# BlackBerry PlayBook Security: Part one

**Daniel Martin Gomez**
*Principal Security Consultant*
daniel.martin@ngssecure.com

**Andy Davis**
*Research Director*
andy.davis@ngssecure.com


An NGS Secure Research Publication

1 August 2011



http://www.ngssecure.com

# Table of Contents

## 1. Introduction

This is the first in a series of white papers about the security of the BlackBerry PlayBook, the first tablet device released by Research in Motion (RIM) who has had significant success with their BlackBerry smartphones that are used extensively by businesses and consumers around the world. Although the main body of work is primarily aimed at a technical audience, the key findings and conclusions drawn from these are presented at the end of the document.

A breadth-first approach was taken to try to uncover as many components of the PlayBook's attack surface as possible. The primary goal of this phase of research was to gain an understanding of the environment and architecture so that further, more specific research could be planned.

The document is divided into three main sections, covering the security of the Operating System, Applications and Enterprise Data. Each section discusses the main areas of the attack surface exposed by the tablet device. The Adobe Flash and Air runtimes bundled into the PlayBook operating system are not covered in this paper, as the security risks associated with them are considered to be well known.

A combination of first-generation physical PlayBook devices (firmware version 1.0.3) and different VMWare-based PlayBook simulators (mainly version 1.0.1) were used to perform this research.

## 2. Previous work and QNX security history

The PlayBook is based on the QNX Real Time operating system (RTOS)[1] , a POSIX-compliant UNIX-style operating system. It has not really attracted much attention from the security community throughout the years. A review of publicly disclosed vulnerabilities yields around 75 results[2] most of them over five years old.

The majority of these instances are local buffer overflows, many of which affect the command line parsing modules of several applications providing local privilege escalation. Another group of issues corresponds to the insecure use of environment variables by several applications. The most recent[3] (from 2011) is an environment variable arbitrary file overwrite issue affecting Neutrino 6.5.

It should be noted that the majority of the publicly available information and documentation stops at QNX Neutrino 6.5. The PlayBook is based on Neutrino 6.6 (rebranded as "Tablet OS") and it is unclear as to what changes RIM has applied internally to the original 6.5 codebase.

# 3. Research

## 3.1. Operating system

### 3.1.1. Microkernel

Although Neutrino is similar in many ways to a traditional UNIX environment, the QNX microkernel is substantially different from the monolithic Linux kernel. In Neutrino most of the core services such as file system, protocol stack, audio drivers, etc. run in user land. Only address-space management, thread management, and inter-process communication are handled within the microkernel. Figure 1 shows an overview of the microkernel architecture used by QNX.

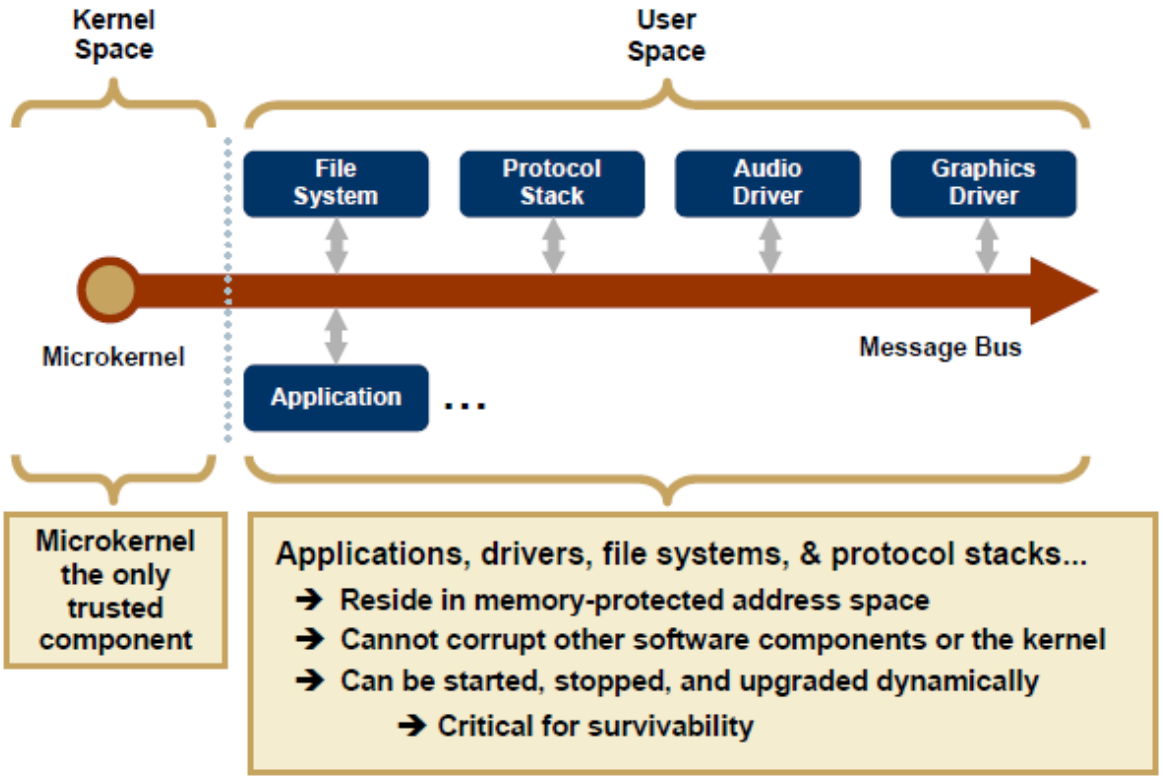


Figure 1: QNX Neutrino Microkernel Architecture (source[4])

### 3.1.2. General considerations

As a general purpose POSIX-compatible UNIX-style environment a large number of tools and binaries were bundled with the main operating system. Apart from general utilities, a Python interpreter and a SQLite3 client were available in the simulator. However, only a few dozen tools remained in the final release of the PlayBook and as discussed below, restrictive file permissions had been set on those that remained.

