Dynamic Location Service Modification (Group 11)

Royce Lim Jie Han
National University of Singapore
e0030923@u.nus.edu

Andrew Ong Ming En
National University of Singapore
e0175461@u.nus.edu

Wei Jianxin
National University of Singapore
e0454490@u.nus.edu

Ryan Christian Harrs
National University of Singapore
e0445611@u.nus.edu

Joel Ng Zhi Hao
National University of Singapore
e0191323@u.nus.edu

ABSTRACT
This project looks into Dynamic Location Service Modification where location services are spoofed through falsifying GPS data and/or WIFI MAC Addresses. We used Software Defined Radios (SDR), more specifically - LimeSDR Mini which is connected to a Raspberry PI 3. We also used NodeMCU to broadcast beacon frames. In this project, we have two key objectives that we would like to accomplish - a successful attack (using both GPS and WIFI) on static and dynamic location spoofing as well as provide a software that allows location spoofed victims to assess the probability that their location has been spoofed. Due to the prevalence of smartphones, our group focused on our attacks on smartphones, and have developed an android application for android smartphone users.

Categories and Subject Descriptors
C.2.0 [**Computer-Communication Networks**]: General—Security and Protection;

General Terms
Security, Location Service

Keywords
GPS, Wi-Fi, Location, Spoofing

1. INTRODUCTION
With the rise of driverless technology in the market, we believe that it would only be a matter of time when all vehicles would be self-driven, aided by accurate location services technology. Dynamic location spoofing presents a threat to technologies that are dependent on location services. Gaming applications that uses location services heavily like Pokemon Go, will also be affected. Users can spoof their location while they are playing the game.

Location spoofing has been proven to be fairly easy to do. Without physically touching a device or jailbreaking one, an anonymous hacker can easily trick your mobile device into thinking its time and position data are somewhere and something completely different from what they actually are. Mobile devices use both Wi-Fi and GPS to determine where they are. With SDR platforms and Wi-Fi based positioning spoofing, both Android and iPhone devices are vulnerable to location spoofing.

When mobile devices use GPS only to locate themselves, they are receiving radio waves between satellites and a physical receiver inside the mobile. This can be exploited by sending false radio waves to the phone making it think it is somewhere it isn't.

Wi-Fi is able to tell a device where it is based off of where other phones have been when connected to the same Wi-Fi. When GPS is used to find the current location, the phone will also scan nearby publicly accessible networks and records the information online. Then when another phone is near the network, it can use the data of the nearby networks and the previous person's GPS data to determine where they are, instead of using their own GPS. This can be exploited by creating false Wi-Fi zones that have GPS data saying they are somewhere completely different from where you may be.

Our main contributions in the project are listed as follows.

-We deceived nearby smartphones to display a fixed location or moving path

-We build an application which can successfully detect attacks on mobile phone

-We research on why new version of iphone is resistant to location spoofing to some extent and how can it prevent those attacks.

In this paper, we first introduce the theory of Global Positioning System and Wi-Fi Positioning System in Section 2. In Section 3 & 4, we illustrate the design details including hardware, software, steps and code. Subsequently, we conduct extensive experiments on GPS spoofing and Wi-Fi spoofing in Section 5 & 6. Moreover, we analysis our experiment results in Section 7. Finally, we further research on how to detect and defense against location spoofing attack in Section 8 and propose some thoughts about what can be done in the future in Section 9.

2. ABOUT GPS AND WPS

2.1 GPS (Global Positioning System)

GPS is a satellite-based radio navigation system. It provides humans with accurate navigation, positioning and timing services. When there is an unobstructed line from receiver to four or more GPS satellites, the receiver can receive and send signals to the satellites and compute its position and time deviation. As shown in Figure 1, GPS space segment is composed of 24-32 satellites, 6 orbits, whose planes have approximately 55° inclination. On each orbit, there are at least 4 satellites, whose orbital period is about 11 hours and 58 minutes [1]. The orbits are arranged so that at least six satellites are always within line of sight from everywhere on the Earth's surface.

