CBC Hacking 2010 (Part 1)
Jeremy Rand, Matthew Thompson, Braden McDorman
Norman Advanced Botball, Nease High School, Norman Advanced Botball
biolizard89@gmail.com, matthewbot@gmail.com, braden@betabot.org

CBC Hacking 2010 (Part 1)

1 Introduction

When we published *Hacking the CBC Botball Controller* at GCER 2009 [1], we said that CBC hacking was still in its infancy. We expected many advances to be made in the year following the paper's release. We were right. With the release of the CBCv2 in January 2010, hackers were given full source code to the CBC firmware, allowing new innovative hacks to be developed without extensive reverse-engineering. In CBC Hacking 2010, we will document our hacks developed over the past year, and explain how readers can develop their own hacks. Let's get started!

DISCLAIMER: Installing unofficial firmware on your CBC carries an inherent risk of bricking your CBC. Usually, reflashing an official firmware will cure such a brick, but in rare cases this will not work. KIPR's warranty does not cover damage caused by an unofficial firmware, and we are unable to offer a warranty ourselves (but if something does happen, please do notify us so that we can attempt to help you fix it). If this concerns you, that should be a hint that CBC hacking is not for you.

2 Analysis of KIPR's 2010 Changes

The CBCv2 firmware contains various changes and improvements over that of the CBCv1. Probably the most important change is that all of the CBC software (except KISS-Sim) is now open-source under the GNU General Public License (GPL). The code is available on KIPR's GitHub [2]. This is a huge help to hackers, who now no longer have to reverse-engineer every last detail of the CBC's workings.

A major change within the actual software was that Chumby firmware was upgraded to 1.7.1 [3]. The new Chumby firmware uses the EABI binary format instead of the older OABI format used by the CBCv1's Chumby firmware. The effect is that programs written for the CBCv1 must be recompiled (using updated compilers) for use on the CBCv2. The three large CBCv1 hacks which needed a recompile are the custom Wi-Fi driver, the VNC plugin, and the GDB debugger. We will discuss these hacks in their respective sections.

Another major software change, which KIPR announced at GCER 2009, was that the CBOB now has a bootloader. We were quite excited to hear this, but upon looking through the bootloader's source code, we were disappointed to find that it does not prevent bricks from bad firmwares; it only allows a new firmware to be loaded if the existing firmware works properly. As we don't have a JTAG for unbricking a CBOB, and we have minimal confidence in our

ability to keep our modifications safe (we don't even know which toolchain the CBOB firmware was built with), we decided not to touch the CBOB. If any KIPR officials are reading this paper, please, make a proper brick-proof bootloader so that the hackers can innovate!

The last major software change is the Chumby-CBOB link. In the CBCv1, a fixed-format packet was exchanged between the Chumby and the CBOB roughly every 25ms. This caused immense lag, as the round-trip ping was approximately 50ms. Since the CBOB sits between the Chumby and the Create, Create commands lagged even worse; we encountered pings to the Create that took 80ms or longer. The CBCv2 uses a different protocol. Each time a sensor is read or a motor is written to by a user program, a packet is sent to the CBCv2. The ping is approximately 6ms. This is better than the 50ms ping from the CBCv1, but this is for each sensor and motor command, rather than for all commands at once. It also means that consecutive motor commands are implemented at least 6ms apart, which actually makes motor synchronization (e.g. for locomotion) worse than on the CBCv1, which executed all motor commands at once (unless the programmer was sufficiently unlucky that the packet with 50ms ping got sent between his/her motor function calls).

There are many other changes; these can be seen by looking through the GitHub commits from KIPR programmers Jorge Villatoro and Matt Roman between GCER 2009 and now [2].

3 NHS Patchset

We have developed an alternative userhook0 for the CBCv2, called the NHS Patchset. The file is publicly available at our GitHub [4]. The NHS Patchset contains many new features and improvements over the official KIPR userhook0, which this paper will discuss. Before continuing with this paper, we recommend installing the latest userhook0 from the NHS Patchset. Simply load onto your CBC as you would for an official firmware update.

4 Built-In Network Control Panel

Chumby firmware 1.x's built-in Wi-Fi driver does not yet incorporate the Wi-Fi fixes which we contributed to Chumby last year, and the driver we compiled last year does not work on the newer firmware, so we recompiled the latest Wi-Fi driver from Chumby, with our improvements merged in. This enabled Wi-Fi on the CBCv2 with the same feature set which the CBCv1 had.

