Android and Wireless Data-Extraction Using WI-FI

Bert Busstra[1], N-A. Le-Khac[2]
[1] Digital Forensic Investigator
Central Unit Dutch Police
Zoetermeer, The Netherlands
[2] School of Computer Science & Informatics
University College Dublin
Dublin 4, Ireland
bert.busstra@tcca.nl, an.lekhac@ucd.ie

ABSTRACT: *Today, mobile phones are very popular and a fast growing technology. Mobile phones of the present day are more and more like small computers. The so-called "smartphone" contains a wealth of information. This information has been proven to be very useful in crime investigations, because relevant evidence can be found in data retrieved from mobile phones used by criminals.*

In traditional methods, the data from mobile phones can be extracted using an USB-cable. However, for some reason this USB-cable connection cannot be made, the data should be extracted in an alternative way. In this paper, we study the possibility of extracting data from mobile devices using a Wi-Fi connection.

We describe our approach on mobile devices containing the Android Operating System. Through our experiments, we also give recommendation on which application and protocol can be used best to retrieve data.

Keywords: Android, Wireless Data Extraction, Network Protocols, SSHDroid, Rsync.

Received: 2 November 2014, Revised 19 December 2014, Accepted 29 December 2014

1. Introduction

Almost every citizen carries a mobile phone. According to Central Bureau of Statistics (CBS) (Statistics Netherlands) Mobile Internet use continues to grow. CBS states in a press release PB12-060[2] on 23 October 2012 that: (i) *Six in ten internet users go*

[1]Wi-Fi: http://en.wikipedia.org/wiki/Wi-Fi

online with mobiles; (ii) Young people use mobiles most; (iii) Mobile phone is most popular (iv) Used for communication, news and fun

Mobile phones of the present day are more and more like small computers. The so-called "*smartphones*" contain a wealth of information each. This information has been proven to be very useful in crime investigations, because relevant evidence can be found in data retrieved from mobile phones used by criminals.

For instance evidence might be extracted from: Address books, SMS/MMS, Call-history, Twitter, Web browsing cache, E-mail, Social media apps like Facebook, Instant messaging (for example: Whats-up, Blackberry messenger and Skype.)

Today even more mobile phone users posses a smartphone. One of the major operating systems found on these smartphones is Android.[3] According to the International Data Corporation (IDC) Worldwide Quarterly Mobile Phone Tracker, total Android smartphone shipments worldwide reached 136.0 million units, accounting for 75.0% of the 181.1 million smartphones shipped in third quarter of 2012.[4] It is no surprise that most smartphones onto which digital evidence is conducted these days is an Android device. Therefore, we only focus on Android devices in this paper.

The purpose of this paper is to focus on the unsuccessful cases and show a non-conventional way to extract evidence electronically, even if access to the device is limited.

The focus will be on the possibility to extract data from an Android file-system by accessing it through the air using a Wi-Fi1[1] network connection. Most Android devices can be connected to a personal computer by using an USB-connection. Using this USB-connection the content of an Android device can be examined. During forensic[5] investigations, software-packages, like XRY [6] or UFED [7], can be used to acquire data from the device.

When a device is "*rooted*"[8] also full access to the file-system can be gained. In that case more information can be retrieved from the device.

Android rooting is the process which allows the users of Android-devices to gain privileged control (known as "*root access"*) within the Android's subsystem.

Using the ADB tool combined with root-access, the examiner can even make physical dumps from the partitions found on the device, using the Unix "*convert and copy file*" command: "*dd*"[9]. In this case all information can be retrieved for later analysis.

Android Debug Bridge (adb)[10] is a versatile command line tool that lets you communicate with an emulator-instance or connected Android-powered device.

Sometimes, while conducting an investigation on an Android device, it may happen that it isn't possible to retrieve data from it using an USB-connection.

Reasons for these might be a failing cable or a dirty connector. In these cases an examiner will only investigate the external memory-card if it's available in the device. However, not all devices come with external memory. Sometimes the device only contains internal (in-build) memory. The contents of this memory can normally only be extracted using a USB cable-connection.

At that point there should be other methods to acquire data from the in-build memory storage Searching the Google Android Play app-store[11] we discovered several applications that could be used to gain remote access to the Android file-system. These applications supported different network protocols like: SMB[12], FTP[13], SSH[14], RSYNC[15], HTTP[16].

When the connection is made, using one of the mentioned protocols, no artefacts should be accidentally changed on the target during transfer. Also when the files and folders are transferred, the attributes on them must be preserved.

This process of retrieving data can be automated using common Linux tools and bash scripting.

In this paper we investigate the feasibility in making a connection to the Android file-system using wireless networking (Wi-Fi). The rest of this paper is organized as follows. In Section 2, we present the background where we review related work in this context. Section 3 is the problem statement where we formulate our research questions. In Section 4 we describe our approach of extracting data using a Wi-Fi connection. It contains the analysis of research on the used applications and protocols.

In Section 5, we describe, evaluate and discuss on results, including the answers on the research-questions formulated in Section 3. The answers are based on the outcome of the research described in Section 4. Finally, we conclude in Section 6.

2. Background

A lot is written about Android devices in general but less on the forensic aspects of retrieving data from the Android operating system.

To the best of our knowledge there has been no research on using Wi-Fi for retrieving data from mobile devices (in particular from Android devices) in a forensic[5] setting.

An USB cable-connection looks like the most commonly technique among forensic software that is used for retrieving data from mobile devices.

On the subject of forensic examination on Android devices, author Andrew Hoog[17] wrote a book called: Android Forensics. In chapter 6 of his book the author describes several Android Forensic Techniques to acquire data from an Android device in a logical or physical way using an USB cable-connection. The author continues that to gain access to the data on the file-system of the Android-device, the developer menu-option: "*USB-debugging*" must be enabled in the settings-menu of the device.

For this reason Hoog describes in chapter 3 how the access to this menu can be gained even when the user-interface of the device is locked.

If the access to the settings-menu is blocked, it is nearly impossible to gain access to the devices memory. In most cases an investigator needs unlimited access to the user-interface of the Android-device to adjust some options in the settings-menu. For our research we will not need an USB-connection to access the file-system, however we still need unlimited access to the Android settings-menu to set some options in order to access the data using Wi-Fi. For instance we need to adjust the Wi-Fi settings to let the device connect to an access-point. (See chapter 4.3)

In chapter 3 of the Hoog explains the installation and usage of the Android Software Development Kit (SDK)[18] on the investigating computer. We used this SDK during our research. (See paragraph 4.2 of this paper)

Another author[19] focused on evaluating the efficiency of software tools that support the extraction of information stored on Android devices. This author also described the architecture of Android systems and the methods for data extraction. He chose three software tools to evaluate: Oxygen Forensic Suite 2012 standard[20], MOBILedit Forensic[21] and AFLogical.[22] Among these three tools, only MOBILedit allows to use Wi-Fi to extract data. However, the author only mentions that MOBILedit can use the wireless connection by installing a small application in the mobile device to pull the data. There has been no further investigation, analysis and discussion on the Wi-Fi capability of this software tool.

