UNIVERSITY OF CALIFORNIA,
IRVINE

A Recommendation System for Predicting Privacy Leaks in Mobile Traffic

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Computer Engineering

by

Milad Asgari Mehrabadi

Thesis Committee:
Professor Athina Markopoulou, Chair
Professor Carter T. Butts
Professor Aparna Chandramowlishwaran

2019

DEDICATION

I dedicate this thesis first to my lovely parents and sister whom I haven't seen for a long time, but they have always been there to support me, second my lovely grandmother whom I miss her so much. Undeniably, I owe all my academic achievements to them, and this dedication is the smallest gratitude that I could express to them, and I hope to see them soon.

TABLE OF CONTENTS

Page

LIST OF FIGURES **iv**

LIST OF TABLES **v**

ACKNOWLEDGMENTS **vi**

ABSTRACT OF THESIS **vii**

1 Introduction **1**

2 Problem Statement & Dataset Overview **4**
2.1 Per-Packet Prediction of PII Leaks 4
2.2 Datasets 6
2.3 Observations 8

3 A Recommendation System for Per-packet PII Prediction **15**
3.1 Recommendation Systems 15
3.2 Baseline Predictor 17
3.3 Neighborhood Predictor 18
3.4 The Weighted Combination of the Predictors 20
3.5 Weighted Correlated Leak Type Predictor 20

4 Evaluation **22**
4.1 Setup 22
4.2 Using the Correlation of Leak Types 25
4.3 Optimal WCLT 26
4.4 Comparison of WCLT to Baselines 31
4.5 WCLT Compared to Deep-packet Inspection 31

5 Related Work **34**

6 Conclusion **36**

Bibliography **37**

LIST OF FIGURES

Page

2.1 PII prediction per-packet using a pre-trained recommendation system (RS) 5
2.2 Privacy leak patterns of apps-domains for *Advertiser ID* (a), and *Android ID* (b) for *Antshield* dataset 8
2.3 Privacy leak patterns of apps-domains for *Advertiser ID* (a), and *Location* (b) for *App-Versions* dataset 9
2.4 Bipartite graph of the background leaks in *Antshield* dataset. Green nodes denote domains and pink ones show the applications. The number in front of each node corresponds to its degree. 10
2.5 Correlation between leak types for both datasets 11
2.6 App similarity graph and communities based on *Advertiser ID* (*Antshield* dataset) . 12
2.7 App similarity graph and communities based on *Advertiser ID* (*App-Versions* dataset) 13
2.8 Comparison of the background leaks vs. the foreground for *mbinc12.mb32b* 14

3.1 Leak prediction problem as a recommendation system framework. "1" represents leak, "0" non-leak, and "?" shows missing values. 16

4.1 Average RMSE of the test sets for each predictor per leak type (*Antshield* dataset) . 24
4.2 Average RMSE of the test sets for each predictor per leak type (*App-Versions* dataset) 24
4.3 Performance of different recommender system models of *Advertiser ID* in *Antshield* dataset 27
4.4 Performance of different recommender system models of *Location* in *Antshield* dataset 28
4.5 Performance of different recommender system models of *Advertiser ID* in *App-Versions* dataset 29
4.6 Average F1 score (a) and hit ratio (b) of *Weighted Correlated Leak Type Predictor* for the first 6 leak types given the best N (N has been annotated for each leak type) for *Antshield* dataset 30
4.7 Average F1 score (a) and hit ratio (b) of *Weighted Correlated Leak Type Predictor* for the first 6 leak types given the best N (N has been annotated for each leak type) for *App-Versions* dataset 30

LIST OF TABLES

Page

2.1 Number of packets per leak type (background *vs.* foreground) for *Antshield* dataset 7
2.2 Number of packets per leak type for *App-Versions* dataset 7
2.3 Summary of considering datasets 8

4.1 Classification of results for the top-N recommendation system 25
4.2 Comparison of WCLT over the best N with decision tree and AdaBoost trained on *Antshield* dataset using apps and domains as features 32
4.3 Comparison of WCLT over the best N with decision tree and AdaBoost trained on *App-Versions* dataset using apps and domains as features 32
4.4 Evaluation of decision tree trained on *Antshield* dataset using deep-packet inspection and key-value pairs in header/payload as features 33

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor Prof. A. Markopoulou for her consistent guidance and her insightful comments. Her unwavering assistance, indispensable advice, and technical instructions showed me the proper direction to excel in this project. My sincere gratitude goes to her because she devotedly cooperated in our publication.

My appreciation also extends to committee members, Prof. Butts and Prof. Chandramowlishwaran not only because of their priceless time and review, but also because I learned many fundamentals in the courses that I had with Prof. Chandramowlishwaran and the meetings with Prof. Butts, which directly or indirectly contributes to this project.

Finally, I want to thank my lab-mates especially A. Shuba and E. Bakopoulou who helped me during this project by providing me the datasets to test my framework, and also when I was disappointed they motivated me and boosted me up.

ABSTRACT OF THE THESIS

A Recommendation System for Predicting Privacy Leaks in Mobile Traffic

By

Milad Asgari Mehrabadi

Master of Science in Computer Engineering

University of California, Irvine, 2019

Professor Athina Markopoulou, Chair

Today's smart phones have access to personal stored data, including personally identifiable information (PII) that can be used to uniquely identify users. It is well-known that a wide range of mobile applications transmit this data to remote servers, including their own servers, third-party advertisers, and trackers, which clearly poses a threat to user privacy. The present study's goal is to detect PII in packets transmitted out of a mobile device, referred to as "privacy leaks". This study build on prior work that developed systems for intercepting each network packet and inspecting it to detect PII, typically using deep-packet inspection (DPI) and/or machine learning techniques. This thesis, develop a lightweight mechanism that can predict if an outgoing packet contains any PII, based on minimal information, namely (i) the application name (package name) that generated the packet and (ii) the second-level destination domain. The problem is formulated as a recommendation system combining baseline and neighborhood predictors that exploit the similarity of mobile app behavior and PII leak types. Two different datasets of popular apps are used to get insights into privacy leak patterns. It is shown that the present framework can successfully detect 89% and 84% of PII in network packets on average while achieving F1 score as high as 0.97 and 0.91 in both datasets.

Chapter 1

Introduction

As mobile devices have access to our personal and sensitive information to conduct their functions, using the growing number of applications can cause privacy risks. Users cannot control the leak of their personally identifiable information (PII) to a wide array of external servers. These external servers may indeed need the PII for providing a service (*e.g.,* the developer or application server), or it may be for the purpose of tracking (ad servers, trackers and analytic services). Most of the related work that tries to detect privacy leaks fall into there categories: (i) *static analysis* for source code analysis like [4, 13], (ii) *dynamic analysis* and runtime analysis like [9, 34], and (iii) *network traffic analysis* for monitoring network traffic [24, 26].

This thesis takes the network-based approach: the goal is to detect PII in outgoing packets transmitted out of the mobile device, which is referred to as "privacy leaks". Prior work has developed mobile software for intercepting each outgoing network packet and inspecting it to determine whether it contains PII, typically using deep-packet inspection (DPI) and/or machine learning techniques with packet-based features [22, 24, 25, 27]. In this thesis, a lightweight mechanism is developed that can predict whether a particular outgoing packet contains a PII or not, based only on minimal information, namely (i) the name of the application that generated the outgoing packet and (ii)

the destination domain that the packet is sent to. Such a lightweight system can be a front line of defense: when the recommendation system predicts that a particular packet transmits PII out of the device, it can trigger more heavyweight and accurate mechanisms based on DPI. Furthermore, as mobile traffic becomes increasingly encrypted, deep-packet inspection is no longer possible, and we need to be able to identify potential privacy leaks based on readily available information on the device (application name) and network packet (destination domain) that cannot be encrypted or obfuscated.

This thesis' contributions are as follows. The problem of predicting whether a particular outgoing packet is likely to contain a privacy leak of particular type (namely *Android ID*, *Advertiser ID*, *Email*, *Location* etc.) is formulated as a top-N recommendation system. The author combines several predictors, including (i) a baseline predictor that accounts for heterogeneity of leak numbers and types among mobile apps and destination domains; (ii) a neighborhood predictor that exploits the similarity of apps in terms of domains they sent PII to; and (iii) a neighborhood predictor that exploits the fact that packets often leak several PIIs together and some leak types co-occur in the same packet (*e.g., Location*, *City* and *Zip Code*). Two different datasets which are provided by [23,25] are used in this thesis. These datasets collected from automatic interactions with popular apps, basically a packet trace in which each packet is annotated with a list of PII it contains. The first dataset has 400 apps (*Antshield* dataset), and the second one contains historical and current versions of 512 popular Android apps which covers $7,665$ app releases over 8 years (*App-Versions* dataset). These datasets are analyzed to obtain insights into privacy leak patterns, which guide the design of the predictors, as well as for evaluation. It is shown that the proposed system can successfully predict most leaks in the test data (more than 89% and 84% in the average case for *Antshield* and *App-Versions* datasets, respectively) while achieving the F1 score up to 0.97 and 0.91 for *Android ID* and *IMEI* leak types for these datasets.

The rest of this thesis is structured as follows. Chapter 2 states the problem, presents the datasets and highlights corresponding observations. Chapter 3 presents the predictors inspired by the pat-

terns found in the dataset. Chapter 4 presents evaluation results. Chapter 5 reviews related work, and finally Chapter 6 concludes the thesis and outlines future work.

Chapter 2

Problem Statement & Dataset Overview

2.1 Per-Packet Prediction of PII Leaks

Mobile applications send packets to different servers, regularly. These packets might contain one or more different PIIs in packet header and/or payload. The goal is to find out for a given packet it contains privacy leaks or not. As mentioned in Chapter 1, prior approaches detect these PII leaks using DPI and/or machine learning with packet-based features [24]. In this thesis, recommender system models are developed to predict leakage of PII using only two features: (i) application name which has generated the packet, and (ii) the destination domain that this packet goes to. Both of these features are readily available for each packet. App name is available on the device and can be mapped to the outgoing packet. Also, it is possible to extract destination domain of each packet using IP address from TCP/IP and HTTP/S headers or the URL address inside the unencrypted packet payload. For this purpose, the recommender system framework is trained offline using the available datasets and then it can be used for new incoming packets. This packet might contain a PII (*e.g., Location*). Fig. 2.1 illustrates an overview of the problem definition. For each packet, the source application that generates this packet and the destination address it goes to are available (the

features). The goal is to identify whether an outgoing packet has specific leak types by looking only at application name and destination domain.

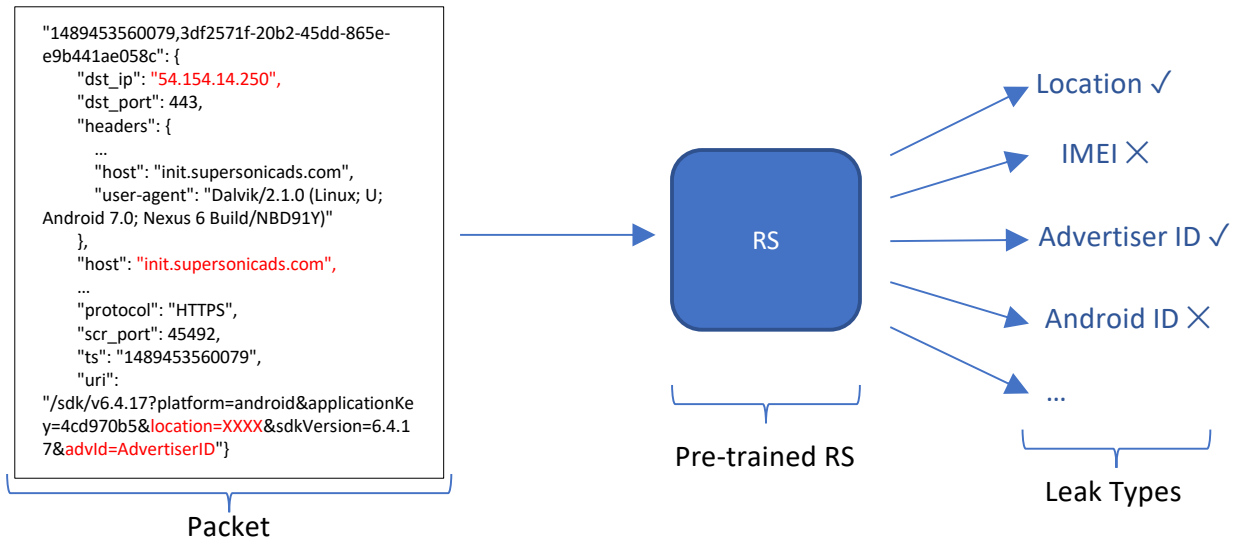


Figure 2.1: PII prediction per-packet using a pre-trained recommendation system (RS)

In this thesis, we consider the most common PIIs of each dataset, which are categorized into four different categories by [25]:

1. *Device Identifiers: IMEI*, *IMSI*, *Android ID*, *Serial Number*, *Advertiser ID*, *Hardware Serial*
2. *Personal Information: Email*, *First Name*
3. *Location: City*, *Zip Code*, *Location* coordinates
4. *Credentials: Username*

This thesis' goal is to predict the PII leaks of each packet based on only two properties: (i) package name of corresponding app and (ii) the second-level destination domain. Hence, throughout the remainder of this thesis, we refer to packet and the pair of <*app, domain*> interchangeably.

