clkscrew on nexus6

1.环境配置

本部分主要参考https://github.com/AxelRoudaut/How_to_build_android_kernel_to_load_modules_dynamically/blob/master/README.md

nexus6手机不支持动态加载模块进入内核，需要重新编译

1.安装adb和fastboot，使用fastboot解锁bootloader。

```
sudo apt-get install adb fastboot
adb start-server
adb devices
adb reboot bootloader
fastboot devices
fastboot oem unlock
```

这一步也可以在手机上操作，首先调到开发者模式，然后在设置中解锁oem。

2.交叉编译器配置

arm-linux-androideabi-gcc会出现编译异常，因此使用**arm-eabi-gcc**。

下载地址arm-eabi-4.8

选择4.8版本的原因：在内核映像的配置文件*build.config*有如下定义：

```
LINUX_GCC_CROSS_COMPILE_PREBUILTS_BIN=prebuilts/gcc/linux-x86/arm/arm-eabi-
4.8/bin
```

3.编译内核

The Nexus 6 (shamu) we received was configured as follow:

- Android: 6.0.1
- Kernel: 3.10.40 - g557ba38 Nov 4 2015 armv7l
- Security Update: Dec 2015
- Build: MMB29K
- Board Platform: msm 8084
- Every other info here

下载地址 the official Google kernel android-msm-shamu-3.10-marshmallow

更改$MYKERNEL/arch/arm/configs/shamu_config

添加

```
CONFIG_MODULES=y
```

允许添加内核模块

将该更改后的文件作为.config文件并交叉编译：

```
cp arch/arm/configs/shamu_config .config
make ARCH=arm SUBARCH=arm CROSS_COMPILE=arm-eabi- shamu_defconfig
make ARCH=arm SUBARCH=arm CROSS_COMPILE=arm-eabi- menuconfig
```

在弹出的窗口查看是否权限已经更改

最后编译整个内核

```
make ARCH=arm SUBARCH=arm CROSS_COMPILE=arm-eabi- -j4
```

4.在bootimg中更新内核映像

Download the **factory image** for the Nexus6 (shamu) Android 6.0.1 (MMB29K) here.

使用abootimg解压后文件的对应关系：

initrd.img (= ramdisk), zImage (= kernel) and bootimg.cfg (= kernel config)

将bootimg.cfg改为已经更改过的zImage

```
abootimg -u boot.img -f bootimg.cfg -k $MYKERNEL/arch/arm/boot/zImage-dtb -r
initrd.img [-s <secondstage>] -c bootsize=[SIZE_IT_WANTS]
```

5.刷机

```
fastboot flash boot boot.img
fastboot reboot
```

6.root手机

使用TWRP+supersu实现

先安装supersu，此时会提示尚未root，将supersu对应的包使用TWRP的恢复模式安装，完成root

2.实验代码

地址 **https://github.com/ghost-in-a-shell/myclkscrew**

运行环境：ubuntu 20.04 物理机

代码实现了上下层分离的自动化故障注入和日志记录

1.运行方法

运行时要更改代码内的绝对路径

更改harness.py中的参数列表

更改glitchutils.py中的日志文件名和USB端口

运行监视器程序之后运行自动化程序：

```
python3 monior.py
python3 harness.py
```

如果需要生成输出参数列表

注意：上层自动化模块和下层之间的接口是攻击参数 1.频率 2.持续时间 3.predelay 4.温度

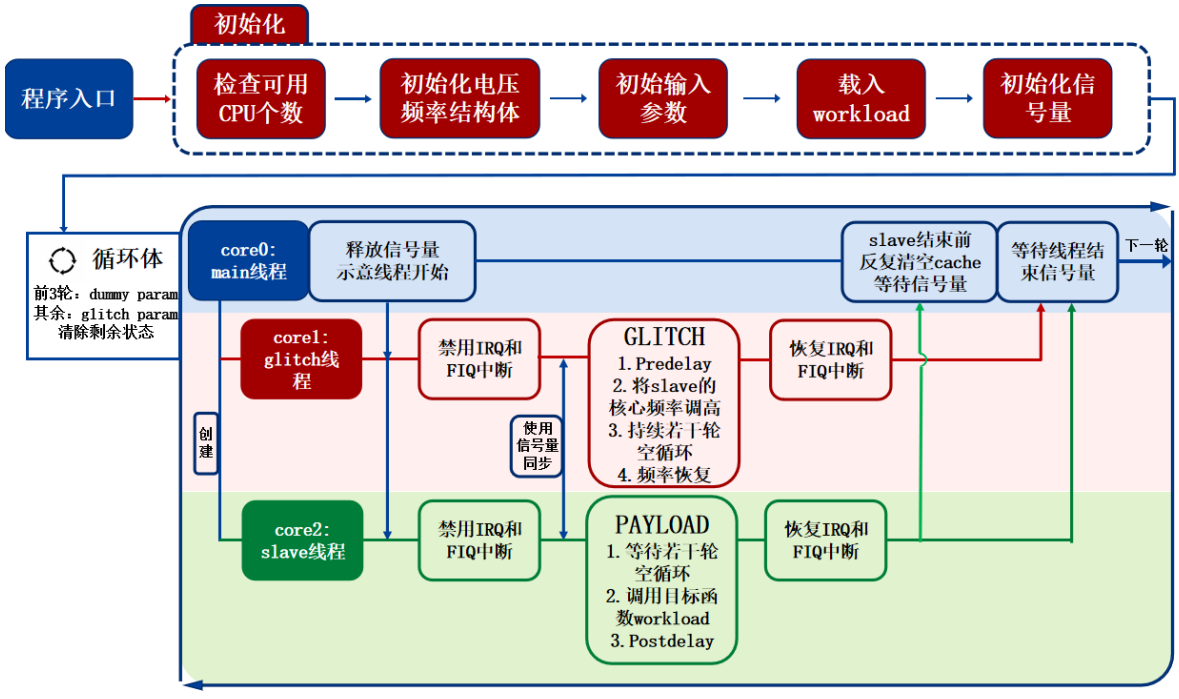
如果更换下层的内核模块需要注意接口格式如下例：

```
/system/bin/insmod /data/local/tmp/glitchmin.ko PARAM_iter=15 PARAM_volt=1055000
PARAM_gdelay=%s PARAM_delaypre=%s PARAM_gval=%s
```