In [23], the author focuses on the design and implementation of an Android application ("*app*") that automates the collection of useful data for investigating. However, this approach also uses USB cable-connection to extract relevant data.

In [24], the authors introduce a method for acquiring forensic-grade evidence from Android smartphones using open source tools. The authors investigated in cases where the suspects used the Wi-Fi or Bluetooth interfaces of smartphones. However, the investigators also used an USB cable-connection in their solution.

To the best of our knowledge there has been no research on using Wi-Fi for retrieving data from (Android) mobile devices, in the context of a forensic setting. An USB cable-connection looks like the most commonly used technique among forensic software that is used for retrieving data from mobile devices. However MOBILedit Forensic allows a Wi-Fi connection to extract data, its protocols are not disclosed and moreover, the software is not free [25].

To investigate a mobile device an investigator needs to interact with the device directly. The examiner must be sufficiently trained to examine the device. He has to think before he acts, because he must know exactly what effects his actions have on the

data, and he must be prepared to prove it. If the examiner did change the data, he must be prepared to explain why it was necessary and what data has changed. For this reason the examiner should document every action taken. For forensic examiners in the United Kingdom the usage of some principles as a guideline during their research is quite common. The Association of Chief Police Officers (ACPO)[26] in the United Kingdom produces this guideline.

3. Problem Statement

The preferred method of connecting to the file-system of an Android device is by using an USB cable-connection. The "*adb*" command line tool is used to communicate with the device. The advantages of using this toolset are:

- Transfer data to and from the device,
- Fetch selected files and directories. (Logical extraction),
- Make a copy of block-devices (dump partitions) using the Unix-command "*dd*" (data description).

There is however one disadvantage using an USB-connection. Like we mentioned in the introduction chapter, in some occasions it will for some reason not be possible to make a connection to the Android device using an USB cable-connection. In that case it might not be possible to fetch the data stored on the Android device.

If the device contains an external memory-card, like a micro-SD memory-card, in most cases it will be possible to acquire data from that particular memory-card using a memory-card-reader. But if the phone contains only "*in-build*" memory another option to acquire the data might be needed. This leaves a couple of questions unanswered. The questions we would like to answer in this paper are:

- Is data extraction from an Android-device possible using Wi-Fi?
- If so, will it be a good alternative for using an USB-connection with the adb-toolset?
- Can we extract all data as we do as with USB, like extracting deleted data?
- Which network-protocols can be used and which is best?
- Which applications in the Google Play app-store that could be used for this?
- Will the file-system of the Android target-device remain untouched?
- Will the file attributes be preserved during file-transfer?
- Will the given solution work on different versions of the Android operating system?

4. Adopted Approach

Examining the Android operation-system, we discovered that there was no available option for a user to remotely connect to the Android file-system using Wi-Fi. When an Android device is purchased it does not come with functionality to touch the file-system remotely, in order to back-up (user) data. Based on this we concluded that to acquire data from an Android device using Wi-Fi, an application (app) should be installed on the device first to make the wireless communication possible.

Like mentioned in chapter 1 there are several apps available from the Google Play app-store to connect to the Android file-system using Wi-Fi. These apps use common network-protocols to connect. Connected to the Android target-device, using Wi-Fi and the proper app, it is possible that a personal computer can be used to retrieve data from the target's Android file-system. After the transfer, the data can be processed for further examination.

Because the Android target-device contains a "*live*" environment, installing apps on the device will off course have its impact on the forensic (clean) state of it. So some research is needed to gain insight into the memory footprint of the installed package on the device.

According to the ACPO[26] guidelines, it is necessary to describe in detail what steps are undertaken and which artefacts might have been changed on the Android target-device during an investigation. We will discuss the possible forensic implementations

in paragraph 4.2.

4.1 Research on protocols

For testing the different protocols to communicate with an Android-device using Wi-Fi, different applications for each protocol might be needed. First it has to be determined which app(s) fit best to suit the job. During research we found out about the application named: Servers Ultimate. [27]

Though the application (app) was not free, it could be used for a small trial period. We discovered that:

• The app could be used best to install and test different types of servers based on the Internet Protocol (IP).

• It needed additional modules, which had to be downloaded separately after the installation of the main package. This would leave even more artefacts on the Android-device, which, seen from a forensic perspective, is not what we wanted.

We decided to use this app only for testing purposes during our investigations.

Rsync

Rsync can be used to sync data across a network, even in both ways, where it can be instructed to preserve the file attributes. To retrieve data, the rsync-program is needed on the Android target-device. Since rsync needs a "*listening*" service on the target side, we searched Google Play app-store for an rsync-server. Because we could not find any rsync-server app, except for the rsync-server functionality in the Servers Ultimate app, we decided to test it, merely to discover if it should work or not. During our research we discovered that it didn't work at all with a non-rooted Android-target-device. It's uncertain why this was. It might be caused by the usage of a non-conventional tcp-port, which has to be used in a non-rooted environment, other then the standard 873 port under normal conditions. This is because on non-rooted Android devices the tcp-ports below 1024 may not be used for communication.

Using a rooted android device it is quit easy to retrieve data, using the rsync command:

```
investigator@host:~$ rsync -avv rsync://192.168.1.119:/sdcard/

opening tcp connection to 192.168.1.119 port 873

sending daemon args: – – server ––sender – –vvlogDtpre.iLsf . sdcard/

receiving incremental file list

delta-transmission enabled

drwxrwxr–x        4096 2013/07/11 10:31:41 .

–rw –rw –r– –          82 2013/07/11 10:31:40 .tmpsu

–rw –rw –r– –       63316 2013/06/24 12:50:41 DevIcon.fil

..

..

..

drwxrwxr-x        4096 2013/07/11 10:31:41 tempdat

-rw-rw-r– –          82 2013/07/11 10:31:41 tempdat/.tmpsu

sent 229 bytes  received 42511 bytes  28493.33 bytes/sec

total size is 903355279  speedup is 21136.06
```

Conclusion:

1. There is currently no standalone rsync-server app to be found in the Google Play app-store, which can be used to remotely access the Android file-system to transfer data locally using Wi-Fi.

