Hacking the Wallaby

Julian Palmanshofer, Julius Pürzelmayer, Manuel Reinsperger, Lukas Rysavy
(all in HTL Spengergasse: "Almighty 7")

***Abstract*—This paper shows an approach on how to significantly improve the software on the KIPR Wallaby in both security and performance. This is especially important, because the security measures already installed on the device are by far not enough and can be overcome by anyone who has a bit of experience in Linux operating systems and access to a Wallaby. Another point is performance, which is crucial for the Botball tournament, as well as the questionable usability of the C programming language for beginners.**
We want to highlight the parts of the Wallaby software that are either not secure or not conforming to our standards of performance and usability. The results includes compiling different firmware for the controller and show how to improve the current one to the point where unauthorized access is prevented and usability is increased to the point where an actual IDE can be used for programming the robots.

***Index Terms*—Computer languages, Runtime library, Computer performance, Microcomputers, Robot programming.**

I. Introduction

The authors identified multiple problems in the sectors of security, usability and extendability and give a quick overview of these in this section.

The first and biggest problem of the Wallaby is security. After looking into the `wifi_configurator.py` file, one can see that the WiFi password is just the Wallaby ID hashed and modified. Since the ID is also shown in the SSID, the password can be recovered from just looking at the SSID. After connecting, anyone can execute commands using SSH, given that the owner of the Wallaby has not set a root password.

Another problem with the WiFi is that it is started in the `.bashrc` file, which is normally used for defining aliases and variables for the bash. This leads to the password being printed to the terminal, which allows anyone that passes by to read it.

Since the Wallaby itself does not support connecting to another network by default, it is unable to query software repositories, and therefore it is not possible to install new software or upgrade already installed packages.

Both the BotUI (the user interface on the touch screen) and the web interface are big issues concerning usability. The web interface, which is meant for programming and compiling source code, does not provide essential features for beginners like auto-completion or error highlighting and locks the user into one programming language.

In addition, the touch screen interface does not always respond to touches, forcing the user to repeatedly press a button before the program realizes that it was clicked.

What is more, using the C programming language may not suit the needs of some programmers and may not be the best choice for people who are new to programming, since it is not the easiest to learn.

Since modifying the software on a fundamental level can lead to breaking the operating system, there also has to be a way to backup and restore all data on the internal storage.

II. Concept

To address these issues we propose the following changes and modifications which will be discussed in greater detail in the next section.

One easy and quick solution to the most vulnerable security hole, the WiFi password, would be manually changing it to something static which cannot be calculated from other parameters. Another change in WiFi configuration is not to start it in the `.bashrc` file, but instead enable the service and let it start automatically.

The next important security measure is to set a root password or enable private key authentication, respectively. This ensures that unwarranted access to the Wallaby is prevented, even if someone manages to gain access to the WiFi provided by the Wallaby. However, it has to be said that arbitraty code execution is not fully prevented when solely relying on locking the root account as the web interface is still fully operational with only WiFi authorization.

Since updating installed packages is required for ensuring perfect operation, an internet connection needs to be established. Therefore, the Wallaby either needs to be connected to an external wireless hotspot directly or a computer which connects to the internal WiFi network and an ethernet cable needs to be used to forward traffic between these two interfaces. After this is accomplished, updates can be installed.

The easiest solution to the user and web interface is to disable or uninstall it completely and work using SSH from that point on. Of course, rewriting the interface would also be an option, but that is not in the scope of this paper.

The problem that the C programming language may not be suitable can be overcome in many ways. One approach is to write bindings for the libwallaby library in any common scripting language like Python [1], Perl [2] or Ruby [3].

Since making mistakes in any of the above operations can lead to the destruction of the operating system, creating an image of the internal storage which can be restored afterward could prove to be useful.

III. Implementation

As the steps to get to an, in the authors opinion, ideal Wallaby can be complex to some, we have put together a detailed description on all necessary steps.