While we were working on Wi-Fi, we got tired of having to launch and then kill the Chumby Control Panel to connect to or switch networks. Among other grievances we had were that the Chumby Control Panel conflicts with `cbcui`'s framebuffer and touchscreen drivers, that the Chumby Control Panel hangs the CBC when the network does not have reachable Internet access (common for Botball networks), and that we wanted network support to be hot-swappable.

To fix these issues, network support was built directly into `cbcui` in the form of the Wi-Fi Panel, accessible from the firmware's main menu by clicking Utilities → Wifi . The present version can tackle both wired Ethernet and normal Wi-Fi operations, like viewing a list of networks and connecting to both open networks and those encrypted using WEP, WPA Personal, and WPA2 Personal. It does lack some functionality from the Chumby Control Panel, like connecting to

hidden SSIDs. It also lacks an onscreen keyboard for entering encryption keys, opting instead to load them from a USB drive. It does, however, contain a far more streamlined connection process than the Chumby Control Panel, connecting to networks or reporting an error in only a few seconds. It also cleanly handles multiple removals and insertions of the wireless adapter, automatically reconnecting to the previous network in only a few seconds.

5 Brightness Control Panel

The Brightness Control Panel is a simple addition to `cbcui` that enables the CBC's backlight to be adjusted. To further enhance the CBCv2's battery life, the screen can also be set to dim or turn off after a certain period of inactivity. The Brightness Control Panel was the first moderate size enhancement to the CBC software stack to be merged by KIPR, and is set to appear in KIPR's next official software release.

6 CBOB Driver Fixes

The NHS Patchset also contains an optimized CBOB driver. Its main improvement is that it can communicate at the CBOB's rather particular timings without resorting to busy-waiting (a nice term for wasting CPU cycles). This change has a big impact on the CPU usage while communicating with the Create, entirely eliminating the need for a kernel work queue that would previously require as much as 40% CPU usage. It also gives small reductions to `cbcui` CPU usage on the graph and sensor screens. As a side effect of some difficult debugging, the driver also tolerates and can eventually recover from desynchronizations with the CBOB.

7 Miscellaneous CBC-Side Improvements

The NHS Patchset firmware has various other minor improvements. Shell scripts can be executed from the File Manager, which previously required a PC with SSH. In addition, custom startup scripts can be placed in `/mnt/user/code/$codedir/chumby_startup.sh`, where `$codedir` is any directory. This allows hackers using the NHS Patchset to add startup code without interfering with each other's hacks or requiring modifications to the `/mnt/usb/` or `/mnt/kiss/` partitions.

The compile script also has some enhancements: it now displays the last modified date of the program, and plays a sound effect when downloading or compiling code. (The sound effect feature was added by KIPR several months after we developed it.) Both of these enhancements serve to reduce the risk that a programmer will repeatedly download wirelessly to the wrong CBC and not notice what is happening (an error that cost us a total of several hours last year).

8 Code::Blocks CBC Integration: Alternative to KISS-C

While KISS-C [5] is a nice IDE, it is not well-suited to our hacks. Our alternative to KISS-C is a set of customizations to integrate the Code::Blocks IDE [6] with the CBC. At the moment, these instructions are for Windows PC's, but we are working on a Linux-compatible version as well.

Because the relevant tools are not readily available for Mac OS X, OS X will probably not be supported anytime soon (but if you find a way to make it work, please let us know!). To set up the Code::Blocks IDE with our CBC customizations, follow these steps:

1. (WARNING: to avoid issues in steps 2-6, make sure that you extract/install to a path that DOES NOT have spaces.)
2. Extract the `.zip` file provided with this paper to an easily accessible location.
3. Install Cygwin from `http://www.cygwin.com/setup.exe`. In the installer, make sure to install all of the default packages, plus `openssh` and `rsync`.
4. Install the CodeSourcery G++ Lite for ARM GNU/Linux package from `http://www.codesourcery.com/sgpp/lite/arm/portal/package5386/public/arm-none-linux-gnueabi/arm-2009q3-67-arm-none-linux-gnueabi.exe`. Chumby recommends version `2008q3-72`, but we used version `2009q3-67` with no obvious problems. We make no guarantee that this upgrade will not introduce subtle problems which will cost you weeks of debugging time.
5. Install RealVNC from `http://www.realvnc.com/products/free/4.1/download.html`. Only the viewer is necessary; you don't need the server.
6. Install Code::Blocks from `http://downloads.sourceforge.net/codeblocks/codeblocks-8.02mingw-setup.exe`. In the installer, make sure that CB Share Config gets installed.

