# 安全编译选项

## 安全编译选项介绍

操作系统提供了许多安全机制来尝试降低或阻止缓冲区溢出攻击带来的安全风险，例如 NX ASLR PIE CANARY FORTIFY RELRO 等手段。该主题即为通过扫描源码编译生成的二进制包，分析不安全函数，不安全配置，疑似后门，不安全三方组件等风险，帮助安全测试人员了解产品整体的基础安全风险状况，为安全整改提供参考。

**典型安全编译选项如下：**

### 1.1 ND\_NOW 立即绑定

**编译选项**：-z,now

标记ELF程序在启动时就解析所有动态符号（而不是“按需形式”(延时绑定)，也称为“立即绑定”），以便可以将GOT完全设为只读（与上面的RELRO结合使用时）。

### NX 堆栈不可执行

**编译选项**：-Wl,-z noexecstack

NX选项作用是让堆栈上的数据不可被当做指令执行。因为利用缓冲区溢出的shellcode一般都是放在堆栈上，如果将堆栈设置不可执行，即使溢出成功跳转到shellcode，程序在尝试执行指令时CPU就会抛出异常，而不是去执行恶意指令，提升了攻击者的难度。

### PIC 地址无关

**编译选项**：-fPIC

fpic的目的是为了做到动态加载so文件，每次运行都通过全局偏移表(GOT)索引获取到so中的函数真实地址（其实就是间接寻址），所以每次so中的函数地址都不是固定的

### PIE 随机化

**编译选项**：-pie,-fpie(-fPIE)

fpie与fpic类似，它使本程序内的代码变得位置无关，让本程序内的函数地址变得随机。有了PIE的地址随机化后，无法利用本程序内的函数。

### RELRO GOT表保护

**编译选项**：-Wl,-z,relro,-z,now

RELRO（RELocation Read-Only） 选项顾名思义，正是用来防止恶意代码修改 GOT 表的。增加该选项后，程序在装载(Load) 而不是运行的时候就解析所有的动态库函数地址，并将这些地址填充到相应的 GOT 表中，然后当装载程序移交控制权时，将 GOT 表设置为只读，这样就保证了运行时 GOT 表不允许覆写。

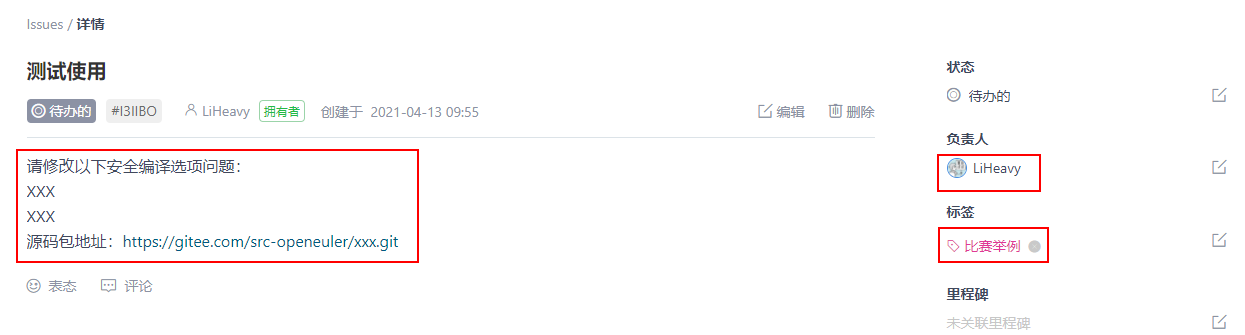
### SP 栈保护

**编译选项**：-fstack-protector-all/-fstack-protector-strong

启用栈保护后，函数执行前先往栈里插入校验信息（canary），在函数返回时进行验证。如果不正确就中止程序。因为校验信息的位置是随机的，攻击者很难猜到和避开，所以开启以后攻击难度非常高。

## 2、题目发布

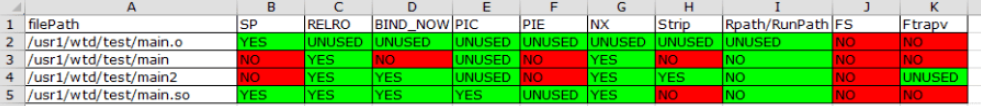
通过扫描iso镜像文件筛选出有问题的rpm包后，出题者将会在gitee（openEuler/src-openEuler）对应rpm的仓库中提交Issue，添加对应的比赛标签和issue责任人。在issue描述中，我们会提供有问题rpm包对应源码包的下载地址。