2. The Android-device must be rooted in order to make the rsync of data possible.

HTTP (WebDAV)
WebDAV uses the HTTP protocol. Its data can be accessed using a web-browser. Several (free) WebDAV server-apps can be found on the Google Play app-store.

When the WebDAV server-service is started on the Android-target it listens for incoming connections. Then the remote WebDAV-share on the Android target-device can be mounted locally on the computer of the investigator. Therefor the application davfs2[28] must be installed as root:

```
host:~# apt-get install davfs2
```

The remote file-system of the Android target-device will be mounted read-only on a mount-point defined on the investigating computer:

```
host:~# mount -t davfs 'http://192.168.1.119:8888' /media/davfs/ -o ro
```

Then rsync can be used to fetch data to the computer of the investigator:

```
host:~$ rsync – –avz –progress /media/davfs/ /home/investigator/evidence
```

Conclusion:

The remote file-system must first be mounted on the investigating computer in order to retrieve the data.

It is also possible to use the "*wget*"[29] command to fetch files and folders from a WebDAV folder directly, without mounting the remote-folder first. But using wget a lot of index.html files are generated. This will pollute the folder into which the data is placed onto the investigating computer. Also, with using wget, not all permissions (owner, group, authorization) will be preserved.

FTP
There are several FTP server-apps available in the Google Play app-store. Once an FTP-app is installed, the ftp command or wget (below) can be used to retrieve data:

```
host~$ wget -r – –timestamping – –user=root – –password = toor -c ftp://
192.168.1.119:2121/sdcard/
```

```
lftp 192.168.1.119:/> mirror
mirror: Access failed: 550 That path is inaccessible (/cache
mirror: Access failed: 550 That path is inaccessible (/config)
mirror: Access failed: 550 That path is inaccessible (/data)
mirror: Access failed: 550 That path is inaccessible (/dev/com.noshufou.android.su)
..
..
Interrupt
lftp 192.168.1.119:/>
```

Conclusion:

FTP is fast, but the permissions (owner, group, authorization) might not be transferred properly when the commands wget or ftp are used. Also for some reason the access to some folders is denied on a rooted device, using ftp, as listed in the following

example, where the lftp[1] "*mirror*" command is used on a rooted Android target-device:

Samba (SMB)
It seems to be rather difficult to make a connection with an SMB-service on a non-rooted Android target-device once an SMB-app is installed. It is likely that this might be caused by the SMB-service running on the Android target, which uses non-standard TCP-ports. Microsoft Windows and Linux operating systems only connect to SMB-servers running on conventional TCP-ports: 137-139. TCP-ports below 1024 are not usable on non-rooted Android devices because of security issues. Because of this limitation we decided to skip researching the usage of this protocol any further.

SSH (SSHFS)
We tested most of the SSH server-apps which are available through the Google Play app-store.
Several SSH-servers need root access to install or to function properly. After thorough research we decided that our SSH-server of choice in this paper would be: SSHDroid.[31]

We chose SSHDroid mostly because it is quick to set-up and runs smoothly on rooted and non-rooted Android devices, taking almost no system-resources.

> **Note:**
>
> The default password for the user root in SSHDroid (and most other SSH-server packages) is: admin. The default port-number to connect on non-rooted devices is: 2222. But this is changed to the standard port: 22 on rooted devices.

With SSHFS[32] it is possible to mount a remote share using an SSH-connection. Once the share is mounted locally, data can be transferred using rsync or the Linux copy command: "*cp*". First the ssfs-package must be installed on the investigating computer:

```
host:~# apt-get install sshfs
```

Secondly the SSH-server (for example: SSHDroid) must be installed on the target-device, which will then act as an SSH-server.

Next the Android file-system will be mounted read-only (ro) on the computer of the investigator using sshfs:

```
host:~# sshfs root@192.168.0.126:/ /media/android/ -o ro
```

Copy data from the target-device to the investigating-computer with rsync:

```
host:~# rsync -av – –progress /media/android ~/android-investigation
```

Also the Linux copy command ("cp") could be used to transfer the data:

```
host:~# cp -av /media/android ~/android-investigation
```

Conclusion: the use of SSHFS is ok, but we still need to mount the remote file-system first.

SSH + SFTP or SCP
With an SSH-service in place it is also possible to transfer data using the commands: SFTP (Secure-FTP)[33] and SCP (Secure copy)[34].

```
host:~$ scp –P2222 –prv root@192.168.200.120:/storage/sdcard0/WhatsApp /tmp
host:~$ sftp -P2222 -rp – – preserve 192.168.200.120:/storage/sdcard0/WhatsApp /tmp
```

Conclusion:

Despite of using the correct "*attributes preserving*" parameters, we discovered that both SFTP and SCP did not proper transfer the ownership and permissions of the data.

```
host:~$ scp –P2222 rsync \
root@192.168.1.119:/data/data/berserker.android.apps.sshdroid/dropbear/
```

SSH + rsync
The SSH-server can also be used as a "*listening*" server for rsync. Once the SSH-server is installed on the Android target-device, the rsync-executable is needed on the target-device. Most of the time the executable has to be transferred manually to the target-device because many SSH-server apps from the Google Play app-store do not ship with rsync.

Also in the SSHDroid app the rsync-executable is not included. It has to be installed manually. An rsync executable for the android platform is freely available on the internet.[35]

For SSHDroid the rsync-executable must be put in the
/data/data/berserker.android.apps.sshdroid/dropbear folder in order to function properly:

The rsync executable must be made executable:

```
host:~$ ssh -p2222 root@192.168.200.119 chmod 755 \
/data/data/berserker.android.apps.sshdroid/dropbear/rsync
```

Check the executable bit with:

```
host:~$ ssh -p2002 192.168.200.125 'ls –l'
\ data/data/berserker.android.apps.sshdroid/dropbear/rsync
-rwxr-xr-x u0_a236  u0_a236    916300 2013-07-11 22:31 rsync
```

Alternate SSH app
On a non-rooted environment the application: SSHelper[36] would be an alternative for using SSHDroid.

Advantage:
SSHelper includes an rsync-executable.

Disadvantage:

- SSHelper only runs on non-rooted devices. Also SSHelper requires Android version 3.2, API 13, or newer, so SSHelper will not work on older Android devices.

- Timestamps on transferred files and folders will be displayed in UTC.

4.2 Forensic implementations of installing apps on Android target
To measure the impact of installing software on an Android system, the installation and de-installation of the software can be monitored. Both states can be compared with each other using the program Winmerge.[37] The program can be used to make visible which artefacts were placed on the system during the installation and which were left after the de-installation of an application.
To test this, an Android Emulator can be used. The Android Emulator and the Android Software Development Kit (SDK) are part of the Android software Development Toolkit (ADT) Bundle[18].