7. Congrats, all necessary software is installed!

8. Start Code::Blocks from the Start Menu; answer any prompts, and then exit Code::Blocks.
9. Start the CB Share Config program from the Code::Blocks menu in the Start Menu.
10. Click the ... next to Source Configuration File.
11. Choose the CBC IDE Backup file which is provided in the pc_tools directory of the `.zip` file you extracted.
12. Click the ... next to Destination Configuration File.
13. If you are on Windows 7, Navigate to the `AppData\Roaming\codeblocks\default.conf` file; `AppData` will be in your user profile directory. Windows XP and Linux users may have a different but comparable path.
14. Check each box on the left.
15. Click Transfer.
16. Click the ... next to Source Configuration File.
17. Choose the CBC User Backup file which is provided in the `pc_tools` directory.
18. Check each box on the left.
19. Click Transfer.
20. Close CB Share Config.

21. Congrats, Code::Blocks can now compile CBC programs!

22. Start Code::Blocks.
23. Click Settings → Global Variables.

24. Replace the default settings in the right-hand column with those for your computer:
    a. `cbcip` should be the IP address reported by the CBC's Network Control Panel.
    b. `utilsbin` should be the path to the bin directory of your Cygwin installation.
    c. `armlinux_path` should be the path where you installed CodeSourcery G++ Lite.
    d. `cbc_path` should be the code directory from the `.zip` you extracted.
    e. `gdbport` should be the TCP port on which you would like the CBC's `gdbserver` to run. If you're not sure, leave it at the default and it should work fine.
    f. `vnc` should be the full path and filename of the `vncviewer.exe` which RealVNC installed.
25. Open the `CBCProject.cbp` project which is provided in the `code/CBCProject/` directory of the `.zip` file you extracted.
26. Click File → Save Project As User-Template.
27. Enter the name `CBCProject`.
28. Open the `libCBCLibrary.cbp` project which is provided in the `code/libCBCLibrary` directory of the `.zip` file you extracted.
29. Click File → Save Project As User-Template.
30. Enter the name `libCBCLibrary`.

31. Congrats, Code::Blocks is now configured for your computer! See the next section for usage instructions.

9 Creating a CBC Program with Code::Blocks

1. In Code::Blocks, click File → New → From User-Template.
2. Choose the CBCProject template, a directory where you would like the project's code to be stored, and a new name for the project.
3. For typical CBC programs, on the Build Target drop-down menu, choose Release. The other targets will be explained later in this paper.
4. To add existing `.c`, `.cpp`, or `.h` files to the project, click Project → Add Files.
5. To create new files, click File → New → Empty File, and choose to add the file to the project when prompted. You will be asked to type a filename; make sure that you choose the correct `.c` or `.h` extension depending on what type of file you are creating (`.c` files are compiled stand-alone; `.h` files are `#include`d from other files).
6. To compile a program, click the Build button on the toolbar (it looks like a gear). It will automatically be downloaded to the CBC if the CBC is present, and placed on the CBC's Run toolbar. A current known bug is that the program name in the Run toolbar doesn't update; this is purely cosmetic and it will still run the new program. For the OCD among us, compiling the .bin file from the CBC menu will refresh the Run toolbar, as will rebooting the CBC.

7. TROUBLESHOOTING: If your downloads are timing out and you are sure that the IP address for your CBC is correct, you can try increasing the `networktimeout` variable under Settings → Global Variables. Keep in mind that increasing this value will increase the amount of time which Code::Blocks waits before concluding your CBC is disconnected and canceling the download, so setting it too high can be annoying if you're trying to compile code without being connected to a CBC. Similarly, if your connection

quality is good, you can decrease the value (the default is 20, but 2 works well for me on most networks), which will make Code::Blocks much more responsive.

10 Code Completion

While writing code, Code::Blocks's code completion feature will display all available CBC functions in a drop down menu as you type. Be warned that some functions only work on the CBC, while some others only work on the Simulator.