## 3、整改指导

### 3.1 整改流程

出题者将会在Issue中附加rpm包扫描结果截图，如下所示：



一般来说红色背景是不及格的，出题者会先筛选误报问题，但有一定可能仍存在误报问题，（编译脚本中已经有对应的安全编译选项，但是仍然扫描报错），误报问题可直接在Issue或者PR中通过评论说明。问题项请根据以下常见情况进行分析和修改。

**每项扫描结果具体含义如下所示：**



### 3.2 修改并验证

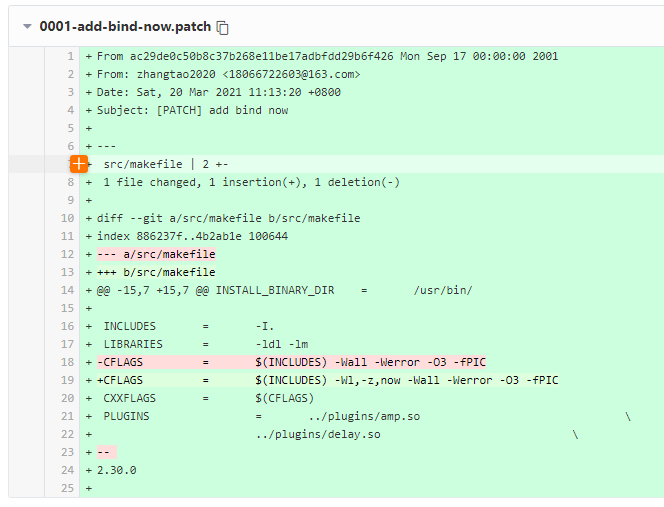
#### 3.2.1 修改案例

**(1) BIND\_NOW 立即绑定**

* **Issue描述**

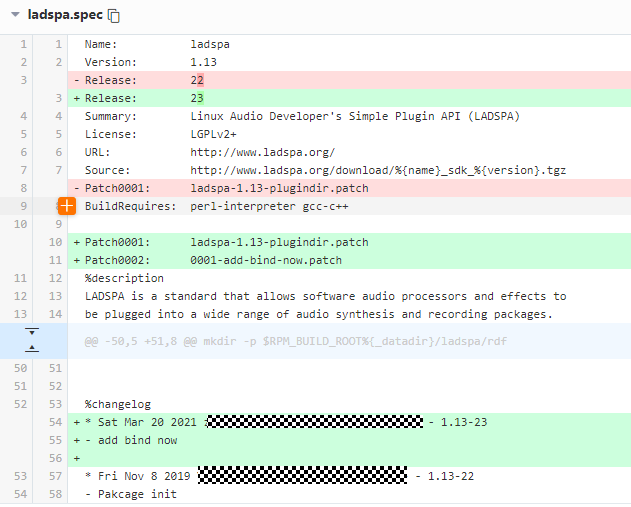


* **添加了-wl、-z、now，生成0001-add-bind-now.patch**



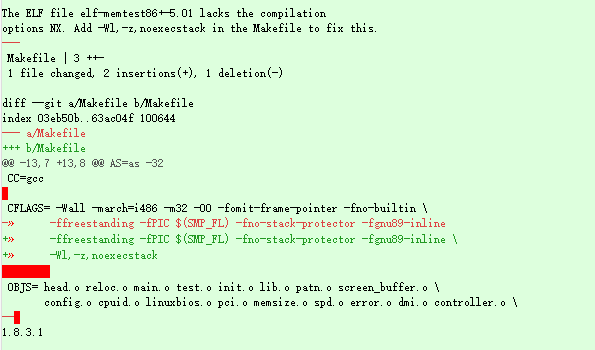
* **修改spec文件**

由于在11行增加了0001-add-bind-now.patch, 修改了spec文件，因此release需要+1，同时，增加一条changelog。



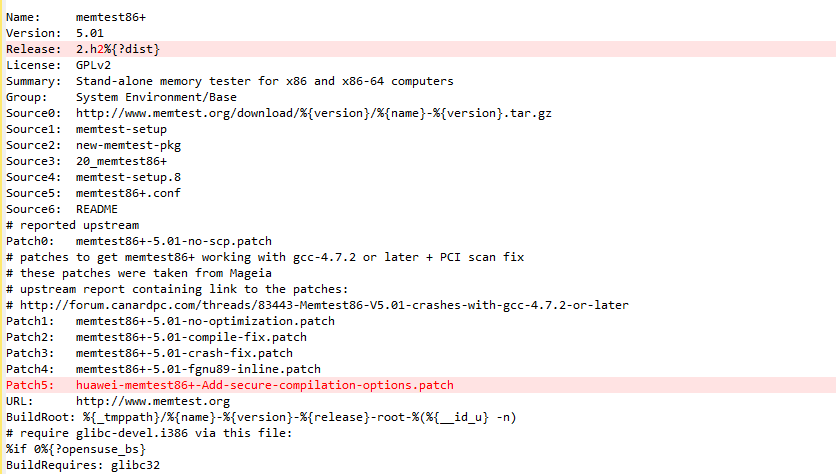
**(2) NX 堆栈不可执行**

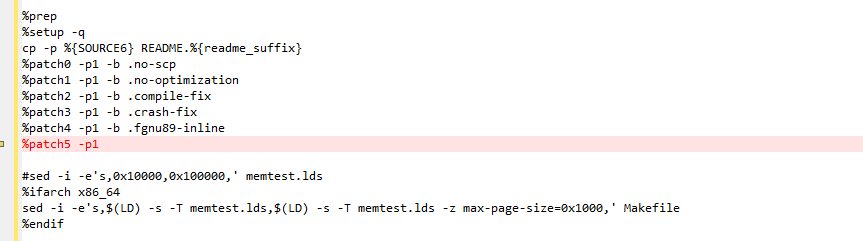
* **添加-W1, -z, noexecstack， 生成Add-secure-compilation-options.patch**

