

# 使用Docker编译Android系统内核

研究Android软件安全，很多时候需要修改与编译Android系统的源代码，对于系统层面的安全加固与定制，则少不了需要与Android内核打交道。

Android系统的内核源自于GNU Linux，且它的内核架构以及基础设施都与Linux保持着一致。为了让Linux内核更好的适应Android系统，Android官方裁剪与增强了Linux内核的部分功能，比如增加了binder组件通信的驱动、Android日志系统的驱动，低内存管理器、Yaffs2文件系统等特性。而在内核模块的编译与开发方面，它保持着与Linux系统几乎相同的流程。

## 准备编译环境

Android官方推荐的系统编译环境是Linux平台的Ubuntu系统。Android系统的源代码可以在macOS与Ubuntu系统上原生的编译成功，但系统内核的代码并不能直接在macOS与Windows系统上编译。对于常年使用macOS与Windows系统的用户来说，优先的选择方案是购置一台Ubuntu系统的开发PC，或者在现有的开发环境中，安装Ubuntu系统的虚拟机。

Docker 的出现，让事情变得简单有趣起来。这个支持多平台的Linux系统虚拟化软件，几乎支持所有的Linux系统的发行版本。使用它来安装与部署Linux系统环境，可以在编译Android系统内核代码这件事情上，省掉大量的时间与精力。

下面，我们打算使用 Docker，配合Ubuntu 16.04版本的系统，来编译Android系统内核的源代码。

## 安装Docker

Docker 可以到其官网下载安装免费的社区版本。地址是：<https://www.docker.com/>。

Docker 原生支持在macOS系统上虚拟化Linux系统，不需要安装 virtualbox 来进行“桥接”（作为可选项，也可以安装 virtualbox 进行“桥接”）。在Windows系统上，目前最新的Windows 10版本配合支持VT技术的CPU，也可以做到原生级别的虚拟化。

在 macOS 系统上，最简单的方法是执行下面的命令进行安装：

```
$ brew install docker
```

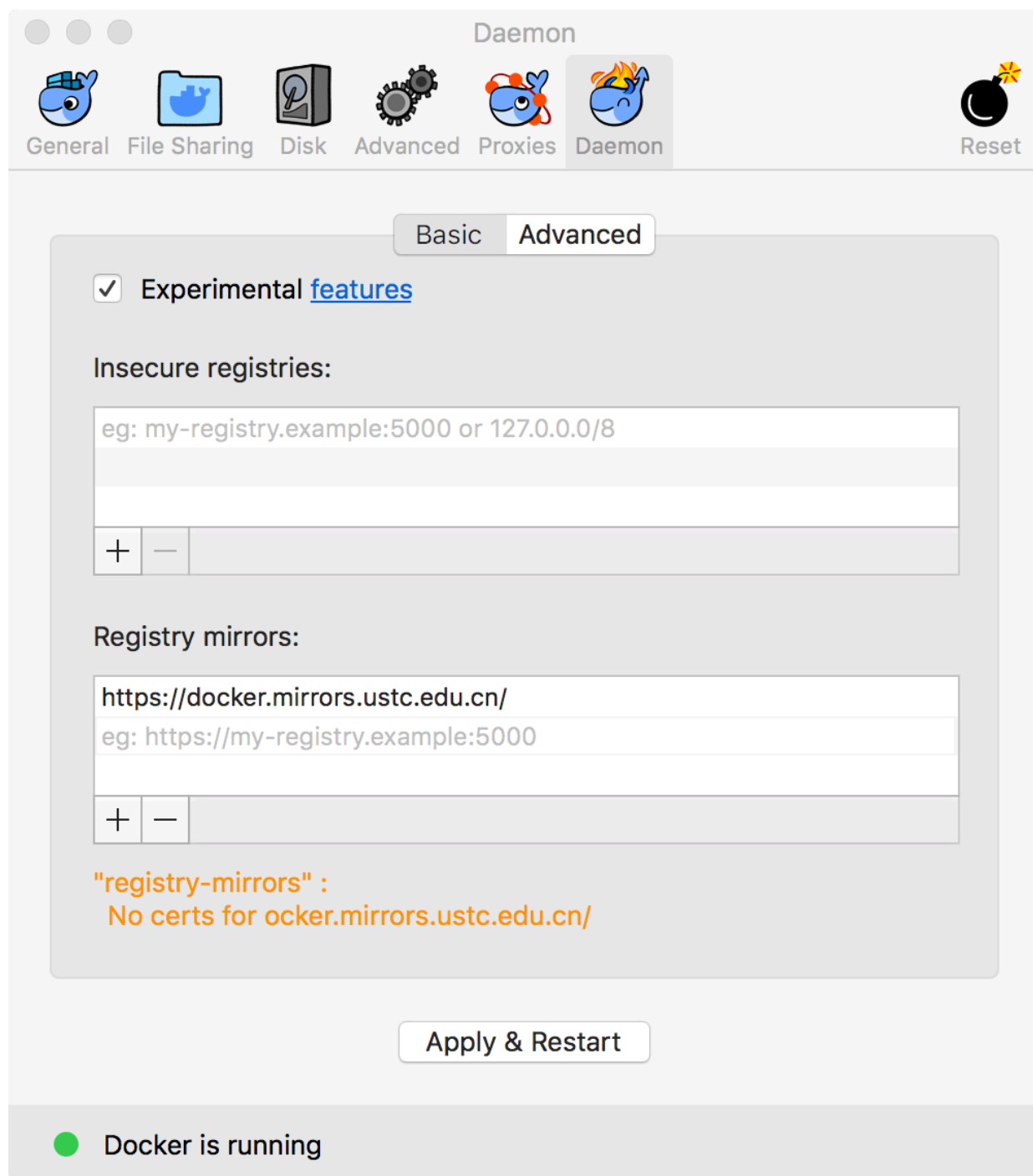
这样安装好的 Docker 是命令行版本的，如果想要安装GUI版本的 Docker，即官方的“Docker for macOS”，执行下面的命令即可：

```
$ brew cask install docker
```