### 3.1.2.1 Connecting to the device

In order to interact with the PlayBook using a commandline interface, the first step is to set the tablet in development mode through the *Settings -> Security* panel (before development mode can be enabled, a password must be set on the device). Using the *QConn*[5] protocol (see below) the remote machine needs to be authenticated to the device, requiring a password and a strong SSH key pair (at least 4096 bits in length).

An RSA key-pair were generated:

```
ssh-keygen -t rsa -b 4096
```

The file name assigned to the key is subsequently required to establish the *QConn* connection:

```
./blackberry-connect -targetHost <ip> \
                     -devicePassword <password> \
                     -sshPublicKey <ssh-key.pub>
```

The tool *blackberry-connect* is actually just a wrapper around *Connect.jar* (from the SDK[6]) and that the same results could be invoked by calling:

```
java –jar Connect.jar <ip> -password <PIN> -sshPublicKey <ssh-key.pub>
```

Once the connection is authenticated, the standard SSH port (TCP port 22) becomes available and an SSH session can be established using *devuser* username:

```
ssh devuser@<ip> -i <key>
```

### 3.1.2.2 Users and access rights

Early iterations of the simulator ran every application as the *root* user. This has since been changed and applications loaded via developer mode are owned and are run as the user *devuser*. In general each installed application will be assigned their own user and group at install time to enforce the sandboxing model.

The */etc/passwd* file from one of the simulators:

```
root:x:0:0:Superuser:/root:/bin/sh
bin:x:1:1:Binaries Commands and Source:/bin:
sshd:x:15:6:sshd:/var/chroot/sshd:/bin/false
logger:x:25:25:Logging:/var/log:
upd:x:88:88:Software Update Service:/:
guest:x:90:90:Guest:/:
nobody:x:99:99:Nobody:/:
devuser:x:100:100:Development User:/accounts/devuser:/bin/sh
dtm:x:101:101:Desktop Manager:/:
```

The *devuser* has very few privileges and in general not a great deal can be performed using this account. Apart from the traditional *root* account, the *upd* account is the one that has the broadest set of privileges at the file system level (read-write access across a broad number of locations), potentially to allow comprehensive system updates to be performed.

#### *3.1.2.3 File permissions*

A review was performed of the permissions of key components within the file system. Care had clearly been taken to ensure that sensitive files could not be read or altered by low privilege users.

The layout of the *root* (/) partition of the 1.0.1 simulator was as follows:

```
drwxr-xr-x   4 root      root           4096 Jan 15 14:30 accounts
lrwxrwxrwx   1 root      root              9 Apr 20 10:11 air -> /base/air
drwxr-xr-x   5 root      root           4096 Apr 20 09:56 apps
drwxrwxr-x  15 root      root           4096 Feb 17 14:31 base
lrwxrwxrwx   1 root      root              9 Apr 20 10:11 bin -> /base/bin
lrwxrwxrwx   1 root      root             17 Apr 20 10:11 db -> /accounts/1000/db
dr-xr-xr-x   2 root      root              0 Apr 20 10:11 dev
drwxr-xr-x   2 root      root           4096 Jan 15 14:33 etc
lrwxrwxrwx   1 root      root              9 Apr 20 10:11 lib -> /base/lib
lrwxrwxrwx   1 root      root              1 Apr 20 10:11 mlc -> /
lrwxrwxrwx   1 root      root             15 Apr 20 10:11 navigator -> /base/navigator
lrwxrwxrwx   1 root      root              9 Apr 20 10:11 opt -> /base/opt
dr-xr-xr-x   5 root      root              0 Apr 20 09:56 pps
dr-xr-xr-x   2 root      root      899395584 Apr 20 10:11 proc
drwxrwxrwx   2 root      root           4096 Apr 20 09:24 root
lrwxrwxrwx   1 root      root             10 Apr 20 10:11 sbin -> /base/sbin
lrwxrwxrwx   1 root      root             13 Apr 20 10:11 scripts -> /base/scripts
lrwxrwxrwx   1 root      root              1 Apr 20 10:11 slc -> /
lrwx------   1 root      root             10 Feb 09 18:31 tmp -> /dev/shmem
lrwxrwxrwx   1 root      root              9 Apr 20 10:11 usr -> /base/usr
drwxr-xr-x  15 root      root           4096 Apr 20 09:24 var
```

UNIX tools that could be used to search through the file system had either been removed or restricted, significantly increasing the effort required to perform a low level review of the security for the entire file system. In addition, the PlayBook has been formatted using the *qnx6* file system and therefore, it cannot simply be mounted using QNX 6.5 even if the *qnx6* file system driver (*/lib/dll/fs-qnx6.so*) is copied across, which would have enabled the use of tools available in QNX 6.5 to be used instead.

### 3.1.3. Boot-up sequence

The simulator boot-up sequence showed the QNX boot loader and it seems that options could potentially be passed to this loader. A sample boot sequence is shown in Figure 2.

