PESC: Per System-Call Stack Canary

该仓库为PESC的仓库，包含测试使用的内核以及PESC的代码补丁。

一.仓库结构

该仓库共包含7个分支，其内容具体如下：

- master分支
  - 包含patch文件夹，为PESC应用到内核所需要的代码补丁
    - per_task.diff：用于ARM64 Linux 5.0以前启用per-task canary的代码补丁
    - pesc_pmc.diff：在per task canary的基础上，用于启用PESC-PMC的代码补丁
    - pesc_rng.diff：在per task canary的基础上，用于启用PESC-RNG的代码补丁
    - verify.diff：在PESC的基础上，用于验证PESC是否已经成功被应用到kernel的代码补丁
    - pmc_sec_getpid.diff：测试getpid连续系统调用canary差值的代码补丁
    - pmc_sec_read.diff：测试read连续系统调用canary差值的代码补丁
    - pmc_sec_fork.diff：测试fork连续系统调用canary差值的的代码补丁
  - 包含test文件夹，包含用户空间测试程序源码
    - perf_eva文件夹：包含了android ndk工程，用于测试getpid、read、fork系统调用的性能
    - sec_eva文件夹：包含了android ndk工程，用于测试read系统调用的性能
    - testdata：测试read时使用的数据，也可以使用其他数据，大于1MB即可。
    - verify文件夹：包含了android ndk工程，用于验证PESC是否已经成功被应用到kernel
- 其余分支：包含实验中所用的hikey960内核的源码
  - original分支：hikey960原始内核的源码
  - per_task分支：启用per task canary后的hikey960内核源码
  - pesc_pmc：启用PESC-PMC后的hikey960内核源码
  - pesc_pmc_verify：用于验证PESC-PMC是否启用的内核源码
  - pesc_rng：启用PESC-RNG后的hikey960内核源码
  - pesc_rng_verify：用于验证PESC-RNG是否启用的内核源码
  - pmc_sec_getpid：测试getpid连续系统调用canary差值的内核源码
  - pmc_sec_read：测试read连续系统调用canary差值的内核源码
  - pmc_sec_fork：测试fork连续系统调用canary差值的内核源码

二.使用方法

1.前期准备

- 操作系统：Ubuntu18.04
- 硬件环境：hikey960开发板（附带电源和type-c USB数据线）或其他硬件环境
- 安卓版本：android-10.0.0_r1（推荐）
- 安卓内核版本：4.19.36
- 内核编译器版本：AARCH64 GCC 9.1交叉编译器
- 其他工具：Android ndk

2.内核编译及镜像刷写

启用per-task canary

对于ARM64 Linux v5.0及以上版本的内核，请跳过此步骤。对于ARM64 Linux v5.0及以下版本的内核，需要首先启用per-task canary，本repo已经给出了启用per-task canary后的hikey960内核源码，即per_task分支，对于其他ARM64 Linux v5.0及以下版本的内核，请将per_task.diff中的代码补丁添加到内核中。

启用PESC

本仓库已经给出了启用PESC后的hikey960内核源码，pesc_pmc分支和pesc_rng分支分别对应启用PESC-PMC和PESC-RNG的分支。对于其他内核，请添加对应的代码补丁pesc_pmc.diff或pesc_rng.diff到内核中。

添加测试代码

如无需进行测试，则跳过此步骤。

- 功能验证：使用系统调用读取当前进行的canary，需要添加一个新的系统调用get_canary。对于hikey960，本仓库已经提供了对应的内核源码，包含在pesc_pmc_verify分支和pesc_rng_verify分支中。对于其他内核，请根据verify.diff的内容自行添加系统调用。
- 性能测试：无需修改内核代码
- 安全性测试：只有PESC-PMC需要进行安全性测试。在每次特定系统调用（getpid、read、fork）被调用时记录当时的canary值，对于hikey960，本仓库已经提供了对应的内核源码，包含在pmc_sec_getpid，pmc_sec_read以及pmc_sec_fork分支中，分别对应getpid、read、fork的测试。对于其他内核，请根据pmc_sec_getpid.diff，pmc_sec_read.diff，pmc_sec_fork.diff的内容自行添加代码。

编译内核

此步骤必须使用**GCC 9.0及以上版本**的的交叉编译器编译，对于hikey960内核，请使用如下命令编译，对于其他内核，请自行编译。

```
#编译内核
$ make ARCH=arm64 hikey960_defconfig
$ make ARCH=arm64 CROSS_COMPILE=aarch64-linux- -j$(nproc)   #如有必要请更改
CORSS_COMPILE变量

#拷贝编好的内核到aosp目录下,请将$(AOSP)替换成aosp的根目录
$ cp arch/arm64/boot/Image.gz-dtb $(AOSP)/device/linaro/hikey-kernel/Image.gz-
dtb-hikey960-4.19
$ cp arch/arm64/boot/dts/hisilicon/hi3660-hikey960.dtb
 $(AOSP)/device/linaro/hikey-kernel/hi3660-hikey960.dtb-4.19
```

