Privacy Leak Classification on Mobile Devices

Anastasia Shuba, Evita Bakopoulou, Athina Markopoulou
EECS Dept, UC Irvine
{*ashuba, ebakopou, athina*}*@uci.edu*

***Abstract*—Mobile devices have access to personal, potentially sensitive data, and there is a growing number of mobile apps that have access to it and often transmit this personally identifiable information (PII) over the network. In this paper, we present an approach for detecting such PII "leaks" in network packets going out of the device, by first monitoring network packets on the device itself and then applying classifiers that can predict with high accuracy whether a packet contains a PII leak and of which type. We evaluate the performance of our classifiers using datasets that we collected and analyzed from scratch. We also report preliminary results that show that collaboration among users can further improve classification accuracy, thus motivating crowdsourcing and/or distributed learning of privacy leaks.**

***Index Terms*—machine learning, data analytics, privacy**

I. Introduction

Mobile devices have access to a wealth of personal, potentially sensitive information and there is a growing number of mobile apps that access, process and transmit some of this information over the network to remote servers. Sometimes this is justified (required for the intended operation of the applications, *e.g.,* location is needed by GoogleMaps) and controllable (*e.g.,* by the user through permissions); but for the most part, users are not in control of their data today. Applications and third party libraries routinely transmit user data to remote servers, including advertising servers and trackers, and users are typically unaware of how their personal data is shared and for what purpose.

To address this problem, we take a network centric approach. Since, by definition, personal information is transmitted from mobile apps over the network interface towards remote servers, network traffic is a natural vantage point for comprehensive detection and control of such leaks. We develop AntShield – an approach that (i) intercepts outgoing network packets on the device, and (ii) employs a hybrid string matching and classification approach to detect leaks of personally identifiable information (PII). The contributions of this paper are the following:

- *Classification Methodology.* We present a *multi-label* classification methodology (Binary Relevance with Decision Trees) that leverages the information available on the device but also meets the resource constraints and requirements of the mobile device. Using a dataset that we collected from scratch, we show that it achieves significantly higher accuracy (8-25% improvement) and lower variance (a factor of 2-5) compared to state-of-the-art. We also design and advocate for per-app, instead of prior per-domain, classifiers: they achieve similar classification accuracy, but allow faster and more scalable operation while covering more traffic.
- *Collaboration.* We also experimented with training and testing classifiers on different real users. Preliminary results showcase the potential of collaboration among users to further improve classification accuracy.

PII leak detection can provide users with transparency about where their data is going, and can enable action (*e.g.,* blocking or obfuscating the transmitted PII). Furthermore, this work is a first step towards a distributed system for detecting of privacy leaks, which is still an open problem.

The structure of the rest of the paper is as follows. Section II reviews closely related work. Section III describes the problem of PII leakage and the approach AntShield is taking. Section IV describes the classification methodology in detail (while system aspects of AntShield are deferred to [6]). Section V presents our collected dataset and evaluates AntShield's classification accuracy. Section VI presents preliminary results on collaboration among users to further improve classification accuracy, and Section VII concludes the paper.

II. Related Work

In this paper, we focus only on network-centric approaches that identify PII leaks at the packet level, and we do not discuss other approaches, such as custom OS, rooting the phone, etc. One design decision is *where* to intercept and inspect network packets: in the middle of the network (as in Recon [5]) or on the device itself (as in AntMonitor [7] and Lumen (a.k.a. Haystack) [4]). Another important design decision is *what* information to extract from the packet in order to detect whether it contains a privacy leak, and the state-of-the-art consists of two complementary approaches. On one hand, AntMonitor [7] and Lumen [4], keep a blacklist of strings (potential PII leaks) and perform deep packet inspection (DPI) on each packet to match any PII strings in headers and/or payload; therefore, they are unable to detect leakage of information that changes dynamically, is not part of the list, or is obfuscated. On the other hand, Recon [5] trained classifiers to detect PII within packets, but relied on a fully centralized approach: traffic was routed to and analyzed at a remote server, potentially impacting scalability and security.

Our AntShield approach combines the best of both worlds: it can be applied on the device (in addition to the middle of the network) and it uses machine learning (which finds PII without a priori knowledge). Operating on the device presents both opportunities and challenges.[1] The closest related to this work, and our baseline for comparison, is Recon [5]. Our work builds on and improves over it by (i) leveraging information accessible on the device and using multi-label

This work has been supported by NSF Award 1649372, a DTL Grant 2016, and CPCC at UCI. A. Shuba has been partially supported by an ARCS Fellowship, and E. Bakopoulou has been partially supported by a H. Samueli and a NetSYS Fellowship.

