Defeating Spyware & Forensic Acquisition on BlackBerry Handhelds

Sheran A. Gunasekera

ZenConsult Labs
http://chirashi.zensay.com
E-mail: sheran@zenconsult.net

ABSTRACT

Malware has proven to be the scourge of the converging desktop and mobile ecosystems. With more powerful mobile devices, comes the proportionate growth of a myriad of malware. Traditionally, malware on the desktop platform has been thought to be the work of individuals out to make their mark on society and find their 15 minutes of notoriety. This is no longer the case. Malware is now used for anything from organized crime[1] to spying on cheating spouses. Commercial entities have made their business models around the sale of 'commercial spyware' designed to put the power of spying into the hands of the layperson [2,3]. Other companies maintain a low-profile and sell similar products only to law enforcement agencies. [4,5]

The foundation on which all of these malware products is built is the exfiltration of relevant information. Whether it is email, call records or GPS co-ordinates, data is collected and forwarded either at timed intervals or in real-time to a collection end-point. This paper discusses some of the key points of how to render this collected information useless. To counter malware, the traditional approach has been to develop detection and removal tools. This has limits in that only known malware can be detected. An alternate approach is to feed malware with an abundance of useless information or a significantly reduced Signal-to-noise ratio. This paper will discuss and provide examples of how such a technique can be implemented on a BlackBerry handheld.

INTRODUCTION

Malware on a BlackBerry handheld operates in a predictable manner. Third party applications will run in a sandboxed environment with many granular controls and permissions that limit its functional capabilities. So why then does malware thrive on the platform? Because a user is still required to administer these controls or permissions. Invariably, a user's 'click-through-to-reach-your-goal' attitude will have pop-up windows with requests for permission to be granted. Blaming a user is not going to solve the problem. Educating the user is. Providing a user with free tools to detect spyware is another way to solve the problem. However, letting a user continue to use his device with little to no intervention while at the same time confounding the people spying on him seems by far a more elegant solution.

To properly understand how to achieve this, a good understanding of how malware works is necessary. Next, knowledge of data repositories within the BlackBerry handheld need to be identified and altered in a manner that will not inconvenience the end-user, but will significantly reduce the Signal-to-noise ratio for the data-collector. These are all topics that will be covered in this paper. As a bonus, techniques on hampering a forensic investigation of the device will be included.

ACTORS

Lets first identify the actors in this ecosystem. There is the end-user who is the owner of the device and the information contained therein. Then there is the data-collector or spy who will be the person or entity collecting and sifting through the end-user's information without his consent.

HOW MALWARE WORKS

For the purposes of this paper, the term malware is used to mean malicious applications installed on an end-user's device to collect and extract private information without consent.

Malware usually targets several information repositories on the BlackBerry handheld and will go after specific types of information. These repositories are listed in Table 1 below:

Repository	Type of Information
Email Folders	Incoming & outgoing email messages
SMS/MMS Folders**	Incoming & outgoing Short Messages and Multimedia Messages
Call History Log	Incoming & outgoing phone call numbers and call durations
Phone Book	Contact information like name, address, phone number
BlackBerry Messenger Chats	Incoming & outgoing instant messages exchanged between users or groups
GPS Module**	Physical location of the handset and end-user

Table 1. Information Repositories

*** It is not entirely possible to defeat malware that hooks these repositories.*

The BlackBerry API consists of a set of Listeners that aid the developer in writing event-driven applications. Listeners alert the program when certain tasks take place. In the repositories listed in Table 1, every repository can have a Listener added to run some code when an event is received. Due to a bug, either deliberate or inadvertent, the BlackBerry Messenger Listener does not work to its fullest capability and thus alternate methods are generally used by the malware vendors.

MALWARE CRIPPLING TECHNIQUES

The premise of this paper is to discuss new or alternate methods for defeating malware. Rather than attempting to find and remove the malware, these techniques attack the source of information that the malware is attempting to capture. These techniques will poison the information stores in a manner that only affects malware (Fig. 1). This will serve to reduce the quality of information presented to a data-collector. The cost to the end-user is minimized. Some of the techniques described may elicit unexpected behavior from malware which

further serves to alert the end-user that, 'something is not right' with the handheld.

