ATTACK DEFENSE

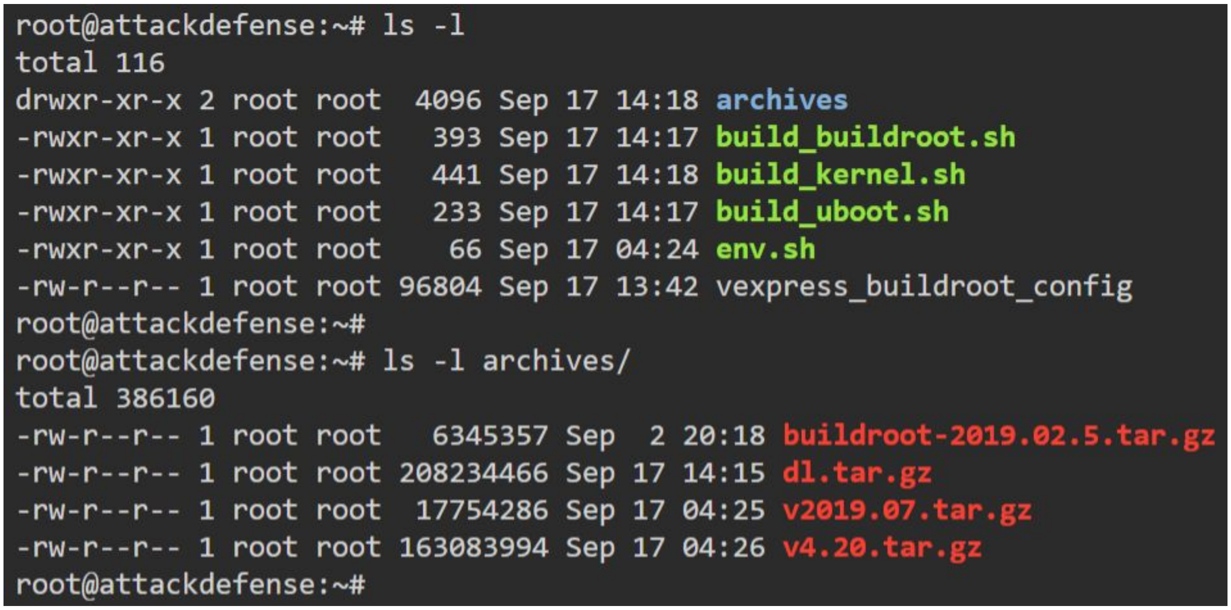
by PentesterAcademy

Name	Build Lab: ARM vexpress Board
URL	https://www.attackdefense.com/challengedetails?cid=1233
Type	IoT : Bootloader

Important Note: This document illustrates all the important steps required to complete this lab. This is by no means a comprehensive step-by-step solution for this exercise. This is only provided as a reference to various commands needed to complete this exercise and for your further research on this topic.

Objective I: Compile U-boot, kernel and file system for the board.

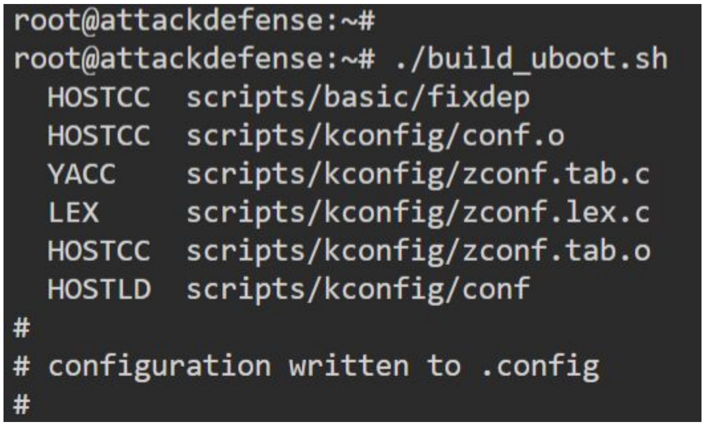
Step 1: The build scripts are present in the /root directory. And all the source archives are present in /root/archives directory.



```
root@attackdefense:~# ls -l
total 116
drwxr-xr-x 2 root root  4096 Sep 17 14:18 archives
-rwxr-xr-x 1 root root   393 Sep 17 14:17 build_buildroot.sh
-rwxr-xr-x 1 root root   441 Sep 17 14:18 build_kernel.sh
-rwxr-xr-x 1 root root   233 Sep 17 14:17 build_uboot.sh
-rwxr-xr-x 1 root root    66 Sep 17 04:24 env.sh
-rw-r--r-- 1 root root 96804 Sep 17 13:42 vexpress_buildroot_config
root@attackdefense:~#
root@attackdefense:~# ls -l archives/
total 386160
-rw-r--r-- 1 root root   6345357 Sep  2 20:18 buildroot-2019.02.5.tar.gz
-rw-r--r-- 1 root root 208234466 Sep 17 14:15 dl.tar.gz
-rw-r--r-- 1 root root  17754286 Sep 17 04:25 v2019.07.tar.gz
-rw-r--r-- 1 root root 163083994 Sep 17 04:26 v4.20.tar.gz
root@attackdefense:~#
```

