Geographic Analysis of Traffic Flow for Google Play Store Applications Final Report

Erick Hernandez-Ascencio
Stanford University

Daniel Garcia Lopez
Stanford University

Modupe Esther Akintan
Stanford University

Abstract

In this project, we captured and studied over 30,000 unique HTTP requests sent out to servers while using the top 100 Android apps. We narrowed our list down to 77 apps in order to focus our study on apps that contain user-generated data. The goal of this project is to determine how common it is for applications to send requests to foreign servers (i.e. non-American servers) and to analyze exactly what kind of data is being exchanged with these servers. We capturing and analyzing HTTP requests sent by establishing a man in the middle connection for the applications we tested, even when using applications that enforce certificate pinning. Limitations in our methodology prevented us from collecting data on all 77 apps, so we collected request data from 57 apps. After developing a script that analyzes these results, we found that most foreign requests being sent out were to servers owned by advertisement companies. Most of these requests were directed to European countries such as Sweden and the Netherlands, with a small number of requests sent to Eastern countries like Singapore, China, and India. In this paper we dive into the contents of these international requests and investigate some of the companies involved in the associated domains.

1 Introduction

This project is inspired by the 2020 discovery that Zoom was sending TLS AES-128 keys to a server in China [1]. This is significant because anyone with these keys has the potential to decrypt all of the user's private meeting data, including recordings. Soon after being brought to the public's attention, Zoom quickly rectified this data export and ensured their users that this was a geo-fencing error and any subsequent connections will be kept domestic. Given the high profile users that use Zoom including governmental agencies, having their AES keys in China raised a discussion on how this data export can weaken user privacy under the threat of nation state adversaries.

With our project we wanted to explore the question of how frequent it is for applications to send American-based user data to a server in a foreign country. We wanted to research a space that is commonly utilized by the general population while using an operating system that can be accessibly simulated and studied in a sandbox. With these considerations in mind, we decided to focus our research on the top 100 most downloaded apps on the Google play store. Given that Android share 44% of the American mobile phone market and 70% of the global market, our research is a fair representation of the mobile phone user community [2].

The rest of the paper is structured as follows: in Section 2 we discuss the related work that has been done on Android application privacy research and contextualize our work within the existing literature. In Section 3, we give more context to the Android ecosystem and the Google Play application vetting process. In Section 4, we discuss our methodology for testing the applications and analyzing our data. In Section 5, we discuss our statsitical results and provide a deeper analysis in Section 6. In Section 7, we discuss some of the limitations and issues we ran into with our methodology and in Section 8 we conclude.

2 Related Work and Events

Drawing inspiration from the Zoom incident in 2020, we conducted an extensive amount of research in the field of network traffic study within the mobile phone space. There have been quite a few number of papers published in the past that attempted to capture network packets with the goal of studying Android data privacy. Shen et. al's work used app telemetry, DNS lookup databases, and geolocation databases to track the endpoint of all requests sent by the Android device as it scans through a number of different applications. [3] This paper's methodology gave us inspiration for developing our own methodology for collecting network traffic data. The paper found that 87% of surveyed applications were sending private data (e.g. SIM card info, OS version, geolocation info, call logs, etc.) to five distinct domains. The paper concludes that most of these flows' endpoints were within the United States, with China absorbing about 7% of these. This result

was not surprising to us: with the current climate surrounding data privacy in the tech industry, it isn't anything new to learn how much data is being collected by companies. However, the 7% figure for China showed us that there is potential for researching how foreign endpoints are using American data flows.

In another study by Liu et. al. reported in their 2018 study of 7,601 popular Google Play applications that 42% of them are collecting phone sensor data on top of basic application metadeta. This data includes accelerometer data, magnetic field data, and gyroscope data. [4] A majority of this data collection is being done without user knowledge, which raises discussion about how many of these applications are sending this data to foreign servers. Unfortunately, this particular study did not identify the culprit applications or how popular they were. However, this provided good motivation to further analyze what data is being sent to foreign servers.