A. WiFi

The standard way to change the WiFi password to a static, not as insecure one which can be chosen individually is to modify the file `/usr/bin/wifi_configurator.py` and change the line:

```
wpa_passphrase=
                'wpa_passphrase='+hash_id+'\n'
```

to:

```
wpa_passphrase='wpa_passphrase=password\n'
```

This will update the WiFi configuration file to use your password every time the WiFi starts. In the same file, the SSID and channel can be modified.

However, since the configuration does not need to be overwritten every time the WiFi starts, the following section could be removed. From:

```
# ==== generate the hostapd.conf file ====
```

To:

```
# ==== Start Wi-Fi ====
```

If these lines are removed, the WiFi can be configured in the file `/etc/hostapd_wallaby.conf`. To start the WiFi on startup the line

```
systemctl start wifi.service
```

has to be removed from the `.bashrc` file. After that, the command

```
systemctl enable wifi.service
```

enables the WiFi daemon, registering it for an automatic start as soon as the operating system is loaded [4].

B. Authentication

To set a root password, the command `passwd` is used. To ensure additional security, password authentication over SSH can be disabled by modifying `/etc/ssh/sshd_config`, adding or modifying the following lines to match:

```
ChallengeResponseAuthentication no
PasswordAuthentication no
UsePAM no
```

However, at least one public key has to be added to `~/.ssh/authorized_keys` before restarting the ssh daemon with

```
systemctl restart sshd
```

Otherwise, logging in via SSH is not possible anymore and the key has to be added by plugging in a keyboard to one of the USB ports on the Wallaby. Then ssh keys have to be either typed in by hand or copied from a USB stick before successful login will be possible again.

C. Software installation

In order to use the package manager which is installed on the Wallaby, `smart` [5], one has to establish a connection to the internet. In order to do that, a PC or laptop running Linux has to be connected to the WiFi of the concerned Wallaby, as well as other networks which have access to the internet (e.g. using an ethernet cable).

After that is accomplished, IP address forwarding has to be enabled on the PC or Laptop, here a Linux system, using the command:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Then, the firewall has to be configured to not block packets that should be forwarded to the Wallaby. For that, three commands are needed:

```
iptables -A FORWARD -i wlan0 -o eth0
    -j ACCEPT
iptables -A FORWARD -i eth0 -o wlan0
    -m state --state ESTABLISHED,RELATED
    -j ACCEPT
iptables -t nat -A POSTROUTING -o eth0
    -j MASQUERADE
```

In this case, `wlan0` is the interface which is connected to the Wallaby and `eth0` is the interface which is connected to the internet. Now, the default gateway of the Wallaby needs to be set using

```
route add default gw 192.168.125.2
```

where `192.168.125.2` is the IP address of the PC. After these steps are completed, the Wallaby should be able to access the internet.

The first thing that should be done after connecting to the internet is to perform package upgrades. Upgrades can be performed using `smart upgrade`, which updates around 600 packages in the current version (2016/03/01).

The `smart` command is also used to install new packages. To do that, pass install as the first and the package name as the second argument. These names can be determined using

```
smart search <keyword>
```

In addition, once the package manager works unneeded packages (e.g. PHP) can be uninstalled in order to free memory which can be used for other software. This can be accomplished using the shell command

```
smart remove <package name>
```

D. User and Web-Interface

As stated before, the touchscreen interface and the web interface are not needed in order to write programs for the Botball contest. Therefore, disabling them grants more processing power to other applications and lets the battery last longer. Using SSH, the web interface and the touchscreen user interface can be disabled using the following two commands

```
systemctl disable xserver-nodm.service
systemctl disable harrogate.service
```

However, both services can be uninstalled completely using the package manager (see section 'Software installation'), but at a tournament the programs would therefore have to be started using a directly connected keyboard.

E. Python bindings

To increase efficiency while programming, we have written a port of the `libwallaby` library to python called `walla.py` [6]. It uses the python ctypes module to call functions from the C library and implements some of the functions in native python code. Some of these native functions include a direct serial connection to the IRobot Create or a `wait_for_light` method that is safe from being triggered by camera flashes.