11 Tools

The Tools menu in Code::Blocks houses some useful features which we thought CBC users might want to access directly from Code::Blocks, without going to the Start Menu or a command prompt. These are:

- Run Via SSH
  - Executes your Release-targeted CBC program within an SSH shell [7], allowing the program's I/O to be visible from your PC's command prompt window. Useful for testing a program without walking over to the CBC or messing with VNC.
- Remote Desktop Via VNC
  - Launches a VNC remote desktop session; see VNC Remote Desktop Enhancements for details.
- Interact/Debug Via GDBServer
  - Launches your Debug-targeted CBC program in a debugging session; see Fixing GDB Debugging for details.
- SSH Shell
  - Opens an SSH connection [7] to your CBC and displays the bash prompt. Refer to a manual on Linux and Bash for information on how to use this. Be warned that you will be logged in as the root user, meaning that you can really screw stuff up if you type the wrong commands (although a competent Linux user should be able to use it safely).
- Fix Permissions
  - If you get interesting errors indicating permission errors, try this Tool and see if it fixes the error.
- [ADVANCED] Unprotect KISS Partition
  - Remounts `/mnt/kiss/` in writable mode. This can be useful if you're creating hacks and need to modify a firmware file, but don't want to re-download a `userhook0` file. As improper modifications of the KISS partition can cause problems that can only be fixed by a `userhook0` reflash, we do not recommend this Tool's use unless you know what you are doing.
- [ADVANCED] Start GDBServer
  - Starts `gdbserver` on the CBC with your Debug-targeted program, but does not launch the GDB prompt on the PC side. See Fixing GDB Debugging for details.
- [ADVANCED] Connect to GDBServer
  - Connects to a `gdbserver` instance that you started with [ADVANCED] Start GDBServer. See Fixing GDB Debugging for details.

12 Faster Compiles with Cross-Compiling

Last year, we explained how to get faster download speeds by using Wi-Fi rather than the CBC's USB/serial port. However, download speeds are not the whole story. The other factor is compile speed. KISS-C does the compile operation on the CBC-side. Because the Chumby is much slower than a PC, this presents a bottleneck. Our solution is to use a cross-compiler to compile Chumby programs on the PC-side. This is the same compiler (CodeSourcery G++ Lite for ARM GNU/Linux) [8] which was used to build the CBC firmware. Our Code::Blocks customizations include full support for the Chumby cross-compiler. When you build a program from within Code::Blocks, it is compiled using CodeSourcery G++ Lite and then downloaded in binary form to the CBC via Wi-Fi.

13 Even Faster Downloads with `rsync`

Last year's mods used `cp` with an `ssh` tunnel [7] to download code. This introduces some bottlenecks for large projects, because every file in the project must be re-copied over Wi-Fi each time the user downloads. A better solution is `rsync` [9]. `rsync` is a Linux program similar to `scp` (Secure Copy) which only transfers changes in files rather than the entire file. Unchanged files are not transferred at all. The Chumby firmware does not include the `rsync` daemon, but we were able to compile it with the CodeSourcery compiler and load it onto the CBC. With our `userhook0`, the `rsync` daemon starts on boot, and Code::Blocks is configured to download using `rsync`.

14 End of Part 1

We're out of space for Part 1. Be sure to check out Part 2 for more hacks, and to find out how you can make your own hacks. We'll see you in Part 2!

References

[1] J. Rand, M. Thompson, B. McDorman. Hacking the CBC Botball Controller: Because It Wouldn't Be a Botball Controller if It Couldn't Be Hacked. Proceedings of the 2009 Global Conference on Educational Robotics, July 2009.
[2] KISS Institute for Practical Robotics. CBC. http://github.com/kipr/cbc/, April 2010.
[3] Chumby Industries. Chumby Firmware Switching Toolchains.
http://forum.chumby.com/viewtopic.php?id=3737, April 2009.
[4] J. Rand, M. Thompson, B. McDorman, KIPR. NHS Patchset.
http://github.com/matthewbot/cbc/downloads, May 2010.
[5] KISS Institute for Practical Robotics. KISS-C. http://www.botball.org/kiss, December 2009.
[6] The Code::Blocks Team. Code::Blocks. http://www.codeblocks.org/, May 2010.
[7] Wikipedia contributors. Secure Shell. http://en.wikipedia.org/wiki/Secure_Shell, June 2010.
[8] CodeSourcery. GNU Toolchain for ARM Processors.
http://www.codesourcery.com/sgpp/lite/arm, 2010.
[9] W. Davison. rsync. http://samba.anu.edu.au/rsync/, December 2009.