Real-Time Unremitting Spoofing of Location Coordinates for Users' Privacy

By

Anum Arshad

A thesis submitted to the faculty of Information Security Department,
Military College of Signals, National University of Sciences and Technology,
Islamabad, Pakistan, in partial fulfillment of the requirements for the degree of MS in Information Security

September 2021

DECEIVING EAVESDROPPERS BY REAL TIME PERSISTENT SPOOFING OF ANDROID USERS' LOCATION COORDINATES FOR PRIVACY ENHANCEMENT

By

Anum Arshad

A thesis submitted to the faculty of Information Security Department,
Military College of Signals, National University of Sciences and Technology,
Islamabad, Pakistan, in partial fulfillment of the requirements for the degree of MS in Information Security

J U L Y 2021

THESIS ACCEPTANCE CERTIFICATE

It is certified that final copy of MS Thesis written by **Anum Arshad** Registration No. **00000204852**, of Military College of Signals has been vetted by undersigned, found complete in all respect as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial, fulfillment for award of MS degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have been also incorporated in the said thesis.

Signature: [illegible]

Supervisor: Assoc. Prof. Dr. Haider Abbas

Date: ____________

Signature (HoD): ____________

Date: ____________

Signature (Dean): ____________

Date: ____________

ABSTRACT

Location-based services have exponentially escalated in the past few decades. They appear to be very practical, however, the location of user is constantly being tracked, hoarded, and monitored by tech giants without user knowledge and consent. Google keeps an eye on every motion of its users, irrespective of their security settings. For this study, various Android smartphones were carried through different places for one week without the availability of internet and location services. Later, thorough network traffic analysis revealed that the archived data collected by Google apps and services, contained location data as well. This data is transferred to Google as soon as the internet connection becomes available. Moreover, a detailed performance analysis of existing fake GPS location applications was conducted that revealed a plethora of weaknesses in their performance and none of them aimed to secure users' real location coordinates. In this paper, we present an interesting solution to obfuscate Android users' real location coordinates, even in off-line mode, thereby, guaranteeing location privacy.

DEDICATION

This thesis is dedicated to

MY FAMILY AND TEACHERS

for their love, endless support and encouragement

ACKNOWLEDGEMENTS

I am grateful to God Almighty who has bestowed me with the strength and the passion to accomplish this thesis and I am thankful to Him for His mercy and benevolence. Without his consent I could not have indulged myself in this task.
I am also thankful to my supervisor especially and committee members who have always guided me with their keen and useful counselling in achieving my research objectives.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

Location Based Services	LBSs
Location Based Applications	LBAs
Global Positioning System	GPS
Point-Of-Interest	POI
Augmented Reality	AR
End User License Agreement	EULA
Personal Identifiable Information	PII
Mobile Crowd Sensing	MCS
Private Information Retrieval	PIR
Root Certificate Authority	CA
Encrypted Mobile User Identity	EMUI
International Mobile Equipment Identity	IMEI
Lower Bound	LB
Upper Bound	UB

Chapter 1

INTRODUCTION

Usage of Location Based Services (LBSs) and Location Based Applications (LBAs) have seen a significant surge due to the increasing popularity of smartphones over the decades. A satellite navigation device, colloquially called a GPS (Global Positioning System) receiver is now one of the most essential components in mobile devices. According to statistics [1], in 2018, the LBS users in US reached a number of 242 million. Such massive boom in their usage will rise further because of their application in a wide range of areas like navigation and maps [2], finding point-of-interest (POI), health and sports assistants [3], mobile social networks [4] [5], Augmented Reality (AR) games [6], tourist guide information [7], proximity-based notification [8] and a lot more. This increased reliance on LBSs has exposed users to vulnerabilities as their location data can also be accessed and eavesdropped without their consent, thereby, accentuating the need for its protection.

Tech giants (Google, Facebook etc.) [9], [10], [11], [12] and a number of third party applications are continuously monitoring and tracking user location based on the End User License Agreement (EULA) and privacy policies under the banner of improving their services. According to the current Google Privacy Policy [13]; Google apps, sites, devices, platforms (Android Operating System (OS) and Chrome) and products that are embedded in third-party apps like Google Maps etc. are tracking and monitoring user's data for providing better services. They are collecting and storing Personal Identifiable Information (PII), especially location coordinates even when users have turned off their location sharing information. GPS, IP address, sensor data from the device, Wi-Fi access points, Bluetooth-enabled device, and cell towers are all means of collecting location information. This entire process is so covert that it never hampers normal user activity and the user never realizes what the smartphone is doing in the background and what sort of information it is leaking out to those who want it. Hence, Android users are not left with much of the alternatives.

1.1 Problem Statement and Objectives

The location of user is constantly being tracked and monitored by tech giants (like Google etc.)without user knowledge and consent. Google keeps an eye on every motion of its users, irrespective of their security settings. Android users real location coordinates, even in off-line mode; are transferred to Google as soon as the internet connection becomes available. Objectives of this thesis are:

- To prove that Google is continuously tracking and monitoring users real location even when user has limited the location sharing access.
- Performance analysis of fake GPS location applications; available in Google Play Store which fail to secure the real location coordinates.
- The proposed solution guards the real location coordinates by keeping the mock location within span-radius which appears to be realistic, hence adjusting the location coordinates automatically.
- The proposed algorithm is efficient in terms of performance and securing the user's real location coordinates on an Android smart phone, practically.

1.2 Thesis Outline

This thesis is divided into seven chapters:

- Chapter 1: This chapter contains introduction, problem statement and objectives. It also contains the contributions we have made in this thesis report.
- Chapter 2: In this chapter, review of literature and background is given along with brief description and comparison of existing techniques in this report.
- Chapter 3: This chapter contains location retention and analysis of GPS spoofing applications. This chapter is further divided into two sections. First section deals with analysis of location data retention and its tracking by Google. Second section deals with performance analysis of GPS spoofing applications on Google Play Store.
- Chapter 4: This chapter deals with the proposed data leakage prevention mechanism along with software architecture of the proposed mechanism-MobiShark.

- Chapter 5: This chapter deals with the practical implementation of the proposed mechanism-MobiShark application.
- Chapter 6: This chapter deals with the evaluation of the proposed mechanism.
- Chapter 7: This chapter concludes the report and future work is proposed.

Chapter 2

PRELIMINARIES

2.1 Literature Review - Major Location Privacy Techniques

There are four major techniques of location privacy preservation [14]; obfuscation, anonymization, cryptography, and limiting location information sharing. Each group has different techniques whose comparison has been carried out based on the type of mechanism used, targets, and impact on location information along with their shortcomings as shown in Table 2.1.

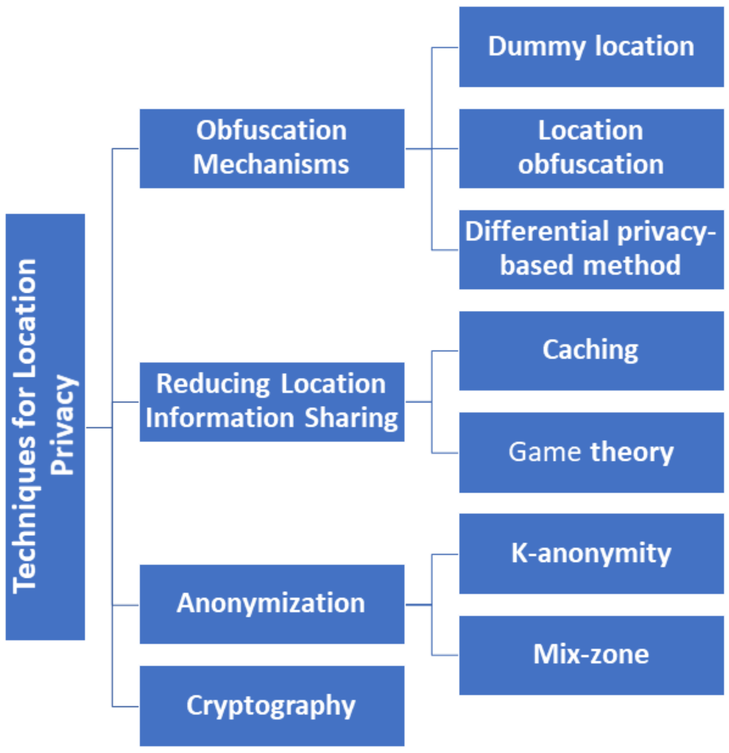


Figure 2.1: Different location preservation mechanisms

1. **Obfuscation mechanism:** This approach includes three major techniques: dummy location, location mocking, and differential privacy-based method. Hara et al. [15] devised a method to anonymize user location by creating dummy location coordinates. Do et al. [16] devised a method of conditional probabilities in order to give rise to rational false locations at which user is highly likely to be located. Location obfusca-

tion tries to preserve position information by deliberately reducing the precision sent to LBS servers by the users. Xiao and Xiong [17] devised a mechanism to maintain position privacy through temporal correlations in spatial data. A new δ-location set is proposed to protect true user location at every timestamp under temporal correlation. Several recent papers proposed differential privacy-based mechanisms. Andres et al. [18] proposed a mechanism to protect user location within a certain radius **r** with certain level of privacy that depends on **r** to achieve geo-indistinguishability with the addition of measured random noise to user's actual location. Olteanu et al. [19] proposed some inference algorithms including a solution, that relies on the belief propagation algorithm executed on a general Bayesian network model. In short, dummy location adds false location to the original location. Location obfuscation adds noise to the original location while differential privacy-based method makes the original location indistinguishable as shown in Fig.2.2.

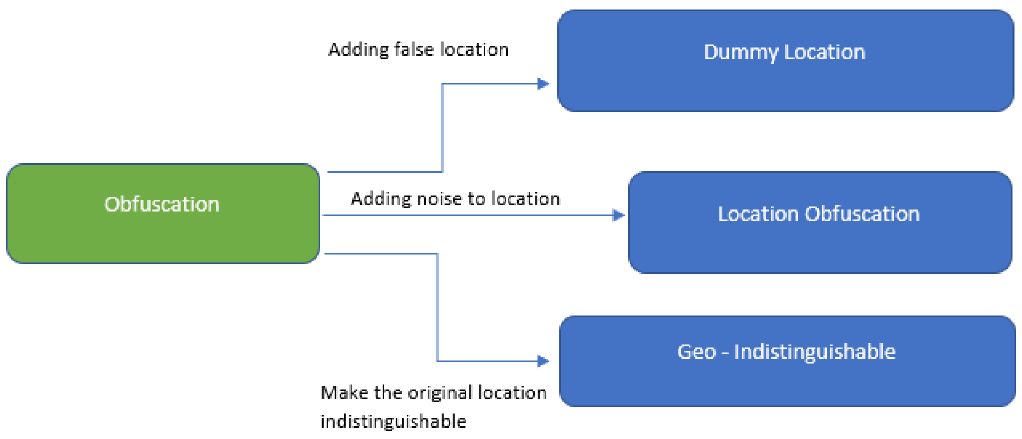


Figure 2.2: Obfuscation techniques

2. **Reducing location information sharing:** This approach has two techniques; caching and game theory. Cache systems improve user privacy by pre-fetching and storing large amount of data in a cache before arriving at an area. Niu et al. [20] put forward two algorithms. In the first algorithm, k-anonymity is achieved by selecting optimal set of dummies which contributes most to the cache hit ratio. Second algorithm deals with cache performance. Game theory mechanism can be used to minimize the sharing of location information. Liu et al. [21] presented a framework that increases location

protection by minimizing the assignment and bidding steps in the Mobile Crowd Sensing (MCS) cycle in such applications.Two approaches are shown in Fig.2.3

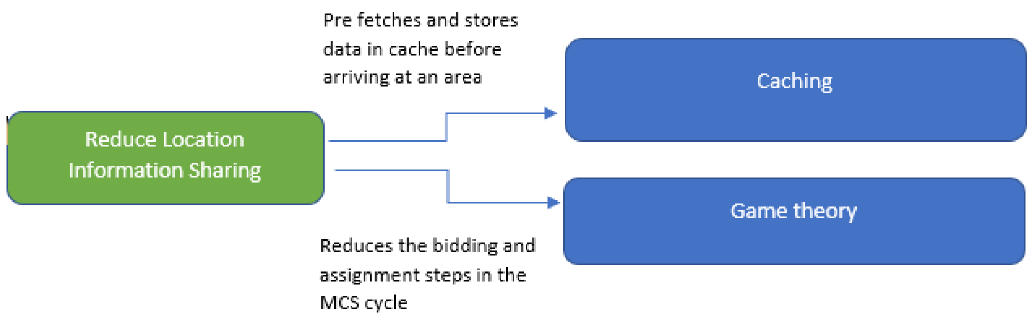

Pre fetches and stores data in cache before arriving at an area
Caching
Reduce Location Information Sharing
Game theory
Reduces the bidding and assignment steps in the MCS cycle

Figure 2.3: Reducing location information sharing techniques

3. **Anonymization methodology:** This approach comprises of two major techniques [14]; k-anonymity and mix-zone. The basic idea of the k-anonymous technique [22] is to blur a user's exact location into a cloaked area that satisfies user specified privacy requirements. Li et al. [23] proposed a concept of multiple distributed location servers along with pseudo-identity in the query to protect user identity. In contrast to this, mix-zones is used without prior knowledge of PII. Gong et al. [24] proposed a greedy algorithm that myopically determines users' strategies, based on the social group utility derived from only those users whose strategies have already been identified. Liu et al. [25] proposed MobiMix, to safeguard location privacy by proposing a framework based on mix-zones on road networks. In short k-anonymity hides the user among similar users while mix-zone change the original user identity as shown in Fig.2.4

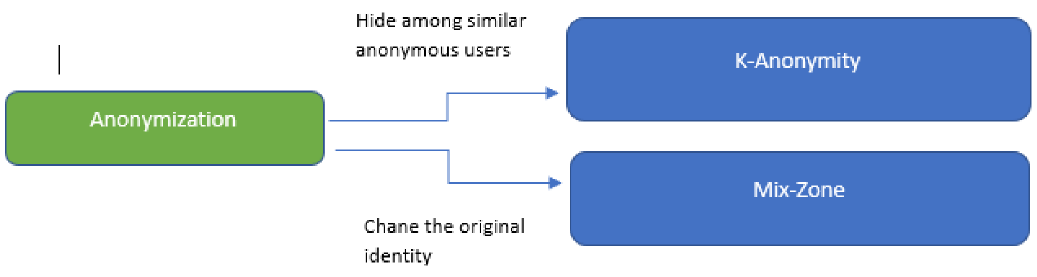

Hide among similar anonymous users
K-Anonymity
Anonymization
Mix-Zone
Chane the original identity

Figure 2.4: Anonymization techniques

4. **Cryptography:** Frameworks based on cryptography encrypt the user location information. Chen et al. [26] proposed a secure query protocol, where various data providers can use multiple secret keys to encrypt private data so that the location server fails to infer the content of the user's queried data. Mascetti et al. [27] presented a technique to inform users when their friends are in close vicinity but does not disclose the user's current position to the service providers. Ghinita et al. [28] use the technique of private information retrieval (PIR), in which LBS servers try to answer the query without revealing or learning any information about the query. Marias et al. [29] proposed an approach in which location information is divided into shares and then distributed among LBS servers. To reassemble the location information, these shares need to be retrieved and LBS servers will not be able to disclose the information as they don't possess the complete position information.

Table 2.1: Comparison Between Different Location Privacy Techniques

Location Privacy Techniques	**Types of Location Privacy Techniques**	**Mechanisms used**	**Impact on location information**	**Targets (Position/ Identity/ Time)**	**Shortcomings**
Obfuscation Mechanisms	Dummy location [15] [16]	Adds mock location	Blurs the information	Position/ Identity	Generating runtime dummies is one of the biggest challenges.
	Location obfuscation [17]	Adds noise to real location			
	Differential privacy-based methods [18] [19]	Making location indistinct			
Anonymization	K-anonymity [22] [23]	Hides among identical anonymous users	Destroys the connection between identities and locations	Position/ Identity	Computing k-1 fake locations, comes with computation & communication overhead
	Mix-zone [24] [25]	Changes identity			
Cryptography	PIR [26] [27] [28] [29]	Protects user positions through encryption	Reduces the effect of inferring information	Position/ Identity	Heavy computation cost involved in encryption
Reducing Location Information Sharing	Caching [20]	Prefetches and stores data in cache before arriving at an area.	Reduces trasmission of information through whole system	Position/ Identity/ Time	Large storage space is required
	Game theory [21]	Reduces the bidding and assignment steps in the MCS cycle.			

2.2 Comparison between Location Privacy Techniques

Anonymization breaks the connection between user's identity and location to make the location information useless. Obfuscation tries to blur the information to reduce the leakage or disclosure of the information. Reduce location information sharing tries to reduce the amount of data to transmit through the whole process to reduce the risk associated with it. Cryptography decreases the risk of an adversary to conclude useful information from the encrypted data.
Among the reviewed techniques, anonymization and obfuscation are the most predominant. Cryptography is also one of the classical techniques but it needs enhancements in terms of computation, implementation and secrecy.

2.3 Conclusion

However, these proposed mechanisms [30] have more theoretical approach than practical, and cannot be applied directly to real world scenarios. Fig.2.5 shows that these four techniques are not mutually exclusive but can be used in different combinations.

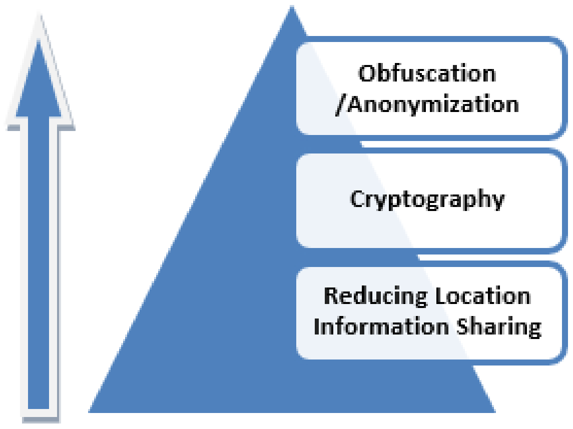

Obfuscation /Anonymization
Cryptography
Reducing Location Information Sharing

Figure 2.5: Four location preservation techniques not mutually exclusive

Apurva K. Kini [31] proposed real-time k-fake generation algorithm to preserve user's location while using location-based services. Nonetheless, computing k-1 fake locations cause computational and communication overhead to the existing system. ReCon [32] is a cross-platform system that reveals PII leaks by inspecting network traffic. It allows the user to have control over PII either by blocking or substituting them and needs a certificate for installation. However, ReCon only handles HTTP traffic and not HTTPS.

Chapter 3

LOCATION RETENTION AND ANALYSIS OF GPS SPOOFING APPLICATIONS

3.1 Analysis of Location Data Retention and its Tracking by Google

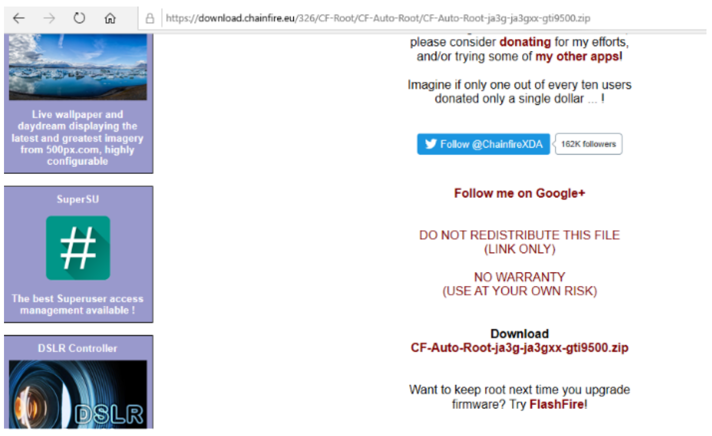
The following questions arise regarding storing and transferring of data by Google:

1. Is the location data being tracked and stored even when the user has disabled all location services and internet?
2. Is stored data transfer dependent on network availability?
3. If yes, then how stored data is transferred via network connectivity?

To answer these questions, research experiment shown in Fig. 3.7 was conducted on different Samsung smartphones operating on various Android versions from Lollipop up-to Oreo, which targets 78.9% [33] of the overall Android devices.

