QEMU在嵌入式逆向分析中的应用

CryptoPentest / 2019-03-07 08:29:00 / 浏览数 1407 安全技术 IoT安全 顶(1) 踩(0)

本文详细汇总介绍了应用QEMU模拟器进行嵌入式环境构建及应用级、内核级逆向调试的技术方法，进行了详尽的实例讲解。

一、用QEMU模拟嵌入式调试环境

1. 安装arm的交叉编译工具链

如果订制一个交叉编译工具链，可使用crosstool-ng开源软件来构建。但在这里建议直接安装arm的交叉编译工具链：

```
sudo apt-get install gcc-arm-linux-gnueabi
```

或针对特定版本安装：

```
sudo apt-get install gcc-4.9-arm-linux-gnueabi
sudo apt-get install gcc-4.9-arm-linux-gnueabi-base
```

建立需要的软链接

```
sudo ln -s /usr/bin/arm-linux-gnueabi-gcc-4.9 /usr/bin/arm-linux-gnueabi-gcc
```

2. 编译Linux内核

生成vexpress开发板的config文件：

```
make CROSS_COMPILE=arm-linux-gnueabi- ARCH=arm O=./out_vexpress_3_16  vexpress_defconfig
```

然后执行如下命令：

```
make CROSS_COMPILE=arm-linux-gnueabi- ARCH=arm O=./out_vexpress_3_16 menuconfig
```

将：

```
System Type  --->

■    [ ] Enable the L2x0 outer cache controller
```

即，■ `Enable the L2x0 outer cache controller` 取消， 否则Qemu会起不来， 暂时还不知道为什么。

编译：

```
make CROSS_COMPILE=arm-linux-gnueabi- ARCH=arm O=./out_vexpress_3_16 zImage -j2
```

生成的内核镜像位于./out_vexpress_3_16/arch/arm/boot/zImage， 后续qemu启动时需要使用该镜像。

3. 下载和安装qemu模拟器

下载qemu，用的版本是2.4版本，可以用如下方式下载，然后checkout到2.4分支上即可

```
git clone git://git.qemu-project.org/qemu.git
cd qemu
git checkout remotes/origin/stable-2.4 -b stable-2.4
```

配置qemu前，需要安装几个软件包：

```
sudo apt-get install zlib1g-dev

sudo apt-get install libglib2.0-0

sudo apt-get install libglib2.0-dev

sudo apt-get install libtool

sudo apt-get install libsdl1.2-dev

sudo apt-get install autoconf
```

配置qemu，支持模拟arm架构下的所有单板，为了使qemu的代码干净一些，采用如下方式编译，最后生成的中间文件都在build下

```
mkdir build
cd build
../qemu/configure --target-list=arm-softmmu --audio-drv-list=
```

编译和安装：

```
make
make install
```

查看qemu支持哪些板子

qemu-system-arm -M help

4. 测试qemu和内核能否运行成功

```
qemu-system-arm \
   -M vexpress-a9 \
   -m 512M \
   -kernel /root/tq2440_work/kernel/linux-stable/out_vexpress_3_16/arch/arm/boot/zImage \
   -nographic \
   -append "console=ttyAMA0"
```

这里简单介绍下qemu命令的参数：

-M vexpress-a9 模拟vexpress-a9单板，你可以使用-M ?参数来获取该qemu版本支持的所有单板

-m 512M 单板运行物理内存512M

-kernel /root/tq2440_work/kernel/linux-stable/out_vexpress_3_16/arch/arm/boot/zImage 告诉qemu单板运行内核镜像路径

-nographic 不使用图形化界面，只使用串口

-append "console=ttyAMA0" 内核启动参数，这里告诉内核vexpress单板运行，串口设备是哪个tty。

注意：

我每次搭建，都忘了内核启动参数中的console=参数应该填上哪个tty，因为不同单板串口驱动类型不尽相同，创建的tty设备名当然也是不相同的。那vexpress单板的tty设备名是哪个呢？ 其实这个值可以从生成的.config文件CONFIG_CONSOLE宏找到。

如果搭建其它单板，需要注意内核启动参数的console=参数值，同样地，可从生成的.config文件中找到。

5. 制作根文件系统

依赖于每个开发板支持的存储设备，根文件系统可以放到Nor
Flash上，也可以放到SD卡，甚至外部磁盘上。最关键的一点是你要清楚知道开发板有什么存储设备。本文直接使用SD卡做为存储空间，文件格式为ext3格式。