The Android SDK provides developer tools necessary to build, test, and debug apps for the Android operating system. The set-up of such an environment is documented online[38], which we used to download, install and set-up the developer environment.

Using the "*adb install*" command the SSHDroid-package can be installed on the emulator. Before the installation a snapshot of the /data directory is made using the "*adb pull*" command. After the installation again a snapshot is made. After de-installation of SSHDroid a third snapshot is made. Using the program Winmerge the contents of both snapshot 1 and 2 are compared in order to see which artefacts would be visible after the installation related to the SSHDroid app. Winmerge generated the following list:

Filename	Folder
berserker.android.apps.sshdroidpro-1.apk	app
data@app@berserker.android.apps.sshdroidpro-1.apk@classes.dex	dalvik-cache
berserker.android.apps.sshdroidpro	data
cache	data\berserker.android.apps.sshdroidpro
dropbear	data\berserker.android.apps.sshdroidpro
home	data\berserker.android.apps.sshdroidpro
lib	data\berserker.android.apps.sshdroidpro
shared_prefs	data\berserker.android.apps.sshdroidpro
packages.list	system
packages.xml	system

After installation of SSHDroid
It showed that the app was installed in a newly created directory named: **"/data/data/berserker.android.apps.sshdroidpro"** Then snapshot 2 and 3 were compared. This time only two files had changed: packages.list and packages.xml

The Android operating-system stores the names of the installed packages the file: packages.list. This method of package-listing can be compared to common Linux operating-systems like Debian GNU/Linux[39] or Ubuntu[40] where the Aptitude[41] package-manager stores a list of installed packages in the file: /var/lib/aptitude/pkgstates. When an application is removed from the system, the entry, with the name of the package in it, is removed from this list, thus the package-file is modified.

4.2.1 How to Check Root Status of Android Device

Note: Only if Android devices are "*rooted*" all data can be accessed and logically retrieved using the in this paper described methods. If the device is non-rooted, access will be permitted to several system-folders. In that case only the accessible files and folders will be preserved.

Before the start of an investigation it might sometimes be useful to determine if an Android-device is rooted or not. But how can one know if an Android-device is rooted[8] or not?

1. If an Android device is rooted often an application called "*SuperUser*" or "*SU*" is installed. This is the program that will determine if certain "*super user*" commands may be used or not. If you see a super user program installed, then it is most likely rooted.

2. If a terminal app like Connectbot is installed, open it. If the prompt is showing the pound sign (**#**) the device is rooted. If it presents a dollar sign (**$**), try entering the command "*su*" command. (without quotes) You might see a dialog asking for permission from the *Superuser*-app, which you can safely grant. If the prompt changes to the pound sign, then the device is rooted.

4.3 The test-environment

In order to make the wireless connection, every basic Wi-Fi access-point can be used. It should however be verified that the access-point is NOT connected to the internet. A Wi-Fi network must be created on the access-point for the Android-devices to connect. Recommended is to use an easy to remember network-name (SSID)[42].

> **Warning**:
>
> The menu-settings of the Android target-device MUST be accessible in order to setup a Wi-Fi connection with the access-point. If the screen of the Android-device is locked, than it's not possible to set up the device for a file-transmission using Wi-Fi.

The Wi-Fi access-point uses DHCP[43] to assign an IP-address to the target-device. For this reason the DHCP feature should be enabled on the access-point.

We won't use Google Play app-store for downloading and installing the SSHDroid-package. Instead the investigating computer will run a web-server (Apache) to serve the SSH-application to the Android target-device. For this to happen the investigating computer must be connected to one of the LAN ports on the access point. It can be advised to give the investigating computer a static IP.

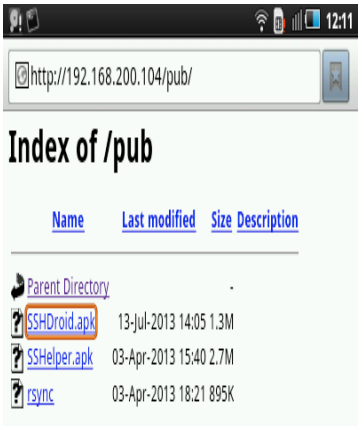
> **Note:** Please check that the access-point will provide an IP to the Android target-device in the same IP range as the investigating computer. So the DHCP scope on the access-point should be set-up correctly.

On the investigating computer a Linux operating system is installed, running an Apache[44] web-server and an OpenSSH-server. The SSHDroid-package will be served to the network using Apache-webserver. For example: the SSHDroid.apk package-file can be put in a shared web-directory named "*pub*" or "*public*": /var/www/pub on the investigator computer.

It must be checked that the web-server is up and running and that the "public" share is accessible. If not, the webserver configuration should be adjusted accordingly. Instead of Apache another web-server may be used. The setup and configuration of a webserver and an access-point is beyond the scope of this paper.

4.3.1 Package installation

Use the native web-browser on the Android target-device and point it to the web-address of the "*public*" share on the web-server onto which the SSHDroid package is served. (For example: http://192.168.1.100/public/) A directory listing will be shown. Download the SSHDroid package by tapping on the "*SSHDroid.apk*" link.



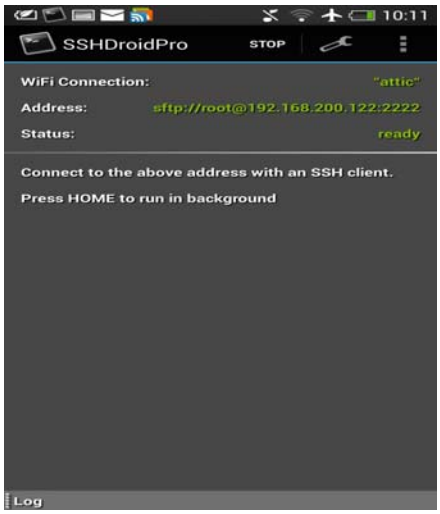
This screenshot shows the Samsung Apollo, GT5800 Android device, browsing to the public folder on the webserver.

> **Note:**
> Before installing an app other then using the Google Play app-store, in the menu of the Android target-device, the option: "*Allow installations if apps from unknown sources*", *or* "*Install from unknown/untrusted market*" must be activated. In most cases, as of Android version 4.0, this can be found here:
> **Settings -> Security -> Device administration -> Unknown sources**

After downloading, the installer-dialog will start. If that is not the case: tap the download- notification-bar (sliding the upper-bar down) and tap on the downloaded package. The installer message will appear. Follow the instructions on the screen to install the package.

SSHDroid will be started automatically after install. This will take a little time only the first time the app is started. Just press the Home button to continue.

