University of Amsterdam

# Security Assessment of BlackBerry Messenger for Android

**Analyzing BBM communication security**

*Authors:*
Cedric Van Bockhaven
Peter van Bolhuis
Connor Dillon
Andy Ptasinski

January 5, 2014

# Contents

**Abstract**

Security and privacy of communications is a hot topic nowadays. Secret agencies listen in on private communications *en masse*. However, sometimes agencies and governments state to have trouble listening in on private communication channels. This was the case during the riots in London 2011. The police said to have problems with the messages rioters were sending each other via BlackBerry Messenger (BBM) [1]. Recently, BBM was also released for Android phones. Given the fact that the company behind Blackberry, Research In Motion (RIM), has a reputation in regard to security, the security of this app should be interesting.

# 1 Introduction

With the shocking revelations of things happening behind the closed doors of intelligence agencies, security and privacy are very common topics today. RIM, the creators of BBM, have in the past co-operated with governments, allowing access to consumer messages [2][1]. This was only done in specific cases where it was clear that crimes were being committed.

According to RIM, this was only possible as long as the end users were not using BlackBerry Enterprise Server (BES) due to secret keys that were configured by the admins of the BES were not known to the company. It has already been proven that NSA, and potentially other intelligence agencies, are capable of cracking the encryption [3] of messages, even if they were sent over a BES, however as far as the public is aware, this is not done with RIM offering a backdoor, but rather by being able to bruteforce the encryption keys or otherwise break the encryption algorithm.

Blackberry, in spite of losing the majority of its market share over the past few years is still the phone of choice for many state officials, including the US and German government organizations [4]. In an attempt to regain some of it's market share, Blackberry has released their BBM app on iOS and Android devices in late 2013.

Interestingly, the US security agencies do not consider the use of the app as a secure alternative to using the original BlackBerry phone [5]. Given the NSA's track record of pressuring companies into handing over access to encrypted content or installing backdoors [6] this raises suspicion about potential weaknesses that could hamper the security of the application.

This paper attempts to investigate the app, by looking at both the way the application traffic is sent over the network and a thorough analysis of decompiled source code, in an attempt to discover any potential backdoors or security flaws.

Our research question is: *Can we find any security flaws in the implementation of RIM's BBM on Android?*

In order to properly answer that question, we have defined a number of questions that are more specific.

- What errors exist in the implementation of TLS?
- Are there any backdoors in the BBM app?
- Is any privacy-sensitive data being leaked?

This report is structured as follows: chapter 2 gives a brief introduction on the architecture of Android. Chapter 3 and 4 describe the static and dynamic analysis we have performed the code of the app. Chapter 5 and 6 describe practical security tests and traffic analysis. In chapter 7 we analyze a security flaw in a third party statistics solution. Chapter 8 offers a conclusion and our opinion of the security of the app, and

finally in chapter 9 we propose subjects for further research.

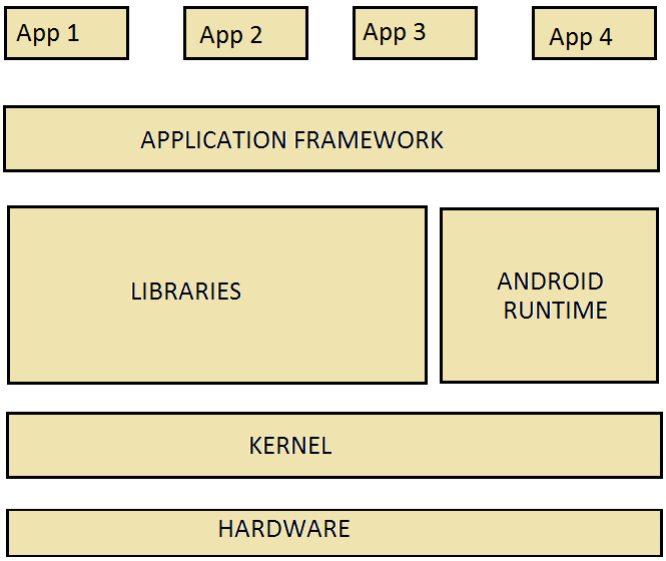
## 1.1 Related Research

Recently a lot of research has been done on the security of Android apps in general. Egele, Brumley, Fratantonio and Kruegel performed a study on over 10.000 Android apps, in which they check for mistakes in the use of cryptographic API's. They found that over 80 % off the apps make at least one mistake [7]. Fahl, Harbach and Smith did a comparable study, specifically for errors in the implementation of TLS. They picked 100 popular apps and found that 41 of these fail to correctly verify TLS certificates [8]. Clearly the state of security in Android apps has much room for improvement. Whatsapp, which is comparable in functionality to BBM, has had numerous security issues in the past [9]. BBM has only recently been released for Android, so it has not been subject to a independent security audit. Our contribution to the field of research is an in depth security analysis of the app.