（1）下载、编译和安装busybox

下载地址：https://busybox.net/downloads/

配置：

在busybox下执行 make menuconfig

做如下配置：

```
Busybox Settings  --->

■    Build Options  --->

■        [*] Build BusyBox as a static binary (no shared libs)

■        (arm-linux-gnueabi-) Cross Compiler prefix
```

然后执行

```
make
make install
```

安装完成后，会在busybox目录下生成_install目录，该目录下的程序就是单板运行所需要的命令。

（2）形成根目录结构

先在Ubuntu主机环境下，形成目录结构，里面存放的文件和目录与单板上运行所需要的目录结构完全一样，然后再打包成镜像（在开发板看来就是SD卡），这个临时的目录
脚本 mkrootfs.sh 完成这个任务：

```
#!/bin/bash

sudo rm -rf rootfs
sudo rm -rf tmpfs
sudo rm -f a9rootfs.ext3

sudo mkdir rootfs
sudo cp busybox/_install/*  rootfs/ -raf

sudo mkdir -p rootfs/proc/
sudo mkdir -p rootfs/sys/
sudo mkdir -p rootfs/tmp/
sudo mkdir -p rootfs/root/
sudo mkdir -p rootfs/var/
sudo mkdir -p rootfs/mnt/

sudo cp etc rootfs/ -arf

sudo cp -arf /usr/arm-linux-gnueabi/lib rootfs/

sudo rm rootfs/lib/*.a
sudo arm-linux-gnueabi-strip rootfs/lib/*

sudo mkdir -p rootfs/dev/
sudo mknod rootfs/dev/tty1 c 4 1
sudo mknod rootfs/dev/tty2 c 4 2
sudo mknod rootfs/dev/tty3 c 4 3
sudo mknod rootfs/dev/tty4 c 4 4
sudo mknod rootfs/dev/console c 5 1
sudo mknod rootfs/dev/null c 1 3

sudo dd if=/dev/zero of=a9rootfs.ext3 bs=1M count=32
sudo mkfs.ext3 a9rootfs.ext3

sudo mkdir -p tmpfs
sudo mount -t ext3 a9rootfs.ext3 tmpfs/ -o loop
sudo cp -r rootfs/*  tmpfs/
sudo umount tmpfs
```

其中，etc下是启动配置文件，可以的到这里下载：

http://files.cnblogs.com/files/pengdonglin137/etc.tar.gz

（3）系统启动运行

完成上述所有步骤之后，就可以启动qemu来模拟vexpress开发板了，命令参数如下：

```
qemu-system-arm \
   -M vexpress-a9 \
   -m 512M \
   -kernel /root/tq2440_work/kernel/linux-stable/out_vexpress_3_16/arch/arm/boot/zImage \
   -nographic \
   -append "root=/dev/mmcblk0  console=ttyAMA0" \
   -sd /root/tq2440_work/busybox_study/a9rootfs.ext3
```

上面是不太图形界面的，下面的命令可以产生一个图形界面：

```
qemu-system-arm \
   -M vexpress-a9 \
   -serial stdio \
   -m 512M \
   -kernel /root/tq2440_work/kernel/linux-stable/out_vexpress_3_16/arch/arm/boot/zImage \
   -append "root=/dev/mmcblk0  console=ttyAMA0 console=tty0" \
   -sd /root/tq2440_work/busybox_study/a9rootfs.ext3
```

5. 下载、编译u-boot代码

u-boot从下面的网址获得：

http://ftp.denx.de/pub/u-boot/

解压后，配置，编译：

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- vexpress_ca9x4_defconfig
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-
```

使用qemu测试：

```
qemu-system-arm -M vexpress-a9 \
    -kernel u-boot \
    -nographic \
    -m 512M
```

6. 实现用u-boot引导Linux内核

这里采用的方法是，利用网络引导的方式启动Linux内核。开启Qemu的网络支持功能，启动u-boot，设置u-boot的环境变量，u-boot采用tftp的方式将uImage格式的Linu
include/configs/vexpress_common.h。如果用Qemu直接启动Kernel，是通过-append parameter
的方式给kernel传参的，现在是通过u-boot，那么需要通过u-boot的环境变量bootargs，可以设置为如下值 `setenv bootargs 'root=/dev/mmcblk0 console=ttyAMA0 console=tty0'`。 然后设置u-boot环境变量bootcmd，如下:`setenv bootcmd 'tftp 0x60008000 uImage; bootm 0x60008000'`。

具体方式如下：

（1）启动Qemu的网络支持

- 输入如下命令安装必要的工具包:

```
sudo apt-get install uml-utilities
sudo apt-get install bridge-utils
```

- 输入如下命令查看 /dev/net/tun 文件:

```
ls -l /dev/net/tun
crw-rw-rwT 1 root root 10, 200 Apr 15 02:23 /dev/net/tun
```

- 创建 /etc/qemu-ifdown 脚本，内容如下所示：

```
#!/bin/sh

echo sudo brctl delif br0 $1
sudo brctl delif br0 $1

echo sudo tunctl -d $1
sudo tunctl -d $1

echo brctl show
brctl show
```

输入如下命令为 /etc/qemu-ifup 和 /etc/qemu-ifdown 脚本加上可执行权限：

```
chmod +x /etc/qemu-ifup
chmod +x /etc/qemu-ifdown
```

那么先手动执行如下命令：
`/etc/qemu-ifup tap0`

（2）配置u-boot

主要是修改include/configs/vexpress_common.h

```
diff --git a/include/configs/vexpress_common.h b/include/configs/vexpress_common.h
index 0c1da01..9fa7d9e 100644
--- a/include/configs/vexpress_common.h
+++ b/include/configs/vexpress_common.h
@@ -48,6 +48,11 @@
#define CONFIG_SYS_TEXT_BASE   0x80800000
#endif

+/* netmask */
```

```
+#define CONFIG_IPADDR   192.168.11.5
+#define CONFIG_NETMASK  255.255.255.0
+#define CONFIG_SERVERIP 192.168.11.20
+
/*
 * Physical addresses, offset from V2M_PA_CS0-3
 */
@@ -202,7 +207,9 @@
#define CONFIG_SYS_INIT_SP_ADDR               CONFIG_SYS_GBL_DATA_OFFSET

/* Basic environment settings */
-#define CONFIG_BOOTCOMMAND             "run bootflash;"
+/* #define CONFIG_BOOTCOMMAND          "run bootflash;" */
+#define CONFIG_BOOTCOMMAND             "tftp 0x60008000 uImage; setenv bootargs 'root=/dev/mmcblk0 console=ttyAMA0'; bootm 0x
+
```

说明：这里把ipaddr等设置为了192.168.11.x网段，这个需要跟br0的网址一致，br0的地址在/etc/qemu-ifuo中修改：

```
#!/bin/sh

echo sudo tunctl -u $(id -un) -t $1
sudo tunctl -u $(id -un) -t $1

echo sudo ifconfig $1 0.0.0.0 promisc up
sudo ifconfig $1 0.0.0.0 promisc up

echo sudo brctl addif br0 $1
sudo brctl addif br0 $1

echo brctl show
brctl show

sudo ifconfig br0 192.168.11.20
```

然后重新编译u-boot代码

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- -j2
```

测试：

```
qemu-system-arm -M vexpress-a9 \
    -kernel u-boot \
    -nographic \
    -m 512M \
    -net nic,vlan=0 -net tap,vlan=0,ifname=tap0
```

（3）配置Linux Kernel

因为要用u-boot引导，所以需要把Kernel编译成uImage格式。这里需要我们制定LOADADDR的值，即uImage的加载地址，根据u-boot，我们可以将其设置为0x6000800

命令如下：

```
make CROSS_COMPILE=arm-linux-gnueabi- ARCH=arm O=./out_vexpress_3_16 LOADADDR=0x60008000  uImage -j2
```

编译生成的uImage在 linux-stable/out_vexpress_3_16/arch/arm/boot下，然后将uImage拷贝到/tftpboot目录下。

执行如下命令：

```
qemu-system-arm -M vexpress-a9 \
    -kernel /root/tq2440_work/u-boot/u-boot/u-boot \
    -nographic \
    -m 512M \
    -net nic,vlan=0 -net tap,vlan=0,ifname=tap0 \
    -sd /root/tq2440_work/busybox_study/a9rootfs.ext3
```

（4）开启图形界面

修改u-boot的bootargs环境变量为：

```
setenv bootargs 'root=/dev/mmcblk0 console=ttyAMA0 console=tty0';
```

执行命令：

```
qemu-system-arm -M vexpress-a9 \
    -kernel /root/tq2440_work/u-boot/u-boot/u-boot \
    -nographic \
    -m 512M \
    -net nic,vlan=0 -net tap,vlan=0,ifname=tap0 \
    -sd /root/tq2440_work/busybox_study/a9rootfs.ext3
```

