 00 00 fa fa 00 00 00 00 00 00  =>0x9f848500: fa fa fd fd fd fd fd fd fa fa 00 00 00 00[04]fc  0x9f848510: fa fa 00 00 00 00 00 00 fa fa 00 00 00 00 00 00  0x9f848520: fa fa fd fd fd fd fd fd fa fa fd fd fd fd fd fd  0x9f848530: fa fa fd fd fd fd fd fd fa fa fd fd fd fd fd fd  0x9f848540: fa fa 00 00 00 00 00 00 fa fa 00 00 00 00 00 00  0x9f848550: fa fa 00 00 00 00 00 00 fa fa 00 00 00 00 00 00  Shadow byte legend (one shadow byte represents 8 application bytes):  Addressable: 00  Partially addressable: 01 02 03 04 05 06 07  Heap left redzone: fa  Freed heap region: fd  Stack left redzone: f1  Stack mid redzone: f2  Stack right redzone: f3  Stack after return: f5  Stack use after scope: f8  Global redzone: f9  Global init order: f6  Poisoned by user: f7  Container overflow: fc  Array cookie: ac  Intra object redzone: bb  ASan internal: fe  Left alloca redzone: ca  Right alloca redzone: cb  Shadow gap: cc  ==12504==ABORTING |

### 分析

调用栈看 在 sanitizedNames.push\_back 处理的name过程中 发现有container-overflow

|  |
| --- |
| #9 0xaa8ed81c in frameworks/av/media/codec2/vndk/util/C2ParamUtils.cpp:168  118 //static  119 std::vector<C2String> \_C2EnumUtils::sanitizeEnumValueNames(  120 const std::vector<C2StringLiteral> names,  121 C2StringLiteral \_prefix) {  122 std::vector<C2String> sanitizedNames;  123 C2String prefix;  124 size\_t extraUnderscores = 0;  125 bool first = true;  126 if (\_prefix) {  127 extraUnderscores = countLeadingUnderscores(\_prefix);  128 prefix = \_prefix + extraUnderscores;  129 first = false;  130 C2\_LOG(VERBOSE) << "prefix:" << prefix << ", underscores:" << extraUnderscores;  131 }  132  133 // calculate prefix and minimum leading underscores  134 for (C2StringLiteral s : names) {  135 C2\_LOG(VERBOSE) << s;  136 size\_t underscores = countLeadingUnderscores(s);  137 if (first) {  138 extraUnderscores = underscores;  139 prefix = s + underscores;  140 first = false;  141 } else {  142 size\_t matching = countMatching(  143 s + underscores,  144 prefix);  145 prefix.resize(matching);  146 extraUnderscores = std::min(underscores, extraUnderscores);  147 }  148 C2\_LOG(VERBOSE) << "prefix:" << prefix << ", underscores:" << extraUnderscores;  149 if (prefix.size() == 0 && extraUnderscores == 0) {  150 break;  151 }  152 }  153  154 // we swallow the first underscore after upper case prefixes  155 bool upperCasePrefix = true;  156 for (size\_t i = 0; i < prefix.size(); ++i) {  157 if (islower(prefix[i])) {  158 upperCasePrefix = false;  159 break;  160 }  161 }  162  163 for (C2StringLiteral s : names) {  164 size\_t underscores = countLeadingUnderscores(s);  165 C2String sanitized = C2String(s, underscores - extraUnderscores);  166 sanitized.append(s + prefix.size() + underscores + (upperCasePrefix && s[prefix.size() + underscores] == '\_'));  168 sanitizedNames.push\_back(camelCaseToDashed(sanitized)); <---------------------  169 }  170  171 for (C2String s : sanitizedNames) {  172 C2\_LOG(VERBOSE) << s;  173 }  174  175 return sanitizedNames;  176 } |

