

---

# BLACKBERRY Z10 RESEARCH PRIMER

---

“Dissecting Blackberry 10 – An initial analysis”

Alexander Antukh



SEC Consult Vulnerability Lab, Vienna, 05/2013

V 1.0

---

## TABLE OF CONTENTS

---

Abstract.....	3
Introduction .....	4
Getting shell access to the device .....	9
Permissions .....	10
System utilities.....	14
Fuzzers .....	19
/boot.....	21
Playing with the browser.....	30
Future research / afterword .....	33
References.....	35

---

## ABSTRACT

---

In 2013, Blackberry has presented a brand new operating system which significantly differs from others presented on the smartphone market. A very high security level is announced, and the expectations are corresponding. Some analytics consider this as the last chance for Blackberry “to get back in the big game” and stand in the row with such giants as iOS and Android.

The goal of this whitepaper is to show an approach for testing the new Blackberry 10 operating system and to identify vulnerabilities on a new Blackberry 10 device.

A set of methods and tools has been developed. In the paper we will:

- Discuss specifics of the operating system
- Check for vulnerabilities “by design”
- Talk about fuzzers
- Test default utilities
- Dump the “boot sector”
- Mention other interesting entry points / notices
- Propose further steps for future research

---

## INTRODUCTION

---

### OPERATING SYSTEM

Blackberry OS is a closed-source Unix-like mobile operating system based on QNX. Until now, Blackberry is the only mobile device family which supports such a secure OS. Let's look closer at it (Figure 1)

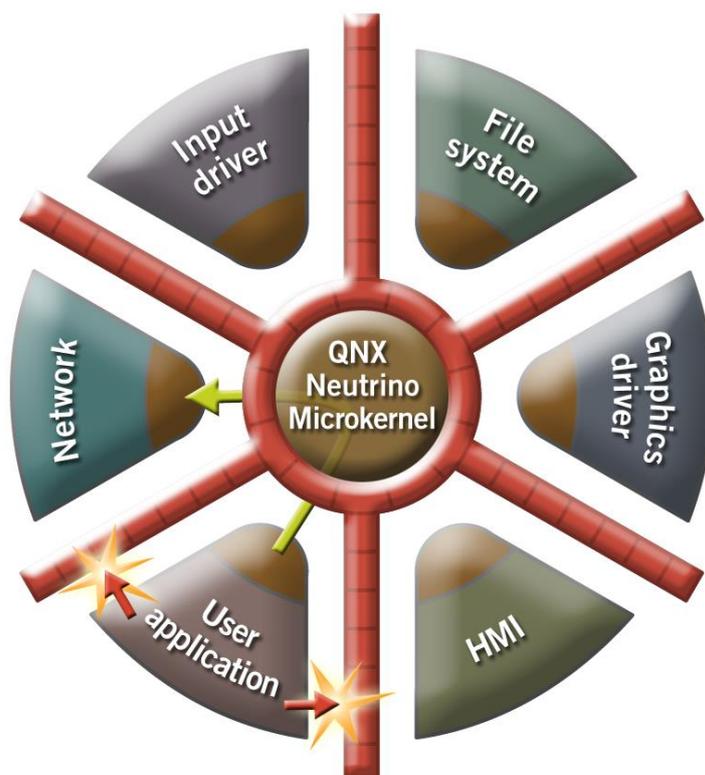


FIGURE 1: QNX SIMPLIFIED SCHEME. SOURCE: CRACKBERRY.COM

QNX is built on a principle of micro kernel architecture (1). That means that except the micro kernel (MK) itself and the process manager (PM) everything else, including drivers, runs as a process. Every process is managed by a PM and handed off to the MK for execution. If an application tries to write into memory that it doesn't control, the PM will recognize it as an address not allocated to the app, and tell the kernel to shut down the problem on the spot. An application only has access to where it was permitted by OS to have access to.

A schematic figure of a MK is presented in Figure 2.

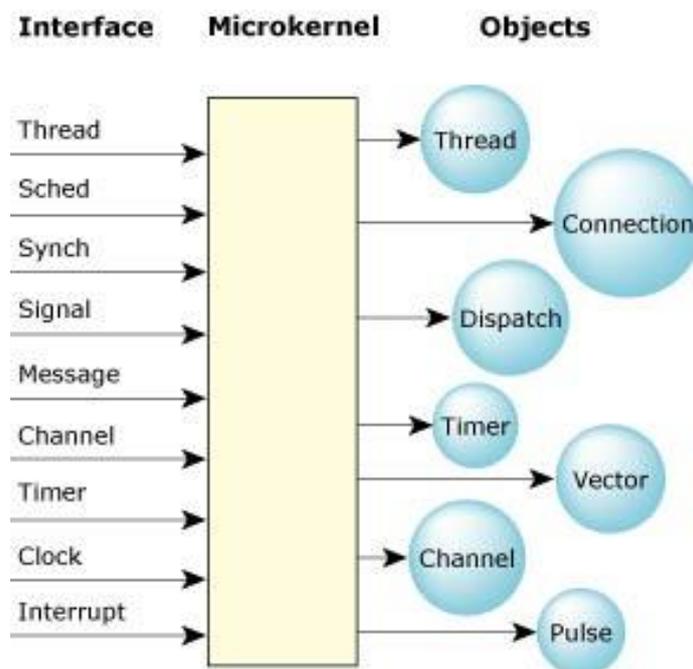


FIGURE 2: SCHEMATIC FIGURE OF A MICROKERNEL. SOURCE: QNX.COM

QNX interprocess communication consists of sending a message from one process to another and waiting for a reply. This is a single operation, called *MsgSend*. The message is copied, by the kernel, from the address space of the sending process to that of the receiving process. If the receiving process is waiting for the message, control of the CPU is transferred at the same time, without a pass through the CPU scheduler. Message handling is prioritized thread priority.

The boot loader is the other key component of the minimal microkernel system (2). Because user programs can be built into the boot image, the set of device drivers and support libraries needed for startup need not be, and are not, in the kernel. Such functions as program loading are not in the kernel, but instead are in shared user-space libraries loaded as part of the boot image. It is possible to put an entire boot image into ROM, which is used for diskless embedded systems.

QNX Neutrino microkernel supports symmetric multiprocessing and processor affinity called bound multiprocessing (BMP). It is used to improve cache hitting and to ease the migration of non-SMP-safe applications to multi-processor computers. QNX Neutrino is also fully POSIX-compliant. It means that it's very easy to develop an application just like in Unix, Linux and MacOS.

## ARMV7 PROCESSOR ARCHITECTURE

General characteristics:

- 32-bit RISC processor
- 32-bit addressing – linear address space 4 Gb in size
- Thirty one 32-bit general registers plus six state registers
- Cyclic shift device and multiplier
- Three-stage pipeline
- Big Endian / Little Endian operating modes
- Fully static performance
- Fast interruption response (real-time)
- Virtual memory systems support
- Simple but powerful system of commands

A block-scheme of the ARMv7 kernel is presented on Figure 3.