2.2 Datasets

To train the models, we use two different datasets in this thesis. These datasets collected by [25] and [23] and contain a set of recorded packets exchanged between mobile applications and servers by running *UI/Application Exerciser Monkey* [3]. *Monkey* is a tool that automatically interacts with apps to simulates user activities.

Antshield Dataset. All entries in this dataset have been collected using AntMonitor [26]; an open-source application for monitoring and collecting packet measurements from mobile devices. The dataset consists of packet traces annotated with: the PII information (could be none, one or more PII), the name of the application, server which the app is communicating with (here, to deal with the sparsity of the matrix, the second-level domain of each server is considered, for example, google.com instead of api.google.com), the protocol of the packet (HTTP or HTTPS), destination IP and port number, timestamp, headers, payload data, and whether the packet is classified as a background or a foreground activity. A foreground activity entails that a data packet is generated and sent to the server while the user is interacting with the application. A background activity implies that a data packet is generated and sent to the server automatically by the application when the application window is not active. A background activity typically remains undetected to the user, and can even occur while application is asleep or in standby mode. For this reason, background activities are particularly interesting to monitor, as they could potentially reveal a deliberate underhanded behavior of an application which may be indicative of a security threat. This dataset was generated by simulating five minutes of user interaction with each of the 400 most popular applications on Google Play. Within the packets generated during background activities, there are 351 different domains, 203 applications, and $2,779$ PII leaks in total. These leak types and number of corresponding packets containing each of them for background and foreground activities is summarized in Table 2.1.

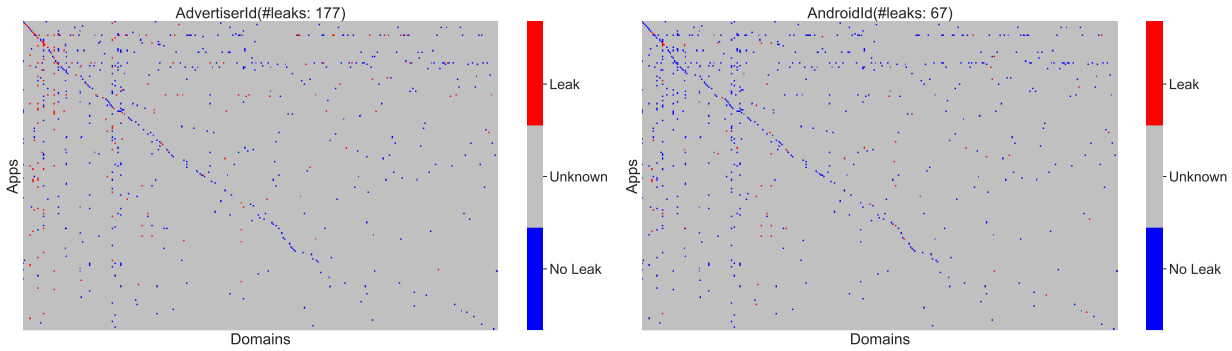
	# of Background	# of Foreground
Non-leaks	6354	7562
Advertiser ID	1528	1690
Location	744	125
Android ID	319	904
City	66	59
Zip Code	37	36
IMEI	27	106
Serial Number	23	80
Username	16	66
IMSI	16	2
First Name	2	35
Email	1	0
Total Packets: 18205	**8172**	**10033**

Table 2.1: Number of packets per leak type (background *vs.* foreground) for *Antshield* dataset

	# of packets
Non-leaks	26837
Advertiser ID	6726
IMEI	2670
Android ID	2487
Email	2247
Location	2237
Hardware Serial	1310
Other leak types	23140
Total Packets	**67654**

Table 2.2: Number of packets per leak type for *App-Versions* dataset

App-Versions Dataset. This publicly available dataset also includes packet traces of historical and current versions of 512 popular Android apps over 8 years of app version history which covers $7,665$ app releases. These traces of packets have package name of each app, second-level domain, protocol, time stamp and a list of PIIs that exist in each packet. These packet-logs are generated by simulating of $5,000$ Monkey automated interactions and monitored using *Mitmproxy* [1]. To avoid sparsity, all of the domains that different versions of an application is contacting with are combined together. After this aggregation, we have 2258 unique domains and 512 applications. Detailed information about this dataset is mentioned in Table 2.2.



(a) *Advertiser ID* (b) *Android ID*

Figure 2.2: Privacy leak patterns of apps-domains for *Advertiser ID* (a), and *Android ID* (b) for *Antshield* dataset

Also, the summarized information of both datasets is mentioned in Table 2.3.

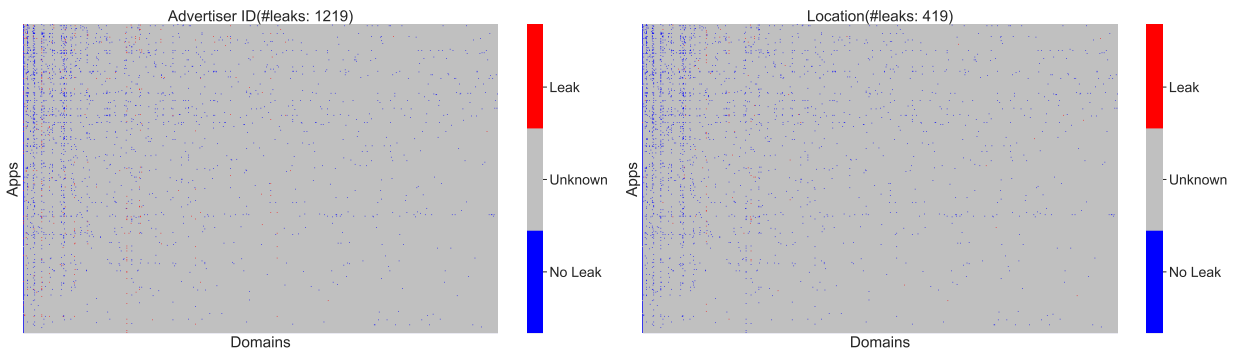
	# apps	# domains	# leaking packets	# non-leaking packets	source
Antshield (Background)	203	351	2779	6354	[25]
App-Versions	512	2258	40817	26837	[23]

Table 2.3: Summary of considering datasets

2.3 Observations

As the datasets contain very few leaks for some of the leak types, Fig. 2.2 and Fig. 2.3 show the matrix, and the number of *unique* leaks (in terms of <*app, domain*>) for the first two PIIs with the highest number of leaks (*Advertiser ID* and *Android ID* for *Antshield* dataset and *Advertiser ID* and *Location* for *App-Versions* dataset). Also, *Advertiser ID* is analyzed in detaild since it has the highest number of *unique* leaks within considering leak types in both datasets. The ones marked as red denote the existence of a PII leak for the corresponding <*app, domain*> pairs. Blue marks represent the non-leaking pairs and finally, the gray ones show missing values.

Fig. 2.2, 2.3 show some biases in data. In other words, some apps send a specific leak type to more different domains. The app with the highest number of domains in *Antshield* dataset is *com.myyearbook.m1 (MeetMe)*, a chatting app, which sends *Advertiser ID* to 20 different domains.



(a) *Advertiser ID* (b) *Location*

Figure 2.3: Privacy leak patterns of apps-domains for *Advertiser ID* (a), and *Location* (b) for *App-Versions* dataset

The corresponding app in *App-Versions* dataset is *com.withbuddies.yahtzee (YAHTZEE)*, a group dice game, with 31 domains. Also, *mopub.com*, a monetization and analytical platform for mobile app developers and publishers, receives *Advertiser ID* from the highest number of different apps in *Antshield* dataset (16) and *facebook.com* receives *Advertiser ID* from 114 different apps in *App-Versions* dataset.

In addition, it is worth mentioning that there are some talkative applications communicating with multiple domains by sending PII regardless of its type. The first two talkative applications from *Antshield* dataset are *com.myyearbook.m* and *com.freecraft.pocket.edition* that talk to 22 and 13 domains, respectively. Also, *applovin.com* and *mopub.com*, have the highest number of apps talking to them (16). Such an observation is shown in Fig. 2.4 which is a bipartite graph of *Antshield* dataset for the background PII transmissions. In *App-Versions* dataset, *mobi.MultiCraft* with 35, and *com.withbuddies.yahtzee* with 32 different domains are the first two talkative apps. Also, *googleapis.com* (438) and *google.com* (133) are the first two talkative domains regardless of considering leak types.

Furthermore, by examining the correlation between leak types, it is observed that there are high correlations between co-occurrence of leak types. Fig. 2.5 illustrates these correlations (Pearson correlation) between different leak types for both datasets. For example, Fig. 2.5a shows that there

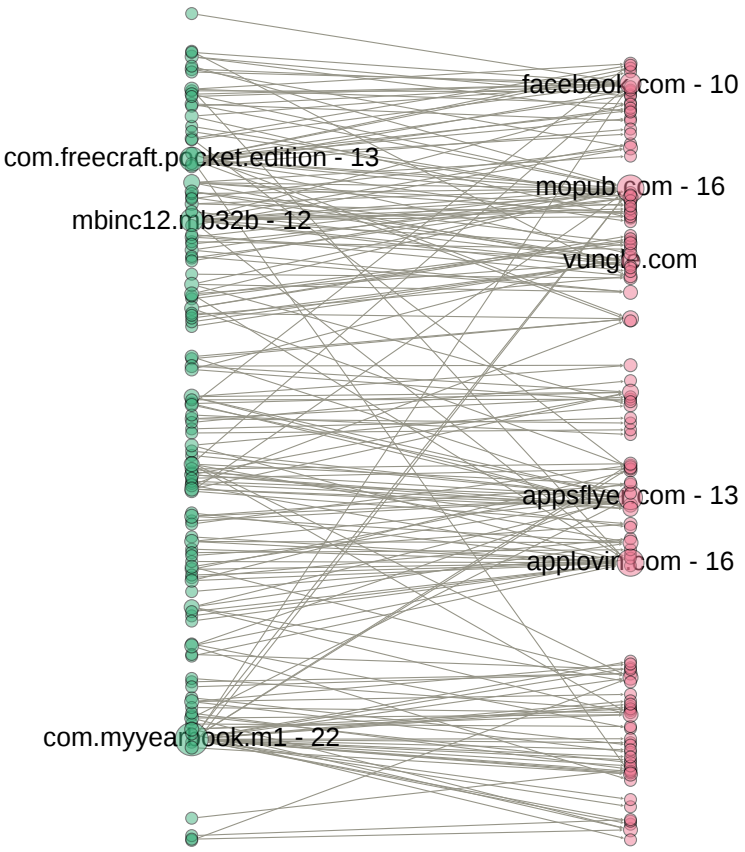
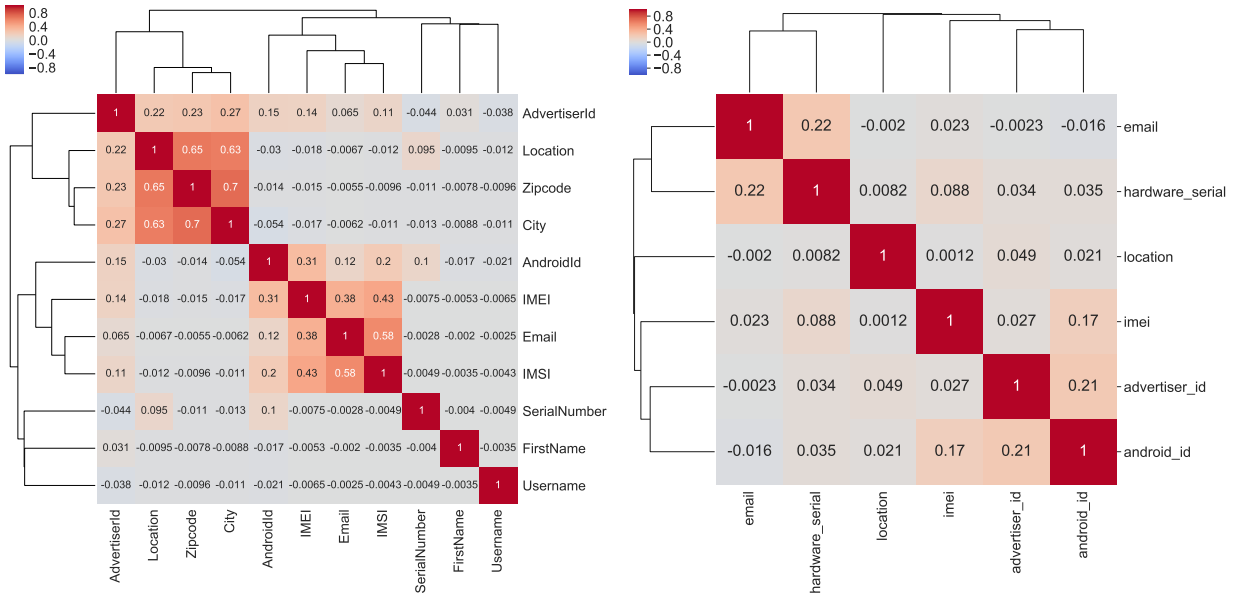


