KaliPI custom build

Cross-compile ARM (Rpi2) on a Kali 2.0 host VM or PC

- and gain 256 MB RAM on your Rpi2 -

CyberSec2k's copy-paste recipe, update n.459, 09/03/2015

Table of Contents

When I started to build my custom Kali ARM image for the Raspberry PI 2, I spent a lot of time looking for info and failing lengthy builds. So I kept these notes as an exercise to understand the process and speed up further builds.

I like cut-paste recipes for my experiments. By executing every step manually I gain better knowledge and control over what I'm doing, so I can manage over time the evolution of my needs, computing environment, external tools.

Only if and when it will be worthwhile, I will eventually automate the process with a script, however most of the time cut-paste is enough.

Dedicated CyberSecNow blog post: http://cybersecnow.blogspot.it/2015/09/kalipi-custom-build-how-to-cross.html

Dedicated Kali forum post: https://forums.kali.org/showthread.php?27249-Rpi2-Kali-2-0-1-how-to-get-almost-all-the-1GB-RAM-instead-of-only-750MB

Updated version of this how-to and related files: https://github.com/cybersec2k/kalipi-kernel

Typing conventions:

Yellow – things you may want to customize

Green – things to pay attention to

`Monospace` – code

1) Build environment

The recipe described here has been tested using the following build environment:

- Kali 2.0 stock i686 VM with at least 20GB free hard disk space and 2GB RAM. I tested it also on my Kali 2.0 notebook (I7, 8GB RAM), but with an i686 VM it's easier for more people to reproduce the whole process.
- A Raspberry PI 2 model B (1GB RAM), an empty 8GB+ microSD card, a microSD to SD adapter to write the microSD on the notebook.

2) One-time setup ARM cross-compile environment

Source: http://docs.kali.org/development/arm-cross-compilation-environment

NOTE: if you follow the above guide, skip **`apt-get install ia32-libs`**. Since debian Weezy **`ia32-libs`** doesn't exists anymore, required libs install automatically by using **`dpkg --add-architecture i386`**

```
# cd ; mkdir Rpi2 ; cd Rpi2
# dpkg --add-architecture i386
# apt-get update && sudo apt-get dist-upgrade
# mkdir -p arm-stuff/kernel/toolchains
# cd arm-stuff/kernel/toolchains
# git clone http://github.com/offensive-security/gcc-arm-eabi-linaro-4.6.2.git
# cd ../../..
# git clone https://github.com/offensive-security/kali-arm-build-scripts.git
# cd kali-arm-build-scripts
# ./build-deps.sh
```

3) Setup apt-cacher-ng (optional)

On further builds **`apt-cacher-ng`** saves about 500MB worth of download from repos. With a good DSL internet connection it's not a big deal, as it saves only few minutes out of a couple of hours.

```
# apt-get install apt-cacher-ng              One-time installation

# nano /etc/apt/apt.conf.d/10aptcacher       Comment this for normal operations
Acquire::http { Proxy "http://127.0.0.1:3142/"; };

# /etc/init.d/apt-cacher-ng start            You may want to start it only before kernel builds
# ps aux|grep apt-cacher | grep -v grep      Check it started

# du -hs /var/cache/apt-cacher-ng/           Get cache current size

# rm -rf /var/cache/apt-cacher-ng/*          Clean cache
```

If you use the standard **`rpi2.sh`** build script, you must search it for **`http_proxy`** and uncomment as indicated.

Statistics and management: http://localhost:3142/acng-report.html

4) Customize build script and kernel .config

a) Locate and cd where there is **rpi.sh**, usually:
 # cd ~/Rpi2/kali-arm-build-scripts

b) Get custom files from github:
 # wget https://raw.githubusercontent.com/cybersec2k/kalipi-kernel/master/rpi2-KaliPI.sh
 # chmod +x rpi2-KaliPI.sh
 # wget https://raw.githubusercontent.com/cybersec2k/kalipi-kernel/master/rpi2-KaliPI.config
 # wget https://raw.githubusercontent.com/cybersec2k/kalipi-kernel/master/rpi2-KaliPI-3.18.config.diff
 # patch kernel-configs/rpi2-3.18.config -o kernel-configs/rpi2-KaliPI-3.18.config < rpi2-KaliPI-3.18.config.diff

a) Notes on CyberSec2k's build recipe: rpi2-KaliPI.sh

rpi2-KaliPI.sh is based on **rpi2.sh** as of Sep 5, 2015:
https://github.com/offensive-security/kali-arm-build-scripts/blob/master/rpi2.sh