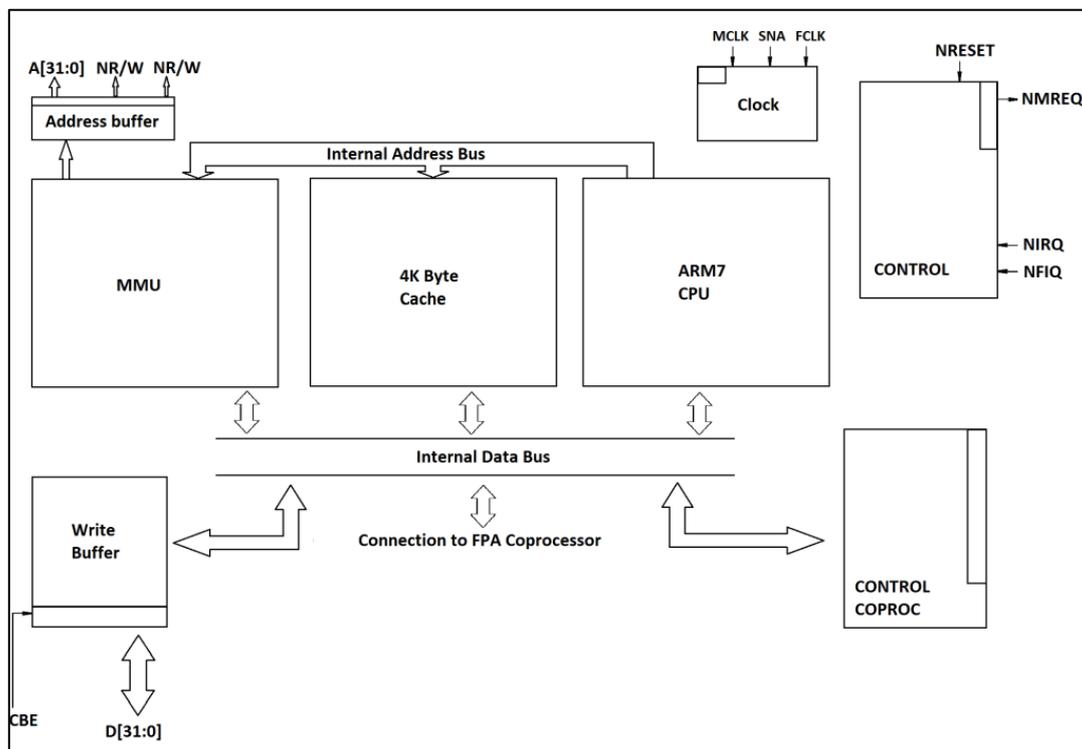


FIGURE 3: BLOCK SCHEME OF THE ARMV7 KERNEL.

Responsible: A. Antukh  
Version/Date: 1.0/23.05.2013  
Confidentiality Class: Public

---

Notable improvement comparing to previous ARM architectures is the Thumb-2 technology. This is a technology first introduced in the ARM1156 core, which extends the limited 16-bit instruction set of Thumb with additional 32-bit instructions to give the instruction set more breadth, thus producing a variable-length instruction set. A stated aim for Thumb-2 was to achieve code density similar to Thumb with performance similar to the ARM instruction set on 32-bit memory. In ARMv7 this goal can be said to have been met. Thumb-2 extends both the ARM and Thumb instruction set with bit-field manipulation, table branches, and conditional execution. A new "Unified Assembly Language" (UAL) supports generation of either Thumb-2 or ARM instructions from the same source code; versions of Thumb seen on ARMv7 processors are essentially as capable as ARM code (including the ability to write interrupt handlers).

## TEST DEVICE

The test subject used in this project was RIM's latest smartphone, Blackberry Z10. Model number is STL100-2. The installed firmware version is 10.0.10.690. The following list contains the full version information:

- OS: BlackBerry 10
- OS Version: 10.0.10.690
- QNX Version: 8.0.0
- Flash Player Version: 11.1.121.108
- AIR Version: 3.1.0.108
- Cryptographic Kernel Version: 5.6.2.44214
- WLAN Version: 1.1
- Radio Version: 10.0.10.691
- WebKit Version: 10.0.10.251
- Browser Version: 10.0.10.288

Blackberry Z10 is shipped with a large set of applications which are presented below.

- BlackBerry Hub
- Contacts
- BlackBerry Browser
- BlackBerry Calendar
- BBM
- Text Messages
- BlackBerry World
- BlackBerry Remember

Responsible: A. Antukh  
Version/Date: 1.0/23.05.2013  
Confidentiality Class: Public

---

- Docs To Go
- Multimedia (Pictures, Music, Videos)
- Story Maker
- Social network apps (Facebook, Twitter, LinkedIn, Foursquare)
- BlackBerry Maps
- Games
- YouTube
- Voice Control
- Weather
- Clock
- Calculator
- Compass
- File Manager
- Box
- BlackBerry Connect for Dropbox
- Print To Go
- Smart Tags
- Adobe Reader
- Camera/Video Camera/Time Shift

---

## GETTING SHELL ACCESS TO THE DEVICE

---

The first problem a researcher has to solve in order to start the analysis is to connect to the device. Luckily, this can be easily done in case of Blackberry Z10. The first step in order to be able to connect remotely is to turn on developer mode (Settings → Security and Privacy → Development Mode → On). After that, it is necessary to initiate a key exchange from the host machine. This might be done at least in two ways:

- 1) Using Blackberry SDK Tools
  - a. Generate a 4096-bit RSA key using “ssh-keygen -b 4096” on a Linux box or with puttygen.exe (SSH2-RSA) on Windows.
  - b. Invoke “blackberry-connect” tool with the following syntax:  
`<blackberry-connect> <target-host> -password <device-password> -sshPublicKey <ssh-public-key>`
  - c. You’re connected now. Don’t close the window, launch another terminal/putty and SSH to the IP specified in developer’s mode menu on a device (default is 169.254.0.1)
- 2) Using Dingleberry (for lazies)
  - a. Click on “Install SSH” → “Dingle SSH” :]

Dingleberry is a jailbreak tool used to root Blackberry PlayBook. Current version is 3.3.2. Obviously, the jailbreak doesn’t work for the Blackberry 10, however it comes very handy when one wants to quickly connect to the device.

After the connection has been established, a console opens with the home folder of `/accounts/devuser/`. Let’s explore what we have in there.

## PERMISSIONS

The known vulnerability which led Blackberry Playbook up to version 2.0.0.6149 to jailbreak, was based on an incorrect permission setting which the *setuidgid* utility initially has had. By default, a limited-privilege *devuser* was able to copy and launch the utility which gave him an ability to launch */bin/sh* as a root. Currently this is not possible anymore for either the PlayBook or for the new Blackberry OS.

As stated, Blackberry OS was built as a very secure system with a strong privilege separation mechanism. The first sight confirms that – even different social networking software has its own *uid:gid*, in contrast with having a single user responsible for all the installed applications (see Figure 4)

```
$ cd sys.facebook/
$ ls -l
total 2
-rw-rw----+ 1 100731000 10073      1024 Mar 15 11:55 default
$ cd ../sys.linkedin/
$ ls -l
total 2
-rw-rw----+ 1 100741000 10074      1024 Mar 15 11:54 default
```

FIGURE 4: ACLS FOR SOCIAL NETWORKING APPLICATIONS

Thanks to this approach, even if an attacker manages to find and remotely exploit a buggy application, he wouldn't have much to do in there until a way to exploit privilege escalations is being discovered.

In here, to check permissions set on files and folders, we used a simple script which uses the functionality of a *find* system utility. In particular, worldwide write permissions were checked for all files and folders, as well as *sgid/suid* bits. Partial output of the script (for writable files) is presented on the Listing 1.

```
('pps', '-rw-rw-rw- 1 root      nto                25 Apr 16 11:59
./services/audio/audio_router_control')
('pps', '-rw-rw-rw- 1 root      nto                9 Apr 11 11:29
./services/automation/navigator/control')
('pps', '-rw-rw-rw- 1 root      nto                187 Apr 16 15:51
./services/automation/navigator/output')
```

Responsible: A. Antukh  
 Version/Date: 1.0/23.05.2013  
 Confidentiality Class: Public

```

('pps', '-rw-rw-rw- 1 root      nto                8 Apr 11 12:16
./services/automation/navigator/notify')
('pps', '-rw-rw-rw- 1 pps-bt-media bluetooth      9 Apr 12 14:37
./services/bluetooth/public/control')
('pps', '-rw-rw-rw- 1 apps       air_services     12 Apr 15 08:47
./services/dialog/control')
('pps', '-rw-rw-rw- 1 keyboard-imf nto            12 Apr 16 11:59
./services/input/control')
('pps', '-rw-rw-rw- 1 nowplaying now_playing     9 Apr 15 13:59
./services/multimedia/mediacontroller/control')
('pps', '-rw-rw-rw- 1 nowplaying now_playing     9 Apr 11 12:16
./services/multimedia/mediaplayer/control')
('pps', '-rw-rw-rw- 1 root      nto                12 Apr 12 14:37
./services/multimedia/rend')
('db', '-rw-rw-rw- 1 media      media           16 Apr 11 11:29 ./strg/BASE.SVB')
('db', '-rw-rw-rw- 1 media      media           36864 Apr 15 08:05
./strg/IMAGECORE.SVD')
  
```

#### LISTING 1: WRITABLE FILES

Unfortunately, the output file for `suid` analysis was empty. The picture for `sgid`-bit looks similar – the only exception is in several folders owned by `1000_shared` group, which is a default group for a `devuser`.