## 2 Android architecture

In order to understand how we did the security analysis on BBM for Android, it is necessary to understand the basic architecture of Android, as shown in figure 1. Android apps are written in Java and are compiled for the Android runtime system. The Android runtime uses the Dalvik VM, which can be compared to other Java VM's. Apps can also use native libraries. These are written C/C++ using the Native Developer Kit (NDK). The native libraries are called through the Java Native Interface (JNI)[13]. Apps are packaged into an APK file, which contains all the binaries and files the app needs to run. The APK file can be used to install an apps on an Android system.

Figure 1: Android architecture[1]




[1] http://www.cprogramming.com/android/android_getting_started.html

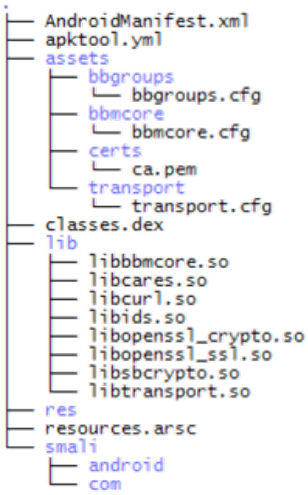
# 3 Static analysis

## 3.1 APK unpacking

We unpacked the APK file with apktool so that we could access the binaries and resources of the app. The APK contains a number of different files.

- Dalvik bytecode: classes.dex
- Native libraries: shared object files (.so)
- Certificate Authorities: ca.pem
- Other resources: images, config files, etc.

Figure 2: Unpacked APK directory tree




## 3.2 Java code

We were able to partially decompile the application. First we converted the dex file in the APK to a jar file using the tool dex2jar. Then we decompiled the jar file with Java Decompiler (jd-gui). The code was clearly obfuscated, probably by Proguard. Proguard makes the code harder to reverse engineer and tries to optimize the code. It is possible to manually deobfuscate the code by assigning logical names to classes, but this would require a lot of time and effort. We partially remapped class names on the bytecode using the tool d2j-jar-remap, in order to make better sense of the application structure.

There was a lot of Java code to spit through, however most of it was related to the GUI. We did notice that the obfuscation introduced some bugs in the code. In some cases classes are called dynamically, which will not work because the class name has been

changed by the obfuscation. These kinds of errors, along with a lot of superfluous and unused code, suggest that the application was put together in a hurry.

By decompiling, we were able to see which Java libraries the app uses. There are a number of interesting libraries, including Crittercism and MixPanelAPI. MixPanelAPI is a third party application which is used for keeping statistics. More on MixPanelAPI in chapter 7.

Fahl, Harbach and Smith developed a tool called MalloDroid, which does an automated static analysis on Android apps to find broken SSL certificate validation in the app. It does this by looking for (modified) signatures of standard Android SSL implementations [8]. We executed the tool on BBM and it was unable to find any errors.

### 3.3 Native code

The app uses a lot native libraries, which are written in C/C++ and are compiled for ARM. We were unable to properly decompile these libraries using tools available to us. However, we could disassemble them, and print all the strings, which included some interesting results:

- c1ph3rk3y40r@l@$k@

  This looks like *ciperkeyforalasaka*, which suggests it's a static cipher key coded into the app.

- Key size is %d bits - should be 256 bits

  This exact string is found in an example AES implementation using the OpenSSL EVP library.

- BBG::core::Backdoor

  We know from analyzing the code that BBG is the group chat functionality. So this suggests that there is a backdoor in the group chat.

We found out more the first two strings during the dynamic analysis, which is explained in chapter 4 and we take a look at the backdoor in 3.4

The app uses a number of interesting native libraries:

- OpenSSL

  OpenSSL 1.0.1e is packaged with the application, which is at the moment of writing the report the most recent version of the OpenSSL library. The OpenSSL library is already present on Android devices however. A reason for this could be that they chose to include their own version to avoid security issues with outdated OpenSSL versions.

- SBCrypto

SBCrypto. The Security Builder Crypto Library is available in the BlackBerry native development kit and offers cryptographic functions. We only found it being used in a specific part of the code for problem reporting. When a user choses to submit a problem report, an e-mail is generated for the BlackBerry developers with log files attached to it encrypted in AES-256-CBC, using the user's PIN as a key, or 00000000 if no PIN is available.

- BBMCore (core functionality)

  Contains the core functionality: chatting, profile management, settings, sqlite interface.