7. 用NFS挂载文件系统

（1）配置u-boot的环境变量bootargs

```
setenv bootargs 'root=/dev/nfs rw nfsroot=192.168.11.20:/nfs_rootfs/rootfs init=/linuxrc console=ttyAMA0 ip=192.168.11.5'
```

（2）配置kernel

配置内核，使其支持nfs挂载根文件系统

```
make CROSS_COMPILE=arm-linux-gnueabi- ARCH=arm O=./out_vexpress_3_16/ menuconfig
```

配置：

File systems --->

```
[*] Network File Systems  --->

   <*>   NFS client support

   <*>     NFS client support for NFS version 3

   [*]       NFS client support for the NFSv3 ACL protocol extension

   [*]   Root file system on NFS
```

然后重新编译内核

```
make CROSS_COMPILE=arm-linux-gnueabi- ARCH=arm O=./out_vexpress_3_16 LOADADDR=0x60003000 uImage -j2
```

将生成的uImage拷贝到/tftpboot下。

启动：

```
qemu-system-arm -M vexpress-a9 \
    -kernel /root/tq2440_work/u-boot/u-boot/u-boot \
    -nographic \
    -m 512M \
    -net nic,vlan=0 -net tap,vlan=0,ifname=tap0
    -sd /root/tq2440_work/busybox_study/a9rootfs.ext3
```

二、用QEMU辅助动态调试

1. 用QEMU调试RAW格式ARM程序

```
.text
   start:                      @ Label, not really required
           mov   r0, #5        @ Load register r0 with the value 5
           mov   r1, #4        @ Load register r1 with the value 4
           add   r2, r1, r0    @ Add r0 and r1 and store in r2

   stop:   b stop              @ Infinite loop to stop execution
```

该ARM汇编程序功能是两数相加，而后进入死循环。保存该文件名为add.S。

用as进行编译：

```
$ arm-none-eabi-as -o add.o add.s
```

用ld进行链接：

```
$ arm-none-eabi-ld -Ttext=0x0 -o add.elf add.o
```

`-Ttext=0x0`指示地址与标签对齐，即指令从 0x0 开始。可以用nm命令查看各标签对齐情况：

```
$ arm-none-eabi-nm add.elf
... clip ...
00000000 t start
0000000c t stop
```

ld链接生成的是elf格式，只能在操作系统下运行。应用如下命令将其转换成二进制格式：

```
$ arm-none-eabi-objcopy -O binary add.elf add.bin
```

当ARM处理器重启时，其从地址0x0处开始执行指令。在connex开发板上有一16MB的flash空间位于地址0x0。利用qemu模拟connex开发板，并用flash运行：

dd生成16MB flash.bin文件。

```
$ dd if=/dev/zero of=flash.bin bs=4096 count=4096
```

将add.bin拷贝到flash.bin起始位置：

```
$ dd if=add.bin of=flash.bin bs=4096 conv=notrunc
```

启动运行：

```
$ qemu-system-arm -M connex -pflash flash.bin -nographic -serial /dev/null
```

其中-pflash选项指示flash.bin文件作为flash memory.

可在qemu命令提示符下查看系统状态：

```
(qemu) info registers
R00=00000005 R01=00000004 R02=00000009 R03=00000000
R04=00000000 R05=00000000 R06=00000000 R07=00000000
R08=00000000 R09=00000000 R10=00000000 R11=00000000
R12=00000000 R13=00000000 R14=00000000 R15=0000000c
PSR=400001d3 -Z-- A svc32

(qemu) xp /4iw 0x0
0x00000000:  mov        r0, #5  ; 0x5
0x00000004:  mov        r1, #4  ; 0x4
0x00000008:  add        r2, r1, r0
0x0000000c:  b  0xc
```

可在qemu命令提示符下执行gdbserver命令启动gdb server，默认端口为1234。

```
~/o/arm ❯❯❯ gdb main.o
(gdb) target remote localhost:1234
Remote debugging using localhost:1234
```

即可远程连接。

为实现从第一条指令开始控制程序运行，运行命令如下：

```
~/o/arm ❯❯❯ qemu-system-arm -S -M connex -drive file=flash.bin,if=pflash,format=raw  -nographic -serial /dev/null
QEMU 2.4.1 monitor - type 'help' for more information
(qemu) gdbserver
```