However, it should be noted that one can write to a folder (and modify files) owned by `media` user (see Figure 5)

```

$ pwd
/db/strg
$ ls -l
total 1537
-rw-rw-rw- 1 media      media           4096 Apr 10 13:58 0.SVI
-rw-rw-rw- 1 media      media           4096 Apr 08 11:36 1.SVI
-rw-rw-rw- 1 media      media           4096 Apr 08 11:36 2.SVI
-rw-rw-rw- 1 media      media           4096 Apr 10 13:58 3.SVI
-rw-rw-rw- 1 media      media           4096 Apr 08 11:36 4.SVI
-rw-rw-rw- 1 media      media           4096 Apr 08 11:36 5.SVI
-rw-rw-rw- 1 media      media           4096 Apr 08 11:36 6.SVI
-rw-rw-rw- 1 media      media            16 Apr 11 11:29 BASE.SVB
-rw-rw-rw- 1 media      media           36864 Apr 10 13:58 IMAGECORE.SVD
-rw-rw-rw- 1 media      media          536576 Apr 10 13:58 JPEG_256X256_personal.SVD
-rw-rw-rw- 1 media      media          36864 Apr 08 11:36 JPEG_256X256_tonelibrary.SVD
-rw-rw-rw- 1 media      media          36864 Apr 08 11:36 JPEG_384X384_personal.SVD
-rw-rw-rw- 1 media      media          36864 Apr 08 11:36 JPEG_384X384_tonelibrary.SVD
-rw-rw-rw- 1 media      media          36864 Apr 08 11:36 JPEG_768X768_personal.SVD
-rw-rw-rw- 1 media      media          36864 Apr 08 11:36 JPEG_768X768_tonelibrary.SVD
  
```

FIGURE 5: WRITABLE FILES OWNED BY MEDIA USER

Responsible: A. Antukh  
Version/Date: 1.0/23.05.2013  
Confidentiality Class: Public

Generally, the initial permission analysis didn't result in exploitable findings. However, an interesting detail appeared while the script was running. Several crashes occurred with the following output:

```
*** stack smashing detected ***: find terminated
```

Buffer overflow in a system utility? Keeping in mind that the most one will get is code execution with privileges of a *devuser*, only superficial analysis was made. The crash occurs when the *find* utility tries to open a directory which it doesn't have permissions to. In Figure 6, contents of a register used as an argument to open a directory, are presented. If we decode it from hex to characters, we'll have the following string:

```
/accounts/1000/appserv/sys.cfs.dropbox.gYABgKi0Cs_hMocaoCB7UgqkaIU/fs
```

```
Breakpoint 9, 0x78005938 in ?? ()
(gdb) x/20xw 0x77ff2e34
0x77ff2e34: 0x6363612f      0x746e756f      0x30312f73      0x612f3030
0x77ff2e44: 0x65737070      0x732f7672      0x632e7379      0x642e7366
0x77ff2e54: 0x62706f72      0x672e786f      0x67424159      0x4330694b
0x77ff2e64: 0x4d685f73      0x6f61636f      0x55374243      0x616b7167
0x77ff2e74: 0x662f5549      0x54520073      0x6766546d      0x00007338
(gdb)
0x77ff2e84: 0x00000000      0x00000000      0x00000000      0x00000000
0x77ff2e94: 0x00000000      0x00000000      0x00000000      0x00000000
0x77ff2ea4: 0x00000000      0x00000000      0x00000000      0x00000000
0x77ff2eb4: 0x00000000      0x00000000      0x00000000      0x00000000
0x77ff2ec4: 0x00000000      0x00000000      0x00000000      0x00000000
(gdb) c
Continuing.
*** stack smashing detected ***: findfind terminated
Program received signal SIGABRT, Aborted.
```

FIGURE 6: MEMORY AREA CONTAINING INPUT ARGUMENT FOR OPENDIR()

In the *fs* folder there's another folder marked as read-writable. This is what happens when we try to proceed in it (see Figure 7)

Responsible: A. Antukh  
Version/Date: 1.0/23.05.2013  
Confidentiality Class: Public

```
$ pwd
/accounts/1000/appserv/sys.cfs.dropbox.gYABgKi0Cs_hMocaoCB7UggkaIU/fs
$ ls -l
total 0
drw-rw-rw-  1 100361000 10036          0 Apr 11 11:28 mounts
$ cd mounts/
sh: cd: /accounts/1000/appserv/sys.cfs.dropbox.gYABgKi0Cs_hMocaoCB7UggkaIU/fs/mounts - Permission denied
```

FIGURE 7: THE CRASH REASON

The exact place where the reason of the crash is presented is shown on an excerpt from IDA in Figure 8.

```
.text:0000592A          LDR          R3, =(unk_12730 - 0x5936)
.text:0000592C          ADD.W       R2, R8, #1
.text:00005930          MOU        R0, R4 ; name
.text:00005932          ADD        R3, PC ; unk_12730
.text:00005934          STR.W     R2, [R3, #(dword_135C0 - 0x12730)]
.text:00005938          BLX        opendir
```

FIGURE 8: IDA LISTING OF A CRASH PLACE

Responsible: A. Antukh  
Version/Date: 1.0/23.05.2013  
Confidentiality Class: Public

---

## SYSTEM UTILITIES

---

As we could see, it's actually not difficult to identify the buffer overflow. The other question is – what's the profit? Since we are *devuser*, we already can run shell and python scripts as well as compiled binaries built for QNX. So our next target is actually not to find buffer overflows but to enumerate tools to have an idea what we are dealing with.

Of course, the most interesting folder for us is */proc/boot/*. There are not only tools which are launched when the system starts but also configuration files as well as startup scripts and, finally, the microkernel. Of course, all the most interesting stuff is protected from accessing by anyone but root. Let's explore what we have.

In contrast with earlier versions of Blackberry PlayBook, many utilities (i.e. *setuidgid*, *id* and *dumpifs*) are missing. Among the accessible ones, one can distinguish the following tools:

- *confstr* – current configuration including path, architecture and network info
- *dmc* – digital media controller
- *fsmon* – file system monitor
- *jsc* – JavaScript engine for Webkit used on a device
- *ldo-msm* – LDO Driver
- *mkdosfs* – format a DOS filesystem (FAT-12/16/32)
- *mkqnx6fs* – format a filesystem (for QNX6, however, is presented in Blackberry OS)

and also tools such as *mount*, *on*, *nfcservice*, *nvs\_write\_bin* and *displayctl*.

Keeping in mind those vulnerabilities of ACLs, developers have restricted access to many services, hence a limited user cannot fully use almost any of the utilities listed above anymore. However, this leads to funny consequences – some of the utilities (like *find* from the previous chapter) get crashed with null pointer dereferences or even fail the boot to load next time.

Several crashes were registered during testing default utilities (see Listing 2)

```
Process 57340127 (displayctl) terminated SIGSEGV code=1 fltno=11
ip=788293d2 (/base/usr/lib/graphics/msm8960/displayHAL-
r086.so@dsi_get_pclk_freq+0x121) mapaddr=000093d2. ref=00000008
Process 249935086 (nowplaying) terminated SIGSEGV code=1 fltno=11
ip=78102cce (/usr/sbin/nowplaying@main+0x19d) ref=00000000
Process 1547274689 (resource_seed) terminated SIGSEGV code=1 fltno=11
ip=01386d44 (/usr/lib/ldqnx.so.2@_Stoint+0x20) mapaddr=00036d44. ref=00000000
Process 1543295477 (shutdown) terminated SIGSEGV code=1 fltno=11
ip=78117c3e (/proc/boot/shutdown-msm8960.so@pmic_ssbi_read+0x15) mapaddr=00001c3e.
ref=ffffffff
```

Responsible: A. Antukh  
Version/Date: 1.0/23.05.2013  
Confidentiality Class: Public

```
Process 1545237780 (charge_monitor) terminated SIGSEGV code=1 fltno=11  
ip=010b998c(/usr/lib/ldqnx.so.2@message_detach+0x8) mapaddr=0003998c. ref=00000028
```

