IoT: Client Devices

Installing Linux, Buildroot

Software Installation & Use

NOT EVERY STEP

- I won’t cover every single step in installations

NOT EASY STUFF

- I won’t cover easy, straight forward stuff either
- This stuff is on the internet, or in documentation

THINGS THAT ARE COMPLEX, NEED EXPLANATION

- I WILL cover complex stuff where you’re likely to get hung up

This is a graduate level course.

Download Linux

Ubuntu 16.04

- This is the LTS release
- Relatively stable

Based on Debian

- Easy installation
- Download ISO from https://ubuntu.com

…or your favorite distro

- If you have one
- Examples will be in Ubuntu

Type of Installation

Server or Workstation?

- Either one is fine, but server generally has no GUI

GUI or no?

- I generally do not use a GUI (but you can)
- Makes for a smaller image

Development

- No GUI may be more difficult if not comfortable in Linux

Create Virtual Image

Create a new image

- In either Virtualbox or VMWare
- Follow directions, internet guidance

Follow the prompts

- Make sure you install SSH during installation
- You'll want to be able to SSH into the system

Create Strong Passwords

- This is your development workstation after all

IoT: Client Devices

Vagrant & Buildroot

What is Buildroot?

Embedded linux build environment

- Makes building embedded linux distributions easier

Uses make semantics

- Generate makefiles to build embedded linux images

Variety of interfaces

- I use the basic interface, but you can use others

What is Vagrant

Portable Development Environments

- Uniform development environments
- Great for distributed teams, QA, build

Vagrant & Buildroot

- Buildroot supports Vagrant
- This is how we'll install it

Dependencies

- Vagrant uses Virtualbox natively, must pay for VMWare support

Installing Vagrant

VAGRANT BASE INSTALLATION

- Download: https://www.vagrantup.com/
- Virtualbox? you should be covered.
- VMWare? You'll need to buy the VMWare provider
  - I use VMWare, but tweaked Buildroot to do so
  - Easier if you use Virtualbox in this case

VAGRANT DOES NOT USE OWN VIRTUALIZATION

- Uses providers to attach to virtualization software
- Virtualbox provider installed by default, not VMWare

Installing Buildroot

Buildroot is complex and powerful

- RTM: https://buildroot.org/downloads/manual/manual.html

Getting Started

- Starting here
- RTM: https://buildroot.org/downloads/manual/manual.html#getting-buildroot

Downloads VagrantFile, starts Vagrant

- VagrantFile is an internal Ruby DSL
- Vagrant reads, downloads Linux VM, installs Buildroot

BuildRoot Running

BUILDROOT VM RUNNING

- The Buildroot VM should be running
- Useful Vagrant Commands:
  - vagrant port, vagrant ssh, vagrant halt, vagrant suspend
  - type ‘vagrant’ at command line to see more

PORTS USED

- vagrant port shows active ports
  - note your Vagrant VM is active on port 2222
  - You need this to SCP images from your Buildroot VM

IoT: Client Devices

Client Emulation and Virtual Machines

IoT Clients

LINUX

- Embedded linux, old Linux versions, standard services

BUSYBOX, BUILDROOT

- https://www.busybox,net ; https://buildroot.org

APPLICATION SOFTWARE

- Standard C application running in linux system

Emulating Workflow

Virtualbox / VMWare / Qemu

- Develop on Linux Workstation
- Deploy to stripped Linux image
- Both on host

Build root

- Deployment target's x86 (We'll pretend it's MIPS or ARM)

Linux Workstation

- Gcc tools
- Cross-compilation

VMWare or Virtualbox?

Virtualbox

- Open source, free to use
- https://www.virtualbox.org

VMWare

- Not free, not exactly cheap either
- commercial support, more robust

Which one?

- VMWare is more stable and robust
- VIrtualbox is more picky

QEMU?

Processor Emulator

- Emulates different processor types

Virtualization Too

- We’re not using it for that though

Runs IoT Client

- Copy and run programs on client

IoT: Client Devices

Testing our Development Environment

Start up your VM

Log into your VM

- Start up virtualization
- Log in
- Open a few windows

Other Tools

- I use things like Tmux and Powerline (you don’t have to)

Build an ARM Image

Buildroot

- Versatile Platform Baseboard
- Configure with defaults and Build
- cd to Buildroot directory
- $ make qemu_arm_versatile_defconfig

What does this do?

- Creates a base .config file with defaults for board
- Build stock version first

Now Add SSH

WHY?