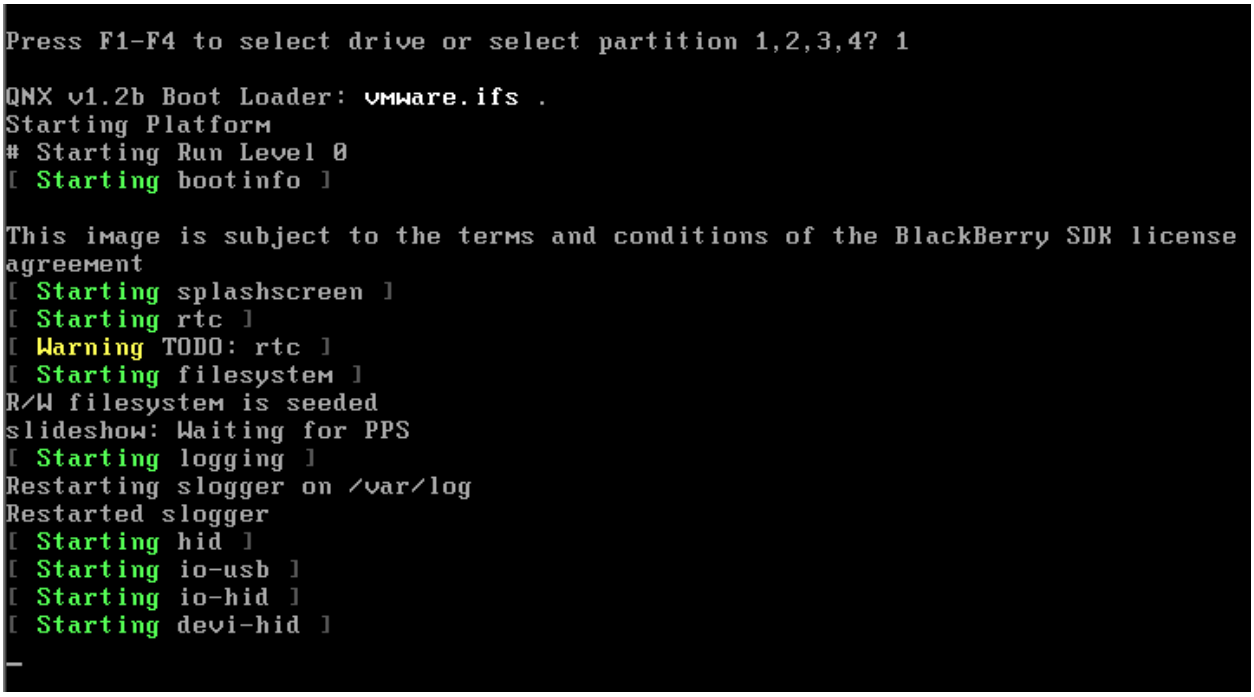


Figure 2: PlayBook simulator boot sequence

The first release of the PlayBook tablet does not include any USB host support – it just acts as a USB device like a BlackBerry smartphone. Therefore, unlike a PC, a USB keyboard could not be attached in an attempt to manipulate the start-up sequence. It is rumoured[7] that USB host support will be added in subsequent releases of the PlayBook.

## 3.1.4. Networking and services

### *3.1.4.1 HTTP services (TCP ports 80 and 443)*

**HTTP service**

QNX runs a version of NetBSD's bozotic HTTP server[8]. This is a robust HTTP daemon implementation that supports CGI 1.1 and several versions of HTTP. A code review of the server didn't reveal any significant issues. However, contrary to security good practice, the service was run as *root* (from */etc/inetd.conf*):

```
#http server is required in production to do backups and development installs
http    stream  tcp     nowait  root    /usr/sbin/bozohttpd httpd -c /opt/www/cgi /opt/ww/root
https   stream  tcp     nowait:120      root    /usr/sbin/bozohttpd httpds -c /opt/www/scgi -Z
/etc/www/cacert.pem /etc/www/privkey.pem /opt/www/root
```

The only other issue uncovered was an unauthenticated remote memory exhaustion bug in the HTTP header parsing code of the server that could potentially lead to Denial of Service. However, the impact of this issue is significantly reduced by the existence of a software timer that would trigger (and halt processing) if a request takes too long to reply. Of course, depending on the memory usage of the device and the number of applications open, during the attack it may be possible to cause a memory exhaustion condition before the timer triggers. The bozotic HTTP server team were contacted and a fix will be included in the next release.

**CGI scripts**

In version 0.9.4 of the simulator, the web server running on TCP port 443 was used by the SDK tools to deploy applications through CGI scripts. The scripts were located in */opt/www/scgi*:

- *appInstaller.cgi*
- *backup.cgi*
- *dynamicProperties.cgi*
- *login.cgi*
- *reset.cgi*
- *update.cgi*
- *wipe.cgi*

However, in version 1.0.1 of the simulator and in the device, access to the CGI directories was restricted.

Fuzz testing against the CGIs (ELF 32-bit binaries) was performed, but did not reveal any security issues. In order to use any of the CGIs a valid session had to be created through *login.cgi* and it was noted that the very first time the simulator was run after installation it was possible to bypass the authentication by directly browsing to

```
https://<ip>/cgi-bin/login.cgi?request_version=1
```

Response:

```
<RimTabletResponse>
<Auth>
  <Status>Success</Status>
  <Smb><User>dtm</User><Pwd>dtm</Pwd></Smb>
</Auth>
</RimTabletResponse>
```

The HTTP cookie automatically set by this script could subsequently be used to query the other CGIs installed.

**HTTP fuzzing**

Extensive fuzzing using a combination of freely available and in-house tools resulted in both the HTTP and HTTPS services crashing from an unauthenticated perspective. However it has not yet been possible to identify a single test case that resulted in the crash and it seemed that the specific sequence of successive requests sent by the fuzzer was critical in triggering the bugs. Also, it could not be determined whether the crashes were exploitable or not, as only limited debugging facilities were available during the research.

