令和 2 年度 東邦大学理学部情報科学科 卒業研究

Android アプリケーションにおける IPv4 アドレスのハードコーディングに関する調査と分析

学籍番号　5517044

小林 裕

金岡研究室

目 次

1　はじめに

インターネット上に接続された機器同士がデータをやりとりする際、ネットワーク上で通信相手を間違わないよう、それぞれを唯一に特定するために割り当てられいてる識別子 Internet Protocol（以後 IP）アドレスが存在する。

現在 IP Version 4（以後 IPv4）が広く利用されているが、インターネットの利用者が増え続けたことにより IPv4 アドレスを枯渇していることが問題になっている。IPv4 アドレスは 32 ビットのビット長を持ち、約 43 億個のアドレスを表現することができる。この数はインターネット黎明期の頃は十分な大きさと考えられていたが、インターネットに接続される機器が増え続けたことにより、割り当ての限界を迎えつつある。そのためアドレス空間を IPv4 から大幅に拡張するなど対策が取られた IP Version6（以後 IPv6）の導入がされ、IPv4 アドレスから IPv6 アドレスへの移行が進んでいくと予想される。しかしながら、IPv4 アドレスと IPv6 アドレスの間には互換性がなく、相互通信を行うことが出来ないため IPv4 アドレスと IPv6 アドレスの混在環境が続き、いずれ混在する環境から IPv6 だけの環境（IPv6 Single Stack 環境）へと移行していく。

モバイル環境においては、Apple の iphone や iPad に提供されるアプリケーション (以後アプリ) では IPv6 に対応することが必須とされている。一方、Android OS 上ではそういった要件は示されていない。すでに IPv6 Single Stack 環境からのアクセスにおいて、正しく動作しない Android アプリが複数発見されている。

過去 2012 年には IETF にて IPv6 Only Network についての情報共有がなされ、2018 年には北口らによって OS 各種の IPv6 対応状況調査が行われ、その後、加茂らによって Android に焦点を当て v6 のみの環境で OS が稼働するか、マーケットが対応しているか、ANdroid アプリが動くかという複数の視点で行われた。その結果、古い OS での未対応状況、マーケットの未対応状況、アプリの大部分が対応していないことが分かったが、Android アプリの未対応である原因についてまでは書かれていなかった。

IPv4 と IPv6 の混在環境や IPv6 Single Stack 環境で動くように OS などは対応されているが、アプリ側で特定の IP バージョンを指定してしまい IPv6 Single Stack 環境で動かないケースが考えられる。仮に、IP アドレスの指定がされていたら、脆弱性として利用者が攻撃者によって Man in the middle（中間者攻撃）されてしまう問題が考えられる。

そこで本研究では、Android アプリを対象にアプリ内に IPv4 アドレスの指定がされているのか調査と分析を行なっていく

中間発表の段階では、テスト環境として 2520 個のアプリを対象に、IPv4 アドレスの指定が行われいるか調査したところ、2520 個中 241 個のアプリに IPv4 アドレスの記載があることが分かった。更なる調査として記載があった IPv4 アドレスの出現数を数えた結果、一番出現頻度が多かった IPv4 アドレスは”127.0.0.1”のループバックアドレスであった。

本論文ではさらなる調査結果を元に、脅威や原因の考察をした。

本論ぶの構成は以下の通りである。第 2 章で本研究に関する技術についての解説を行い、第 3 章では関連研究について説明を行う。第 4 章では本研究の調査内容と手法を述べ、第 5 章で調査結果を述べる。第 6 章で考察し、最後に第 7 章で本研究のまとめを述べる。

2　前提知識

2.1　Android

Android とは Google 社によって開発された携帯汎用オペレーティングシステムである。スマートフォンやタブレット、時計、テレビなどに搭載されている。

2.1.1　アプリケーション配布マーケット

アプリケーション配布マーケット（以後アプリマーケット）とはアプリを提供するマーケットのことで、公式アプリマーケットは Google が提供する Google Play や Apple が提供する App Store がある。また、Android アプリに関してはサードパーティ製のアプリマーケットも存在する。マーケットからソフトウェアを入手しインストールするために、マーケットへのアクセス専用アプリが提供されている。Android では Google Play アプリが提供されており、多くの端末においてあらかじめインストールされている。