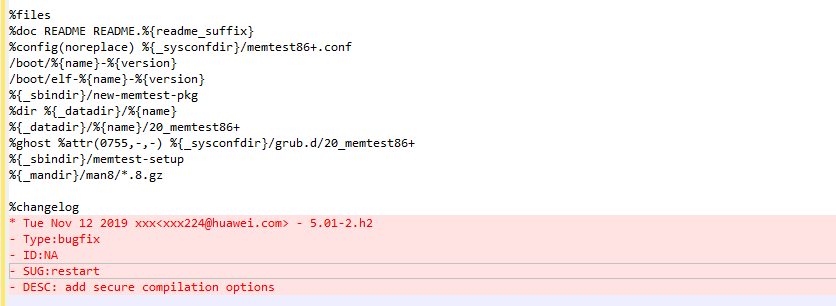


* **修改spec文件**

**添加patch5, release+1, 添加-p1参数。**







**（3）PIC地址无关 、 PIE随机化**

**具体Issue如下：**



**添加CFLAGS="-fPIE $RPM\_OPT\_FLAGS"，release+1，增加一条changelog。**

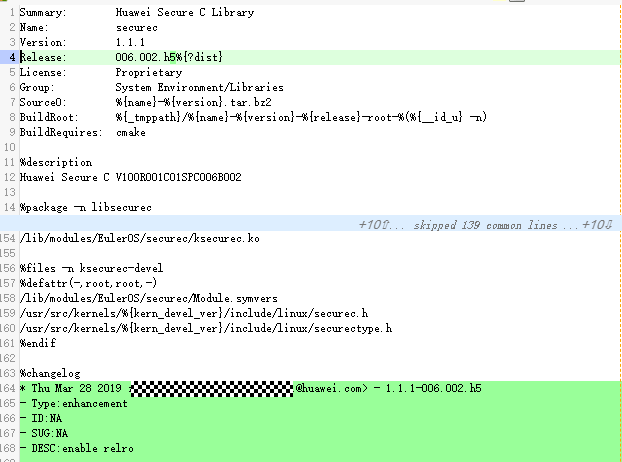


**（4）RELRO GOT表保护**

* **添加-Wl,-z,relro,-z,now,-z,noexecstack**

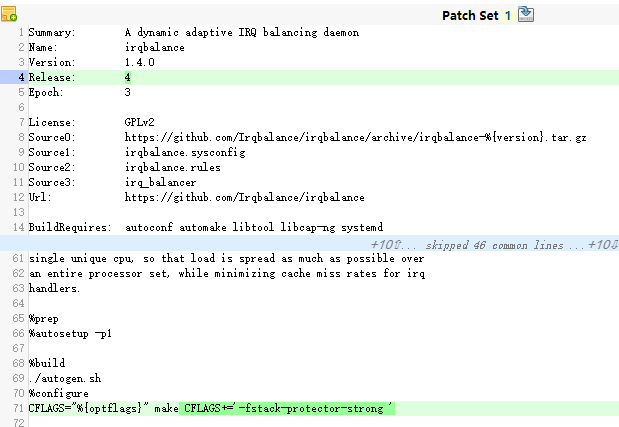


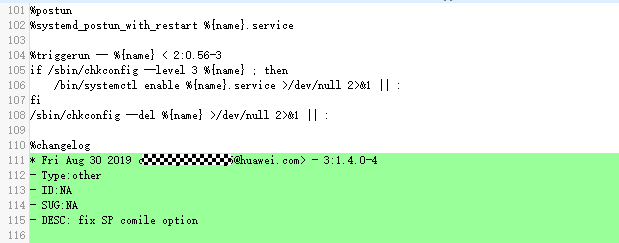
* **修改spec文件，release+1，增加一条changelog**



**（5）SP 栈保护**

**增加CFLAGS+='-fstack-protector-strong'，release+1,添加一条changelog。**





## 4、验证流程

当前没有提供给开发者使用的安全编译选项扫描工具，所以验证工作由Issue责任人代为操作：

* 开发者修改完成后提交PR，PR提交流程请参考：

<https://gitee.com/openeuler/community/blob/master/zh/contributors/pull-request.md>

PR中请指定审核人为领取Issue的责任人。

* 提交PR后，Issue责任人会根据PR中构建的rpm包进行验证，由于Issue责任人负责事项较多，所以每天16:00-17:00会统一进行验证。
* Issue责任人验证完成后会将验证结果反馈到PR评论中，请及时关注PR状态。

若验证成功：PR会合入对应仓库，关联Issue将会关闭，开发者得到奖励分。

若验证失败：PR评论中会有验证失败的截图和说明，开发者可继续修改。



* 若18:00之后Issue责任人仍未回复，请根据任务领取处责任人邮箱发送邮件提醒。

## 5、FQA

#### 5.1 社区常见疑问及解答

**（1）如何确认软件包的sig组及maintainer**

 **确认sig组**

在<https://gitee.com/openeuler/community/blob/master/sig/sigs.yaml>链接中，先找到软件包名，然后往上找到对应的-name，即为该软件包的sig组。 例如：包名为pkgship的软件包，EasyLife为该软件包的sig组。