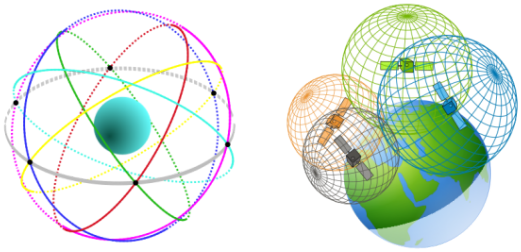

Figure 1: World Geodetic System [2]

2.2 WPS (Wi-Fi positioning system)

However, when GPS navigation is not accurate due to various causes including weak signal or obstructed by skyscrapers, WPS (Wi-Fi positioning system) may be useful. Nowadays, more and more Wi-Fi hotspots (also known as APs, or wireless routers) tend to receive at least one AP signal at any point in the city. Each wireless AP has a globally unique MAC address, and generally most of wireless AP does not move for a long period of time. When our device is turned on Wi-Fi, it can scan and collect the surrounding AP signals no matter whether it is encrypted or connected. Even if the signal strength is insufficient to be displayed in the wireless signal list, the MAC address of AP can be obtained. The current metering methods using MAC addresses are displayed in Figure 2. The device then sends the data indicating the APs to the location server, which is based on an AP database containing the relationship between MAC address and geographic address. The server calculates the geographic location of the device and sends back according to the geographic location of each AP and its signal strength. The location service provider should constantly update and supplement its database to ensure the accuracy of positioning in case the AP location changes. Currently, WPS providers such as Google and Apple continually collect data through the phones of their customers and keep their Wi-Fi location databases updated.

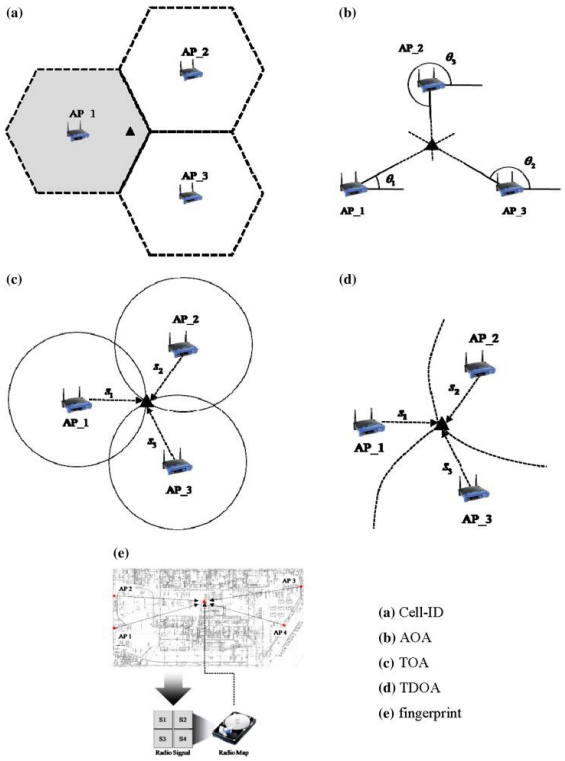

Figure 2: Positioning techniques [3]

3. EXPERIMENT SETUP

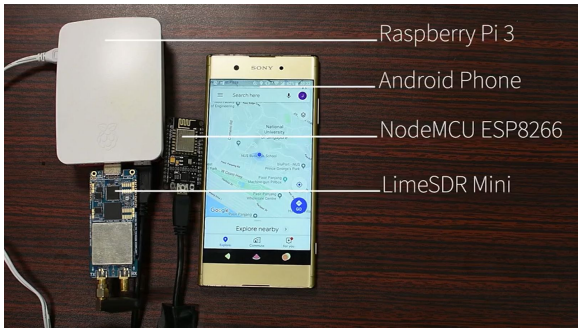

Figure 3: Location Spoofing Experiment Setup

3.1 Hardware

- RaspberryPI Model 3
- LimeSDR Mini
- NodeMCU ESP8266
- Sony Xperia XA1 Plus
- iPhone XS

3.2 Software

- Operating System: Raspbian
- GPS Signal Simulator: gps-sdr-sim
- Microcontroller IDE: Arduino
- Android 8.0 Oreo (for the mobile application)
- Application: GPS Spoof Detector

4. INSTALLATION

4.1 Setup for Raspberry Pi 3

Install newest packages in Rasbian OS:

```
> sudo apt-get update
> sudo apt-get upgrade
```

Install gcc and cmake to build and compile C code:

```
> sudo apt-get install software-properties-common
> sudo apt-get install cmake
```

4.2 Packages for GPS Spoofing

4.2.1 LimeSuite library

We download ScratchRadio as the tool contains package dependencies for SDR components (i.e. LimeSuite)

Installation commands:

```
> git clone https://github.com/myriadrf/ScratchRadio.git
> sudo ./scripts/install_deps.sh
> make LimeSuite
> make clean
```

4.2.2 gps-sdr-sim

gps-sdr-sim is a tool to generate GPS signal data streams used by software-defined radios (SDRs) to convert into radio frequency (RF).