2.1.2　アプリ開発

Android アプリを開発する際の言語は、Java、Kotlin、C/C++で書くことが可能であり、開発環境として Android Studio が公式に提供されている。また、ios アプリを開発する際は、Swift、Objective-C で書くことが可能である。開発環境として Xcode が公式に提供されている。

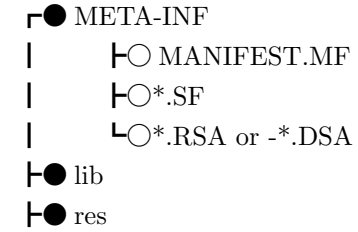
2.1.3　ライブラリ

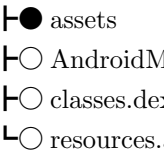
ライブラリとは、ある特定の機能をもったプログラムを他のプログラムから呼び出して利用できるように細分化し、それらをまとめて一つにしたものがライブラリと言う。また、それ単体ではプログラムとして動作させることはできないもの。Android 開発においては、Google 社から提供されているライブラリやサードパーティライブラリが存在する。

2.1.4　APK

APK とは、Google 社によって開発された Android 専用ソフトウェアパッケージのことである。APK の入手方法は APK ストアからダウンロードする方法や、単体で公開されている APK ファイルをダウンロードする方法などが存在する。一般に APK の拡張子は”.apk”であり、内部の構造は zip ファイルと同様である。

APK ファイルに対して zip ファイルと同様の解凍処理を行、得られる内部フォルダは以下のような後世になっている。（●はフォルダ、○はファイルを表す。）





2.1.5 APK ストア

APK ストアとは、開発者の作成した Android アプリの配信を代行するサービス、及びそれを行っている Web サイトのことである。Android の公式 APK ストアは Google Play 1 つのみであり、非公式の APK ストアは数多く存在する。

2.1.6 Smali ファイル

Smali ファイルとは、APK に含まれるソースコードを Dalvik バイトコードとして表記したものであり、その情報は classes.dex に含まれている。Dalvik バイトコードとは、Android における中間言語である。Apktool 等を用いて APK より取得できる Smali ファイルは、Dalvik バイトコードで記述されている。

2.1.7 APK から Smali ファイルの変換

APK から Smali ファイルを展開する方法は、Apktool を使う方法がある。Apktool は APK に含まれる全てのリソース・ソースコード・XML を展開する。import されたパッケージがディレクトリとして存在し、これにより使用ライブラリ

2.2 IP

IP とは、複数の通信ネットワークを相互に接続し、データを中継・伝送して一つの大きなネットワークにすることができる通信規約である。また、IP においてパケットを送受信する機器を判別するための番号を IP アドレスという。

現在 IP は IPv4 と IPv6 が混在した状態にある。

IPv4 アドレスは 32 ビットで構成される世界で重複のないアドレスで、8 ビット毎 4 つにピリオドで区切った数値列を 10 進数で表され、例えば”192.168.255.255”のように表記されます。一方、IPv6 アドレスは 128 ビットを 16 ビット毎 8 つにコロンで区切った数値列を 16 進数で表され、例えば”2001:0DB8:0000:0000:0008:0800:200C:417A”のように表記されます。IPv6 アドレス表記では連続する 0 は省略可能であるが、各フィールドに少なくとも 1 つの数値を含ませなければならない。また、16 ビットの 0 が複数連続している場合は”::”と何処でも省略することができるが、省略できるのは一カ所のみ。例えば”2001:0DB8:0000:0000:0008:0800:200C:417A”という IPv6 アドレスは”2001:DB8::8:800:200C:417A”と省略する。IPv4 アドレスではネットワーク部とホスト部に別れて構成される。一方、IPv6 アドレスはサブネットプレフィックスとインターフェース ID に別れて構成される。IPv6 のサブネットプレフィックスは、IPv4 のネットワーク部に該当し、IPv6 のインターフェース ID は、IPv4 のホスト部に該当する。サブネットフィックスは 64 ビットが標準的な値であるため、インターフェース ID も 64 ビットであることが一般的である。IPv4 アドレスと IPv6 アドレスの大きな違いとしては、アドレスビット長が 32 ビットから 128 ビットと大幅に拡張