3.1.1 Experimental Settings (with TCPDUMP)

To take tcpdump its necessary to root the mobile first. To carry out experiments, Samsung S4 Model:GT-I9500 is used. To root the mobile CF-Auto-Root file for the particular model is downloaded from CF-Auto-Root Repository as shown in Fig.3.1.


https://download.chainfire.eu/326/CF-Root/CF-Auto-Root/CF-Auto-Root-ja3g-ja3gxx-gti9500.zip

Live wallpaper and daydream displaying the latest and greatest imagery from 500px.com, highly configurable

SuperSU

The best Superuser access management available !

DSLR Controller

please consider **donating** for my efforts, and/or trying some of **my other apps!**

Imagine if only one out of every ten users donated only a single dollar ... !

Follow @ChainfireXDA 162K followers

Follow me on Google+

DO NOT REDISTRIBUTE THIS FILE (LINK ONLY)

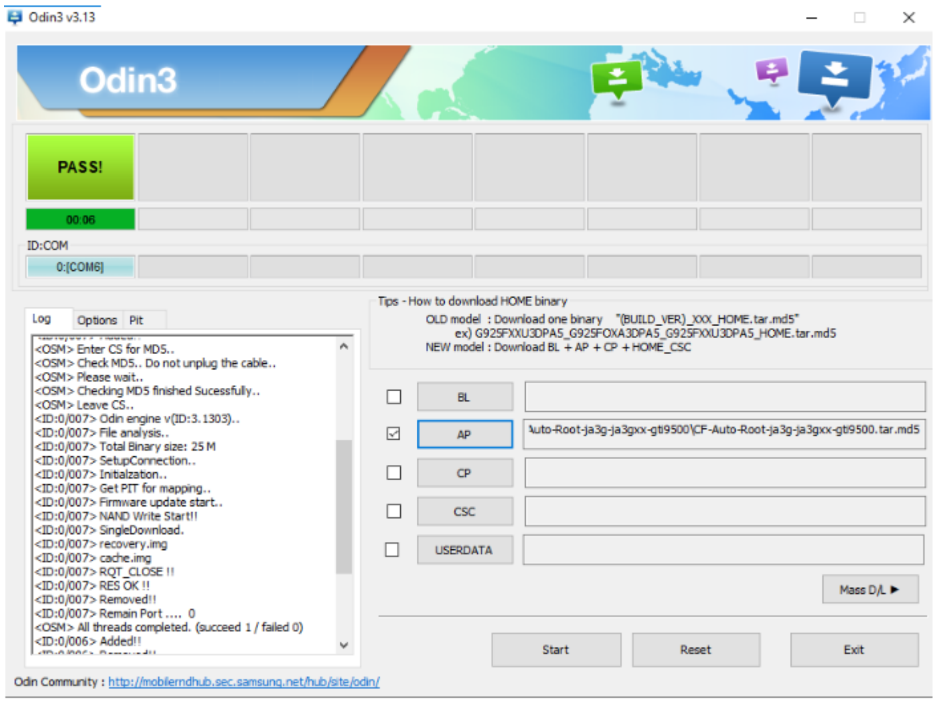
NO WARRANTY (USE AT YOUR OWN RISK)

Download
CF-Auto-Root-ja3g-ja3gxx-gti9500.zip

Want to keep root next time you upgrade firmware? Try **FlashFire!**

Figure 3.1: CF-Auto-Root Repository for the experimental device

Odin 3 v3.13 is used to carry out rooting shown in Fig.3.2. Loading AP file in Odin after starting the debugging mode in mobile .After loading press the start button to start rooting . After completely all the threads; mobile restarts and SuperSU is created which indicates that mobile has successfully rooted .To make confirmation , install Root Checker which will show the current status of mobile.



```
<OSM> Enter CS for MD5..
<OSM> Check MD5.. Do not unplug the cable..
<OSM> Please wait..
<OSM> Checking MD5 finished Sucessfully..
<OSM> Leave CS..
<ID:0/007> Odin engine v(ID:3.1303)..
<ID:0/007> File analysis..
<ID:0/007> Total Binary size: 25 M
<ID:0/007> SetupConnection..
<ID:0/007> Initialzation..
<ID:0/007> Get PIT for mapping..
<ID:0/007> Firmware update start..
<ID:0/007> NAND Write Start!!
<ID:0/007> SingleDownload.
<ID:0/007> recovery.img
<ID:0/007> cache.img
<ID:0/007> RQT_CLOSE !!
<ID:0/007> RES OK !!
<ID:0/007> Removed!!
<ID:0/007> Remain Port ....  0
<OSM> All threads completed. (succeed 1 / failed 0)
<ID:0/006> Added!!
```

Tips - How to download HOME binary
OLD model : Download one binary "(BUILD_VER)_XXX_HOME.tar.md5"
ex) G925FXXU3DPA5_G925FOXA3DPA5_G925FXXU3DPA5_HOME.tar.md5
NEW model : Download BL + AP + CP + HOME_CSC

[✓ AP] \uto-Root-ja3g-ja3gxx-gti9500\CF-Auto-Root-ja3g-ja3gxx-gti9500.tar.md5

Odin Community : http://mobilerndhub.sec.samsung.net/hub/site/odin/

Figure 3.2: Odin3 for rooting

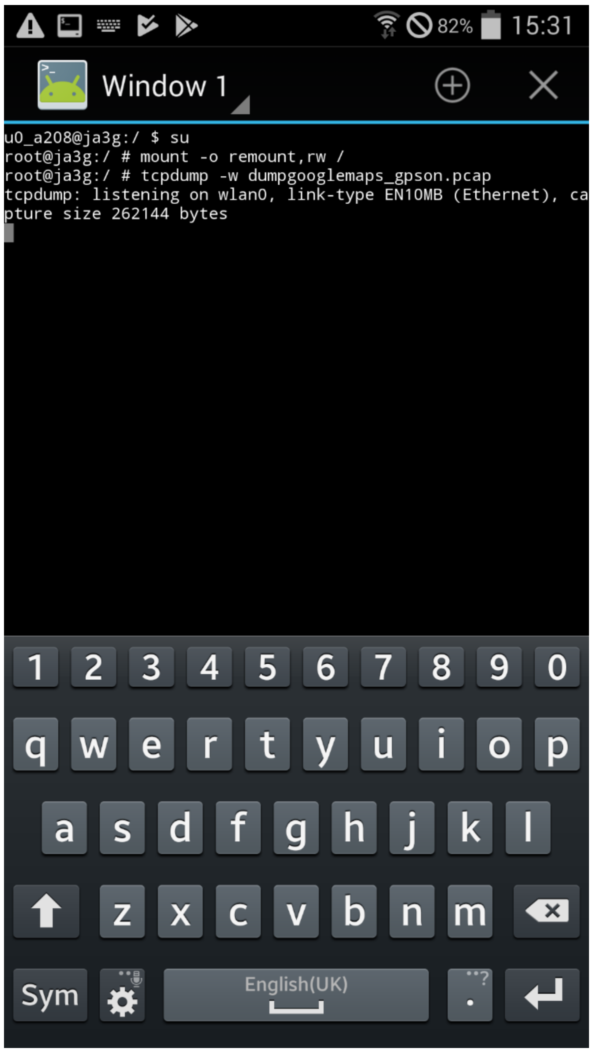
Install Terminal Emulator shown in Fig.3.3 and type following commands and open Google Maps .It will dump the network traffic generated.

Install Root Power Explorer to view the Root directory shown in Fig.3.4. Change the permissions of the file and copy the file from Root to SDCard as direct copy from Root folder is not allowed.

3.1.2 Experimental Results

Transfer the file from mobile to laptop and view in Wireshark shown in Fig.3.5. Its clear that Coordinates are travelling in GET request. It means that coordinates are travelling in plain text as well.

Checking the coordinates on MAPS.ie shows the exact location shown in Fig.3.6.



```
u0_a208@ja3g:/ $ su
root@ja3g:/ # mount -o remount,rw /
root@ja3g:/ # tcpdump -w dumpgooglemaps_gpson.pcap
tcpdump: listening on wlan0, link-type EN10MB (Ethernet), ca
pture size 262144 bytes
```

Figure 3.3: Terminal emulator on the experimental device

3.1.3 Experimental Settings (with FIDDLER)

The smart phones were rooted and factory settings were reset. All of the location services, location history, web & application activity along with browser activity were disabled. Root Certificate Authority (CA) of Fiddler which is a free web debugging proxy tool; was pre-installed on each experimental device to view the generated traffic. Fiddler was configured to view both HTTP and HTTPS traffic. As a next step, these experimental phones were taken to different locations for a week with no internet connectivity and all the location services were turned off. Steps are shown below:

1. The smartphones were rooted and factory settings were reset.

2. All of the location services, location history, web & application activity along with browser activity were disabled.

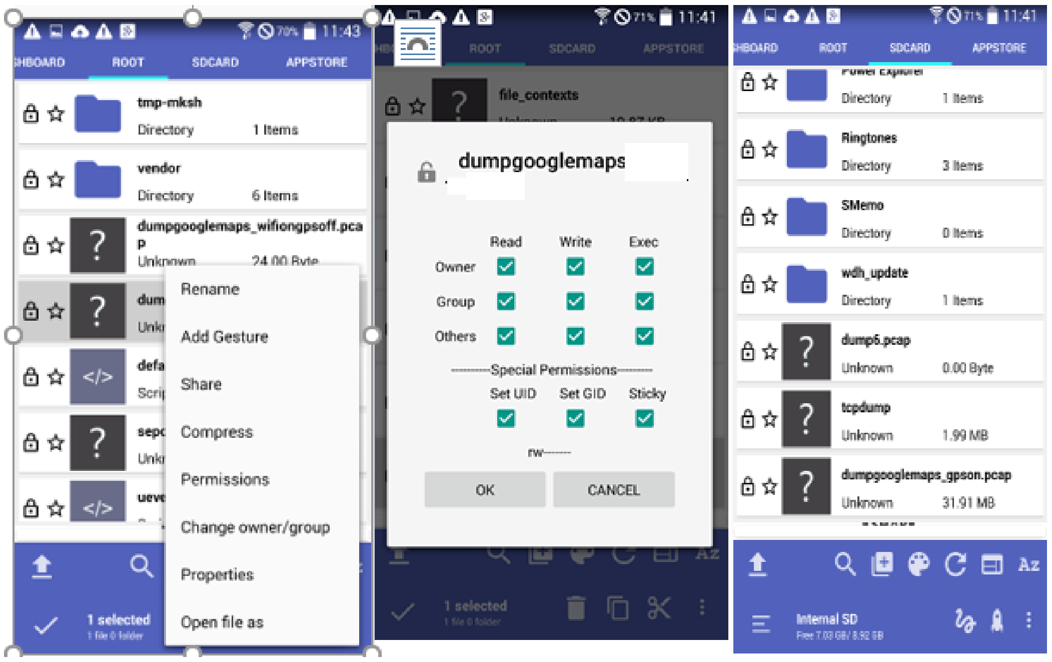


Figure 3.4: Copy file from Root to SDCard with Root Power Explorer

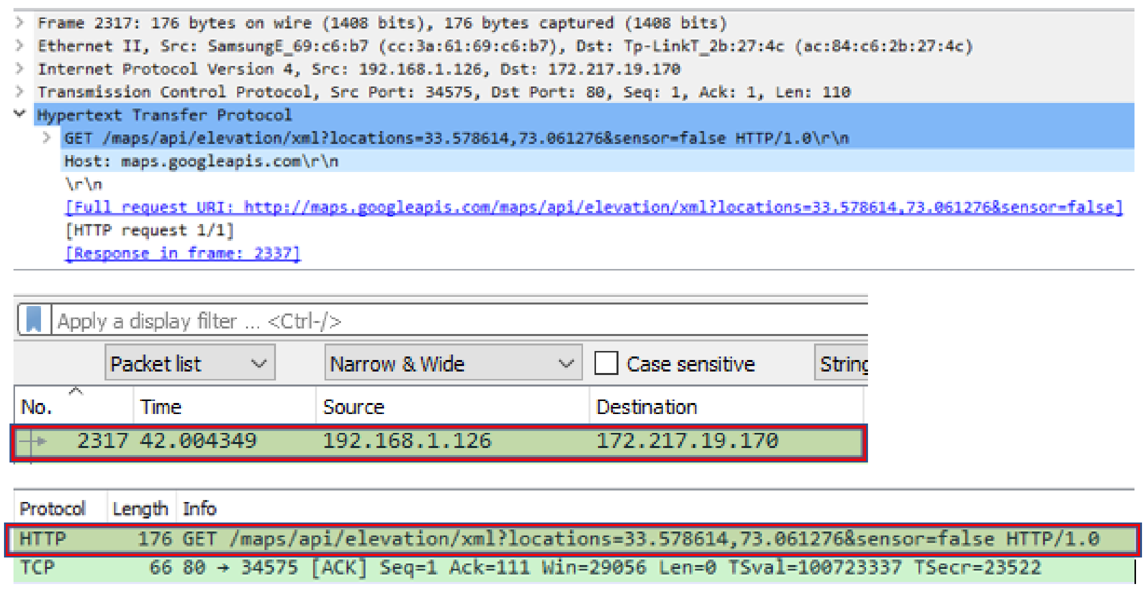
3. Root Certificate Authority (CA) of Fiddler was pre-installed on each experimental device to view the generated traffic.

4. Fiddler was configured to view both HTTP and HTTPS traffic.

5. Experimental phones were taken to different locations for a week with no internet connectivity and all the location services were turned off.

3.1.4 Experimental Results

It was found out that there were log/batch files which can be seen in Fig. 3.8 containing the archived location data. This data was transferred to **play.googleapis.com** over different intervals of time. The requested URL of **play.googleapis.com** was unreachable and threw an error exception. Some log/batch files were transferred immediately after the availability of internet while some were transferred after an hour or two. These log/batch files contained extensive location data which clearly indicated that location was being tracked even in the offline mode where user had restricted to access the location services. Even if the user had opted for USER_LOCATION_REPORTING_DISABLED option as shown in Fig.3.9 still most of the Google applications such as google maps, google earth, google location, and google backup transport were still able to gather most of the location related information from devices as seen in Fig. 3.10 and Fig.3.11. This result has further strengthened our



```
> Frame 2317: 176 bytes on wire (1408 bits), 176 bytes captured (1408 bits)
> Ethernet II, Src: SamsungE_69:c6:b7 (cc:3a:61:69:c6:b7), Dst: Tp-LinkT_2b:27:4c (ac:84:c6:2b:27:4c)
> Internet Protocol Version 4, Src: 192.168.1.126, Dst: 172.217.19.170
> Transmission Control Protocol, Src Port: 34575, Dst Port: 80, Seq: 1, Ack: 1, Len: 110
v Hypertext Transfer Protocol
  > GET /maps/api/elevation/xml?locations=33.578614,73.061276&sensor=false HTTP/1.0\r\n
    Host: maps.googleapis.com\r\n
    \r\n
    [Full request URI: http://maps.googleapis.com/maps/api/elevation/xml?locations=33.578614,73.061276&sensor=false]
    [HTTP request 1/1]
    [Response in frame: 2337]
```

```
No.   Time        Source          Destination
2317 42.004349    192.168.1.126   172.217.19.170

Protocol  Length  Info
HTTP        176 GET /maps/api/elevation/xml?locations=33.578614,73.061276&sensor=false HTTP/1.0
TCP          66 80 → 34575 [ACK] Seq=1 Ack=111 Win=29056 Len=0 TSval=100723337 TSecr=23522
```

Figure 3.5: WireShark view of the captured .pcap file

confidence that Google was tracking users' movements irrespective of the fact whether the user has toggled on/off all the location related services and applications.

Apart from the location specific data, decrypted data also contained information about the device like name, model, build number, EMUI version, Android version, Kernel version, IMEI, CPU, RAM, internal storage, and baseband version as shown in Fig. 3.12 . Traffic analysis clearly indicated that the ongoing API calls to Google applications and services forewarn that sheer volume of personal information including location was collected and transferred regardless of the settings, users believe they control.

3.2 Performance Analysis of GPS Spoofing Applications on Google Play Store

There are number of GPS spoofing applications freely available on Google Play Store whose implementations are very different from theoretical proposals. Some of the most famous applications are, Fake GPS Location-Lexa, Fake GPS GO Location-IncorporateApps, Fake GPS-ByteRev, Fake GPS Location-Hola, Fake GPS location-Digital Center, Location Changer (Fake GPS Location)-Netlinkd, GeoTag-Fake & Spoof GPS Location-Codeberry Finland, Fake GPS Location PRO-Just4Fun etc. Their performance efficiency is measured on the basis of following parameters:

1. **Movement Simulation:** Mocks the location in real motion environment

2. **Mocking Accuracy:** Mocks the location to some realistic location (location that is

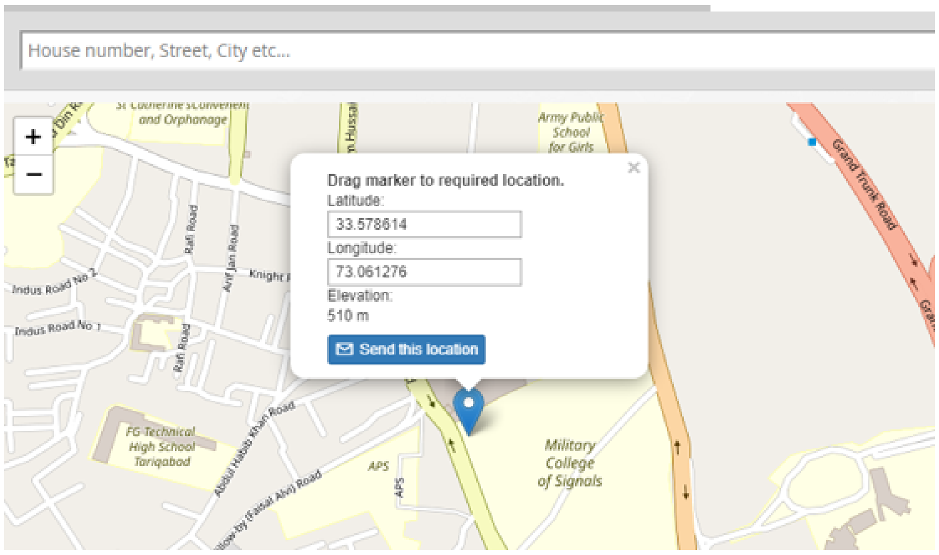

House number, Street, City etc...
Drag marker to required location.
Latitude:
33.578614
Longitude:
73.061276
Elevation:
510 m
Send this location
Military College of Signals
Grand Trunk Road

Figure 3.6: Map view of the captured coordinates

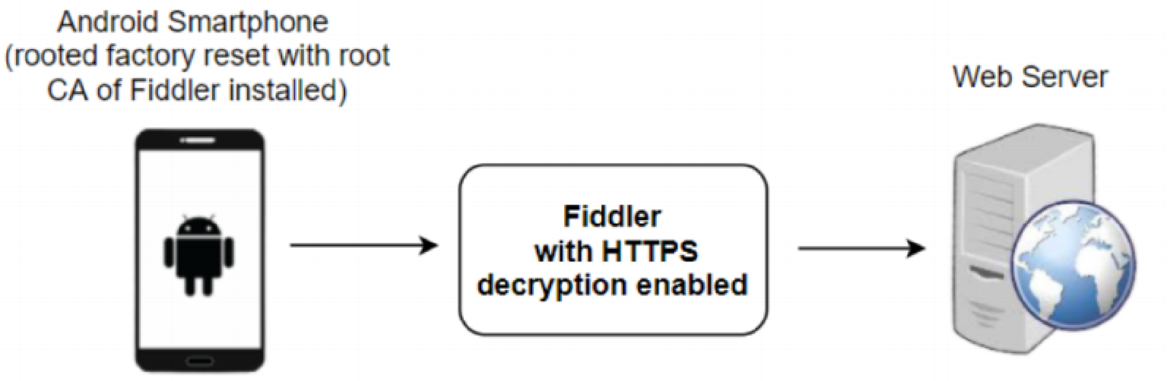

Android Smartphone (rooted factory reset with root CA of Fiddler installed)
Fiddler with HTTPS decryption enabled
Web Server

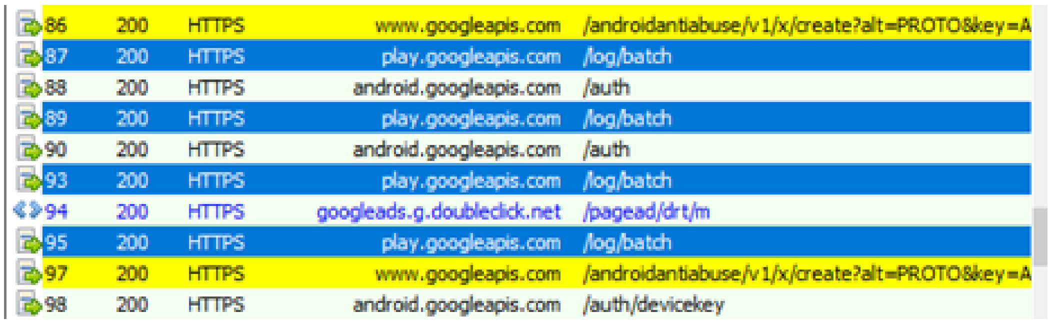
Figure 3.7: Android traffic analysis through Fiddler

sensibly expected to be real and practical including near the seashores, ocean-front, coastal-areas etc. rather than somewhere in the middle of the oceans etc.)