Installation commands:

```
> git clone https://github.com/osqzss/gps-sdr-sim.git
> cd gps-sdr-sim
> gcc gpssim.c -lm -O3 -o gps-sdr-sim
-DUSER_MOTION_SIZE=4000
> cd player
> make limeplayer
```

4.3 Packages for Wi-Fi/ BSSID Spoofing

There are various hardware mediums used for Wi-Fi spoofing, such as Wi-Fi adaptors or access points. For our case, we will be using NodeMCU which runs on ESP8266 Wi-Fi chip. As such, we will need to install Arduino to compile and upload C code to the hardware.

4.3.1 Arduino

The simplest way to install Arduino is to run:

```
> sudo apt-get install arduino
```

However, this is a bare bones installation without any add-on tools, including the ability to interact with Arduino command line interface (CLI). Alternatively, launch Chrome on Raspberry Pi, head to magpi.cc/2tPw8ht, and download Arduino using the Linux ARM link. Extract the file to /opt directory , then open a Terminal and run the install.sh script.

Installation commands:

```
> cd ~/Downloads
> tar -xf arduino-1.8.3-linuxarm.tar.xz
> sudo mv arduino-1.8.3 /opt
> sudo /opt/arduino-1.8.3/install.sh
```

Arduino by default does not include ESP8266 board. We can run the following commands to add ESP8266 board URL and install board packages:

```
> /opt/arduino-1.8.3/arduino --pref
"boardsmanager.additional.urls=http://arduino.esp8266.com
/stable/package_esp8266com_index.json" --save-prefs
> /opt/arduino-1.8.3/arduino --install-boards
esp8266:esp8266 --save-prefs
```

5. GPS SIGNALS SPOOFING

Spoofing GPS signals is a simple task with the help of a SDR (i.e. LimeSDR mini). The idea here is to simulate a stream of GPS signal based on our specifications and playback using an SDR so that the signal can be received by GPS receivers nearby.

5.1 Obtain GPS Broadcast Ephemerides

GPS broadcast ephemerides (BRDC) are extrapolated satellite orbit data transmitted by satellites to receiver for navigation purposes. These data are archived daily and are available publicly through the NASA's Crustal Dynamics Data Information System (CDDIS) website. The data is required for gps-sdr-sim tool to generate GPS data streams.

Installation command format:

```
> wget ftp://cddis.nasa.gov/gnss/data/daily/<4-digit
year>/brdc/brdc<3-digit date>0.<2-digit year>n.Z
> uncompress brdc<3-digit date>0.<2-digit year>n.Z
```

Formatted example (year - 2019, date - 25 October):

```
> wget
ftp://cddis.nasa.gov/gnss/data/daily/2019/brdc/brdc2980.19
n.Z
> uncompress brdc2980.19n.Z
```

3-digit date refer to the number of days since the start of the year at 1st January, with values running from 1 to 365. The value can be obtained by running the command:

```
> date +%j
```

5.2 Obtain GPS Sample Signal File

We can feed the downloaded ephemerides data through gps-sdr-sim to generate a binary file containing GPS signal.

Command format:

```
> gps-sdr-sim \
        -e <brdc file> \ # GPS ephemerides
        -u <user motion> # NMEA GGA in csv
        —l <lat,lon,alt> \ # static location
        -s <sampling rate> \ # default 2.6MHz
        -b <IQ bit> \ # 1 for limeplayer
        -o <output> \ # default gpssim.bin
```

For static GPS location, specify the '-l' parameter instead of '-u' parameter, while inserting latitude, longitude and altitude values of our intended spoofing location.

For dynamic GPS locations, specify the '-u' parameter instead of '-l'. This parameter requires a National Marine Electronics Association (NMEA) Fix information (GGA) formatted CSV file. To generate the file, these steps are executed in-order:

1. Sketch a route using Google Earth
2. Export the route file in KML format.
3. Load the file into SatGen's NMEA simulator to convert KML file to NMEA GGA text file. (Note: set output rate to 10Hz)
4. Convert NMEA GGA text file to user motion CSV file using the 'nmea2um' tool provided in gps-sdr-sim/satgen:

```
> ./nmea2um <nmea_gga> <user_motion>
```

5.3 Playback GPS Signals

Use the generated GPS signal file as an input for limeplayer to broadcast the signals to its surroundings.