している事が挙げられる。その他に、IPv4 ではヘッダチェックサムがあるが、TCP やアプリケーション側でエラーチェックを行なっているため IP レベルのエラーチェックは不要だとして、IPv6 ではヘッダチェックサムが削除された。また、品質制御利用のためのフローラベルが追加されるなどの違いがある。IPv4 での通信の種類は、ユニキャスト、ブロードキャスト、マルチキャストの 3 種類でしたが、IPv6 ではブロードキャストがマルチキャストへと統合されました。IPv4 でブロードキャストで行われていたものは、IPv6 ではマルチキャストを利用してい行われている。一方、IPv6 での通信の種類は、ユニキャスト、マルチキャスト、エニーキャストの 3 種類である。

2.2.1 IPv4 アドレス枯渇問題

現在 IPv4 が広く利用されているが、インターネットの利用者が増えたことにより IPv4 アドレスが枯渇していることが問題になっている。IPv4 アドレスは 32 ビットのビット長を持ち、約 43 億個のアドレスを表現することができる。この数はインターネット黎明期の頃は十分な大きさと考えられていたが、インターネットに接続される機器が増え続けたことにより、割り当ての限界を迎えつつある。そのためアドレス空間を IPv4 から大幅に拡張するなど対策のとられた IPv6 が導入され、IPv4 から IPv6 への移行が進んでいくと予想される。しかしながら、IPv4 と IPv6 の間には互換性がなく、相互通信を行うことができないため IPv4 と IPv6 の混在環境が続き、いずれ混在する環境から IPv6 だけの環境 (IPv6 Single Stack) へと移行していく。

2.2.2 IPv6 Single Stack

IPv6 Single Stack とは IPv6 のみで動作させる仕組みである。IPv6 Single Stack 環境構築には IPv6 over IPv4 トンネリングを利用する。IPIP トンネリングとは IPv6 パケット全体を一つのデータとして扱い、そのデータの先頭に IPv4 ヘッダを付加しカプセル化することで IPv6 ネットワークから IPv4 ネットワークをトンネリングし、IPv6 通信を実現させる通信方法である。

2.2.3 IPv4/IPv6 共存技術

IPv4/IPv6 共存環境には Dual Stack 環境や NAT64/DNS64 が存在する。Dual Stack は現在一般的に用いられている IPv4/IPv6 共存技術であり、単一の危機において IPv4 と IPv6 を同時に動作させる事ができる。IPv4 対応危機と通信を行う際には IPv4 を使用し、IPv6 対応機器と通信を行う際には IPv6 を使用する。一方 NAT64/DNS64 は IPv6 ネットワークから IPv4 ネットワークに接続する技術である。Dual Stack が IPv4 アドレスと IPv6 アドレスの両方を必要と s るのに対し、NAT64/DNS64 は IPv6 アドレスのみを与え利用される技術である。IPv4 アドレス枯渇問題対策として有効である。

Dual Stack 環境は、RA を動作させているルータマシンで DHCP サーバのソフトウェアをインストールし起動させプライベート IP アドレス等必要な設定を行、NAT で IPv4 ネット w ー国アクセスする。NAT とはプライベート IPv4 アドレスとグローバル IPv4 アドレスの間でアドレス変換をする技術である。

3　関連研究

3.1　北口らによる調査

北口らは、IPv6 対応状況の研究として OS の観点から調査を行なった。そこでは各種 OS における IPv6 実装状況の検証がされ、さらにネットワーク運用管理に与える影響について考察を行なっていた。表 2 は各 OS における SLAAC で用いるインターフェース ID の生成手法、RA における RDNSS オプションの実装、DHCPv6 の実装及び IPv6 Single Stack 環境での検証結果を示している。ただし Microsoft 社のインターフェース ID の生成方法は確認できていない。