However, some of the original functions like msleep or the threading functions are excluded, because there are python functions which performs the same action.

There has been another attempt at porting the `libwallaby` to Python by PembrokeRobotics [7], however this is as of 2016/04/04 less a library and more a collection of methods and files that don't allow full interaction with the Wallaby and none with the Create and is therefore not recommended for use. Our library on the other hand does provide all the needed functionality and more.

Our library can be used in combination with our Eclipse plugin [8], which handles syntax checking and code completion as well as uploading and running the code on the Wallaby. This enables the developer to remotely control the Wallaby as well as to read any output their program produces on their PC or laptop screen. The plugin is meant to be used together with the `pyball` package which supplies applications to set up the Wallaby and install all needed dependencies like the `walla.py` library or a run script automatically.

F. Backup

The Wallaby stores it's operating system on an 8GB SD card. This card can be removed by opening the Wallaby, the "Warranty void if removed" sticker does not need to be removed for any disassembly. One should still remain careful as any broken parts have to be rebought on own budget.

To create an image of the card, there are two standard options: using `dd` and using `cat`. The command for `cat` would be

```
cat /dev/sdb > wallaby.img
```

while the one for `dd` would be

```
dd if=/dev/sdb of=wallaby.img
```

In these examples, `/dev/sdb` is the SD card (which can be determined using `fdisk -l`) and `wallaby.img` is the output file. To restore the backup, one simply has to swap these two parameters. Alternatively, other programs like most partition managers can be used for backing up all partitions on other operating systems like Microsoft Windows.

G. SD Card

By default, the Wallaby has four partitions, a boot partition, a file system partition, a second file system and a user file system. The second file system contains a development build of the Wallaby firmware. To switch to the development build the `L` button need to be pressed down during the boot process. The user file system is a shared partition between the main firmware and the development build.

H. Replacing the OS

Even though the Wallaby becomes a lot more usable with the aforementioned changes there are still a few things that cannot be done with the resulting setup. To reach those a very skilled and determined Botball team can compile their own firmware to use on the Wallaby.

1) Requirements for compiling the firmware: Unfortunately, building the firmware is complicated and requires skills in Linux. The Wallaby is based on a Linux distribution called OpenEmbedded [9]. OpenEmbedded provides a build tool called `bitbake`.

There is little to no documentation to set up a build environment for compiling the Wallaby firmware and the repository of the firmware is outdated which is why we provide a quick overview of the process.

A Linux computer or virtual machine is required for setting up a build environment, an Ubuntu-based distribution is preferred. The Linux computer must have at least `150GB` of disk space and `16GB` main memory, or at least enough swap memory for the linker to finish the compilation.

2) Setting up the build environment: In order to build the firmware, the `repo` tool [10] is required. To install the tool run the following commands:

```
curl commondatastorage.googleapis.com/
    git-repo-downloads/repo > repo
chmod a+x repo
mv repo /usr/local/bin/
```

The `repo` tool is used for downloading different git repositories that are required for building an OpenEmbedded based Linux. Which repositories are required is defined in a `manifest` file and usually provided by the developers of an OpenEmbedded based Linux. Unfortunately, KIPR does not provide a `manifest` for the Wallaby firmware. A workaround for this problem is to use the `gumstix` manifest. Gumstix [11] has designed and produces the Pepper single board computer [12], the board the Wallabies and their firmware is based on. To initialize a `repo` client run the commands:

```
repo init -u git://github.com/gumstix/
    yocto-manifest.git
repo sync
```

The `repo` tool should now download most of the required repositories for building the Wallaby firmware. Some changes have to be made in order to provide the missing repositories for compiling. To do that, enter the `poky` folder and download the missing meta-kipr [13], meta-browser [14], meta-qt5 [15] repositories.