- We need to move a cross-compiled executable to the image

NAVIGATE TO BUILDROOT

- Open the configuration menu (make nconfig)
- Target Packages -> Networking applications -> openssh
- Rebuild (just type make)

Build Results

So what did we build?

- Take a look in $buildroot_home/output/images
  - you should see: zImage, rootfs.ext2, versatile-pb.dtb
  - what are these things?
    - zImage: Kernel image
    - rootfs.ext2: Root filesystem
    - versatile-pb.dtb: Device tree blob (contains hardware info)

Let's Run

Build Instructions

- $buildroot_home/board/qemu/arm-versatile/readme.txt
- The command line for the new QEMU image is in the readme (see next page for script)

Running in QEMU

Might be better as a script

qemu-versatile.sh:

```
qemu-system-arm \
  -M versatilepb \
  -kernel output/images/zImage \
  -dtb output/images/versatile-pb.dtb \
  -drive file=output/images/rootfs.ext2,if=scsi,format=raw -append "root=/dev/sda console=ttyAMA0,115200" \
  -serial stdio \
  -net nic,model=rtl8139 -net user \
  -redir tcp:2222:22
```

IoT: Client Devices

Testing the QEMU Image

QEMU VM is Up

Our VM is running

- Excellent!
- Slightly different - running in terminal
- Go ahead and log in
  - root user has no password (we can set this in Buildroot)

We need to add a user

- adduser to the rescue
- $ adduser -h /<username> -s /bin/sh <username>

Test SSH

Now test SSH

- from your Linux host, attempt to SSH to QEMU image
  - $ ssh -p 2222 <username>@localhost
  - Remember the line -redir tcp:2222:22

Failed!

- It keeps asking to change my password, then won't let me in!

Configuration Details

…/ETC/SHADOW PROBLEMS

- the /etc/shadow file is not updated, and we're not running password management services
- we'll need to update manually
- as root edit /etc/shadow so the last line looks like this:

<username>:$1$7UvOizz/$SOcRUgT9PVcpyaQ9O3E9I0:10933:0:99999:7:::

This was 0, now 10933

Test Cross-Compilation

NOW TEST CROSS-COMPILATION

- Simple test program in C:

```
#include <stdio.h>
int main(void) {
  printf("running.\n");
  return 0;
}
```

- $ arm-linux-gnueabi-gcc -static -o test main.c
- $ scp -P 2222 ./test cclamb@localhost:~/
- …run test program on ARM image

Solid!

CONGRATS!

- Embedded linux system on emulated ARM processor
- Cross-compiled a C program
- Moved program to ARM host
- Ran program

WE CAN FINALLY GET STARTED WITH CODE

IoT: Client Devices

Binwalk

What is Binwalk?

Firmware Analysis

- Allows you to list the contents of a firmware file
- Perform entropy analysis
- Identify platform dependencies

Firmware Extraction

- Filesystems
- Operating systems
- Application software

Installation

Runs on Linux

- You'll need to use your linux VM

Complex installation process

- Many dependencies

Written in python

- Dependencies are in both Python and at OS level

Installation

INSTALLATION INSTRUCTIONS *MOSTLY* ACCURATE

- Packages correct
- Some things better installed with APT, not PIP

INSTALLATION TOOLS

- PIP: An installation framework for Python
- APT: An installation framework for Debian

YOU'LL USE BOTH

- Python libs with PIP, OS packages with APT

Installation

Python 2 or Python 3?

- I use Python 2, but you can use Python 3
- Don't install both

How do I install?

- See https://github.com/devttys0/binwalk/blob/master/INSTALL.md
- I suggest you install libcapstone3 as well via APT
  - sudo apt install libcapstone3

Installation

How do I know it worked?

```
cclamb@angr:~$ binwalk

Binwalk v2.1.1
Craig Heffner, http://www.binwalk.org

Usage: binwalk [OPTIONS] [FILE1] [FILE2] [FILE3] ...

Disassembly Scan Options:
    -Y, --disasm                 Identify the CPU architecture of a file using t
he capstone disassembler
    -T, --minsn=<int>            Minimum number of consecutive instructions to b
e considered valid (default: 500)
```

It didn't work!