- PJSIP (in libtransport.so)

  PJSIP is an open-source multimedia communication library which supports the SIP protocol. SIP is the underlying protocol used for sending messages with BBM.

- libids.so

  The libids.so file contais functions to establish a secure channel with the BlackBerry servers and supplies authentication tokens/challenges.

- Other: SQLite, cURL, image libraries.

### 3.4 IDA Pro

IDA Pro is the Swiss army knife for reverse engineering. It allows to disassemble and decompile an extensive list of binary formats (provided you've paid for expensive licenses). Although it is also possible to dynamically debug applications (even Android apps), this functionality isn't included in the Demo version. We used IDA to research the "BBG::core::Backdoor" string we documented earlier. This string is accessed from the group chat functionality in the handleAddChatMessageRequest subroutine.

We can't make any conclusions, but want to share the following remarks:

- Right after the "BBG::core::Backdoor" string we found a switch statement which references the following strings: "None", "Download", "Upload", "GetMccUrl", "LookupMcc", "GetAuthToken", "ExpireToken", "RegisterPin", "ListFiles".
- The suspicious strings: "EncryptionOn", "EncryptionOff" are preceding the backdoor string.
- The string "eNULL:!SSLv2" is an argument that is passed to the OpenSSL library to set up the connection. It effectively implies that NULL encryption (no encryption) will be used, along with an SSL/TLS version newer than SSLv2. The default cipher string however is "ALL:!aNULL:!eNULL", meaning no NULL authentication and no NULL encryption may be used.

- The service that is being addressed here, is called Janus. In ancient Roman religion and myth, Janus is the god of beginnings and transitions, thence also of gates, doors, passages, endings and time [14].

```
.rodata:00319C43 aXOlympiaSvcBbg DCB "X-Olympia-Svc: bbg",0 ; DATA XREF: sub_18CB6C+32↑o
.rodata:00319C43                                         ; .text:off_18CCB8↑o
.rodata:00319C56 aContentTypeA_0 DCB "Content-Type: application/octet-stream",0
.rodata:00319C56                                         ; DATA XREF: sub_18CB6C+50↑o
.rodata:00319C56                                         ; .text:off_18CCC0↑o
.rodata:00319C7D aContentTypeA_1 DCB "Content-Type: application/x-www-form-urlencoded",0
.rodata:00319C7D                                         ; DATA XREF: sub_18CB6C+6A↑o
.rodata:00319C7D                                         ; .text:off_18CCC4↑o
.rodata:00319CAD aEnullSslv2     DCB "eNULL:!SSLv2",0    ; DATA XREF: sub_18CB6C+80↑o
.rodata:00319CAD                                         ; .text:off_18CCC8↑o
.rodata:00319CBA aFileid09       DCB "fileId=",0x22,"([0-9]+)",0x22,0
.rodata:00319CBA                                         ; DATA XREF: sub_18CB6C+96↑o
.rodata:00319CBA                                         ; .text:off_18CCCC↑o
.rodata:00319CCC aNameIpaddrtoco DCB "name=",0x22,"ipAddrToCountryCodeConverter",0x22," value=",
.rodata:00319CCC                                         ; DATA XREF: sub_18CB6C+C4↑o
.rodata:00319CCC                                         ; .text:off_18CCD8↑o
.rodata:00319CCC                 DCB "/\/)([\w\.\-]*(\/)?)*(\w)*\??(\&?\w+=\w+)*)",0x22,0
.rodata:00319D30 aAuthheaderValu DCB "authHeader value=",0x22,"([a-zA-Z0-9:+/]*=*)",0x22,0
.rodata:00319D30                                         ; DATA XREF: sub_18CB6C+EE↑o
.rodata:00319D30                                         ; .text:off_18CCE0↑o
.rodata:00319D57 aDownload       DCB "Download",0        ; DATA XREF: sub_18CCEC+1C↑o
.rodata:00319D60 aUpload         DCB "Upload",0          ; DATA XREF: sub_18CCEC+24↑o
.rodata:00319D67 aGetmccurl      DCB "GetMccUrl",0       ; DATA XREF: sub_18CCEC+2A↑o
.rodata:00319D71 aLookupmcc      DCB "LookupMcc",0       ; DATA XREF: sub_18CCEC+30↑o
.rodata:00319D7B aGetauthtoken   DCB "GetAuthToken",0    ; DATA XREF: sub_18CCEC+36↑o
.rodata:00319D88 aExpiretoken    DCB "ExpireToken",0     ; DATA XREF: sub_18CCEC+3C↑o
.rodata:00319D94 aRegisterpin    DCB "RegisterPin",0     ; DATA XREF: sub_18CCEC+42↑o
.rodata:00319DA0 aListfiles      DCB "ListFiles",0       ; DATA XREF: sub_18CCEC+48↑o
.rodata:00319DAA aEncryptionon   DCB "EncryptionOn",0    ; DATA XREF: .text:0018CD72↑o
.rodata:00319DAA                                         ; .text:off_18CD84↑o
.rodata:00319DB7 aEncryptionoff  DCB "EncryptionOff",0   ; DATA XREF: .text:0018CD78↑o
.rodata:00319DB7                                         ; .text:off_18CD88↑o
.rodata:00319DC5 aBbgCoreBackdoo DCB "BBG::core::Backdoor",0 ; DATA XREF: .text:0018CDC2↑o
.rodata:00319DC5                                         ; .text:off_18CDC8↑o
```