2.faultmin_SD805

作用：

内核模块，完成内核超频的过程。目标程序是RSA中的filpendianess



使用的是原作者库中的模块：https://github.com/0x0atang/clkscrew

3.dofever

用于升高设备温度，多线程运行大计算量程序

升温效果有限，可以辅助物理保温措施(如泡沫塑料)

4.harness 上层自动化测试模块：

地址：https://github.com/ghost-in-a-shell/myclkscrew/tree/main/glitchharness

pres.sh:

按照clkscrew官方库的adb实现方法不可行，显示无权限

因此将初始化配置的shell脚本在传入设备，并使用adb命令调用脚本，成功执行。

命令是：

```
stop thermal-engine
stop mpdecision
echo 1 > /sys/devices/system/cpu/cpu1/online
echo 1 > /sys/devices/system/cpu/cpu2/online
echo 0 > /sys/devices/system/cpu/cpu3/online
echo userspace > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
echo userspace > /sys/devices/system/cpu/cpu1/cpufreq/scaling_governor
echo userspace > /sys/devices/system/cpu/cpu2/cpufreq/scaling_governor
echo 2649600 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
echo 2649600 > /sys/devices/system/cpu/cpu1/cpufreq/scaling_setspeed
echo 2649600 > /sys/devices/system/cpu/cpu2/cpufreq/scaling_setspeed
echo 0 > /proc/sys/kernel/randomize_va_space
cat /sys/devices/virtual/thermal/thermal_zone0/temp
```

前两个命令（`stop thermal-engine`和`stop mpdecision`）停止了两个与热管理和电源管理相关的系统服务。

接下来的三个命令（`echo 1 > /sys/devices/system/cpu/cpu1/online`，`echo 1 > /sys/devices/system/cpu/cpu2/online`和`echo 0 > /sys/devices/system/cpu/cpu3/online`）启用了两个CPU核心（`cpu1`和`cpu2`），同时禁用了另一个（`cpu3`）。

接下来的三个命令（`echo userspace > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor`，`echo userspace > /sys/devices/system/cpu/cpu1/cpufreq/scaling_governor`和`echo userspace > /sys/devices/system/cpu/cpu2/cpufreq/scaling_governor`）将CPU频率调节器设置为`userspace`，用于前三个CPU核心（`cpu0`，`cpu1`和`cpu2`）。

接下来的三个命令（`echo 2649600 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed`，`echo 2649600 > /sys/devices/system/cpu/cpu1/cpufreq/scaling_setspeed`和`echo 2649600 > /sys/devices/system/cpu/cpu2/cpufreq/scaling_setspeed`）将前三个CPU核心的最大CPU频率设置为2649600 Hz。

接下来的命令（`echo 0 > /proc/sys/kernel/randomize_va_space`）禁用了地址空间布局随机化（ASLR），这是一种安全功能，用于随机化进程的内存地址空间，以使攻击者更难利用漏洞。

最后一个命令（`cat /sys/devices/virtual/thermal/thermal_zone0/temp`）显示设备热区0的当前温度。

glitchutils.py:

adb_exec_cmd_one函数，执行一次攻击

write_log函数，将结果写入日志文件

presets函数，执行上述的pres.sh

faulthandler函数，自动化运行的关键程序。因为实验需要纯净的安卓系统环境，以及实验过程中系统经常崩溃重启，所以需要经常性检测设备是否存活。有的时候检测到设备但始终offline（多见于崩溃重启情况），尝试重启adb服务也不管用，需要物理插拔数据线恢复连接，**实现中禁用设备的usb口并重启模拟物理插拔，因此在新的设备上运行该代码需要更改如下位置**，将1-12.4更改为实际的USB口序号。

```
84                          os.system('sudo echo \'1-12.4\' > /sys/bus/usb/drivers/usb/unbind')
85                          time.sleep(1)
86                          os.system('sudo chmod 777 /sys/bus/usb/drivers/usb/bind')
87                          os.system('sudo echo \'1-12.4\' > /sys/bus/usb/drivers/usb/bind')
```

harnness.py

实现控制包括温度的实验参数，进行实验

输入为参数列表

参数列表包含1.频率 2.持续时间 3.predelay 4.温度

temperctrl.py

调用fever的app，提升温度到实验指定值

monitor.py

监视输出并写日志，并在设备离线时循环等待

3.当前问题

1.时间效率问题：

实验需要经常性重启设备，而安卓系统重启时间较长，对于nexus6手机，一次重启大约需要45s。特别是确定攻击参数组时，故障注入成功率小于百分之五，且需要大量实验的时候，时间效率的问题尤为明显。现在的性能大概是每分钟两次故障注入。

同时，如果在新的目标程序进行故障攻击，需要重新找到一组攻击参数，耗时较长。

2.故障位置和数值单一

实验结果中发现，虽然找到了一组成功率约为百分之五的参数，但是发现故障点位置和数值集中在几个特定的位置和值。当然这可以胜任对于RSA的故障攻击，但是对于AES的DFA，需要故障注入位置尽量均匀，覆盖所有的S盒输入，且数值不能固定，是否可以攻击成功仍然存疑。

3.trustzone实验困难

需要借助特定的漏洞导入TA