Command format:

```
> limeplayer \
        -g 0.5 \ # RF gain [0.0 … 1.0]
        -b 1 \ # IQ bit count
        -s 2600000 \ # sampling rate
        < gpssim.bin \ # GPS signal file
```

5.4 Pipelining the Process

To make the tools work in conjunction with one another, our team have made an assembly line chaining the necessary tools with the help of a bash script. The spoof_gps script can be found under our Github repository in gps-sim-sdr folder:

https://github.com/royceljh/LocationServicesSpoofing.

The pipeline starts from obtaining the latest (if not already downloaded) BRDC data from CDDIS FTP server. The user will then input a GPS signal file name to be generated. If the file name was previously created, the script will use the existing file and launch it on limeplayer. Else, the user will input the latitude and longitude data which will be used to generate the 400 seconds worth of GPS sample data. It then proceed to be broadcasted by limeplayer using pre-defined parameters.

6. WI-FI SIGNALS SPOOFING

The Wi-Fi Positioning System (WPS) makes use of nearby Wi-Fi hotspots and other wireless access points (APs) to triangulate or "guess" the location of a device. This is especially useful indoors where GPS signals are limited. Thus, by fabricating APs from another location, the WPS on a smartphone can be fooled into thinking that the device is currently in that specified location. As discussed in the earlier sections, we know that WPS infers the location based on BSSIDs or MAC addresses of nearby APs. In the following subsections, we explore the usage of some tools to crawl AP information and to broadcast beacons of spoofed BSSID information.

6.1 Obtain Network Information From WiGLE

WiGle.net is a consolidated central database containing wireless network information across the world [4]. Queries can be fired from directly through their web interface or using their API.

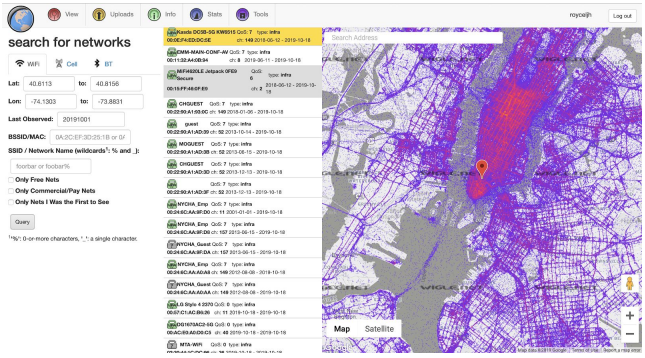

Figure 4: WiGLE.net web user interface's query results

For the purpose of being able to automate the crawling process, we used API tokens (allocated to account holders) to query the web server by sending our crafted HTTP request detailing the specifications of our search criteria.

HTTP request format:

```
https://api.wigle.net/api/v2/network/search?onlymine=false&first=0&latrange1=<1>&latrange2=<2>&longrange1=<3>&longrange2=<4>&lastupdt=<5>&encryption=<6>&freenet=false&paynet=false&minQoS=<7>&resultsPerPage=<8>
```

Explanation of fields labelled <>

<1> and **<2>**: Latitude lower and upper bounds

<3> and **<4>**: Longitude lower and upper bounds

<5>: Last update time - this is set to 3 months prior to current date as setting it too recent may result in zero search results.

<6>: Encryption criteria is set to WPA2

<7>: Minimum quality of signal is set to 2 to exclude weak signals for better location accuracy.

<8>: The maximum number of results returned by the query is set to 50.

We wrote a python code read in user-specified latitude and longitude values and fire the crafted HTTP request to the web server using authentic API tokens. The results containing network information are stored in JSON format. By extracting the field 'netid', we can collate all the MAC addresses or BSSIDs of the wireless networks found in the area of interest and write them to a text file.

These information are used to build C code which is then uploaded to NodeMCU. Since the code has to be compiled before uploading to NodeMCU, there is no simple way to construct a variable containing all the MAC addresses during run time. One method to overcome this is to establish communication between Raspberry Pi and NodeMCU so that Raspberry Pi will read the contents of the MAC addresses file and transmit the information to NodeMCU, which will receive and construct a variable to hold the values. We implemented a much simpler approach, which is to directly insert/replace into a predetermined line of code in C program with the contents of the file specifying the variable that holds all the MAC addresses. In this way, the code will be already contain the variable before being compiled.