### *3.1.4.2 The QConnDoor protocol (TCP port 4455)*

Version 1 of *QConnDoor* protocol was analysed in the context of the 0.9.4 simulator. There is only limited information available regarding the internals of the *QConn family of* protocols. It is used while developing QNX applications to perform debugging operations and connect to the target system with the QNX development IDE. The original plan was apparently to use *QConn* for remote QNX upgrades[9].

After successfully authenticating over TCP port 4455, two new services were launched on the device, an SSH daemon on TCP port 22 and another *QConn* server on TCP port 8000. Disassembling the Java bytecode included in the SDK yielded some interesting results.

*QConn* version 1 used a custom NTLM implementation for authentication. This was discovered by following the execution flow of the following command:

```
java -Xmx512M -jar Connect.jar -targetHost <ip> -devicePassword <pwd>
```

There were a number of hard-coded NTLM protocol parameters:

- Hard-coded workstation name "workstationName": *com.qnx.tools.bbt.qconndoor.internal.rtas.RTASConnection#288*
- hard-coded NONCE: *com.qnx.tools.bbt.qconndoor.internal.ntlm.NTLMType3Message#19*
- Client-challenge: *com.qnx.tools.bbt.qconndoor.internal.ntlm.NTLMType3Message#71 (NTLMType3Message.NONCE)*
- SESSION KEY: *com.qnx.tools.bbt.qconndoor.internal.ntlm.NTLMType3Message#22*

The use of hardcoded elements such as those listed above, is not in line with security good practice, however, the real security impact of their presence in this instance is not yet clear.

### 3.1.5. PPS / QDB

The QNX Persistent Publish/Subscribe (PPS) service is a small, extensible publish/subscribe service that offers persistence across reboots. It is designed to provide a simple and easy to use solution for both publish/subscribe and persistence in embedded systems, answering a need for building loosely connected systems using asynchronous publications and notifications.

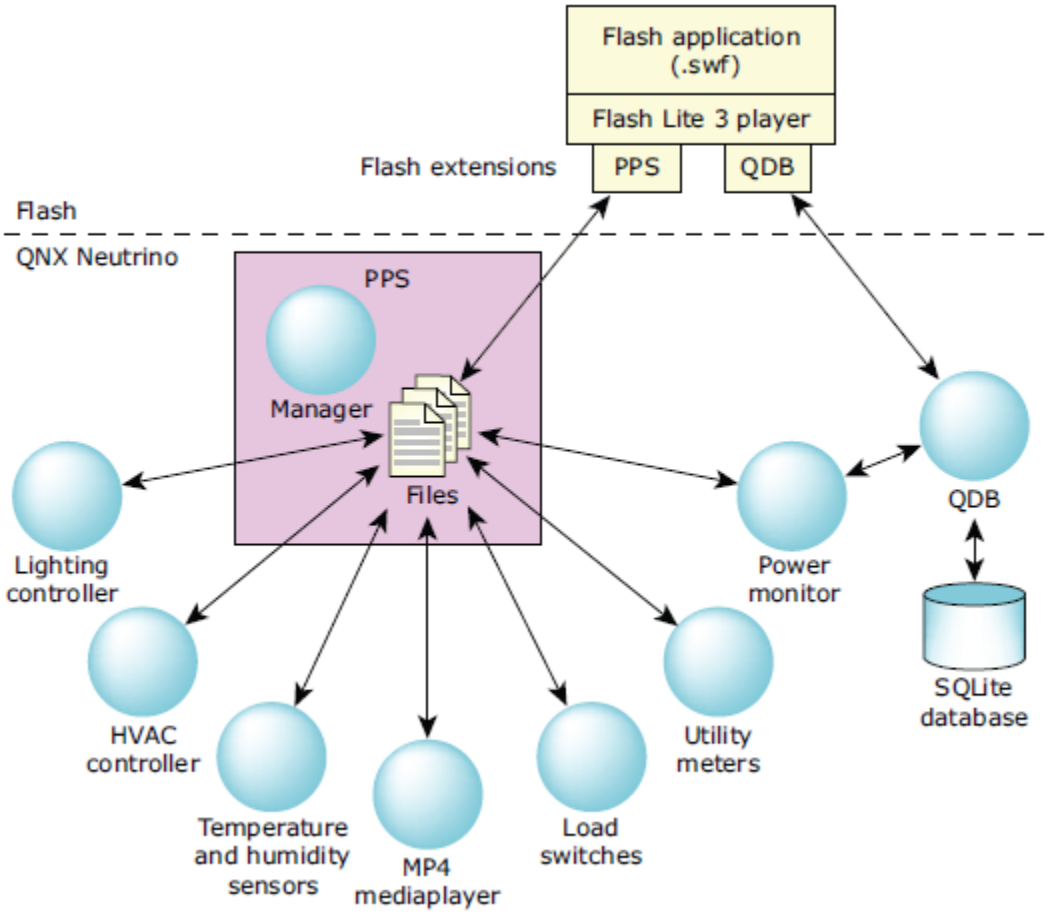


Figure 3: QNX PPS overview (source[10])

There are producers and consumers of PPS objects and in theory any process in the system can be a producer, a consumer or both. QDB is a layer on top of SQLite3 that applications can use to manage local relational databases. Although PPS components can be found in the device's file system, the PPS/QDB layer is not being made accessible to PlayBook developers through the Adobe Air or WebWorks SDKs. It is unclear whether the QDB layer is deployed in the device.