编译完内核后，请对编译完的vmlinux使用

```
$ aarch64-linux-gnu-objdump -d vmlinux > vmlinux.objdump
```

检查vmlinux.objdump中是否确实使用了per task canary，启用per_task_canary后对于编译完的内核，应该能看到如下结果：

```
ffff000010085610 <user_enable_single_step>:
ffff000010085610:       a9be7bfd        stp     x29, x30, [sp, #-32]!
ffff000010085614:       d5384102        mrs     x2, sp_el0
ffff000010085618:       aa0003e1        mov     x1, x0
ffff00001008561c:       910003fd        mov     x29, sp
```

```
ffff000010085620:       f9423840        ldr     x0, [x2, #1136]
ffff000010085624:       f9000fe0        str     x0, [sp, #24]
ffff000010085628:       d2800000        mov     x0, #0x0
// #0
ffff00001008562c:       f9400020        ldr     x0, [x1]
ffff000010085630:       37a80080        tbnz    w0, #21, ffff000010085640
<user_enable_single_step+0x30>
ffff000010085634:       d2a00400        mov     x0, #0x200000
// #2097152
ffff000010085638:       942c3a16        bl      ffff000010b93e90
<__ll_sc_arch_atomic64_fetch_or>
ffff00001008563c:       36a80100        tbz     w0, #21, ffff00001008565c
<user_enable_single_step+0x4c>
ffff000010085640:       9111c042        add     x2, x2, #0x470
ffff000010085644:       f9400fe1        ldr     x1, [sp, #24]
ffff000010085648:       f9400040        ldr     x0, [x2]
ffff00001008564c:       ca000020        eor     x0, x1, x0
ffff000010085650:       b5000100        cbnz    x0, ffff000010085670
...
ffff000010085670:       94015e60        bl      ffff0000100dcff0
<__stack_chk_fail>
ffff000010085674:       d503201f        nop
ffff000010085678:       d503201f        nop
ffff00001008567c:       d503201f        nop
```

如上述汇编代码所示，sp_el0的值被加载到x2中，从[x2, #1136]取出的值被放到了栈上，在ffff00001008564c处，这两个值又被进行了异或操作，下一条指令根据异或的结果是否为0决定是否要跳转到ffff000010085670处，该处指令会跳转到__stack_chk_fail。一个更简单的方法判断是否开启了per task canary可以通过寻找含有指令

```
bl      ffff0000100dcff0 <__stack_chk_fail>         #地址可以是其他值
```

的函数编译完后的汇编指令中是否出现了sp_el0寄存器（存放task_struct指针），如果没有开启per task canary是不会出现sp_el0这个寄存器的。

编译系统镜像

对于hikey960请按如下步骤进行，其余环境请自行编译

```
$ cd $(AOSP)    #请将$(AOSP)替换成aosp的根目录
$ . ./build/envsetup.sh
$ lunch hikey960-userdebug
$ make -j12
```

刷写镜像

对于hikey960请按如下步骤进行。

1.打开开关 3，选择 fastboot 模式（有关详情，请参阅 https://www.96boards.org/documentation/consumer/hikey/hikey960/getting-started/），连接USB数据线。

2.启动开发板。

3.刷写镜像

```
$ cd $(AOSP)/device/linaro/hikey/installer/hikey960 #请将$(AOSP)替换成aosp的根目录
$ ./flash-all.sh
```

4.关闭开关 3，然后重启开发板。

3.测试

如无需测试，请跳过此步骤。测试前请切换到master分支。

- 功能测试

  对于其他内核，请先修改test/verify/jni/test.c中的系统调用号为之前添加的系统调用号

  ```
  $ cd test/verify/jni
  $ $(andorid-ndk)/ndk-build  #请将$(andorid-ndk)替换成android-ndk的根目录
  $ cd ../libs/arm64-v8a
  $ adb push test /data/local/tmp
  $ adb shell
  ```

  进入adb的shell后

  ```
  $ cd /data/local/tmp
  $ ./test
  ```

  运行test后程序会打印出每一次得到的canary，如果各次canary互不相同且不为0，则说明PESC的功能正常。

- 性能测试

  该仓库包含了对三个system call的性能测试，首先需要编译三个system call的用户态测试文件,对于getpid系统调用的测试方法如下：

  ```
  $ cd test/perf_eva/getpid/jni
  $ $(andorid-ndk)/ndk-build  #请将$(andorid-ndk)替换成android-ndk的根目录
  $ cd ../libs/arm64-v8a
  $ adb push getpidtester /data/local/tmp
  $ adb shell
  ```