其中 -S即-singlestep，告诉qemu在执行第一条指令前停住。

2. 用QEMU调试ELF格式ARM程序

如ELF文件没有用到其它系统资源，如socket，设备等，可直接运行调试。

```
qemu-arm -singlestep -g 1234 file.bin
```

如需用到系统资源，则可用QEMU模拟操作系统，并在操作系统中部署gdbserver实现。

这里以调试GNU Hello为例：

```
$ wget http://ftp.gnu.org/gnu/hello/hello-2.6.tar.gz
$ tar xzf hello-2.6.tar.gz
$ cd hello-2.6
$ ./configure --host=arm-none-linux-gnueabi
$ make
$ cd ..
```

还需要在busybox文件系统中添加运行库支持，通过查看hello的依赖库：

```
$ arm-none-linux-gnueabi-readelf hello-2.6/src/hello -a |grep lib
[Requesting program interpreter: /lib/ld-linux.so.3]
0x00000001 (NEEDED)                     Shared library: [libgcc_s.so.1]
0x00000001 (NEEDED)                     Shared library: [libc.so.6]
00010694  00000216 R_ARM_JUMP_SLOT   0000835c   __libc_start_main
2: 0000835c     0 FUNC    GLOBAL DEFAULT  UND __libc_start_main@GLIBC_2.4 (2)
89: 0000844c     4 FUNC    GLOBAL DEFAULT   12 __libc_csu_fini
91: 0000835c     0 FUNC    GLOBAL DEFAULT  UND __libc_start_main@@GLIBC_
101: 00008450   204 FUNC    GLOBAL DEFAULT   12 __libc_csu_init
000000: Version: 1  File: libgcc_s.so.1  Cnt: 1
0x0020: Version: 1  File: libc.so.6  Cnt: 1
```

可以看到hello依赖库包括：“ld-linux.so.3”, “libgcc_s.so.1” 和 “libc.so.6”。

因此，需要拷贝相应链接库到busybox文件系统中。

```
$ cd busybox-1.17.1/_install
$ mkdir -p lib
$ cp /home/francesco/CodeSourcery/Sourcery_G++_Lite/arm-none-linux-gnueabi/libc/lib/ld-linux.so.3 lib/
$ cp /home/francesco/CodeSourcery/Sourcery_G++_Lite/arm-none-linux-gnueabi/libc/lib/libgcc_s.so.1 lib/
$ cp /home/francesco/CodeSourcery/Sourcery_G++_Lite/arm-none-linux-gnueabi/libc/lib/libm.so.6 lib/
$ cp /home/francesco/CodeSourcery/Sourcery_G++_Lite/arm-none-linux-gnueabi/libc/lib/libc.so.6 lib/
$ cp /home/francesco/CodeSourcery/Sourcery_G++_Lite/arm-none-linux-gnueabi/libc/lib/libdl.so.2 lib/
$ cp /home/francesco/CodeSourcery/Sourcery_G++_Lite/arm-none-linux-gnueabi/libc/usr/bin/gdbserver usr/bin/
$ cp ../../hello-2.6/src/hello usr/bin/
$ cd ../../
```

制作rcS自启动文件：

```
#!/bin/sh
mount -t proc none /proc
mount -t sysfs none /sys
/sbin/mdev -s
ifconfig lo up
ifconfig eth0 10.0.2.15 netmask 255.255.255.0
route add default gw 10.0.2.1
```

制作文件系统：

```
$ cd busybox-1.17.1/_install
$ mkdir -p proc sys dev etc etc/init.d
$ cp ../../rcS etc/init.d
$ chmod +x etc/init.d/rcS
$ find . | cpio -o --format=newc | gzip > ../../rootfs.img.gz
$ cd ../../
```

QEMU中运行该系统：

```
$ ./qemu-0.12.5/arm-softmmu/qemu-system-arm -M versatilepb -m 128M -kernel ./linux-2.6.35/arch/arm/boot/zImage -initrd ./rootf
```

进入提示符：

```
# gdbserver --multi 10.0.2.15:1234
```

```
$ ddd --debugger arm-none-linux-gnueabi-gdb
set solib-absolute-prefix nonexistantpath
set solib-search-path /home/francesco/CodeSourcery/Sourcery_G++_Lite/arm-none-linux-gnueabi/libc/lib/
file ./hello-2.6/src/hello
target extended-remote localhost:1234
set remote exec-file /usr/bin/hello
break main
run
```

3. 使用Qemu+gdb来调试内核

(1)编译调试版内核

对内核进行调试需要解析符号信息，所以得编译一个调试版内核。

```
$ cd linux-4.14
$ make menuconfig
$ make -j 20
```