3. **Mocking Persistency:** Scale (high, medium, low) is defined to analyze the mocking capability of the tested application on a number of experimental applications (Google Maps, maps.me, Careem, Bykea, Facebook, Foursquare, Sygic etc.). Each level of the scale is gauged on the basis of mocking, not mocking or re-shifts to the original location after mocking.

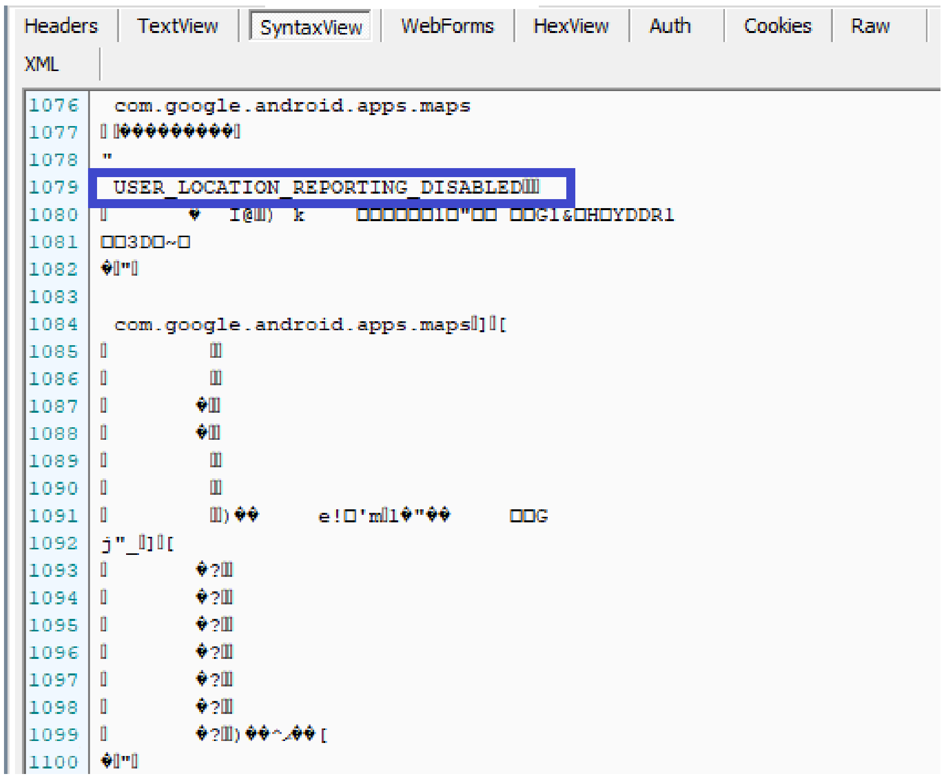
4. **Security of Real Location Co-ordinates:** To determine if the real location coordinates are being secured from external parties.

5. **Permission Requirements at Installation Time:** Requirements of the application at the installation time (either access to location, photos, media, files etc.).



```
86   200  HTTPS        www.googleapis.com  /androidantiabuse/v1/x/create?alt=PROTO&key=A
87   200  HTTPS        play.googleapis.com  /log/batch
88   200  HTTPS     android.googleapis.com  /auth
89   200  HTTPS        play.googleapis.com  /log/batch
90   200  HTTPS     android.googleapis.com  /auth
93   200  HTTPS        play.googleapis.com  /log/batch
94   200  HTTPS  googleads.g.doubleclick.net  /pagead/drt/m
95   200  HTTPS        play.googleapis.com  /log/batch
97   200  HTTPS        www.googleapis.com  /androidantiabuse/v1/x/create?alt=PROTO&key=A
98   200  HTTPS     android.googleapis.com  /auth/devicekey
```

Figure 3.8: Log/batch files



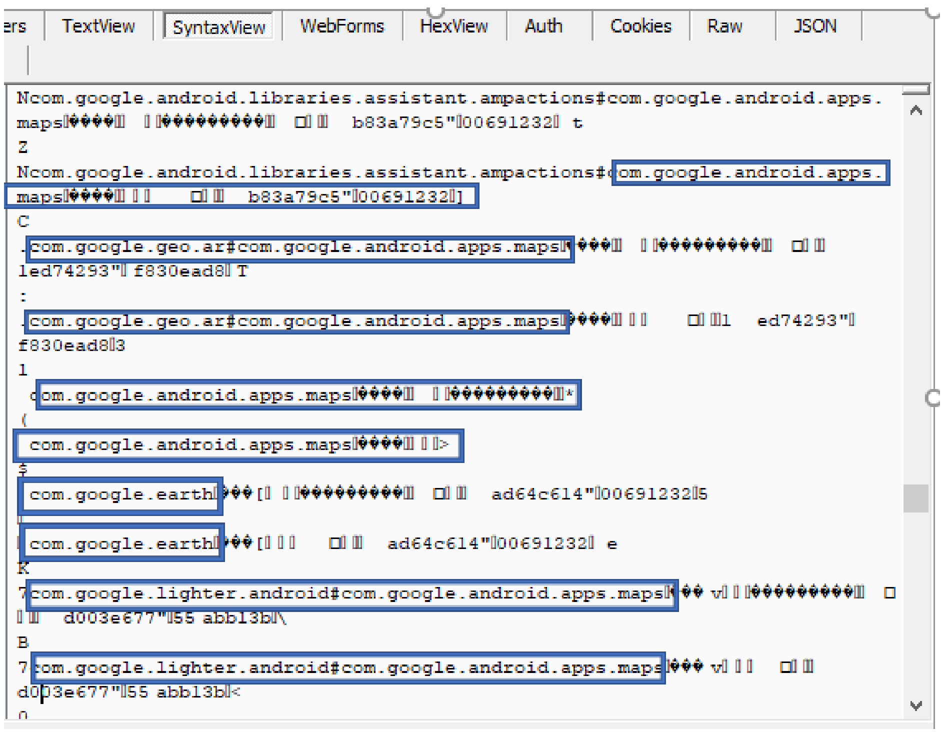
```
 com.google.android.apps.maps
▯ ▯����������▯
"
 USER_LOCATION_REPORTING_DISABLED▯▯
▯         ♥  I@▯▯)  k     ▯▯▯▯▯▯▯I▯"▯▯ ▯▯G1&▯H▯YDDR1
▯▯3D▯~▯
�▯"▯

 com.google.android.apps.maps▯]▯[
▯         ▯▯
▯         ▯▯
▯        �▯▯
▯        �▯▯
▯         ▯▯
▯         ▯▯
▯         ▯▯)��      e!▯'m▯1�"��      ▯▯G
j"_▯]▯[
▯        �?▯▯
▯        �?▯▯
▯        �?▯▯
▯        �?▯▯
▯        �?▯▯
▯        �?▯▯
▯        �?▯▯)��^✓��[
�▯"▯
```

Figure 3.9: USER_LOCATION_REPORTING_DISABLED option ENABLED

6. **Show Mock Location after Uninstalling:** To determine if the fake location application keeps showing mocked location even after its has been uninstalled.

Table 3.1 shows the performance analysis of different Fake GPS location applications available on the Google Play store. We collected and analyzed the applications that had higher number of downloads and best user reviews on Google play store. Later, we thoroughly evaluated their performance in different testing scenarios.

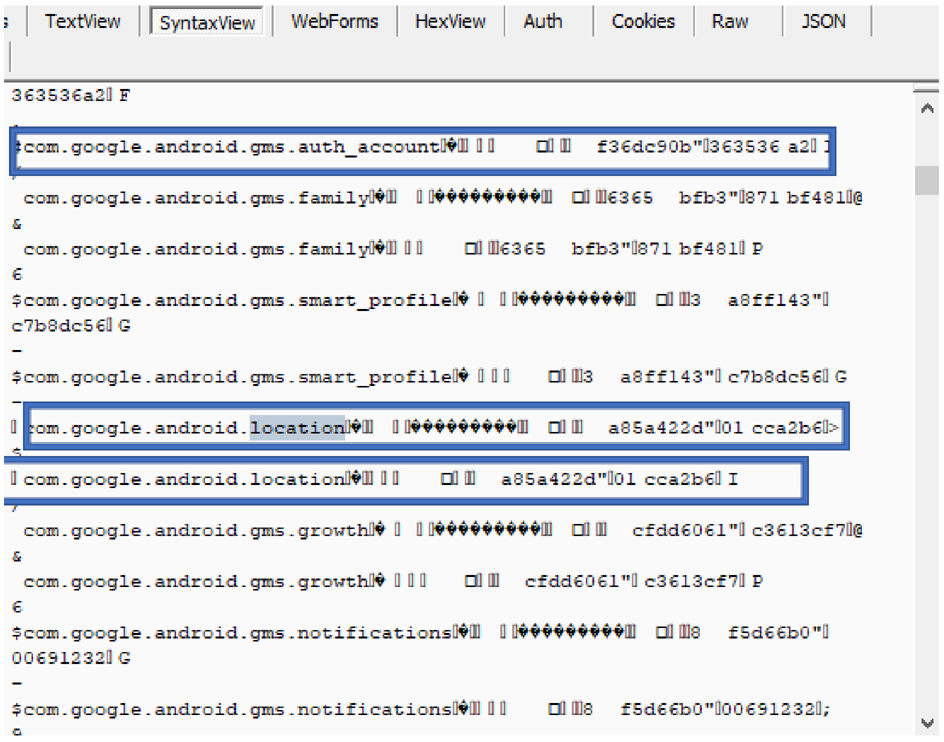
- Operating system under memory pressure (low memory state)
- Unstable GPS connection state (when living in high elevated buildings)



```
Ncom.google.android.libraries.assistant.ampactions#com.google.android.apps.
maps�����  ����������� ��  b83a79c5"00691232 t
Z
Ncom.google.android.libraries.assistant.ampactions#com.google.android.apps.
maps����� ��    ��  b83a79c5"00691232]
C
.com.google.geo.ar#com.google.android.apps.maps����  �����������  ��
led74293" f830ead8 T
:
.com.google.geo.ar#com.google.android.apps.maps���� ��    ��1   ed74293"
f830ead83
1
 com.google.android.apps.maps�����  �����������*
(
 com.google.android.apps.maps����� ��>
$
 com.google.earth���[  �����������  ��   ad64c614"006912325

 com.google.earth���[ ��    ��   ad64c614"00691232 e
K
7com.google.lighter.android#com.google.android.apps.maps�� v ������������  □
��   d003e677"55 abb13b\
B
7com.google.lighter.android#com.google.android.apps.maps��� v ��   ��
d003e677"55 abb13b<
0
```

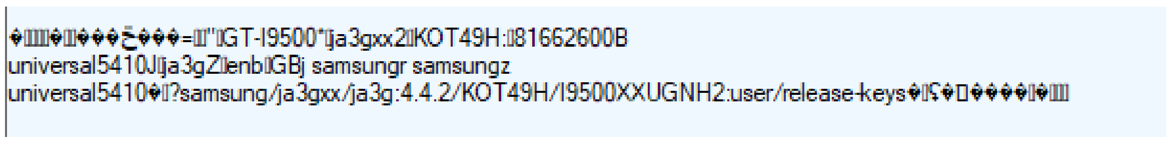
Figure 3.10: Traffic analysis of log/batch files (a)

It depicts that none of these fake GPS location applications aim to secure the real location co-ordinates. Applications having high mocking persistence require more permissions at installation time. Moreover, most of the applications mock to unrealistic locations. Our proposed solution not only persistently secures real location coordinates but also prevents mocking to unrealistic locations with very minimal permission requirements at installation time.



```
363536a2▯ F

$com.google.android.gms.auth_account▯�▯ ▯ ▯    □▯ ▯   f36dc90b"▯363536 a2▯ I

 com.google.android.gms.family▯�▯  ▯ ▯����������▯  □▯ ▯6365   bfb3"▯871 bf481▯@
&
 com.google.android.gms.family▯�▯ ▯ ▯    □▯ ▯6365   bfb3"▯871 bf481▯ P
6
$com.google.android.gms.smart_profile▯� ▯  ▯ ▯����������▯  □▯ ▯3   a8ff143"▯
c7b8dc56▯ G
-
$com.google.android.gms.smart_profile▯� ▯ ▯ ▯    □▯ ▯3   a8ff143"▯ c7b8dc56▯ G
-
▯ com.google.android.location▯�▯  ▯ ▯����������▯  □▯ ▯   a85a422d"▯01 cca2b6▯>
$
▯ com.google.android.location▯�▯ ▯ ▯    □▯ ▯   a85a422d"▯01 cca2b6▯ I

 com.google.android.gms.growth▯� ▯  ▯ ▯����������▯  □▯ ▯   cfdd6061"▯ c3613cf7▯@
&
 com.google.android.gms.growth▯� ▯ ▯ ▯    □▯ ▯   cfdd6061"▯ c3613cf7▯ P
6
$com.google.android.gms.notifications▯�▯  ▯ ▯����������▯  □▯ ▯8   f5d66b0"▯
00691232▯ G
-
$com.google.android.gms.notifications▯�▯ ▯ ▯    □▯ ▯8   f5d66b0"▯00691232▯;
```

Figure 3.11: Traffic analysis of log/batch files (b)



```
�▯▯▯�▯▯���ڿ���=▯▯"▯GT-I9500*▯ja3gxx2▯KOT49H:▯81662600B
universal5410J▯ja3gZ▯enb▯GBj samsungr samsungz
universal5410�▯?samsung/ja3gxx/ja3g:4.4.2/KOT49H/I9500XXUGNH2:user/release-keys�▯Ϛ�□����▯�▯▯▯
```

Figure 3.12: Device related captured information

Table 3.1: Analysis of Different Fake GPS Applications

Applications	**Movement simulation**	**Mocking accuracy**	**Mocking persistency**	**Security of real location coordinates**	**Permission requirments at installation time**	**Show mock location after uninstallation**
Fake GPS Location-Lexa	No	No	High	No	Photos/media/files	Yes
Fake GPS GO Location-IncorporateApps	No	No	Medium	No	Location	No
Fake GPS-ByteRev	No	No	Medium	No	Location	No
Fake GPS Location-Hola	No	No	Low	No	Location	No
Fake gps location-Digital Center	No	No	Medium	No	Location	No
Location Changer (Fake GPS Location)-Netlinkd	No	No	Medium	No	Location	Yes
GeoTag-Fake &amp; Spoof GPS Location-Codeberry Finland	No	No	Medium	No	Location	No
Fake GPS Location PRO-Just4Fun	Yes	No	Low	No	Location	No
MobiShark	Yes	Yes	High	Yes	Location	No

Chapter 4

PROPOSED DATA LEAKAGE PREVENTION MECHANISM AND SOFTWARE ARCHITECTURE - MOBISHARK

In order to protect user's location, the general working of the Android application-MobiShark is shown in Fig. 4.1. Location interfaces are been spoofed by the MobiShark Spoofer. This spoofed location is provided to all the android applications through Location Providers.

4.1 Charachteristics of Android Application-MobiShark Spoofer

Following are the characteristics of the Android application-MobiShark.

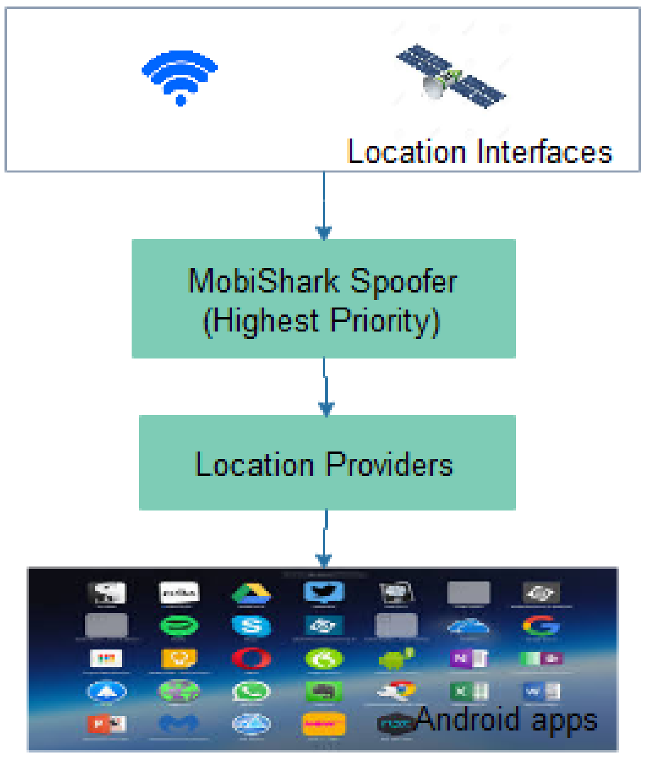


Figure 4.1: Architecture of MobiShark

- It secures the real location coordinates.

- It continuously updates the location providers with mock location so that they do not fetch original location.

- It mocks the location within span-radius having upper and lower bounds so that mock location appears to be rational and practical.

- It prevents mocking to some unrealistic location like somewhere in the middle of oceans etc.

4.2 Software Architecture of Android Application-MobiShark Spoofer

Software architecture of the MobiShark application is shown in Fig. 4.2. Following are the short description of the components used in software architecture.

- Phone Manager is an Android application that helps one to see and manage all the running processes, running services and installed applications of one device.

- Alarm Manager helps to schedule the specific application to run at the scheduled time in the future. It basically holds the CPU wake lock which ensures that the phone will not sleep until broadcast is handled.

- GPS Manager and Network Manager are the Android services that allow applications to get periodic updates related to device's geographical location.

The proposed application launches the Smart Mocking Service that runs continuously in the background and is managed/controlled by the Alarm Manager to keep it at the highest priority. Broadcast message is sent through the phone manager to all the Geolocation Client Applications which get location coordinates from the GPS Manager and Network Manager, and would ultimately get mocked coordinates instead of the original ones. This efficient and smart mocking service would continuously feed the GPS Manager and Network Manager with the mock coordinates within span-radius (3km $\leq$ mock location $\leq$ 15km) so that the mock location appears to be realistic and practical. These Client Applications would then pass mock location to their respective servers.

