

Detecting and Defending against Location based Permission Leakage in Applications

¹Prof. Vishal Shinde, ²Mr.Nitin Varkute, ³Miss.Ankita Darekar, ⁴Mr.Rupesh Thakare, ⁵Mr.Pratik Padwal

¹Asst. Professor, ^{2,3,4,5}UG Student, ^{1,2,3,4,5}Computer Engg. Dept. Shivajirao S. Jondhle College of Engineering & Technology, Asangaon, Maharashtra, India.

¹mailme.vishalshinde@gmail.com, ²nitinvarkute007bond@gmail.com, ³ankitadarekar98@gmail.com,

⁴rupesh123thakare@gmail.com, ⁵padwalpratik3@gmail.com

Abstract - With the pervasiveness of smart phones, location-based services (LBS) have received considerable attention and become more popular and vital recently. However, the use of LBS also poses a potential threat to user's location privacy. In this paper, aiming at spatial range query, a popular LBS providing information about points of interest (POIs) within a given distance, we present an efficient and privacy-preserving location-based query solution, called EPLQ. However, Android components may leak permissions, location & user data either carelessly or maliciously. Therefore, users are required to grant permissions to apps during app installation, which may lead to permission mismanaged. In this project, it proposes the software that aims to detect location-based permission leakage by proposing a light-weight mechanism.

Keywords - Android Security, Inter-component communication, Permission leaks, Static Analysis, Location permissionleakages, Location Based services.

I. INTRODUCTION

As smart phones have become more popular, the focus of mobile computing has shifted from laptops to phones and tablets. When a user attempts to install the application, Android will warn the user that the application requires certain restricted resources (for instance, location data), and that by installing the application, it is granting permission for the application to use the specifies resources. If the user declines to authorize these permissions, the application will not be installed. However, statically requiring permissions does not inform the user how the resource will be used once granted. A maps application, for example, will require access to the Internet in order to download updated map tiles, route information and traffic reports. It will also require access to the phone's location in order to adjust the displayed map and give real-time directions. The application's functionality requires sending location data to the maps server, which is expected and acceptable given the purpose of the application. However, if the application is ad-supported it may also leak location data to advertisers for targeted ads, which may compromise a user's privacy. Given the only information currently presented to users is a list of required permissions, a user will not be able to tell how the maps application is handling her location information. To address this issue, System present Android Leaks, a static analysis framework

designed to identify potential leaks of personal information in Android applications. on a- large scale.

II. RELATED WORK

A. Vulnerability Detection in Smartphone Applications:

Recent researches carried out in the field of Smartphone security have concentrated mostly on Android operating system (OS). The security issues in Android are a concern owing to the fact that it is open source and developing of android application is very easy. Some of the commonly noted behaviours were, collecting user information, sending premium-rate Short Messaging Service (SMS), malware written for amusement, for credential theft, search engine fraud and ransom. Some of the algorithms used in android permission leakage detection and defending are given as bellows:

Algorithm1: The Algorithm that Extracts the Concrete Permissions Names (String Analysis)

Input: Method Call Stack, Target Method, Target Method Parameter

Result: Set of Permission Strings

stack ← Method Call Stack;

tm ← Target Method;

tp ← Target Parameter;

pSet ← set ();

pSet ← find Permission (tm, tp);

if pSet is empty then

tp ← getCurrentMethodParameter ();

```

N ← size(stack)-1;
r ← String Analysis (stack [1...N], stack[N], tp);
pSet ← pSetUr;
return pSet;

```

This plug-in performs an intra-method analysis and manages the following scenarios: either (1) the permission is directly given as a literal parameter, or (2) the permission value is initialized in a variable which is given as a parameter, or (3) an array is initialized with several permissions and is given as a parameter. In the case where only a single permission is given to the method, statements in the unit graph containing a reference to a valid Android permission String is extracted and the permission added to the list of the permissions needed by the method under analysis. In case of an array, all permissions of references to Android Permission Strings are added to the list. It can happen that the permission string cannot be found in the current method MI's body. This happens when it is referenced from a local variable initialized with one of the current method's parameter P. The solution is for the analysis to go one method down in the method call-stack (lines6-10). At this point the analysis goes through the statements of M_{i-1} looking for a call to M. When a call is found the parameter, P is extracted and the string analysis starts again from there.