Android OS については、バージョン 4、５、６、７を対処に調査が行われており、全てのバージョンで DHCPv6 に未対応であることと、バージョン 4 では IPv6 Single Stack 環境では動作しないことが示されている。また運用管理の課題としては、セキュリティインシデント時のトレーサビリティ困難性が挙げられている。

3.2　Durdagi らによる調査

Durdagi らは IPv6 と IPv4 のセキュリティと脅威の比較調査を行なった。しかし、技術使用の視点からの指摘はされているが、運用の視点や混在環境での脅威といった視点では議論は行われいない。

3.3　Rosil らによる調査

Rosli らは IPv6 環境の脅威に焦点をあて、セキュリティ対策の提案をした。しかし、技術使用の視点からの指摘はされているが、運用の視点や混在環境での脅威といった視点では議論は行われていない。

3.4　加茂らよる調査

加茂らは Android に焦点を当て IPv6 Single Stack 環境で OS が稼働するか、マーケットが対応しているか、Android アプリが動くか複数の視点で行われた。しかし、Android アプリの未対応である原因についてまでは書かれていなかった。

4　調査及び分析内容・手法

4.1　APK から Smali ファイルの変換プログラム

Smali ファイルの取得は apktool を用いて apk ファイルの変換を行う。APK から Smali ファイルの変換を行うために以下のコマンドを行う。

Listing 1: APK から Smali ファイルの変換のコマンド

```
1         apktool d $APK_FILE  -o /ssd1/$name/$OUTPUT_FOLDER
```

今回は、多くの APK ファイルを調査する必要であるため、以上のコマンドを複数回実行する必要がある。そのため、以上のコマンドをシェルスクリプトにまとめて実行することを推奨する。シェルスクリプトとは、複数の UNIX コマンドを一つのファイルに会場区切りで記述したものである。シェルスクリプトを作成し実行を行うと、ファイルに記載されているコマンドが上から順に実行される。シェルスクリプトの作成の仕方は様々だが、シェルの一つである「sh」を例に、APK から Smali ファイルの変換を行うためのシェルスクリプトを以下に記載する。

Listing 2: APK から Smali ファイルの変換を行うシェルスクリプト

```
1 #!/bin/sh
2 name = "フォルダ"
3 APK_FILES = $(find /media/$(name)/*.apk)
4 for APK_FILE in $APK_FILES
5 do
6         echo $APK_FILE
7         OUTPUT_FOLDER=$(basename $(APK_FILE).apk)
8         apktool d $APK_FILE -o /ssd1/$name/$OUTPUT_FOLDER
9 done
```

シェルスクリプトを実行するためには以下のコマンドを実行する。

Listing 3: シェルスクリプトの実行方法

```
1 bash ./Smali1.sh
```

以上で APK を Smali ファイルに変換を行うことができる。

4.2　Smali ファイル内から IPv4 アドレスの探索プログラム

Smali ファイル内から IPv4 アドレスの探索を行うためのスクリプトを Python で作成する。

Listing 4: Smali ファイル内から IPv4 アドレスの探索を行うための Python スクリプト

```
1 import glob
2 import os
3 import pprint #結果を見やすくするためのモジュール
4 import sys
5 import pathlib
6 import re #正規表現のためのモジュール
7 import csv
8 import sys,os
```

```
9
10 def Find_IPv4(apk_file):
11         args = '(([1-9]?[0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.)
               {3}([1-9]?[0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])'
12         count =0
13         result_path = "/ssd1/IP_result/IP_File/"+apk_file+".csv"
14         print("通過します")
15         path = "/ssd1/"+apk_file+"/*/smali*/**/*.smali"
16         result = pathlib.Path(result_path)
17         result.touch()
18         smail_files = glob.glob(path,recursive = True)
19         pprint.pprint(args)
20         for file_path in smail_files:
21                 with open(file_path,encoding="utf-8_sig") as f:
22                         lines = f.readlines()
23                         txtcount=0
24                         for contents in lines:
25                                 txtcount +=1
26                                 match2 = re.search(r'\s'+args+r'\s',contents)
27                                 match3 = re.search(r'^'+args+r'$',contents)
28                                 match4 = re.search(r'\"'+args+r'\"',contents)
29                                 match5 = re.search(r"\s"+args+r"$",contents)
30                                 match6 = re.search(r"^"+args+r"\s",contents)
31                                 match7 = re.search(r"'"+args+r"'",contents)
32                                 contents = f.readline()
33                                 if match2 or match3 or match4 or match5 or match6
                                        or match7:
34                                    pprint.pprint(file_path)
35                                    pan = [file_path,str(txtcount)]
36                                    count += 1
37                                    with open(result_path,"a", newline='') as
                                           n:
38                                       writer = csv.writer(n,
                                             lineterminator='\n')
39                                       writer.writerow(pan)
40                                    n.close
41                                 else:
42                                         continue
43                 f.close
44         print("Smali内のIPv4探索を終了します")
45         return(result_path)
```