To accomplish this, the format of the MAC address file is as shown below.

macAddr.txt

```
const uint8_t mac[][6] = {{0x00,0x11,0x32,0xA4,0x0B,0x94},
{0x00,0x15,0xFF,0x46,0x0F,0xE9},{0x00,0x22,0x90,0xA1,0x9F,0xD9},
{0x00,0x22,0x90,0xA1,0x9F,0xDB},{0x00,0x22,0x90,0xA1,0x9F,0xDF},
{0x00,0x22,0x90,0xA1,0xAD,0x39},{0x00,0x22,0x90,0xA1,0xAD,0x3B},
{0x00,0x22,0x90,0xA1,0xAD,0x3D},{0x00,0x22,0x90,0xA1,0xAD,0x3F},
{0x00,0x24,0x6C,0xAA,0x9F,0xD0},{0x00,0x24,0x6C,0xAA,0x9F,0xD8},
{0x00,0x24,0x6C,0xAA,0xA0,0xA8},{0x00,0x24,0x6C,0xAA,0xA4,0xC8},
{0x00,0x2C,0xC8,0x1B,0x6C,0xC7},{0x00,0x2C,0xC8,0x1B,0x6C,0xC8},
{0x00,0x2C,0xC8,0x1B,0x6C,0xCE},{0x00,0xF2,0x8B,0xF0,0x33,0x71},
{0x00,0xF2,0x8B,0xF0,0x33,0x72},{0x00,0xF2,0x8B,0xF0,0x33,0x7E},
{0x00,0xF7,0x6F,0xCF,0xD4,0xE7},{0x02,0xF7,0x6F,0xCF,0xD4,0xE7},
{0x0C,0xF4,0xD5,0x4E,0x36,0xB8},{0x10,0x0E,0x7E,0x03,0x92,0x83},
{0x10,0x0E,0x7E,0x03,0x92,0x85},{0x10,0x0E,0x7E,0x03,0x92,0x86},
{0x10,0x0E,0x7E,0x03,0x92,0x87},{0x10,0x0E,0x7E,0x03,0x92,0x88},
{0x10,0x0E,0x7E,0x03,0x92,0x89},{0x10,0x0E,0x7E,0x03,0x92,0x8A},
{0x10,0x0E,0x7E,0x03,0x92,0x8B},{0x18,0x1B,0xEB,0xE6,0x1C,0x4F},
{0x1C,0xB0,0x44,0xDE,0xA7,0x1F},{0x24,0x79,0x2A,0x34,0x44,0xA8},
{0x24,0xB6,0x57,0xF8,0xBB,0xC9},{0x24,0xB6,0x57,0xF8,0xBB,0xCB},
{0x24,0xB6,0x57,0xF8,0xBB,0xCD},{0x24,0xB6,0x57,0xF8,0xBB,0xCF},
{0x2C,0x33,0x11,0xED,0xBC,0x87},{0x2C,0x33,0x11,0xED,0xBC,0x88},
{0x2C,0x33,0x11,0xED,0xBC,0x8E},{0x2C,0x33,0x11,0xED,0xC8,0xB7},
{0x2C,0x33,0x11,0xED,0xC8,0xB8},{0x2C,0x33,0x11,0xED,0xC8,0xBE},
{0x2C,0x33,0x11,0xED,0xC8,0xE7},{0x2C,0x33,0x11,0xED,0xC8,0xE8},
{0x2C,0x33,0x11,0xED,0xC8,0xEE},{0x2C,0xFD,0xA1,0x35,0x10,0x28},
{0x34,0x12,0x98,0x0A,0x3C,0xC4},{0x34,0x12,0x98,0x0A,0x3C,0xC5},
{0x34,0xFA,0x9F,0x75,0x87,0xDC}};
```

Figure 5: Sample text file — a variable holding 50 MAC addresses

Next, a command line text stream editor called SED is used to insert/replace a line in the C code with our text. Specifically for our code, the global variable will sit at the 2nd line of code to avoid tampering with the main code beneath it.