 **确认maintainer**

从<https://gitee.com/openeuler/community/tree/master/sig>找到已经确认的sig组，例如sig-EasyLife。



打开该sig组的README.md文档，可查看maintainer列表。



**（2） 如何联系maintainer**

开发者打开gitee中提交PR的网页，在评论区艾特该软件包对应的sig组maintainer，即可联系。

**（3） 如何申诉**

发送邮件到以下任意一个邮箱，联系负责人，说明领取的任务以及申诉的具体原因，必要时可附上截图或word文档。

邮箱：

xiasenlin1@huawei.com

lihaiwei8@huawei.com

[tushenmei@huawei.com](mailto:tushenmei@huawei.com)

#### 5.2 安全编译选项常见疑问及解答

1. **添加了 -Wl,-z,relro,-z,now后RELRO或BIND\_NOW扫描仍然为NO**
   * 请从编译时的打印日志（编译日志可在PR门禁中查看）确认**目标文件(扫描的文件)**的**链接阶段**是否有 -Wl,-z,relro,-z,now 命令，如果没有，请添加。以编译日志的为准
   * 请确认编译命令里是否有 -static ，如果有，需要了解是否必须静态链接使用static选项。如果不需要-static的话，改为动态链接方式，将static选项去掉。
   * 请确认是否有 -T 链接脚本命令，如果有，去掉试下。
2. **堆栈保护(SP)扫描为NO，应该怎样整改？**

* 请从编译时打印日志确认是否有相对应的编译命令，如果没有，请添加
* 网页搜索关于 -fstack-protector 、-fstack-protector-all 、-fstack-protector-strong的命令作用范围。
* 是否激活了SP，并不能确认是用了-fstack-protector、-fstack-protector-all 还是-fstack-protector-strong，**SP应该根据编译日志和扫描结果确认是否合乎编译安全选项**

**（3）编译时添加了-fstack-protector ,但为什么SP显示为NO**

* 如果源码的函数没有局部字符串数组，编译时添加-fstack-protector，SP显示可能为NO
* 如果有 -static -s 命令编译时会删除符号表，同样也不会检测到canary word，SP显示为NO
* \_\_stack\_chk\_fail是在函数返回之前进行插桩，如果函数没有得到返回，则不会插桩，例如：死循环、return之前有exit等；