Figure 3: IDA Pro backdoor research

### 3.5 SQLite databases

Apart from a few SQLite databases that were being created by the Java code (for Google Analytics and MixPanel), we also found that others were being created by the native shared libraries:

- bbgroups.db: some tables, like Settings which contains your PIN and other personal data.
- master.db: contains your Contacts, Locations (if that's being tracked), Profile, and also your sent and received messages in unencrypted form.

The fact that all of this is unencrypted is remarakble, while there are unused SQL fields called *EncryptedEncryptionKey*, *KeyExchangeRequestPrivateKey*, *EncryptedRegistrationKey* and *KeyNegoEncryptionKey*. It is probable that future versions of BBM for Android will actually make use of these encryption keys.

# 4 Dynamic analysis

In order to comprehend the execution flow of the BBM app better, we performed a dynamic analysis. With dynamic analysis we mean the interaction with the program while it is in a running state. We ran several forms of dynamic analysis which we will document below:

## 4.1 Smali debugging

To understand how we debugged the BBM app, we have to understand what smali is. Smali/baksmali is an assembler/disassembler for the dex format used by dalvik, Android's Java VM implementation. The syntax is loosely based on Jasmin's/dedexer's syntax, and supports the full functionality of the dex format (annotations, debug info, line info, etc.) [12]. Apktool contains functionality to decompile an APK file into smali code, and repackage the APK file with debug info.

We then push the repackaged APK file to our device. Using an IDE like IntelliJ IDEA we load the java-smali hybrid source generated by apktool and attach to the BBM process using DDMS (Dalvik Debug Monitor Server) which comes with the Android SDK.

For an example of how such a file looks, we refer to page 26. The "a=1" is just bogus code needed for the code to be valid. The actual magic happens by adding debug information to the dex file which contains references to the line numbers.

## 4.2 Remote GDB

GDB (The GNU Debugger) has a debugging server for Android available (gdbserver). gdbserver can be started by supplying it a port it has to listen on, and a process id that it'll attach to.

After starting the server, we can connect to it remotely using a gdb client and add breakpoints. We tried adding breakpoints to known function names inside the native libraries. The problem however is that the code was compiled without debugging symbols and it wasn't possible to output local variables. The content of registers can still be accessed.

## 4.3 LD_PRELOAD

During debugging, we noticed a particular text file being written to the data folder, called "persistent.txt". Afterwards, in the strings of *libbbmcore.so*, we found this filename alongside the strings "c1ph3rk3y40r@l@$k@" and "Key size is %d bits - should be 256

bits". A google search quickly revealed that the latter string appears in an example implementation of AES encryption using the OpenSSL EVP library.

The OpenSSL EVP library provides a high-level interface to cryptographic functions [11], of which they used *EVP_BytesToKey* and *EVP_EncryptUpdate* which are of interest to us. *EVP_BytesToKey* takes the cipher key, a salt, and the number of rounds as an argument. *EVP_EncryptUpdate* transforms the provided plaintext into ciphertext.

In order to decrypt the file successfully using the sample AES encryption code, we would've also needed the salt, which wasn't easily recoverable from the static analysis.

We can however log the input to these functions during execution by overriding them with LD_PRELOAD. LD_PRELOAD is an environment variable which we can use to point to a shared object file that has to be preloaded by the system, overriding other library functions (eg. for patching bugs). We can conveniently (mis)use it for overriding functions in the packaged OpenSSL EVP library.