Commands used:

```
> sed -i '2d' macAddr.txt \ # Remove 2nd line
> sed -i '2i\<text to insert>' macAddr.txt \ # Insert text into 2nd line
```

6.2 NodeMCU Beacon Spammer

A beacon spamming tool that works on the ESP8266 chip is used as a base template for our Wi-Fi spoofing purposes [5]. For the original code, random MAC addresses were used for each SSIDs. Beacon packets are crafted and repeatedly sent to 14 different Wi-Fi channels. The format of the beacon frame is as follows [6]:

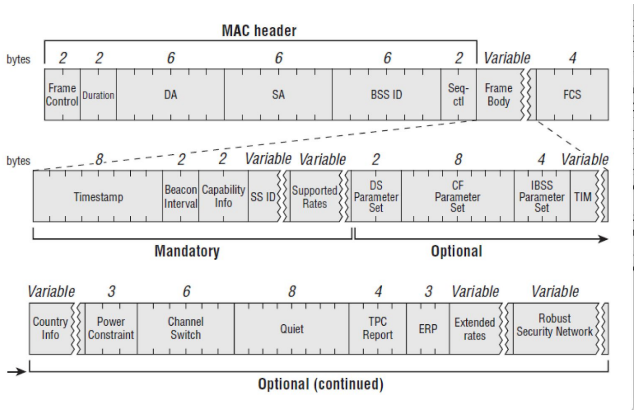

Figure 6: Beacon Frame structure

By changing the bytes for the field 'BSSID', the beacon frame can be configured to carry our intended BSSID/MAC address.

As explained in section 6.1, the constant variable containing all the MAC addresses will be inserted into this beacon spamming code. By editing the base code, these MAC addresses can be attached to dummy SSIDs and broadcasted its surroundings.

6.3 Pipelining the Process

Similar to GPS spoofing, we pipeline the process by chaining all required tools through bash script. The spoof_wifi script can be found under our Github repository in beacon-spammer folder:

https://github.com/royceljh/LocationServicesSpoofing.

The user is expected to input a static location with latitude and longitude information. The surrounding networks from the location will be automatically crawled from wigle.net and BSSIDs are extracted from the JSON object while formatting the output. The output is then inserted directly into the Arduino code which is then verified and uploaded to NodeMCU using Arduino CLI.

The command to verify and upload:

```
> /opt/arduino-1.8.3/arduino -v verify —-board esp8266:esp8266:nodemcuv2 --port /dev/ttyUSB0 beaconSpam.ino
> /opt/arduino-1.8.3/arduino -v upload —-board esp8266:esp8266:nodemcuv2 --port /dev/ttyUSB0 beaconSpam.ino
```

7. ANALYSING ATTACK

Based on our experimentation Android and IPhone, namely the Sony Xperia XA1+ (Android 8.0) and IPhone XS respectively, we find that each individual attack vector is not sufficient to fool the phone's location services on default settings. The default settings for Android is set to "High accuracy" which uses both GPS and Wi-Fi information for location services. The iPhone location services uses both information as well. Thus, we require both attack vectors for a more reliable spoofing attempt.

7.1 Static spoofing

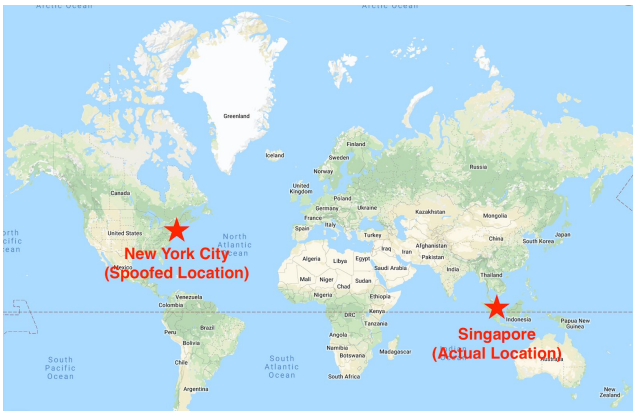

Figure 7: Spoofing NYC location from Singapore

The static latitude and longitude values used were 40.7131019, -74.0072986.

The duration of smartphone being fooled by the spoofed signals may vary for different phones. This is likely tied to the location caching in phones. For Android, we can quickly refresh the cache and fetch new location information by switching off and on the location services function. On the other hand, iPhone does not seem to respond to the same method, which makes the time taken to spoof a location unpredictable. There may be other factors that can contribute to the unreliability of location spoofing. These can range from having signal interference nearby to relying on cached location instead of fetching fresh data.

