# AddressSanitizer

目录

[AddressSanitizer 1](#_Toc495665146)

[AddressSanitizer vs Valgrind (Memcheck) 1](#_Toc495665147)

[构建应用 2](#_Toc495665148)

[ROOT设备 2](#_Toc495665149)

[配置设备 3](#_Toc495665150)

[Asan完整安装脚本 3](#_Toc495665151)

AddressSanitizer (ASan) 是一种基于编译器的快速检测工具，用于检测原生代码中的内存错误。本文档将介绍如何使用AddressSanitizer构建和运行Android平台的程序。该工具可以检测一下类型的错误：

⚫ 越界访问堆、栈和全局变量

⚫ 释放后使用

⚫ 释放后返回（运行时标记ASAN\_OPTIONS = detect\_stack\_use\_after\_return = 1）

⚫ 重复释放，无效释放

⚫ 内存溢出

由AddressSanitizer引入的典型的减速是2x。

在Android平台上，ASan从KitKat开始往后的所有版本都支持，除了Lollipop的初始版本（Lollipop MR1 更新也支持）。

如果想在android平台使用ASan, 设备需要是root 过的，可以是-eng或-userdebug build，或任何其他允许编辑/ system分区内容的设置。

AddressSanitizer vs Valgrind (Memcheck)

|  |  |  |
| --- | --- | --- |
|  | AddressSanitizer | Valgrind |
| Heap out-of-bounds | YES | YES |
| Stack out-of-bounds | YES | NO |
| Global out-of-bounds | YES | NO |
| Use-after-free | YES | YES |
| Use after scope | YES | NO |
| Use-after-return | Sometimes/YES | NO |
| Memory leaks | YES | YES |
| Uninitialized reads | NO | YES |
| Overhead | 1.5x-3x | 10x-30x |
| Platforms | Same as LLVM \* | Linux,Mac,Android |

构建应用

Android NDK从r10d开始支持AddressSanitizer，但是在r10d上不支持64位的ABIs，编译时在Application.mk架构配置使用：

|  |
| --- |
| APP\_ABI := armeabi armeabi-v7a x86 |

启用AddressSanitizer 需要在Android.mk中添加 –fsanitize=address编译选项，为了获得更好的堆栈跟踪，可以添加-fno-omit-frame-pointer和尾部呼叫消除（-fno-optimize-sibling-call）编译选项，要获得合理的性能，可以添加-O1或更高：

|  |
| --- |
| LOCAL\_CPPFLAGS += -fsanitize=address -O1 -fno-omit-frame-pointer -fno-optimize-sibling-call  LOCAL\_CFLAGS += -fsanitize=address -O1 -fno-omit-frame-pointer -fno-optimize-sibling-call  LOCAL\_LDFLAGS += -fsanitize=address -O1  LOCAL\_ARM\_MODE += arm |

执行 ndk-build 后，程序就已经获得了AddressSanitizer的支持，但是要想能够运行起来我们还需要做一些工作。首先确保手机root了，如果没有root可以参照下面的方法进行root，已经root过了可以跳过。

ROOT设备

首先下载[root精灵](http://www.shuame.com/root/) 或者其他的一些root工具，要获得更多的root工具，可以在[百度](https://www.baidu.com/)或者[google](http://www.google.com)中搜索Android Root。安装完工具后插上设备，开启设备的开发者选项以及打开USB调试，然后对设备进行ROOT，ROOT完毕后在命令行中执行 adb root,如果提示 adbd cannot run as root in production builds，那么说明设备还没有完全root，这时候我们可以安装【超级adbd】来进行root权限提升，在设备上安装完超级adbd后，再次运行adb root，如果提示adbd is already running as root那说明已经可以使用设备root权限。另外可以安装【Root Explorer】来进行文件操作。

配置设备

Root 完设备后，将 对应设备架构的asan运行时库拷贝到/system/lib下面，比如arm架构设备对应的asan运行时库libclang\_rt.asan-arm-android.so。

|  |
| --- |
| adb shell push /temp/libclang\_rt.asan-arm-android.so /system/lib/ |

拷贝完成后还需要设置该库的权限：

|  |
| --- |
| adb shell chmod 644 /system/lib/libclang\_rt.asan-arm-android.so |

运行程序之前，必须优先加载libclang\_rt.asan-arm-android.so，LD\_PRELOAD=libclang\_rt.asan-arm-android.so.

我们可以使用wrap属性来完成，首先编写 asanwrapper.sh，内容如下

|  |
| --- |
| #!/system/bin/sh  LD\_PRELOAD= libclang\_rt.asan-arm-android.so  exec $@ |

然后将 asanwrapper.sh 拷贝到/system/bin/目录下，并提升权限：

|  |
| --- |
| adb shell push /temp/ asanwrapper.sh /system/bin/  adb shell chmod 755 /system/bin/asanwrapper.sh |