安装好 Docker 后，就可以执行 `docker pull` 命令从 Docker 下载Linux系统的镜像了。

但由于一些众所周知的原因，Docker 的网络访问受限，并不能很好的在国内直接使用。最好的方案是使用第三方的镜像服务器来下载 Docker 镜像。

以中国科技大学的镜像为例：打开“Docker.app”，点击“Preferences...”，进入偏好设置页面，点击“Daemon”选项卡，在“Registry mirrors”中添加“<https://docker.mirrors.ustc.edu.cn/>”，最后，点击下方的“Apply & Restart”按钮重启生效。如图所示：



如果你使用的是命令行版本的 Docker ，由手动修改“`~/.docker/daemon.json`”文件，添加“registry-mirrors”的值为“<https://docker.mirrors.ustc.edu.cn/>”，然后重启 Docker 即可，如下所示：

```
$ cat ~/.docker/daemon.json
{
  "debug" : true,
  "registry-mirrors" : [
    "https://docker.mirrors.ustc.edu.cn/"
  ],
  "experimental" : true
}
```

完整后，可以执行 `docker info`，查看输出信息中“Registry Mirrors:”一项的值是否为设置好的服务器地址。

执行 `docker run hello-world`，运行一个 Docker 版本的“Hello World”。如果输出类似如下，说明安装与配置成功了。

```
$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs
the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent
it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

## 下载Android内核代码

与AOSP一样，Android的内核代码也是开源的，在Android的官网服务器上，有着属于内核的专门代码仓库。同时，不同系统的版本由不同的分支进行更新与维护。

Android系统的内核与具体的设备型号与版本相关，这里要编译的是Android模拟器的内核代码。它位于一个叫goldfish的内核代码分支中。Android 4.4低版本与6.0、7.0版本使用的是3.4与3.10版本的内核。这里就以这两个版本的代码为例，进行编译。

goldfish的内核代码主分支是：<https://android.googlesource.com/kernel/goldfish>。

如图所示，可以看到主线上有很多分支，其中就包含用到的3.4与3.10版本。

## [android](#) / [kernel](#) / **goldfish**

### Branches

[master](#)

[android-3.10](#)

[android-3.18](#)

[android-4.14](#)

[android-4.4](#)

[android-goldfish-2.6.29](#)

[android-goldfish-3.10](#)

[android-goldfish-3.10-k-dev](#)

[android-goldfish-3.10-l-mr1-dev](#)

[android-goldfish-3.10-m-dev](#)

[android-goldfish-3.10-n-dev](#)

[android-goldfish-3.18](#)

[android-goldfish-3.18-dev](#)

[android-goldfish-3.4](#)

[android-goldfish-3.4-l-mr1-dev](#)

[android-goldfish-4.14-dev](#)

[android-goldfish-4.4-dev](#)

[android-goldfish-4.9-dev](#)

[heads/for/android-goldfish-3.18-dev](#)

[linux-goldfish-3.0-wip](#)

这个时候，正规的做法是执行如下命令下载代码：