3 Android Ecosystem

The Android ecosystem is developed by Google as an open source mobile platform based on Linux. Recently, it has also been used in different platforms like TVs and Tablets, but Androids core functionality and largest user bases is on the mobile platform. Currently, there are estimated to be more than 2.5 billion Android devices in the world, with the largest share being the mobile platform market [5]. The Android OS provides a lot more user freedom than other mobile platforms (e.g. iOS) as it allows users to run commands through a shell, allows a root user to inject their own system trusted certificate authorities, and also gives the user more control over their device. This made it the perfect fit for our research.

To add third party applications to an Android device, applications can be installed through the internet or the Google Play Store. Google claims that it has an automated way to scan the applications posted on the Google Play store for malware or suspicious behavior, but has still made mistakes such as ExpensiveWall in 2017. ExpensiveWall was a malware found across multiple Android apps that totalled to over 5.9 million downloads. The malware would connect to command and control servers and send SMS messages to register users for services without their knowledge, undermining Googles credibility in securely scanning Google Play applications [6].

We interfaced with the Android OS using a simulated version that is run on Android Studio. This simulated device gave us all the controls we would have over a real phone: device orientation, volume control, phone call and messaging capabilities, a complete filesystem, etc.

We decided to conduct our research on Android 12.0 because it is the most modern OS version offered by Android Studio and it holds a significant share of the global Android user base. The statistics on Android user base in Table 1 is pulled from Android Studio.

Table 1: Approximate Share of Users Using Various Android Versions

Android Version	Percentage of Users Running
Android 4.1	0.2
Android 4.2	0.1
Android 4.3	0.1
Android 4.4	0.9
Android 5.0	0.4
Android 5.1	2.2
Android 6.0	3.5
Android 7.0	3.5
Android 7.1	2.2
Android 8.0	3.0
Android 8.1	7.9
Android 9.0	14.5
Android 10.	22.3
Android 11.	27.0
Android 12.	13.5

4 Methodology

For our experimental setup, we set up an Android Emulator that was packaged with Google's Android Studio SDK version 2023.3.1. Using the Android Device Manager we emulated a Pixel 6 Android device running Android 12.0. We decided to go with this pairing because we wanted to conduct our testing on a modern Android device that is popular amongst American consumers.

For testing, we were unable to use the Google Play store to download applications because Google does not allow for their Google Play interface to be used on rooted Android emulators, only on non-rooted Google Play builds. Since we would not have been able to conduct our experiments without root privileges (root privledges are necessary to inject our certificate authority into the system), we had to fetch the APKs from third-party sites. Most of our APKs were downloaded from APKPure.com, APKCombo.com, or APKMirror.com. In general, these sites had the latest APK version available and there were no noticeable changes in in-app behavior.

Once we had the application downloaded, we used the application for 10-15 minutes each, trying to model typical behavior. This includes but is not limited to making accounts, posting photos, interacting with other users, and adding things to our cart on shopping apps.

To snoop on network traffic, we used a tool called HTTP Toolkit [7]. HTTP Toolkit connects to the Android Emulator and injects its own certificate authority into the system list of trusted CAs. Afterward, it establishes a VPN on the device to a local MITM server, which would then forward the re-

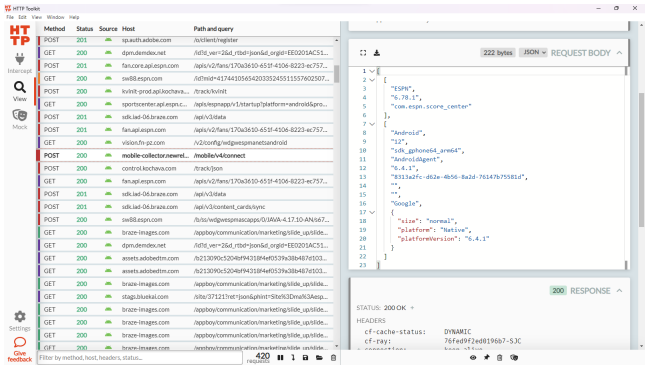

Figure 1: A screenshot of the HTTP Toolkit dashboard for capturing traffic sent by the ESPN application. The interface allows us to see the request headers (e.g. domain, along with any data sent by the request via the body.

```
def analyzeRequests():
    for file in scandir(HAR_FILE_DIR):
        parsed_file = HarFileParser(file)
        data = None
        for entry in parsed_file:
            ip = resolver.resolve(entry.domain)
            geoip_res = geolite2.lookup(ip)
            if geoip_res.country != 'US':
                data = updateData(geoip_res)
        writeDataToTXT(data)
```