Step 2: Run build_uboot.sh to compile the U-boot.

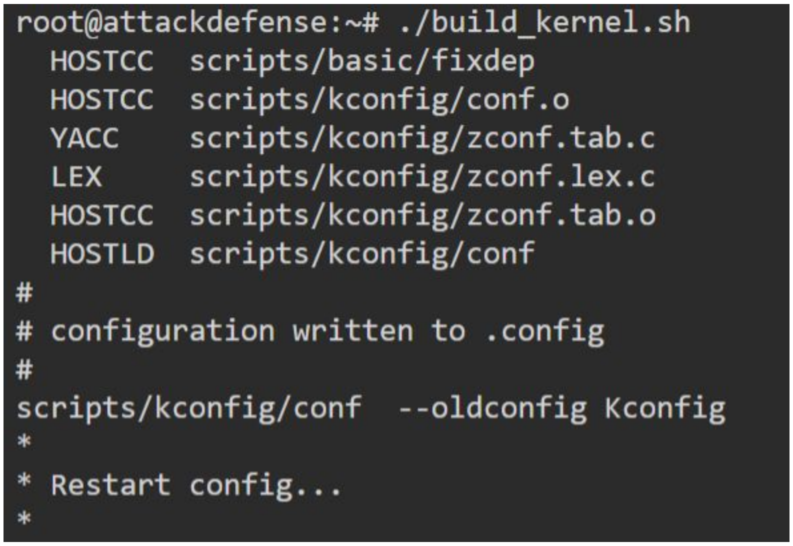
Command: ./build_uboot.sh



```
root@attackdefense:~#
root@attackdefense:~# ./build_uboot.sh
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  YACC    scripts/kconfig/zconf.tab.c
  LEX     scripts/kconfig/zconf.lex.c
  HOSTCC  scripts/kconfig/zconf.tab.o
  HOSTLD  scripts/kconfig/conf
#
# configuration written to .config
#
```

Step 3: Run build_kernel.sh to compile the Kernel and Device Tree Blob (Flat Device Tree).

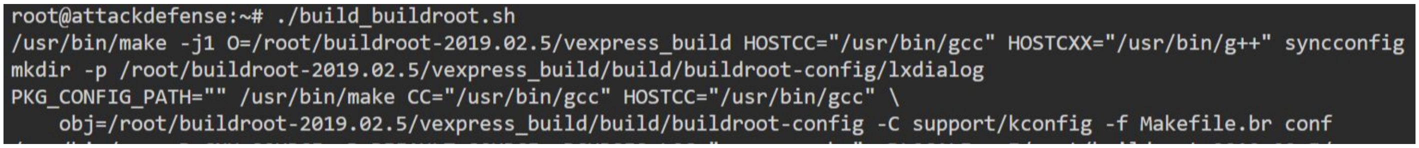
Command: ./build_kernel.sh



```
root@attackdefense:~# ./build_kernel.sh
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  YACC    scripts/kconfig/zconf.tab.c
  LEX     scripts/kconfig/zconf.lex.c
  HOSTCC  scripts/kconfig/zconf.tab.o
  HOSTLD  scripts/kconfig/conf
#
# configuration written to .config
#
scripts/kconfig/conf  --oldconfig Kconfig
*
* Restart config...
*
```

Step 4: Run build_buildroot.sh to create the file system image.