```
$ git clone https://android.googlesource.com/kernel/goldfish -b android-goldfish-3.4
```

但由于一些众所周知的原因，这样并不能下载到内核的代码，这里同样选择了镜像服务器来完成这个操作。执行如下命令：

```
$ git clone https://aosp.tuna.tsinghua.edu.cn/kernel/goldfish -b android-goldfish-3.4 android-goldfish-3.4
Cloning into 'android-goldfish-3.4'...
remote: Counting objects: 5954665, done.
remote: Total 5954665 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (5954665/5954665), 1.26 GiB | 453.00 KiB/s, done.
Resolving deltas: 100% (5010208/5010208), done.
Checking out files: 100% (38922/38922), done.

$ git clone https://aosp.tuna.tsinghua.edu.cn/kernel/goldfish -b android-goldfish-3.10 android-goldfish-3.10
Cloning into 'android-goldfish-3.10'...
```

这一次，我们感谢清华的镜像服务器，让我们顺利的下载到了两个版本的内核代码。

## Android NDK

接下来是配置编译器了。在AOSP中编译Android内核有“platform/built/ndk”目录下提供的NDK，这里由于脱离了AOSP的环境，需要手动的配置Android NDK。这里选择从官网下载一份Linux版本的NDK，版本由于需要兼容到低版本的内核，选择了Android ndk r10e。下载地址是：[https://dl.google.com/android/repository/android-ndk-r10e-linux-x86\\_64.zip](https://dl.google.com/android/repository/android-ndk-r10e-linux-x86_64.zip)

下载下来，解压即可。

## 编写Dockerfile

现在来编写 `Docker` 的配置文件Dockerfile。首先是系统环境，选择Ubuntu 16.04。那么我们新建Dockerfile文件，加入如下行：

```
#Base image
FROM ubuntu:16.04
```

第一行是注释，第二行指明基础镜像为Ubuntu的16.04版本。

现在写个版权信息呗，如下：

```
#Labels and Credits
LABEL \
    name="docker-android-kernel" \
    author="fei_cong <fei_cong@hotmail.com>" \
    maintainer="fei_cong <fei_cong@hotmail.com>" \
    contributor_1="fei_cong <fei_cong@hotmail.com>" \
    description="Docker image for building Android goldfish kernel."
```

接下来是安装编译内核时需要用到的命令行工具，首先是 `build-essential`，这是Ubuntu系统上的编译器套件，包含了编译器 `gcc` 与其依赖的所有组件库。然后是 `make` 命令，它负责编译内核提供的Makefile文件。安装完这两个就可以编译3.4版本的内核了，3.10版本的内核还需要用到一个 `bc` 命令，这是一个GNU命令行的计算器程序。最后，添加一行 `CMD ["bash"]` 作为最后镜像配置好后启

动的第一条命令。

## 编译

所有的准备工作完成后，接下来是编译环节。

### 编译Dockerfile

首先需要通过Dockerfile编译出一个Ubuntu 16.04的系统镜像。切换到Dockerfile所在的目录，执行如下命令：