PPS is implemented at the file system layer under */pps* and PPS objects can be accessed through meta-files, such as:

```
$ cat /pps/services/.all
[n]@deviceproperties
device_os:::BlackBerry Tablet OS
hardwareid::0x00000000
scmbundle::1.0.1.1630
vendorid::?
@timezoneservice
@alarmservice
@timerservice
$
```

The list of these meta-files (named *special objects* in the documentation) includes:

| | |
|---|---|
| **.all** | Open to receive notification of changes to any object in this directory. |
| **.notify** | Open a notification file descriptor in the PPS file system root. |
| **.server** | When opened by client, a unique "instance" or "channel" of the object is created, which only that client can see. |

Writing to and reading from PPS objects could potentially be achieved by using standard file manipulation tools such as *cat* and *echo*. However, PPS objects were protected by the same restrictive ACLs as the rest of the files and directories in the file system. It was not possible to write to or read from potentially sensitive PPS objects such as those residing in */pps/services* or */pps/system*.

At the time of writing third-party applications are not able to use PPS directly in the production release of the PlayBook. However, as mentioned above, there is a chance to interact with the PPS system through the standard file system APIs.

### 3.1.6. Firmware updates and pdebug

The traditional way of upgrading the firmware of a QNX-based embedded device would be through the native QNX *pdebug*[11] protocol. However, RIM introduced a *System Upgrade* panel in the *Settings* menu that made use of several BlackBerry hosted services and the device GUI to search for software updates.

With the Flash runtime installed, there are likely to be frequent updates. As discussed in section 3.1.2, the upgrade service user is considered a high-privilege account and as a result any weaknesses in the software upgrade process could have a serious impact on the platform's security although no vulnerabilities have yet been discovered.

### 3.1.7. Universal Serial Bus (USB)

As mentioned in section 3.1.3, USB host functionality is not available in the current release of the PlayBook, which significantly reduces the USB attack surface of the device. The security of the mass-storage class driver that enables the device to function as a USB storage device was assessed using an in-house USB fuzzing tool to exercise all the different commands that can be sent to a device in a USB SETUP packet, which contains the following "interesting" fields:

- *bmRequestType*
- *bRequest*
- *wvalue*

No vulnerabilities were identified using this approach, however if future versions of the PlayBook implement USB host functionality this will significantly increase the size of codebase accessible via the USB port and hence result in a higher risk of USB-based vulnerabilities being identified and potentially exploited.

### 3.1.8. High-Definition Multimedia Interface (HDMI)

The PlayBook is equipped with a MicroHDMI socket, which enables the display on the device to be mirrored on an external HDMI-compatible display.

When an external display is connected to an HDMI port the *Hot Plug Detect* (pin 19) line goes high (+5V) and indicates to the display that it should send its capabilities to the host device so that display driver software can be appropriately configured. This data is stored in non-volatile memory within the display device in an EDID[12] data structure and comprises information such as timing data and vendor strings. A basic EDID structure is only 128 bytes in length, but there are a number of EDID extension structures[13] that can also be implemented, containing further data, which would be useful to a host's display driver.

The EDID data is sent to the host using the E-DDC[14] protocol over the $I^2C$[15] interface on the HDMI *SCL* and *SDA* (pins 15 and 16 respectively) lines where it is processed by the display driver.

Using an in-house designed, microcontroller-based HDMI EDID fuzzer, the security of the display driver software on the PlayBook was assessed and one specific test case resulted in the device no longer responding when an external display was connected, indicating a potential security vulnerability. Due to the lack of debugging capabilities the full impact of this issue is not currently known, however RIM are investigating to establish if this is indeed a security issue.

## 3.2. Application security

### 3.2.1. Application development

Applications for the PlayBook can be developed using any of the following approaches:

- Adobe AIR
- HTML5, JavaScript, and CSS using BlackBerry WebWorks SDK
- Native C/C++ (not currently available)
- Java

The security of each of these specific approaches to application development on the PlayBook will not be covered in depth in this paper; instead we will investigate some of the features that apply to PlayBook applications in general.

### 3.2.2. Inter-application communication

PlayBook applications do not appear to be allowed to communicate with each other. Applications only have direct file system access to their corresponding sandboxed folder in:

```
/apps/<appName><randHash>.<appId>/
```

Applications are only expected to interface with themselves and not directly with others. If an application needs to use device-related functions, it should do so through the BlackBerry API layer and for the PlayBook, the API is restricted to:

| API | Description |
|---|---|
| **Application** | The Application object provides functions and properties for the currently running application. |
| **Application Events** | The Application Event object allows you to access events triggered by the application (i.e. onBackground, onForeground, onSwipeDown, onSwipeStart). |
| **Invoke** | The Invoke object contains methods that interact with other applications on a BlackBerry PlayBook |
| **System** | The System object allows you to get access to system level functions and attributes of the BlackBerry PlayBook. |
| **System Events** | The System Event object allows you to access events triggered by the system. |
| **User Interface** | The Dialog object contains functions for manipulating system dialog boxes. |
| **Utilities** | The Utils object provides useful utility functions and properties. |

Table 1: Adobe Air API restrictions for the PlayBook

The *Invoke* API can only be used to invoke standard applications[16] like the camera, maps or media player, as shown below:

```
const Number  APP_CAMERA  = 4
const Number  APP_MAPS  = 5
const Number  APP_BROWSER  = 11
const Number  APP_MUSIC  = 13
const Number  APP_PHOTOS  = 14
const Number  APP_VIDEOS  = 15
const Number  APP_APPWORLD  = 16
static  void   invoke ( appType : Number ,  [args : Object ] )
```

In order for applications to be able to use the different modules of the API, they have to declare the requirements inside *widget.xml* (the application descriptor). For example, to use the *Application* API:

```
<?xml version="1.0" encoding="UTF-8"?>
<widget xmlns="http://www.w3.org/ns/widgets" version="1.0.0.0">
  <name>helloWorld</name>
  <icon src="icon.png"/>
  <content src="index.html"/>
  <feature id="blackberry.app" required="true" version="1.0.0.0"/>
</widget>
```

No mechanisms or features were identified during our research that could be used to circumvent the restrictions around inter-process communications.