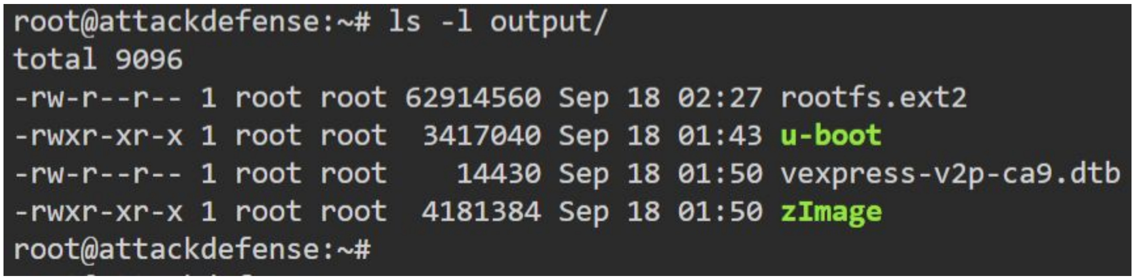
Command: ./build_buildroot.sh



```
root@attackdefense:~# ./build_buildroot.sh
/usr/bin/make -j1 O=/root/buildroot-2019.02.5/vexpress_build HOSTCC="/usr/bin/gcc" HOSTCXX="/usr/bin/g++" syncconfig
mkdir -p /root/buildroot-2019.02.5/vexpress_build/build/buildroot-config/lxdialog
PKG_CONFIG_PATH="" /usr/bin/make CC="/usr/bin/gcc" HOSTCC="/usr/bin/gcc" \
    obj=/root/buildroot-2019.02.5/vexpress_build/build/buildroot-config -C support/kconfig -f Makefile.br conf
```

Step 5: Check /root/output directory to find the output of these scripts. Change to the output directory and list the components.

Commands:
cd output
ls -l



```
root@attackdefense:~# ls -l output/
total 9096
-rw-r--r-- 1 root root 62914560 Sep 18 02:27 rootfs.ext2
-rwxr-xr-x 1 root root  3417040 Sep 18 01:43 u-boot
-rw-r--r-- 1 root root    14430 Sep 18 01:50 vexpress-v2p-ca9.dtb
-rwxr-xr-x 1 root root  4181384 Sep 18 01:50 zImage
root@attackdefense:~#
```

Objective II: Use compiled components to start the emulation. Check the contents of the root file system.

There are two ways of doing this.

Option 1: One can build all components and switch to output directory. As all built files are dropped in /root/output directory, the machine can be started from there.

Option 2: If user doesn't want to build the components, he can use pre-built components available in /root/pre-created directory. The user has to change to the directory and start the machine from there.

After switching to the correct directory as mentioned in Option 1 and 2, the remaining process is same.

Run the emulation from the output directory..

Command: qemu-system-arm -M vexpress-a9 -m 512M -dtb vexpress-v2p-ca9.dtb -kernel zImage -initrd rootfs.ext2 -append "console=ttyAMA0 console=tty0 root=/dev/ram rw" -serial mon:stdio -nographic

Here,

-M vexpress-a9: Virtual machine selection
For more on Vexpress: https://crux-arm.nu/SupportedDevices/Vexpress

-m 512 : Memory to be allocated to virtual machine

-dtb vexpress-v2p-ca9.dtb: Device Tree Blob (Flat Device Tree) to use

-kernel zImage: Linux kernel image to use

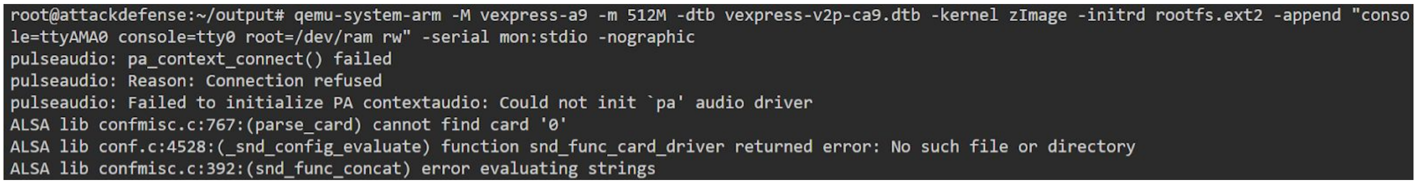
-initrd rootfs.ext2: Root filesystem image to use as Initial RAM disk

-append : to define Boot parameters (or arguments to the kernel)

- console=ttyAMA0: Redirect first serial port (on ARM architecture) to current session
- console=tty0: Redirect Qemu virtual serial port to current session
- root=/dev/ram: RAM device location
- rw : Mounting disk image in read/write mode

-serial mon:stdio: Used when the virtual serial port and QEMU monitor are multiplexed onto the same console device. One can use “Ctrl-a c” to switch among these two modes

