Hacking the CBC Botball Controller:
Because It Wouldn't Be a Botball Controller if It Couldn't Be Hacked
Part 1

Jeremy Rand, Matt Thompson, and Braden McDorman
Norman High School, Nease High School, Norman High School
jeremy@asofok.org, matthewbot@gmail.com, bmcdorman@gmail.com

Hacking the CBC Botball Controller: Because It Wouldn't Be a Botball Controller if It Couldn't Be Hacked Part 1

1. Introduction

The CBC Botball Controller presented various challenges to hackers when it was released to Botball students in January 2009. Unlike the XBC firmware, which had freely available source code (if you knew where to look), the CBC's firmware was only available in binary form, and KIPR didn't share as much CBC information as Charmed Labs did with the XBC. Despite the issue of not having source code to look through or an expert to talk to, we made good (and hopefully useful) progress in reverse-engineering the CBC, which we present to you now. Because it wouldn't be a Botball controller if it couldn't be hacked.

DISCLAIMER: Hacking the CBC's file system in any way can brick your CBC if done improperly. While we have tried to be as accurate as possible, we are not responsible for any damage that may result from the use of this information. As with any Linux system that gives you root access, you should not mess with files whose purpose you do not understand. KIPR will most likely charge you money to repair a CBC which you brick through improper use of these methods, and don't expect us to fix your CBC for you either. If this scares you, that should be a hint that CBC hacking is probably not for you.

2. Cygwin

Much of this paper relies on having a Linux-like system on your PC. If you're on Windows, we recommend Cygwin. The official Cygwin installer is not particularly user-friendly, but Charmed Labs offers a simplified installer [1], which should work for all of the topics in this paper. The Charmed Labs installer takes under 5 minutes to set up. Once installed, you can run Cygwin by executing `C:\cygwin\cygwin.bat` .

You can use the `cd` and `ls` commands to navigate folders. The C drive is mounted at `/cygdrive/C/` . So for example, you would type `cd /cygdrive/C/botball` to get to the folder `C:\botball`. Please note that if you're the kind of person who can't stand typing simple console commands, and wants a graphical interface for everything, you're going to have to get used to the command line, or CBC hacking may not be for you.

3. How the CBC Works

KIPR will happily tell you that the C in CBC is recursive; that is, C stands for CBC, i.e. nothing in particular. However, this is mainly for legal reasons. A more accurate expansion (albeit one which would infringe on trademarks) is Chumby Botball Controller. What is a Chumby? This thing:

The CBC is a very heavily modified Chumby, with a Breakout Board called CBOB added. The Chumby (using an ARM9 processor) handles all of the user interaction and the camera processing, while the CBOB (using an ARM7 processor) handles motors, servos, sensors, and the FTDI and TTL serial ports. The Chumby and CBOB are connected by an SPI interface, and swap data approximately every 25ms.

The CBOB is completely closed to outside interference at the moment, so we decided not to pursue hacking it. (Doing so would likely brick it, and would also void the warranty.) The Chumby, in contrast, is a reasonably open platform, although KIPR doesn't make it obvious.

The Chumby runs Linux, and this itself is an excellent piece of knowledge to exploit, because if you can do something on a Linux PC, you have a good chance of being able to do it on the Chumby. The Chumby's software runs as the root user, meaning that you can do anything without permission issues. A Linux application called cbcui is responsible for most of what the Chumby does related to Botball. cbcui is responsible for camera processing, CBOB interaction, and everything you see on the screen.

Hardware-wise, the Chumby has been modified for Botball in at least a few important ways besides the CBOB. The internal Wi-Fi USB card in the Chumby was replaced with a 1GiB USB flash drive. Presumably, this is to discourage attempts to cheat with a Wi-Fi remote control. (And the Botball firmware takes up a good bit of flash storage.) The Chumby's accelerometers

were moved to the CBOB, so attempts to use the Chumby accelerometer driver (which has more features than the Botball driver) will fail.

The Chumby's internal flash (64MiB NAND) is mostly unchanged between official Chumbys and CBC Chumbys. All that seems to be modified is a boot script, which is used to launch the Botball-specific firmware stored on the 1GiB USB flash drive. The 1GiB drive contains three partitions, mounted at `/mnt/usb`, `/mnt/kiss`, and `/mnt/user`. `/mnt/usb` stores some low-level drivers, `/mnt/kiss` stores the KISS-C compiler and `cbcui`, and `/mnt/user` (the only writable partition by default) stores user programs and camera models.