[1] On the upside, it obviates the need for a trusted infrastructure and gives full control to the user, which we believe is the right approach in privacy. Devices also have access to important contextual information, such as which mobile apps generate the packets in question. On the downside, mobile devices have limited resources to conduct traffic analysis using DPI and machine learning.

classification (instead of a combination of binary classification and heuristics) to improve classification accuracy; and (ii) applying classifiers in real-time on the mobile device for the first time.

III. Problem and Approach

Personally Identifiable Information (PII). In this paper we are interested in the following PII:

- Device Identifiers: IMEI, Device ID, phone number, serial number, ICCID, MAC Address.
- User identifiers: credentials (per app, usually transmitted over HTTPS), Advertiser ID, email.
- User demographics: first/last name, gender, zipcode, city, *etc.* - unavailable through Android APIs.
- Location: latitude and longitude coordinates, available through Android APIs.
- User-defined: the user can also define any custom string that should be monitored, such as digits of her credit card.

We refer to the transmission of a packet from the device to the network, containing at least one PII, as a *privacy leak or exposure*.[2] Our goal is to detect such privacy leaks in packets going out of a device accurately and in real-time.

System. AntShield is a regular mobile app that runs in the background and is able to intercept and inspect packets before forwarding them over to their intended destinations. IP packets are the input to our methodology, and the classification of each packet to one or more PII types it may contain is the output.

AntShield leverages the AntMonitor Library (which we previously developed in [7]) to intercept and inspect network traffic. AntMonitor Library's VPN-based implementation is lean and allows AntShield to inspect packets in real-time (~1ms per packet), including encrypted packets (via a TLS proxy), without impacting user experience (see [6] and [7]). AntShield is able detect PII in outgoing packets and then provide users with an option to take some action (*e.g.,* block or obfuscate PII before forwarding the packet). [3]

Although challenging, the design and evaluation of the AntShield system is out of scope and details are deferred to the technical report [6]; except for constraints that affect the classification methodology, which is the focus of this paper.

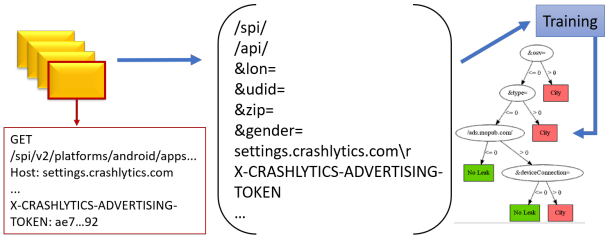
Packet Classification. In this paper, we train and apply packet classifiers to network packets (at the IP layer), in real-time on the mobile device, to detect PII leaks. Learning is a powerful approach for this problem because it can predict leaks without prior knowledge of the actual user's private information. For example, we do not need to know the user's email to detect the fact that the user's email is leaking.

IV. Classification Methodology

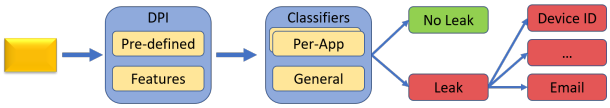
Our **methodology** is depicted in Fig. 1, and it consists of the following steps.

[2]This transmission may be: (i) benign, *e.g.,* necessary for the operation of the app, acceptable to the user; (ii) of the honest-but-curious nature; or (iii) intended to collect information about the user. Distinguishing between privacy *exposure* and an actual privacy *leak* is out of the scope of this paper, and we refer to the two terms interchangeably, meaning "exposure."

[3]Although the current AntShield prototype is built for Android, it is feasible to implement it on iOS as well since the VPN API is available for devices of versions 9.0 and above.



(a) Training phase: packets are broken into words based on various delimiters (e.g. ":", "/", "?") and are used as features in training a decision tree (an example is shown in Fig. 4).



(b) Testing phase: each packet is searched in parallel for *predefined* PIIs and for any features appearing in the trees generated in the training phase. The features found are then passed to the classifiers for predicting PII leaks.

Fig. 1. Overview of AntShield's classification.

1) Feature Extraction: Similar to prior state of the art [5], we break a packet into words based on various delimiters (*e.g.,* ":", "/", "?") and use these words as features in classification. Words from all packets in a given dataset are extracted, and those that appear too infrequently or too frequently (*e.g.,* common HTTP headers) are discarded.

2) Multilabel Classification: Unlike prior state-of-the-art, we treat PII detection as a Multi-Label problem, since a packet may contain zero, one, or multiple PII. Our classifiers decide, in one step, if any PII are contained in a packet, and if so - of what type. More specifically, we use Mulan [9] to perform multi-label classification using the Binary Relevance (BR) transformation method [8]. The idea is to train a separate binary classifier for each label. Since the C4.5 Decision Trees (DTs) worked well for classifying leak vs non-leak, we use them as the independent classifiers in BR.