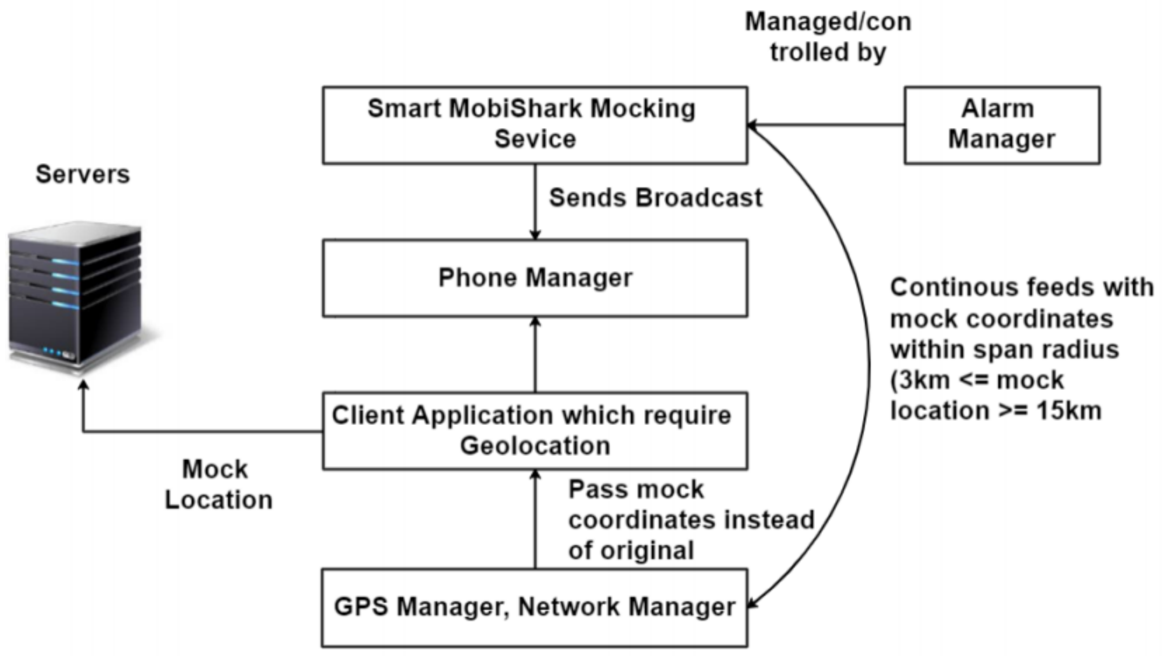

Managed/con trolled by
Smart MobiShark Mocking Sevice
Alarm Manager
Servers
Sends Broadcast
Phone Manager
Continous feeds with mock coordinates within span radius (3km <= mock location >= 15km
Client Application which require Geolocation
Mock Location
Pass mock coordinates instead of original
GPS Manager, Network Manager

Figure 4.2: Software architecture Of MobiShark

4.3 Proposed Algorithm for the Android Application-MobiShark Spoofer

Location providers need to be updated frequently, approximately 10Hz (10 times per second) with mock coordinates so that, whichever application or service is requesting for a location update, gets the mock coordinates. Automatic generation of span-radius around user's actual location as shown in Fig. 4.3 whose lower and upper bounds are set in a way that mock location falls within this range so that LBSs and LBAs are unable to detect this location anomaly. For experimental purposes, lower bound has been set to **n** km and upper bound to **m** km. Value of **n** should be in range of 3km $\leq$ n $\leq$ 5km, while **m** should be in range of **n** $<$ **m** $\leq$ 15km. These ranges will allow the mock location to fall in radius of 3km $\leq$ mock location $\leq$ 15km, so that mock location appears to be practical and real. General formulas used to calculate new longitude, new latitude, random angle and random radius are as follows:

- **New Latitude** = Math.asin (Math.sin (original_latitude) * Math.cos (distance/Radius) + Math.cos (original_laltitude) * Math.sin (random_radius /Radius) * Math.cos (random_angle))
- **New Longitude** = original_longitude + Math.atan2 (Math.sin (random_angle) * Math.sin (random_radius /Radius) * Math.cos (original_latitude) , Math.cos (ran-

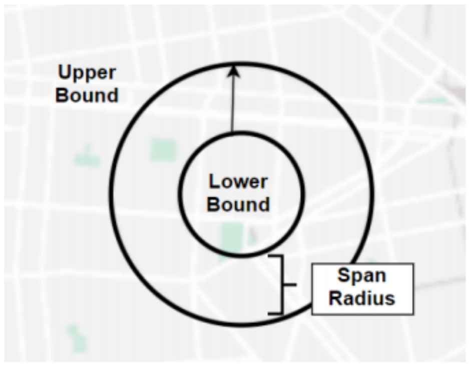


Figure 4.3: Upper and lower bounds of span-radius

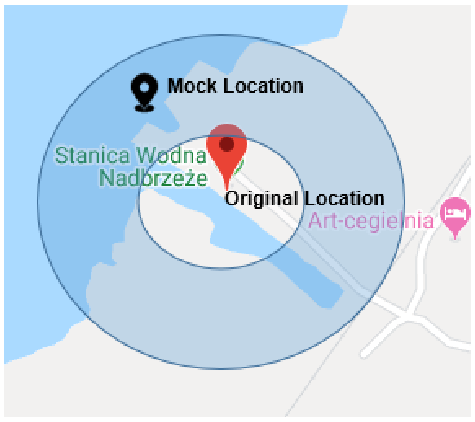


Figure 4.4: Inability to mock to unrealistic location

dom_radius /Radius)- Math.sin (original_latitude) * Math.sin (new_latitude))

- **Random Radius** = Random Radius is calculated by subtracting Lower Bound from Upper Bound and adding the Lower Bound to the result to keep it within range of 3km $\leq$ 15km.

  random radius(d) = ran.nextDouble(UB-LB)+LB

- **Random Angle** = Random angle is calculated by multiplying 2 with value of PI and then multiplying the result with the random number generated.

  random_angle(ϕ) = 2* Math.PI * ran.nextDouble()

Algorithm 1 shows the methodology being followed to generate mock location within span-radius.

Instead of caching the whole earth repository for all location coordinates and applying validation to confirm that the mocked coordinates do not fall in the category of unrealistic locations, the strategy of mocking location in accordance with span-radius has been used which will automatically eliminate the probability of unrealistic mocking. If the location is mocked to somewhere near water as shown in Fig. 4.4 then, it appears to be realistic like near the seashore, ocean-front, coastal areas etc. Not in the centre of seas, oceans , rivers which might give a clue that the user might be using mocking service.

Algorithm 1: Proposed algorithm of generating fake coordinates with-in span-radius

```
Lower Bound Range = 3km $\leq$ n $\leq$ 5km
Upper Bound Range = n $<$ m $\leq$ 15km
LB = n km
UB = m km
Generating fake location coordinates in span-radius(original longitude X1, original
  latitude Y1, lowerbound LB, upperbound UB)
{
    Random ran = new Random ();
    random radius(d) = ran.nextDouble(UB-LB)+LB;
    random_angle($\phi$) = 2* Math.PI * ran.nextDouble();
    Earth Radius = R;

    //Formula used to calculate new longitude and new latitude are as follows:

    //New Latitude:

    New_latitude = Math.asin (Math.sin (original_latitude) * Math.cos
  (distance/Radius) + Math.cos (original_laltitude) * Math.sin (random_radius
  /Radius) * Math.cos (random_angle));

    Y2 = Math.asin( Math.sin(Y1) * Math.cos(d/R) + Math.cos(Y1) * Math.sin(d/R) *
  Math.cos($\phi$));

    //New Longitude:

    New_longitude = original_longitude + Math.atan2 (Math.sin (random_angle) *
  Math.sin ( random_radius /Radius) * Math.cos (original_latitude) , Math.cos
  (random_radius /Radius)- Math.sin (original_latitude) * Math.sin (new_latitude));

    X2 = X1 + Math.atan2( Math.sin($\phi$) * Math.sin(d/R) * Math.cos(Y1),
  Math.cos(d/R) - Math.sin(Y1) * Math.sin(Y2));

    return (X2, Y2);
}
```

PRACTICAL IMPLEMENTATION

Complete implementation of the Android Mockin Service-Mobishark is shown below: A new project is created in android studio in Fig. 5.1 File->New->New Project.

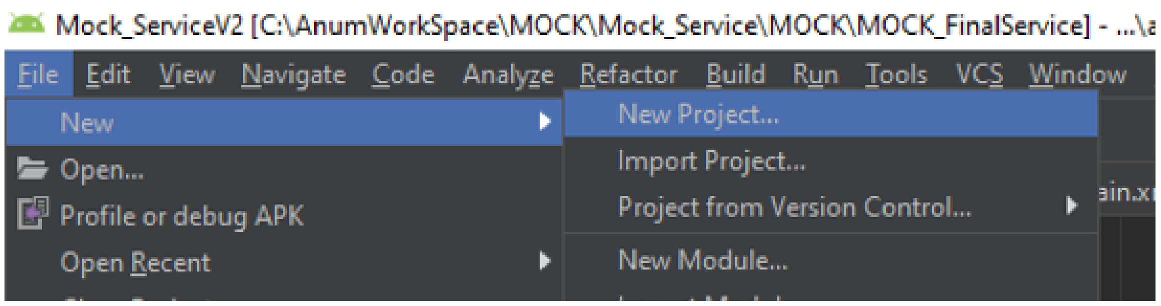

Mock_ServiceV2 [C:\AnumWorkSpace\MOCK\Mock_Service\MOCK\MOCK_FinalService] - ...\a
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window
New
Open...
Profile or debug APK
Open Recent
New Project...
Import Project...
Project from Version Control...
New Module...
ain.x

Figure 5.1: Creation of new project in Android Studio

Empty Activity is selected shown in Fig. 5.2

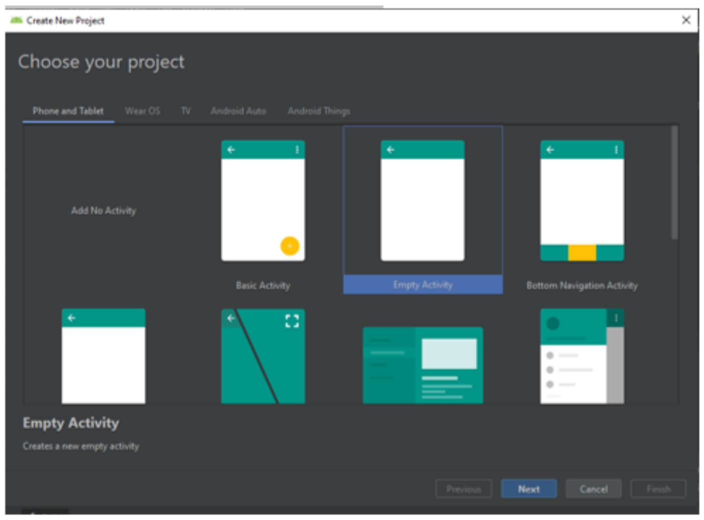
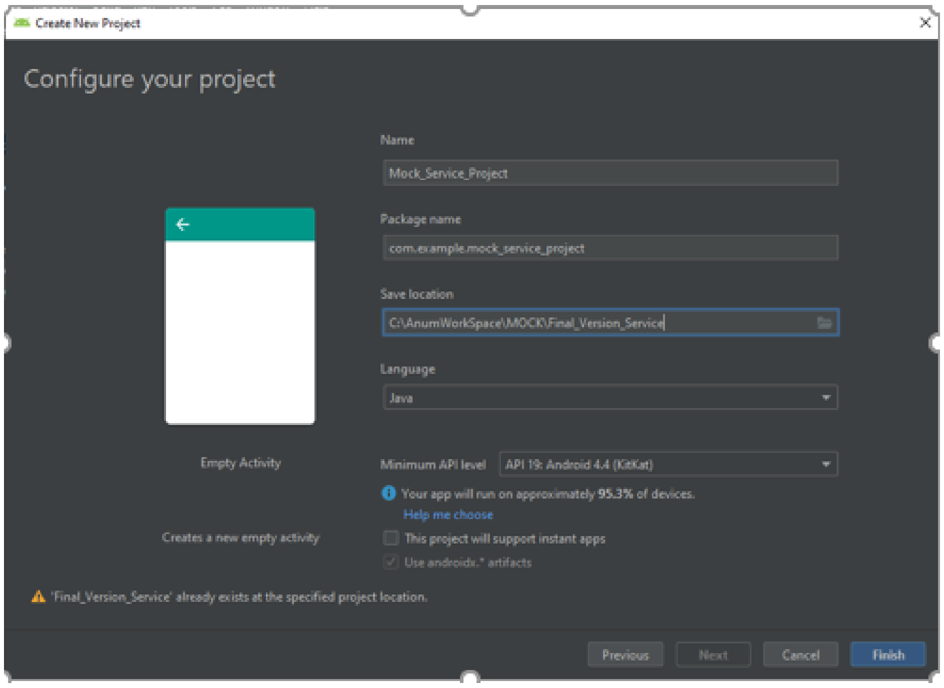


Figure 5.2: Creating an empty activity in Android Studio

In Fig. 5.3 Mock_Service_Project is the name selected for the project. Select the save location. For the following project C:\Anum\WorkSpace\Mock\Final_Version_Service is selected in C directory API 19: Android 4.4 (KitKat) Minimum API level selected. Press Finish to complete with the configuration of the project.

Complete hierarchy of the project is shown in Fig. 5.4



Configure your project

Name: Mock_Service_Project

Package name: com.example.mock_service_project

Save location: C:\AnumWorkSpace\MOCK\Final_Version_Service

Language: Java

Minimum API level: API 19: Android 4.4 (KitKat)

Your app will run on approximately 95.3% of devices.

[ ] This project will support instant apps

[✓] Use androidx.* artifacts

'Final_Version_Service' already exists at the specified project location.

Figure 5.3: Configuration settings for the project

5.1 AndroidManifest.xml File

Double click the AndroidManifest.xml file shown in Fig. 5.5 and add the following <uses-permissions>in that file shown in Fig. 5.6

- **android.permission.ACTION_BOOT_COMPLETED** is the broadcast intent received by applications after the system done with the booting.

- **android.permission.ACCESS_FINE_LOCATION** allows the application to access precise location.

- **android.permission.ACCESS_COARSE_LOCATION** allows the application to access approximate location.

Complete Code of AndroidManifest.xml file shown in Fig.5.7 and Fig.5.8

Receiver is created in AndroidManifest.xml to receive boot up events to restart service again in case if the system boots. Receiver is named as android:name=".StartUpBroadCastReceiver" with the action named android:name="android.intent.action.BOOT_COMPLETED . This will keep the BroadCastReceiver to be alive all the time even when activity is not running.

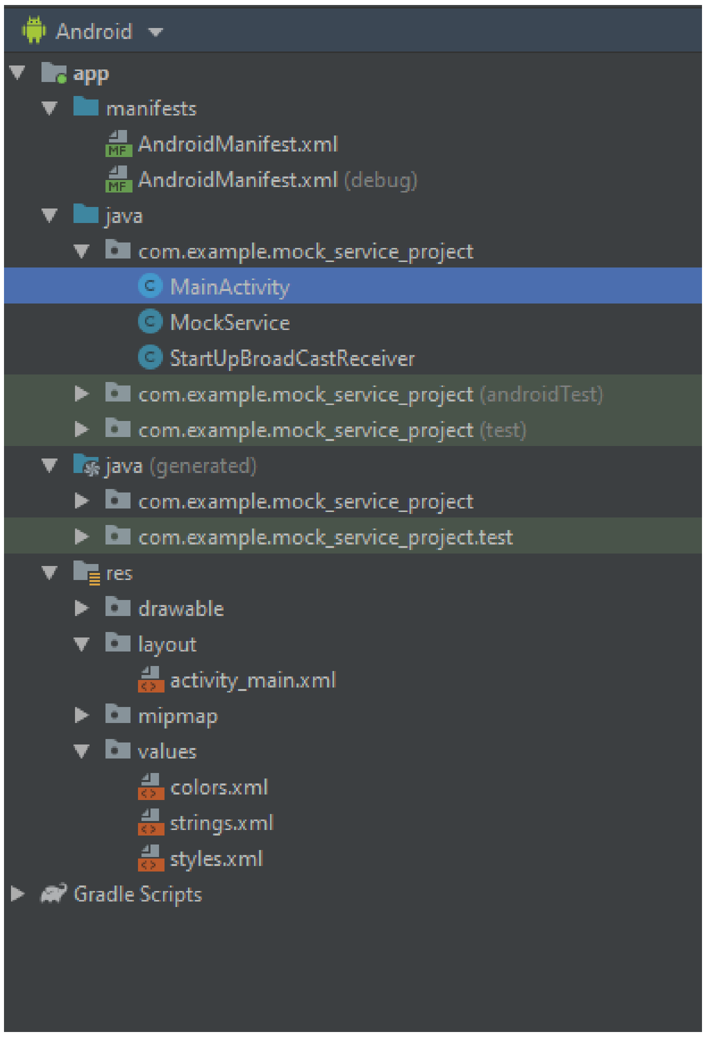

```
Android
▼ app
    ▼ manifests
          AndroidManifest.xml
          AndroidManifest.xml (debug)
    ▼ java
        ▼ com.example.mock_service_project
              MainActivity
              MockService
              StartUpBroadCastReceiver
        ▶ com.example.mock_service_project (androidTest)
        ▶ com.example.mock_service_project (test)
    ▼ java (generated)
        ▶ com.example.mock_service_project
        ▶ com.example.mock_service_project.test
    ▼ res
        ▶ drawable
        ▼ layout
              activity_main.xml
        ▶ mipmap
        ▼ values
              colors.xml
              strings.xml
              styles.xml
▶ Gradle Scripts
```

Figure 5.4: Complete hierarchy of the project

5.2 AndroidManifest.xml(debug) File

Complete code of AndroidManifest.xml(debug) Fig. 5.9 is shown below in Fig. 5.10. android.permission.ACCESS_MOCK_LOCATION allows the application to override the location or status returned by other real location sources such as GPS or location providers. Malicious location applications can use this to override the original location or status returned by real-location sources such as Network or GPS providers.