LIBCAPSTONE3

- Try installing libcapstone3 from APT if you didn't originally.
- $ sudo apt install libcapstone3

PYTHON ERRORS

- Use only Python 2 or Python 3
- Deinstall all your python, they try again with one or the other, not both

IoT: Client Devices

Other Tools

Radare (optional)

Disassembler

- Allows disassembly of multiple instruction sets

Reverse Engineering Toolset

- Very complex though

GUI and CLI

- Bokken, WebUI
- http://radare.org for more info

Git

You need a version control system

- Rollback to working versions
- Tracking changes
- Configuration files, source code, other

Github

- http://www.github.com
- We'll use it for assignment evaluation too
- Register for an account, it's free

Gnu Binutils

Tools for Binary Analysis, Debugging

- readelf, objdump, strip, strings, nm and more
- Visibility into binary files

READELF

- -a (use less, this does everything) others

OBJDUMP

- -h, -f, -x, -d, -D, -t

Gnu Cross-Compilers

Compile on x86 Linux for other Architectures

- Sorcery Workbench (https://www.mentor.com/embedded-software/sourcery-tools/sourcery-codebench/editions/lite-edition/)
- Ubuntu distros (ARM has best support out of the box)

How to install?

GNU BINUTILS

- $ sudo apt install binutils-mips-linux-gnu binutils-mipsel-linux-gnu binutils-arm-linux-gnueabi

GNU CROSS-COMPILERS

- $ sudo apt install gcc-arm-linux-gnueabi

Summary

So what do we have?

- Linux VM
- Version Control System
- Buildroot
- Binwalk
- Binutils for various architectures
- Cross compiler
- QEMU (and qemu-user-static)

IoT: Client Devices

Reversing: Download and Extract

Firmware Image

TP-LINK HS110 V1

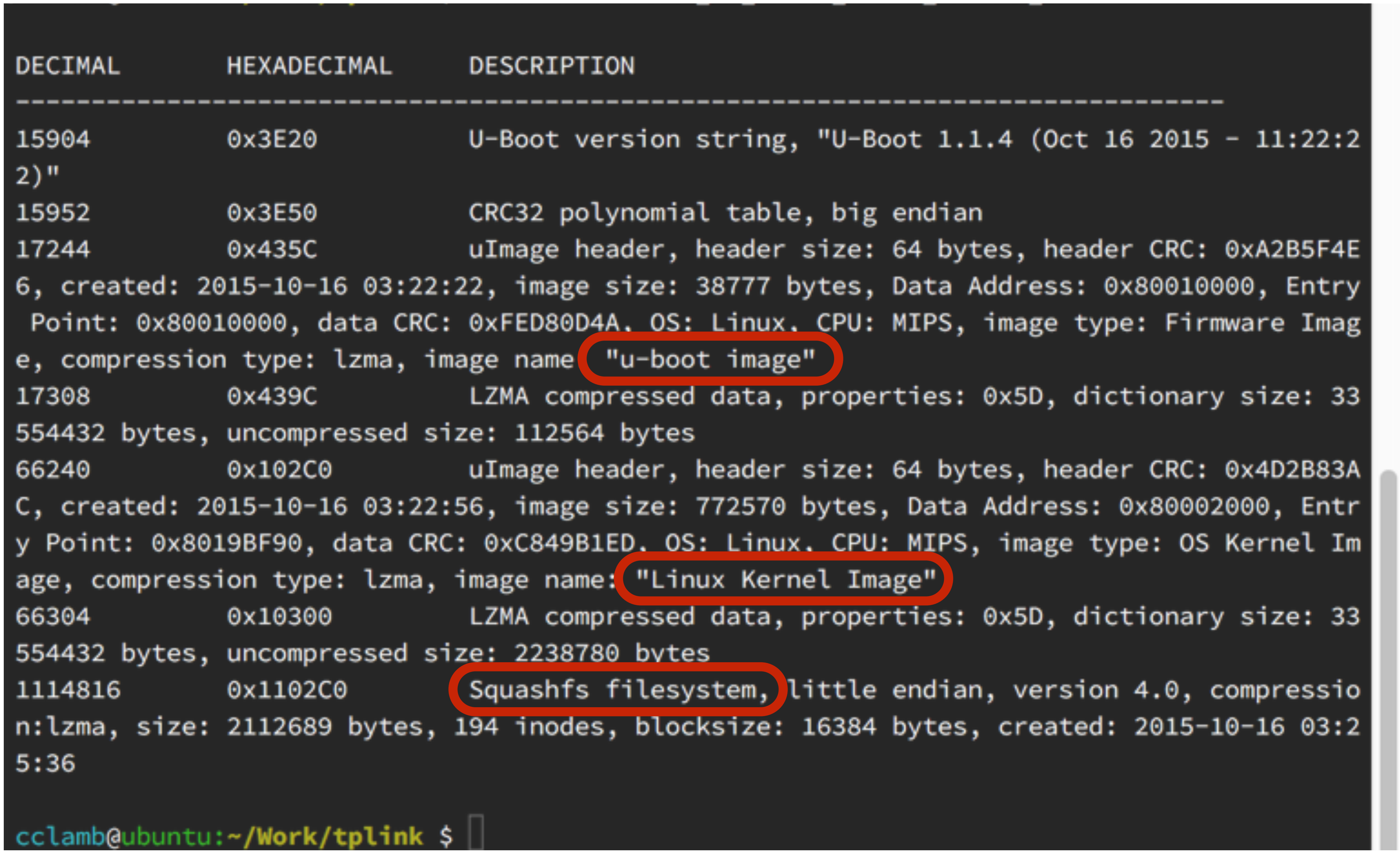
- http://www.tp-link.com/us/download/HS110.html#Firmware
- ZIP archive, go ahead and unzip
- You'll have some files and a BIN file
- BIN file is the firmware image!