在 上面 push\_back 之前打印 name 发现，传入的names是 C2Config::supplemental\_info\_t

|  |
| --- |
| 01-01 08:46:34.056 18269 18269 D app\_process: zzg 1 INFO\_NONE = 0  01-01 08:46:34.056 18269 18269 D app\_process: zzg 1 INFO\_PREFIX\_SEI\_UNIT = 0x10000  01-01 08:46:34.056 18269 18269 D app\_process: zzg 1 INFO\_SUFFIX\_SEI\_UNIT = 0x20000  01-01 08:46:34.056 18269 18269 D app\_process: zzg 1 INFO\_SEI\_USER\_DATA = INFO\_PREFIX\_SEI\_UNIT | 4  01-01 08:46:34.056 18269 18269 D app\_process: zzg 1 INFO\_SEI\_MDCV = INFO\_PREFIX\_SEI\_UNIT | 137  01-01 08:46:34.057 18269 18269 D app\_process: zzg 1 INFO\_SET\_USER\_DATA\_SFX = INFO\_SUFFIX\_SEI\_UNIT | 4  01-01 08:46:34.057 18269 18269 D app\_process: zzg 1 INFO\_VENDOR\_START = 0x70000000  其处理到 INFO\_SEI\_MDCV 的时候出现问题  01-01 08:46:34.057 18269 18269 D app\_process: zzg 2 INFO\_NONE = 0  01-01 08:46:34.057 18269 18269 D app\_process: zzg 2 INFO\_PREFIX\_SEI\_UNIT = 0x10000  01-01 08:46:34.057 18269 18269 D app\_process: zzg 2 INFO\_SUFFIX\_SEI\_UNIT = 0x20000  01-01 08:46:34.057 18269 18269 D app\_process: zzg 2 INFO\_SEI\_USER\_DATA = INFO\_PREFIX\_SEI\_UNIT | 4  01-01 08:46:34.057 18269 18269 D app\_process: zzg 2 INFO\_SEI\_MDCV = INFO\_PREFIX\_SEI\_UNIT | 137  #10 0xaa8bd4fe in frameworks/av/media/codec2/core/include/C2Enum.h:80  #11 0xaa8bd9d6 in frameworks/av/media/codec2/core/include/C2Config.h:722  714 /\*\*  715 \* Supplemental Data.  716 \*  717 \* This is coding-specific supplemental informational data, e.g. SEI for AVC/HEVC.  718 \* This structure is not a configuration so it does not have a parameter key.  719 \* This structure shall be returned in the configuration update, and can be repeated as needed  720 \* in the same update.  721 \*/  722 C2ENUM(C2Config::supplemental\_info\_t, uint32\_t, <-------------  723 INFO\_NONE = 0,  724  725 INFO\_PREFIX\_SEI\_UNIT = 0x10000, ///< prefix SEI payload types add this flag  726 INFO\_SUFFIX\_SEI\_UNIT = 0x20000, ///< suffix SEI payload types add this flag  727  728 INFO\_SEI\_USER\_DATA = INFO\_PREFIX\_SEI\_UNIT | 4, ///< closed-captioning data (ITU-T T35)  729 INFO\_SEI\_MDCV = INFO\_PREFIX\_SEI\_UNIT | 137, ///< mastering display color volume  730 INFO\_SET\_USER\_DATA\_SFX = INFO\_SUFFIX\_SEI\_UNIT | 4, ///< closed-captioning data (ITU-T T35)  731  732 INFO\_VENDOR\_START = 0x70000000  733 ) |

对比 androidQ 代码，external/libcxx/ 模块改动很小，该asan问题相关路径没有代码改动 该 supplemental\_info\_t 代码没有改动

实际测试中，sanitizedNames 设置 reserve 为 64 可以解决此问题，但会有报其他 container-overflow。

我们遇到的问题是 vactor 的 push\_back 添加过程这个过程，这个过程中c++ 容器 是自行扩展的，当容量不够时，会自行扩展添加。而asan container-overflow 不能正确区分扩展添加的情况。会有误报

因此需要设置 detect\_odr\_violation=0 关闭相关检查

代码中有注释明确说明有误报需要关闭此检测

|  |
| --- |
| http://10.0.1.79:8081/xref/sprdroidr\_trunk/system/core/init/main.cpp#33  31 #if \_\_has\_feature(address\_sanitizer)  32 // Load asan.options if it exists since these are not yet in the environment.  33 // Always ensure detect\_container\_overflow=0 as there are false positives with this check. // <----  34 // Always ensure abort\_on\_error=1 to ensure we reboot to bootloader for development builds.  35 extern "C" const char\* \_\_asan\_default\_options() {  36 return "include\_if\_exists=/system/asan.options:detect\_container\_overflow=0:abort\_on\_error=1";  37 } |

asan.options 的选项也是关闭 此项的

|  |
| --- |
| http://10.0.1.79:8081/xref/sprdroidr\_trunk/system/core/rootdir/asan.options#5  1 allow\_user\_segv\_handler=1  2 detect\_odr\_violation=0  3 alloc\_dealloc\_mismatch=0  4 allocator\_may\_return\_null=1  5 detect\_container\_overflow=0 <-----  6 abort\_on\_error=1 |

因此,此处asan 检测中的 container-overflow 是误报，关闭该检测类型。

### 解决

通过在修改下面rc文件 增加设置 ASAN\_OPTIONS detect\_container\_overflow=0

|  |
| --- |
| sprdroidr\_trunk/out/target/product/sp9832e\_1h10\_go$ cat ./system/etc/init/hw/init.zygote32.rc  service zygote /system/bin/app\_process -Xzygote /system/bin --zygote --start-system-server  class main  priority -20  user root  group root readproc reserved\_disk  setenv LD\_LIBRARY\_PATH /data/asan/system/lib:/system/lib  setenv ASAN\_OPTIONS allow\_user\_segv\_handler=true:detect\_container\_overflow=0  socket zygote stream 660 root system  socket usap\_pool\_primary stream 660 root system  onrestart exec\_background - system system -- /system/bin/vdc volume abort\_fuse  onrestart write /sys/power/state on  onrestart restart audioserver  onrestart restart cameraserver  onrestart restart media  onrestart restart netd  onrestart restart wificond  writepid /dev/cpuset/foreground/tasks |