In general, `/mnt/user` is safest to mess with, followed by `/mnt/kiss`, `/mnt/usb`, and finally the Chumby's NAND (which we do not recommend touching at all).

More information about the Chumby (including some excellent technical documentation relevant to the CBC) can be found at the Chumby website [2].

4. Extracting Firmware Updates

A good first step to reverse-engineering the CBC firmware is to take a look at a CBC firmware update file, called `userhook0`. Try opening it in your text editor of choice (we used Programmer's Notepad [3]). Here are the first few lines of `userhook0` v1.0.0.

```
#!/bin/bash
### THIS IS A CHUMBY BOTBALL CONTROLLER FIRMWARE UPDATE ###
VERSION=1.0.0
echo "#!/usr/bin/perl -w" > /tmp/extract.pl
echo "open IN, \"<\$ARGV[0]\"; seek(IN,\$ARGV[1],0); \$count=\$ARGV[2]; while
(\$count) { \$count -= read(IN, \$_, \$count < 1024 ? \$count : 1024); print;
}" >> /tmp/extract.pl
chmod +x /tmp/extract.pl
EXTRACT_fb_print='/tmp/extract.pl userhook0 8192 30107'
EXTRACT_ext2_ko='/tmp/extract.pl userhook0 38299 49317'
EXTRACT_vfat_tgz='/tmp/extract.pl userhook0 87616 1974030'
EXTRACT_ext2_tgz='/tmp/extract.pl userhook0 2061646 37385129'
EXTRACT_rcS='/tmp/extract.pl userhook0 39446775 1278'
EXTRACT_block_probe='/tmp/extract.pl userhook0 39448053 8367'
EXTRACT_mkdosfs='/tmp/extract.pl userhook0 39456420 35288'
EXTRACT_ptable_bin='/tmp/extract.pl userhook0 39491708 512'
EXTRACT_mkfs_ext2='/tmp/extract.pl userhook0 39492220 795765'
#!/bin/sh

#upgrade

set –x

if echo $0 | grep userhook0 > /dev/null; then
```

If you've used Linux, this should be ringing a bell. It's just a shell script. Take a look a few pages down. It appears to be binary data appended to the shell script. And look at the script –

see all those EXTRACT lines? It certainly looks like we could take those numbers after the EXTRACT lines and use them as offsets and sizes.

Let's try that. Open up a Cygwin or bash prompt on your PC, and navigate to the location of `userhook0`. Let's try to extract `vfat_tgz`; the name contains a useful-sounding file extension. Type the following Cygwin command:

```
split --bytes=2061646 userhook0 removed_end_
```

Several files will be spewed out. What we've done is chopped up the `userhook0` file into chunks 2061646 bytes in size. Meaning that the first split in the file is at the boundary between `vfat_tgz` and the next file, `ext2_tgz`. So, delete all but the first resulting file (`removed_end_aa`).

Next, we want to remove the stuff before `vfat_tgz`. To do this, we simply do a similar command on the resulting file, but with the size 87616 (the start of `vfat_tgz`).

```
split --bytes=87616 removed_end_aa removed_start_
```

Run it, and more files will be spewed out. Now to remove the stuff before `vfat_tgz`, we delete the first file, `removed_start_aa`. What to do with the rest of the files?

```
cat removed_start_* > vfat.tgz
```

If you've done this correctly, `vfat.tgz` will now open with WinRAR. Extract the files from `vfat.tgz` and have fun inspecting them! (Feel free to delete the `removed_end` and `removed_start` files you created.) You can utilize the same method to extract `ext2_tgz`. `vfat_tgz` is installed onto `/mnt/usb` on the CBC, while `ext2_tgz` is installed onto `/mnt/kiss`.

5. The `system()` Function

Probably the most helpful piece of knowledge to a CBC hacker is that the CBC runs standard Linux. This means that many useful POSIX functions are available to your KISS-C programs. The most useful of those functions is `system()`. `system()` takes a string as input, and executes that string on the command line. Since you can do almost anything in Linux from a command prompt, `system()` is extremely powerful. For example, this simple KISS-C command will reboot the CBC, demonstrating the power available to the user when utilizing the `system()` function:

```
system ("reboot");
```

In this paper, whenever we give shell commands (commands for a Linux command prompt) for the CBC, you can run them by using the `system()` function. We will usually not include the `system()` function in our examples, because there are other ways of executing shell commands, which we will explain shortly.