#### LISTING 2: CRASHES FOUND WHEN REVIEWING DEFAULT UTILITIES

Additionally, due to strict policies it's impossible to open `/dev/i2c2` which is necessary in order to properly use the `displayctl` utility. However, by specifying certain environment variables it's still possible to launch it which leads to crash of a display until the next reboot of the device (see Figure 9).

```
$ on -d displayctl -o -s -b -B1 -f charge_and_drain.bmp  
$ display: 1 nile_init - v:0 to stdout  
display ERROR: 1 MSM8960_gpio_config - pad_init() failed. error=-1  
  
display ERROR: 1 lm3585_hw_init - failed to open /dev/i2c2: Permission denied  
display ERROR: 1 dsi_set_interface_clocks - dsi_set_interface_clocks is not implemented  
alloc_image_bmp  
display ERROR: 1 nile_set_power_state - nile_set power state: Failed to open /dev/npa for read/write  
display ERROR: 1 i2c_read_reg - ../../../../peripherals/backlights/common/i2c_common.c Could not send and receive i2c master  
data  
  
display ERROR: 1 lm3585_hw_set_charge_pump_voltage - Failed to read register R01 - do not change charge pump  
display ERROR: 1 hypernova_panel_get_id - No panel connected (no data returned from ID read)  
display ERROR: 1 hypernova_panel_get_id - No panel connected (no data returned from ID read)  
display ERROR: 1 hypernova_panel_get_id - No panel connected (no data returned from ID read)  
display ERROR: 1 nile_set_power_state - nile_set power state: Failed to open /dev/npa for read/write  
display ERROR: 1 i2c_read_reg - ../../../../peripherals/backlights/common/i2c_common.c Could not send and receive i2c master  
data  
  
display ERROR: 1 lm3585_hw_set_charge_pump_voltage - Failed to read register R01 - do not change charge pump  
display ERROR: 1 hypernova_panel_get_id - No panel connected (no data returned from ID read)  
display ERROR: 1 hypernova_panel_get_id - No panel connected (no data returned from ID read)  
display ERROR: 1 hypernova_panel_get_id - No panel connected (no data returned from ID read)  
display ERROR: 1 nile_set_power_state - nile_set power state: Failed to open /dev/npa for read/write  
display ERROR: 1 i2c_read_reg - ../../../../peripherals/backlights/common/i2c_common.c Could not send and receive i2c master  
data
```

FIGURE 9: CRASH OF A DISPLAY SERVICE

Finally, it is also possible to corrupt the device's non-volatile memory by launching the `nvs_write_bin` utility. Although it is stated in the output that the filesystem is mounted as read-only (see Figure 10), after turning the device off one can never turn it on again without reloading the firmware with Blackberry Link. The only thing one can have in this situation is a brick with annoying blinking red light.

```
login as: devuser  
Authenticating with public key "imported-openssh-key"  
$ nvs_write_bin /lib/firmware/ti1283/nvs_map.bin CABCC88F7201  
Could not open file for writing. Mount fs as rw and try again.: Read-only file system  
$
```

FIGURE 10: COMMANDS WHICH LEAD TO BRICKING THE DEVICE

Responsible: A. Antukh  
Version/Date: 1.0/23.05.2013  
Confidentiality Class: Public

The consequences, however, might be much more serious – such a behavior in both cases means certain system calls and/or devices don't have proper permissions. In the case it is `/dev/hd5`, which links to `/dev/emmc/boot0`, one can disable signature checking and/or even have the jailbreak.

In other words, writing a tool which invokes vulnerable syscalls might lead up to total control over the device.

It is interesting that permissions actually don't look vulnerable there. None of the `/dev/hd*` devices is accessible for anyone but users from `Disk_Drivers` group and `root`:

```
$ ls -alF /dev/emmc/os0  
brw-rw---- 1 root      Disk_Drivers  8,  0 May 13 12:51 /dev/emmc/os0
```

Function of writing to the specified file in the `nvs_write_bin` utility looks like the presented on the Figure 11.

```
{  
    const void *v1; // r501  
    FILE *v2; // r001  
    FILE *v3; // r401  
    signed int result; // r002  
  
    v1 = a1;  
    v2 = (FILE *)fopen64(dword_30DC, "w");  
    v3 = v2;  
    if ( v2 )  
    {  
        fwrite(v1, 0x18u, 1u, v2);  
        fclose(v3);  
        result = 0;  
    }  
    else  
    {  
        perror("Could not open file for writing. Mount fs as rw and try again.");  
        result = 1;  
    }  
    return result;  
}
```

FIGURE 11: LISTING OF NVS\_WRITE\_BIN WRITE FUNCTION (HEX-RAYS DUMP)

In here, variable *dword\_30DC* contains a specified path to open for writing. Let's create by analogy a simple function to see how it works. Program presented on Listing 3 creates a test file in the location */account/devuser*. As a result a new file appears in the folder. Now let's specify another location. By determining the path as */proc/boot*, an error is shown and no files are created.

```
#include <stdio.h>

int main()
{
    FILE *v;
    v = (FILE *)fopen64("/accounts/devuser/test", "w");
    if( !v ){
        printf( "Sad!\n" );
    }
    else{
        printf("Created\n");
    }
    return 0;
}
```

LISTING 3: SIMPLE PROGRAM TO CREATE TEST FILES

However, when the path is at */lib/firmware* (exact is */lib/firmware/ti1283/nvs\_map.bin* taken right from help), the error is shown too but in reality the firmware gets corrupted!

Now when we reboot the device the display looks as in Figure 12.

By abusing this functionality it might be possible to make changes in the firmware and, as mentioned before, exploit it up to jailbreaking the device.

No proof-of-concept has been created for this initial analysis.



FIGURE 12: RESULT OF WRITING IN /LIB/FIRMWARE

---

## FUZZERS

---

No vulnerability research can be done without automation. In this chapter we'll talk about several approaches to find vulnerabilities and get profit with the help of fuzzers.

The first approach is based on IOCTL fuzzing. Although the kernel is not monolithic as we got used to, it still uses certain IOCTLs to 'talk' to certain services via specially crafted IRPs. In particular, *libc.so* library is responsible, inter alia, for handling operations with the filesystem. Since *libc.so* is open for reading for everyone, we can download and look into it. Fortunately for us, *ioctl()* function has plaintext IOCTL codes defined, which can be easily retrieved with the help of a small python script. In order to fuzz them, two scripts were developed.

The first one is the main script (master), which is responsible for cyclic calls of different fuzz options with the given IOCTL code from the list, and creating subprocesses. The other one (slave) is responsible for fuzzing itself. Such architecture has been chosen to be able to catch crashes of the slave processes and accumulate info on the master part. Several fuzzing options are supported:

- Fuzz with no input parameters
- Dynamic search of a max length of the input string and fuzz with overlong strings
- Fuzzing with pre-determined DWORDs for the max length of the input string (usually 1024 bytes)

Typical output of the fuzzer when a crash occurs is presented below:

```
Process 1924486014 (python3.2) terminated SIGSEGV code=1 fltno=11  
ip=011c90c4(/usr/lib/ldqnx.so.2@ioctl+0x113c) mapaddr=000790c4. ref=00000000
```

It is worth to mention that IRP handlers in the Blackberry OS are designed very well – after all the tests only 11 crashes occurred. Unfortunately, all of them (even a couple of intriguing ones with further calls of *j\_fcntl()* and *j\_tcsendbreak()*) only caused denial-of-service.

One of the ideas to continue research in this direction is to implement functionality of a random string fuzzer.

The second fuzzing scenario is based on an idea that vulnerability *EDB-ID #7823* might not be fully closed. Published in 2009, it caused kernel panic in QNX 6.4.0 by launching bitflipped elf binary (*id*). The fuzzer script takes the filename as an argument and writes out two files – bit- and byteflipped binaries which are launched after.

However, the most interesting is the third (yet not finalized) media file format fuzzer. Blackberry Hub allows a user to have simultaneous access to mail, messages, social networks and BBM. Since all those applications use graphical libraries to open files, by exploiting this one can gain much more profit, for example, stealing such sensitive data as emails, contacts and so on.

At least two accessible utilities which use those libraries have been found: *display\_image* and *chat\_service*. The utilities cannot work properly in the context of a *devuser* due to restricted access to several services (in particular, screen context creation). Thus the main problem here is to launch those in order to be able to fuzz. Jailbreak?

---

## /BOOT

---

So far, all we've found were some crashes which only theoretically approach us to the target. What we have currently:

- We have QNX on the device
- We can't access protected areas

Not very nice. When thinking about an alternative approach, a popup window suddenly appeared which offered me to update the firmware. Why not to tamper that?

In several minutes a BAR-file containing a large binary blob has been obtained. The headers of the extracted file can be seen on the Figure 13.