Figure 2: Pseudo code for our Python script used to analyse all of our captured requests.

quests to the intended recipient. The HTTP Toolkit dashboard allowed us to view each request and response sent out from the application while ignoring any other traffic that could be sent out from the device (e.g. OS network traffic or background requests made by another application). See Figure 1 for an example of what traffic appeared on the HTTP Toolkit dashboard while using an application.

This proved to work for the majority of cases, but in some cases, this approach fell short because of certificate pinning. Certificate pinning allows for applications to only refer to their own list of "pinned certificates" when authorizing requests, thus nullifying our custom CA certificate. Luckily, we were able to get past this using a tool called FRIDA [8].

FRIDA is a dynamic instrumentation toolkit, which dynamically catches certificate pinning errors and rewrites the application to bypass the certificate pinning. To do so, we had to install and run a FRIDA server on our emulators as root and run the pinning application through the FRIDA instrument. When the FRIDA instrument catches a certificate pinning error, it uses a provided script to bypass the pinning depending on the application's pinning techniques. The script is a JavaScript file that dynamically modifies the application's behavior. An accompanying script that was provided by FRIDA developers allowed us to use FRIDA to dynamically catch most of the errors thrown on certificate pinning checks and nullify them. This enabled us to capture all of the requests that previously failed from applications that use popular HTTPS libraries such as OkHTTP and SSLContext.

For every application that we were able to successfully test, we used HTTP Toolkit to save an HTTP Archive file (.HAR file) that contained all the HTTP traffic for the application.

Once we aggregated all the files, we wrote a python script that crawled through all the captured traffic to fetch each request's destination domain. Using the domain, we ran a DNS query using the dnspython [9] library. This library resolves a hostname to a list of IP addresses that are associated with the hostname according to DNS records. We traversed the list of IP addresses and ran them through the MaxMind GeoIP database provided via the python-geoip library. If any of the IPs yielded an international result, we would write the IP address, server location, and the full URL associated with the request to a txt corresponding to each application we tested. At the end of the file we would also print out a dictionary that mapped each of the foreign countries to the number of requests that they received. This allowed us to index the international request via the HTTP toolkit dashboard and read its body data in order to discern what kind of information the foreign server is receiving. Through this method, we were able to also gather statistics on what percentage of the requests went internationally and do further analysis on the content of each reqeusts via the HTTP Toolkit dashboard. See figure 2 for a code listing of our pseudo code.

5 Results

Ultimately, our testing yielded 33688 requests and corresponding responses across all of the applications that we tested. Of the 57 applications, we were able to test, we found that the majority sent at least one request abroad, with 43 applications sending at least one request abroad. This figure means that 75% of the top applications we tested are sending American-based requests to foreign IP addresses. These 43 applications generated 4542 international requests. The remaining 27,798 requests we analyzed were domestic. There were a good number of applications that are responsible for large chunks of these international requests. Most of these applications, like Fox Sports and Spotify, are ad-heavy so many of these requests are GET requests for advertisement data. See Table 2 for the top 5 applications with foreign outbound requests.

Of the international requests analyzed, almost all of them went to just a handful of countries: Sweden, Canada, Israel,

Table 2: Applications with most outbound foreign requests.

Application	Number of international requests
Fox Sports	649
Spotify	406
Alibaba	270
Bath & Bodyworks	253
Pinterest	206

UK, and the Netherlands, with Sweden receiving 83% of the international requests. See Table 3 for the percentages of requests that went to different countries.

In the following section, we will analyze the results more closely looking at the trends in what data were sent to each country and do a case studies on the applications that displayed interesting trends.

Table 3: Number of international requests per country

Country	Number of international requests
Sweden	3742
Canada	398
Israel	198
Netherlands	92
Singapore	68
China	33
United Kingdom	17
Germany	4
Japan	3
India	2
Ireland	1

6 Analysis

6.1 Sweden & Netherlands

Sweden & Netherlands accounted for almost all of our international requests. Unlike any of the following countries, requests to these countries were unique in two ways.