3) Compiling the Wallaby Firmware: After entering the working directory, run the following commands:

```
export TEMPLATECONF=meta-kipr/conf
source ./poky/oe-init-build-env
bitbake kipr-console-image
```

The `bitbake` tool will now download and compile every single tool that the Wallaby firmware needs. It will take about six hours to compile the complete firmware, even on a decent computer. During the first time compiling the firmware some errors might occur and need to be fixed. Bitbake saves the current process and will continue where it stopped.

4) Flashing the Wallaby: After the compile process is finished, `bitbake` will create a compressed archive containing the root filesystem for the Wallaby. To flash this created filesystem, you have to take the SD card out of the Wallaby and insert it into a computer. The SD card needs to be mounted, usually, this happens on its own. Now, the new filesystem can be copied to the SD card with the following commands:

```
rm -rf <mount point of rootfs1>
tar xaf ./tmp/deploy/images/pepper/
    kipr-console-image-pepper.tar.bz2
    -C <mount point of rootfs1>
```

The other partitions should not be touched by inexperienced users. After the new filesystem has been copied, the SD card needs to be unmounted and placed back inside the Wallaby. The Wallaby should now boot successfully.

IV. Conclusion

To sum up, the software of the Wallaby can be improved in security, usability, and performance. After all modifications, unauthorized access is prevented, unneeded software is uninstalled, other software can be downloaded and installed and Eclipse can be used to program the robot using Python. Furthermore, custom firmware can be compiled and installed on the Wallaby. These changes enable the team that builds the robot to work in a much more efficient way, leaving more time for other tasks like assembling the robot. This can be a crucial factor, especially for last-minute changes, but also during the entire development period. Further modifications may include using a different operating system or even directly program the controller without using any higher level system at all. Since the hardware may not be modified, the processor architecture has to be kept the same, but other Linux distributions can be compiled for ARM.

References

[1] Python Software Foundation,
Easy scripting language for beginners to experts,
https://www.python.org/ [2016/04/03]

[2] The Perl Programming Language,
Highly capable, feature rich language,
https://www.perl.org/ [2016/04/03]

[3] Ruby Programming Language,
A dynamic, open source programming language,
https://www.ruby-lang.org/ [2016/04/03]

[4] Digital Ocean,
How to use systemctl to manage systemd services and units,
https://www.digitalocean.com/community/tutorials/
how-to-use-systemctl-to-manage-systemd-services-and-units
[2016/04/03]

[5] Linux Documentation,
smart - the smart package manager,
http://linux.die.net/man/8/smart [2016/04/03]

[6] Almighty 7,
Walla.py contains python bindings for the libwallaby library,
https://gitlab.com/Almighty7/wallapy [2016/04/03]

[7] PembrokeRobotics,
Repository for work on the 2016 BotBall competition,
https://github.com/PembrokeRobotics/Botball-2016 [2016/04/04]

[8] Almighty7,
PyBall Eclipse plugin,
https://gitlab.com/Almighty7/pyball [2016/04/03]

[9] OpenEmbedded,
Cross-compile environment for embedded systems,
http://www.openembedded.org/ [2016/04/04]

[10] Google,
Tool for git repository management and Android automation,
https://code.google.com/p/git-repo/ [2016/04/04]

[11] Gumstix,
Gumstix computers and boards,
https://www.gumstix.com/ [2016/04/04]

[12] Gumstix,
Pepper 43C single board Computer,
https://store.gumstix.com/computers/pepper-43c.html/ [2016/04/04]

[13] KISS Institute for Practical Robotics,
BitBake configurations for the Wallaby,
https://github.com/kipr/meta-kipr [2016/04/03]

[14] O.S. Systems Software LTDA.,
OpenEmbedded/Yocto BSP layer for web browsers,
https://github.com/OSSystems/meta-browser.git [2016/04/03]

[15] meta-qt5,
QT5 layer for openembedded,
https://github.com/meta-qt5/meta-qt5.git [2016/04/03]