Using the Android Native Development Kit we compiled a shared object for ARM, of which the source code can be found on page 25. We then pushed this object to the /data folder on the Android device, after which we could preload it in the Dalvik VM:

```
ANDROID $ setprop wrap.com.bbm LD_PRELOAD=/data/libsslover.so
```

The above command sets the wrap property for the com.bbm application, which allows us to set the LD_PRELOAD environment variable. We then start the application from the launcher and grep the output of logcat:

```
PC $ adb logcat | grep -i --line-buffered SSLOver
```

```
D/SSLOverride(27373): Salt: {54321,12345}, Key: c1ph3rk3y40r@l@$k@, Count: 5
D/SSLOverride(27373): Plaintext: username cedric+bbm2@ce3c.be
D/SSLOverride(27373): Plaintext: PIN 7BF72F8C
D/SSLOverride(27373): Plaintext: PINTYPE spin
```

The contents that we found written to the persistent.txt file were rather disappointing: the username and PIN was being stored in this file, but these were already being stored unencrypted in the SQLite database. It is possible that persistent.txt is being used more extensively on other platforms (as the name implies, as a persistent data store).

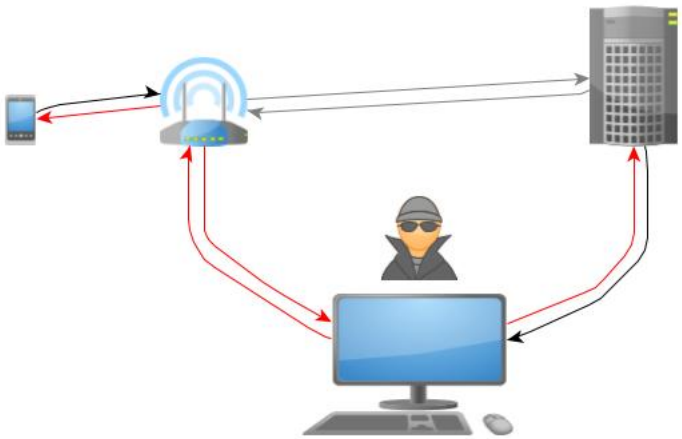
# 5 Testing TLS

To check whether the BBM app was resistant to attacks on TLS, some practical attacks were tested. In a normal scenario, TLS will protect the traffic between two endpoints, by encrypting the contents and verifying the server endpoint with a certificate. This means nobody should be able to listen in on the conversation and nobody can pretend to be the server.

## 5.1 Man in the middle

First off the environment was built to perform these attacks. Using *airbase-ng* a wireless access point was set up. The mobile phone with BBM connects to the rogue wireless network. The diagram 4 shows the flow of the traffic. Normally the client expects the traffic to flow directly to the server (the grey arrows). However, the attacker makes all traffic flow through his pc before sending it towards the server (red and black arrows). As expected, the traffic between the BBM application and the server it connected to was protected with TLS. The attacker could not see any of the contents of the stream by passively listening.

Figure 4: Man in the middle taps all traffic from client to server and back



## 5.2 TLS cipher suite modification

One attack that was performed was trying to downgrade the encryption level from the TLS session. With TLS, both parties agree on the level of encryption to use by the client sending a suite of ciphers it supports. The server will pick one and they will use this. By changing the list of ciphers, the encryption level may be degraded to a crackable level.

(Un)fortunately, the client forces the use of TLS 1.0, which makes downgrading of the cipher suite impossible by including a hash over the handshake and key-material.

### 5.3 Wrong certificates

Because the server uses a certificate to prove its identity, it is impossible to fake its identity. That is if the client checks this certificate correctly. This was tested with *mitmproxy* and *sslsplit*. These programs both use a provided certificate to generate and sign certificates for visited domains. All information in the certificate seems legit, only the signing authority isn't real. The BBM application refused to connect when the certificate chain was broken, which is expected behaviour.

### 5.4 Other known attacks

Other known attacks against TLS 1.0 include the BEAST and CRIME attack and LUCKY13. However, the BBM app includes its own SSL library (version 1.0.1e), which negates these attacks. Regular downgrade attacks to downgrade from TLS 1.2 to older versions weren't possible because the client forces the use of TLS 1.0.

### 5.5 Certificate Pinning

As a last attack vector, the attacker was assumed to be able to import a certificate on a phone. By including the certificate used for signing in the chain of trust, the phone will trust all certificates signed by this one. However, the BBM app used certificate pinning for communication with the SIP server, meaning it has its own built-in list of trusted certificates. Because the signing certificate was not in there, and could not be imported in there without modifying the APK file, the app would not accept the wrong certificates.

# 6 Traffic Analysis

This chapter will explain the traffic analysis that was done after altering the APK of the app. Altering the source was needed to circumvent the certificate pinning that was used by the app (see the chapter on TLS testing). First the traffic endpoints were determined. Thereafter the traffic sent to those endpoints was analyzed.