Figure 2.4: Bipartite graph of the background leaks in *Antshield* dataset. Green nodes denote domains and pink ones show the applications. The number in front of each node corresponds to its degree.

are high correlations between *Location*, *Zip Code*, and *City* which accords with the fact that all of them are for location-based services.

Furthermore, Fig. 2.6 and Fig. 2.7 show the similarity graph of the applications based on their *Advertiser ID* leakage to common domains. Each node in this graph is an application, and the edges show the similarity of these applications in terms of sending the *Advertiser ID* to common domains. The value of this similarity is the number of common domains that two applications are talking with. After running the community detection algorithm provided by the graph analysis tool, Gephi [5], it is shown that these communities capture the most similar apps regarding a specific leak type. For example, the purple community in Fig. 2.6 contains *Talking Tom* and *Talking Hank* which are from the same developer and in the same category. Also, applications from the same



(a) *Antshield* dataset (b) *App-Versions* dataset

Figure 2.5: Correlation between leak types for both datasets

developers are labeled in these figures (colors represent communities). The author hypothesizes that this is due to the developers' use of similar libraries/SDKs. This observation leads the author to take advantage of the neighborhood models, seeking to exploit the fact similar apps (which are talking to the common domains) are likely to exhibit similar leaks.

Fig. 2.8 shows the background *vs.* foreground activity for *mbinc12.mb32b* (a Free Music MP3 Player) for *Antshield* dataset. As shown in this figure, *mbinc12.mb32b* leaks *City* and *Zip Code* only in the background. Besides, this figure demonstrates that there could be more domains in the background for some apps. Therefore, considering background activities is interesting since the user is not directly aware of them, and they could be indicative of a security threat.

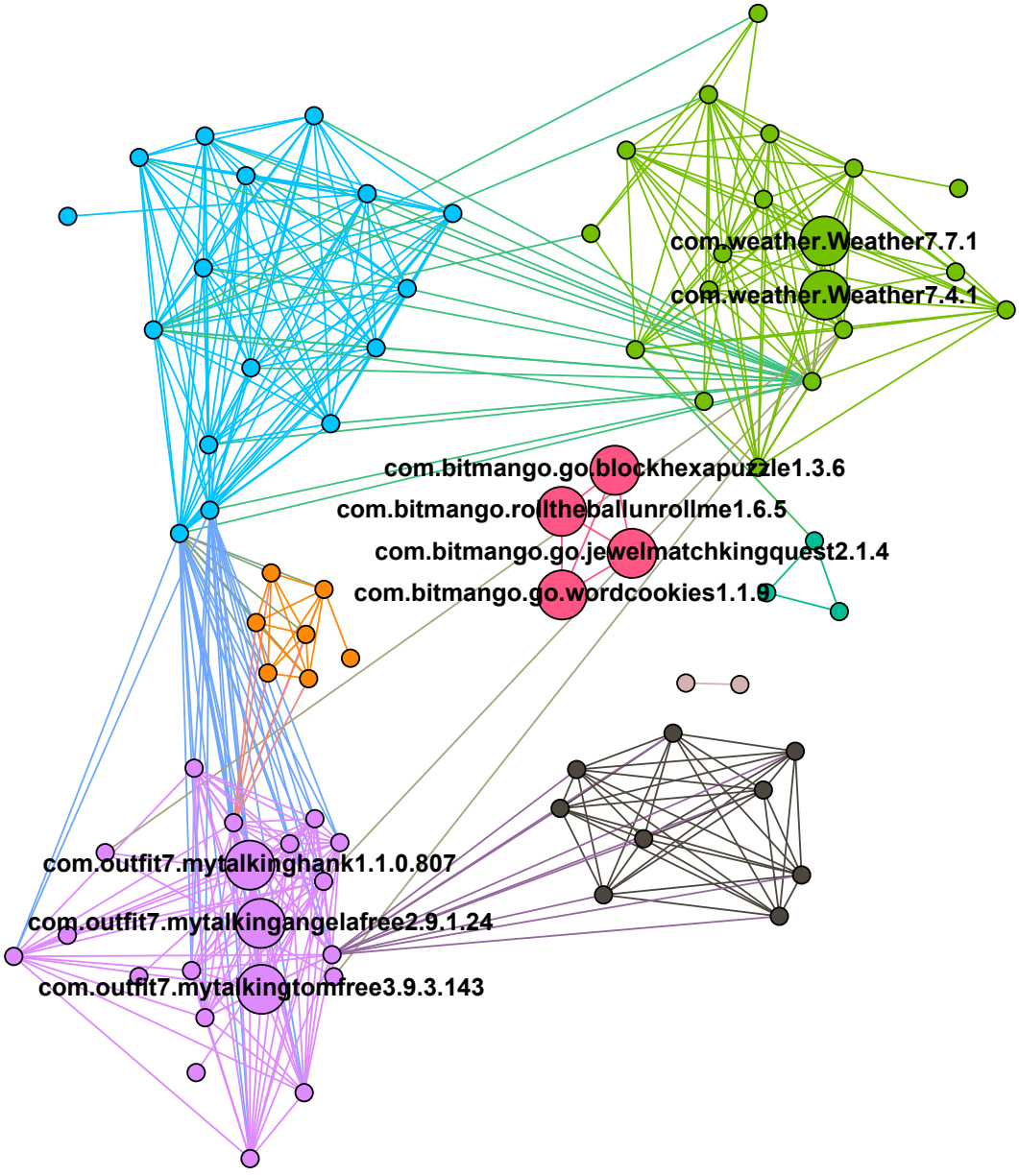


Figure 2.6: App similarity graph and communities based on *Advertiser ID* (*Antshield* dataset)

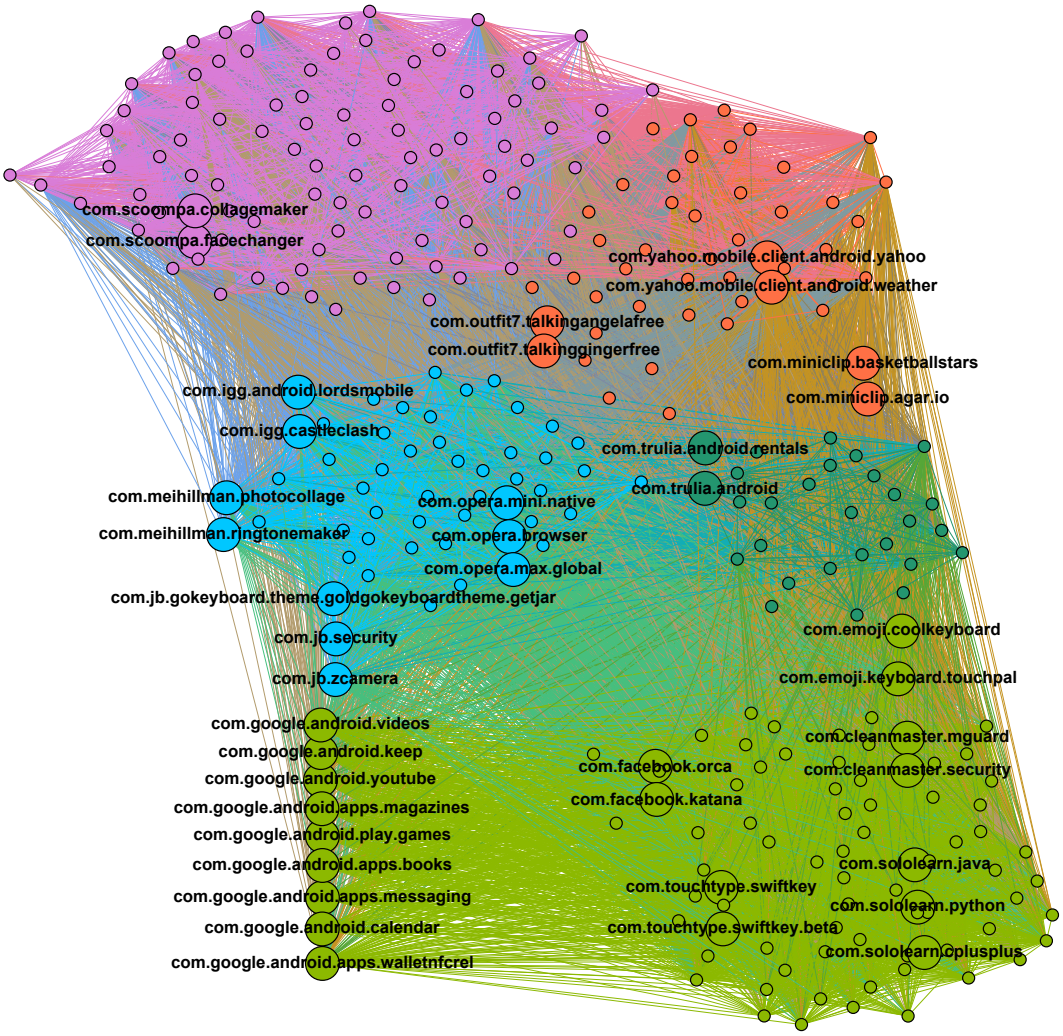


Figure 2.7: App similarity graph and communities based on *Advertiser ID* (*App-Versions* dataset)