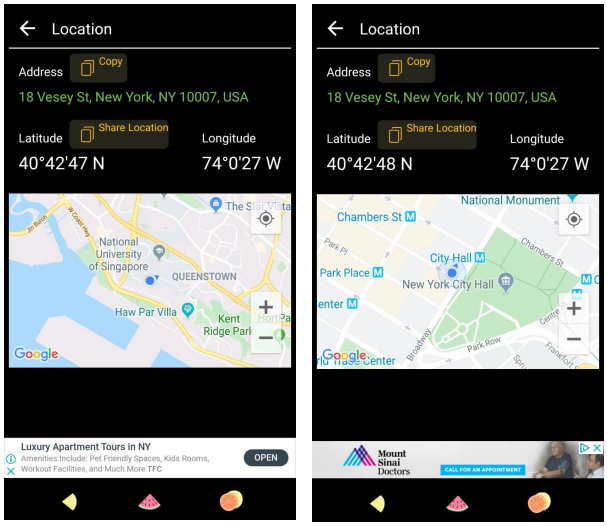

Figure 8: NUS (left) and NYC City Hall (right)

The unresponsiveness of conventional maps like Google maps can be seen in Figure 8. Using a GPS app that ties together with Google maps, we can see that on the left image, the address field was filled with NYC location but the corresponding location on Google maps remained at the actual location on NUS campus. It took some delay for the Google Maps location to update to the one specified in the address field.

7.2 Dynamic Spoofing

In the earlier section describing how we conduct our 2 vector attacks, we only mentioned the steps to simulate dynamic location for GPS. Dynamic location based on Wi-Fi information is no simple task. The main difficulty is knowing the exact location of the ever-changing GPS signal and keeping the network information in sync at all times during the dynamic route.

Nevertheless, dynamic GPS signals and static wide-range Wi-Fi spoofing is sufficient for both the Android and iPhone to display our fabricated route made using Google Earth.

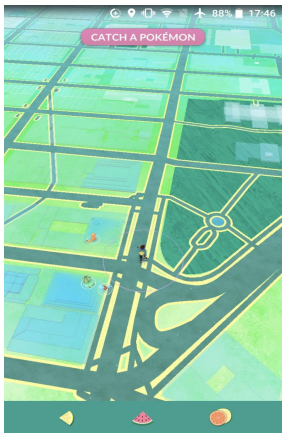

Figure 9: Dynamic spoofing a route in Pokemon Go

7.3 Limitations

Firstly, the time taken to generate the GPS signal file is tremendously slow due to the limited computation power on Raspberry Pi 3, thus the delay for the initial run may nullify the effects of live attacks. Subsequent runs on the same spoofing location will not be affected as the process will use the existing GPS signal file.

Secondly, there is no clear method of making the phone recognise the spoofed signals instantly, and we could not predict the amount of time taken for the spoofed location to take effect as different applications or phones can have different location caching mechanisms.

Last but not least, for this attack to work, our intended spoofing location needs to meet the criteria of having multiple Wi-Fi access points so that we could spoof multiple networks in order to overwhelm the number of access points in the actual location. Also, if there are numerous (>50) network being sensed in the actual location, Wi-Fi spoofing may not work reliably.

8. MITIGATION TECHNIQUES

8.1. Our Mobile Application

We developed a mobile application called "GPS Location Spoofing" to allow users to have an easy way to detect if their phone's location has been spoofed. Please visit our repository at https://github.com/andrewome/GPS-Spoof-Detector to view the source code. In general, to detect that a location has been spoofed, regardless of the spoofing method used, the most common sign of spoofing would be a drastic change in location [7]. Also, we realised that it is likely that attackers would keep the value of the altitude constant, so an altitude that does not change over time, could also be a sign of spoofing. However, we considered that there may be users are taking the lift while using our application. As such, we keep our spoofing check simple, by basing it on distance between each location object stored. This takes advantage of the fact that location spoofing attacks take time for them to take effect on the victim's device, which allows the victim ample time to use our application to detect if a spoofing has taken place.

The steps to use this mobile application is as follows:

1. Start storing the location of the phone when the user presses the start button.
2. For every few (~5) seconds, it stores a new location.
3. The user clicks "Stop Collection and Analyse" after a short span of time.
4. Show Results.

Results are computed based on the distances between each location collected. This allows the application to detect any big jump in location (>100m) which is common when location is spoofed. However, we also recognize the fact that attackers may spoof locations that are near the user's location. As such, it may be wise for the future developments of the application to cater to different modes (e.g. walking, driving) and tweak the distance criteria based on the mode.