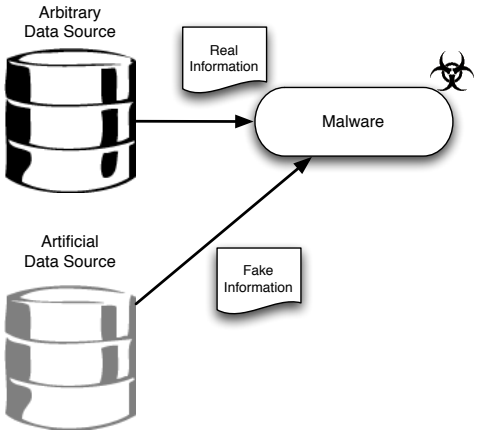

Fig. 1 - Poisoning the data

POEPFLOOD

POEP, which stands for Phony Object Escalation Process, is a technique that can be used to introduce an overwhelming amount of false data to a watched resource. For example, if malware is watching a folder and collecting objects that are introduced to this folder, then conducting a POEPFlood attack will initiate the process of adding collections of fake objects to this watched folder in increasing order of magnitude. This results in the malware collecting not only legitimate objects, but fake ones as well. This large amount of false positives provides a diluting effect on legitimate information and if done correctly, will make the data-collector's task very difficult. An attacker that hopes to spy on a device like this will have to sift through hundreds or thousands of pieces of inaccurate, irrelevant data. This achieves the goal of reducing the Signal-to-noise ratio.

PWNGOAL

PWNGoal is a flooding attack where certain sacrifices will need to be made such as financial expenditure. The idea of the PWNGoal attack is to create an Artificial Data Source (Fig. 1) using the help of a third party. This attack will be most useful when it is not feasible to create an Artificial Data Store on the BlackBerry device. For example, consider the case where SMS or MMS messages need to be spoofed. The two main ways to achieve this are 1) Write a routine where the handheld sends messages to itself 2) Use a third party messaging gateway to generate these messages.

The accompanying source code covers scenario 1). Scenario 2) is currently beyond the scope of this paper. Either way, an additional cost is incurred in either sending SMS messages through the carrier or cost of messages on the messaging gateway.

DDTS

The DDTS technique, or Don't Drop The Soap technique, is better suited to defeating a forensic analysis of the BlackBerry handheld. This technique can be used to disallow or prevent access to the USB Port on the BlackBerry handheld. Generally, a forensic analysis will aim to gather information from the BlackBerry through the USB Port. DDTS will watch for any connections to the USB Port and take appropriate action when one is detected. This can be achieved by setting an *IOPortListener* or *USBPortListener* and triggering an action when a *connectionRequested()*. Alternatively, setting a *SystemListener2* and triggering an action on *usbConnectionStateChange()* is also a possibility.

The DDTS technique by itself often serves no purpose. Instead, DDTS should be coupled with one of the other types of attacks so that appropriate action can be taken when a USB Connection is requested. Typically, a complete DDTS attack will employ a POEPFlood, PWNGoal or FMLog attack to destroy or overwrite the contents of a BlackBerry handheld.

FMLog

The FMLog attack or Flush My Log attack is another technique used to defeat forensic analysis of BlackBerry devices. The BlackBerry handheld has a built-in event logging mechanism that system and third-party applications can write to. This event log can be viewed on the BlackBerry handheld by invoking the key combination <ALT> + LGLG (Holding down the Alt key and type LGLG). The entire size of this log is 16Kb [ref to log size] and any third party application can write to this log using the *EventLogger* class on the device.

To conduct this attack, a timed process that writes more than 16Kb of data to the log is all that is required. Further, this attack can be enhanced by using the DDTS attack to only flush the log when a USB cable is connected to the handheld. Figure 2 and 3 depicts a 'before' and 'after' look at the FMLog attack.