## 6.1 Traffic endpoints

With the *man in the middle* set up, the traffic endpoints could be seen. When the app started, first a connection was set up to *blackberryid.blackberry.com*. After a response from this one, the rest of the traffic would go to *global.uci.blackberry.com*. See figure 5 for the topology.

## 6.2 Traffic contents

Having altered the source of the app, the certificate used for signing fake certificates was now trusted by the app. This meant the contents of the traffic could now be analyzed.

### 6.2.1 blackberryid.blackberry.com

This server was used to log on to. Provided the correct login was sent, this server replied with a token. This token is used to authenticate to the *global.uci.blackberry.com* server. This server uses TLS 1.2. The authentication screen is a webpage that is loaded into a WebView panel, that uses the internal trust store of Android.

### 6.2.2 global.uci.blackberry.com

All messaging in the BlackBerry Messenger app runs through this server. Within the TLS 1.0 tunnel, cleartext messages are sent with a sender and receiver in the header. The messages are forwarded to an internal server *sip.voip.blackberry.com*, which probably handles the messages. The TLS communication to this server uses certificate pinning.

### 6.2.3 sip.voip.blackberry.com

This server receives the messages from the *global.uci.blackberry.com* server. The messages use the RFC3428 [10] format which specifies an extension to the SIP protocol for instant messaging.

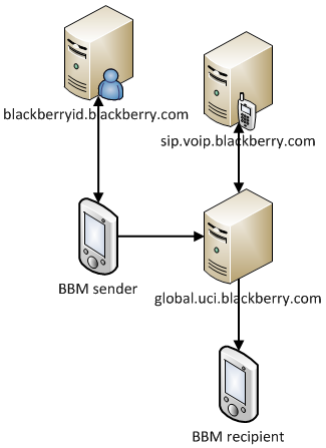



Figure 5: Network topology for BBM communications

### 6.3 Internal server names

We found references to multiple internal services:

- Alaska

  The name Alaska is being referred to in both the Java and native code. We believe that this server is the internal name for the SIP server.

- Olympia

  We found Olympia in the native code, which supposedly is the service that handles file/image uploads. This service also references an URL resource called Kronos. It is interesting to note that the phone's MCC (Mobile Country Code) and MNC (Mobile Network Code) is sent to the BlackBerry servers to locate the closest mirror.

- Janus

  We found Janus in the native code being used for the uploading of avatars. This server is particularly interesting though, as documented in section 3.4 on IDA Pro.

# 7 MixPanelAPI

As documented in the static analysis chapter, we found references to MixPanel being used. MixPanel is an analytics platform for mobile and the web. The Android app reports back to the MixPanel servers on a frequent basis.

In the documentation the following string was found:

“ The library now falls back to HTTP communication if HTTPS communication is unavailable. In particular, this may occur in early versions of Android that only support a small set of certificate authorities. ” [15]

## 7.1 Proof of concept

As a proof of concept, we blocked traffic on port 443, after which we found the library connected to port 80 instead and sent the statistics unencrypted.

An example of the traffic we found is the following:

```
{
  "$app_version":"1.0.2.83",
  "$brand":"JIAYU",
  "$carrier":"NL KPN",
  "$has_nfc": false,
  "$has_telephone": true,
  "$lib_version": "3.3.0",
  "$manufacturer": "JYT",
  "$model": "JY-G5",
  "$os": "Android",
  "$os_version": "4.2.1",
  "$screen_dpi": 320,
  "$screen_height": 1280,
  "$screen_width": 720,
  "Swifi": true,
  "File Transfers - Average Size": 0,
  "File Transfers - Num of Transfers": 0,
  "File Transfers - Total Size": 0,
  "New 1:1 Chats": 2,
  "New Group Chats": 0,
  "New Multi Chats - Average Participants": 2,
  "New Multi Chats - Count of Participants": 2,
  "Number of active chats": 1,
  "Number of active group chats": 0,
  "Number of contacts": 2,
  "Number of groups": 0,
  "Number of new group invites": 0,
  "Number of new group updates": 0,
  "Total Time in Application": 58,
  "distinct_id": "6141e8ff-a85c-412",
  "mp_lib": "android",
  "time": 1386357882,
  "token": "391da7cb4ad8ddedd89a"
}
```

Certain information like the *distinct_id* can be used to identify a specific user. In regards to the BlackBerry riots, the count of participants in group chats could be of interest for governments trying to track down riot groups using BBM to communicate.

### 7.2 Proposed fix

The fallback behavior can be disabled, but is the default in current MixPanel versions. According to the above statement, the MixPanel developers enabled HTTP fallback to support older Android versions that don't have the necessary certificate authorities in their internal trust store. We believe that a better solution would be to use certificate pinning. This way, the necessary CA's are packaged with the library, compatibility problems are overcome, all while still maintaining a secure communication channel. The fallback behavior should at least be disabled by default.