Algorithm 2: Capability leak detection

Input: entry points, known method summaries

Output: a set of capability leaks

for each entry point \in entry points do

work list = initial state: start of the entry point

states = initial state summaries = known method summaries

for each state \in work list does

remove state from work list

if state's instruction is a method call then

if a summary does not exist for the target

then summarize (target, summaries);

end

end work list+ = $\delta(\text{state}) - \text{states}$

states+ = $\delta(\text{state})$ end

if a dangerous-call state is flagged then

report the state as a capability leak end.

III. EXISTING SYSTEM

Existing approaches for handling permission leakages focus entirely on specific analysis and detection of permission leakages very slowly. They neither focus on detection of permissions which running background without knowledge of it to you. nor do the fast detection of permission leakages. This work handles both of these detections and defending against permission leakages in android apps.

Recent contributions on detection of leakages deal with the problem of permission leakages in few android apps and concentrates only on single permission but, neither these methods perform effective and fast detection of leakages of inter app permissions. The detection and defending inter app permissions introduces the main idea for detecting permission leakages in android app, as well as the methods for judging permissions and selecting the most important permission first which leak user data from android apps. Android Static Analysis CHEX is the first tool to detect component hijacking vulnerabilities. They use static analysis to discover entry points in apps and leverage entry point permutation techniques to detect the information flows which lead to privacy leakage. A android provide context sensitive taint analysis to detect privacy leakage on Android. Intent Fuzzer leverages fuzzy test to generate different Intent messages to invoke components of Android System apps. It requires to modify Android framework to log which permissions are actually used by the components. Therefore, it can capture what permissions in these apps may be exploited by other apps. Intent Fuzzer leverages fuzzy test to generate different Intent messages to invoke components of Android System apps. It requires to modify Android framework to log which permissions are actually used by the components. Therefore, it can capture what permissions in these apps may be exploited by other apps. However, the previous both systems are focused on detecting intra component sensitive data leaks in single components of apps and cannot deal with cross-app privacy leaks.

IV. PROBLEM STATEMENT

Problem being solved: Detection of Location Based permission leakages.

An android app contains: user data, location data, permissions related to firmware etc. Each android app has different permissions. The project objective is to detect permission leakages and defending against them by finding leakages in that permissions of android which access user data without knowing anything to user.

V. PROPOSED SYSTEM

Ad Libraries nearly every ad library this system looked at leaked phone data and, if available, location information as well. It hypothesizes that nearly any access of sensitive data inside ad code will end up being leaked, as ad libraries provide no separate application functionality which requires accessing such information. Privacy-preserving POI query has been studied in LBS as: 1) public LBS and 2) outsourced LBS. In this paper, it focuses on the latter setting. In the former setting, there is an LBS provider holding a spatial database of POI records in plaintext, and LBS users query POIs at the provider's site. In outsourced LBS, The LBS provider allows authorized users (i.e., LBS users) to utilize its data through location-based queries.

Algorithm:

Step 1: Take length 9 encryption key from the user.

Step 2:

```
for(i=1;i<=9;i++)
{
if(!k[i]repeated)
{
find ASCII value of k[i]
temp=Add the digit of k[i] and take mod 9,
if(temp==0)
{
temp=temp+1;
```

Label 1: if(temp already occupied)

```
{
temp= temp+1;
goto label 1;
}
```

Label 2: if(temp already occupied)

```
{
temp= temp+1;
goto label 2;
}
seq[i]=temp;
} //end of if
else
{
j++;
rem[j]=i;
} //end of else
} //end of for loop
int n=1;
while(rem[n]!=0)
{
for(i=1;i<=9;i++)
{
for(k=1;k<=9;k++)
{
if(i==seq[k])
exit();
else
seq[n]=i;
goto label 3;
}
}
```