4.3 AndroidManifest 内から applicationId を取得するプログラム

AndroidManifest 内から applicationID を取得する Python スクリプトを作成する。

```
1 import csv
2 import re
3 from collections import Counter
```

```
import xml.etree.ElementTree as ET
def AndroidManifest(txt,file):
        file_name_list=[]
        Duplicate_File_Count=[]
        with open(txt,encoding="utf-8") as f:
                path_list = csv.reader(f)
                for path in path_list:
                        file_name = path[0].split('/smali')[0]
                        file_name_list.append(file_name)
                print(file_name_list)
        f.close
        Duplicate = Counter(file_name_list)
        for File in Duplicate.most_common():
                Duplicate_File_Count.append(File)
        file_name_list = "/ssd1/IP_result/Edit_Name_File/"+file+".csv"
        with open(file_name_list, 'w',newline='') as f:
                writer = csv.writer(f)
                writer.writerows(Duplicate_File_Count)
        result_path = "/ssd1/csv_result/PackageName/"+file+".csv"
        list=[]
        with open(file_name_list,encoding="utf-8") as f:
                name_list = csv.reader(f)
                for name in name_list:
                print(name[0])
                file_path = name[0]+"/AndroidManifest.xml"
                tree = ET.parse(file_path)
                root = tree.getroot()
                print(root.attrib["package"])
                list.append([name[0],root.attrib["package"]])
        with open(result_path,"w",newline="") as f:
                writer = csv.writer(f)
                writer.writerows(list)
        return
```

4.4 グローバル IP アドレスとプライベート IP アドレスの割合調査プログラム

グローバル IP アドレスとプライベート IP アドレスの割合調査する Python スクリプトを作成する。

```
import csv
import pprint
import re
def Distinction(txt,IP_Adress):
        ip_list =[]
        ip_list2= []
        #プライベートIPアドレス
        file_list=[]
        #グローバルIPアドレス
        file_list2=[]
```

```
11          PrivateIPv4 =0
12          GlobalIPv4 = 0
13          args = '(^127\.)|(^169\.254\.)|(^10\.)|(^172\.1[6-9]\.)
                |(^172\.2[0-9]\.)|(^172\.3[0-1]\.)|(^192\.168\.)'
14          with open(IP_Adress,encoding="utf-8") as n:
15                  ip_adress = csv.reader(n)
16                  for ip in ip_adress:
17                          ip_list.append([ip[0]])
18          n.close
19          with open(txt, encoding="utf-8") as f:
20                  reader = csv.reader(f)
21                  for row in reader:
22                          file_name = row[0].split('/smali')[0]
23                          a =row[2].strip('"')
24                          ip_name = a.strip()
25                          ip_list2.append([file_name,ip_name])
26          f.close
27          for ip in ip_list:
28                  for ans in ip_list2:
29                          #同じIp_Adressならば通過できる
30                          if ip[0] == ans[1]:
31                                  #print(file_list)
32                                  print(ans[0])
33                                  match = re.search(args,ans[1])
34                                  if match:
35                                          if ans[0] not in file_list:
36                                                  print("プライベートIPアドレスあった
                                                        よ")
37                                                  PrivateIPv4+=1
38                                                  file_list.append(ans[0])
39                                  else:
40                                          if ans[0] not in file_list2:
41                                                  print("グローバルIPアドレス")
42                                                  GlobalIPv4+=1
43                                                  file_list2.append(ans[0])
44          return
```