What Does it Do?

Smart Plug

- Programmed via app
- Provides Usage, time used information
- uses Kasa app (see app store, google play)

Still uses full Linux!



```
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
15904         0x3E20          U-Boot version string, "U-Boot 1.1.4 (Oct 16 2015 - 11:22:2
2)"
15952         0x3E50          CRC32 polynomial table, big endian
17244         0x435C          uImage header, header size: 64 bytes, header CRC: 0xA2B5F4E
6, created: 2015-10-16 03:22:22, image size: 38777 bytes, Data Address: 0x80010000, Entry
 Point: 0x80010000, data CRC: 0xFED80D4A, OS: Linux, CPU: MIPS, image type: Firmware Imag
e, compression type: lzma, image name: "u-boot image"
17308         0x439C          LZMA compressed data, properties: 0x5D, dictionary size: 33
554432 bytes, uncompressed size: 112564 bytes
66240         0x102C0         uImage header, header size: 64 bytes, header CRC: 0x4D2B83A
C, created: 2015-10-16 03:22:56, image size: 772570 bytes, Data Address: 0x80002000, Entr
y Point: 0x8019BF90, data CRC: 0xC849B1ED, OS: Linux, CPU: MIPS, image type: OS Kernel Im
age, compression type: lzma, image name: "Linux Kernel Image"
66304         0x10300         LZMA compressed data, properties: 0x5D, dictionary size: 33
554432 bytes, uncompressed size: 2238780 bytes
1114816       0x1102C0        Squashfs filesystem, little endian, version 4.0, compressio
n:lzma, size: 2112689 bytes, 194 inodes, blocksize: 16384 bytes, created: 2015-10-16 03:2
5:36

cclamb@ubuntu:~/Work/tplink $
```

Now What?

Let’s take a look inside

Onto Analysis!

IoT: Client Devices

Reverse Engineering

Types of Firmware

NO HOST

- No OS, services and operating code mixed
- Hard drives, USB, simple micro controllers
- BIOS, EFI/UEFI, etc.
- Smaller, simpler, less powerful

HOSTED

- Embedded Linux
- Have some kind of OS
- Userspace services on OS
- Larger, more complex, more powerful

Reverse Engineering

Why Reverse Engineering?

- See what others do
- Understand why
- Understand mistakes and avoid them!

Start with downloadable images

- These don't always exist

Reversing a Device

Scan the device

- Scan ports, monitor traffic examine protocols, dynamic analysis

Running software

- We can run code using QEMU
- Real device better though

All good things!

- We're not reverse engineers
- We just want to see the code

What to Reverse?

Publicly available images

- Downloadable, we'll use TP-Link firmware images
- Saves you from extracting or buying the device

NOTE: Should your firmwares be available?

- Controversial
- Hiding images is security by obscurity

IoT: Client Devices

Reversing: HS110 Filesystem

Take a look around!