接下来设置程序的wrap属性

|  |
| --- |
| adb shell setprop wram.com.example.test “asanwrapper.sh” |

然后就可以启动程序查看asan检查的结果了。

Asan完整安装脚本

|  |
| --- |
| #!/bin/bash  set -e  HERE="$(cd "$(dirname "$0")" && pwd)"  revert=no  extra\_options=  device=  lib=  package=  function usage {  echo "usage: $0 [--revert] [--device device-id] [--lib path] [--extra-options options] --package"  echo " --revert: Uninstall ASan from the device."  echo " --lib: Path to ASan runtime library."  echo " --extra-options: Extra ASAN\_OPTIONS."  echo " --package: The application package name."  echo " --device: Install to the given device. Use 'adb devices' to find"  echo " device-id."  echo  exit 1  }  function get\_device\_arch { # OUTVAR  local \_outvar=$1  local \_ABI=$($ADB shell getprop ro.product.cpu.abi)  local \_ARCH=  if [[ $\_ABI == x86\* ]]; then  \_ARCH=i686  elif [[ $\_ABI == armeabi\* ]]; then  \_ARCH=arm  else  echo "Unrecognized device ABI: $\_ABI"  exit 1  fi  eval $\_outvar=\$\_ARCH  }  while [[ $# > 0 ]]; do  case $1 in  --revert)  revert=yes  ;;  --extra-options)  shift  if [[ $# == 0 ]]; then  echo "--extra-options requires an argument."  exit 1  fi  extra\_options="$1"  ;;  --lib)  shift  if [[ $# == 0 ]]; then  echo "--lib requires an argument."  exit 1  fi  lib="$1"  ;;  --package)  shift  if [[ $# == 0 ]]; then  echo "--package requires an argument."  exit 1  fi  package="$1"  ;;  --device)  shift  if [[ $# == 0 ]]; then  echo "--device requires an argument."  exit 1  fi  device="$1"  ;;  \*)  usage  ;;  esac  shift  done  ADB=${ADB:-adb}  if [[ x$device != x ]]; then  ADB="$ADB -s $device"  fi  $ADB shell am force-stop $package  echo '>> Remounting /system rw'  $ADB root  $ADB wait-for-device  $ADB remount  $ADB wait-for-device  get\_device\_arch ARCH  echo "Target architecture: $ARCH"  ASAN\_RT="libclang\_rt.asan-$ARCH-android.so"  if [[ x$revert == xyes ]]; then  echo '>> Uninstalling ASan'  $ADB shell rm /system/bin/asanwrapper  $ADB shell rm /system/lib/$ASAN\_RT  $ADB shell setprop wrap.$package ""  echo '>> Restarting shell'  $ADB shell stop  $ADB shell start  echo '>> Done'  exit 0  fi  if [[ -d "$lib" ]]; then  ASAN\_RT\_PATH="$lib"  elif [[ -f "$lib" && "$lib" == \*"$ASAN\_RT" ]]; then  ASAN\_RT\_PATH=$(dirname "$lib")  elif [[ -f "$HERE/$ASAN\_RT" ]]; then  ASAN\_RT\_PATH="$HERE"  elif [[ $(basename "$HERE") == "bin" ]]; then  # We could be in the toolchain's base directory.  # Consider ../lib, ../lib/asan, ../lib/linux and ../lib/clang/$VERSION/lib/linux.  P=$(ls "$HERE"/../lib/"$ASAN\_RT" "$HERE"/../lib/asan/"$ASAN\_RT" "$HERE"/../lib/linux/"$ASAN\_RT" "$HERE"/../lib/clang/\*/lib/linux/"$ASAN\_RT" 2>/dev/null | sort | tail -1)  if [[ -n "$P" ]]; then  ASAN\_RT\_PATH="$(dirname "$P")"  fi  fi  if [[ -z "$ASAN\_RT\_PATH" || ! -f "$ASAN\_RT\_PATH/$ASAN\_RT" ]]; then  echo ">> ASan runtime library not found"  exit 1  fi  TMPDIRBASE=$(mktemp -d)  TMPDIR="$TMPDIRBASE/new"  mkdir $TMPDIR  echo "TMPDIRBASE: $TMPDIRBASE"  echo "TMPDIR: $TMPDIR"  cp "$ASAN\_RT\_PATH/$ASAN\_RT" "$TMPDIR/"  # FIXME: alloc\_dealloc\_mismatch=0 prevents a failure in libdvm startup,  # which may or may not be a real bug (probably not).  ASAN\_OPTIONS=alloc\_dealloc\_mismatch=0,start\_deactivated=1  # On Android-L not allowing user segv handler breaks some applications.  if $ADB shell 'echo $LD\_PRELOAD' | grep libsigchain.so >&/dev/null; then  ASAN\_OPTIONS="$ASAN\_OPTIONS,allow\_user\_segv\_handler=1"  fi  if [[ x$extra\_options != x ]] ; then  ASAN\_OPTIONS="$ASAN\_OPTIONS,$extra\_options"  fi  cat <<EOF >"$TMPDIR/asanwrapper"  #!/system/bin/sh  ASAN\_OPTIONS=$ASAN\_OPTIONS \\  LD\_PRELOAD=\$LD\_PRELOAD:$ASAN\_RT \\  exec \$@  EOF  echo '>> Pushing files to the device'  $ADB push "$TMPDIR/$ASAN\_RT" /system/lib/  $ADB push "$TMPDIR/asanwrapper" /system/bin/asanwrapper  $ADB shell chown root.shell /system/lib/"$ASAN\_RT" /system/bin/asanwrapper  $ADB shell chmod 644 /system/lib/"$ASAN\_RT"  $ADB shell chmod 755 /system/bin/asanwrapper  $ADB shell setenforce 0 #disable SELinux  $ADB shell setprop wrap.$package "asanwrapper"  echo "wrap.$package: $(adb shell getprop wrap.$package)"  # rm -r "$TMPDIRBASE"  # echo '>> Restarting shell (asynchronous)'  # echo '>> Please wait until the device restarts'  # $ADB shell stop  # $ADB shell start |

只需要将 asan运行时库拷贝到该脚本所在的目录，然后执行该脚本，

同时也可以参考

$NDK\_ROOT/toolchains/llvm-3.5/prebuilt/windows/bin/asan\_device\_setup

参数介绍：

--revert 删除设备上asan环境

--lib asan运行时库的目录，如果不指定，在从当前目录查找。

--extra-options asan配置项

--package 指定程序包名，必选项

--device 如果存在多个设备可以使用该选项指定设备。