Firstly, we saw regular application traffic going to Sweden & the Nethelands, unlike any of the other nations which almost entirely contained advertising metrics.

Secondly, they were the only countries that would vary per DNS resulution. For many domains, we would resolve the the domain and get a list of IPs that would resolve to either America or Sweden & Netherlands. Once we realized that some of the domains were varying depending on the IP address we used, we modified our script to scan all IP addresses under an associated domain instead of the first IP that was returned on our DNS queries.

Upon some digging, we found out that many American companies have been using servers in Sweden in collaboration with Node Pole, a datacenter real estate company that works closely with many American companies. Because of their access to green energy and a colder climate that lessens the need for heat mitigation in these large data centers, Node Pole has made Sweden the go-to for many new server farms with many companies like Microsoft, Google and Meta have all having invested in data centers there [10].

6.2 India

India was only targetted by one appication, CallApp, an application that's developed by an Israeli startup that allows for users to block spam calls. Both requests to India were made to an API endpoint owned by Appodeal, an American based advertising company.

6.3 Japan

Japan was only hit by two applications: Fox Sports and Doordash. Doordash retrieved a javascript file that appeared to be for ad tracking froma a Yahoo server in Japan. Fox Sports hit the tg.socdm.com domain two times and it always responded with a 302 redirect to either sync.taboola.com or pixel.rubiconproject.com, both of which belong to advertising companies and are hosted on American IP addresses. As such, the requests were likley just advertising metrics.

6.4 Germany

Germany was only targeted by two apps: Fox Sports and Mastadon, which only made of the requests. They both targeted the adfarm1.adition.com domain, which is owned by the German advertising company Adition.

6.5 Israel

All requests to Israel were from NewsBreak or Fox Sports, both of which go to Taboola servers in Israel. Taboola is an American web advertising company, so we assume these are likely just collecting advertising data.

6.6 Case Study: Fox Sports

By a sizable margin, the Fox Sports was the application with the most outbound foreign requests. The application made a total of 649 foreign requests and a large majority of the requests were sent to Sweden. A full breakdown of all the requests sent can be seen in figure 3. Analysis of these requests and their target domain revealed that almost all of the requests sent were GET requests to an advertisement service hosted by Taboola. The requests sent over basic metadata like system OS and device architecture (e.g. Android x86) along

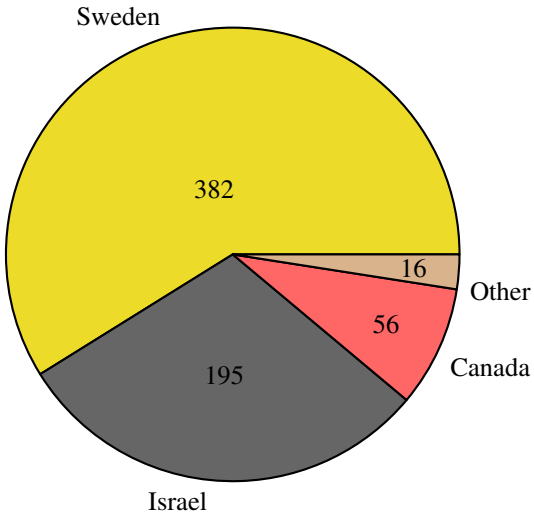

Figure 3: Pie chart reflecting the number of requests received by each foreign country while using the Fox Sports app.

with a cookie that tracks the user's current session. In some instances, the requests would send a JSON through the request body that included more metadeta such as frame size, browser version, and a DAST encoding. DAST, or Dynamic Application Security Testing, is a tool used to query request headers and detect any potential attacks or vulnerabilities in the application. Taboola is most likely using this security protocol as a defense against malicious requests to their servers.

Although Fox Sports is the largest sender of foreign requests, we learned that most of these requests are simply related to advertising with a basic metadata exchange being sent via the request body.