4.5 Android アプリ内に IP アドレスのハードコーディング調査

Android アプリ内に IP アドレスのハードコーディング調査のため、複数の APK を対象に Smali 化を行い Smali ファイル内に IP アドレスの記載があるのか調査した。

4.6 グローバル IP アドレスとプライベート IP アドレスの割合調査

IP アドレスにはグローバル IP アドレスとプライベート IP アドレスの 2 つがある。グローバル IP アドレスはインターネットを利用するときに割り振られる IP アドレスでセキュリティ上の問題でプロバイダ側が時間制や一定サイクルで変更を行なっています。一方、プライベート IP アドレ

スはインターネットのようなグローバルなものではなく、社内 LAN など小規模なネットワークで使用される IP アドレスです。特に、グローバル IP アドレスのハードコーディングがある場合攻撃者によって悪用されるリスクが高まると考えグローバル IP アドレスとプライベート IP アドレスの割合を調査した。

4.7 Android アプリの種類調査

どのようなカテゴリーのアプリに IP アドレスのハードコーディングが多いのか調査を行なった。

5 結果

5.1 Android アプリ内に IP アドレスのハードコーディングがされているのか調査結果

調査の結果、調査を行なったアプリ数 109251 個の中 45305 個のアプリに IPv4 アドレスの記載があることが分かった。以下の表に、出現数の多い IP アドレス別の結果をまとめた。

表 1: 最も多い IP アドレス 1〜20

IP アドレス	アプリ数
10.0.0.172	35151
127.0.0.1	29165
10.0.0.200	20794
0.0.0.0	14177
117.97.87.6	4503
127.0.0.255	4202
2.5.29.15	3718
2.5.4.3	3197
2.5.4.11	3190
2.5.4.6	3188
2.5.4.10	3187
2.5.4.8	3186
2.5.4.7	3183
2.5.29.37	3065
10.0.2.2	2878
2.5.29.19	2876
1.4.1.19	2762
2.0.61.0	2758
2.5.4.4	2360
2.5.4.42	2354

5.2 グローバル IP アドレスとプライベート IP アドレスの記載割合調査結果

調査したアプリ数とグローバル IP アドレスの記載があるアプリ数、プライベート IP アドレスの記載があるアプリ数との記載割合と IP アドレスの記載があるアプリとグローバル IP アドレスがあるアプリ数、プライベート IP アドレスの記載があるアプリ数との記載割合を次のような表 2〜3 でまとめた。

表 2: IP アドレスの記載率

調査したアプリ数	グローバル IP アドレス記載があるアプリ数	記載率
109251	29074	26.6%
調査したアプリ数	プライベート IP アドレス記載があるアプリ数	記載率
109251	35151	29.7%

表 3: IP アドレスの記載率

IP アドレスの記載のあるアプリ数	グローバル IP アドレス記載があるアプリ数	割合率
10.0.0.172	29074	26.6%
IP アドレスの記載のあるアプリ数	プライベート IP アドレス記載があるアプリ数	割合率
10.0.0.172	35151	26.6%

6 考察

6.1 Android アプリ内に IP アドレスのハードコーディング調査の分析

6.2 IP アドレスのハードコーディングにおける脅威

7　まとめ

参考文献

[1] だれだれ, ”https://www.nic.ad.jp/ja/newsletter/No32/090.html”, 年度

[2] だれだれ, ”文献 2”, 年度

[3] だれだれ, ”文献 3”, 年度

[4] だれだれ, ”文献 4”, 年度

[5] だれだれ, ”文献 5”, 年度

[6] だれだれ, ”文献 6”, 年度

[7] だれだれ, ”文献 7”, 年度

[8] だれだれ, ”文献 8”, 年度

[9] だれだれ, ”文献 9”, 年度

[10] だれだれ, ”文献 10”, 年度