  进入adb的shell后

  ```
  $ cd /data/local/tmp
  $ chmod u+x getpidtester
  $ ./getpidtester
  ```

  对于read和fork系统调用，测试方法同理，测试read系统调用时，必须将需要的数据也一同刷入开发板中

  ```
  $ adb push test/testdata /data/local/tmp
  ```

  每个程序运行之后最终都会打印出类似如下输出：

  ```
  avg: xxx    #xxx代表平均花费的时间
  ```

  通过比较添加PESC前后平均花费的时间，可以简单得到PESC对系统调用带来的性能开销。

对于Unixbench的测试，可以用termux App安装必要的开发环境（clang、perl、make等），然后利用adb将Unixbench的源码push到开发板上，并在设置环境变量后（termux安装的linux环境根目录应为/data/data/com.termux/files/），在Unixbench目录下编译Unixbench。编译完后运行Unixbench

- 安全性测试

  安全性测试的目的以及在hikey960上的结果可以参阅前序文档或PESC论文。

  该仓库包含了对三个system call的安全性测试，首先需要编译三个system call的用户态测试文件，对于getpid系统调用的测试方法如下（必须使用对应的内核分支，即pmc_sec_getpid，pmc_sec_read,pmc_sec_fork，或添加对应补丁pmc_sec_getpid.diff,pmc_sec_read.diff,pmc_sec_fork.diff的其他内核）

```
$ cd test/sec_eva/getpid/jni
$ $(andorid-ndk)/ndk-build  #请将$(andorid-ndk)替换成android-ndk的根目录
$ cd ../libs/arm64-v8a
$ adb push getpidtester /data/local/tmp
$ adb shell
```

  进入adb的shell后

```
$ cd /data/local/tmp
$ chmod u+x getpidtester
$ ./getpidtester
```

  程序会运行约10分钟，之后会打印进程的pid，

```
$ echo " 123 " > /proc/hello_proc  #使用进程打印的pid替换123
$ cat /proc/hello_proc
$ dmesg
```

  运行dmesg命令后会打印出内核的日志，将日志中以canary开头的条目提取出来即为程序记录的canary，类似如下结果：

```
[  543.798834] 10Canary: 11ba4508e 11ba45e9a 11ba6b1c9 11ba6b58d 9ad5d891
9ad5de9b 9ad7f9a2 9ad7fe76 9ad93884 9ad93be1
[  543.798836] 20Canary: 9adb0c32 9adb0fa5 9adbce89 9adbd1b6 9add03c9
9add06f6 9adf274c 9adf2c4e 9adfc54f 9adfc8c0
[  543.809273] 30Canary: 9ae08c4d 9ae08f3a 9ae13c5d 9ae13f3d 9ae1df2f
9ae1e217 9ae27db5 9ae28098 9ae32948 9ae32c40
[  543.819359] 40Canary: 9ae3ca42 9ae3cd25 9ae46c55 9ae46f74 9ae50b8e
9ae50e77 9ae5a96e 9ae5ac3c 9ae6330b 9ae6361a
[  543.829449] 50Canary: 9ae6d375 9ae6d64a 9ae7740c 9ae776be 9ae81481
9ae81766 9ae8b396 9ae8b69f 9ae951d2 9ae95502
[  543.839534] 60Canary: 9ae9f0a4 9ae9f398 9aea9202 9aea94ef 9aeb31cf
9aeb34bc 9aebcfa5 9aebd2ba 9aec7491 9aec7788
[  543.849624] 70Canary: 9aed25cf 9aed289f 9aedc5bd 9aedc898 9aee6731
9aee6a37 9aef04f6 9aef07b4 9aefa49b 9aefa78c
[  543.859710] 80Canary: 9af1f219 9af1f567 9af30a0c 9af30d4b 9af3bbfe
9af3bed9 9af4d405 9af4d737 9af58969 9af58c66
...
```

  对每一对canary做差即可得到对应的差值。对其他两个系统调用的测试方法类似，对于read还需要将测试数据也一起push到开发板上，具体参考性能测试中的read测试。

注意：内核中打印的canary数可能不等于用户空间测试程序调用的次数，在做差前，需要人工判断应该从第一个canary还是第二个canary开始将每两个canary当成一对。 判断方法如下：从头开始逐一查看每个canary的值，如果是连续调用的系统调用，其canary值会非常接近，由于存在多个PMU，因此不同CPU上打印出来的canary值存在显著差异，存在较大差异的两个canary一般不是连续调用系统调用产生的，从而可以确定从奇数个开始计数还是从偶数个开始计数，每两个canary作为一对计算差值。此外，最终获取的差值应去掉首位部分（如去掉首位各250个差值），同时去除一些outlier。