6. Our Automatic Installer

Rather than fill this paper with source code, we've combined all of our CBC modified files into one automated USB installer, provided as a .zip file on the GCER CD. Simply unzip the install files to the root of a USB flash drive, insert the drive into your CBC, and run `robot.c` from the Programs menu. Press A when prompted, and all of the mods discussed in this paper will be installed for you. It takes about 45 seconds for us, but the speed is probably dependent on your flash drive. This setup allows us to only discuss the important and/or noteworthy details in this paper, and you can read the entire source code for yourself by looking at the files in the .zip file.

7. Network Support

Many of the hacks detailed in this paper require some kind of network access between your CBC and your PC. There are two ways of accomplishing this: wired Ethernet, and Wi-Fi. Wired Ethernet is probably cheaper, and is often easier to set up, but Wi-Fi has the obvious advantage of being wireless. (We've found that lugging around a long Ethernet cable is very annoying.)

Our installer adds support for both wired Ethernet and Wi-Fi, assuming that you have a supported adapter. We've had success with the Linksys USB200M (wired) and the D-Link WUA-1340 (Wi-Fi). Simply boot your CBC with the network adapter inserted. The network will be connected during the boot sequence.

If you need your CBC's IP address, you can run this CBC program, which will print the IP:

`whatismyip.c:`

```
int main()
{
      printf("Your IP is: \n");
      system ("ifconfig | grep inet | grep -v 127.0.0.1 | sed
's/Bcast.*//'");
      printf("If not shown, your network is disconnected.\n");
      return(0);
}
```

8. Wi-Fi Configuration

If you are using Wi-Fi, you will need to enter your network settings. Run the following CBC program to get to the Wi-Fi control panel screen:

`control_panel.c:`

```
int main()
{
      printf("Press A to Launch Control Panel...\n");
      while(! a_button()) ;
      system ("/usr/chumby/scripts/start_control_panel");
```

```
	return(0);
}
```

If you don't know any of the settings it prompts you for, ask your network administrator. After you enter your Wi-Fi settings, they will be saved. You only will need to rerun the control panel if you change Wi-Fi networks.

9. SSH

The `system()` function is useful for executing shell commands, but it's cumbersome to have to load a KISS-C program through USB, just to run a few preprogrammed `system()` commands. There is a better way: SSH. SSH (secure shell) is a Linux program used to remotely access a computer, giving you a shell prompt. You can do anything from an SSH prompt that a `system()` call can do, but it's interactive and more efficient.

First, you'll need to give your CBC network access as described above. Once you've done that, create the following shell scripts in a new directory called "`code`" on your PC:

```
setip:

#! /bin/bash
echo $1 > cbcip

shell:

#! /bin/bash
CBCIP=`cat cbcip`
ssh root@$CBCIP
```

Once you've done this, open a Cygwin or bash prompt on your PC, navigate to the code directory (using the `cd` command), and type `./setip 192.168.1.5` , where `192.168.1.5` is the IP address of your CBC. A config file called `cbcip` will be created, so you will not have to run setip again until your CBC changes its IP address. Then simply type `./shell` . If you get a connection error, check the IP address of your CBC, and also check your network connection. If it works, you will be presented with a nice ASCII Chumby logo, followed by a Chumby shell prompt. You can now execute shell commands on the CBC, directly from your PC. If you like inspecting things (as we do), feel free to browse around the file system, using the `cat` command to see the contents of files. The Botball firmware installs in `/mnt/usb`, `/mnt/kiss`, and `/mnt/user`, so you may wish to look through those directories to gain an understanding of how the firmware works. To exit the Chumby SSH session and get back to your PC's Cygwin or bash prompt, type use the `exit` command.

10. End of Part 1

That's it for Part 1 of Hacking the CBC. We hope to see you in Part 2!

References

[1] Charmed Labs. Cygwin Installer. http://www.charmedlabs.com/index.php?option=com_docman&task=doc_details&gid=33&Itemid=44 , 2006.
[2] Chumby Industries. http://www.chumby.com , 2009.
[3] S. Steele. Programmer's Notepad. http://www.pnotepad.org/ , 2009.