这里需要开启内核参数CONFIG_DEBUG_INFO和CONFIG_GDB_SCRIPTS。GDB提供了Python接口来扩展功能，内核基于Python接口实现了一系列辅助脚本，简化内核调试

```
Kernel hacking  --->
   [*] Kernel debugging
   Compile-time checks and compiler options  --->
       [*] Compile the kernel with debug info
       [*]   Provide GDB scripts for kernel debugging
```

(2)构建initramfs根文件系统

Linux系统启动阶段，boot
loader加载完内核文件vmlinuz后，内核紧接着需要挂载磁盘根文件系统，但如果此时内核没有相应驱动，无法识别磁盘，就需要先加载驱动，而驱动又位于/lib/module
loader加载initramfs到内存中，内核会将其挂载到根目录/,然后运行/init脚本，挂载真正的磁盘根文件系统。

这里借助Busybox构建极简initramfs，提供基本的用户态可执行程序。

编译BusyBox，配置CONFIG_STATIC参数，编译静态版BusyBox，编译好的可执行文件busybox不依赖动态链接库，可以独立运行，方便构建initramfs。

```
$ cd busybox-1.28.0
$ make menuconfig
Settings  --->
   [*] Build static binary (no shared libs)
$ make -j 20
$ make install
```

会安装在_install目录:

```
$ ls _install
bin  linuxrc  sbin  usr
```

建initramfs，其中包含BusyBox可执行程序、必要的设备文件、启动脚本init。这里没有内核模块，如果需要调试内核模块，可将需要的内核模块包含进来。init脚本只挂

```
$ mkdir initramfs
$ cd initramfs
$ cp ../_install/* -rf ./
$ mkdir dev proc sys
$ sudo cp -a /dev/{null, console, tty, tty1, tty2, tty3, tty4} dev/
$ rm linuxrc
$ vim init
$ chmod a+x init
$ ls
$ bin   dev  init  proc  sbin  sys   usr
```

init文件内容：

```
#!/bin/busybox sh
mount -t proc none /proc
mount -t sysfs none /sys

exec /sbin/init
```

打包initramfs:

```
$ find . -print0 | cpio --null -ov --format=newc | gzip -9 > ../initramfs.cpio.gz
```

(3)调试

启动内核：

```
$ qemu-system-x86_64 -s -kernel /path/to/vmlinux -initrd initramfs.cpio.gz -nographic -append "console=ttyS0"
```

-initrd指定制作的initramfs。

由于系统自带的GDB版本为7.2，内核辅助脚本无法使用，重新编译了一个新版GDB：

```
$ cd gdb-7.9.1
$ ./configure --with-python=$(which python2.7)
$ make -j 20
```

```
$ sudo make install
```

启动GDB:

```
$ cd linux-4.14
$ /usr/local/bin/gdb vmlinux
(gdb) target remote localhost:1234
```

使用内核提供的GDB辅助调试功能：

```
(gdb) apropos lx
function lx_current -- Return current task
function lx_module -- Find module by name and return the module variable
function lx_per_cpu -- Return per-cpu variable
function lx_task_by_pid -- Find Linux task by PID and return the task_struct variable
function lx_thread_info -- Calculate Linux thread_info from task variable
function lx_thread_info_by_pid -- Calculate Linux thread_info from task variable found by pid
lx-cmdline --  Report the Linux Commandline used in the current kernel
lx-cpus -- List CPU status arrays
lx-dmesg -- Print Linux kernel log buffer
lx-fdtdump -- Output Flattened Device Tree header and dump FDT blob to the filename
lx-iomem -- Identify the IO memory resource locations defined by the kernel
lx-ioports -- Identify the IO port resource locations defined by the kernel
lx-list-check -- Verify a list consistency
lx-lsmod -- List currently loaded modules
lx-mounts -- Report the VFS mounts of the current process namespace
lx-ps -- Dump Linux tasks
lx-symbols -- (Re-)load symbols of Linux kernel and currently loaded modules
lx-version --  Report the Linux Version of the current kernel
(gdb) lx-cmdline
console=ttyS0
```

在函数cmdline_proc_show设置断点，虚拟机中运行cat /proc/cmdline命令即会触发。

上一篇：OJ-level5——mmap和m... 下一篇：OJ-level5——mmap和m...

1. 0 条回复
   - 动动手指，沙发就是你的了！