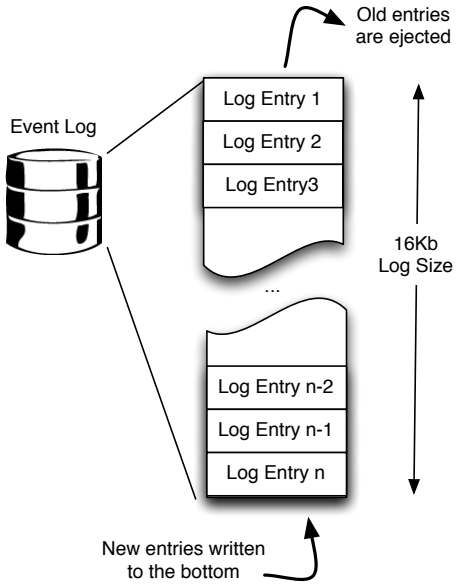

Fig. 2 - Event Log operation before FMLog attack

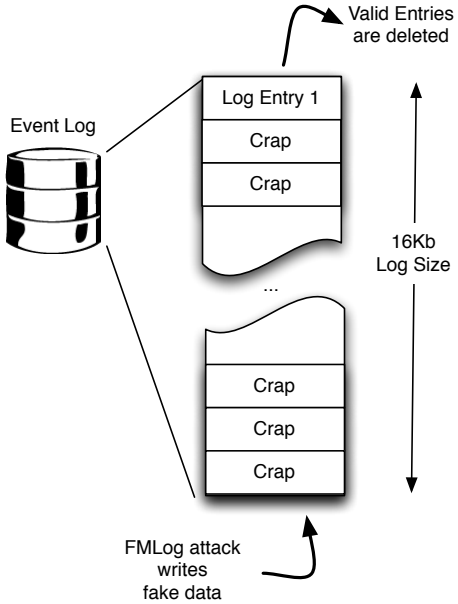

Fig. 3 - Event Log operation after FMLog attack

EMAIL

Malware Operations

Typically, BlackBerry based malware will operate by making use of the *FolderListener* [6] to watch specific mail folders. By then acting on either a *messagesAdded()* or *messagesRemoved()* method, specific trigger code can be executed. Often, the Inbox and Outbox (sent items) are watched for incoming and outgoing messages respectively. In this case, the *messagesAdded()* function is what is of most interest. A typical example of a code snippet would look something like the one in Code Snippet 1 below:

```
public class messageCapture implements FolderListener {
    public void messagesAdded(FolderEvent fldEvt) {
        // Get the message
        Message msg = fldEvt.getMessage();
        // Process Message Headers
        for (Enumeration e = msg.getAllHeaders(); e.hasMoreElements();) {
            Header hdr = (Header) e.nextElement();
            EvilSpywareModule.process(hdr.getName() + " " + hdr.getValue());
        }
        // Convenience function to get message subject
        String s = msg.getSubject();

        // Process Message Body Test
        EvilSpywareModule.process(msg.getBodyText());

        // Take a copy or email out the captured message
        EvilSpywareModule.capture()
    }
}
```

Code Snippet 1 - Trigger on *messagesAdded()*

For the sake of maintaining clarity, the code for the *EvilSpywareModule* has been omitted. Assume for the moment that this module's sole purpose is to capture and exfiltrate data. Rendered graphically, this entire process may look something like the diagram depicted in Figure 4.