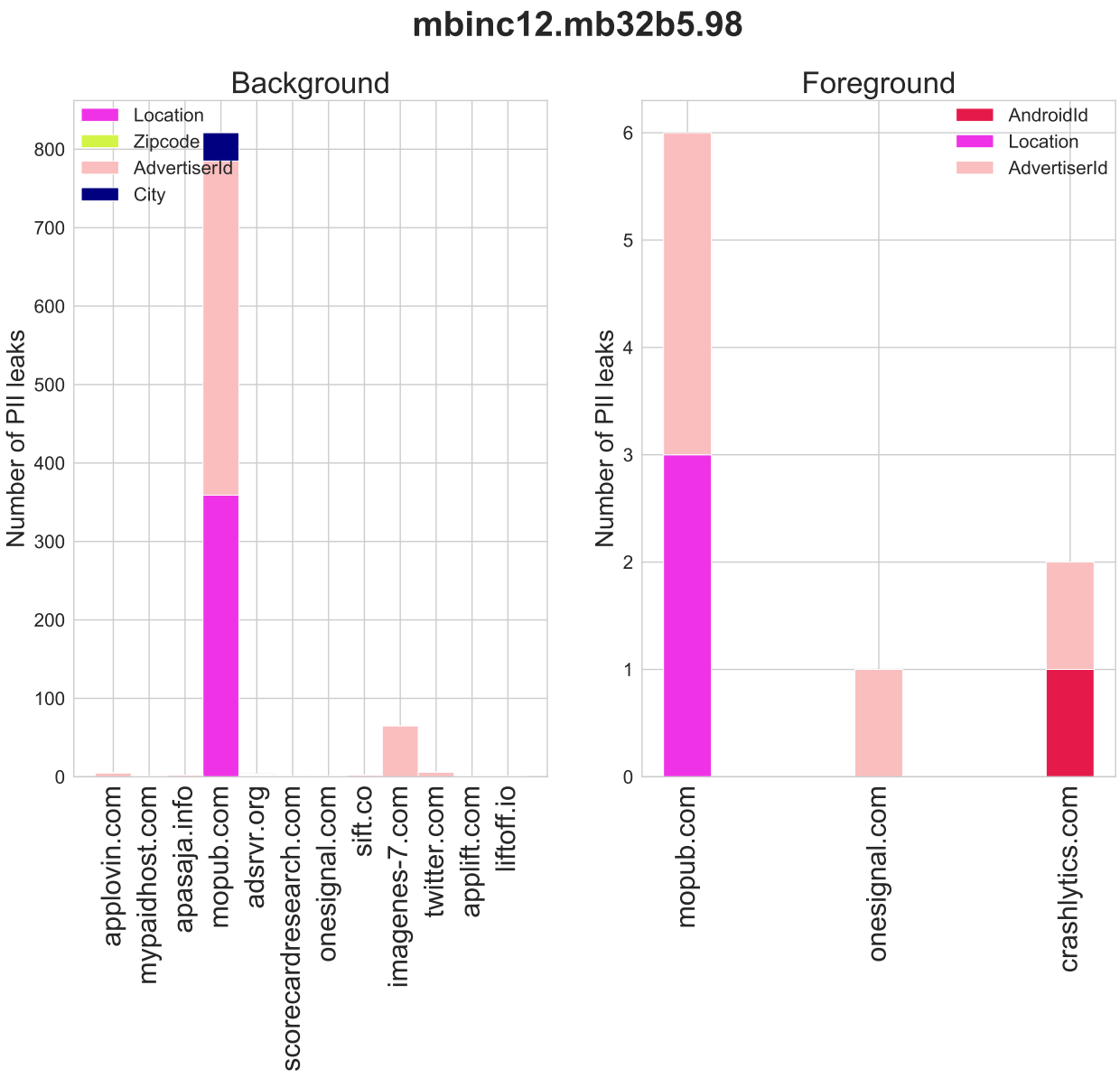


Figure 2.8: Comparison of the background leaks vs. the foreground for *mbinc12.mb32b*

Chapter 3

A Recommendation System for Per-packet PII Prediction

3.1 Recommendation Systems

In this thesis, the problem of leak prediction is formulated as a binary recommendation system. Recommendation systems are widely used to predict unknown ratings. Netflix Cinematch [2] aims to predict unknown movie ratings using the known ratings. Similarly, Amazon [19] uses recommendation systems to show relevant products to its customers based on the known ratings and user interests. However, their usage has been extended to other domains same as database queries [6] and web services [31]. Such services, Netflix for example, could represent their data as a matrix where the rows in this matrix represent the users and the columns denote movies. The matrix entries are the ratings of the corresponding movies. In this thesis context, for each leak type a similar matrix is built, where rows represent apps and the columns denote domains these apps talking to, as illustrated in Fig. 3.1. The entries of this matrix determine whether the corresponding leak type exists ("1"), does not exist ("0"), or is missing ("?").

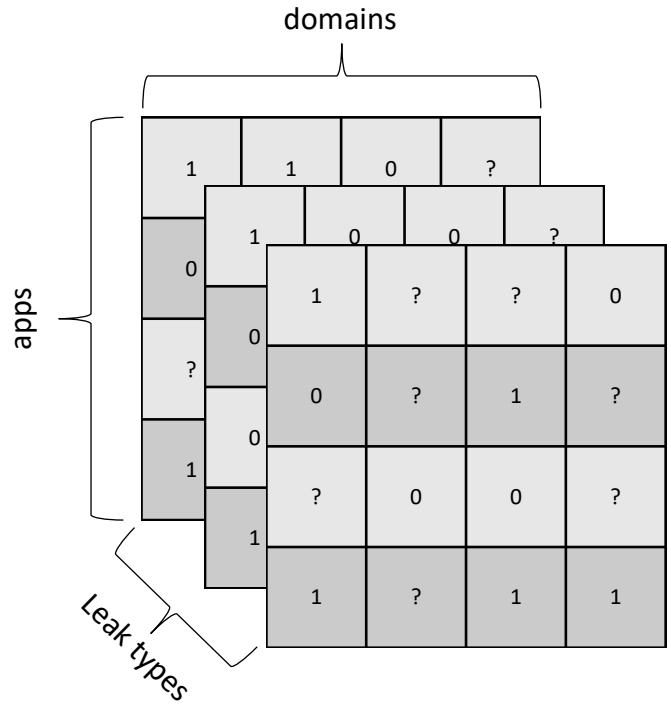


Figure 3.1: Leak prediction problem as a recommendation system framework. "1" represents leak, "0" non-leak, and "?" shows missing values.

The motivation to use recommendation systems is that the PII leak detection can be defined in this context, since unknown PII leaks (unseen packets) can also be predicted using the existing information.

Background. In the classic Netflix recommendation system problem, users' movie ratings are contained in a *Rating Matrix* $\mathcal{R}$, such that each row is a user, and each column is a movie. It follows that entry r_{ij} of this matrix holds $user_i$'s rating for $movie_j$. There are some missing values inside the matrix which are unknown ratings. The goal of this problem is to predict these missing values. Generally, there are two techniques for recommendation systems: content-based filtering and collaborative filtering [7]. In content-based filtering, the idea is to look at each row individually and recommend items based on the user profile. In contrast, collaborative filtering considers all the data of the rating matrix and explores the existing patterns of the whole matrix. In collaborative filtering, there are two main approaches: neighborhood and latent-factor. The neighborhood approach could either use the similarity of users or movies based on the ratings. The latent-factor approach finds the statistical similarity of users in the hidden structure of data

that is called *latent factors* which are factorized version of the rating matrix [7]. In this thesis, the approach is collaborative filtering, and the predictions are made based on the neighborhood model.

To formulate the problem in a recommendation system framework, we build the table of app-domain as a rating matrix for each leak type l. Let R^l be such a matrix. Each element of this matrix, r^l_{ij}, is the existence of the leak type l in at least one packet going from the application i to the domain j (Fig. 3.1). To make the predictions, 20% of the known data is selected to be test data and the rest as the training data. To deal with overfitting caused by the sparsity of the data, regularization factors are added in the models. Finally, to find the value of these factors, 5-fold cross-validation technique is used. The model is built in a hierarchical approach, starting with a basic baseline predictor and gradually improving it.

3.2 Baseline Predictor

At the beginning of the prediction, a simple predictor which does not consider the structure of the entire data like collaborative filtering does, is used. It uses simple averaging and finds biases in the data. Intuitively, it captures the tendency of applications for PII leakage and popularity of domains for receiving a PII. To expose such biases the least squares problem and its optimization solution is utilized. In this thesis context, there are biases for applications and domains, which define ratings relative to the average. Let $\bar{r}^l$ be the average rating for the leak l and b^l_i the bias of each application i relative to $\bar{r}^l$. Parameter b^l_j would be the bias for domain j. In other words, parameters b^l_i and b^l_j capture the deviations of application i and domain j compare to the average [17]. Assuming $\hat{r}^l_{ij}$ to be the predicted rating for leak l of pair $< app_i, domain_j >$, The *Baseline Predictor* (BL) for this pair is:

$$\hat{r}^l_{ij} = \bar{r}^l + b^l_i + b^l_j \tag{3.1}$$

For simplicity of notation, we remove superscript l, and we state our equations for one leak type.

To find the biases (b_i and b_j), one option is to solve Eq. (3.2) which minimizes the Root Mean Square Error (RMSE) of the predicted value and the real value of the ratings for training data ($< i, j >$ pairs) [17].

$$min_{\{b_i,b_j\}} \sum_{<i,j>} (r_{ij} - \hat{r}_{ij})^2 + \lambda(\sum_i b_i^2 + \sum_j b_j^2) \tag{3.2}$$

In Eq. (3.2), the second term ($\lambda(\sum_i b_i^2 + \sum_j b_j^2)$) is added to overcome the produced overfitting.

Removing these biases from the rating matrix eliminates the distorted view of application leaking patterns and provides a better prediction [7]. Therefore, the *Neighborhood Predictor* is built over the *error matrix* ($\tilde{\mathcal{R}}$) (Eq. (3.3)).

$$\tilde{\mathcal{R}} = \mathcal{R} - \hat{\mathcal{R}} \tag{3.3}$$

3.3 Neighborhood Predictor

Neighborhood model is a popular approach in recommendation system. The intuition is that the prediction can be made using information obtained from similar items and/or users. In the considering problem, we can define the neighborhood either in terms of the application or the domain similarities. This thesis chooses to take into account the application similarities with respect to the domains they contact. In particular, two apps are similar if they are sending packets to the same domains. In other words, similar applications exhibit similar leaks to the common domains. The neighborhood models are being used as an adjusting factor for the baseline. The first main property of neighborhood-based methods is *explainability* which tries to explain the reasons for predictions and the effectiveness of users' ratings to the movies. The next feature is its ability to handle *new ratings*. These kinds of models handle new incoming users to the system as soon as they provide

some ratings to the system [17]. This property is useful in privacy leak prediction context in which the goal is to predict the PII leakages of new application. Neighborhood relies on pairwise statistical correlation [7]. Pearson correlation, which is a general form for cosine similarity is used for this purpose.

The proposed model uses the corresponding rows of applications in the error matrix ($\tilde{\mathcal{R}}$), and performs the summations on the common domains for each PII leakage in the training data (k in Eq. (3.4)). Let matrix $\mathcal{D}$ be the similarity matrix of apps. Each entry d_{ij} of this matrix would be the similarity value for each pair of apps (Eq. (3.4)). For each application i, $\bar{\tilde{r}}_i$ is the mean value of ratings given by application i in matrix $\tilde{\mathcal{R}}$.

$$d_{ij} = \frac{\sum_k (\tilde{r}_{ik} - \bar{\tilde{r}}_i)(\tilde{r}_{jk} - \bar{\tilde{r}}_j)}{\sqrt{\sum_k (\tilde{r}_{ik} - \bar{\tilde{r}}_i)^2}\sqrt{\sum_k (\tilde{r}_{jk} - \bar{\tilde{r}}_j)^2}} \tag{3.4}$$

Using matrix $\mathcal{D}$, it is possible to define neighborhood L_i for each application i, which contains a set of applications that have the L largest values of $|d_{ij}|$ for all j's ($j \neq i$). The weighted sum of the ratings for the apps in their neighborhood is the neighborhood predictor. These weights correspond with the defined similarity. Using this weighted sum, which is normalized by the value of weights, we leverage Eq. (3.5) as the neighborhood predictor for each pair of $< app_i, domain_j >$.

$$\hat{r}_{ij}^N = \frac{\sum_{k \in L_i} d_{ik} \tilde{r}_{kj}}{\sum_{k \in L_i} |d_{ik}|} \tag{3.5}$$

Let the corresponding rating matrix for the *Neighborhood Predictor* be $\hat{\mathcal{R}}^N$.