5.3 MainActivity File

Complete code of MainActivity Fig.5.11 is shown in Fig.5.12,Fig.5.13, Fig.5.14 and Fig.5.15. FusedLocationProviderClient is one of the location APIs in Google Play services. It manages the underlying location technology and provides a simple API so that you can specify requirements at a high level, like low power or high accuracy. It also optimizes the

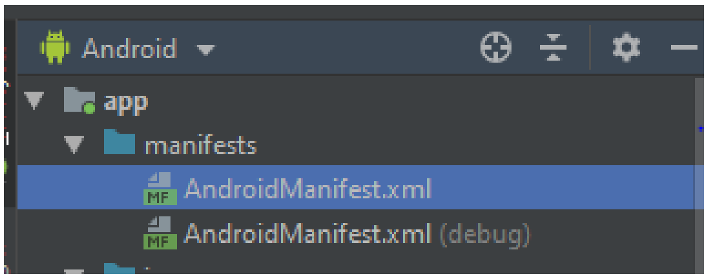
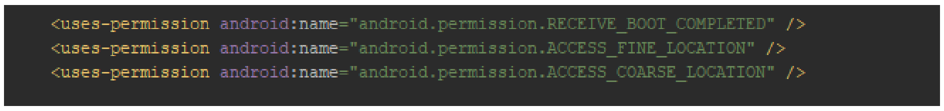


Figure 5.5: Android manifest file



```
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

Figure 5.6: Uses permissions in android manifest file

device's battery power.

In order to launch and run MockService in background **Intent** is created which will help service to run outside the application in a background process. When the application is started; user is been asked to grant the permission to access the location(ACCESS_FINE_LOCATION). If user has granted the right to access the location; then the mock service will be launched to run in background which will mock the original location coordinates.

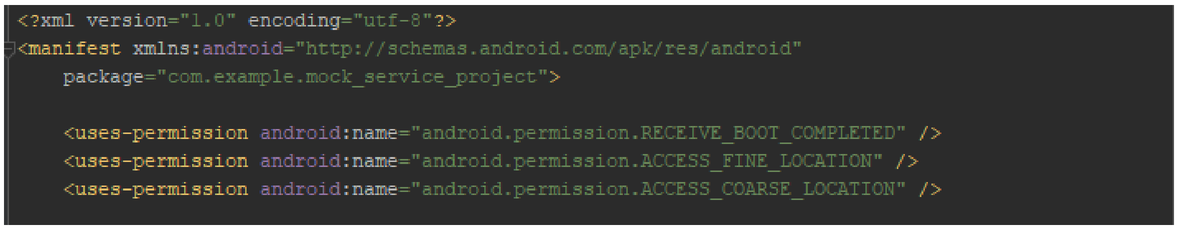
5.4 Mock Service File

Create a new service by selecting com.example.mock_service_project->New->Java Class and name new class as MockService and press OK to configure the class. Steps are shown in Fig.5.16 and Fig.5.17

Complete implementation of the MockService class is shown in Fig.5.18, Fig.5.19, Fig.5.20, Fig.5.21 and Fig.5.22

FusedLocationClient.setMockMode(true) sets the location provider to be in mock mode. FusedLocationClient.setMockLocation(mockLocation) sets the mock location to be used for the location providers (network or GPS). This location will be used in place of any real locations from the underlying providers. New logitude and new latitude are calculated with the help of the following formulas.

- **New Latitude** = Math.asin (Math.sin (original_latitude) * Math.cos (distance/Radius) + Math.cos (original_laltitude) * Math.sin (random_radius /Radius) * Math.cos (ran-



```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.mock_service_project">

    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

Figure 5.7: Android manifest complete code (a)

dom_angle))

- **New Longitude** = original_longitude + Math.atan2 (Math.sin (random_angle) * Math.sin (random_radius /Radius) * Math.cos (original_latitude) , Math.cos (random_radius /Radius)- Math.sin (original_latitude) * Math.sin (new_latitude))

These values of **New Latitude** and **New Longitude** are pass to the methods of .setLatitude() and .setLongitude(). All the attributes of the mockLocation are shown below:

mockLocation.setLatitude(new_latitude);

mockLocation.setLongitude(new_longitude);

mockLocation.setAltitude(0);

mockLocation.setAccuracy(5);

mockLocation.setTime(System.currentTimeMillis());

mockLocation.setElapsedRealtimeNanos(42);

After setting all the attributes of mockLocation ; this mockLocation is pass to the method of .setMockLocation() as shown below:

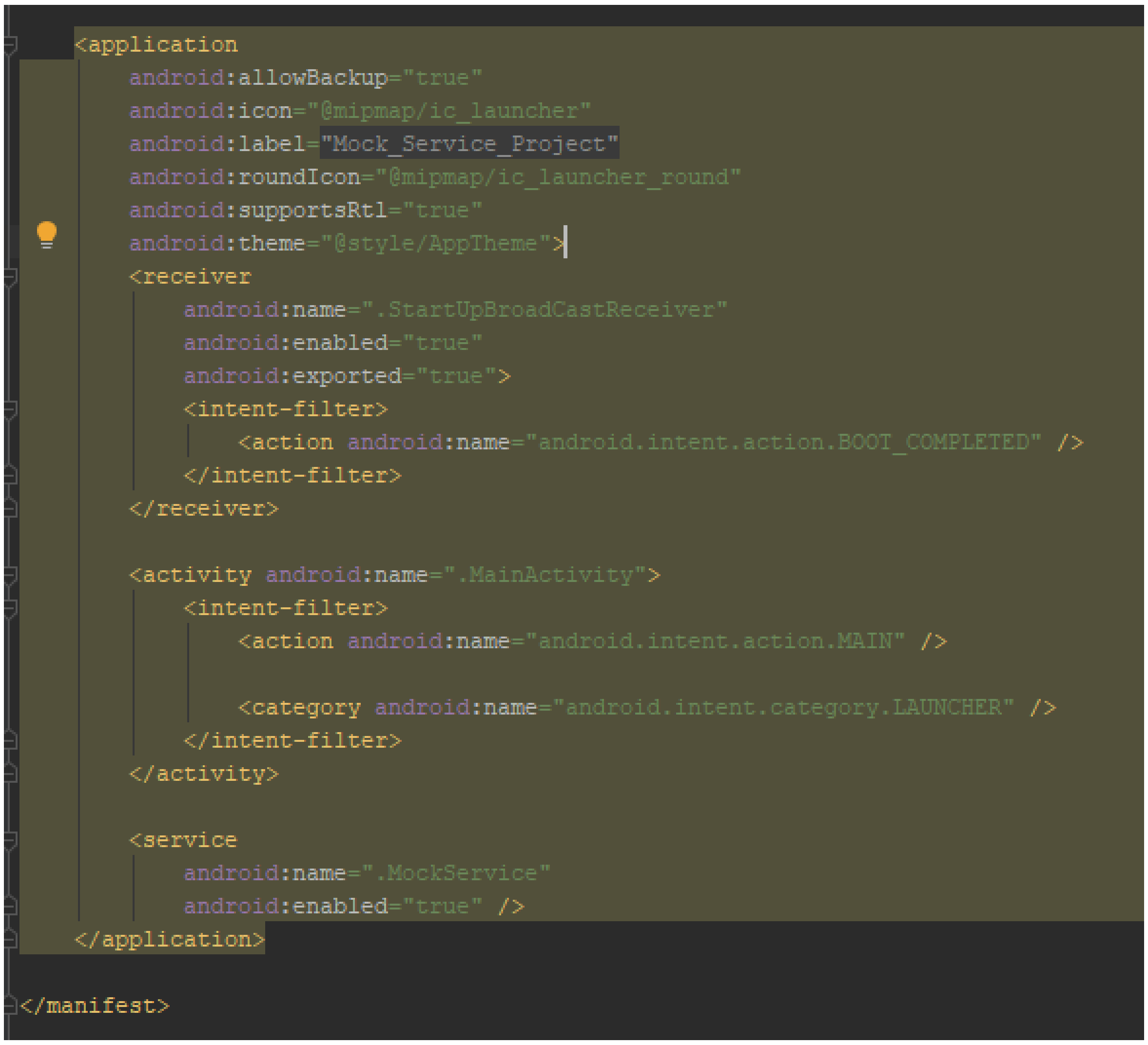
FusedLocationClient.setMockLocation(mockLocation);

new MockService() is called on onDestroy() so that service will be restarted the moment it destroys.

5.5 StartUpBroadCastReceiver File

Create a BroadcastReceiver Fig.5.23 by creating com.example.mock_service_project->New->Other->BroadcastReceiver. Name the BroadcastReceiver as StartUpBroadCastReceiver as shown in Fig.5.24

Complete implementation of the StartUpBroadCastReceiver Fig.5.25 is shown in Fig.5.26.



```
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Mock_Service_Project"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <receiver
            android:name=".StartUpBroadCastReceiver"
            android:enabled="true"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED" />
            </intent-filter>
        </receiver>

        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <service
            android:name=".MockService"
            android:enabled="true" />
    </application>

</manifest>
```

Figure 5.8: Android manifest complete code (b)

Broadcast receiver is an android component which allows you to send or receive android application or system events. All the registered receivers are notified by the android runtime once event happens through broadcast receivers. In this Mobishark application has registered for the ACTION_BOOT_COMPLETED; which will be fired the moment Android system has done or completed with the boot process. onReceive() method is called as soon as the broadcast receiver receives the notification for the event for which it has been registered. Two arguments of the onReceive() method are context and intent. Context is used to access additional information, or to start activities or services. Intent object is used to register the receiver.

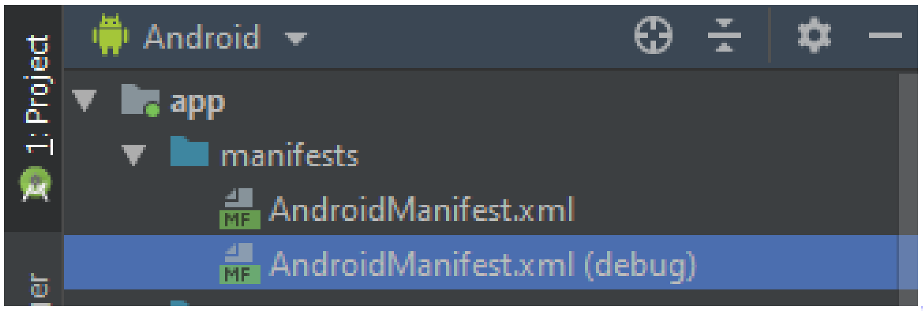
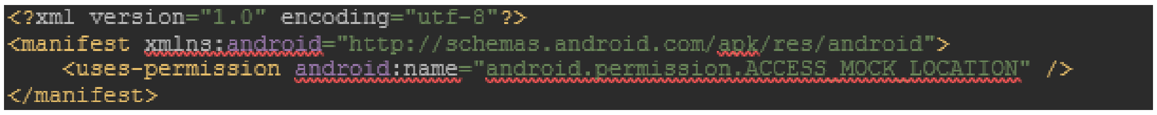


Figure 5.9: Android manifest debug file



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
    <uses-permission android:name="android.permission.ACCESS_MOCK_LOCATION" />
</manifest>
```

Figure 5.10: Permissions in android manifest debug file

5.6 Layout File

Complete implementation of the text and design code of activity_main.xml Fig.5.27 is shown in Fig.5.28 and Fig.5.29 respectively. Layout of the Mobishark application is kept very simple to make it more efficient in terms of performance; instead of loading heavy geographic maps which will consume a lot of Android resources.

5.7 Values

5.7.1 Colors.xml File

Code for the colors.xml Fig.5.30 is shown in Fig.5.31

5.7.2 Strings.xml File

Code for the strings.xml Fig.5.32 is shown in Fig.5.33

5.7.3 Styles.xml File

Code for the styles.xml Fig.5.34 is shown in Fig.5.35

5.8 Build.gradle(Module: app) File

Code for the build.gradle(Module: app) Fig.5.36 is shown in Fig.5.37. compileSdkVersion is 29 and targetSdkVersion is 29 while buildToolsVersion is "29.0.2".

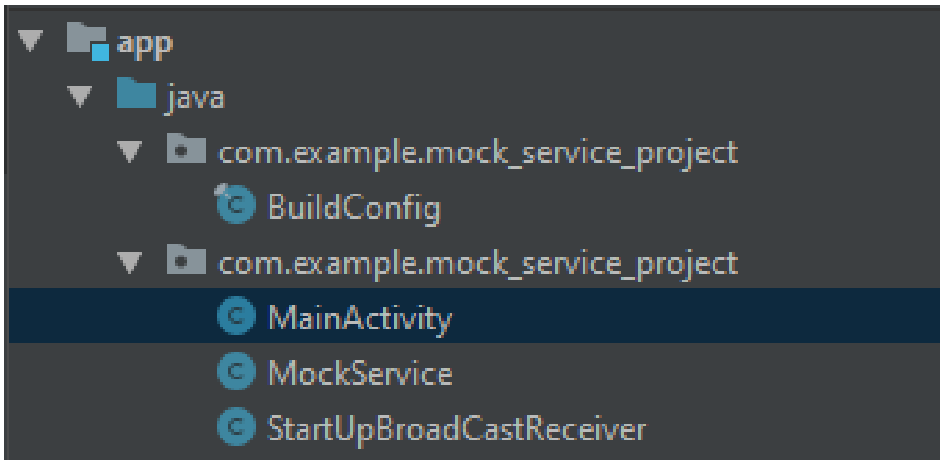
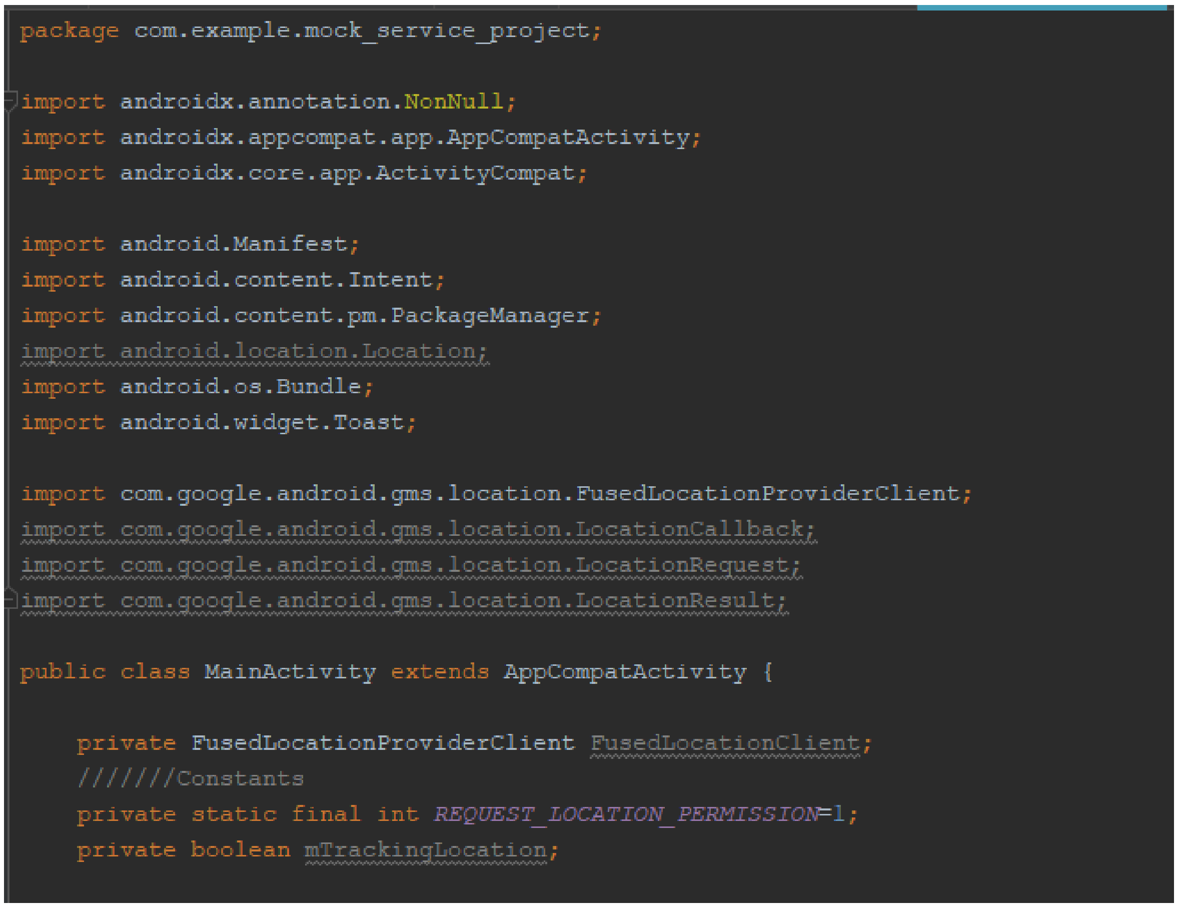
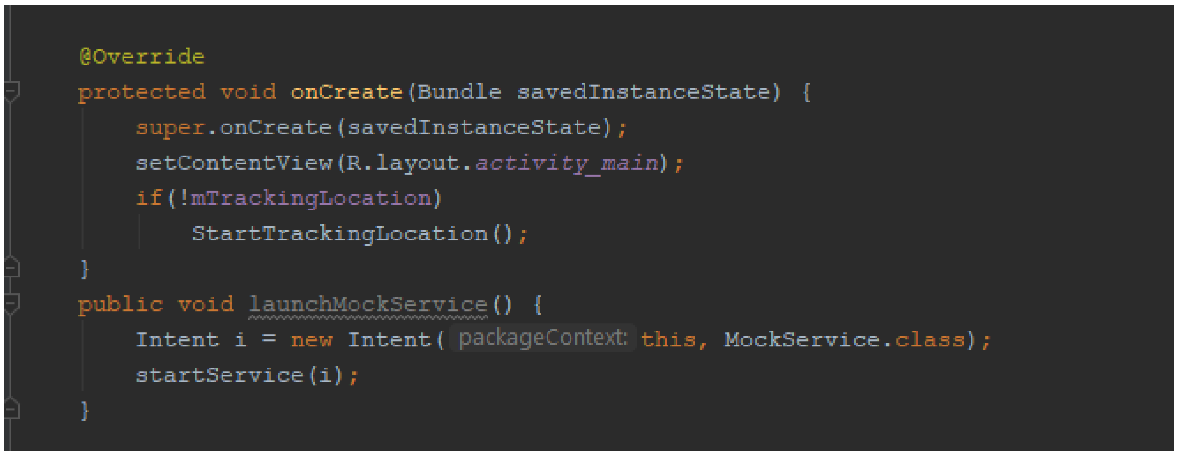


Figure 5.11: Main activity file



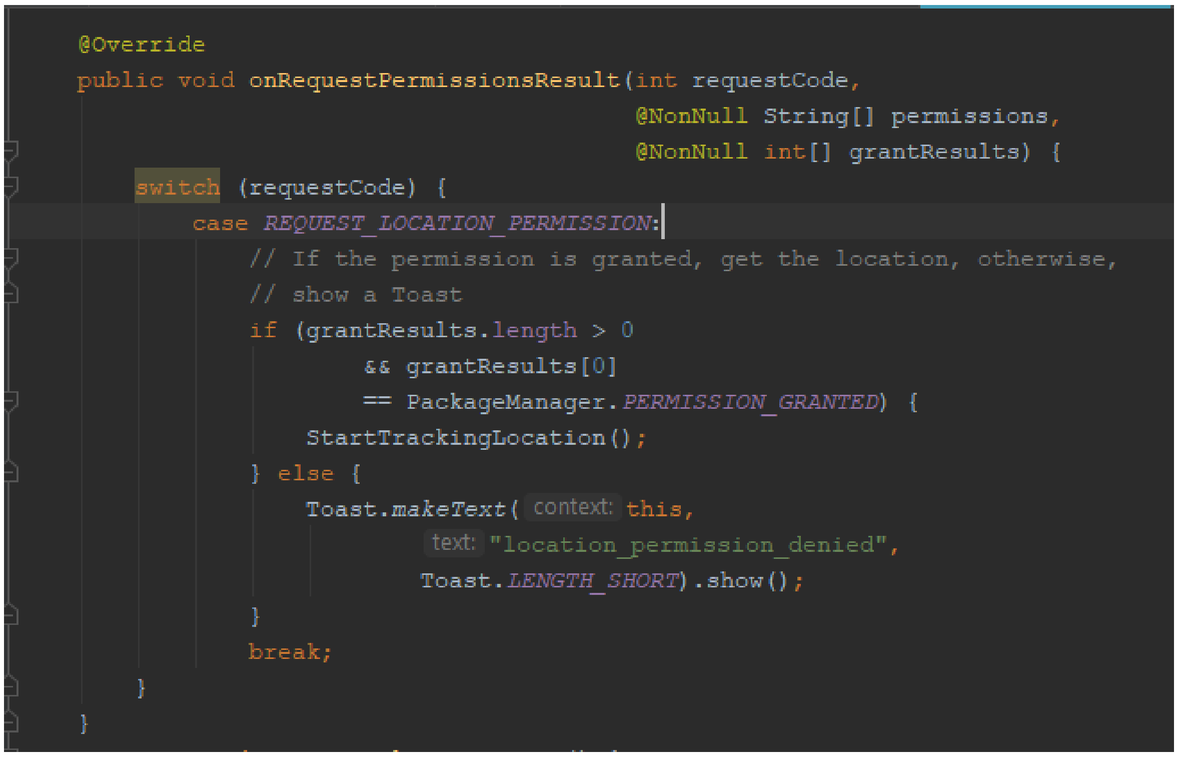
```
package com.example.mock_service_project;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;

import android.Manifest;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.location.Location;
import android.os.Bundle;
import android.widget.Toast;

import com.google.android.gms.location.FusedLocationProviderClient;
import com.google.android.gms.location.LocationCallback;
import com.google.android.gms.location.LocationRequest;
import com.google.android.gms.location.LocationResult;

public class MainActivity extends AppCompatActivity {

    private FusedLocationProviderClient FusedLocationClient;
    ///////Constants
    private static final int REQUEST_LOCATION_PERMISSION=1;
    private boolean mTrackingLocation;
```

Figure 5.12: Complete code of main activity (a)



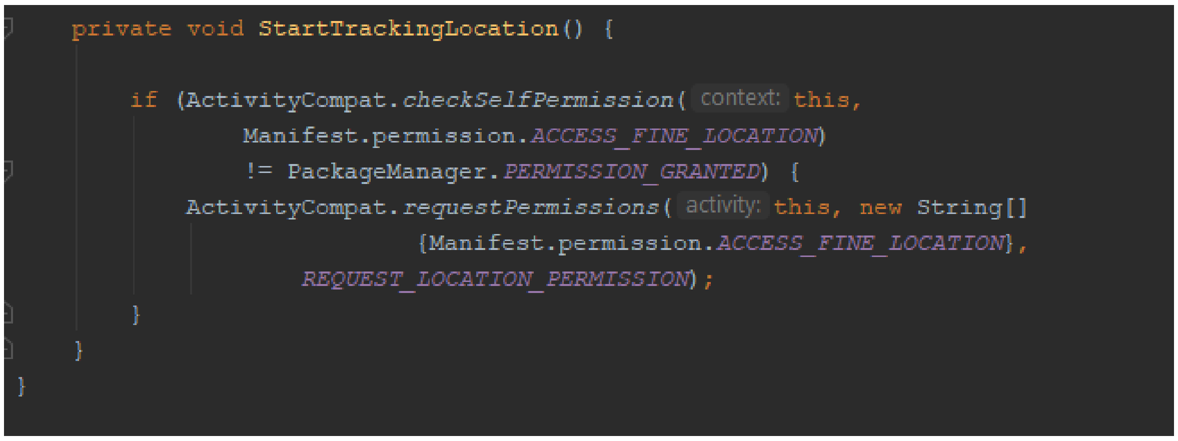
```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    if(!mTrackingLocation)
        StartTrackingLocation();
}
public void launchMockService() {
    Intent i = new Intent( packageContext: this, MockService.class);
    startService(i);
}
```

Figure 5.13: Complete code of main activity (b)