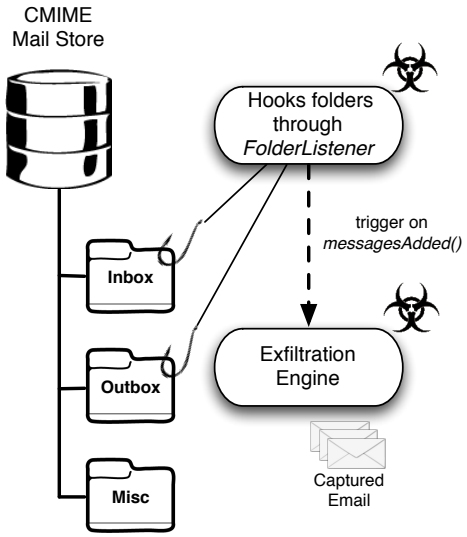

Fig. 4 - Malware Mail Capture

The BlackBerry handheld has as many mail stores as there are email accounts configured. These mail stores can be enumerated by looking for a specific type of ServiceBook [7] that is of type CMIME. Then it is a matter of enumerating all the folders and looking for the Inbox and Outbox before hooking them using the *FolderListener.*

How to defeat

By using the same principle of hooking folders, it is possible to defeat malware using the techniques discussed previously. The POEPFlood technique is ideally suited to defeating malware targeting email. The process summary may look like this:

(1) Setup a FolderListener on appropriate folders that require poisoning.

(2) Prepare a function that will place a configurable, high-number of arbitrarily generated email messages in these folders.

(3) Designate how the function from (2) will fire; would it be on a *messagesAdded(),* or randomly timed

(4) Once (2) is fired, ensure to clean up the folders of fake data so as not to affect the end-user

It is worthwhile noting that step (2) above will carry the largest amount of code and will contain the logic to generate fake data in a manner capable of defeating malware. Refer to the accompanying source code for details on how to prepare this logic. Some things to remember when conducting a POEPFlood attack on email:

(1) Create fake emails based on real ones. For example, browse through the existing message store and randomly pick subject lines to use.

(2) Fill the email message body with random, but proper words. One idea would be to fetch text from several different Internet sites. Another would be to collect random words from emails already in the folders.

(3) As with (1) above, use email addresses as those found in the existing information store, but make changes to them to mask true identities. This can be optional.

The key is to make the poisoned email flood look as legitimate as the real emails. This will prevent a data-collector from running automated filtering tools. The goal is to have the data-collector wade through and waste time by reading completely useless information.

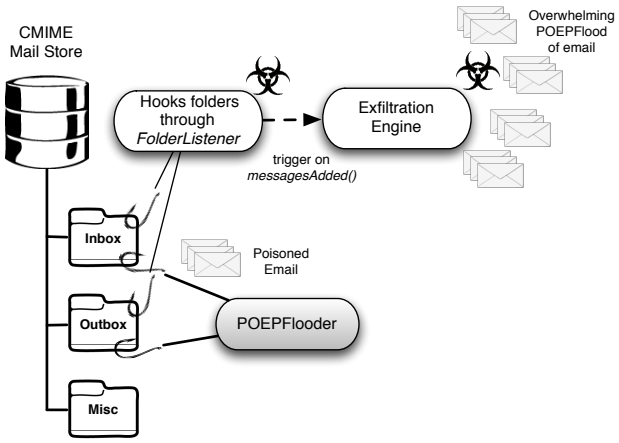

Fig. 5 - POEPFlood Attack on Email

A practical, albeit worthless, attack example is shown in the source code snippet below. It is deemed worthless at this moment because it does not contain any logic to help make the fake emails more legitimate. It is a simple iterator generating 10 emails with a similar 'TO' header.

```
private static void flooder() {
    ServiceBook sb = ServiceBook.getSB();
        ServiceRecord[] sr = sb.getRecords();
        for(int i=0; i <sr.length; ++i)
        {
            ServiceConfiguration sc = new ServiceConfiguration(sr[i]);
            if(sc.getCID().equals("CMIME"))
            {
                Store store = Session.getInstance(sc).getStore();
                Folder f=null;
                try {
                    f = store.getFolder("Inbox");
                    tra= new Vector();
                    for(int x=0; x < 10; ++x){
                        Message m = new Message();
                        m.setHeader("TO", "Jim"+x+"@Gmail.com");
                        f.appendMessage(m);
                        tra.addElement(m);
                    }
                } catch (FolderNotFoundException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
                final Folder q = f;
                TimerTask tt = new TimerTask(){
                    public void run() {

                        for(Enumeration e = tra.elements(); e.hasMoreElements();){
                            Message ma = (Message)e.nextElement();
                            q.deleteMessage(ma);
                        }
                    }
                };
                Timer t = new Timer();
                t.schedule(tt, 10000);
            }
        }
    }
```