Taking neighborhood information into account, the *Baseline & Neighborhood Predictor* (BL+N) is as follows:

$$\hat{\mathcal{R}}^{BL+N} = \hat{\mathcal{R}} + \hat{\mathcal{R}}^N \tag{3.6}$$

3.4 The Weighted Combination of the Predictors

Ensemble prediction exploits and can bring diverse predictors. One way of the combination is just doing a simple averaging. The more efficient way is to combine the predictors by learning an importance weight for each of them. This weight is proportional to the accuracy of each model. For the *Neighborhood Predictor*, we define

$$w_{ij}^{N} = \frac{\sum_{k \in L_i} d_{ik}}{\sum_{k \in L_i} d_{ik} + L_1}$$

as the weight of the predictor. The corresponding matrix for these weights is $\mathcal{W}^N$. The value L_1 is the parameter that needs to be estimated for each leak type. This weight is close to 1 for higher similarity within the neighborhood, and it is close to 0 for lower similarity. By adding BL to the weighted neighborhood, the weighted version (Eq. (3.7)) is defined. This thesis calls it *Baseline & Weighted Neighborhood Predictor* (BL+WN) for each leak type l. Note that $\mathcal{W}_l^N$ and $\hat{\mathcal{R}}_l^N$ are multiplied using the Hadamard product which is the element-wise multiplication.

$$\hat{\mathcal{R}}_l^{BL+WN} = \hat{\mathcal{R}}_l + \mathcal{W}_l^N \odot \hat{\mathcal{R}}_l^N \tag{3.7}$$

3.5 Weighted Correlated Leak Type Predictor

Another predictor which is referred to as *Weighted Correlated Leak Type Predictor* (WCLT) is the result of the observation in Fig. 2.5. This predictor is the ensemble of different *Baseline & Weighted Neighborhood Predictor* of correlated leak types. Likewise, for each leak type a similar weight is defined, which is proportional to the correlation of leak types:

$$w_{i,j}^{C} = \frac{corr(i,j)}{corr(i,j) + L_2}$$

Where i is the leak type we are predicting for, and j would be any leak type which correlates with i. L_2 needs to be estimated here as well. Putting it all together, the final predictor would be Eq. (3.8) which is defined as *Weighted Correlated Leak Type Predictor* (WCLT) for each leak type l. The set $\mathcal{L}$ is the set of all leak types that are correlated with leak type l.

$$\hat{\mathcal{R}}_l^{WCLT} = (\hat{\mathcal{R}}_l + \mathcal{W}_l^N \odot \hat{\mathcal{R}}_l^N) + \sum_{l' \in \mathcal{L}} w_{l,l'}^C \cdot (\hat{\mathcal{R}}_l + \mathcal{W}_l^N \odot \hat{\mathcal{R}}_l^N)$$

$$\hat{\mathcal{R}}_l^{WCLT} = \hat{\mathcal{R}}_l^{BL+WN} + \sum_{l' \in \mathcal{L}} w_{l,l'}^C \cdot \hat{\mathcal{R}}_{l'}^{BL+WN} \tag{3.8}$$

Chapter 4

Evaluation

4.1 Setup

Datasets. The datasets which are used in this thesis, are provided by [23, 25], and contain packets generated by different Android apps using automated interactions, as described in Chapter 2, Section 2.2. For *Antshield* dataset since it has information about background activities, only background activities are considered. For *App-Versions* dataset, since it contains different versions of applications, the corresponding domains of apps (regardless of versions) are aggregated. Then, for each leak type, a binary rating matrix is built, as described in Chapter 3, Section 3.1.

Tool. The implementation of the proposed predictors are with the help of *Surprise* [15], which is a Python library for recommendation system purposes. Also, for evaluation part which compares recommendation system results with baseline machine learning classifiers, the machine learning package of Python is utilized, which is provided by [20].

Parameters. All parameters, *i.e.,* λ (regularization term for baseline predictor), L_1, L_2 (for the neighborhood and correlation weights, respectively) and $|L|$ (the neighborhood size), are estimated

using 5-fold cross-validation. These parameters are learned for each leak type separately. In order to reduce the impact of overfitting and underfitting in the predictions, the hyper-parameters of machine learning classifiers (*i.e.,* decision tree, random forest and AdaBoost) are chosen using grid search and cross-validation techniques to find the best model.

Metrics. To evaluate the proposed model, 80% of non-missing data are sampled as the training data and the rest as the test data. To see how well the proposed models can predict ratings for each leak type, RMSE for the test set is used as a measurement. The experiment is run multiple times using cross-validation with different test sets and average them. Also, to evaluate the performance of the top-N recommendation system the F1 score and hit ratio are used.

Fig. 4.1 and 4.2 illustrate and compare the RMSE of different predictors per leak type for both datasets (sorted by the number of leaks present in the dataset). Using the weighted neighborhood and the correlation of leak types, the error can be improved. Also, Fig. 4.1 shows for most of the leak types (*e.g., Android ID*, *Advertiser ID*, and *Serial Number*) WCLT has a slightly better error compared to BL+WN.

As shown in Fig. 4.1 and 4.2, as the number of leaks decreases, there is a reduction in the average RMSE values. Therefore, the lower values of RMSE for the leaks on the right side of Fig. 4.1 and 4.2 are due to the lack of data points inside the rating matrix. Hence, most of the values are zero, and also the predicted values are close to zero. This sparsity of PII data points causes the lower RMSE. In general, as our predictor sees more leaks in the training phase, it should perform better (due to reduction in overfitting). Therefore, although RMSE is low, some of leak types like *First Name*, suffer from a poor predictor. Hence, RMSE cannot be a good representative of how well a specific model performs. To arrive at a better evaluation for the models, the performance is measured by finding the *top-N recommendations*.

In this thesis context, for each data point in pair of $< app_i, domain_j >$ from the test set, the ratings given by app_i for $domain_j$ and the rest of domains using the considering model is predicted. After

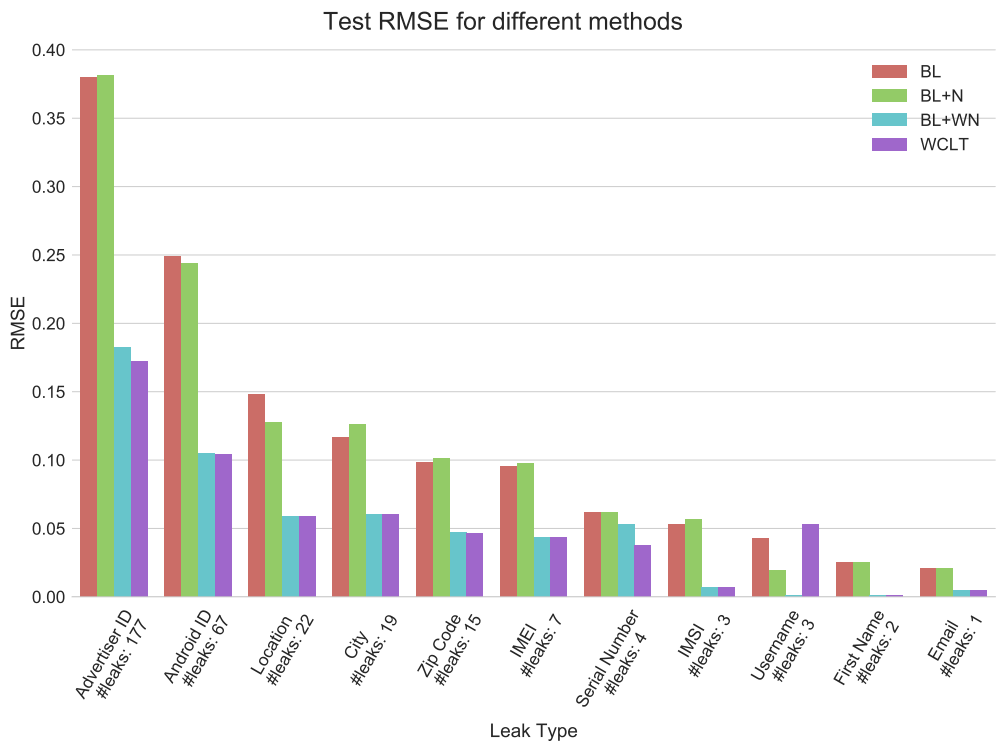


Figure 4.1: Average RMSE of the test sets for each predictor per leak type (*Antshield* dataset)

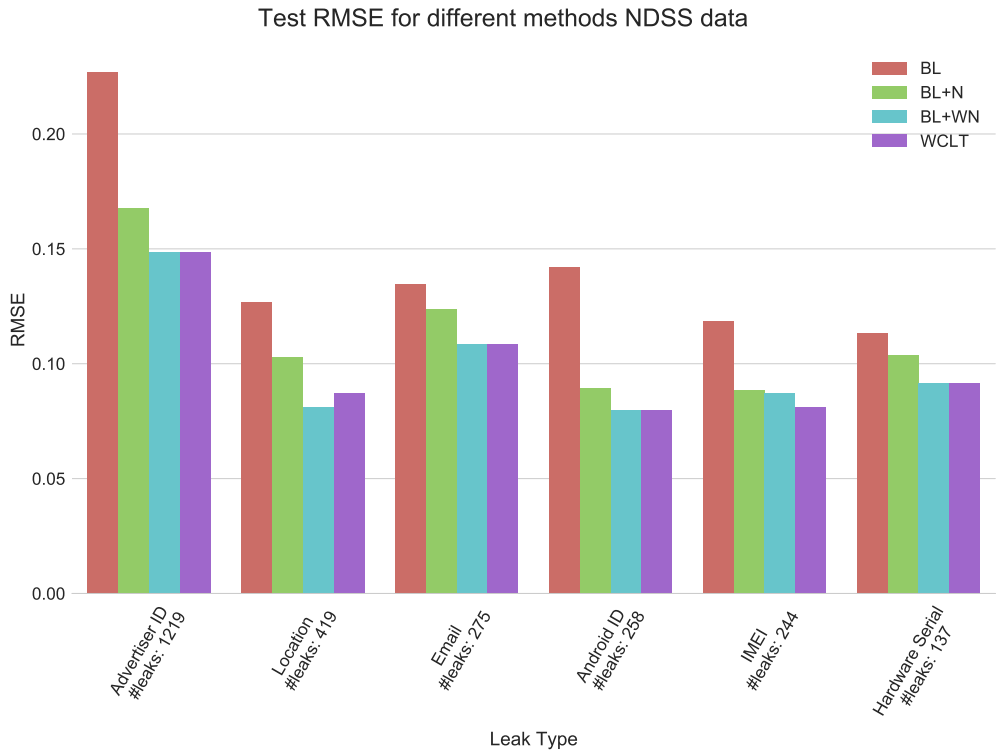


Figure 4.2: Average RMSE of the test sets for each predictor per leak type (*App-Versions* dataset)

sorting this list in descending order, we have a ranked list of ratings. By picking the first N items from this ranked list, we have the top-N recommendations of each model. When $domain_j$ is within this N-sized window of recommendations, then $domain_j$ is recommended to app_i and we have a *hit*.

The classification (confusion matrix) of the results for these recommendations is stated in Table 4.1 which is given by [14].

	Recommended	Not recommended
Preferred	True-Positive (TP)	False-Negative (FN)
Not preferred	False-Positive (FP)	True-Negative (TN)

Table 4.1: Classification of results for the top-N recommendation system

Also, following metrics can be computed using values stated in Table 4.1.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \quad \text{True Positive Rate (Recall)} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{False Positive Rate} = \frac{\text{FP}}{\text{FP} + \text{TN}}, \quad \text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{FP} + \text{TP} + \text{FN} + \text{TN}}$$

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

4.2 Using the Correlation of Leak Types

Fig. 4.3 and 4.4 show the comparison of WCLT, BL+WN, BL+N and BL predictors of *Advertiser ID* and *Location* for *Antshield* dataset (test set). As illustrated in these figures, WCLT performs better than the other predictors after some N ($N = 23$ and $N = 7$ for *Advertiser ID* and *Location*, respectively) in terms of F1 score (Fig. 4.3a and 4.4a). The reason is the lower rate of false positives for aforementioned leak types after these N values (Fig. 4.3b and 4.4b). Since *Location* has a higher correlation with the other leak types (Fig. 2.5a), its WCLT benefits from a higher improvement compare to other leak types, like *Advertiser ID*. Fig. 4.3c and 4.4c which is the rate

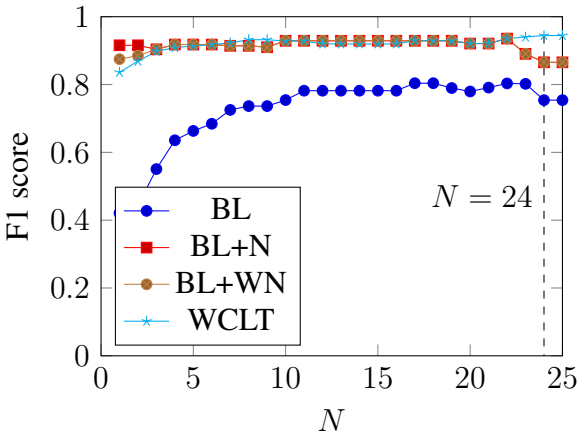
of true positives *vs.* false positives, illustrate this fact. Also, area under curve (AUC) for these leak types illustrated in Fig. 4.3d and 4.4d. Also, in terms of accuracy, as illustrate in Fig. 4.3e and Fig. 4.4e, WCLT performs better compared to the other recommendation system models.