# 8 Conclusion

The research started with the following questions:

- What errors exist in the implementation of TLS?
- Are there any backdoors in the BBM app?
- Is any privacy-sensitive data being leaked?

After our research we can answer these questions. To start with the first one: trying a lot of possible attacks and checking the code, no errors were found in the implementation of TLS in the BlackBerry Messenger app. BBM uses certificate pinning to protect against certificates wrongfully being trusted on the phone. In the aspect of implementing TLS, BBM is secure.

Regarding the possible backdoors in the application: not all code could be decompiled in time, and it is therefore impossible to say there are no backdoors. A specific string indicated a possibility to something like one: *BBG::core::Backdoor*. This can be future research.

About the possible leaking of sensitive information: BBM uses MixPanelAPI as an analytics platform. Information is sent to the MixPanel servers over a TLS connection. However, if no TLS is available, the library falls back to HTTP. Blocking the TLS traffic will result in the information being sent insecurely. By changing the default behaviour in MixPanel and by using certificate pinning, this can be improved.

# 9 Further Research

There are a number of things that are still interesting to research. It would be nice to do a proper comparison between the current state of security in BBM and other mobile chat clients like WhatsApp.

As mentioned in chapter 3, we only had an IDA Pro demo license. With a full license it should be possible to do a proper decompilation of the native libraries. This could make it possible to do more research on the possibility of a backdoor in the app.

It is also possible do manually deobfuscate the decompiled Java code. This would allow for a much better static analysis of the code.

A big feature of BBM is the possibility to use BES to define a company wide encryption key for encryption messages. It would be interesting to research how this is implemented in BBM for Android.

## References


[1] The Economist, *The BlackBerry riots*, 2011
http://www.economist.com/node/21525976

[2] CIO, *RIM Allows India Access to Consumer BlackBerry Messaging*, 2011
http://www.cio.com/article/654438/RIM_Allows_India_Access_to_Consumer_BlackBerry_Messaging

[3] InfoWorld: Security Central, *Report: UK and US spies have cracked Blackberry's BES encryption*, 2013
http://www.infoworld.com/d/security/report-uk-and-us-spies-have-cracked-blackberrys-bes-encryption-226383?page=0,0

[4] The Washington Post, *BlackBerry smartphones still reign in Washington, the slow-poke capital*, 2013
http://articles.washingtonpost.com/2013-09-25/business/42388797_1_federal-government-smartphones-blackberrys

[5] Reuters, *Obama says he's not allowed iPhone for 'security reasons'*, 2013
http://www.reuters.com/article/2013/12/05/us-usa-obama-apple-idUSBRE9B402Y20131205

[6] The Guardian, *Microsoft handed the NSA access to encrypted messages*, 2013
http://www.theguardian.com/world/2013/jul/11/microsoft-nsa-collaboration-user-data

[7] M. Egele, D. Brumley, Y. Fratantonio, C. Kruegel, *An Empirical Study of Cryptographic Misuse in Android Applications*, 2013
http://www.cs.ucsb.edu/~chris/research/doc/ccs13_cryptolint.pdf

[8] S. Fahl, M. Harbach, M. Smith, *Hunting Down Broken SSL in Android Apps* , 2013
https://owasp.com/images/7/77/Hunting_Down_Broken_SSL_in_Android_Apps_-_Sascha_Fahl%2BMarian_Harbach%2BMathew_Smith.pdf

[9] T. Alkemade, *Piercing Through WhatsApp's Encryption*, 2013
https://blog.thijsalkema.de/blog/2013/10/08/piercing-through-whatsapp-s-encryption/

[10] B. Campbell et. al., *Session Initiation Protocol (SIP) Extension for Instant Messaging*, 2002
http://www.ietf.org/rfc/rfc3428.txt

[11] OpenSSL, *Documents, evp(3)*
https://www.openssl.org/docs/crypto/evp.html

[12] Jesus Freke, *Smali - An assembler/disassembler for Android's dex format*,
https://code.google.com/p/smali/

[13] Android, *Android NDK*,
http://developer.android.com/tools/sdk/ndk/index.html

[14] Wikipedia, *Janus*,
http://en.wikipedia.org/wiki/Janus

[15] GitHub, *mixpanel-android readme file*,
https://github.com/mixpanel/mixpanel-android/blob/master/readme.txt#L121