From the running SSHDroid-app on the Android target-device it will be easy to see which dynamic IP address is leased from the access-point's DHCP-server. Take note of this IP, because it is the IP that will be needed to connect to from the investigating computer. (See screenshot below)



The screenshot on the left shows a running SSHDroid application on a non-rooted HTC One X Android device.

As can be seen, SSHDroid assigned tcp-port: "2222" to the ssh-server instead of the common ssh port number: "22". This is known to the fact that non-rooted devices cannot connect locally to TCP ports below 1024.
(See also paragraph 4.1, Rsync, page 12)

4. 4 Putting it all together – scripting

During research, we created a user-interactive bash[45] script (See Chapter 7, Appendices) to ease the usage of commands for downloading (selected) files and folders from an (non-rooted/rooted) android device using Wi-Fi.

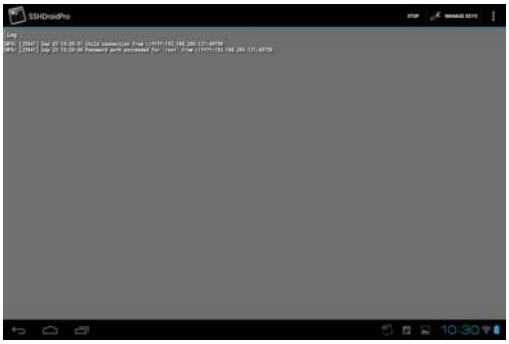
Once the script is started it will ask for input, like the evidence-number, IP-address and SSH-port-number to connect, SSH-username and password, directory to back-up and the inclusion (not implemented yet in script) or exclusion of files and directories.

The script must be executed as user "*root*". A "normal" user cannot run the script, because the loop-function, used in the script, needs elevated user-rights. For security reasons a non-root user will normally not have the privileges to execute a "*mount*" command using a loop-device. The "*loop*" functionality is needed to mount the "*.dd*" container-file locally.

Note:

SSHDroid but also other SSH-apps have a couple of default settings, which can be changed all the time, like the ssh port-number and the root password.

The screenshot below shows a successful login to the SSHDroid ssh-server, which is running on the Archos G9 101 Android tablet.



The script will create a container-file using the Linux-command "dd" [9]. This file will be mounted and opened for reading. Then ssh and rsync commands will be used to retrieve the data, which will directly put into this container. When the retrieval of data is finished the container-file will be un-mounted. When this has been succeeded the container file can be opened in most forensic applications for further examination of its contents.

The script must be run from a Linux-shell or terminal. In order to run the script on the investigating computer, some packages need to be installed on that computer:

host:~# aptitude install ssh apache2 sshpass rsync awk[46]

4.4.1 The SSH connection options

OpenSSH allows us to run commands on remote systems, showing the results directly. It can also been used for logging into systems for automating common tasks with shell-scripts. For those scripts to function properly, sometimes user passwords need to be stored in the script itself. However storing passwords in scripts is not secure.

There are at least two methods having not to store the password in the script itself; use SSH authorized keys or "*sshpass*".

4.4.2 sshpass[47]

With sshpass a password will be served to ssh when it is used in a keyboard-interactive password authentication. A script might show something like this:

```
sshpass -p $pass ssh -p $pnum -l $user \
-o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no $ip \
-o LogLevel=quiet
```

Because the script is interactive the user will be asked for the IP and port to connect to, username and password, which will be assigned to the variables $ip, $pnum, $pass and $user respectively.

The following ssh-options are used to suppress warnings and dialogs when the ssh-connection is set-up:

- UserKnownHostsFile = /dev/null
- StrictHostKeyChecking = no
- LogLevel = quiet

4.4.2 Authorized keys

In the script we used "*sshpass*" to show how to make an ssh-connection using a bash-script without adding the ssh-password to the script itself. It is however easier to use authorized keys when logging in. To enable this, a pair of keys has to be created. One of these keys will be appended to the authorized_keys-file on the remote system. Once this is done it is possible to login without being prompted for a password.

If there is no key pair generated on the investigating computer yet, there has to be created one. If there is already a key pair available it can be used. We choose to generate a key pair using RSA[48]. By default the keys will be stored in:

```
~/.ssh/id_rsa and ~/.ssh/id_rsa.pub
```

If neither of the two files are there, then one should be generated.

Generate a new key pair with the following command will prompt for a location to save the keys, but not for a passphrase.(**-N ""**)

```
host:~# ssh-keygen -t rsa -N ""
```

If the defaults are accepted a pair of files is created, as shown above, with no passphrase. No passphrase means that the key files can be used without the system asking for any password. It is needed because some things must be automated in the script.

```
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
/root/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
65:b4:36:d3:ad:59:38:ac:7b:76:12:fb:ca:e9:0a:45 root@host
The key's randomart image is:
+—[ RSA 2048]——+
|           .      |
|          . + o   |
|           E = o  |
|          = + =   |
|         S o +    |
|          . . o   |
|         . . = .  |
|          . + =   |
|           .o=..  |
+—————————+
```

Next the contents of the .pub file need to be appended to the correct location on the remote server. Using the command: ssh-copy-id will copy the key file and add it to the remote ssh authorized_keys file:

```
host:~# ssh-copy-id "192.168.200.120 -p 2222"
```

The contents of the key file will be appended to the file ~/.ssh/authorized_keys

Now it should be possible to login remotely, and run commands, without being prompted for a password, for example running the command "date" remotely:

4.4.2 Rsync usage

Until now a couple of default rsync parameters were shown. When using rsync in a forensically manner, file- and directory-attributes like: timestamps and permissions must be preserved at the most. One way to do it is to add a couple of parameters to the rsync-command like in this example:

```
rsync -avzHx —progress —numeric-ids --exclude -exclude . . .
```

Where: "–av" means to archive ("-a") the source directory and all subdirectories to the destination and be verbose ("-v") about what it is doing. This is equivalent to the options: -rlptgoD. This is a quick way of saying you want recursion and want to preserve almost everything.