Fig. 4.5 illustrates comparison of our different recommender system models of *Advertiser ID* for *App-Versions* dataset. As Fig. 4.5a shows, by using correlation of leak types, F1 score improves (after $N = 10$). That is because of the lower rate of false positives after $N = 10$ (4.5b). Also, based on Fig. 4.5c, the ratio of true positives *vs.* false positives is higher for WCLT compared to other predictors. Fig. 4.5d shows the AUC values for these models. In terms of accuracy, as illustrated in Fig. 4.5e, WCLT has a higher accuracy per each N compared to other proposed recommendation system models (after $N = 4$).

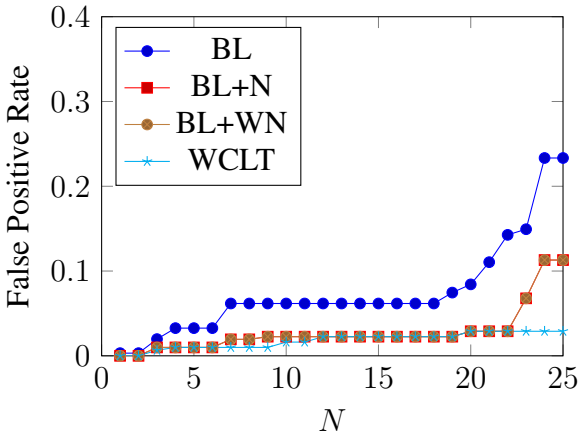
4.3 Optimal WCLT

In order to compare the proposed recommendation system with other machine learning models, the value of N with the highest F1 score for WCLT is chosen and fixed. Hence, the results are provided based on the best value of N for each leak type throughout the rest of this thesis. Besides, in order to have a more fair comparison, the results for the first six leak types in terms of number of PII leaks in the scope of considering leak types, are reported for both datasets.

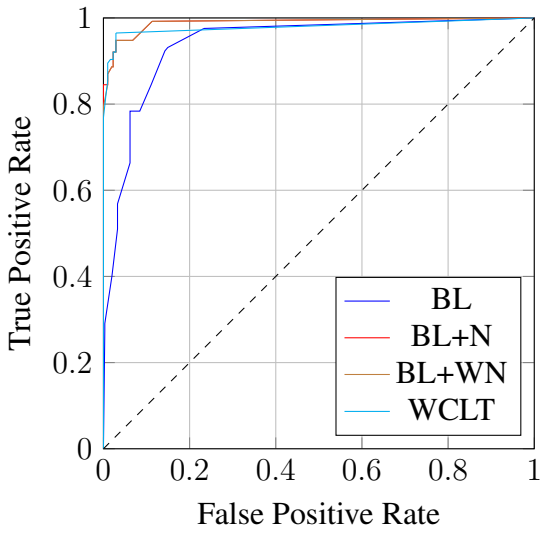
Fig. 4.6 and 4.7 show the F1 score (a) and the average hit ratio (b) for WCLT per leak types (having the same order as in Fig. 4.1 and 4.2) for a given N (N with the best F1 score). The F1 score in Fig. 4.6a and 4.7a, indicates the rate of true positives compared to the false positives and false negatives. Fig. 4.6b and 4.7b show the average ratio of leaks captured by the model compared to the number of leaks exist in the test set (hit ratio). For example in Fig. 4.6b, on average WCLT with $N = 24$ (which has the best F1 score between different values of N) reveals 96% of packets containing *Advertiser ID*.



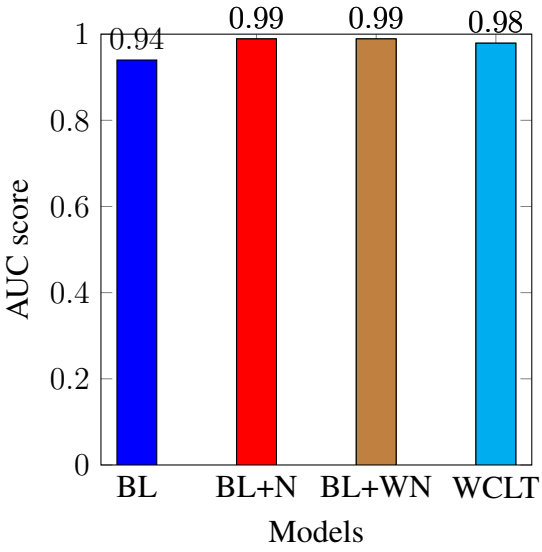
(a) F1 score of *Advertiser ID*



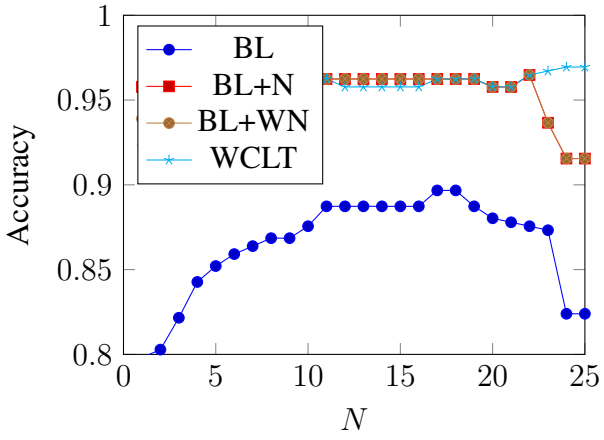
(b) False positive rate score of *Advertiser ID*



(c) ROC curve of *Advertiser ID*

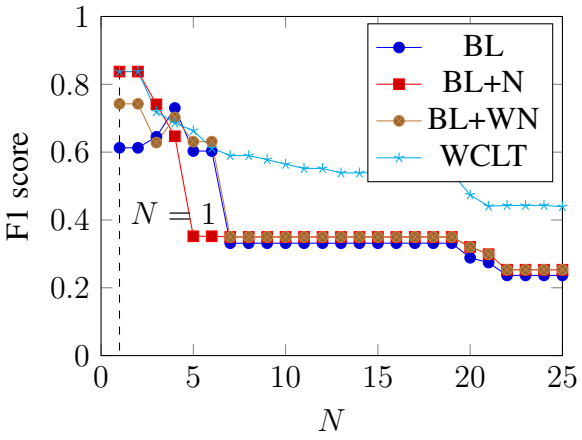


(d) AUC score of *Advertiser ID*

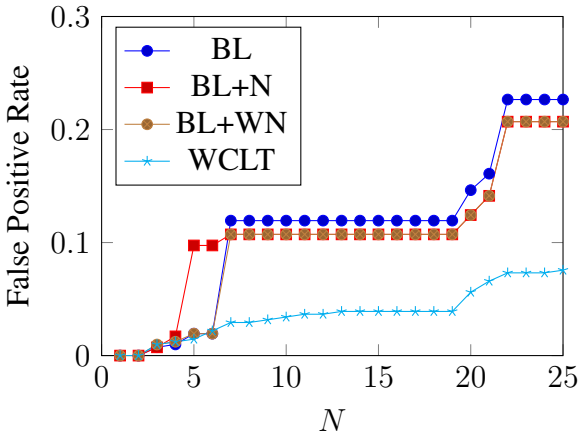


(e) Accuracy of *Advertiser ID*

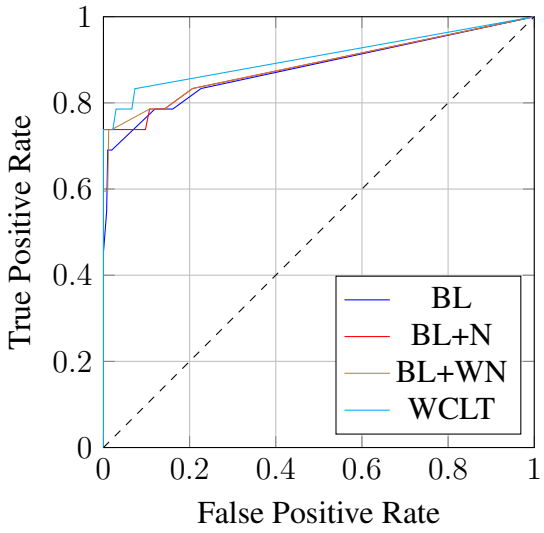
Figure 4.3: Performance of different recommender system models of *Advertiser ID* in *Antshield* dataset



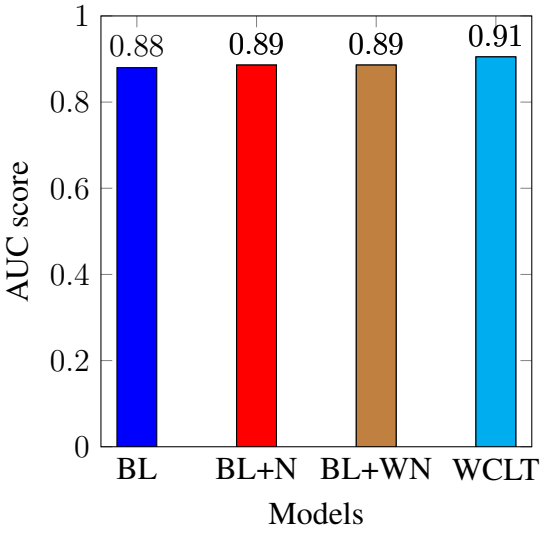
(a) F1 score of *Location*



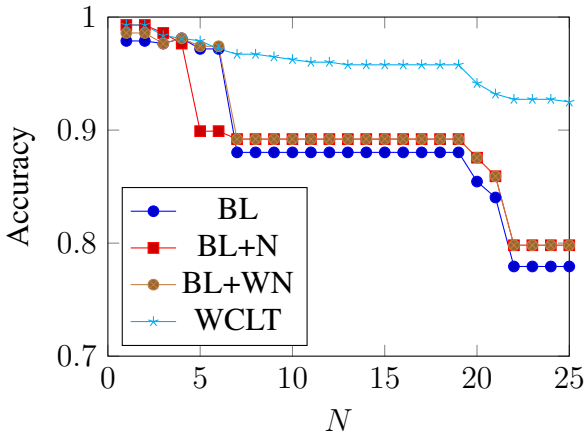
(b) False positive rate of *Location*



(c) ROC curve of *Location*

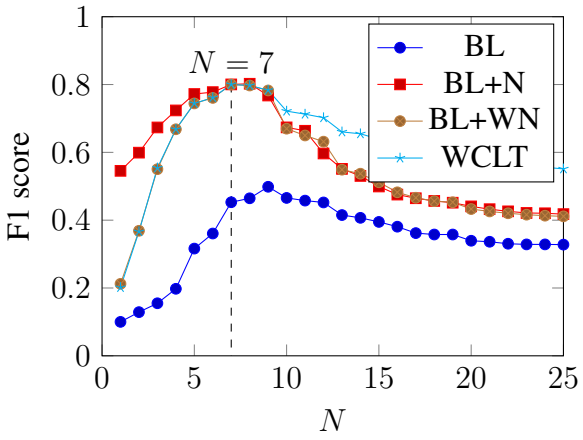


(d) AUC score of *Location*

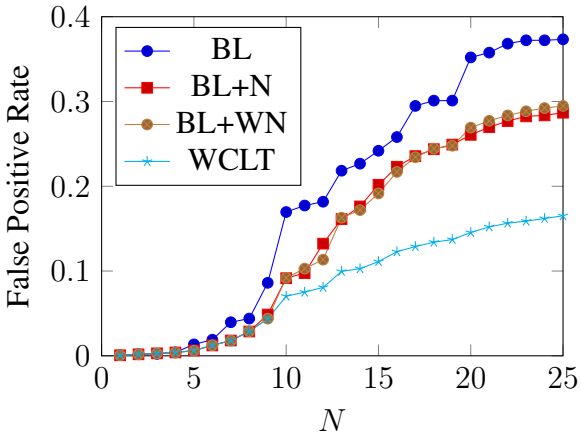


(e) Accuracy of *Location*

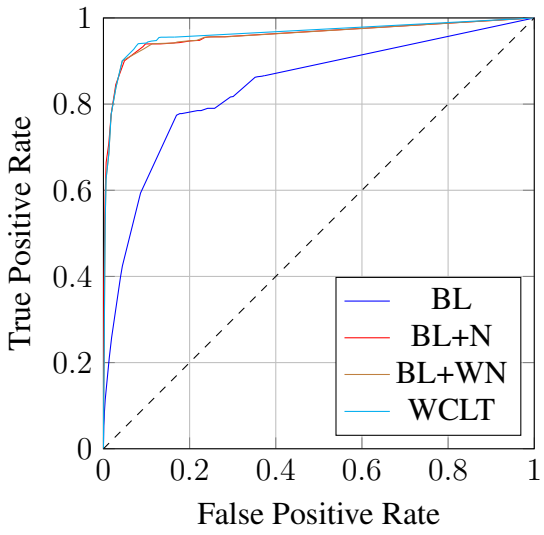
Figure 4.4: Performance of different recommender system models of *Location* in *Antshield* dataset