Here, we've listed the /bin directory (I used tree; to install, type *sudo apt install tree* at the command line)

```
cclamb — ssh -X 192.168.120.134 — 58×31
cclamb@ubuntu:~/Work/tplink/hs110/squashfs-root $ ls
bin  etc  linuxrc  proc  sbin  tmp
dev  lib  mnt      root  sys   usr
cclamb@ubuntu:~/Work/tplink/hs110/squashfs-root $ tree bin
bin
├── ash -> busybox
├── busybox
├── cat -> busybox
├── chmod -> busybox
├── cp -> busybox
├── date -> busybox
├── echo -> busybox
├── ip -> busybox
├── kill -> busybox
├── length-decode
├── login -> busybox
├── ls -> busybox
├── minidump
├── mkdir -> busybox
├── mount -> busybox
├── ping -> busybox
├── ps -> busybox
├── pwd -> busybox
├── rm -> busybox
├── rmdir -> busybox
├── sh -> busybox
├── umount -> busybox
└── vi -> busybox

0 directories, 23 files
cclamb@ubuntu:~/Work/tplink/hs110/squashfs-root $
```

```
cclamb — ssh -X 192.168.120.135 — 89×26
cclamb@ubuntu:~/Work/tplink/hs110/squashfs-root/etc $ tree .
.
├── 2048_newroot.cer
├── ath
│   ├── activateVAP
│   ├── apcfg
│   └── apup
├── fstab
├── group
├── inittab
├── passwd
├── public.key
├── rc.d
│   ├── rcS
│   └── rc.wlan
├── resolv.conf -> /tmp/resolv.conf
├── shadow
├── sw.version
└── wpa2

3 directories, 14 files
cclamb@ubuntu:~/Work/tplink/hs110/squashfs-root/etc $
```

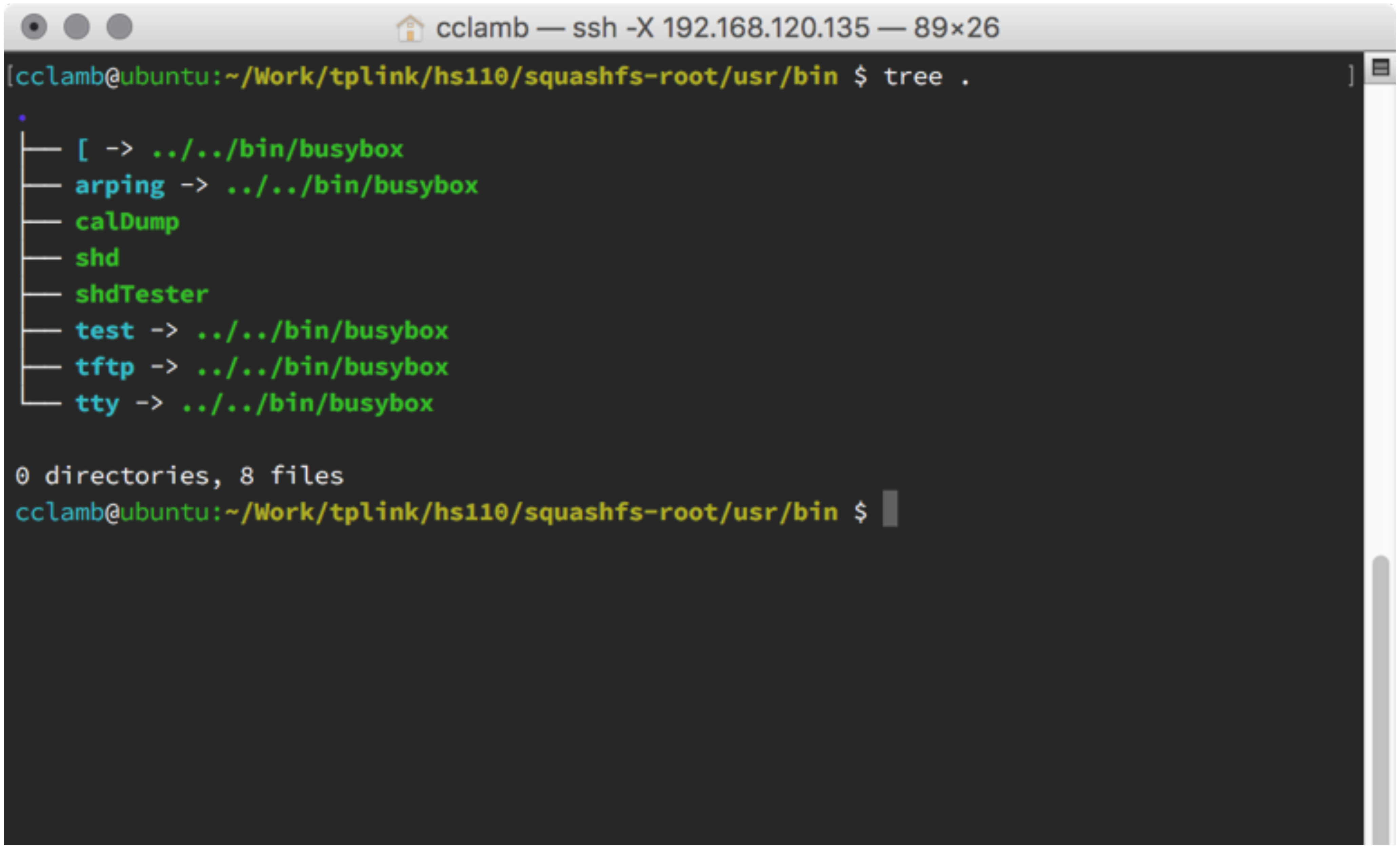
/etc is always fun!