The parameters:

```
The authenticity of host '[192.168.200.120]:2222 ([192.168.200.120]:2222)' can't be established.
RSA key fingerprint is ed:cc:72:d2:9a:10:d0:c6:aa:17:33:f3:a9:95:72:34.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[192.168.200.120]:2222' (RSA) to the list of known hosts.
SSHDroid
Use 'root' as username
Default password is 'admin'
root@192.168.200.120's password:
Now try logging into the machine, with "ssh '192.168.200.120 -p 2222'", and check in:
  .ssh/authorized_keys
to make sure we haven't added extra keys that you weren't expecting.
host:~# ssh-copy-id "192.168.200.120 -p 2222"
```

```
host:~# ssh -p2222 192.168.200.120 bin/date
SSHDroid
Use 'root' as username
Default password is 'admin'
Mon Jul 22 13:34:44 CEST 2013
```

-z, —compress	- compress file data during the transfer
-H, —hard-links	- This tells rsync to look for hard-linked files in the source and link together the corresponding files on the destination. Without this option, hard-linked files in the source are treated as though they were separate files.
-x, —one-file-system	- This tells rsync to avoid crossing a file system boundary when recursing.
—progress	- This option tells rsync to print information showing the progress of the transfer.
—numeric-ids	- with this option rsync will transfer numeric group and user IDs rather than using user and group names and mapping them at both ends.
—exclude=PATTERN	- exclude the following files
(—include=PATTERN	- include the following files) note: not implemented in script yet

5. Description, Evaluation and Discussion of Results

In chapter 3: Problem Statement we defined the research questions. In this chapter we will mention them one by one and give feedback on each of them.

• Is data extraction from an Android-device possible using Wi-Fi?
in this paper we described an alternative method of fetching data from Android-devices. It showed that it is possible to make a connection to the Android file-system using Wi-Fi and retrieve data successfully.

• If so, will it be a good alternative for using an USB-connection with the adb-toolset? In other words: Can we extract all data as we do as with USB, like extracting deleted data?

Wi-Fi data acquisition could be a valid alternative to an USB cable-connection for transfer of data. In both cases applies that the Android devices must be "*rooted*" to gain to all data. If the device is non-rooted, access will be permitted to several system-folders. In that case only the accessible files and folders will be preserved.

Only on rooted Android devices and with using an USB-connection, it is possible to gain physical access to a memory block-device (for example: a memory-card). The "dd" command can be used with the adb-toolset to acquire a forensic image from an Android memory-device. Only using this last method it is possible to preserve deleted data.

When a Wi-Fi connection is used for transferring data, the data can only logical be extracted from the Android file-system, not physical. Deleted data can never be secured or recovered using logical extraction. Only physical extraction can.

• Which network-protocols for Wi-Fi extraction can be used and which is best?

For wireless communication using Wi-Fi, different protocols can be used. Research in this paper was focused on some of these protocols. The advantages and disadvantages of using these protocols were described.

An answer was given which protocol and application could be used. Narrowing down the options a recommendation was made based on the usage of the rsync and SSH commands and protocols.

• Are there existing applications in the Google Play app-store that could be used for this and if so, which app could be used best??

Several apps were mentioned and tested. Finally the usage of the SSHDroid application, found in the Google Play app-store, was recommended for use in this paper.

• Will the file-system of the Android target-device remain untouched?

It is clear that the file-system of the device during an examination will be changed. In order to communicate with the Android-device using Wi-Fi, an application must be installed onto the device. The app will be installed in the **/data/app** folder of the root file-system of the device. The impact of installing the SSHDroid-app on a Android target-device was described in paragraph 4.2.

• Will the file attributes be preserved during file-transfer?

When data from an Android target-device is retrieved using rsync with proper parameters, the file-attributes of the transferred data, like: timestamps and user-privileges, will be preserved.

• Will the given solution work on all versions of the Android operating system? For testing purposes we used the following Android target-devices:• Samsung Apollo GT-I5800, Android version 2.2 (rooted)

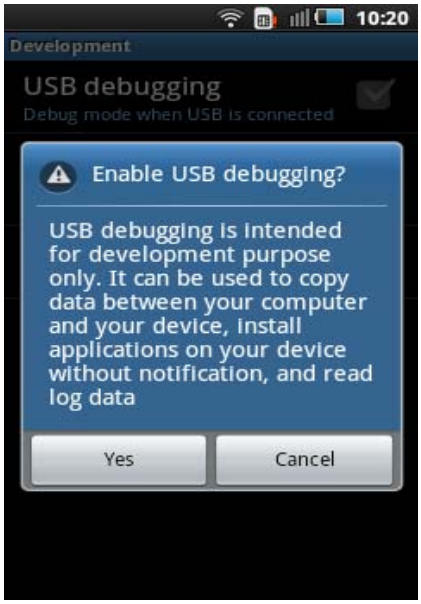
• Archos 101G9 Tablet, Android version 4.0.4 rooted)

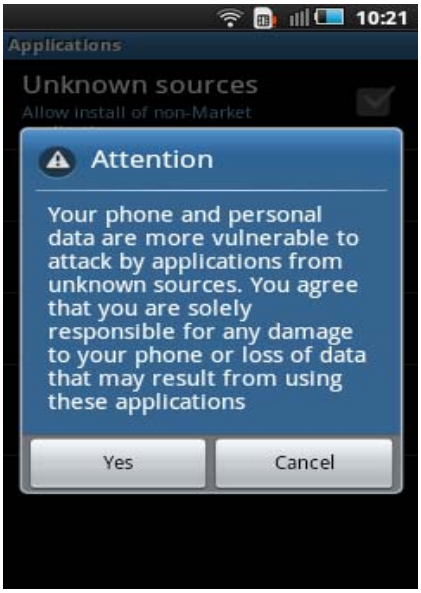
• HTC One X, Android version 4.1.1 (non-rooted)

On all of these Android devices it was possible to install and use the SSHDroid app and connect to the file-system and retrieve data using SSH and rsync successfully.
In this paper we explained how to target an Android-device with a set of Linux shell commands, using SSH and rsync. Additional we showed some commands, which can be scripted using Linux bash scripting. Like mentioned earlier the used script is appended to this paper.

(See: Appendices, Chapter 7)

The screenshots below are taken from the Samsung Apollo GT5800 Android Device. To use a USB cable-connection the option: "*USB-debugging*" must be activated. For data transfers using a Wi-Fi connection this is option is not needed. However installing applications other then from the Google Play app-store requires the option "*Unknown sources*" to be enabled.





Note:
For the screenshots throughout this paper we used the SSHDroid Pro application because in this version no advertisements were shown. However we recommend the usage of the free SSHDroid application, because the Pro version needs to have a valid internet-connection to verify the license. It won't run if the license cannot be verified properly.

Also, when the free version is used it might not run properly if an advertisement blocking application is running on the device. Apps that block advertisements only work in rooted devices. We recommend, where appropriate, to temporarily stop the ad-blocking application from running, using the application settings-menu of the Android device.

6. Conclusion and future work

In this paper we showed how Wi-Fi could be used to acquire data from an Android-device. In order to make a connection using Wi-Fi an application must be installed on the device. This application can be found in the Google Play app-store. Using a Linux-computer and a wireless access-point, data from an Android-device can be retrieved. The best results are obtained when ssh and rsync protocols are combined. Due to our research we recommend the usage of the Android application named: SSHDroid. However, the SSHDroid-package does not contain the rsync executable. Therefore it might be useful in the future to write an SSH-server application to include this rsync-executable.