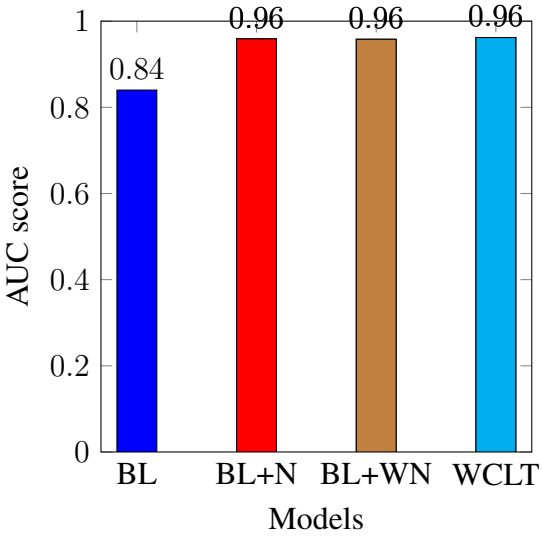
(a) F1 score of *Advertiser ID*



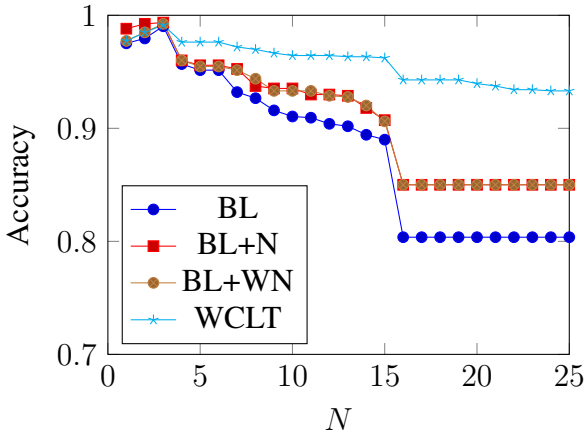
(b) False positive rate of *Advertiser ID*



(c) ROC curve of *Advertiser ID*



(d) AUC score of *Advertiser ID*



(e) Accuracy of *Advertiser ID*

Figure 4.5: Performance of different recommender system models of *Advertiser ID* in *App-Versions* dataset

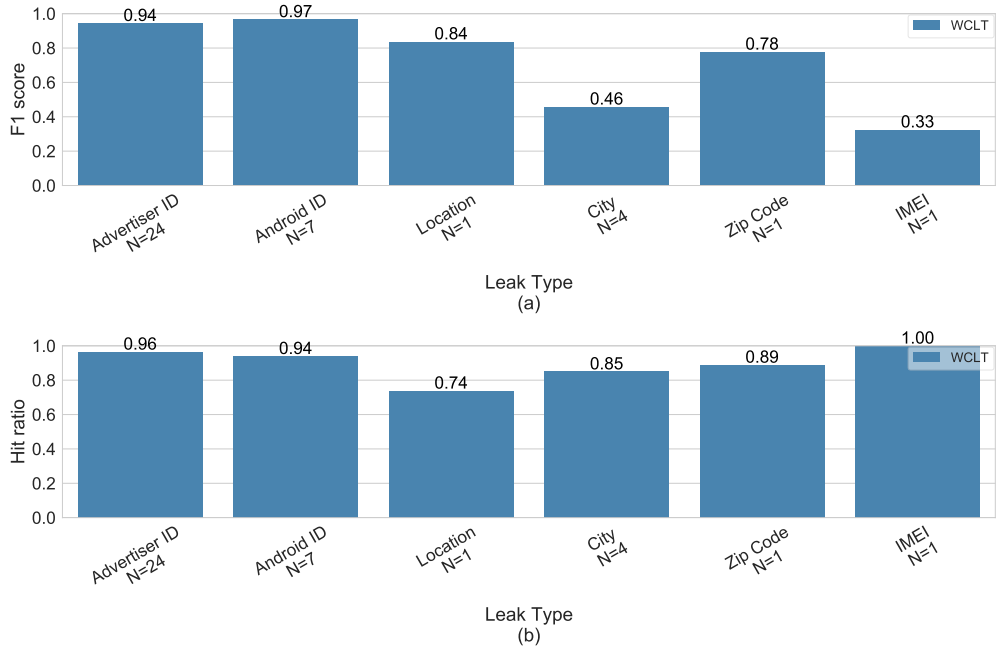


Figure 4.6: Average F1 score (a) and hit ratio (b) of *Weighted Correlated Leak Type Predictor* for the first 6 leak types given the best N (N has been annotated for each leak type) for *Antshield* dataset

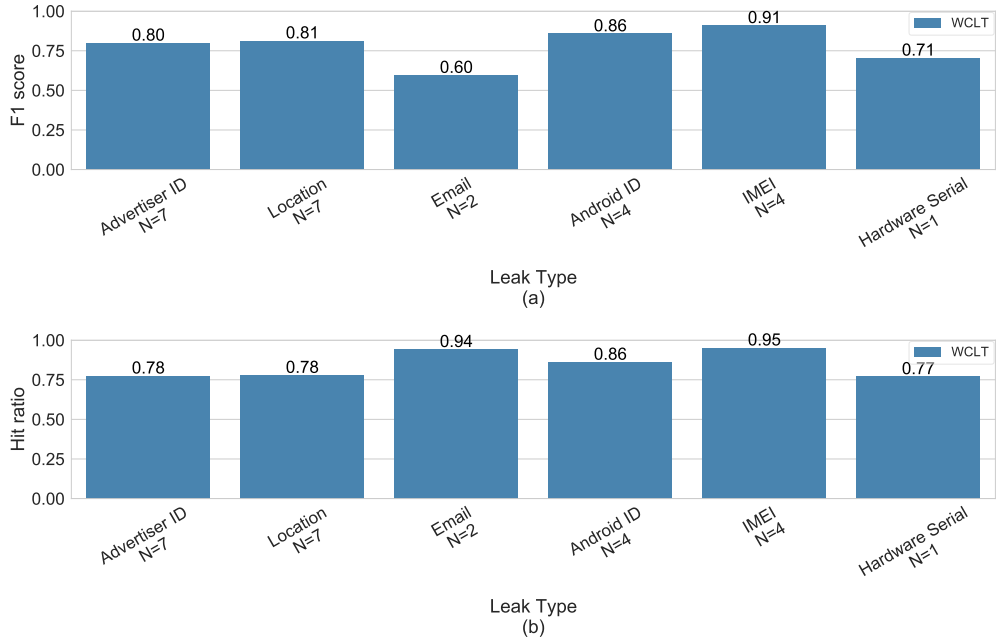


Figure 4.7: Average F1 score (a) and hit ratio (b) of *Weighted Correlated Leak Type Predictor* for the first 6 leak types given the best N (N has been annotated for each leak type) for *App-Versions* dataset

4.4 Comparison of WCLT to Baselines

In this section, the optimal WCLT model is compared to the state-of-the-art that uses machine learning models, namely decision tree and AdaBoost classifiers. To train the classifiers the apps and domains are used in one-hot representation as features, which are the same inputs for the recommendation system framework. Also, random forests are trained. Because of the structure of data, which has apps and domains as features, random forests suffer the lousy performance. So, the results of decision tree and AdaBoost are reported here. As illustrated in Tables 4.2 and 4.3, AdaBoost has no significant improvement over our datasets. So, only decision trees are compared with WCLT model. For most cases like *Advertiser ID* in *Antshield* dataset and *Hardware Serial* in *App-Versions* dataset, F1 score of WCLT is better than decision tree, as mentioned in Tables 4.2 and 4.3. Also based on Table 4.2 for *Antshield* dataset, WCLT has lower false negative rates in comparison with decision tree, for most cases. In terms of false positive rate there is no significant improvement by using WCLT over *Antshield* dataset. Similarly, based on Table 4.3 for *App-Versions* dataset, WCLT ends up with a lower false negative rate for most of the leak types (*e.g., IMEI*). However, both models have nearly similar false positive rates. Since the goal is to predict PII leaks, false negatives which are misclassified PII leaks, are more important than false positives. So, WCLT is a better option when we are considering only apps and domains as features.

4.5 WCLT Compared to Deep-packet Inspection

As mentioned in Chapter 1, state-of-the-art uses deep-packet inspection to predict PII leaks ([24, 25, 27]). In DPI, beside apps and domains, they extract more features from packet headers and/or payload [27]. By means of these packet-based features (bag of words from key-values extracted from each packet, Fig. 2.1), classifiers are being trained. The author trains decision tree classifiers given by [25] on only *Antshield* dataset, since the header and payload information are available for

Method	Leak Type	F1	Accuracy	FN Rate	Recall	Precision	FP Rate
Decision Tree	*Advertiser ID*	0.43	0.74	0.65	0.34	0.60	0.08
	Android ID	0.5	0.94	0.6	0.4	0.66	0.01
	Location	0.90	0.99	0.16	0.83	1	0.0
	City	0.44	0.96	0.5	0.5	0.4	0.02
	Zip Code	0.66	0.98	0.33	0.66	0.66	0.007
	IMEI	0.0	0.97	1	0.0	0.0	0.0
Ada Boosting	*Advertiser ID*	0.45	0.74	0.63	0.36	0.60	0.09
	Android ID	0.5	0.94	0.6	0.4	0.66	0.01
	Location	0.9	0.99	0.16	0.83	1	0
	City	0.44	0.96	0.5	0.5	0.40	0.02
	Zip Code	0.66	0.98	0.33	0.67	0.67	0.007
	IMEI	0.0	0.97	1	0.0	0.0	0.0
WCLT (best N)	*Advertiser ID* ($N = 24$)	0.94	0.96	0.035	0.96	0.92	0.11
	Android ID ($N = 7$)	0.97	0.99	0.06	0.93	1	0.002
	Location ($N = 1$)	0.84	0.99	0.26	0.73	1	0.0
	City ($N = 4$)	0.46	0.94	0.15	0.85	0.31	0.08
	Zip Code ($N = 1$)	0.78	0.99	0.11	0.88	0.72	0.009
	IMEI ($N = 1$)	0.33	0.98	0.0	1	0.25	0.0

Table 4.2: Comparison of WCLT over the best N with decision tree and AdaBoost trained on *Antshield* dataset using apps and domains as features