FIGURE 13: FIRST BYTES OF THE EXTRACTED FIRMWARE

It starts with the *mfcq* header (0x6d, 0x66, 0x63, 0x71) which is typical for QNX images. After the header, several partitions are defined (each of them starts with a *qcfp* header). In the description of each partition such info as format/sync time, number of blocks, block size and so on is presented (we'll see that later).

In 2012, an initial research on Blackberry PlayBook was made by Zach Lanier and Ben Nell (1), who published their tools at <https://github.com/intrepidusgroup/pbtools>.

In particular, with the help of those tools it is possible to automatically extract mentioned partitions by using the *qcfm\_parser.py* script.

In our case, we have six partitions (see Figure 14)

Responsible: A. Antukh  
Version/Date: 1.0/23.05.2013  
Confidentiality Class: Public

```
C:\BlackBerry\OS>qcfm_parse.py qcfm.image.com.qnx.coreos.qcfm.os.qc8960.factory_sfi.desktop.BB10_0_10.690.447500.signed
wrote out: qcfm.image.com.qnx.coreos.qcfm.os.qc8960.factory_sfi.desktop.BB10_0_10.690.447500.signed_qcfp_0.bin
wrote out: qcfm.image.com.qnx.coreos.qcfm.os.qc8960.factory_sfi.desktop.BB10_0_10.690.447500.signed_qcfp_1.bin
wrote out: qcfm.image.com.qnx.coreos.qcfm.os.qc8960.factory_sfi.desktop.BB10_0_10.690.447500.signed_qcfp_2.bin
wrote out: qcfm.image.com.qnx.coreos.qcfm.os.qc8960.factory_sfi.desktop.BB10_0_10.690.447500.signed_qcfp_3.bin
wrote out: qcfm.image.com.qnx.coreos.qcfm.os.qc8960.factory_sfi.desktop.BB10_0_10.690.447500.signed_qcfp_4.bin
wrote out: qcfm.image.com.qnx.coreos.qcfm.os.qc8960.factory_sfi.desktop.BB10_0_10.690.447500.signed_qcfp_5.bin
```

FIGURE 14: THE OUTPUT OF QCFM\_PARSER.PY SHOWING SIX PARTITIONS

Among them the most interesting is the largest partition which is an IFS image (*3.bin*) and the system partition (*4.bin*). To have a better idea of what information is placed in the description section of the image, one can use the *chkqnx6fs* utility. Its output is presented on the Figure 15.

```
** Display fs-qnx6 Superblock **
Ondisk format: v4, LE (native)
Format time   : Fri Apr  5 10:10:48 2013
Volume UUID   : 1571a8df-af1f-4e6c-a2c0-85c726996759
Sync time     : Fri Apr  5 10:24:55 2013
Sync sequence : 7 (sblk #0)
Flags         : 00000100
Blocks        : 2490416 total,    20154 used,  2470262 free
Inodes        : 77832 total,      2 used,   77830 free
Block size    : 1024
Reserved blks: 3% (74712 blks)
Alloc groups  : 8
```

FIGURE 15: THE OUTPUT OF CHKQNX6FS FOR THE IFS IMAGE

To extract data from the IFS image, the *dumpifs* utility is used. Unfortunately, we don't have the tool among default ones in a new Blackberry OS. Fortunately, we do have it in a Desktop QNX Neutrino. By specifying the output folder with the help of another script from the PBTtools (*ifs\_parse.py*), the image is perfectly dumped. Inside we have three folders: *base/*, *proc/* and *root/*. While *base/* contains libs and graphical tools, *root/* has the *.profile* file with a defined *BOOT\_LOADER=RIMBOOT* environment variable, *proc/* has the most interesting set of tools for us. In particular, */proc/boot/* is exactly those tools which are launched when the device turns on. It's no wonder the microkernel itself (*procnto-smp-instr*) is presented in there too, which makes it accessible to be reversed later.

The first thing what we can see in the folder is a *.script* file. It has all the information the system needs when starting up and is launched at booting time. An excerpt from the *.script* is presented in Figure 16. Here one can see, among other information, the IFS name which is *qc8960-rimboot-secure*.

Responsible: A. Antukh  
Version/Date: 1.0/23.05.2013  
Confidentiality Class: Public

```
4 @ ../../proc/boot/libc.so.3 /usr/lib/ldqnx.so.2 T @ Welcome to QNX Neutrino Trunk on the Qualcomm MSM8960 (ARMv7 Scorpion core)
  ( @ IFS Name: qc8960-rimboot-secure
    \ @bmetrics bmetrics -c splash_fadeout MALLOC_ARENA_SIZE=16384 LIBC_STRINGS=krait X @bmetrics_cli bmetrics_cli start M
MALLOC_ARENA_SIZE=16384 LIBC_STRINGS=krait d @bmetrics_cli bmetrics_cli service_start ifs MALLOC_ARENA_SIZE=16384 LIBC_STRINGS=krait h
    @bmetrics_cli bmetrics_cli service_start dumper MALLOC_ARENA_SIZE=16384 LIBC_STRINGS=krait @dumper dumper -U 27:412 -S -F
-d /tmp MALLOC_ARENA_SIZE=16384 LIBC_STRINGS=krait h @bmetrics_cli bmetrics_cli service_stop dumper MALLOC_ARENA_SIZE=16384 LIBC_ST
GS=krait p @bmetrics_cli bmetrics_cli service_start resource_seed MALLOC_ARENA_SIZE=16384 LIBC_STRINGS=krait \ @resou
_seed resource_seed dma=0,31 MALLOC_ARENA_SIZE=16384 LIBC_STRINGS=krait l @bmetrics_cli bmetrics_cli service_stop resource_seed MALL
ARENA_SIZE=16384 LIBC_STRINGS=krait $ @ Starting preserved memory...
  1 @bmetrics_cli bmetrics_cli service_start preservedmem MALLOC_ARENA_SIZE=16384 LIBC_STRINGS=krait | @preservedmem prese
dmem -n preservedmem -U 34:34 -p /tmp -m 0660 MALLOC_ARENA_SIZE=16384 LIBC_STRINGS=krait l @bmetrics_cli bmetrics_cli service_stop
reservedmem MALLOC_ARENA_SIZE=16384 LIBC_STRINGS=krait h @bmetrics_cli bmetrics_cli service_start slogger MALLOC_ARENA_SIZE=16384 LI
STRINGS=krait X @slogger2 slogger2 -U 26:25 MALLOC_ARENA_SIZE=16384 LIBC_STRINGS=krait_neon \ @slogger slogger -U 25:25 --
48k MALLOC_ARENA_SIZE=16384 LIBC_STRINGS=krait_neon L @pipe pipe -U 36:36 MALLOC_ARENA_SIZE=16384 LIBC_STRINGS=krait h
metrics_cli bmetrics_cli service_stop slogger MALLOC_ARENA_SIZE=16384 LIBC_STRINGS=krait h @bmetrics_cli bmetrics_cli service_start
erial MALLOC_ARENA_SIZE=16384 LIBC_STRINGS=krait @ starting serial
  X @sh sh /proc/boot/start_serial MALLOC_ARENA_SIZE=16384 LIBC_STRINGS=krait @ ( /dev/ser1 @ 2 /dev/ser1 h @bmetr
_cli bmetrics_cli service_stop serial MALLOC_ARENA_SIZE=16384 LIBC_STRINGS=krait @sh sh /proc/boot/debug_session /dev/ser1 MALL
ARENA_SIZE=16384 LIBC_STRINGS=krait @ " @sh sh MALLOC_ARENA_SIZE=16384 LIBC_STRINGS=krait SYSNAME=nto TERM=ansi HOME=/var PATH=/proc/b
:/base/bin:/base/sbin:/base/usr/bin:/base/usr/sbin:/base/opt/bin:/base/usr/sbin LD_LIBRARY_PATH=/proc/boot:/base/lib:/base/usr/lib:/base/lib/d
:/base/opt/lib BOOT LOADER=RIMBOOT IFS BOOT ENV=yes @ starting trustzone-msm8x60
```

FIGURE 16: EXCERPT FROM THE .SCRIPT FILE

Another interesting file there is *ifs\_variables.sh*, which is used in all the other shell scripts and provides them with system variables. In particular, information such as GSBI base address and all the links to the configuration files are defined there. All the configuration files are available as well. And one more example is *os\_device\_image\_check* which, as it follows from the name, is used to check the compatibility of the image with the device.