```
$ docker build -t tiantian_android_kernel .
Sending build context to Docker daemon 392.7kB
Step 1/4 : FROM ubuntu:16.04
16.04: Pulling from library/ubuntu
3b37166ec614: Downloading [=====>
] 9.732MB/43.25MB
504facff238f: Download complete
ebbcacd28e10: Download complete
c7fb3351ecad: Download complete
2e3debadcbf7: Download complete
```

编译时间与网络环境与PC的配置有关，可能几分钟到几十分钟不等。编译完成后，执行 `docker image list` 可以看到输出中包含了tiantian\_android\_kernel镜像。

### 编译3.4内核

编译内核需要进行到tiantian\_android\_kernel镜像环境中，启动镜像时，需要指定内核代码与NDK的完整路径，如下所示：

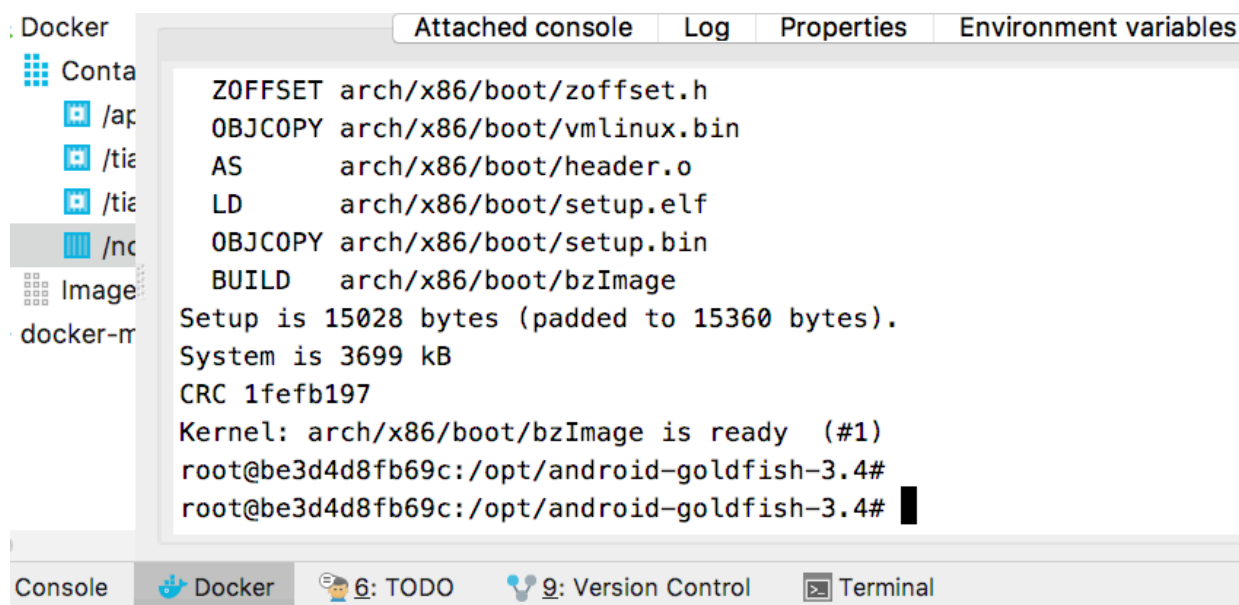
```
$ docker run -v ~/android-goldfish-3.4:/opt/android-goldfish-3.4 -v
~/android-ndk-r10e:/opt/android-ndk -it tiantian_android_kernel bash
```

执行成功后，就会进行到 Docker 环境，并且内核代码与NDK都映射到了镜像的"/opt"目录下。然后，执行如下命令编译内核：

```
root@be3d4d8fb69c:/# cd /opt/android-goldfish-3.4/
root@be3d4d8fb69c:/opt/android-goldfish-3.4# export CROSS_COMPILE=i686-
linux-android-
root@be3d4d8fb69c:/opt/android-goldfish-3.4# export ARCH=x86
root@be3d4d8fb69c:/opt/android-goldfish-3.4# export
PATH=$PATH:/opt/android-ndk/toolchains/x86-4.9/prebuilt/linux-x86_64/bin
root@be3d4d8fb69c:/opt/android-goldfish-3.4# make goldfish_defconfig
root@be3d4d8fb69c:/opt/android-goldfish-3.4# make -j4
```

在编译时，如果遇到错误，通常是低版本内核中的配置有问题。可以在.config文件中注释掉相应的选择，重新编译即可。

编译完成后，如图所示：