Method	Leak Type	F1	Accuracy	FN Rate	Recall	Precision	FP Rate
Decision Tree	*Advertiser ID*	0.8	0.96	0.22	0.77	0.84	0.015
	Location	0.78	0.98	0.27	0.72	0.84	0.004
	Email	0.69	0.98	0.36	0.63	0.76	0.007
	Android ID	0.84	0.98	0.17	0.82	0.86	0.004
	IMEI	0.86	0.98	0.15	0.84	0.88	0.004
	Hardware Serial	0.62	0.98	0.44	0.55	0.71	0.004
Ada Boosting	*Advertiser ID*	0.8	0.96	0.22	0.77	0.83	0.016
	Location	0.77	0.98	0.27	0.72	0.83	0.004
	Email	0.69	0.98	0.36	0.63	0.76	0.007
	Android ID	0.84	0.98	0.17	0.82	0.86	0.004
	IMEI	0.86	0.98	0.15	0.84	0.88	0.004
	Hardware Serial	0.62	0.98	0.43	0.57	0.68	0.005
WCLT (best N)	*Advertiser ID* ($N = 7$)	0.80	0.96	0.22	0.77	0.82	0.018
	Location ($N = 7$)	0.81	0.98	0.22	0.77	0.85	0.004
	Email ($N = 2$)	0.60	0.96	0.05	0.94	0.43	0.03
	Android ID ($N = 4$)	0.86	0.99	0.13	0.86	0.85	0.006
	IMEI ($N = 4$)	0.91	0.99	0.04	0.95	0.87	0.005
	Hardware Serial ($N = 1$)	0.71	0.98	0.22	0.77	0.65	0.008

Table 4.3: Comparison of WCLT over the best N with decision tree and AdaBoost trained on *App-Versions* dataset using apps and domains as features

this dataset. The measurements of this experiment are summarized in Table 4.4. Since we have more PII leaks for the first two leak types (*i.e., Advertiser ID* and *Android ID*), the F1 score is close

enough to the DPI approach. For the rest of leak types which we have fewer PII leaks, by adding more features using deep-packet inspection we will have more accurate and precise models with higher F1 score, which is not surprising. Basically, by adding more rules (features) inside decision trees, the rate of false positives decreases. Therefore, higher precision leads to a higher F1 score for the prediction model. Comparison between Table 4.2 and Table 4.4 confirms these findings.

Method	Leak Type	F1	Accuracy	Recall	Precision
Decision Tree	*Advertiser ID*	0.96	0.96	0.94	0.98
	Android ID	0.94	0.94	0.92	0.97
	Location	0.98	0.98	0.97	0.99
	City	0.95	0.95	0.92	0.98
	Zip Code	0.92	0.92	0.91	0.94
	IMEI	0.96	0.96	0.94	0.97

Table 4.4: Evaluation of decision tree trained on *Antshield* dataset using deep-packet inspection and key-value pairs in header/payload as features

Chapter 5

Related Work

Leak Detection. Detection of privacy leaks by mobile applications has been previously addressed by three different approaches. The *static analysis* approach inspects decompiled .apk files without any runtime analysis to capture privacy violations by analysis of the control flow graph and tracking Android callbacks which are responsible for returning various information like location [4, 10, 12, 13] and mining permission-set of applications and their corresponding API calls [30]. Furthermore, some of the libraries which are used by the app developers are a source of potential privacy leaks [18].

In the *dynamic analysis* approaches, they perform runtime analysis to detect information leakage. Running such an examination requires rooting the Android device to track memory references [23]. TaintDroid [9] and VetDroid [34] are commonly used in the dynamic analysis. DroidScope [32] captures more detailed native and Dalvik instruction traces regarding information leakage through Java and native components. Also, dynamic analysis approach has been explored by [8].

Finally, the *network traffic analysis* or *VPN-based* approaches monitor all the Internet traffic of the mobile device in order to capture PII contained in IP traffic. The datasets which have been analyzed in this thesis were obtained using this approach. State-of-the-art tools using this approach

are PrivacyGuard [29], AntMonitor [26], HayStack [22], and ReCon [24] which uses machine learning to identify PIIs. Recently, a machine learning approach was applied to detect ad-requests, in addition to PIIs, in outgoing packets [27].

The main drawback of these approaches is that they all suffer the overhead of analysis. For example, static analysis has the decompilation process and analysis of the code. In the dynamic analysis, we have to root the phone and monitor flows, and finally, in VPN-based techniques, DPI costs a lot in terms of resources and computations. In this thesis, a new lightweight method, which uses minimal information related to each packet (namely the application package name and destination domain) is proposed, that has a lower overhead, which is important when applied in real-time on every outgoing packet on the mobile device. For packets where a potential PII is detected, a more heavyweight and accurate (DPI-based) mechanism can be applied, but this would be only a subset of the packets. Furthermore, with the increased use of encryption, it is important to rely on minimal, easily accessible features to detect PII.

Similar Datasets. There are similar datasets to the ones that are used in this thesis, typically captured using a VPN-based approach. The study of [21] used a dataset that has been collected using *Lumen* [22], to detect third-party advertising and tracking services. This dataset has information of flows as wells as app permissions. To identify PII leakage using machine learning, ReCon [24] has collected packet traces as well.

Blacklisting. There is also prior work on analyzing IDS logs and blacklisting malicious sources which follows a similar approach: they predict future attacks based on the observed attacks. In Highly Predictive Blacklisting (HPB) [33] Zhang et al. have defined similarity between victims based on their common attackers and have formulated the problem of future attack prediction as Google's PageRank using this similarity metric. Soldo et al. [28] predict future attacks using a recommendation system setup. Katti et al. [16] have studied the concept of correlated attacks by considering the correlation among victims. Freudiger el al. [11] have studied the forecasting attack sources based on the collaboration of victims in a privacy-preserving way.

Chapter 6

Conclusion

In this thesis, the problem of privacy leak prediction, on a per-packet basis based on only two features, as a recommendation system has been studied. The proposed predictors have been designed inspired by the leak patterns observed in two relevant dataset ([23,25]). The proposed recommendation system performs well for most of the leak types, and it can successfully detect 89% and 84% of considering leak types on average for *Antshield* and *App-Versions* datasets while the best F1 score achieved was 0.97 and 0.91 for *Android ID* and *IMEI* detection, respectively.

Directions for future work include the following: (1) look into other datasets that provide information about the underlying causes of the observed patterns (*e.g.,* libraries and SDKs which were not available in our dataset) (2) incorporate additional features external to the network dataset (*e.g.,* category of apps etc.) (3) implement our system on a mobile device as a first-line of defense which can trigger further deep-packet inspections.

Bibliography

[1] Mitmproxy. https://mitmproxy.org/.

[2] Netflix Prize. https://netflixprize.com/.

[3] Ui/application exerciser monkey. https://developer.android.com/studio/test/monkey.html.

[4] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *Acm Sigplan Notices*, 49(6):259–269, 2014.

[5] M. Bastian, S. Heymann, M. Jacomy, et al. Gephi: an open source software for exploring and manipulating networks. *Icwsm*, 8(2009):361–362, 2009.

[6] G. Chatzopoulou, M. Eirinaki, and N. Polyzotis. Query recommendations for interactive database exploration. In *International Conference on Scientific and Statistical Database Management*, pages 3–18. Springer, 2009.

[7] M. Chiang. *Networked Life: 20 Questions and Answers*. Cambridge University Press, 2012.

[8] S. Chitkara, N. Gothoskar, S. Harish, J. I. Hong, and Y. Agarwal. Does this app really need my location?: Context-aware privacy management for smartphones. *Proc. of UBICOMP*, 1(3):42, 2017.

[9] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *TOCS*, 32(2):5, 2014.

[10] Y. Feng, S. Anand, I. Dillig, and A. Aiken. Apposcopy: Semantics-based detection of android malware through static analysis. In *Proc. of the 22nd ACM SIGSOFT Int. Symp. on Foundations of Software Eng.*, pages 576–587. ACM, 2014.

[11] J. Freudiger, E. De Cristofaro, and A. E. Brito. Controlled data sharing for collaborative predictive blacklisting. In *DIMVA*, pages 327–349. Springer, 2015.

[12] A. P. Fuchs, A. Chaudhuri, and J. S. Foster. Scandroid: Automated security certification of android. Technical report, 2009.

[13] C. Gibler, J. Crussell, J. Erickson, and H. Chen. Androidleaks: automatically detecting potential privacy leaks in android applications on a large scale. In *TRUST*, pages 291–307. Springer, 2012.

[14] A. Gunawardana and G. Shani. A survey of accuracy evaluation metrics of recommendation tasks. *J. of Machine Learning Res.*, 10(Dec):2935–2962, 2009.

[15] N. Hug. Surprise, a Python library for recommender systems. `http://surpriselib.com`, 2017.

[16] S. Katti, B. Krishnamurthy, and D. Katabi. Collaborating against common enemies. In *Proc. of the 5th ACM SIGCOMM conf. on Internet Measurement*, pages 34–34. USENIX Association, 2005.

[17] Y. Koren. Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM TKDD*, 4(1):1, 2010.

[18] I. Leontiadis, C. Efstratiou, M. Picone, and C. Mascolo. Don't kill my ads!: balancing privacy in an ad-supported mobile application market. In *Proc. of ACM HotMobile*, page 2. ACM, 2012.

[19] G. Linden, B. Smith, and J. York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 7(1):76–80, 2003.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[21] A. Razaghpanah, R. Nithyanand, N. Vallina-Rodriguez, S. Sundaresan, M. Allman, C. Kreibich, and P. Gill. Apps, trackers, privacy, and regulators: A global study of the mobile tracking ecosystem. 2018.

[22] A. Razaghpanah, N. Vallina-Rodriguez, S. Sundaresan, C. Kreibich, P. Gill, M. Allman, and V. Paxson. Haystack: In situ mobile traffic analysis in user space. *ArXiv e-prints*, 2015.

[23] J. Ren, M. Lindorfer, D. J. Dubois, A. Rao, D. R. Choffnes, and N. Vallina-Rodriguez. Bug fixes, improvements, ... and privacy leaks: A longitudinal study of pii leaks across android app versions. 2018.

[24] J. Ren, A. Rao, M. Lindorfer, A. Legout, and D. Choffnes. Recon: Revealing and controlling pii leaks in mobile network traffic. In *Proc. of MobiSys*, pages 361–374. ACM, 2016.

[25] A. Shuba, E. Bakopoulou, M. A. Mehrabadi, H. Le, D. Choffnes, and A. Markopoulou. Antshield: On-device detection of personal information exposure. *arXiv:1803.01261*, 2018.

[26] A. Shuba, A. Le, E. Alimpertis, M. Gjoka, and A. Markopoulou. Antmonitor: System and applications. *arXiv:1611.04268*, 2016.

[27] A. Shuba, A. Markopoulou, and Z. Shafiq. Nomoads: Effective and efficient cross-app mobile ad-blocking. *Proc. on PETS*, 4:125–140, 2018.

[28] F. Soldo, A. Le, and A. Markopoulou. Predictive blacklisting as an implicit recommendation system. In *In Proc. of IEEE INFOCOM*, pages 1–9, 2010.

[29] Y. Song and U. Hengartner. Privacyguard: A vpn-based platform to detect information leakage on android devices. In *Proc. of ACM CCS*, pages 15–26. ACM, 2015.

[30] G. Tao, Z. Zheng, Z. Guo, and M. R. Lyu. Malpat: Mining patterns of malicious and benign android apps via permission-related apis. *IEEE Trans. on Reliability*, 67(1):355–369, 2018.

[31] K. Tserpes, F. Aisopos, D. Kyriazis, and T. Varvarigou. A recommender mechanism for service selection in service-oriented environments. *Future Generation Computer Systems*, 28(8):1285–1294, 2012.

[32] L. Yan and H. Yin. Droidscope: Seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis. In *USENIX security symp.*, pages 569–584, 2012.

[33] J. Zhang, P. A. Porras, and J. Ullrich. Highly predictive blacklisting. In *USENIX Security Symp.*, pages 107–122, 2008.

[34] Y. Zhang, M. Yang, B. Xu, Z. Yang, G. Gu, P. Ning, X. S. Wang, and B. Zang. Vetting undesirable behaviors in android apps with permission use analysis. In *Proc. of ACM SIGSAC*, pages 611–622. ACM, 2013.