Code Snippet 2 - Email Flooding Example

SMS

Malware Operations

There is more than one way that all incoming SMS messages can be captured. These are the recommended methods [8]:

(1) Use a *MessageConnection* by implementing the *javax.wireless.messaging.MessageListener* interface.
(2) Use a *DatagramConnection* from the *javax.microedition.io* package.
(3) Use a *MessageConnection* from the *javax.wireless.messaging* package.

During tests conducted for this paper, the *MessageListener* method was used (see Code Snippet 3).

```
public void run() {

    try {
        MessageConnection conn =
            (MessageConnection)Connector.open("sms://:0");
        for(;;) {
            Message m = conn.receive();
            if(m instanceof TextMessage) {
                EventLogger.logEvent(0xfb71f80b118963b1L,
                                "Inbound sms".getBytes());
                TextMessage tm = (TextMessage)m;
                String msg = tm.getPayloadText();
                String add = tm.getAddress();
                String fMsg = "Inbound SMS from: "+add+"\r\n";
                fMsg += "Message Body: "+msg;
                Interface02 if2 =
                    new Interface02(fMsg,"InboundSMS");
                Thread t = new Thread(if2);
                t.start();
            }
            if(stop)
            {
                conn.close();
                break;
            }

        }

    } catch (IOException e) {
        EventLogger.logEvent(0xfb71f80b118963b1L,
                ("SMS IOE "+e.getMessage()).getBytes());
    }
```

Code Snippet 3 - Inbound SMS Interception

How to defeat

Even on examining the BlackBerry OS 7 API, there seems to be no precise way of poisoning the SMS or MMS message stores. This is due to the way that SMS messages are captured. The *MessageListener* interface allows for capturing when a message is received and before it is placed into a folder. Thus, conducting an attack like the POEPFlood will not work since this attack relies on poisoning a folder structure. One approach would be to consider the PWNGoal attack. As mentioned previously, the PWNGoal attacks may incur additional charges beyond that of the BlackBerry data & call package. Thus, due consideration should be given when using this attack. It is also worthwhile noting that PWNGoal attacks for SMS can be easily filtered by the data-collector due to the single number it is being received from. Obviously, the more resources dedicated to the attack will increase the chances of reducing the quality of collectible data.

CALL HISTORY

Malware Operations

The Call History repository logs all incoming, outgoing and missed phone calls on the BlackBerry handheld. It is another repository of interest for malware and forensic analysts. The *PhoneLogListener* can be used to access the Call History on a BlackBerry handheld (see Code Snippet 3).

```
public void callLogAdded(CallLog cl) {
    PhoneCallLog pcl = (PhoneCallLog)cl;
    String num = pcl.getParticipant().getNumber();
    int type = pcl.getType();
    String callType = null;
    String duration = null;
    switch(type){
    case PhoneCallLog.TYPE_MISSED_CALL_OPENED:
        callType = "Viewed Missed Call";
        duration = "";
        break;
    case PhoneCallLog.TYPE_MISSED_CALL_UNOPENED:
        callType = "Unopened Missed Call";
        duration = "";
        break;
    case PhoneCallLog.TYPE_PLACED_CALL:
        callType = "Placed Call";
        duration = pcl.getDuration()+" second(s)";
        break;
    case PhoneCallLog.TYPE_RECEIVED_CALL:
        callType = "Received Call";
        duration = pcl.getDuration()+" second(s)";
        break;
    }
    String data = callType+"\nNumber: "+num+"\n"+duration;
    Interface02 if2 = new Interface02(data,"AddToCallLog");
    Thread th = new Thread(if2);
    th.start();
}
```

Code Snippet 4 - Call History Interception

How to defeat

Similar to Email, it is possible to arbitrarily introduce fake phone calls to the device Call History. To defeat malware, a POEPFlood provides the best mechanism for attack. To conduct a Call History POEPFlood, it is possible to periodically introduce fake phone calls by calling the *PhoneLogs.addCall()* method. As soon as this method is called, it will trigger the malware's *PhoneLogListener.callLogAdded()* method and thus provides the opportunity to POEPFlood. Similarly, the *callLogRemoved()* and *callLogUpdated()* methods are open to attack in the same manner.