Main changes over Kali stock build script are:

- Start by loading the build script configuration **rpi2-KaliPI.config**
- Make sure that the cross compiler can be found in the path.
- Completely remove the previous build directory.
- Use apt-cacher-ng depending on the build script configuration.
- Added some **sync** to solve some pesky time-dependent issues.
- Light modifications to the kernel build section.
- Create the boot partition **config.txt** file.
- Other minor changes, look at the differences with a nice gui using:
  # meld rpi2.sh rpi2-KaliPI.sh

b) **rpi2-KaliPI.config** example:

```
homedir=/root/Rpi2
useaptcacher=NO
kernelconf=rpi2-KaliPI-3.18.config
kalipirelease=150905kp
ARCH=arm
CROSS_COMPILE=${homedir}/arm-stuff/kernel/toolchains/gcc-arm-eabi-linaro-4.6.2/bin/arm-eabi-
```

c) Kernel recipe: reduce RAM dedicated to GPU from 256MB to 16MB, giving 976MB to the CPU instead of 750MB

WARNING: while 256MB more memory is a big deal for most usages, it may break SonicPI (http://sonic-pi.net/) and few other tricky apps.

Patch file: **kernel-configs/rpi2-KaliPI-3.18.config.diff**

To apply it:
 # patch kernel-configs/rpi2-3.18.config -o kernel-configs/rpi2-KaliPI-3.18.config < kernel-configs/rpi2-KaliPI-3.18.config.diff

```
446c446,447
< # CONFIG_HIGHMEM is not set
---
> CONFIG_HIGHMEM=y
```

```
> # CONFIG_HIGHPTE is not set
460a462
> CONFIG_BOUNCE=y
3516d3517
< CONFIG_MMC_SPI=m
4154a4156
> # CONFIG_DEBUG_HIGHMEM is not set
```

Interesting post, even if it does not apply to kernel 3.18:
https://raspberrypi.stackexchange.com/questions/24092/kernel-config-necessary-options

5) Build

a) Configure build

```
# nano rpi2-KaliPI.config            Configure your parameters (home, cacher, release, ...)
```

WARNING: kernel patches version must match kernel version, look for the string **`patch`** inside the build script.

b) Last checks... before the rocket launch

```
# . rpi2-KaliPI.config
# arm-eabi-gcc –version
If “command not found”, update PATH:
# export PATH=${PATH}:${homedir}/arm-stuff/kernel/toolchains/gcc-arm-eabi-linaro-4.6.2/bin
# arm-eabi-gcc --version

# ping -c 3 security.kali.org        Check repos are reachable
```

c) Execute

Cross your fingers, hope for stable internet connection, functional remote servers, naughty aliens far away from your computer.

```
# time sudo PS4='`date +%T` Line ${LINENO}: ' bash -x rpi2-KaliPI.sh ${kalipirelease} 2>&1 | tee rpi2-${kalipirelease}.log
```

The build script should run unattended for one hour or even more, depending on the speed of your computer, internet connection, external services. When finished, you should look for errors in the build log.

6) Check the build log

You may look at the build log **`rpi2-${kalipirelease}.log`**
It includes script line numbers and each expanded command executed.

My build script recipe includes date/time commands to pinpoint which commands consume the most of time, so one can try to optimize them and measure the benefit of using apt-cacher-ng.