Label 3 : n++;

Encryption:

```
encryption()
{
for(i=1;i<=9;i++)
{
insert seq[i]th element of grid to ith position of new enc-grid
```

```
}
```

```
}
```

Decryption:

```
decryption()
{
for(i=1;i<=9;i++)
{
insert ith element of enc-grid to seq[i]th value position of new
dec-grid
}
}
```

VI. WORKING OF ALGORITHM

The general idea of working of proposed system algorithm is given as bellows:

- This algorithm takes values in two arrays seq [] & rem [].
- After taking array values from the user it started the calculation of sequence seq [] array values and the Remaining rem [] array values.
- If any kind of value is repeated then it stores the value of the index for the future processing.
- Then it again does the processing for remaining repeated characters.
- Finally, it generates a shuffled values array.

Mathematical Model:

The mathematical model & working of algorithm used in this system is given as bellows:

- 1) Data to be encrypted: ABCDEFGHI (i.e. Divided into nine parts).
- 2) Make Initial Arrangement of data into grid.
- 3) Take 9 length of key from user. i.e. For example k [9] = Karnataka.
- 4) Check if k is repeated. // For characters other than first character.
- 5) If not, find the ASCII value of the given k[i] i.e. For k=75 then calculate as (7+5) %9 =3.
- 6) Check whether K is in "seq []" array? If not then add value three (3) into the array i.e. Seq [1] =3
- 7) similarly, let's do for the remaining characters. For a=65, (6+5) %9=2 Not in seq [] then seq[2]=2. For r=82, (8+2) %9=1 Not in seq [] then seq [3] = 1. For n=78, (7+8) %9=6 Not in seq [] then seq [3] = 6. For a=65, (6+5) %9=2 repeated seq [2] = 2.

Here, a=65 is repeated. i.e. 'a' has come second time. Rem[++] =i. // i.e. we store index of the 'a' for future processing. i.e. seq [5].

For t=84, (8+4) %9=3 is in the seq [] then increment it by 1. i.e. 3+1=4 Not in the seq [] then seq [6] = 4. For a=65, (6+5) %9=2 repeated seq [2] = 2. Here, a=65 is repeated. i.e. 'a' has come third time.

Rem[++j] = i. // i.e. we store index of the 'a' for future processing. i.e. seq [7].

For k=75, (7+5) %9=3 repeated seq [1] = 3.

Here, k=75 is repeated. i.e. 'k' has come 2nd time.

Rem[++j] = i. // i.e. we store index of the 'k' for future processing. i.e. seq [8]

For a=65, (6+5) %9=2 repeated seq [2] = 2.

Here, a=65 is repeated. i.e. 'a' has come fourth time.

Rem[++j] = i. // i.e. we store index of the 'a' for future processing. i.e. seq [9].

Processing for remaining repeated characters:

After First step: seq= [0,3,2,1,6,0,4,0,0,0] and the rem=[0,5,7,8,9,0,0,0,0,0].

8) now we check which numbers are still available to use in seq [] and fill them at remaining places sequentially. i.e. we have 5,7,8,9 remaining to be used in seq [] which are filled at index 5,7,8,9 respectively of seq [].

9) finally, our seq [] array took like this: seq [] = [0,3,2,1,6,5,4,7,8,9]. This is the shuffling sequence we generated by our algorithm.