**（4）编译时添加了-fstack-protector-all ,但为什么SP显示为NO**

* 请从编译时的打印日志确认**目标文件(扫描的文件)**的**编译阶段**是否有​-fstack-protector-all命令，如果没有，请添加。以编译日志的为谁
* 确定源码是否有函数，如果没有函数，不会插入保护代码
* 如果函数最后有 exit() 也不会插入保护代码
* 确认源代码有没有宏，以致所有源代码被宏隔离开了，一行源代码也没有被编译到
* 是否有 -OS 或 -O2 等优化命令，有的编译器如果有这些优化命令，会优化掉，去掉这些优化命令重新编译试下是否这些命令影响了，如果是这些命令影响了，不用去掉这些命令。
* 如果有 -static -s 命令编译时会删除符号表，同样也不会检测到canary word，SP显示为NO 。
* -fstack-protector-all命令在android-ndk编译环境和在普通gcc编译环境是不一样的。
* \_\_stack\_chk\_fail是在函数返回之前进行插桩，如果函数没有得到返回，则不会插桩，例如：死循环、return之前有exit等；
* 如果一个动态库只由其它动态库链接而成，其它动态库已正确添加SP选项，但被封装的动态库SP选项无法生效的，属于已知例外场景。如： c.so 由 a.so 和 b.so 链接而成的，也即是说 c.so相当于封装了两个so（ a.so、 b.so ），虽然a.so和b.so已经正确添加了SP选项，但是c.so由于没有编译过程，编译选项SP无法生效，技术上属于合理的例外场景。

**（5）编译时添加了-fstack-protector-strong,但为什么SP显示为NO**

* 请从编译时的打印日志确认**目标文件(扫描的文件)**的**编译阶段**是否有 -fstack-protector-strong 命令，如果没有，请添加。以编译日志的为准
* 只有源代码函数里有以下2种代码之一才会插入保护代码:

      A:  局部数组

      B:  局部变量地址作为赋值语句的右值或函数参数

* 是否有 -OS 或 -O2 等优化命令，有的编译器如果有这些优化命令，会优化掉，去掉这些优化命令重新编译试下是否这些命令影响了，如果是这些命令影响了，不用去掉这些命令。
* 如果有 -static -s 命令编译时会删除符号表，同样也不会检测到canary word，SP显示为NO 。
* \_\_stack\_chk\_fail是在函数返回之前进行插桩，如果函数没有得到返回，则不会插桩，例如：死循环、return之前有exit等；
* 确定源码是否有函数，如果没有函数，不会插入保护代码
* 确认源代码有没有宏，以致所有源代码被宏隔离开了，一行源代码也没有被编译到。
* 如果一个动态库只由其它动态库链接而成，其它动态库已正确添加SP选项，但被封装的动态库SP选项无法生效的，属于已知例外场景。如： c.so 由 a.so 和 b.so 链接而成的，也即是说 c.so相当于封装了两个so（ a.so、 b.so ），虽然a.so和b.so已经正确添加了SP选项，但是c.so由于没有编译过程，编译选项SP无法生效，技术上属于合理的例外场景。

1. **编译时添加了-D\_FORTIFY\_SOURCE=2 -O2选项，但为什么FS显示为NO**

* 请从编译时的打印日志确认**目标文件(扫描的文件)**的**编译阶段**是否有 **-D\_FORTIFY\_SOURCE=2 -O2** 命令，如果没有，请添加。以编译日志的为准
* FS简单来说就是GCC做的安全函数（\_原函数名\_chk）替换 不安全函数。binscope就是检查二进制文件有没有安全函数（\_原函数名\_chk），如果有，报 YES,否则报 NO

**可能造成不进行chk函数替换的原因有:**

* 如果源代码里没有用到 **不安全函数** ，当然不会进行chk函数替换了。关于哪些是不安全函数，请看技术手册的附录。
* 如果FS选项所检查的函数没有造成溢出，-O2选项会对其进行优化，而不进行chk函数替换，导致FS安全特征显示为NO。详情请看技术手册的验证测试--2.1.5.2。
* 由于该选项的原理是编译器通过计算目的空间大小来判断是否越界，因此，对于动态长度大小的目的空间，该选项无法检测。详情请看技术手册的局限性--2.1.6 。