We have password files, certificates, and startup configs

```
cclamb — ssh -X 192.168.120.135 — 89×26
cclamb@ubuntu:~/Work/tplink/hs110/squashfs-root/etc $ tail rc.d/rcS
# lets force them to write
echo 20 > /proc/sys/vm/dirty_ratio

# when the dirty pages cross more than 5% of sys memory,
# kick in the pdflush
echo 5 > /proc/sys/vm/dirty_background_ratio

#/usr/sbin/telnetd &

/usr/bin/shd &
cclamb@ubuntu:~/Work/tplink/hs110/squashfs-root/etc $
```

Startup Files

Various services and a non-standard daemon



/usr/bin

Application level software!

IoT: Client Devices

So what have we learned?

So Far…

EXTRACTED FIRMWARE IMAGE

- Full OS, embedded linux image, for TP-Link smart plug

EXAMINED THE OS AND BOOTLOADER

- An older version of Linux
- And equally old version of U-Boot

LOOKED THROUGH THE FILESYSTEM

- Checked out config files
- Found application daemon

All Your Product!

YOU ARE RESPONSIBLE!

IoT: Client Devices

Running Application Executables

Application Analysis

Regular applications

- GDB, LLDB
- Binutils
- Tracing Tools (DTrace, STrace, etc.)

Cross-compilation problems

- Cross-compiled binutils

But What about dynamic analysis?

QEMU to the Rescue!

QEMU can run applications locally

- chroot
- qemu-*-static (e.g. qemu-mips-static)

How to use

- Easiest if you use the extracted filesystem
- copy the appropriate static execution utility
- run cross-compiled

```
cclamb@ubuntu:~/Work/tplink/hs110/squashfs-root $ pwd
/home/cclamb/Work/tplink/hs110/squashfs-root
cclamb@ubuntu:~/Work/tplink/hs110/squashfs-root $ ls
bin  dev  etc  lib  linuxrc  mnt  proc  root  sbin  sys  tmp  usr
cclamb@ubuntu:~/Work/tplink/hs110/squashfs-root $ which qemu-mips-static
/usr/bin/qemu-mips-static
cclamb@ubuntu:~/Work/tplink/hs110/squashfs-root $ cp /usr/bin/qemu-mips-static .
cclamb@ubuntu:~/Work/tplink/hs110/squashfs-root $ ls
bin  dev  etc  lib  linuxrc  mnt  proc  qemu-mips-static  root  sbin  sys  tmp  usr
cclamb@ubuntu:~/Work/tplink/hs110/squashfs-root $ sudo chroot . ./qemu-mips-static /usr/b
in/shd
[sudo] password for cclamb:
qemu: uncaught target signal 11 (Segmentation fault) - core dumped
Segmentation fault (core dumped)
cclamb@ubuntu:~/Work/tplink/hs110/squashfs-root $ sudo chroot . ./qemu-mips-static /usr/b
in/shd -h
sudo: unable to resolve host HS110(US)
options for /usr/bin/shd
        -k      kill all shd process
        -t      enter test mode
        -h      print this help
cclamb@ubuntu:~/Work/tplink/hs110/squashfs-root $
```

Running /usr/bin/shd

with flags and without!

Why chroot?