VII. SYSTEM ARCHITECTURE

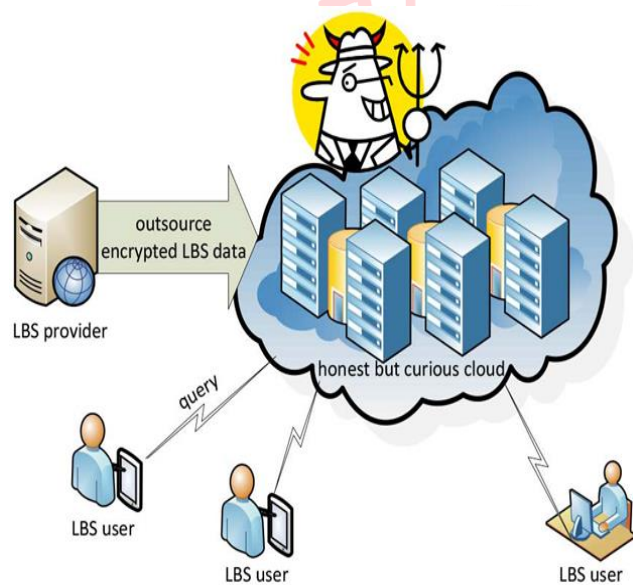


Fig.1 System model of outsourced LBS under consideration

LBS provider allows authorized users (i.e., LBS users) to utilize its data through location-based queries. Therefore, the LBS provider encrypts the LBS data, and outsources the encrypted data to the cloud.

LBS User: In this Module, the user sends location-based queries to the LBS provider (or called the LBS server) and receives location-based service from the provider.

LBS Provider: In this Module, the LBS provider provides location-based services to the user. LBS allows clients to query a service provider.