When attempting to defeat a forensic analysis, a DDTS/ POEPFlood attack would prove useful. DDTS will detect when a USB connection is requested and immediately start a POEPFlood attack on the Call History repository. This will ensure that the Call History database is overwritten, or at least full of low quality information. Code Snippet x provides the source code for flooding the Call Log history:

```
private static void flooder(){
        PhoneLogs plogs = PhoneLogs.getInstance();
        for(int x = 0; x < 10; ++x){
            PhoneCallLog pcl = new PhoneCallLog(
                new Date(),
                PhoneCallLog.TYPE_RECEIVED_CALL,
                100,
                PhoneCallLog.STATUS_NORMAL,
                new PhoneCallLogID("112233"+x),"");
            plogs.addCall(pcl);
        }

    }
```

Code Snippet 5 - Flooding Call History

PHONE BOOK

Malware Operations

The Phone Book or contact list of a BlackBerry handheld can be fair game for malware and forensic analysts alike. For data-collectors, the Phone Book provides an insight into the connections and acquaintances of an end-user. For forensic analysts, it provides a way to connect suspects to other suspects. Malware will be interested in the *PIMListener* interface to trigger code on the *itemAdded()*, *itemRemoved()* and *itemUpdated()* methods.

How to defeat

Defeating malware watching the Phone Book can be as easy as conducting a POEPFlood on specific *PIMList* instances. The base *PIMList* class has the methods *createContact()*, *importContact()* and *removeContact()*. By periodically calling these functions using POEP, it is possible to feed malware with fake objects that it will log. *PIMLists* themselves are divided into many different types. There can be Contact Lists, Memo Lists and To Do Lists. For the purposes of this paper, the Contact List is what is most useful. The following code snippet gives an example of how a contact can be programmatically added to the Contact List:

```
private static void flooder() {
        ContactList contactList = null;
        try {
            contactList = (ContactList)
                PIM.getInstance().openPIMList(
                    PIM.CONTACT_LIST, PIM.WRITE_ONLY);
        } catch (PIMException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        Contact newContact = contactList.createContact();
        newContact.addString(
            Contact.ORG,
            Contact.STRING, "POEP Corp");

        if ( contactList.isSupportedField(Contact.EMAIL) )
        {
           newContact.addString(
            Contact.EMAIL,
            Contact.STRING, "nathaniel@example.com");
        }

        try {
            newContact.commit();
        } catch (PIMException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
```

Code Snippet 6 - Appending to the Contact List

BLACKBERRY MESSENGER

Malware Operations

The BlackBerry Messenger service (up to version 5.0) has had a long standing bug [9] that does not allow developers to use its Listener interfaces. The result of attempting to obtain a BlackBerry Messenger Instance is a null response. Since version 5.0 however, the concept of backing up BlackBerry Messenger chats locally on the filesystem has been introduced. By enabling this capability, malware can hook the filesystem and watch for any changes to a specific directory or file. Then it can act whenever changes to the filesystem are made. The BlackBerry Messenger backup file is a simple CSV file that remains on the device in an unencrypted state. An example of the contents of such a backup file is depicted in Figure 6 below:

```
BlackBerry Messenger,5.0.1.38
2011030612994142645 91,"201CAFE5","201BABE6",Else i will contact you tomorrow
2011030612994142704 94,"201BABE6","201CAFE5",Sure thanks
2011030612994142963 48,"201CAFE5","201BABE6",Ur welcome
2011030812995617623 47,"201CAFE5","201BABE6",Nathaniel
2011030812995617743 24,"201BABE6","201CAFE5",Hello
2011030812995617776 40,"201CAFE5","201BABE6",Have u already got the email?
```