```java
@Override
public void onRequestPermissionsResult(int requestCode,
                                       @NonNull String[] permissions,
                                       @NonNull int[] grantResults) {
    switch (requestCode) {
        case REQUEST_LOCATION_PERMISSION:
            // If the permission is granted, get the location, otherwise,
            // show a Toast
            if (grantResults.length > 0
                    && grantResults[0]
                    == PackageManager.PERMISSION_GRANTED) {
                StartTrackingLocation();
            } else {
                Toast.makeText( context: this,
                         text: "location_permission_denied",
                        Toast.LENGTH_SHORT).show();
            }
            break;
    }
}
```

Figure 5.14: Complete code of main activity (c)



```
    private void StartTrackingLocation() {

        if (ActivityCompat.checkSelfPermission( context: this,
                Manifest.permission.ACCESS_FINE_LOCATION)
                != PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions( activity: this, new String[]
                            {Manifest.permission.ACCESS_FINE_LOCATION},
                    REQUEST_LOCATION_PERMISSION);
        }
    }
}
```

Figure 5.15: Complete code of main activity (d)

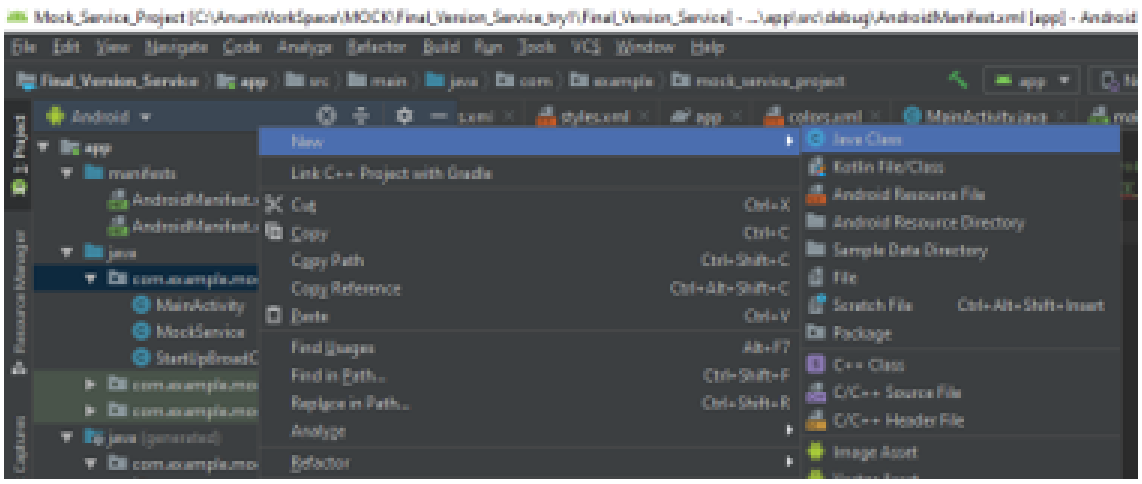
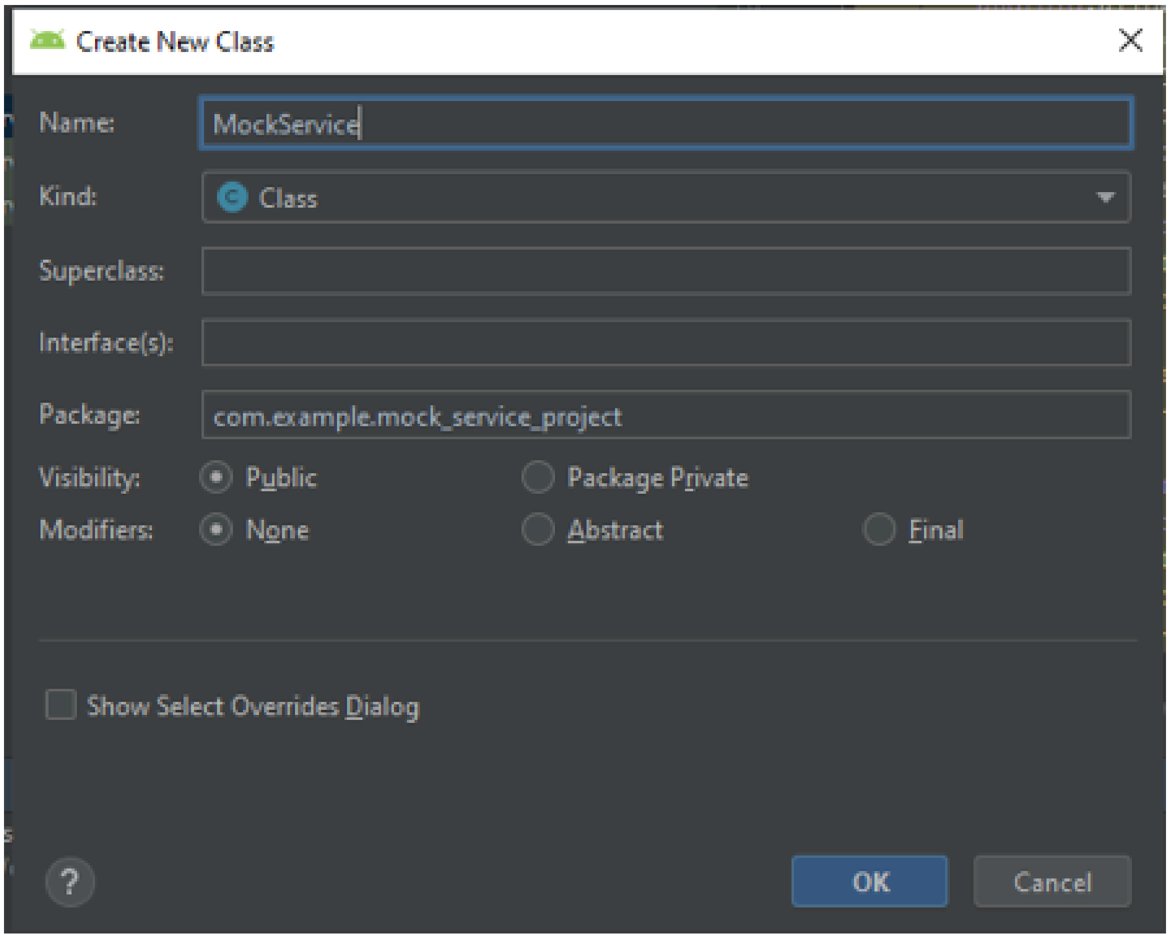

Mock_Service_Project [C:\AnamWorkSpace\MOCK\Final_Version_Service_try1\Final_Version_Service] - ...\app\src\debug\AndroidManifest.xml [app] - Android
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Final_Version_Service app src main java com example mock_service_project
Android
app
manifests
AndroidManifest.x
AndroidManifest.x
java
com.example.mo
MainActivity
MockService
StartUpBroadC
com.example.mo
com.example.mo
java (generated)
com.example.mo
New
Link C++ Project with Gradle
Cut Ctrl+X
Copy Ctrl+C
Copy Path Ctrl+Shift+C
Copy Reference Ctrl+Alt+Shift+C
Paste Ctrl+V
Find Usages Alt+F7
Find in Path... Ctrl+Shift+F
Replace in Path... Ctrl+Shift+R
Analyze
Refactor
Java Class
Kotlin File/Class
Android Resource File
Android Resource Directory
Sample Data Directory
File
Scratch File Ctrl+Alt+Shift+Insert
Package
C++ Class
C/C++ Source File
C/C++ Header File
Image Asset
styles.xml
colors.xml
MainActivity.java

Figure 5.16: Creation of mock service (a)



Name: MockService

Kind: Class

Superclass:

Interface(s):

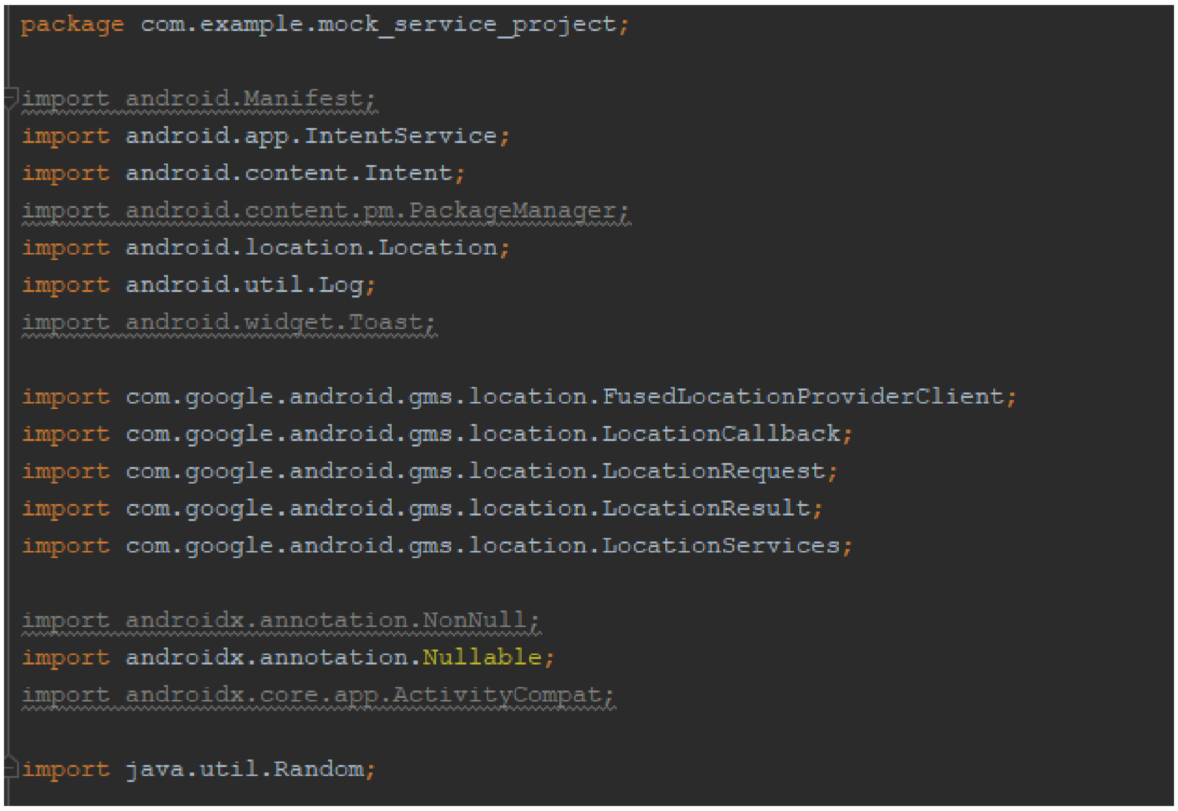
Package: com.example.mock_service_project

Visibility: [✓ Public] Package Private

Modifiers: [✓ None] Abstract Final

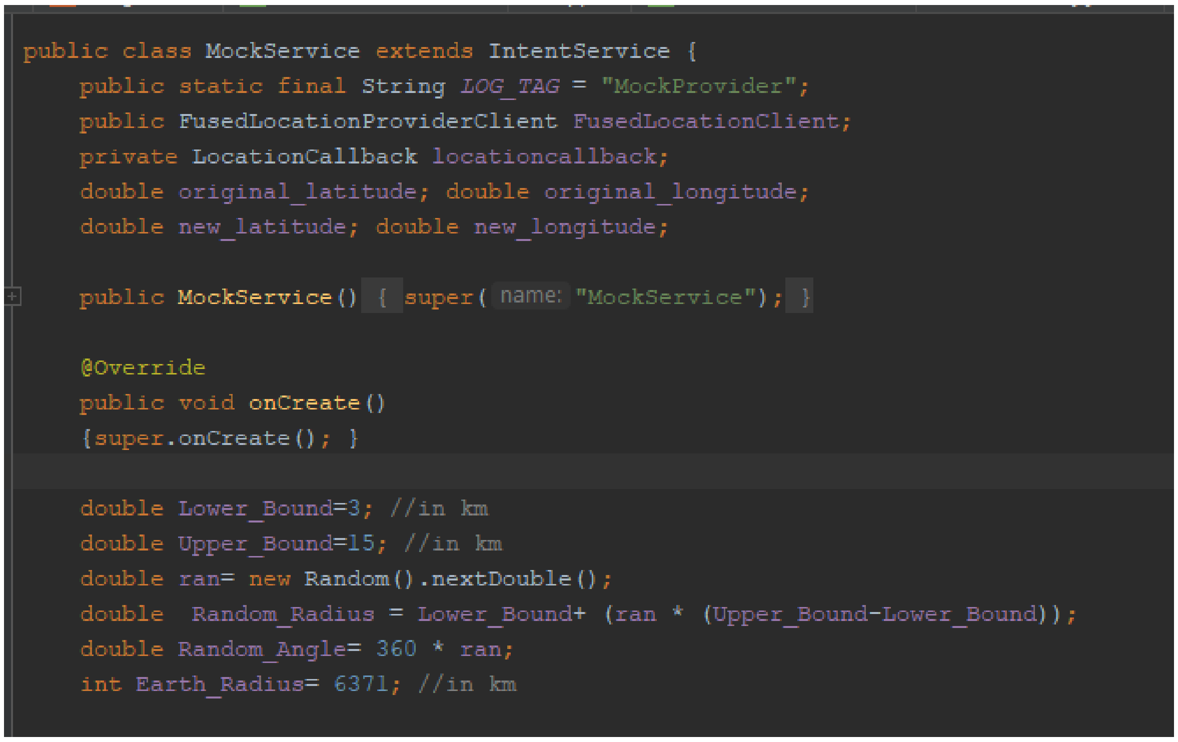
Show Select Overrides Dialog

Figure 5.17: Creation of mock service (b)



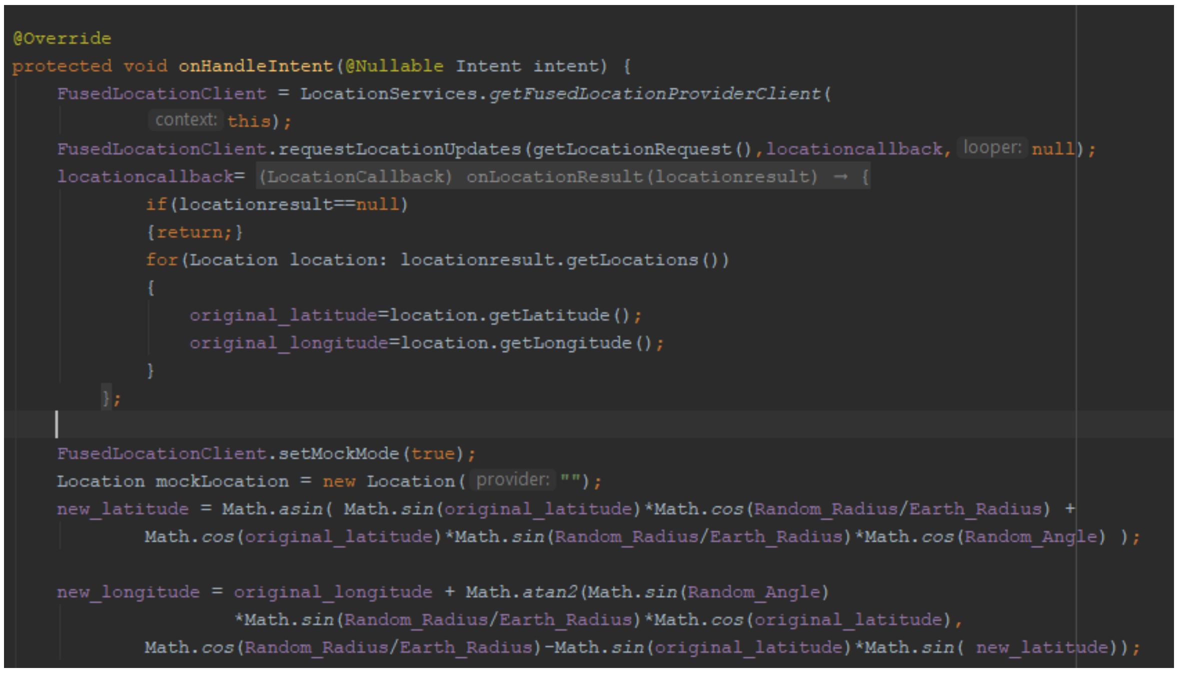
```
package com.example.mock_service_project;

import android.Manifest;
import android.app.IntentService;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.location.Location;
import android.util.Log;
import android.widget.Toast;

import com.google.android.gms.location.FusedLocationProviderClient;
import com.google.android.gms.location.LocationCallback;
import com.google.android.gms.location.LocationRequest;
import com.google.android.gms.location.LocationResult;
import com.google.android.gms.location.LocationServices;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.core.app.ActivityCompat;

import java.util.Random;
```

Figure 5.18: Code of mock service (a)



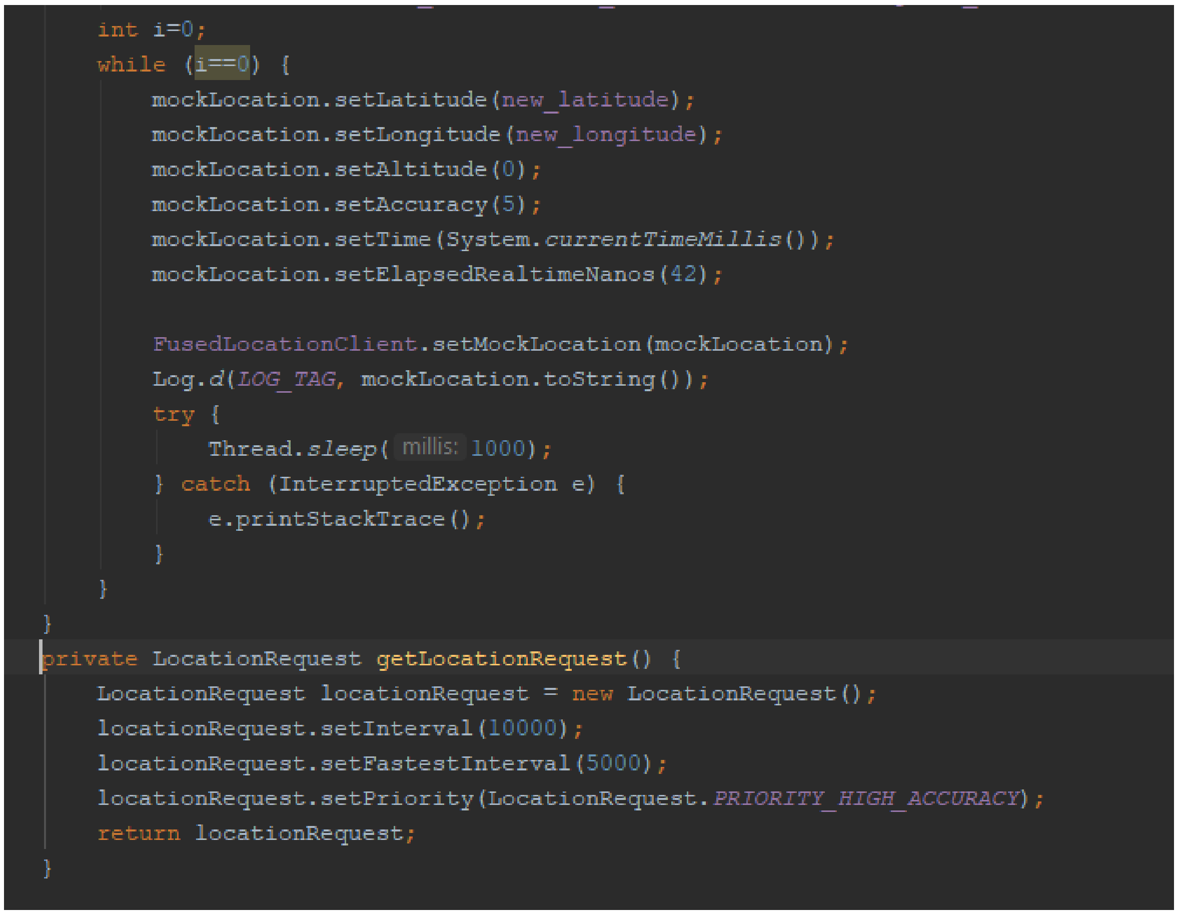
```
public class MockService extends IntentService {
    public static final String LOG_TAG = "MockProvider";
    public FusedLocationProviderClient FusedLocationClient;
    private LocationCallback locationcallback;
    double original_latitude; double original_longitude;
    double new_latitude; double new_longitude;