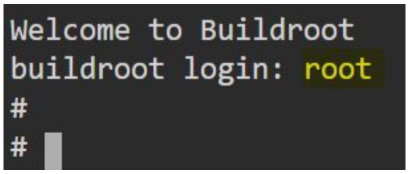
-nographic : To invoke qemu from CLI



```
root@attackdefense:~/output# qemu-system-arm -M vexpress-a9 -m 512M -dtb vexpress-v2p-ca9.dtb -kernel zImage -initrd rootfs.ext2 -append "conso
le=ttyAMA0 console=tty0 root=/dev/ram rw" -serial mon:stdio -nographic
pulseaudio: pa_context_connect() failed
pulseaudio: Reason: Connection refused
pulseaudio: Failed to initialize PA contextaudio: Could not init `pa' audio driver
ALSA lib confmisc.c:767:(parse_card) cannot find card '0'
ALSA lib conf.c:4528:(_snd_config_evaluate) function snd_func_card_driver returned error: No such file or directory
ALSA lib confmisc.c:392:(snd_func_concat) error evaluating strings
```

The machine will start and after going through boot sequence, eventually present console login to the user. The user has to use the following credentials:

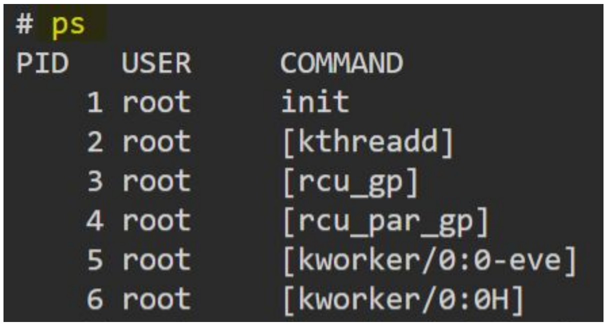
Username: root
Password: <none>



```
Welcome to Buildroot
buildroot login: root
#
#
```

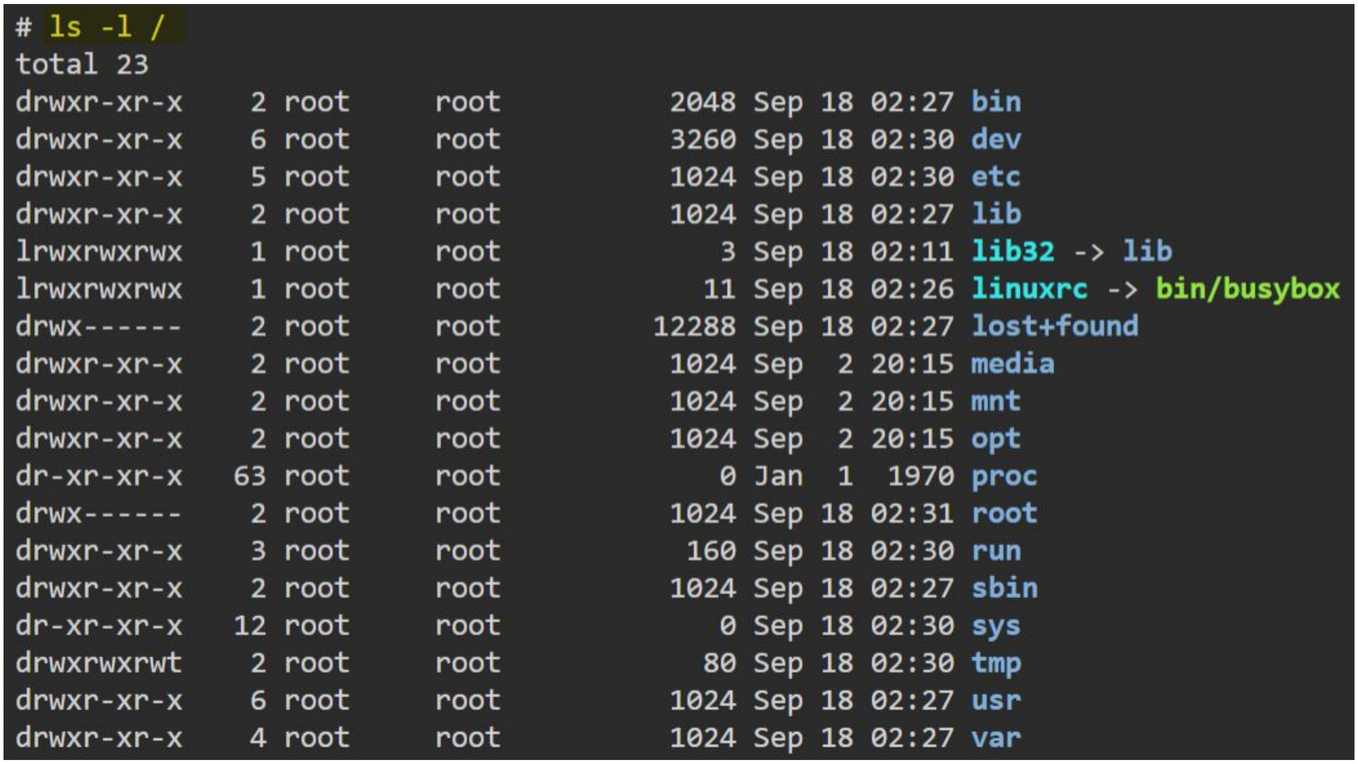
After logging into the machine, the user can run common Linux commands.