```
Docker
Attached console Log Properties Environment variables

ZOFFSET arch/x86/boot/zoffset.h
OBJCOPY arch/x86/boot/vmlinux.bin
AS arch/x86/boot/header.o
LD arch/x86/boot/setup.elf
OBJCOPY arch/x86/boot/setup.bin
BUILD arch/x86/boot/bzImage
Setup is 15028 bytes (padded to 15360 bytes).
System is 3699 kB
CRC 1fefb197
Kernel: arch/x86/boot/bzImage is ready (#1)
root@be3d4d8fb69c: /opt/android-goldfish-3.4#
root@be3d4d8fb69c: /opt/android-goldfish-3.4#
```

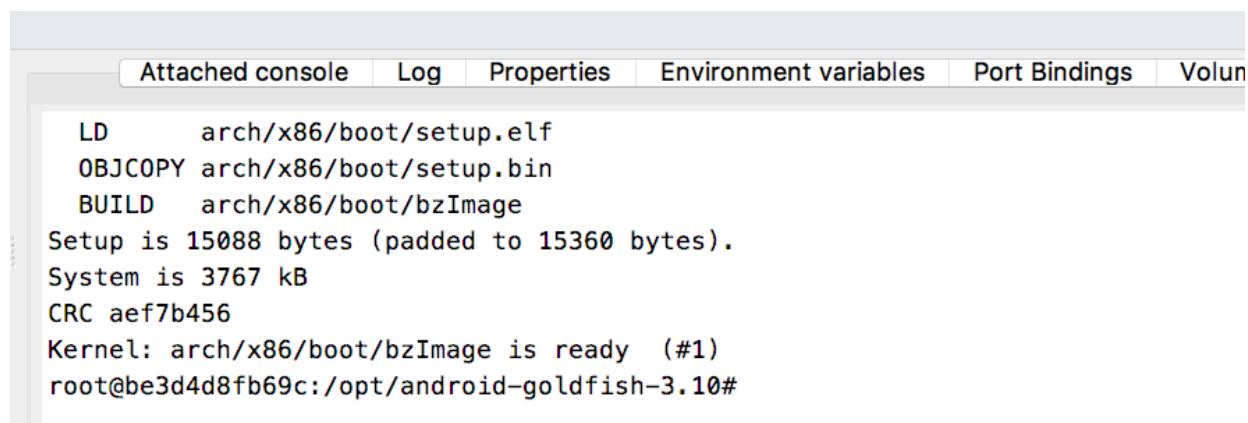
## 编译3.10内核

编译3.10内核与编译3.4的区别不大，主要体现在一些小细节上。首先，同样执行如下命令，映射目录并启动 Docker：

```
$ docker run -v ~/android-goldfish-3.10:/opt/android-goldfish-3.10 -v  
~/android-ndk-r10e:/opt/android-ndk -it tiantian_android_kernel bash
```

然后在编译时，指定config文件名为i386\_qemu\_defconfig，这与3.4的goldfish\_defconfig文件名不同，主要原因是arch/x86/configs目录下的文件名不同造成的。最后，所有的操作如下所示：

```
root@be3d4d8fb69c: /# cd /opt/android-goldfish-3.10/  
root@be3d4d8fb69c: /opt/android-goldfish-3.10# export CROSS_COMPILE=i686-  
linux-android-  
root@be3d4d8fb69c: /opt/android-goldfish-3.10# export ARCH=x86  
root@be3d4d8fb69c: /opt/android-goldfish-3.10# export  
PATH=$PATH:/opt/android-ndk/toolchains/x86-4.9/prebuilt/linux-x86_64/bin  
root@be3d4d8fb69c: /opt/android-goldfish-3.10# make i386_qemu_defconfig  
root@be3d4d8fb69c: /opt/android-goldfish-3.10# make -j4
```



```
Attached console Log Properties Environment variables Port Bindings Volun

LD arch/x86/boot/setup.elf
OBJCOPY arch/x86/boot/setup.bin
BUILD arch/x86/boot/bzImage
Setup is 15088 bytes (padded to 15360 bytes).
System is 3767 kB
CRC aef7b456
Kernel: arch/x86/boot/bzImage is ready (#1)
root@be3d4d8fb69c: /opt/android-goldfish-3.10#
```



# 测试内核

内核编译完成后，可以使用Android SDK中的 `emulator` 模拟器测试，执行如下命令：

```
$ emulator -kernel ./kernel/goldfish-android-3.4/arch/x86/boot/zImage &
```

## 小结

本篇主要介绍了，如何使用 `Docker` 编译Android系统的内核代码，如何配置 `Docker` 环境，如何下载与编译内核代码等，编译完内核后，需要进一步的修改定制内核、开发内核模块，这些内容以后有机会再与大家讨论。

文章精美排版PDF与代码，知识星球会员可以在知识星球：**【软件安全与逆向分析】**（ID：**86753808**）中下载。



更多精彩内容，欢迎关注微信公众号 **【feicong\_sec】**