6.7 Case Study: Alibaba & AliExpress

Alibaba and AliExpress are two applications that became of interest to us during testing, as they were the only applications that sent requests to China. Alibaba and Aliexpress are two merchant apps, with Alibaba being for business to business cross-border wholesale and AliExpress being used for business to customer purchases. They are both owned by AliBaba, a Chinese company.

During the 15 minute period while we used AliExpress, it sent three POST requests to the following url: https://h-adashx4ae.ut.taobao.com/upload.

In each POST request, the application uploaed 18kb of data 3 times within a 15 minute testing period. The data within the request was base64 encoded, and when we tried to decode it to various different formats like UTF-8 and UTF-16, we were unsuccessful in decoding the message. This may be left to future work to investigate what data is being sent in these post requests.

We dug deeper into the request and found that the server it was connecting to was a Tangine, an open source fork of Nginx maintained by Alibaba [11]. A further dive into the codebase and modifications may be useful in understanding the request contents.

6.8 Takeaways

Generally speaking, there was no glaringly malicious behavior present in any of the applications that we tested. Most of the requests that we saw go abroad were to countries that American countries have established server farms in or to countries that are home to advertising companies that partner with American app developers.

7 Limitations

Our methodology had some limitations that are worth discussing as they limited the data that we were able to collect. The first limitation not having access to the Google Play API. Since we were unable to download the applications from the Google Play store as a typical user would, we had to resort to downloading APKs from third-party sites such as APKPure to install the latest version of every application that we intended to test. This proved to have some faults as some of the builds that we found were broken and did not launch on the emulator. Additionally, since we are not downloading the verified build there is a possibility that these builds have been altered without our knowledge, though from our testing the apps seemed to run properly when they were successful in launching.

A second limitation we found that significantly hampered our efforts to test all the applications was the lack of a SIM card. The virtual android device came with a fake phone number, but without a SIM card we couldn't use this phone number when creating an account. As a result, any applications that restrict functionality to registered users (e.g. Uber, Google Chat. etc.) were unable to be tested due to the security measures employed. We were unable to find a workaround bar purchasing a phone plan, since as virtual phone numbers did not work either.

A third limitation that slowed down our testing is the use of certificate pinning by various applications. Certificate pinning is a common method employed by developers to prevent DNS mismatch attacks or invalid certificate attacks. Certificate pinning essentially resolves the host for a given certificate and then pins the certificate to the host [12]. This clashes directly with our method of capturing network traffic, since we add our mock certificate to the system CA list to validate the requests being sent from the device to an external server. This resulted in many applications simply rejecting our certificate because it didn't match the pinned certificate associated with the host. Fortunately, we were able to find a workaround using FRIDA, as documented in our methodology.

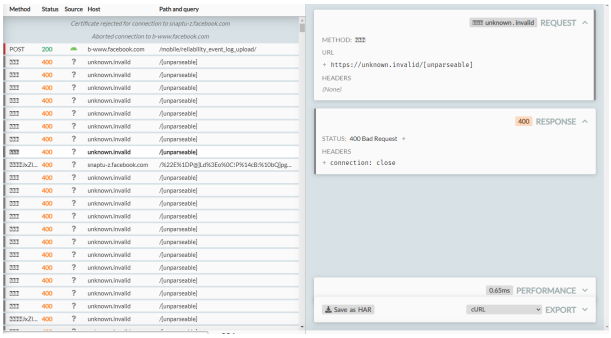

Figure 4: A screenshot of the HTTP Toolkit tool and its failure to be able to capture and read network traffic from Facebook Lite.

A fourth limitation is the pool of the applications that we were able to test was relatively small. There seems no way to get around this since the testing cannot be automated given that each application has a unique setup process and requires different testing strategies.

Another limitation we found was that some applications seemed to be detecting that we were using a simulator to run the application. For example, Walmart would frequently display a pop up window on the screen to prompt the user to verify that they are not a bot, despite already completing the verification process before. This would impede with our testing of the app because we were unable to use the basic functionality without being interrupted. It is worth noting that this functionality may have been used by other applications to detect that it was running in a virtualized environment and thus disabling malicous behavior.