The appended bash-script (See chapter 7) can be used as an example to show what could be done to automate the process of file-transfer using Linux bash scripting. The script can surely be improved because it is very basic in nature. For example additional error checking and hashing-functionality (md5 / sha1) could be added. To make it platform-independent it could be rewritten using program-languages like Python or Perl. Perhaps a GUI could be added.

Existing tools could also be used to gain access to the Android file-system using SSH. For example tools with a graphical interface (GUI), like: Winscp[49] (Windows) or Cyberduck[50]

The downside of using existing tools is that it will not always be visible what the tools are doing. Will they touch the remote file-system and how? Will they preserve attributes of the retrieved data in a forensic[5] manner?

Using rsync, like shown in this paper, the process of file-transfer can be controlled into detail. When additional rsync-parameters are used and thoroughly tested, it is ensured that the output will be valid, matching the original file-system.

References

[1] Wi-Fi: http://en.wikipedia.org/wiki/Wi-Fi

[2] Dutch Central Bureau of Statistics (CBS), Press release PB12-060: Mobile internet use continues to grow (http://www.cbs.nl/NR/rdonlyres/8D9B5556-6C3C-4CCD-9358-B979D506AF32/0/pb12e060.pdf), The Netherlands: CBS, October 23, 2012.

[3] Google Android mobile platform: http://www.android.com/about/

[4] International Data Corporation (IDC), Press release: Android Marks Fourth Anniversary Since Launch with 75.0% Market Share in Third Quarter, According to IDC (http://www.idc.com/getdoc.jsp?containerId=prUS23771812), Framingham, Massachusetts: IDC, November 1, 2012.

[5] http://en.wikipedia.org/wiki/Forensic

[6] http://www.msab.com/xry/what-is-xry

[7] http://www.cellebrite.com/mobile-forensic-products/ufed-applications.html

[8] Android "*rooting*": http://en.wikipedia.org/wiki/Android_rooting

[9] The "*dd*" command (Unix): http://en.wikipedia.org/wiki/Dd_%28Unix%29

[10] Android Debug Bridge (ADB): http://developer.android.com/tools/help/adb.html

[11] Google Play Android app-store: https://play.google.com/store?hl=nl

[12] SMB protocol: http://en.wikipedia.org/wiki/Server_Message_Block

[13] File Transfer protocol (FTP) protocol: http://tools.ietf.org/html/rfc959

[14] The Secure Shell (SSH) authentication protocol: http://tools.ietf.org/html/rfc4252

[15] Rsync command and protocol: http://rsync.samba.org/features.html, http://en.wikipedia.org/wiki/Rsync

[16] Hypertext Transfer Protocol (HTTP): http://tools.ietf.org/html/rfc1945

[17] Andrew Hoog, *Android Forensics*, 1st Edition. *ISBN*-9781597496513, ReleaseDate: 2011

[18] Android Software Development Kit (SDK): http://developer.android.com/sdk/index.html

[19] V. Vijayan, Android Forensic Capability and Evaluation of Extraction Tools, Master Thesis, Edinburgh Napier University, April 2012

[20] Oxygen Forensic® Suite – http://www.oxygen-forensic.com/

[21] MOBILedit – http://www.mobiledit.com/

[22] AFLogical™ –https://viaforensics.com/resources/tools/android-forensics-tool/

[23] J. Grover, Android Forensics: Automated Data Collection and Reporting from a Mobile Device, Master Thesis, Rochester Institute of Technology, January 2013

[24] P. Andriotis, G. Oikonomou, T. Tryfonas, Forensic Analysis of Wireless Networking Evidence of Android Smartphones WIFS'2012, December, 2-5, 2012, Tenerife, Spain

[25] http://www.pcworld.com/article/2144767/mobiledit-review-put-your-pc-and-phone-in-sync.html

[26] Association Of Chief Police Officers (ACPO): http://www.acpo.police.uk/About/AboutACPO.aspx

[27] Server Ultimate app: https://play.google.com/store/apps/details?id=com.icecoldapps.serversultimate

[28] davfs2: http://savannah.nongnu.org/projects/davfs2

[29] wget command: http://www.gnu.org/software/wget/

[30] lftp command: http://lftp.yar.ru/

[31] SSHDroid app: https://play.google.com/store/apps/details?id=berserker.android.apps.sshdroid&hl=nl

[32] SSH Filesystem (SSHFS): http://en.wikipedia.org/wiki/Sshfs

[33] Secure File Transfer Protocol(SFTP): http://en.wikipedia.org/wiki/SSH_file_transfer_protocol

[34] Secure copy (SCP): http://en.wikipedia.org/wiki/Secure_copy

[35] Android rsync-executable: http://adqmisc.googlecode.com/svn/trunk/androidutils/rsync/rsync-3.0.6-arm-softfloat-linux-gnueabi.gz

[36] SSHelper app: https://play.google.com/store/apps/details?id=com.arachnoid.sshelper

[37] Winmerge: http://winmerge.org/

[38] Setting up the ADT bundle: http://developer.android.com/sdk/installing/bundle.html

[39] Debian: http://www.debian.org/intro/about

[40] Ubuntu: https://help.ubuntu.com/lts/installation-guide/powerpc/what-is-ubuntu.html

[41] Aptitude package manager: http://wiki.debian.org/Aptitude

[42] Service Set Identifier (SSID): http://www.ietf.org/rfc/rfc3770.txt

[43] Dynamic Host Control Protocol (DHCP): http://www.ietf.org/rfc/rfc2131.txt

[44] Apache webserver: http://httpd.apache.org/

[45] Bourne again shell (bash): http://www.gnu.org/software/bash/

[46] Alfred Aho, Peter Weinberger & Brian Kernighan (AWK): http://awk.info/

[47] sshpass: http://sourceforge.net/projects/sshpass/

[48] Ron Rivest, Adi Shamir and Leonard Adleman (RSA) algorithm: http://en.wikipedia.org/wiki/RSA_%28algorithm%29