### 3.2.3. Signature verifiers

The PlayBook simulator can run in “normal mode” or “development mode”. In “normal mode” applications have to be signed. A certificate can be requested through RIM’s site[17] and the SDK can be used to sign and deploy applications. However, in “development mode”, this is not enforced.

Clearly, the enforcement of signed code on the PlayBook is of paramount importance from a security standpoint. The enforcement of signed code imposes rigorous controls upon which applications can actually be deployed on the PlayBook and therefore, significantly reduces the likelihood of malware being able to execute on the device.

### 3.2.4. Payment Service SDK

An area of particular interest is the Payment Service layer exposed to all PlayBooks. During our review, no information was available regarding the integration of PlayBook applications with the Payment Service SDK (the upgrade to the WebWorks SDK released in June 23, 2011 introduced Payment Service integration). It should be noted that the service is not specific to the PlayBook SDK and is part of the BlackBerry App World storefront integration. This is definitely an area that will be investigated further due to the possible financial impact to both device owners and platform developers.

### 3.2.5. WebKit embedded browser

The browser included with the PlayBook is a version of Apple's WebKit[18] and is identified by the following *User-Agent* string:

```
Mozilla/5.0 (PlayBook; U; RIM Tablet OS 1.0.0; en-US) AppleWebKit/534.8+
(KHTML, like Gecko) Version/0.0.1 Safari/534.8+
```

As with the Flash and Air runtimes, a thorough investigation of the WebKit browser component was not performed during this phase of research. However a number of interesting issues we identified, which appeared to affect the browser's sandboxing model.

#### *3.2.5.1 Directory listing and file contents browsing*

A vulnerability was discovered that allowed directory listings to be performed, based on the privilege level of the browser process. This allowed for a more detailed view of the file system than was intended by RIM. This vulnerability was present in version 1.0.1 of the simulator and firmware version 1.0.3 of the physical device. Figure 4 shows a screenshot of the impact of the vulnerability:

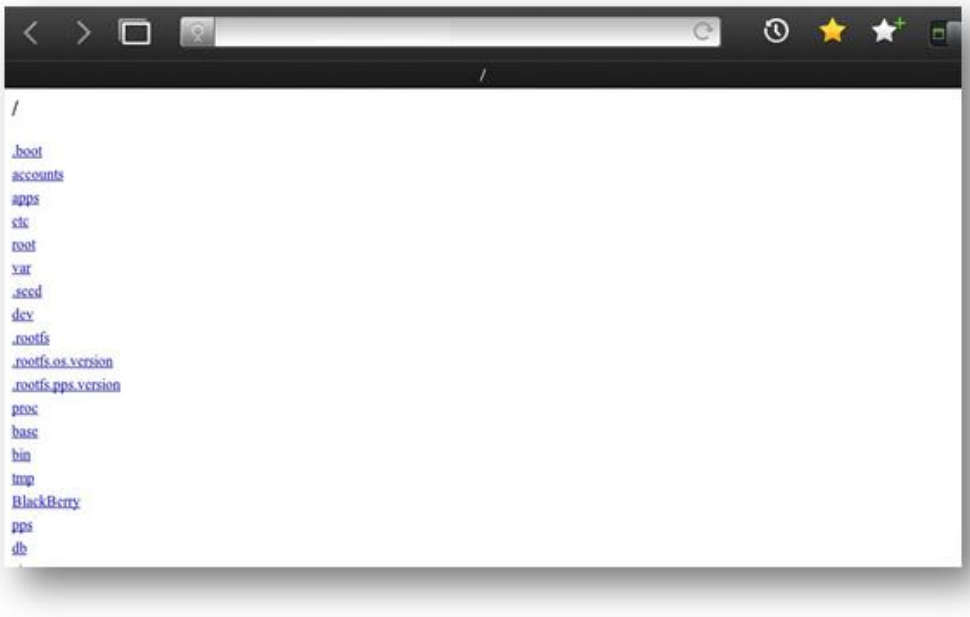


Figure 4: A screenshot of the root directory (/) of the device (vulnerability details obfuscated)

It should also be noted that this was available even when the device was not in “Development Mode”.

The vulnerability could also be used to access files (to which the process had read-access to), as can be seen in Figure 5:

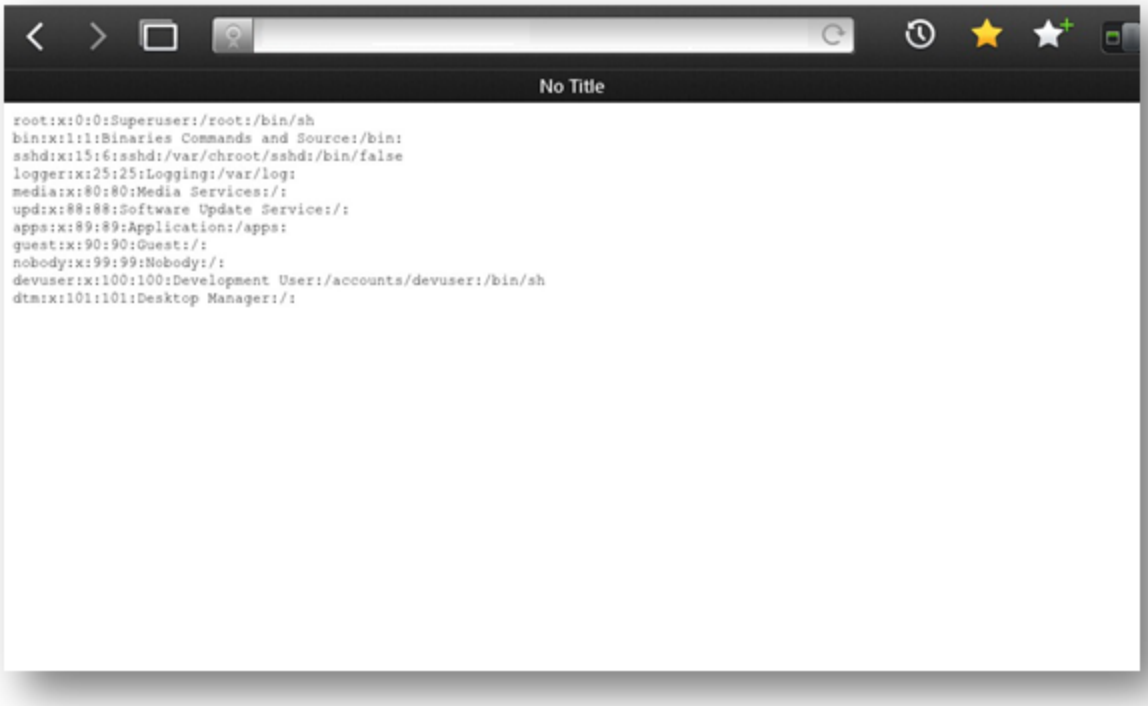


Figure 5: The browser displaying the contents of /etc/passwd (vulnerability details obfuscated)

The vulnerability has been reported to RIM and is likely to be restricted in a future update, especially when considering that the PlayBook does not ship with any file browsing applications.

### *3.2.5.2 Save to arbitrary folder*

It was also possible to save files to folders other than the default */accounts/1000/shared/downloads* folder, as shown in Figures 6 and 7 (this vulnerability has also been reported to RIM):

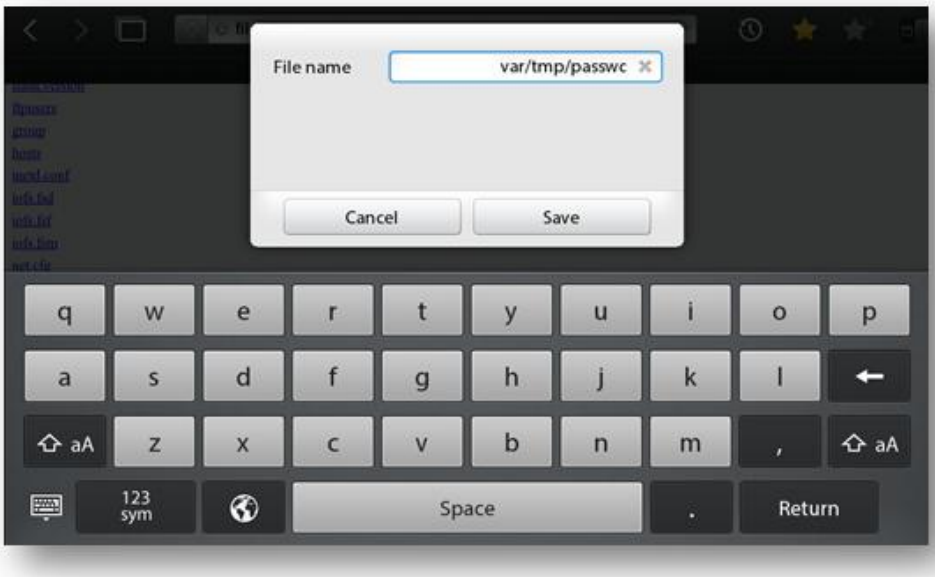


Figure 6: Saving to an arbitrary folder (vulnerability details obfuscated)

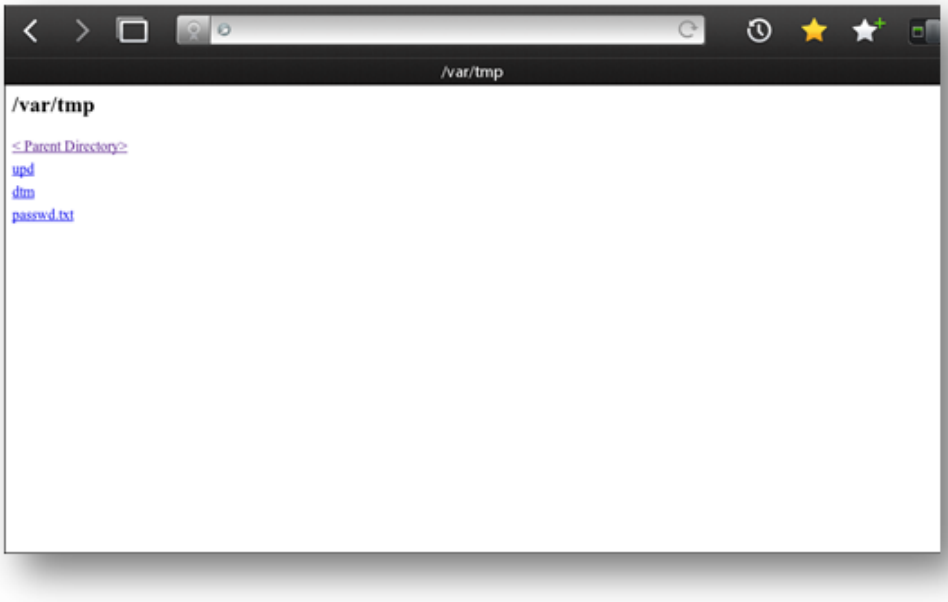


Figure 7: Directory listing of saved file location (vulnerability details obfuscated)

This vulnerability was present in version 1.0.1 of the simulator and firmware version 1.0.3 of the physical device.