## Appendix: OpenSSL EVP sample AES implementation

```
/**
  AES encryption/decryption demo program using OpenSSL EVP apis
  gcc -Wall openssl_aes.c -lcrypto

  this is public domain code.
  Saju Pillai (saju.pillai@gmail.com)
**/
...
/**
 * Create an 256 bit key and IV using the supplied key_data. salt can be added for taste.
 * Fills in the encryption and decryption ctx objects and returns 0 on success
 **/
int aes_init(unsigned char *key_data, int key_data_len, unsigned char *salt, EVP_CIPHER_CTX *e_ctx,
             EVP_CIPHER_CTX *d_ctx)
{
  int i, nrounds = 5;
  unsigned char key[32], iv[32];
  /*
   * Gen key & IV for AES 256 CBC mode. A SHA1 digest is used to hash the supplied key material.
   * nrounds is the number of times the we hash the material. More rounds are more secure but
   * slower.
   */
  i = EVP_BytesToKey(EVP_aes_256_cbc(), EVP_sha1(), salt, key_data, key_data_len, nrounds, key, iv);
  if (i != 32) {
    printf("Key size is %d bits - should be 256 bits\n", i);
    return -1;
  }

  ...
}

/*
 * Encrypt *len bytes of data
 * All data going in & out is considered binary (unsigned char[])
 */
unsigned char *aes_encrypt(EVP_CIPHER_CTX *e, unsigned char *plaintext, int *len)
{
  ...
  EVP_EncryptUpdate(e, ciphertext, &c_len, plaintext, *len);
  ...
}

int main(int argc, char **argv)
{
  ...
  /* 8 bytes to salt the key_data during key generation. This is an example of
     compiled in salt. We just read the bit pattern created by these two 4 byte
     integers on the stack as 64 bits of contigous salt material -
     ofcourse this only works if sizeof(int) >= 4 */
  unsigned int salt[] = {12345, 54321};
  unsigned char *key_data;
  int key_data_len, i;
  char *input[] = {"a", "abcd", "this is a test", "this is a bigger test",
                   "\nWho are you ?\nI am the 'Doctor'.\n'Doctor' who ?\nPrecisely!",
                   NULL};
  ...
}
```

## Appendix: LD_PRELOAD shared object for overriding OpenSSL functions

```c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <dlfcn.h>
#include <openssl/evp.h>
#include <android/log.h>

int EVP_BytesToKey(const EVP_CIPHER *type,const EVP_MD *md,
    const unsigned char *salt, const unsigned char *data, int datal,
    int count, unsigned char *key,unsigned char *iv)
{
    int (*new_evp_bytestokey)(const EVP_CIPHER *type,const EVP_MD *md,
        const unsigned char *salt, const unsigned char *data, int datal,
        int count, unsigned char *key,unsigned char *iv);

    __android_log_print(ANDROID_LOG_DEBUG, "SSLOverride",
        "Salt: {%d,%d}, Key: %s, Count: %d\n",
        *(unsigned int*)salt, *(unsigned int*)(salt+4), data, count);

    new_evp_bytestokey = dlsym(RTLD_NEXT, "EVP_BytesToKey");
    return new_evp_bytestokey(type, md, salt, data, datal, count, key, iv);
}

int EVP_EncryptUpdate(EVP_CIPHER_CTX *ctx, unsigned char *out,
    int *outl, const unsigned char *in, int inl)
{
    int (*new_evp_encryptupdate)(EVP_CIPHER_CTX *ctx, unsigned char *out,
        int *outl, const unsigned char *in, int inl);

    __android_log_print(ANDROID_LOG_DEBUG, "SSLOverride",
        "Plaintext: %s \n", in);

    new_evp_encryptupdate = dlsym(RTLD_NEXT, "EVP_EncryptUpdate");
    return new_evp_encryptupdate(ctx, out, outl, in, inl);
}
```

**Appendix: Sample smali-java hybrid class**

```
package com.bbm; class a { void a() { int a;
a=0;// .class Lcom/bbm/a;
a=0;// .super Ljava/lang/Thread;
a=0;//
a=0;//
a=0;// # instance fields
a=0;// .field final synthetic a:Lcom/bbm/Alaska;
a=0;//
a=0;//
a=0;// # direct methods
a=0;// .method constructor <init>(Lcom/bbm/Alaska;)V
a=0;//     .locals 0
a=0;//
a=0;//     iput-object p1, p0, Lcom/bbm/a;->a:Lcom/bbm/Alaska;
a=0;//
a=0;//     invoke-direct {p0}, Ljava/lang/Thread;-><init>()V
a=0;//
a=0;//     #p0=(Reference,Lcom/bbm/a;);
a=0;//     return-void
a=0;// .end method
} }
```