Apt-cacher-ng status:

- Cache size: `$ du -hs /var/cache/apt-cacher-ng/`
- Hit/miss statistics and management: http://localhost:3142/acng-report.html

```
# . rpi2-KaliPI.config
# grep -i -e abort -e fail -e error -e 'no such file' -e 'not found' rpi2-${kalipirelease}.log | grep -vi -e libgpg-error -e liberror -e libkmod-module.c -e localized-error-pages -e 'Failed to dump keymap' -e ' CC ' -e ' AS ' -e '_failure_' -e warning
```

7) Create a microSD with the customized KaliPI

The build script compresses the generated image only on 64bit machines, as **pixz** does not have enoug resources on 32bit machines, and **xz** takes a lot of time to do its job.

On a recent I7 8GB RAM notebook, **pixz** took 5 minutes, while on the Kali 2.0 VM with 2GB RAM used for this tutorial (hosted on the above notebook) **xz** took 48 minutes:

```
# time xz -z kali-150905-rpi2.img kali-150905-rpi2.img.xz
```

a) Check the image integrity

```
# sha1sum -c kali-150904-rpi2.img.xz.sha1sum
```

b) Copy the image on the microSD

Insert the microSD in the host PC, probably using an SD adapter or a generally slower USB adapter.

```
# fdisk -l                    Locate this SD, usually you will find it on /dev/mmcblk0p1 or /dev/sdb
# mount | grep -e '/sd' -e '/mmc'    Look for this SD mounted partitions, then unmount all of them
# umount /dev/mmcblk0p1
# umount /dev/mmcblk0p2
# time xzcat kali-150904-rpi2.img.xz | sudo dd of=/dev/mmcblk0 bs=16M
# gparted /dev/mmcblk0        Resize the ext4 partition to max
# sync
```

On a modern notebook with SD slot, dd takes about 10 minutes using a Samsung EVO HCI 16GB microSD (12.5 MB/s):
http://www.amazon.com/Samsung-Memory-MicroSDHC-without-Adapter/dp/B00J2973JG

Insert the microSD in the Rpi, connect HDMI, keyboard and ethernet cables, cross your fingers and power-on the Rpi.

If the Rpi is wired to the LAN, you cand try to find it using **nmap -sP *YourIpSubnet***
My Rpi ethernet has OUI= B8:27:EB (MAC address B8:27:EB:XX:XX:XX).
Kali by default enables **ssh** and allows **root** password login, so you can connect to the Rpi using **ssh root@*YourPiIpAddress*** - the stock password is **toor** (change it ASAP).

8) Mission accomplished

Well, by using this cut-paste guide I was able to customize and rebuild Kali 2.0 for the Raspberry PI 2 model B (1GB RAM), including a customized kernel which gives to the CPU almost all the 1GB RAM available, instead of 750MB, by dedicating only 16M to the GPU.

Here is the result:

```
root@KaliPI:~# free
             total       used       free     shared    buffers     cached
Mem:        998136     152244     845892      13168      13464      84704
-/+ buffers/cache:      54076     944060
Swap:            0          0          0
```

By disabling the **gpu_mem=16** directive in the boot partition **config.txt** the free RAM is **48K** less, however considerably more than the original **750M**:

```
root@KaliPI:~# free
             total       used       free     shared    buffers     cached
```

```
Mem:        949368     126384     822984       6624      12816      60348
-/+ buffers/cache:      53220     896148
Swap:            0          0          0
```

The Kali 2.0.1 for Raspberry PI 2 stock build reports:

```
root@KaliPI-247ADC:~# free
             total       used       free     shared    buffers     cached
Mem:        762468      97792     664676       5416      10276      44072
-/+ buffers/cache:      43444     719024
Swap:            0          0          0
```

NOTE: here there is 97M instead of 126M used RAM because here I disabled the graphical login.

9) Cleanup temporary files

If all went ok and you don't work on the built image anymore, it's time to cleanup temporary files:

```
# . rpi2-KaliPI.config
# export basedir=${homedir}/kali-arm-build-scripts/rpi2-${kalipirelease}
# ls -l ${basedir}
# rm -rf ${basedir}/kernel ${basedir}/bootp ${basedir}/root ${basedir}/kali-armhf ${basedir}/boot ${basedir}/patches
```

- o -