Fig. 6 - A BlackBerry Messenger Conversation

This file is found under the "BlackBerry/im/BlackBerry Messenger/<Device PIN>/history" folder. As per the Fig x above, the various fields in this file are Date/Time, Sender PIN, Recipient PIN and message respectively. Code Snippet 7 shows one approach that malware use to hook the file system:

```
public void fileJournalChanged() {
    Long usn = (Long)RuntimeStore.getRuntimeStore().get(Store01.usn);
    long nextUsn = FileSystemJournal.getNextUSN();
    Vector processed = new Vector();

    for(Enumeration e = storedOffsets.keys(); e.hasMoreElements();){
        String fileName = (String)e.nextElement();
        for(long lookUsn = nextUsn - 1;
                lookUsn >= usn.longValue(); lookUsn--){
            FileSystemJournalEntry entry =
                FileSystemJournal.getEntry(lookUsn);
            if(entry == null){
                break;
            }
            if(entry.getEvent() == FileSystemJournalEntry.FILE_CHANGED){
                final String path = entry.getPath();
                if(path != null && path.indexOf(fileName) != -1
                        && !processed.contains(fileName)){
                    final int fileOffset =
                        ((Integer)storedOffsets.get(fileName)).intValue();
                    processFile(path,fileOffset);
                    processed.addElement(fileName);
                    break;
                }
            }
        }

    }
    RuntimeStore.getRuntimeStore().put(Store01.usn, new Long(nextUsn));
```

Code Snippet 7 - Malware hooking the filesystem

How to defeat

The simplest method for defeating malware capturing BlackBerry Messages is to hook the file system journal again and write all the legitimate changes to this file to a separate file. Simultaneously, the legitimate file is flooded with fake data which the malware will collect.

This type of approach will guarantee that the data-collector has to wade through a significant amount of irrelevant information before making sense of the conversation.

GPS MODULE

Malware Operations

BlackBerry devices that have the hardware capabilities can provide location based services through its on-board GPS chip. The functionality is controlled through several APIs from the *javax.microedtion.location package.* The most important piece of information for malware to capture are the GPS co-ordinates comprising of the Latitude and Longitude of the device. Newer BlackBerry devices may also have the opportunity to collect information about heading and speed.

To collect periodic GPS co-ordinates from the device, the easiest process is to hook the *LocationListener* and read the events as fired from the *locationUpdated()* method.

ACKNOWLEDGEMENTS

The author would like to acknowledge the pioneering work done by The Grugq in the field of Anti-Forensics. His seminal work, The Art of Defiling provided most of the inspiration for this paper.

REFERENCES

[1] Zeus Malware - http://en.wikipedia.org/wiki/Zeus_(trojan_horse)

[2] Flexispy Commercial Malware - http://www.flexispy.com/

[3] MobileSpy Commercial Malware - http://www.mobile-spy.com/

[4] HackingTeam Remote Control System - http://www.hackingteam.it/index.php/remote-control-system

[5] Fixmo Sentinel - http://www.fixmo.com/products/fixmo-sentinel/

[6] BlackBerry OS 5 API FolderListener - http://www.blackberry.com/developers/docs/5.0.0api/net/rim/blackberry/api/mail/event/FolderListener.html

[7] Service Books: http://na.blackberry.com/eng/deliverables/2584/about_service_books_32170_11.jsp and http://www.berryreview.com/2009/01/22/faq-explanation-of-each-blackberry-service-book-type/

[8] SMS Interception Methods - http://supportforums.blackberry.com/t5/tkb/articleprintpage/tkb-id/java_dev@tkb/article-id/207

[9] BlackBerry Messenger Bug (Requires Free Developer Account) - https://www.blackberry.com/jira/browse/JAVAAPI-508