Command: ps



```
# ps
PID   USER     COMMAND
    1 root     init
    2 root     [kthreadd]
    3 root     [rcu_gp]
    4 root     [rcu_par_gp]
    5 root     [kworker/0:0-eve]
    6 root     [kworker/0:0H]
```

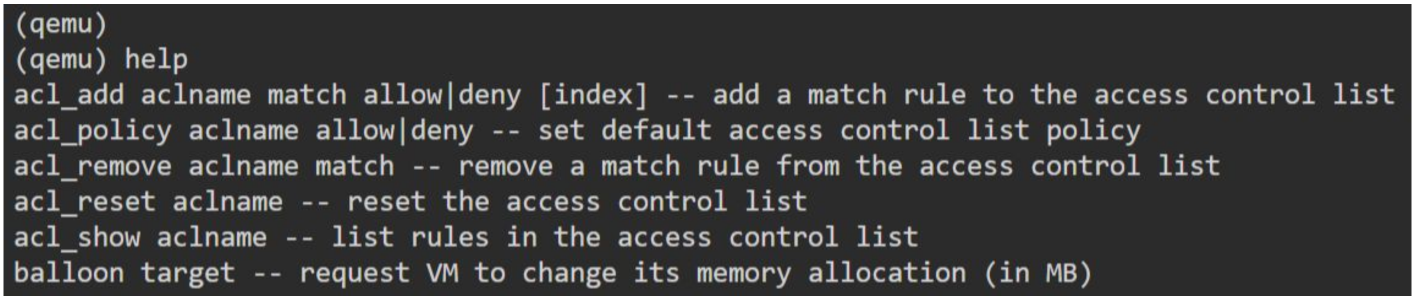
Command: ls -l /



```
# ls -l /
total 23
drwxr-xr-x    2 root     root          2048 Sep 18 02:27 bin
drwxr-xr-x    6 root     root          3260 Sep 18 02:30 dev
drwxr-xr-x    5 root     root          1024 Sep 18 02:30 etc
drwxr-xr-x    2 root     root          1024 Sep 18 02:27 lib
lrwxrwxrwx    1 root     root             3 Sep 18 02:11 lib32 -> lib
lrwxrwxrwx    1 root     root            11 Sep 18 02:26 linuxrc -> bin/busybox
drwx------    2 root     root         12288 Sep 18 02:27 lost+found
drwxr-xr-x    2 root     root          1024 Sep  2 20:15 media
drwxr-xr-x    2 root     root          1024 Sep  2 20:15 mnt
drwxr-xr-x    2 root     root          1024 Sep  2 20:15 opt
dr-xr-xr-x   63 root     root             0 Jan  1  1970 proc
drwx------    2 root     root          1024 Sep 18 02:31 root
drwxr-xr-x    3 root     root           160 Sep 18 02:30 run
drwxr-xr-x    2 root     root          1024 Sep 18 02:27 sbin
dr-xr-xr-x   12 root     root             0 Sep 18 02:30 sys
drwxrwxrwt    2 root     root            80 Sep 18 02:30 tmp
drwxr-xr-x    6 root     root          1024 Sep 18 02:27 usr
drwxr-xr-x    4 root     root          1024 Sep 18 02:27 var
```

The emulation is successfully booted and one can inspect the contents of the file system.

As mentioned above, one can switch to Qemu console using “Ctrl-a c”



```
(qemu)
(qemu) help
acl_add aclname match allow|deny [index] -- add a match rule to the access control list
acl_policy aclname allow|deny -- set default access control list policy
acl_remove aclname match -- remove a match rule from the access control list
acl_reset aclname -- reset the access control list
acl_show aclname -- list rules in the access control list
balloon target -- request VM to change its memory allocation (in MB)
```

References:

- U-boot source: https://github.com/u-boot/u-boot/archive/v2019.07.tar.gz
- Kernel source: https://github.com/torvalds/linux/archive/v4.20.tar.gz
- Buildroot source: https://buildroot.org/downloads/buildroot-2019.02.5.tar.gz