[49] http://winscp.net/eng/docs/introduction

[50] http://cyberduck.ch/

Appendices

rsync bash script

```
#!/bin/bash
################################################################################
## This script attempts to download (selected) files and folders from an
## (non-rooted/rooted) android device using rsync, WITHOUT the need of ADB.
## For this to work an SSH daemon need to be installed on the android device
##
## Author: Bert Busstra
##
## Date: juli 2013
##
## Note: Script must run as root
##
## Install some packages first:
## aptitude install sshpass rsync awk ssh sshfs
##
##
## Connect android to local wi-fi network
## Enable "install from unknown/untrusted market" in Android settings-menu
##
################################################################################
#
## Test if run as root
#
if [ "$(id -u)" != "0" ]; then
        echo
        echo "This script must be run as root, usage: sudo script.sh " 1>&2
        echo
        exit 1
fi
#
# Check if we are sudo-user or root to define home-dir
#
if [ -z $SUDO_USER ]; then
        home=/root
        else
        home=$(getent passwd $SUDO_USER | cut -d: -f6)
fi
echo $home
#
```

```
## Defined functions
#

function pause(){
   read -p "$*"
}

function check_rdir()
{
        sshpass -p $pass ssh -n -p $pnum -l $user \
        -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no $ip \
        -o LogLevel=quiet "test -d $rdir && echo exists"
}
function check_ssh()
{
        sshpass -p $pass ssh -n -p $pnum -l $user \
        -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no $ip \
        -o LogLevel=quiet "exit"
        if [ $? = 0 ]; then
                echo
                echo "The username and password are correct."
                echo
        else
                echo "The username and/or password are/is incorrect."
                echo "Exiting..."
                exit
                echo
        fi
}


function make_choice ()
{
        echo -n "Type \"yes\" to continue or press Enter key or CTRL+C to abort: "
        echo
        read choice
        if [ "$choice" != "yes" ]; then
                echo "Exiting..."
                echo
                exit
        fi
}
```

```
clear
echo "############################################################"
echo "## This is a script to retrieve files and folders from your android        ##"
echo                    "##                    using                    wifi.
##"
echo "## Install and enable a ssh-server on Android targetdevice first.          ##"
echo "## SSHDroid defaults to SSH port: 2222, user=root, password=admin ##"
echo "############################################################"
echo ""
echo ""

echo -n "Please enter evidence-number: "
read enum
if [ -z "${enum}" ]; then
        echo ""
        echo "Evidence-number may not be empty..."
        echo ""
        echo -n "Please enter an evidence-number: "
        read enum
        if [ -z "${enum}" ]; then
                echo ""
                echo "We need an evidence-number here! Exiting now..."
                echo ""
echo -n "Specify IP address to connect to: "
read ip

#
## Simple validity check of IP
#

if [[ $ip =~ ^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$ ]]; then
        echo "This looks like it's a valid IP address"
        ret=0
        else
echo
                echo "We need a valid IP address here! Exiting now..."
                echo
                exit
       fi
echo -n "Specify SSH port. Press ENTER for default (2222): "
read pnum
if [ -z "${pnum}" ]; then
```

```
    pnum=2222
fi

#
## Define SSH user
#

echo -n "Specify SSH user. Press ENTER for default username (root): "
read user
if [ -z "${user}" ]; then
        user=root
fi

#
## Define SSH password
#

echo -n "Specify SSH password. Press ENTER for default password (admin): "
read pass
if [ -z "${pass}" ]; then
        pass=admin
fi

#
## Checking ssh-connection to host with given username, password and port
#

echo
echo "Checking credentials for user: "$user" with password "$pass" on host: "$ip" ..."
echo
check_ssh #Calling function

#
## Define remote dir
#
echo -n "Specify remote directory: "
read rdir
if [ -z "${rdir}" ]; then
        echo ""
        echo "You need to specify a remote directory..."
```

```
          echo ""
        echo -n "Please enter a remote directory: "
        read rdir
        if [ -z "${rdir}" ]; then
                        echo ""
echo "We need a remote directory! Exiting now..."
                        echo ""
                        exit
        fi
fi
#
## Check if remote-dir exists
#
exist_rdir=$(check_rdir) #Calling function (check_rdir)
 ##echo $exist_rdir  ##test the var
if [ -z "$exist_rdir" ]; then
        echo "Checking if remote directory exists..."
        echo
        echo "The remote directory $rdir does not exist"
        echo "Exiting..."
        echo
        exit
        else
                echo "The remote dir $exist_rdir."
                echo
fi
echo -n "Exclude file, dir or use pattern ie: *.txt. Press Enter to skip: "
read exclude
echo ""
echo "You specified this information:"
echo "Evidence number : $enum"
echo "IP address      : $ip"
echo "SSH portnumber  : $pnum"
echo "SSH user        : $user"
echo "SSH password    : $pass"
echo "Remote directory: $rdir"
echo "Exclusions      : $exclude"
echo
echo "Is the above information correct?"
echo
make_choice  ##calling function
```

```
echo “”
#
##Check size of rdir
#
echo
echo “Checking the size of \”$rdir\”...”
rsize=$(sshpass -p $pass ssh -p $pnum -l $user \
-o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no $ip \
-o LogLevel=quiet “bin/du -ms $rdir 2>/dev/null” |awk ‘{print $1"M”}’)
echo “The size of the remote dir \”$rdir\” is: “$rsize”Bytes”
echo
echo “Do you want me to create a dd file with the size of “$rsize”MBytes?”
echo
make_choice #calling function
echo
#
## Create evidence file and put a file-system on it
#

echo “Creating a dd-file in home directory $home ...”
dd if=/dev/zero of=$home/$enum.dd bs=$rsize count=1
echo
echo “Creating an ext2 filesystem on dd-file: \”$enum.dd\”...”
echo
mkfs.ext2 -F -q $home/$enum.dd
echo “Mounting the created dd-file on mountpoint: $home/tmp/$enum.dd...”

#
## Check mount-point and (re)create it
#

if mount | grep -q $home/tmp/$enum; then
        umount $home/tmp/$enum
        rmdir $home/tmp/$enum
fi
mkdir -p $home/tmp/$enum
loop=$(losetup -f) #check for next available loopdevice

#
## Mount the evidence using loop device
#
```

```
mount $home/$enum.dd $home/tmp/$enum -o loop=$loop
echo

#
## Start the sync
#

echo "Do you want me to sync "$ip$rdir" to $home/tmp/$enum?"
make_choice  ##Calling function
echo
echo
rsync -avgzHx —progress —numeric-ids —exclude "$exclude" —rsh="sshpass -p $pass ssh \
-p $pnum -l $user -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no \
-o LogLevel=quiet" $ip:/$rdir $home/tmp/$enum
echo
echo "Syncing is Done!"
make_choice
echo
echo "Removing lost+found dir..."
rmdir $home/tmp/$enum/lost+found
echo "Unmounting $home/tmp/$enum..."
umount $home/tmp/$enum
rmdir $home/tmp/$enum
echo "Done!"
echo

echo
echo "Now import $enum.dd in Encase or FTK and examine the contents."
ls $home/$enum.dd
echo
echo
```