You may not always need it

- statically linked applications

Using libraries? You need chroot

- shd uses a few
- how do we know?

```
Dynamic section at offset 0x180 contains 30 entries:
  Tag        Type                         Name/Value
 0x00000001 (NEEDED)                     Shared library: [librt.so.0]
 0x00000001 (NEEDED)                     Shared library: [libm.so.0]
 0x00000001 (NEEDED)                     Shared library: [libpthread.so.0]
 0x00000001 (NEEDED)                     Shared library: [libresolv.so.0]
 0x00000001 (NEEDED)                     Shared library: [libgcc_s.so.1]
 0x00000001 (NEEDED)                     Shared library: [libc.so.0]
 0x0000000c (INIT)                       0x4160b8
 0x0000000d (FINI)                       0x598070
 0x00000004 (HASH)                       0x400298
 0x00000005 (STRTAB)                     0x40c3f4
 0x00000006 (SYMTAB)                     0x4042f4
 0x0000000a (STRSZ)                      35906 (bytes)
 0x0000000b (SYMENT)                     16 (bytes)
 0x70000016 (MIPS_RLD_MAP)               0x5f0860
 0x00000015 (DEBUG)                      0x0
 0x00000003 (PLTGOT)                     0x5f0870
 0x00000011 (REL)                        0x416078
 0x00000012 (RELSZ)                      64 (bytes)
 0x00000013 (RELENT)                     8 (bytes)
 0x70000001 (MIPS_RLD_VERSION)           1
```

Shared Libraries

```
cclamb@ubuntu:~/Work/tplink/hs110/squashfs-root $ ls -al lib
total 928
drwxrwxr-x  3 cclamb cclamb   4096 Oct 15  2015 .
drwxrwxr-x 12 cclamb cclamb   4096 Jan  6 11:03 ..
-rwxr-xr-x  1 cclamb cclamb  20704 Oct 15  2015 ld-uClibc-0.9.30.so
lrwxrwxrwx  1 cclamb cclamb     19 Jan  2 16:11 ld-uClibc.so.0 -> ld-uClibc-0.9.30.so
-rw-r--r--  1 cclamb cclamb  10148 Oct 15  2015 libcrypt-0.9.30.so
lrwxrwxrwx  1 cclamb cclamb     18 Jan  2 16:11 libcrypt.so.0 -> libcrypt-0.9.30.so
lrwxrwxrwx  1 cclamb cclamb     19 Jan  2 16:11 libc.so.0 -> libuClibc-0.9.30.so
-rw-r--r--  1 cclamb cclamb   8312 Oct 15  2015 libdl-0.9.30.so
lrwxrwxrwx  1 cclamb cclamb     15 Jan  2 16:11 libdl.so.0 -> libdl-0.9.30.so
lrwxrwxrwx  1 cclamb cclamb     13 Jan  2 16:11 libgcc_s.so -> libgcc_s.so.1
-rw-r--r--  1 cclamb cclamb 174432 Oct 15  2015 libgcc_s.so.1
-rw-r--r--  1 cclamb cclamb 100968 Oct 15  2015 libm-0.9.30.so
lrwxrwxrwx  1 cclamb cclamb     14 Jan  2 16:11 libm.so.0 -> libm-0.9.30.so
-rw-r--r--  1 cclamb cclamb    917 Oct 15  2015 libnsl-0.9.30.so
lrwxrwxrwx  1 cclamb cclamb     16 Jan  2 16:11 libnsl.so.0 -> libnsl-0.9.30.so
-rw-r--r--  1 cclamb cclamb  71528 Oct 15  2015 libpthread-0.9.30.so
lrwxrwxrwx  1 cclamb cclamb     20 Jan  2 16:11 libpthread.so.0 -> libpthread-0.9.30.so
-rw-r--r--  1 cclamb cclamb    917 Oct 15  2015 libresolv-0.9.30.so
lrwxrwxrwx  1 cclamb cclamb     19 Jan  2 16:11 libresolv.so.0 -> libresolv-0.9.30.so
-rw-r--r--  1 cclamb cclamb   3412 Oct 15  2015 librt-0.9.30.so
lrwxrwxrwx  1 cclamb cclamb     15 Jan  2 16:11 librt.so.0 -> librt-0.9.30.so
-rw-r--r--  1 cclamb cclamb 364240 Oct 15  2015 libuClibc-0.9.30.so
-rw-r--r--  1 cclamb cclamb   3964 Oct 15  2015 libutil-0.9.30.so
```

Packaged Libraries

/usr/bin/shd needs to find these to run!

IoT: Client Devices

Reversing: Image Analysis

So Far…

U-Boot boot loader

- Version 1.1.4, built in late 2015

Linux Kernel

- Also built in late 2015

Filesystem

- Squash-fs filesystem, late 2015

Extracting

NOW LET'S USE BINWALK TO EXTRACT THE FILES

- binwalk -e -C extracted -M <imagename>

WHAT DOES THIS DO?