8 Recommendations

Our research was largely hindered by being unable to have a very clear way to instrument Android emulators to capture traffic from real applications that can be downloaded from the Google Play Store. Google choosing to not provide a way to inspect traffic on Google Play Android builds makes this research setup much harder and creates a reliance on third party sources that should not be required. It would be useful to the security community if Google extended root capabilities to emulators that have access to the Google Play API, allowing us to inspect traffic on the exact application builds that are used in the store. If there are concerns about abuse of the Google Play APIs, checks can be put in place on the Google Play servers to deny malicious requests that may come from rooted Android devices.

9 Conclusion

In this paper, we captured and analyzed over 30,000 different HTTP requests that users of the top 100 Android apps made to servers. To focus our research on apps that provide user-generated data, we whittled down our list to 77 apps. Even when utilizing applications that enforce certificate pinning, we captured and analyzed HTTP requests issued by establishing a man in the middle connection for the applications we evaluated. In our analysis, 75% of the top applications were sending requests from American IP addresses to foreign IP addresses. The 4542 international requests were produced by these 43 applications. The other 27,798 inquiries that we examined were domestic. In general, no malicious data was identified to be transferred. However, it is also important to note that we only examined foreign requests and did not examine the kind of data that was sent to servers worldwide. Most of the requests that we observed going abroad were to nations where American countries had set up server farms or to nations where advertising organizations that collaborate with American app developers were located.

We also want to point out that a lot of requests can be connected to a single company with ties to numerous applications. We found two instances in two different countries during our investigation that support this; Germany (Adition) and Israel (Taboola). NewsBreak and Fox Sports, both of which use Taboola servers in Israel, were the source of all requests to that country. We presume that since Taboola is an American web advertising company, these are probably merely gathering advertising data. Additionally, two apps, Fox Sports and Mastadon, submitted requests to Germany, with both of them targeted the German advertising firm Adition's domain, adfarm1.adition.com.

Based on our findings, we can concur that Google's app vetting process is able to filter applications that send overtly malicious data and concur with Shen et. al's work that the vast majority of data flows stay within the United States or closely allied countries.

Ultimately, our research highlighted gaps that exist within the Android development ecosystem for profiling application network traffic and we make recommendations for improving the platfom.

References

[1] Bill Marczak and John Scott-Railton. Move fast and roll your own crypto. *Report, The Citizen Lab*, 2020. https://hdl.handle.net/1807/104313.

[2] Stats Counter. Mobile operating system market share united states of america. 2022. https://gs.statcounter.com/os-market-share/mobile/united-states-of-america.

[3] Yun Shen, Pierre-Antoine Vervier, and Gianluca Stringhini. Understanding worldwide private information collection on android. In *ISOC Networks and Distributed Systems Security Symposium*, 2021. https://arxiv.org/pdf/2102.12869.pdf.

[4] Xing Liu, Jiqiang Liu, Wei Wang, Yongzhong He, and Xiangliang Zhang. Discovering and understanding android sensor usage behaviors with data flow analysis. *World Wide Web*, 21(1):105–126, 2018.

[5] Google. What is android. 2022. https://www.android.com/what-is-android/.

[6] Check Point Blog. Expensivewall: A dangerous 'packed' malware on google play that will hit your wallet. 2017 September 14. https://blog.checkpoint.com/2017/09/14/expensivewall-dangerous-packed-malware-google-play-will-hit-wallet/.

[7] Tim Perry. Httptoolkit github repository. https://github.com/httptoolkit.

[8] Ole André Vadla Ravnås, David Weinstein, Karl Trygve Kalleberg, and Yotam. Frida github repository. https://github.com/Frida.

[9] rtHalley. dnspython. 2022. https://pypi.org/project/dnspython/.

[10] Luke Upton. How sweden has built the perfect environment for the data center of both today and tomorrow. 2021. https://www.datacenterdynamics.com/en/marketwatch/how-sweden-has-built-the-perfect-environment-for-the-data-center-of-both-today-and-tomorrow/.

[11] Alibaba. Alibaba tengine server github repository. 2022. https://github.com/alibaba/tengine.

[12] Tim Perry. Defeating android certificate pinning with frida. https://httptoolkit.com/blog/frida-certificate-pinning/.