**（7）出现以下情况应该怎么办**

* 如果 SP或FS扫描结果显示为不及格，但是编译选项命令确认加了而且没有加错而且找出了原因，那就给相关的人员(issue负责人)发封邮件说明一下。

**（8）-fPIC 和 -fPIE 的具化场景(编译阶段)**

* 1. 直接编译可执行文件 –fPIE
  2. 直接编译成库 -fPIC
  3. 先编成多个.o再链接成可执行文件 -fPIE
  4. 先编成多个.o再链接成库 -fPIC
  5. 先编成多个.o，链接类型不确定。这时候建议提供两个版本的中间件，一个是加-fPIE，另一个是加-fPIC

**（9）PIC和PIE的区别**

* 首先需要解释的是PIE是两步，编译选项为-fPIE, 链接选项为-pie，-fPIC只是编译选项。
* PIC、PIE不能混用。用PIE(fPIE)编译生成动态库，他不会为全局变量在GOT表中创建对应的条目，与PIC的作用矛盾，动态库时会被其他程序调用的，所以可以在调用时对GOT中的变量地址重定位来确定全局变量地址，而PIE本身的目的不是生成共享库，而是生成位置无关的可执行程序，所以在连接时所有的变量都已经确定好了，包括全局的对外可见变量也默认视为程序内部变量。因此，如果用PIE生成共享库，一旦存在全局对外可见的变量，一方面，作为共享库，该变量地址被期望是可重定位的，另一方面，作为可执行程序（PIE），该变量地址被期望是固定的，就会产生矛盾
* 在 x86 架构的测试里，如果用 -fPIC 编译的中间件.o用来链接成可执行文件，在性能上会有影响。-fPIC消耗的性能比-fPIE的多
* **在支撑的案例中，PIE和PIC的命令混用，造成程序运行崩溃或其他问题的个数不少了。在此建议正确区分使用**

**（10）编译时添加了-fPIC ,但为什么PIC显示为NO**

* 请从编译时的打印日志确认**目标文件(扫描的文件)**的**编译阶段**是否有-fPIC 命令，如果没有，请添加。以编译日志的为准
* \*\*.a(\*\*.o): relocation R\_X86\_64\_32S against `.rodata.str1.8' can not be used when making  a shared object; recompile with -fPIC 如果有类似提示，请重新编译 .o时添加–fPIC。
* 如果是输入文件或依赖文件涉及汇编文件，则PIC不会生效
* 如果是输入文件或依赖文件某一个在编译时没有添加-fPIC，哪也会造成目标文件的 PIC 不会生效

**（11）编译时添加了-fPIE -pie ,但为什么PIE显示为NO**

* 请从编译时的打印日志确认**目标文件(扫描的文件)**的**编译阶段是**否有 -fPIE 命令和**链接阶段**是否有 -pie命令，如果没有，请添加。以编译日志的为准
* \*\*.a(\*\*.o): relocation R\_X86\_64\_32S against `.rodata.str1.8' can not be used when making  a shared object; recompile with -fPIC 如果有类似提示，请重新编译　.o　时添加 –fPIE
* 如果是输入文件或依赖文件涉及汇编文件，则 PIE 不会生效
* 请确认是否有 -T 链接脚本 命令，如果有，去掉试下.

**(12) 扫描报告为什么 Rpath/RunPath 显示为YES**

* 禁选的意思就是编译语句里不能有 "--rpath”相关的命令，如果脚本有，需要去掉.
* 如果编译工具是libtool,会自动添加RPATH,rpath禁选落地的方法，
* 如果rpath指向的路径是系统库，可以直接用chrapth这个工具删掉。

**（13）rpath禁选落地后，可能出现提示找不到相关库的错误，解决方法以下：**

* 把依赖的库直接放到系统路径里(系统路径包含默认的动态库搜索路径 /lib /usr/lib;如果是64位的系统还包括 /lib64 /usr/lib64)------官方推荐
* 动态库依赖动态库能够通过添加路径的方法替代rpath选项测试验证。
* 去除rpath后找不到库文件的解决方法

**（14）为什么Strip显示为NO**

* 请从编译时的打印日志确认**目标文件(扫描的文件)**的**链接阶段是**否有 -s 命令.如果没有，请添加。以编译日志的为准
* 请确认是否直接用 strip 命令来处理过该**目标文件(扫描的文件)**

在android的aar打包前，会自动strip需要打包的文件，所以在遇到此类文件时，要扫描 aar解压后的文件，而不是未打包前的文件