- -e: extract the contents of the image
- -C: place the results in the 'extracted' subdirectory
- -M: recursively scan extracted stuff

cclamb — ssh -X 192.168.120.134 — 89×26

```
cclamb@ubuntu:~/Work/tplink/extracted/_hs110v1_us_1.0.7_Build_151016_Rel.24186.bin.extrac
ted $ ls
10300  10300.7z  _10300.extracted  1102C0.squashfs  439C  439C.7z  squashfs-root
cclamb@ubuntu:~/Work/tplink/extracted/_hs110v1_us_1.0.7_Build_151016_Rel.24186.bin.extrac
ted $ binwalk ../../hs110v1_us_1.0.7_Build_151016_Rel.24186.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
15904         0x3E20          U-Boot version string, "U-Boot 1.1.4 (Oct 16 2015 - 11:22:2
2)"
15952         0x3E50          CRC32 polynomial table, big endian
17244         0x435C          uImage header, header size: 64 bytes, header CRC: 0xA2B5F4E
6, created: 2015-10-16 03:22:22, image size: 38777 bytes, Data Address: 0x80010000, Entry
 Point: 0x80010000, data CRC: 0xFED80D4A, OS: Linux, CPU: MIPS, image type: Firmware Imag
e, compression type: lzma, image name: "u-boot image"
17308         0x439C          LZMA compressed data, properties: 0x5D, dictionary size: 33
554432 bytes, uncompressed size: 112564 bytes
66240         0x102C0         uImage header, header size: 64 bytes, header CRC: 0x4D2B83A
C, created: 2015-10-16 03:22:56, image size: 772570 bytes, Data Address: 0x80002000, Entr
y Point: 0x8019BF90, data CRC: 0xC849B1ED, OS: Linux, CPU: MIPS, image type: OS Kernel Im
age, compression type: lzma, image name: "Linux Kernel Image"
66304         0x10300         LZMA compressed data, properties: 0x5D, dictionary size: 33
554432 bytes, uncompressed size: 2238780 bytes
1114816       0x1102C0        Squashfs filesystem, little endian, version 4.0, compressio
n:lzma, size: 2112689 bytes, 194 inodes, blocksize: 16384 bytes, created: 2015-10-16 03:2
5:36
```

```
cclamb — ssh -X 192.168.120.134 — 89×26
cclamb@ubuntu:~/Work/tplink/hs110 $ strings -n 10 10300 > strings.out
cclamb@ubuntu:~/Work/tplink/hs110 $ head strings.out
initcall_debug
Linux version 2.6.31--LSDK-9.2.0_U11.14 (yt@yangtao.localdomain) (gcc version 4.3.3 (GCC)
 ) #10 Tue Sep 8 15:36:13 HKT 2015
%s version %s (yt@yangtao.localdomain) (gcc version 4.3.3 (GCC) ) %s
plat_time_init
ar7240_serial_setup
ar7240_spi_flash_read_page
ar7240wdt_init
pause_on_oops
Od<2>BUG: recent printk recursion!
printk.time
cclamb@ubuntu:~/Work/tplink/hs110 $ strings -n 10 439C > 439C-strings.out
cclamb@ubuntu:~/Work/tplink/hs110 $ head 439C-strings.out
U-Boot 1.1.4 (Oct 16 2015 - 11:22:19)
ag7240_miiphy_write
ag7240_miiphy_read
ag7240_get_ethaddr
ag7240_mii_setup
reset   - Perform RESET of the CPU
   Image Name:   %.*s
   Created:      %4d-%02d-%02d  %2d:%02d:%02d UTC
   Image Type:
Invalid OS
cclamb@ubuntu:~/Work/tplink/hs110 $
```

Strings

Seems to confirm binwalk results, but now we have a kernel version (released in 2009!)

```
[cclamb@ubuntu:~/Work/tplink/hs110 $ ls
10300     _10300.extracted   439C      439C-strings.out   squashfs-root
10300.7z  1102C0.squashfs    439C.7z   hex.out            strings.out
[cclamb@ubuntu:~/Work/tplink/hs110 $ cd squashfs-root/
[cclamb@ubuntu:~/Work/tplink/hs110/squashfs-root $ ls
bin  dev  etc  lib  linuxrc  mnt  proc  root  sbin  sys  tmp  usr
cclamb@ubuntu:~/Work/tplink/hs110/squashfs-root $
```

The filesystem

Take a look at squashes-root; it has a complete filesystem!