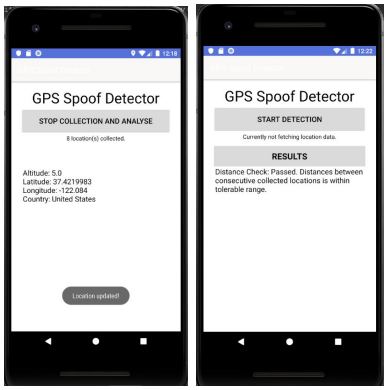

Figure 10: GPS Spoof Detector Result Screen

8.1.1 Our Observations from the Experiment

Initially, during the first few minutes after the WiFi and GPS spoofing attacks have started, both devices showed the accurate location. After around 5 minutes, the devices managed to calibrate to the spoofed locations.

When we stopped the collection, our application show that distance check has failed *(see figure 10)* due to the big jump in distance travelled from Singapore to New York City. This is an indication that spoofing has occurred.

Figure 11: GPS Spoof Detector Failed Result Screen

8.2 Other Techniques Against GPS Spoofing

Most smartphone devices make use of a combination of Wi-Fi, GPS radio signal, Bluetooth, and cellular networks to pinpoint the exact location of the user. This feature will mitigate against GPS Spoofing, as there are multiple sources of data, and the device could pick the one that it deems most reliable. This is also supported by our experiment, where GPS spoofing was conducted with Wi-Fi spoofing, on "High Accuracy" mode, the phone did not respond to the false GPS signals.

Beyond individuals, companies are also vulnerable to GPS spoofing. One form of mitigation would be to position the company's antenna in a less visible and accessible place, where the probability of picking up signals from ground level will be reduced. Multiple copies of antenna could be used on both ends of the building, allowing the company to detect any suspicious activity if the signals received by the antennas differ drastically from one another. An antenna that blocks fraudulent or interference-causing signals could be used as well [8].

8.3 Other Techniques Against Wi-Fi Spoofing

The most characteristic sign of Wi-Fi spoofing is a sudden increase in new networks in a short period of time. To ensure that this technique works for places with few wireless networks, we need to take into account of proportions. As such, to detect this form of attack, we could look at the existing number of wireless networks available, check the number of new wireless networks that have appeared. If the new wireless networks exceeds a relative large proportion of the existing number of wireless networks, then it could be a sign that Wi-Fi spoofing took place.

Also, when there are numerous wireless networks with similar names, it could be a sign of Wi-Fi spoofing as well. This is because there is a possibility that these wireless networks have auto-generated names.

9. FUTURE RESEARCH AND STUDY

Future studies and research on this matter could explore other technological devices (and mobile operating systems like iOS) that involve the use of a location service.

10. CONCLUSION

In conclusion, it is possible to perform GPS and Wi-Fi spoofing to fool today's technological smart devices, both statically and dynamically. There are however, many viable options for individuals and organisations to adopt, when faced with a possible location spoofing situation.

REFERENCES

[1] "USCG Navcen," GPS Frequently Asked Questions (FAQ). [Online]. Available: https://www.navcen.uscg.gov/?pageName=gpsFaq#What. [Accessed: 08-Nov-2019].

[2] C. Allado, Patel, A. B. Kulkarni, and U. K. Sharma, "World Geodetic System (WGS84)," GIS Geography, 18-Feb-2018. [Online]. Available: https://gisgeography.com/wgs84-world-geodetic-system/. [Accessed: 08-Nov-2019].

[3] S. Woo, S. Jeong, E. Mok, L. Xia, C. Choi, M. Pyeon, and J. Heo, "Application of WiFi-based indoor positioning system for labor tracking at construction sites: A case study in Guangzhou MTR," Automation in Construction, 19-Aug-2010.

[4] WIGLE. 2019. WIGLE: Wireless Network Mapping. https://wigle.net/index [Accessed on 2019-10-19].

[5] S. Kremser, “ESP8266 Beacon Spam,” 2017. [Online]. Available: https://github.com/spacehuhn/esp8266_beaconSpam. [Accessed: 22-Oct-2019].
[6] R. Nayanajith, “802.11Mgmt : Beacon Frame,” 08-Oct-2014. [Online]. Available: https://mrncciew.com/2014/10/08/802-11-mgmt-beacon-frame/. [Accessed: 20-Oct-2019].
[7] K. Wang, S. Chen, and A. Pan, “Time and Position Spoofing with Open Source Projects,” 2015.
[8] M. Korolov, “What is GPS spoofing? And how you can defend against it.,” 07-May-2019. [Online]. Available: https://www.csoonline.com/article/3393462/what-is-gps-spoofing-and-how-you-can-defend-against-it.html. [Accessed: 01-Nov-2019].