    public MockService() { super( name: "MockService"); }

    @Override
    public void onCreate()
    {super.onCreate(); }


    double Lower_Bound=3; //in km
    double Upper_Bound=15; //in km
    double ran= new Random().nextDouble();
    double  Random_Radius = Lower_Bound+ (ran * (Upper_Bound-Lower_Bound));
    double Random_Angle= 360 * ran;
    int Earth_Radius= 6371; //in km
```

Figure 5.19: Code of mock service (b)



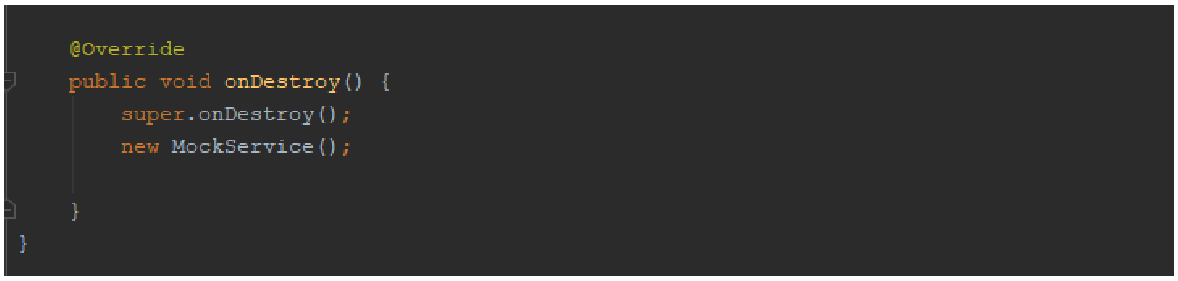
```
@Override
protected void onHandleIntent(@Nullable Intent intent) {
    FusedLocationClient = LocationServices.getFusedLocationProviderClient(
            context: this);
    FusedLocationClient.requestLocationUpdates(getLocationRequest(),locationcallback, looper: null);
    locationcallback= (LocationCallback) onLocationResult(locationresult) → {
            if(locationresult==null)
            {return;}
            for(Location location: locationresult.getLocations())
            {
                original_latitude=location.getLatitude();
                original_longitude=location.getLongitude();
            }
        };

    FusedLocationClient.setMockMode(true);
    Location mockLocation = new Location( provider: "");
    new_latitude = Math.asin( Math.sin(original_latitude)*Math.cos(Random_Radius/Earth_Radius) +
            Math.cos(original_latitude)*Math.sin(Random_Radius/Earth_Radius)*Math.cos(Random_Angle) );

    new_longitude = original_longitude + Math.atan2(Math.sin(Random_Angle)
                    *Math.sin(Random_Radius/Earth_Radius)*Math.cos(original_latitude),
            Math.cos(Random_Radius/Earth_Radius)-Math.sin(original_latitude)*Math.sin( new_latitude));
```

Figure 5.20: Code of mock service (c)



```
    int i=0;
    while (i==0) {
        mockLocation.setLatitude(new_latitude);
        mockLocation.setLongitude(new_longitude);
        mockLocation.setAltitude(0);
        mockLocation.setAccuracy(5);
        mockLocation.setTime(System.currentTimeMillis());
        mockLocation.setElapsedRealtimeNanos(42);

        FusedLocationClient.setMockLocation(mockLocation);
        Log.d(LOG_TAG, mockLocation.toString());
        try {
            Thread.sleep( millis: 1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
private LocationRequest getLocationRequest() {
    LocationRequest locationRequest = new LocationRequest();
    locationRequest.setInterval(10000);
    locationRequest.setFastestInterval(5000);
    locationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
    return locationRequest;
}
```

Figure 5.21: Code of mock service (d)



```
    @Override
    public void onDestroy() {
        super.onDestroy();
        new MockService();

    }
}
```

Figure 5.22: Code of mock service (e)

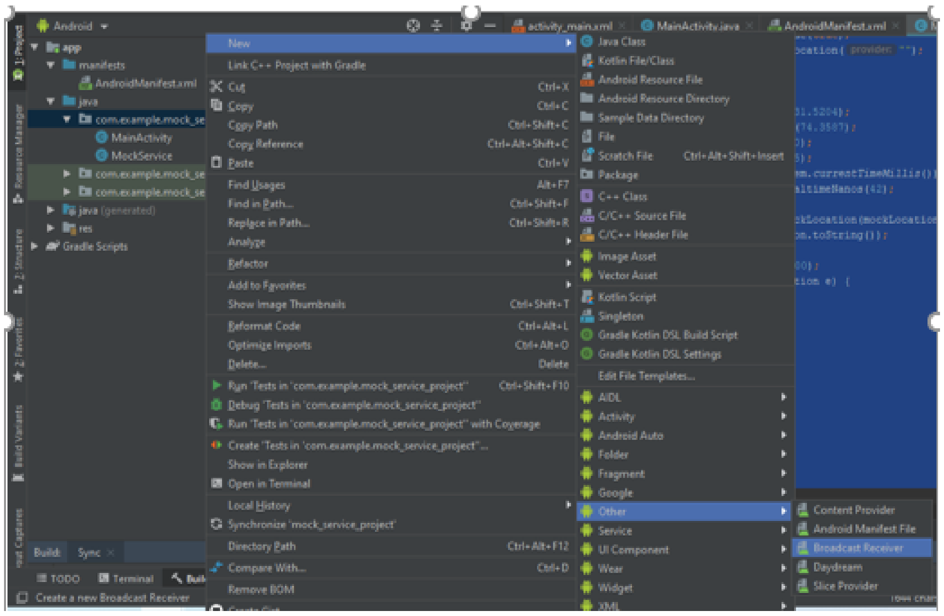
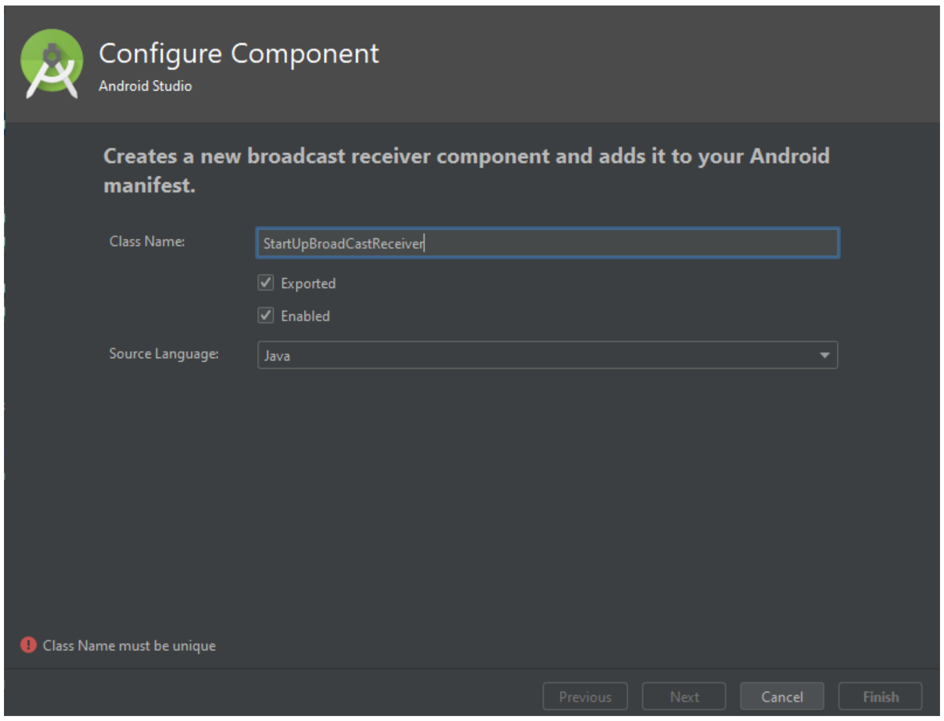


Figure 5.23: Creation of BroadCastReceiver (a)



Configure Component
Android Studio

Creates a new broadcast receiver component and adds it to your Android manifest.

Class Name: StartUpBroadCastReceiver

[✓] Exported

[✓] Enabled

Source Language: Java

Class Name must be unique

Previous Next Cancel Finish

Figure 5.24: Creation of BroadCastReceiver (b)

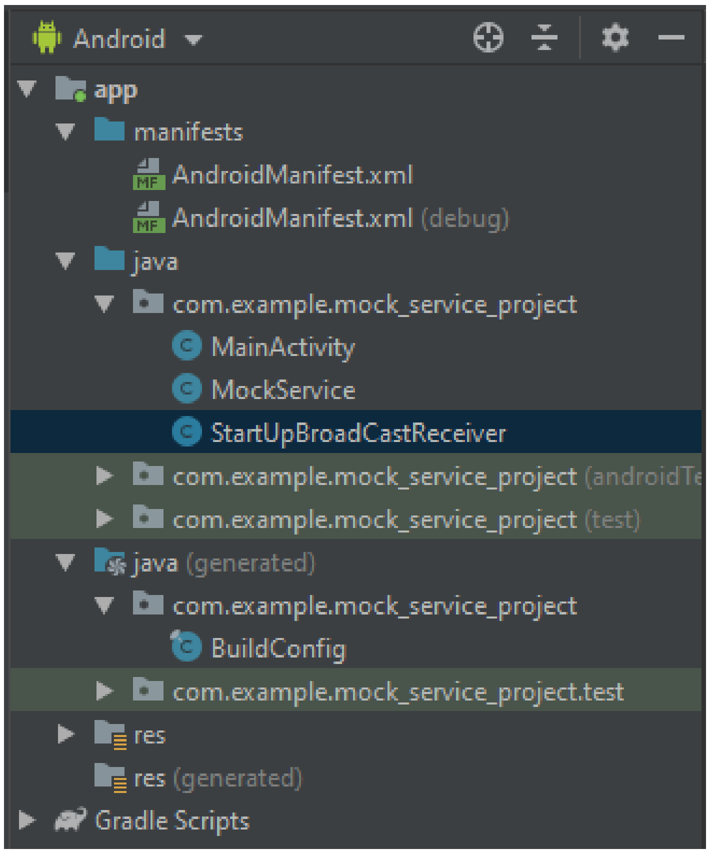
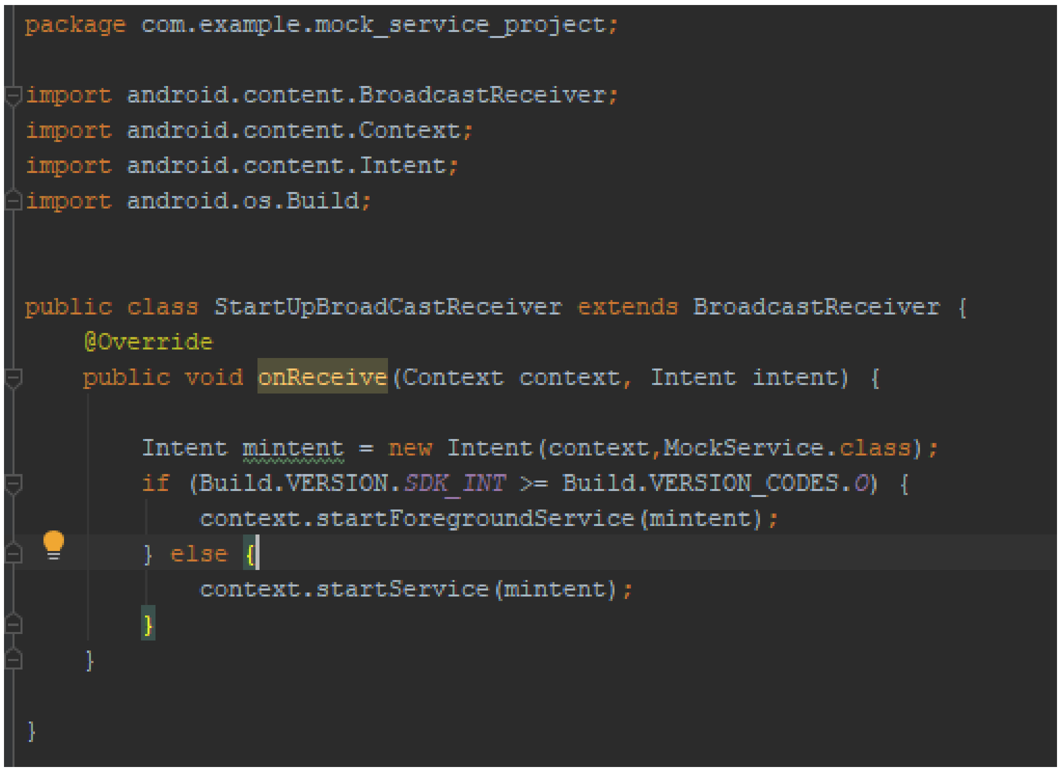

Android
app
manifests
AndroidManifest.xml
AndroidManifest.xml (debug)
java
com.example.mock_service_project
MainActivity
MockService
StartUpBroadCastReceiver
com.example.mock_service_project (androidTe
com.example.mock_service_project (test)
java (generated)
com.example.mock_service_project
BuildConfig
com.example.mock_service_project.test
res
res (generated)
Gradle Scripts

Figure 5.25: StartUpBroadCastReceiver



```
package com.example.mock_service_project;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Build;


public class StartUpBroadCastReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {

        Intent mintent = new Intent(context,MockService.class);
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            context.startForegroundService(mintent);
        } else {
            context.startService(mintent);
        }
    }

}
```

Figure 5.26: StartUpBroadCastReceiver code

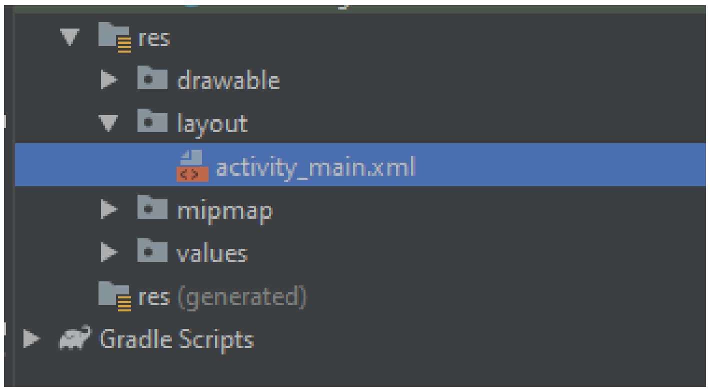
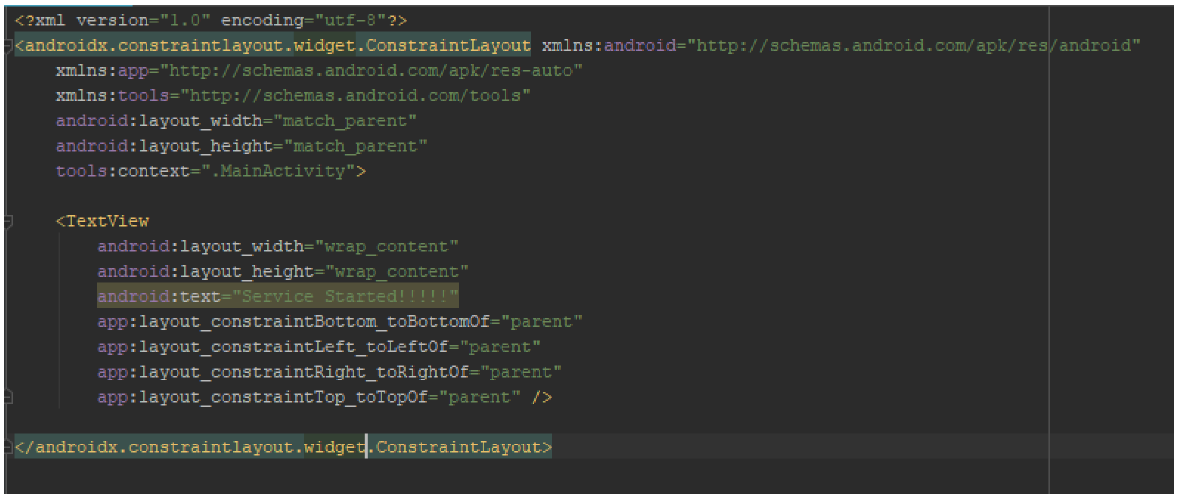


Figure 5.27: Activity_main.xml file



```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Service Started!!!!!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Figure 5.28: Activity_main.xml text code