## 3.3. Enterprise data security

### 3.3.1. BlackBerry Bridge: Enterprise data tethering

One of the most interesting aspects from the security point of view for organisations planning to introduce the PlayBook is enterprise data tethering. The PlayBook does not ship with an email client to connect to a BlackBerry Enterprise Server (BES). However, the "BlackBerry Bridge"[19] technology allows the device to use an enterprise-enabled smartphone to access the user's email. This is achieved by establishing a communications channel between the smartphone and the tablet over Bluetooth. Once the two devices are paired, an email client application is launched on the PlayBook enabling access to the user's inbox and contacts.

After the Bridge application is opened on the device, it presents a QR code[20] that is intended to be captured by the smartphone's camera to initiate the connection. A sample of such code is provided below:

Figure 8: BlackBerry Bridge example QR code

Once decoded, the QR code shown above translated into:

```
btps:mqYqWcLL7GdJuSLW7ny4DD|E83EB6FB032F|89150155|PlayBook-232B
```

Some of the interesting details that are available[21] about the protocol include the use of an Elliptic-Curve Diffie-Hellman (ECDH) handshake to establish a session key to protect the channel as well as a temporary encrypted file system that has been implemented to store transient data on the tablet while the channel is open. It is clear that a great deal of effort has gone into maintaining the separation between data on the PlayBook and that which is accessed from the BlackBerry Enterprise Server.

### 3.3.2. BlackBerry Balance technology

When this research was performed "Balance" technology was not yet available for the PlayBook. However RIM had expressed their intention[22] to implement support for "Balance" into the product.

As their website states, "The Balance technology enables BlackBerry smartphones to be used for business and personal purposes without compromise".

The idea is that different types of data (personal verses business) are completely segregated. This would enable IT administrators to maintain control of the business data without needing to be as concerned about any private data stored on devices.

Once the Balance technology has been implemented within the PlayBook it will certainly be investigated from a security perspective to ensure that the separation between business and personal data has been rigorously implemented.

## 4. Conclusions and further research

The PlayBook introduces a large number of new technologies, many of which have been covered during this phase of research but others require further investigation. The results of these investigations will be published in subsequent parts to this PlayBook Security white paper series.

RIM has built a robust system on top of the existing QNX microkernel. They have restricted file and user permissions at the operating system level, leaving a reduced attack surface. The fact that some of their other technologies (such as PPS) are implemented as an abstraction on top of the file system certainly contributes to ensuring that the attack surface is minimized and that the general controls implemented to protect the file system are also effective to protect these other layers.

They have also evolved some of the protocols inherited from the older versions of QNX by, for instance, adding authentication to the *QConn* protocol and restricting the use of *pdebug*. However, it is still possible that some of these legacy technologies can be exploited in ways not anticipated by RIM. A good example is the DoS issue identified in the bozotic HTTP server, but other examples may be more obscure and difficult to identify. Further research will most likely be focused on these pre-existing protocols, native components and communication channels.

If past performance is any indication of future developments, some of the more user-friendly components included in the PlayBook such as the Flash and Air runtimes or the WebKit browser are most likely to be a source of security issues and system updates for PlayBook users.

Organisations planning on introducing the PlayBook into their IT infrastructure should possibly consider waiting until further work has been published by the security community. Many new technologies are being introduced to the device post-launch (e.g. payment services and hardware device drivers e.g. USB host functionality) and a lot more are in the pipeline (e.g. “BlackBerry Balance”). It is therefore, probably worth waiting until the operating system and core technologies stabilise and the risks they introduce are better understood before embracing this powerful new tablet within the enterprise.

## 5. References and further reading


1 - http://www.qnx.com/

2 - http://osvdb.org/search?search[vuln_title]=QNX&search[text_type]=alltext

3 - http://osvdb.org/show/osvdb/71784

4 - http://support7.qnx.com/download/download/14663/qnx_microkernel_security_paper_021106.pdf

5 - http://www.qnx.com/developers/docs/6.3.0SP3/neutrino/utilities/q/qconn.html

6 - http://us.blackberry.com/developers/tablet/adobe.jsp

7 - https://twitter.com/#!/BlackBerryDev/status/50258020038488064

8 - http://www.eterna.com.au/bozohttpd/

9 - http://www.qnx.com/developers/docs/6.5.0/topic/com.qnx.doc.neutrino_user_guide/security.html

10 - http://www.qnx.com/download/download/20980/pps_book.pdf

11 - http://www.qnx.com/developers/docs/6.3.0SP3/neutrino/utilities/p/pdebug.html

12 - http://en.wikipedia.org/wiki/Extended_display_identification_data

13 - http://en.wikipedia.org/wiki/Extended_display_identification_data#EIA.2FCEA-861_extension_block

14 - http://en.wikipedia.org/wiki/E-DDC#E-DDC

15 - http://en.wikipedia.org/wiki/I2c

16 - http://www.blackberry.com/developers/docs/webworks/api/playbook/

17 - https://www.blackberry.com/SignedKeys/

18 - http://en.wikipedia.org/wiki/WebKit

19 - http://appworld.blackberry.com/webstore/content/19435

20 - http://en.wikipedia.org/wiki/QR_code

21 - http://docs.blackberry.com/playbook_security

22 - http://arstechnica.com/gadgets/news/2011/01/blackberry-balance-coming-in-two-months-to-phones-and-the-playbook.ars