Apart from those, several curious tools are now available. For example, *persist-tool* is able to dump the persistent data areas and obtain bootrom and OS metrics, memory configuration table and other interesting information (see Listing 4). After retrieving the tool from the boot folder, it can be launched with *devuser*. Again, this means system call manipulation is possible which leads to critical consequences (see chapter “System utilities”)

For instance, it is possible to create a tool with similar functionality which will read and dump data from should-be-protected areas.

One of the attack vectors is when such an unprivileged application is able to read sensitive data from another application.

```
Bootrom Metrics:
...
Bootrom Version: 0x0523001D (5.35.0.29)
DeviceString: RIM BlackBerry Device
BuildUserName: ec_agent
BuildDate: Nov 3 2012
BuildTime: 09:10:26
IsInsecureDevice: false
Frequency Map - Version: 0x00000100
Frequency Map - Supported Bands: 0x0000FFFF
Options Supported: 0x0000270D
```

Responsible: A. Antukh  
Version/Date: 1.0/23.05.2013  
Confidentiality Class: Public

```
LegacyEndOfFileSystem: 0x00000000
HWVersionOffset: 0x000000D4
NumberHWVEntries: 0x00000014
MemCfgTableOffset: 0x000000FC
MemCfgTableSize: 0x00000100
Drivers: 0x00000010 [ MMC ]
LDRBlockAddr: 0x2E02FE00
BootromSize: 0x00080000
Processor: 0x01000001
FlashID: 0X00000000
MaxOSSizeSupported: 0x00000004
BRPersistAddr: 0x2E0AFC00
TrustzoneVersion: 0x00000000
TrustzoneBuildFlag: 0x00000000
```

LISTING 4: OUTPUT OF THE PERSIST-TOOL

Another interesting feature of such an approach of analyzing the streams is that much of the data presented in the binary is unencrypted. One can view the source code of different scripts, account information strings and other “curiosities”. For example, by a simple string search it is possible to learn some information about a brand new gesture control system:

```
function setScreenScaling (width, height) {
...
//ZOOM TO POINT IS FULL OF BUGS - Docs state that coordinates should only ever be in center of screen
if ( width < deviceWidth && deviceWidth > deviceHeight ){
    //I need to specify the completely WRONG x value and top coordinate y value in this scenario
    videoScrollView.zoomToPoint( width / 4, 0, ratio);
} else{
    //I need to specify x in center coordinate and y in top coordinate in this scenario
    videoScrollView.zoomToPoint( width / 2, 0, ratio);
}
```

Several other interesting strings might be used in future research (see Listing 5).

```
// TODO: 1229 RC6 has a bug that returns wrong callLogFilter.selectedValue, the
follow line avoids using
```

Responsible: A. Antukh  
Version/Date: 1.0/23.05.2013  
Confidentiality Class: Public

```
// TODO: Once the QML bug about not being to access the page values that are
provided as a parameter to this slot is fixed ...
// The zipfile.ZipFile.write() method has a bug where it raises struct.error:
ushort format requires 0 <= number <= USHRT_MAX
// Too many bytes for PNG signature. Potential overflow in png_zalloc()
```

LISTING 5: COMMENTS EXTRACTED FROM THE UNENCRYPTED STRINGS

Additionally, it is possible to find 'hidden' (otherwise inaccessible) configuration files like those presented in Figure 17.



```
default_groups::core.all,core.fav,core.media,core.games
rotation_enabled::true
demo_mode::0
deactivate_mode::1
autorun::1
autorun_app::sys.firstlaunch.gYABgE1L_1Y.sjW85E1SCBQsrco
systray_manager::sys.systemtray
default_order::/var/etc/default_order
default_dock::/var/etc/default_dock
default_work_wallpaper::/usr/share/wallpaper/wallpaper_work.jpg
demo_on_script::/bin/sh /scripts/demo-mode.sh on
demo_off_script::/bin/sh /scripts/demo-mode.sh off
cleanup_script::/bin/sh /scripts/demo-mode.sh purge
internal_build::0
disable_mirror_lock::1
static_order_files:json:["/etc/static_application_order"]
dyn_order_files:json:["/etc/carrier/branding/application_order"]
dyn_order_triggers:json:["sys.data.ecid"]
screensaver_app::com.rim.bb.app.retaildemoshim
screensaver_delay::3
```

FIGURE 17: "HIDDEN" CONFIGURATION FILE

URI-schemes associated with Blackberry applications, which might be used in cross-browser attacks, can be found the same way (see Figure 18):

Responsible: A. Antukh  
Version/Date: 1.0/23.05.2013  
Confidentiality Class: Public

```
sys.appworld.gYABgNSvaLtte_snIx7wjRsOcyM::appworld://
sys.browser.gYABgJYFHAzbeFMPCCpYwBtHAM0::http://,https://,file://
sys.airtunes.gYABgCwWhIycHhiFjXeIyW1Qvpo::music://
sys.pictures.gYABgFZ.pCiYHqciL1zClEPjimps::photos://
sys.camera.gYABgAvGHb4h9H5WeWdjQhXgeRM::camera://
sys.help.gYABgPG.Su8AzxaqqONbaanIprc::help://
sys.videochat.gYABgHXmq9LYQB023b3XQAWry1k::vchat:
sys.printoutstogo.gYABgPMP3nxNZ1NieZUDetUiQio::ptg://
sys.clock.gYABgKNXug.mDFoFoYHLmJofAts::clock://
sys.pictureeditor.gYABgIRm37_owYKt4P0uCEHSj.o::photoeditor://
sys.video_editor.gYABgOOHXn7fgL9VRbZK9wOga1o::videoeditor://
sys.hotspotBrowser.gYABgF1btu9aXDxJssC7UKfidFU::hotspot://
sys.search.gYABgPp5WMkB_07CE6wzbf1slRQ::search://
sys.simtoolkit_ui_app.gYABgNsM_6zxbmp668bBbRexQiA::simtoolkit://
sys.bridgeMessages.gYABgH_nFAFLgYWPsgIIzCKh7JQ::corp-mailto:,bridge:///messages/
sys.bridgeCalendar.gYABgMyHc.mTKnr5EXdmDe_39e8::bridge:///calendar/
sys.bridgeContacts.gYABgPggBTtmkAlbYC6VFu1G1lo::bridge:///contacts/
sys.bridgeTasks.gYABgGd11iGxTL7m8Y402kyjMXk::bridge:///tasks/
sys.bridgeMemoPad.gYABgNANNsbwVSWZpC4.aBaAv1E::bridge:///memopad/
sys.bridgeBBM.gYABgPzxYrykYf4ijvmGsvE7B1Q::bridge:///bbm/
sys.bridgeApp01.gYABgCDEawLDzU7faEC9XtQaGew::bridge:///app01/
sys.bridgeApp02.gYABgKQs.iUwNLM1CDdenn39w9Q::bridge:///app02/
sys.bridgeApp03.gYABgDzHSEWE78.LHtTApjyFXEY::bridge:///app03/
sys.bridgeApp04.gYABgDkR.SLIz4zMs2afxJj32BQ::bridge:///app04/
sys.bridgeApp05.gYABgEAp.HoYsfaGs3jKkruwRMk::bridge:///app05/
sys.android.shell.gYABgCwPlq.7ipa6NFYT0JaLpt8::android://
com.rim.bb.app.facebook.gYABgDLo0nc9AhDgv2JAPixdyvQ::facebook://
sys.bbm.gYABgLOJBR2Vz7FzS.kdgJchuag::bbmpim://
sys.escreens.gYABgGwZdntFXi6oAVwrCD8H3V8::escreen://
sys.rim.calendar.gYABgG0xyvxB1iABa6DD5o.VL8A::calendar://
```

FIGURE 18: URI SCHEMES ASSOCIATED WITH BB APPLICATIONS

Some internals of the Facebook app which can be used in order to continue the research (such as *APP\_ID*, structure of the requests, hardcoded internal URLs and others) are presented at Figure 19. One can also see those trio of parameters (*access\_token*, *secret*, *session\_key*) presented plaintext on the Figure 20.