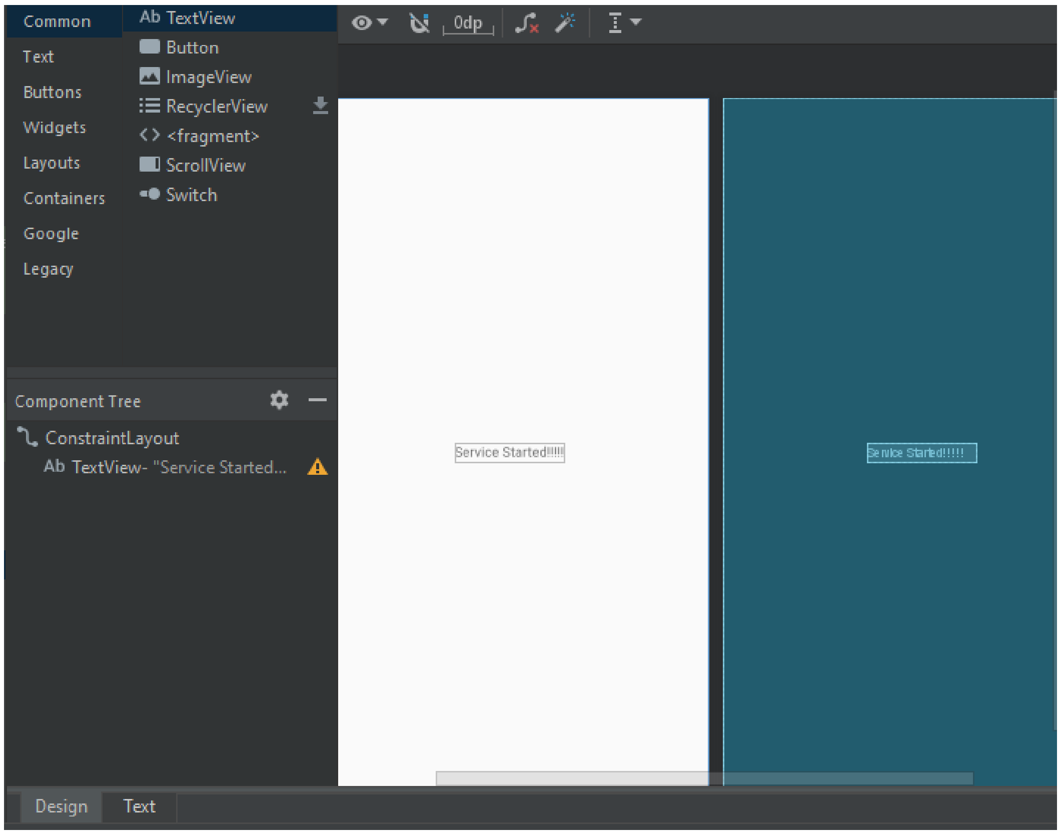


Figure 5.29: Activity_main.xml design code

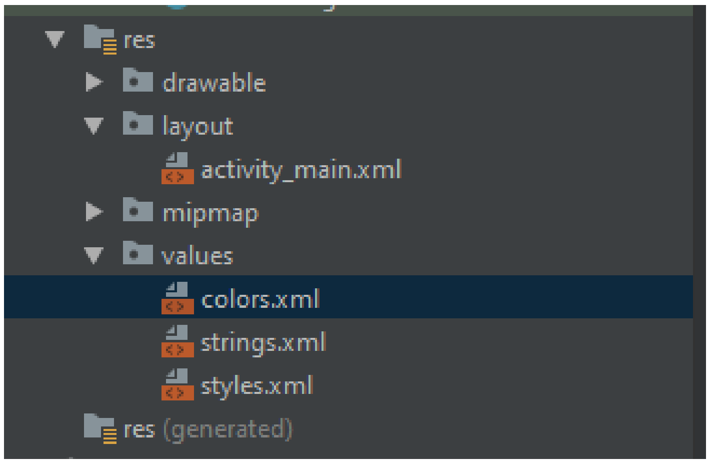
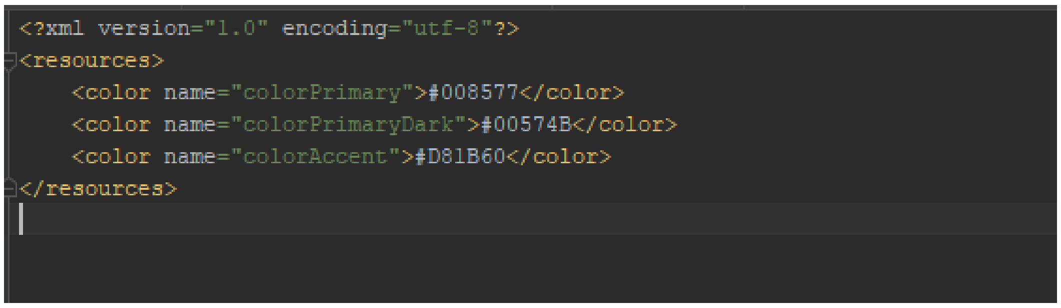


Figure 5.30: Colors.xml file



```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#008577</color>
    <color name="colorPrimaryDark">#00574B</color>
    <color name="colorAccent">#D81B60</color>
</resources>
```

Figure 5.31: Colors.xml code file

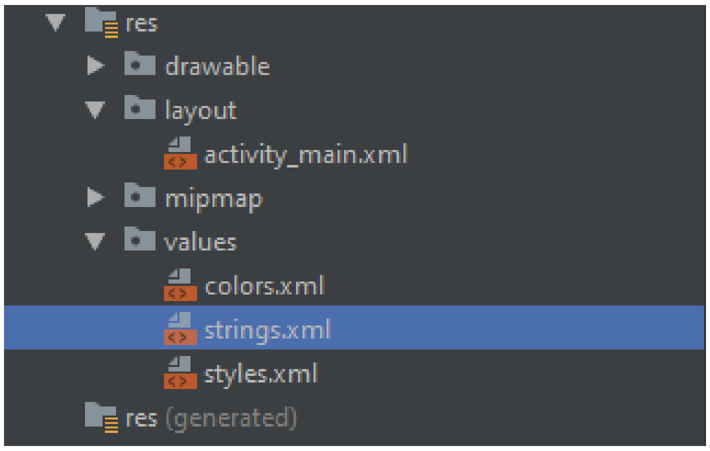
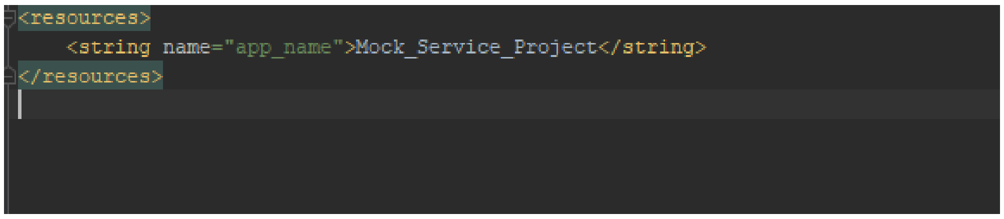


Figure 5.32: Strings.xml file



```
<resources>
    <string name="app_name">Mock_Service_Project</string>
</resources>
```

Figure 5.33: Strings.xml code file

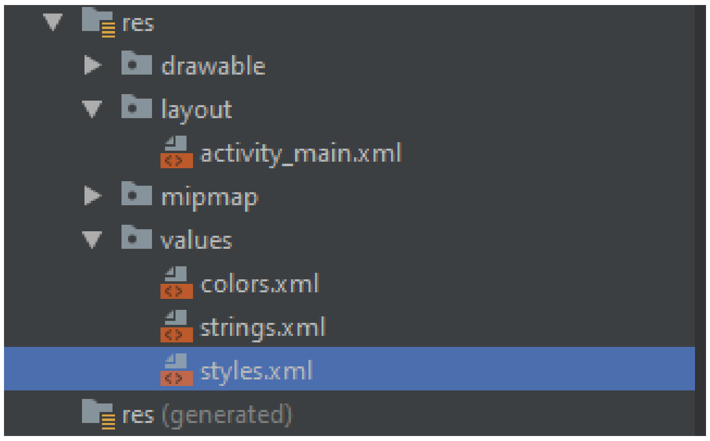
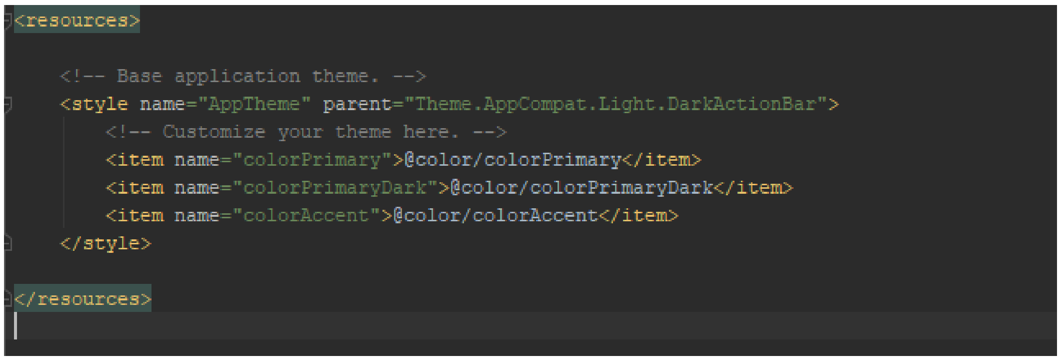


Figure 5.34: Styles.xml file



```
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

</resources>
```

Figure 5.35: Styles.xml code file

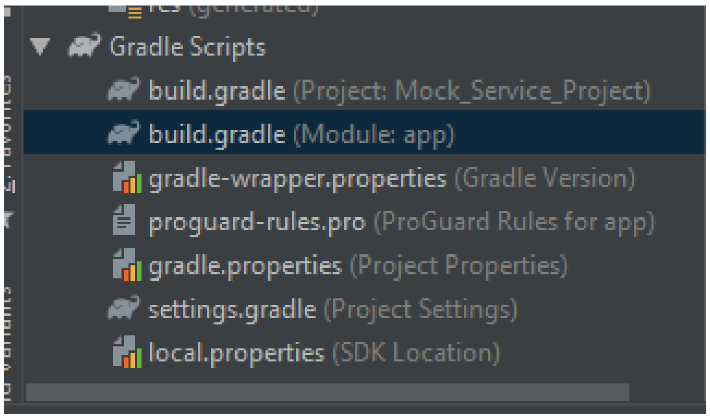
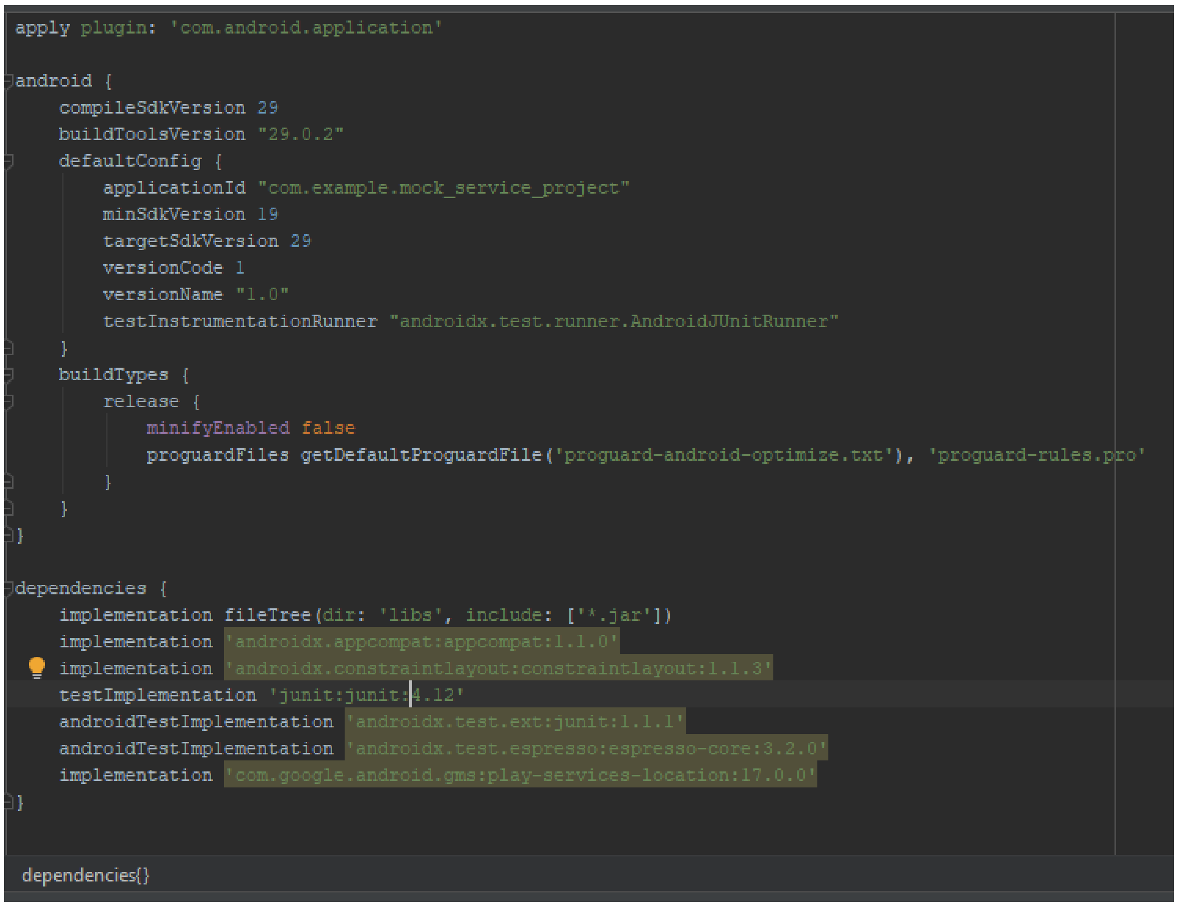

Gradle Scripts
build.gradle (Project: Mock_Service_Project)
build.gradle (Module: app)
gradle-wrapper.properties (Gradle Version)
proguard-rules.pro (ProGuard Rules for app)
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)

Figure 5.36: Build.gradle(Module: app) file



```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 29
    buildToolsVersion "29.0.2"
    defaultConfig {
        applicationId "com.example.mock_service_project"
        minSdkVersion 19
        targetSdkVersion 29
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'
    implementation 'com.google.android.gms:play-services-location:17.0.0'
}
```

Figure 5.37: Build.gradle(Module: app) code file

Chapter 6

EVALUATION OF PROPOSED MECHANISM

Efficiency of the proposed mechanism has been verified on different Android applications like Google maps, maps.me, Careem, Bykea, Facebook, Foursquare etc. for a considerable amount of time. These applications persistently showed mock location throughout the testing duration. Detailed traffic analysis also revealed that the mock coordinates were being sent to different servers including Google. Taken together, these results suggest that the proposed methodology can trick Service Providers to get inaccurate or mock locations instead of tracking original coordinates.

The proposed framework was also compared with others applications under the testing scenarios discussed in Section IV. The single most striking observation emerging out of this comparison was that under severe stress and load, our application showed promising results with high spoofing accuracy and better reliability when it comes to testing in movement simulation. We also found that our proposed application offers more persistence with limited permissions to install. We tested all applications on a variety of Android smartphones bearing different hardware configurations listed in Table 6.1. We also tested our application under stringent memory pressure and found out that it outperforms other GPS spoofing applications with better performance, accuracy, and reliability. We also traversed outside buildings (multiple floors) to break connection between GPS satellites and smartphones for testing purposes and found our application to be a better solution as compared to others. The proposed framework for tracking pilferage of location data along with its safeguard by spoofing will be enhanced further along with more in-depth and rigorous evaluation after further enhancing the current approach with more modules and functionality.

Table 6.1: Smartphone Hardware Configurations

Manufacturer	**Specs**
Samsung I9500 Galaxy S4	Octa-core (4x1.6 GHz Cortex-A15 4x1.2 GHz Cortex-A7), 2 GB Ram
Samsung G900f Galaxy S5	Quad-core 2.5 GHz Krait 400, 2 Gb Ram
Huawei P8lite	Octa-core 1.2 GHz Cortex-A53, 2 Gb Ram
Samsung Galaxy J3 (2016)	Quad-core 1.5 GHz Cortex-A7, 1.5 Gb Ram

Chapter 7

CONCLUSION AND FUTURE WORK DIRECTIONS

This paper gives the detailed analysis of analysed Google apps which keep their users under surveillance and send stored location data at different intervals of time, irrespective of the user's location sharing settings. This paper further presents an insight on the performance analysis of fake GPS location applications which fail to secure the real location coordinates. The proposed solution guards the real location coordinates by keeping the mock location within span-radius which appears to be realistic, hence adjusting the location coordinates automatically. The proposed algorithm is efficient in terms of performance and results. This may be considered as a promising aspect to preserve user's real location coordinates on an Android smartphone, practically. Future work will concentrate on the kernel level solution to completely ensure that the location coordinates are not being compromised at all.

BIBLIOGRAPHY

[1] U.S. location-based service users 2013-2018 Published by Statista Research Department, Jun 17, 2015.

[2] "Google Maps", Play.google.com. [Accessed: 15- Feb- 2020].

[3] "Fitbit", Play.google.com. [Accessed: 15- Feb- 2020].

[4] "Facebook", Play.google.com. [Accessed: 15- Feb- 2020].

[5] "WeChat", Play.google.com. [Accessed: 15- Feb- 2020].

[6] "Pokémon GO", Play.google.com. [Accessed: 15- Feb- 2020].

[7] "Tripadvisor Hotel, Flight & Restaurant Bookings", Play.google.com. [Accessed: 15- Feb- 2020].

[8] "NearBee - Discover what's buzzing around you", Play.google.com. [Accessed: 15- Feb- 2020].

[9] "Google and Facebook Data Retention and Location Tracking through Forensic Cloud Analysis," 3-22-2019.

[10] M. Bridge, Google at risk of £3bn fine over location tracking", Thetimes.co.uk.

[11] K. Collins, Google collects Android users' locations even when location services are disabled, Quartz., 2017.

[12] A. Esteve, The business of personal data: Google, Facebook, and privacy issues in the EU and the USA, Vols. 7, Issue 1, International Data Privacy Law, March 2017, 2017.

[13] Policies.google.com. [Online]. Available: https://policies.google.com/. [Accessed: 26- Feb- 2020].

[14] B. Liu, W. Zhou, T. Zhu, L. Gao and Y. Xiang, "Location privacy and its applications: A systematic study," IEEE access, vol. 6, pp. 17606-17624, 2018.

[15] T. Hara, A. Suzuki, M. Iwata, Y. Arase and X. Xie, "Dummy-based user location anonymization under real-world constraints," IEEE Access, vol. 4, pp. 673-687, 2016.

[16] H. J. Do, Y.-S. Jeong, H.-J. Choi and K. Kim, "Another dummy generation technique in location-based services," in 2016 International Conference on Big Data and Smart Computing (BigComp), 2016.

[17] Y. Xiao and L. Xiong, "Protecting locations with differential privacy under temporal correlations," in Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, 2015.

[18] M. E. Andrés, N. E. Bordenabe, K. Chatzikokolakis and C. Palamidessi, "Geo-indistinguishability: Differential privacy for location-based systems," in Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, 2013.

[19] A.-M. Olteanu, K. Huguenin, R. Shokri, M. Humbert and J.-P. Hubaux, "Quantifying interdependent privacy risks with location data," IEEE Transactions on Mobile Computing, vol. 16, pp. 829-842, 2016.

[20] B. Niu, Q. Li, X. Zhu, G. Cao and H. Li, "Enhancing privacy through caching in location-based services," in 2015 IEEE conference on computer communications (INFOCOM), 2015.

[21] B. Liu, W. Zhou, T. Zhu, H. Zhou and X. Lin, "Invisible hand: A privacy preserving mobile crowd sensing framework based on economic models," IEEE Transactions on Vehicular Technology, vol. 66, pp. 4410-4423, 2016.

[22] C.-Y. Chow, M. F. Mokbel and X. Liu, "Spatial cloaking for anonymous location-based services in mobile peer-to-peer environments," GeoInformatica, vol. 15, pp. 351-380, 2011.

[23] J. Li, H. Yan, Z. Liu, X. Chen, X. Huang and D. S. Wong, "Location-sharing systems with enhanced privacy in mobile online social networks," IEEE Systems Journal, vol. 11, pp. 439-448, 2015.

[24] X. Gong, X. Chen, K. Xing, D.-H. Shin, M. Zhang and J. Zhang, "From social group utility maximization to personalized location privacy in mobile networks," IEEE/ACM Transactions on Networking, vol. 25, pp. 1703-1716, 2017.

[25] B. Palanisamy and L. Liu, "Mobimix: Protecting location privacy with mix-zones over road networks," in 2011 IEEE 27th International Conference on Data Engineering, 2011.

[26] P. Chen, Y. Lin, W. Zhang, X. Li and S. Zhang, "Preserving location and content privacy for secure ranked queries in location based services," in 2016 IEEE Trustcom/BigDataSE/ISPA, 2016.

[27] S. Mascetti, D. Freni, C. Bettini, X. S. Wang and S. Jajodia, "Privacy in geo-social networks: proximity notification with untrusted service providers and curious buddies," The VLDB journal, vol. 20, pp. 541-566, 2011.

[28] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K.-L. Tan, “Private queries in location based services: Anonymizers are not necessary,” in Proc. ACM SIGMOD, 2008, pp. 121–132.

[29] G. F. Marias, C. Delakouridis, L. Kazatzopoulos and P. Georgiadis, "Location privacy through secret sharing techniques," in Sixth IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks, 2005.

[30] Location Privacy-Preserving Mechanisms. In: Location Privacy in Mobile Applications., Springer, Singapore.

[31] A. K. Kini and S. A. Kulkarni, "Real time implementation of k fake location generation algorithm to protect location privacy in location based services," in 2017 international conference on advances in computing, communications and informatics (ICACCI), 2017.

[32] J. Ren, A. Rao, M. Lindorfer, A. Legout and D. Choffnes, "Recon: Revealing and controlling pii leaks in mobile network traffic," in Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services, 2016.

[33] "Distribution dashboard|Android Developers". [Online].Available: https://developer.android.com/about/dashboards. [Accessed:15- Feb- 2020].