VIII. DESIGN DETAILS

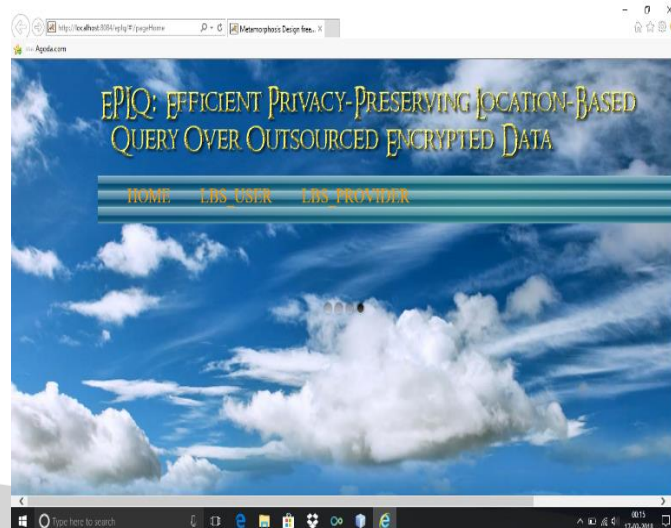


Fig.2LBS System Home Page

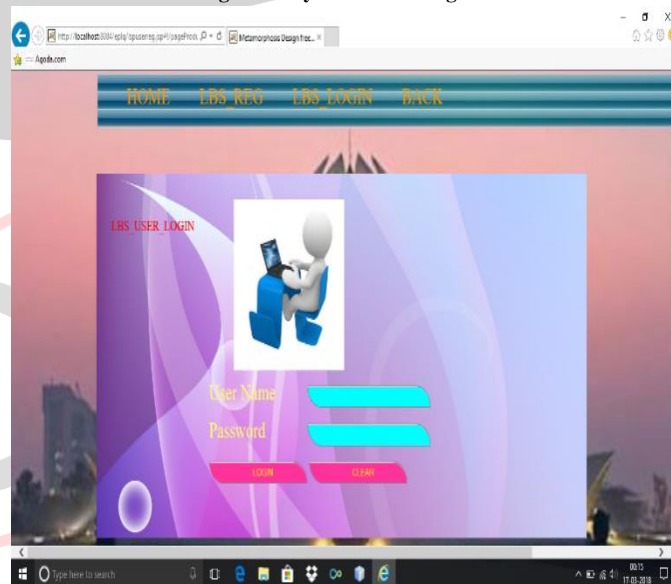


Fig.3LBS System Home Page

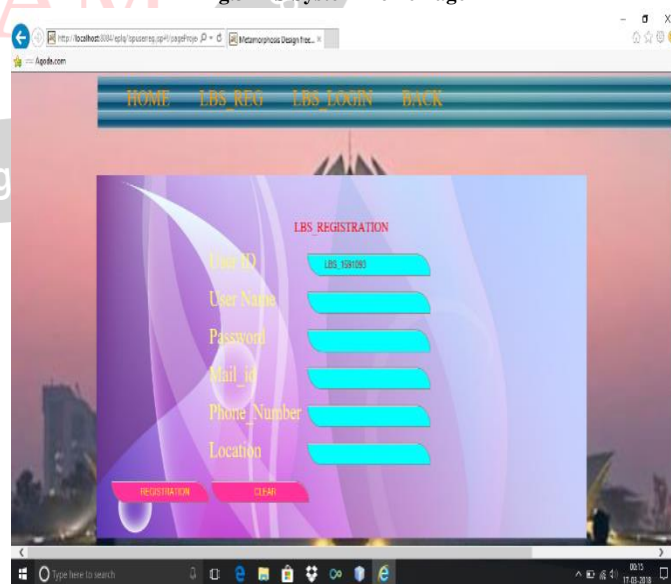


Fig.4LBS System Home Page

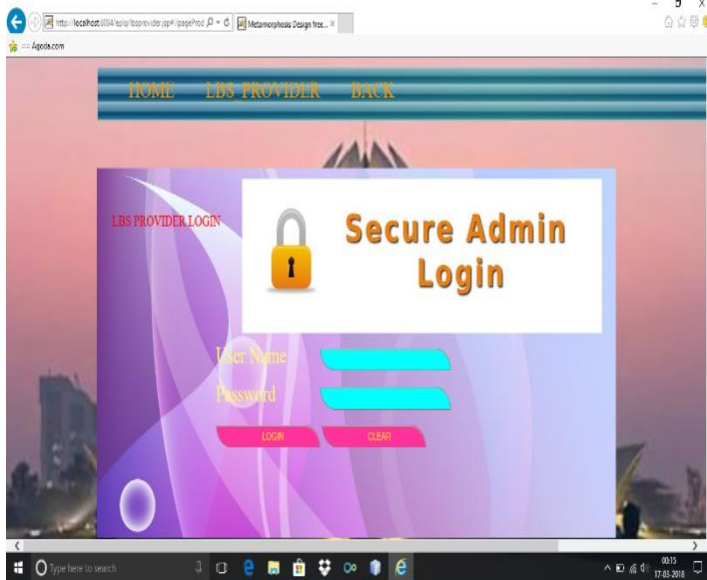


Fig.5LBS System Home Page

X. CONCLUSION

We have tried to implement paper “Detecting & Defending Against Permission Leakages In Android” with combining another paper “EPLQ-Location Based system “using Location based system(LBS). In this paper, the system has proposed EPLQ, an efficient privacy protecting solution for smart phones, which preserves the privacy of user location, and achieves confidentiality of LBS data.

REFERENCES

- [1] Lichun Li, Rongxing Lu, *Senior Member, IEEE*, and Cheng Huang, “EPLQ: Efficient Privacy-Preserving Location-Based Query Over Outsourced Encrypted Data”, **IEEE Internet Of Things Journal**, Vol. 3, No. 2, April 2016.
- [2] Apple Inc. App store review guidelines. Accessed March 30th, 2012.
- [3] 3. A.P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In Proceedings of the 18th ACM conference on Computer and communications security, pages 627–638. ACM, 2011.
- [4] M. Eagle, C. Kruegel, E. Kirda, and G. Vigna. Pio’s: Detecting privacy leaks in iOS applications. In Proceedings of the Network and Distributed System Security Symposium, 2011.
- [5] Smartphone OS Market Share, 2016 Q1, <http://www.statista.com/statistics/266136/global-market-share-held-by-smart-phone-operating-systems/>.
- [6] EPLQ: Efficient Privacy-Preserving Location-Based Query Over Outsourced Encrypted Data”

- [7] <http://www.jpinfotech.org/eplq-efficient-privacy-preserving-location-based-query-outsourced-encrypted-data/>
- [8] Android security and permissions. Accessed <http://d.android.com/guide/topics/security/security.html>
- [9] Manifest android developers. <https://developer.android.com/reference/android/Manifest.permission.html>