Responsible: A. Antukh  
Version/Date: 1.0/23.05.2013  
Confidentiality Class: Public

```
_EVENT Facebook account just being removed %s:%d: %s: received account deletion event! accounts GET Failed to parse PIM response: not JSON
%s provider facebook settings value [SSO] from PIM: access_token is not valid. [SSO] Invalid Account ID. [SSO] Trying REST API.
[SSO] Trying REST API is also failed. No valid account... %s% /pps/services/pim/accounts/settings/ [SSO] PPS Account Info file open fail
led. =====account exist file found, check if registered =====not registered yet, do registration =====unable to g
et access token =====already registered, do nothing =====account exist file does not exist, do nothing Duplicate status message
OAuthException image/ video/ [SSO] Invalid Account Status!!... [SSO] Invalid APP_ID!!... 178645865537776 220353874751486 [SSO] Invali
d PIM APP Access Token!!... fbbb10_gladiator [SSO] Communicating with Facebook Server Failed... error_code error_msg session [SSO] Invali
d FB APP Access Token!!... [SSO] Response parsing error!!... secret [SSO] Invalid FB APP Secret!!... session_key [SSO] Invalid FB APP Ses
sion Key!!... {"access_token": " ", "secret": " ", "session_key": " "} [SSO] sso_authenticate returns valid data.. mail/messages/ /syn
c [SSO] Invalid Account ID!!... accounts/ DELETE data/Settings/push.conf sync ptr 1:%p sync ptr 2:%p 270-3417afr928m0ckc1 http://c
p270.pushapi.na.blackberry.com facebookservice.com.rim.socialconnect.facebook.push_handler SocialConnect Facebook Push Handler plugin initializ
ed provider initialized init_plugin done text url src_path remote_id mime title size auth_operation interrupt_type requ
est_id remove_temp_file token link linkcaption linkname linkdescription place inreplytostatusid type twitter linkedin yout
ube tags privacy album_id statusupdate linkcaption sync_type test test_data locale data/Settings/PushRegistered data/Setting
s/AccountCreated actions=bb.action.SHARE;types=text/*,application/vnd.rim.message.facebook;uris=data://,pim; actions=bb.action.SHARE;type
```

FIGURE 19: EXCERPT OF THE STRINGS OF THE FACEBOOK APP

```
"session_key": "4.0.749ed177c170d14b42b9197a.0-100001481970002",
"uid": 100001481970002,
"secret": "7d9558a9d9a8b2934e6c43c553ce5904"
```

FIGURE 20: CONNECTION PARAMETERS

Not for future research but more for fun (and maybe for profit) – the information about the developers' Facebook accounts including IDs, emails and mobile phones are presented there as well (and yes, even certain passwords are in plaintext! See Figure 21). To check the party and hangout with them one can follow <https://www.facebook.com/bacontrain>

Responsible: A. Antukh  
 Version/Date: 1.0/23.05.2013  
 Confidentiality Class: Public

```

    "error_code": 104,
    "error_msg": "Incorrect signature",
    "request_args": [
      {
        "key": "api_key",
        "value": "220353874751486"
      },
      {
        "key": "format",
        "value": "JSON"
      },
      {
        "key": "method",
        "value": "auth.login"
      },
      {
        "key": "sig",
        "value": "12312412351235"
      },
      {
        "key": "v",
        "value": "1.0"
      },
      {
        "key": "password",
        "value": "asdfasdfasdfasd"
      },
      {
        "key": "email",
        "value": "dprotaso"
      }
    ]
  }

```

FIGURE 21: HARDCODED PASSWORDS

And a couple of words about the *4.bin* file. The system partition has different headers and structure (Figure 22)

```

00000000: 72 69 6D 68-02 00 00 00-66 73 2D 6F-73 20 20 20  rimh  fs-os
00000010: 25 B5 9E 40-E4 DF 85 7E-73 A5 8C 2B-B0 06 75 1A  %RQZ a~sNî+ 2u
00000020: 72 C0 ED 3B-46 F1 C0 C6-F4 2C DE 1D-5D 76 1A DF  r  ;F±L f,  ]v
00000030: FA 8F CE 9E-65 F9 8D 32-B4 D5 8B 72-5C E8 BC 52  ·Ä r e ·i2- fir\0 R
00000040: F1 30 93 03-A8 C8 00 DB-0B B9 29 F9-14 63 05 7D  ±00 ; L  ) ·c}
00000050: AF 53 06 00-B2 42 02 0D-0A 00 00 00-00 00 00 00  »S  B
00000060: 00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00

```

FIGURE 22: FIRST BYTES OF SYSTEM PARTITION

It is possible to mount it to obtain its contents too.

**Note: the mechanism of verifying the integrity of the image is only partially researched thus is not presented in the current version of the paper.**

---

## PLAYING WITH THE BROWSER

---

Another interesting part of the system is the Blackberry browser. Based on the Webkit rendering engine, it has exactly those vulnerabilities as other browsers using this engine (i.e. Google Chrome). An example of a crash is presented in Figure 23.



FIGURE 23: TYPICAL CRASH OF THE BB BROWSER

Due to QNX specifics mentioned in chapter “Operating System”, exploitation of such a vulnerability will be complex in order to steal personal data. In most cases an attacker will only be able to obtain files which the browser (*user:webkit*) has access to.

However, another interesting vulnerability is presented in a current version of the browser. It is known that one can access local files right from the address bar (Figure 24)

Responsible: A. Antukh  
 Version/Date: 1.0/23.05.2013  
 Confidentiality Class: Public

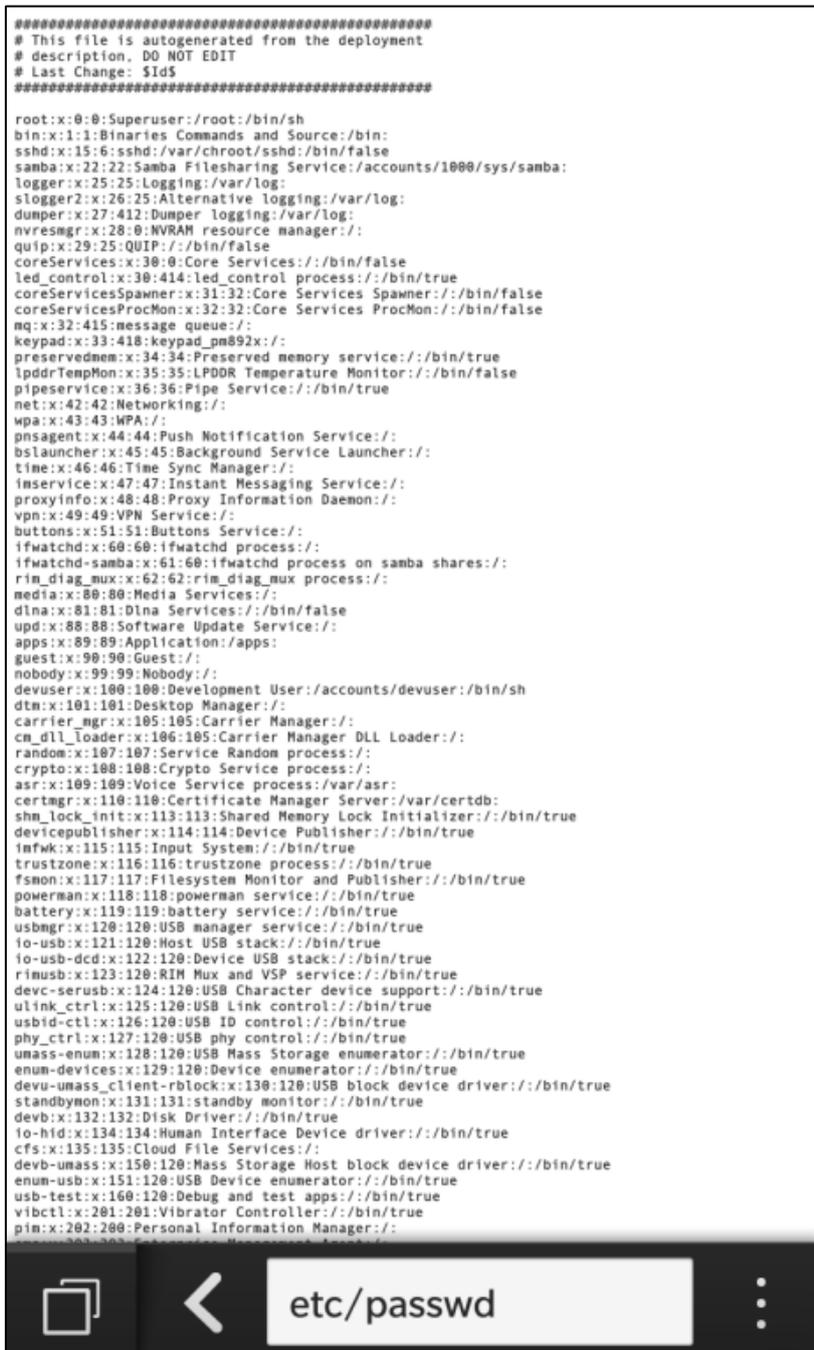


FIGURE 24: ACCESS TO LOCAL FILES FROM THE BROWSER

Responsible: A. Antukh  
Version/Date: 1.0/23.05.2013  
Confidentiality Class: Public

Guys from Nth Direction published an advisory (4) of a Local File Access vulnerability on a Blackberry PlayBook (CVE-2012-5828) where an arbitrary JavaScript code could be executed in the local context of the browser.

