dpdk包安全编译选项问题整改

## **什么是安全编译选项**？

我个人理解，是在编译阶段通过增加某些选项，降低入侵者利用漏洞的风险，来达到保护系统的目的。

安全编译选项的适用范围：C/C++/java/Go 语言编写的应用层或内核态的代码。

以dpdk为例：

通过扫描结果可知，dpdk包的SP，Rpath/Runpath，Strip选项均存在问题，需要整改。



## SP选项（-fstack-protector-all/-fstack-protector-strong）

### 原理：

函数调用时会被分配栈空间，通过栈溢出可以被用于攻击，栈保护就是在函数栈尾加上一随机个校验值，当栈被溢出修改时，这个值会改变，当这个值被改时，函数就会调用\_\_stack\_chk\_fail函数。

二进制特征为每个函数开始前都会压栈多一个变量，弹栈时会做一个检查，检查后根据结果跳转至\_\_stack\_chk\_fail。

作用阶段：编译选项

### 判定方法：

sp选项扫描，可能会出现误报情况，一般通过查询编译日志，来查看选项是否符合要求。

注意：编译时，开启详细的编译日志选项，如
 make V=1 verbose=1 或 ninjia -v



在日志中查找 librte\_meter.so.21.0 相关信息：

通过日志，可以看到编译选项中没有 -fstack-protector-all/-fstack-protector-strong 选项，可以修改spec文件，进行添加。

### 修改：



+278行：export CFLAGS=”%{optflags}”

修改CFLAGS，添加 -fstack-protector-strong 选项。

## Rpath/Runpath选项

### 原理：

rpath选项是链接选项，主要用于防护LD\_LIBRARY\_PATH替换同名动态库的攻击。通过加入此选项可以指定一个运行时动态库搜索的路径，该路径的搜索优先级高于LD\_LIBRARY\_PATH指定的路径。也就是说，可执行文件在运行阶段进行动态库搜索时会首先在--rpath指定的路径查找动态库，然后才会到LD\_LIBRARY\_PATH指定的路径搜索。因此可以有效防御LD\_LIBRARY\_PATH =[attackpath]来替换同名动态库的攻击。

作用阶段：链接选项

### 判定方法：

可以通过 “ readelf -d $file ”命令可以，查看是否包含“rpath/runpath”。

以 dpdk-pdump 为例



### 修改：

可以通过chrpath命令删除rpath，“ chrpath –d $file ”

1. 添加编译依赖 chrpath



1. %install 阶段，使用chrpath 命令删除rpath。

为了防止删除后发生找不到相关库的问题，可以向/etc/ld.so.conf.d中写入xx.conf的配置文件

用来记录路径。



1. 最后记得在 %file阶段，打包配置文件。



## Strip选项（-ldflags=-s）

### 原理：

可执行文件的符号表包含，符号的值、类型、名称，定义的函数，变量等等，链接阶段是基于符号表的信息才能正确的将多个文件链接起来，链接完成后，符号表对可执行文件没有任何作用，还有可能成为攻击者的“利器”，因此删除符号表不仅可以减少文件大小，又可以降低安全风险。

**作用阶段：**链接选项（不推荐，需要两次编译构建），一般使用strip命令

### 判定方法：

使用file命令查看可执行文件，包含“ not stripped ”字段，即为没有strip的意思



### 修改：

1. 在 %install 阶段通过 strip 命令来删除符号表



1. rpm包在编译时会默认生成debugsource和debuginfo包，生成debuginfo包时，在 %file 打包阶段会执行strip去除可执行文件的调试信息，并将这些信息储存在debuginfo包中，以此达到strip的目的。

以当前dpdk包为例，编译后并未生成debugsource包和debuginfo包，查看dpdk.spec文件发现：



在 build阶段，debug\_package 被定义为 0 ，即禁用调试程序包构建。

因此修改dpdk的strip问题时，删除这行即可。