### The vulnerability is still present in Blackberry 10.

One of the attack scenarios is the following. An attacker sends an email attachment with the HTML page to a victim. When a victim chooses to open the attachment, it is first saved to the local folder `/accounts/1000/invoke/sys.browser/` and then launched with trusted access to the Blackberry system. From here, it is possible to access any file the browser itself can access. In Figure 25 a HTML page printing 'hello' and creating an iframe with a loaded local `/etc/passwd` file is shown.

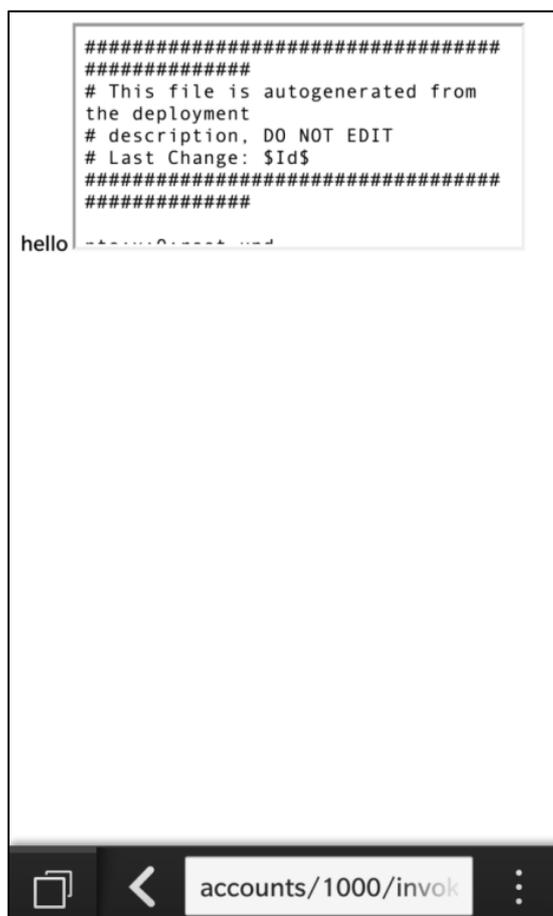


FIGURE 25: DISPLAYING THE HTML FROM THE ATTACHMENT

## FUTURE RESEARCH / AFTERWORD

There are several vectors for future vulnerability research on Blackberry 10.

1) Image parser fuzzing. Displaying image functionality is used in all the default apps including Pictures, Messages, email app (Blackberry Hub) etc. In contrast to the browser, successful exploitation of an image parser would give an attacker access to a wealth of personal data. Currently the two utilities of Blackberry OS (*display\_image* and *chat\_service*) don't have enough privileges to create a context and open an image. However, if one has access to an already rooted Blackberry Playbook, it might be possible to make such a research easier – it's already possible to launch any app thus writing a fuzzer could be done with no problems.

2) Jailbreak. Currently it is possible to get root on the latest version of Blackberry 10 simulator by patching *BlackBerry10Simulator-s001.vmdk* file with the following script:

```
if [ "${BOARD_CONFIG}" != "developer" ]; then  
    rm -rf /root/. > /dev/null 2>&1;  
    fi;cp /usr/bin/setuidgid /tmp && chmod 6755 /tmp/setuidgid;  
(http://pastie.org/private/ytfazol0k6nkgzo6zcb0g)
```

Thus it is possible to get everything from the simulator and to dive deeper in how everything works there. However, one must keep in mind the simulator has many differences compared to the device, in particular, it is not possible to launch binaries compiled for the device, on a simulator, and vice versa. The list of tools is also different.

3) Review another IOCTL family (libsocket.so). IOCTL codes are presented in Figure 26.

LOAD:00005FC6		LDR	R1, =0x80040604
LOAD:00006556	bind	LDR	R1, =0x80100601
LOAD:0000BC7A	getpeername	LDR	R1, =0x40100602
LOAD:0000C54E	getsockname	LDR	R1, =0x40100600
LOAD:0000D728		LDR	R1, =0xEC47
LOAD:0001A59A		LDR	R1, =0x80040605
LOAD:0001A5CC		LDR	R1, =0x40047307

FIGURE 26: IOCTL CODES USED IN LIBSOCKET.SO

4) Detailed analysis of syscall handling.

## 5) Play more with SSH – currently the SSH connection string looks like this:

```
/usr/sbin/sshd -D -f /dev/null -o permitopen="127.0.0.1:8000" -o  
PasswordAuthentication=no -o Protocol=2 -o HostKey=/etc/ssh/ssh_host_rsa_key -o  
AllowUsers=devuser -o DenyUsers=root -o Subsystem=sftp /usr/libexec/sftp-server -R
```

Generating a pair of keys and opening another port on localhost didn't give much results, however, only the initial testing of the method was performed.

A screenshot of strings from the *qconnDoor* utility which is used to establish a connection via SSH is presented in Figure 27.

```
aAllowusersRoot DCB "AllowUsers=root",0 ; DATA XREF: sub_2A00+!  
; .text:off_2C1C↑o ...  
aAllowusersDevu DCB "AllowUsers=devuser",0 ; DATA XREF: sub_2A00+!  
; .text:off_2C20↑o ...  
aPermitrootlogi DCB "PermitRootLogin=yes",0 ; DATA XREF: sub_2A00+!  
; .text:off_2C28↑o ...  
aDenyusersRoot DCB "DenyUsers=root",0 ; DATA XREF: sub_2A00+!  
; .text:off_2C2C↑o ...  
aUsrSbinSshd DCB "/usr/sbin/sshd",0 ; DATA XREF: sub_2A00+!  
; sub_2A00+A0↑o ...  
aD DCB "-D",0 ; DATA XREF: sub_2A00+!  
; .text:off_2C04↑o ...  
aF DCB "-f",0 ; DATA XREF: sub_2A00+!  
; .text:off_2C0C↑o ...  
aDevNull DCB "/dev/null",0 ; DATA XREF: sub_2A00+!  
; .text:off_2C10↑o ...  
aO DCB "-o",0 ; DATA XREF: sub_2A00+!  
; sub_2A00+66↑o ...  
aPermitopen127_ DCB "permitopen=",0x22,"127.0.0.1:8000",0x22,0
```

FIGURE 27: EXCERPT OF THE STRINGS FROM QCONNDOOR UTILITY

## 6) Blackberry Balance is not yet reviewed due to unavailability of BES 10 in Russia.

## REFERENCES

---

---

1. **Crackberry.** Blackberry 10 Review [Online]. <http://crackberry.com/blackberry-10-review>
2. **Blackberry.** Documentation. *Blackberry Developer*. [Online] <http://developer.blackberry.com/native/documentation/bb10/>
3. **Lainer, Zach and Nell, Ben.** 2012 Infiltrate - "Voight-Kampff'ing The BlackBerry PlayBook". *Papers and Presentations / Interpedius Group Mobile Security* [Online]. <http://www.slideshare.net/quineslideshare/voightkampffing-the-blackberry-playbook>
4. **Brown, Tim.** Nth Dimension Security Advisory (NDSA20121030). *Nth Dimension*. [Online] <http://www.nth-dimension.org.uk/pub/NDSA20121030.txt.asc>

### About the Vulnerability Lab

Members of the SEC Consult Vulnerability Lab perform security research in various topics of technical information security. Projects include vulnerability research and the development of cutting edge security tools and methodologies, and are supported by partners like the Technical University of Vienna. The lab has published security vulnerabilities in many high-profile software products, and selected work has been presented at top security conferences like Blackhat and DeepSec.

For more information, see <http://www.sec-consult.com/>