3) Hybrid String Matching-and-Learning: A key insight is the distinction on whether PII of interest is known to the device or not. We refer to PII that consists of strings known a priori on the device (*e.g.,* via Android APIs, or defined by the user) as *predefined*, and the ones that are not known to AntShield as *unknown*. By default, we assume that any PII available via Android API calls are *predefined* (*e.g.,* Advertiser ID, Device ID, IMEI, mac address, email, phone number, location coordinates), and the rest are *unknown* (first/last name, gender, zipcode, city, username, and password).

An inherent advantage of operating on the device is the access to all the *predefined* strings that can be found with DPI; we refer to this method as String Matching. This not only gives us 100% accuracy on finding *predefined* leaks, if they are not obfuscated, but also reduces the set of PII that classifiers must learn, thereby improving the accuracy of finding *unknown* leaks and reducing variance (see Sec. V).

4) App vs. Domain Classifiers: Third, we build classifiers *per-app*, instead of *per-destination-domain* (as was done in prior art [5]). This is only possible because AntShield is running on the device and can accurately map a packet to the app that generated it. Our results show that per-app classifiers perform similarly to per-domain ones, but they also have

	Auto	Manual
# of Apps	414	149
# of packets	21887	25189
# of destination domains	597	379
# of leaks detected	4760	3819
# of *unknown* leaks	483	516
# of leaks in encrypted traffic	1513	1526
# of packets with **multiple leaks**	1506	790
# of leaks in TCP (other ports)	38	7
# of leaks in UDP	17	12

TABLE I
SUMMARY OF THE ANTSHIELD DATASET.

important system advantages.[4] We also show that both types of classifiers outperform a *general* classifier, which can be used for apps/domains for which not enough data was seen.

Real-Time Implementation on the Mobile Device. The classifiers described above have value in their own right. However, it is highly non-trivial to apply them in real-time on a mobile device, with limited CPU and RAM. AntShield is the *first system* to achieve this goal thanks to multiple optimizations in the Android implementation. First, in order to avoid Java string parsing needed to extract words from packets, we exploit the DTs. Specifically, we use DPI (~1ms delay per packet) to search for words that appear in the trees instead of extracting all words from each packet (30ms+ per packet). Second, we use two-phased training to minimize the feature set that needs to be loaded into memory: we first train on all features, and then train again on those features that ended up in the DTs from the first phase. Due to lack of space, we omit the details of these engineering efforts and their respective gains in performance, and defer to our technical report [6].

V. CLASSIFICATION RESULTS

Dataset. In order to evaluate the effectiveness of our methodology, we collected our own dataset, summarized in Table I. We captured all packets generated by different apps on a test device (Nexus 6) and labeled them with the PII types that they leak using AntShield's DPI capability. To cover all PII types, we manually entered *unknown* types into AntShield. We interacted with apps in two different ways. First, in order to assess PII leaks during typical user behavior, we interacted with the 100 most popular and free Android apps, based on rankings in *AppAnnie* [1], spending 5 minutes on each app. Second, we used the *UI/Application Exerciser Monkey* [2] to automatically interact with apps. This does not capture typical user behavior but enables extensive and stress testing of more apps. We had *Monkey* perform 1,000 random actions in each tested app while AntShield logged the generated traffic. We repeated this procedure for the 400 most popular apps.

Methods under Comparison. We use our dataset to compare AntShield's classification accuracy to the previous state-of-the-art Recon approach. Since AntShield combines several ideas, we evaluate each individual idea as well as the entire approach, by considering the following baselines:

1) Complete Recon approach as per Section II: classify all (*predefined* and *unknown*) leaks, using binary classifiers to detect a leak first, then use heuristics to determine the type of PII leak.[5]
2) Recon classifying *unknown* PII only.
3) String Matching on *predefined* PII, Recon trained on *unknown*; testing done on all PII.
4) Multi-Label classification trained and tested on *predefined* and *unknown* PII.
5) Multi-Label classification trained and tested only on *unknown* PII.
6) Complete AntShield approach: String Matching for *predefined* and Multi-Label classification for *unknown* leaks; Multi-Label trained on *unknown* only, tested on all PII.

[4]First, they allow for easy setup and scalability: only the few classifiers for the installed apps on the particular device must be loaded into memory. This is much smaller than hundreds of domains contacted by those apps and the third-party libraries contained within them. Second, they apply to all TCP and UDP traffic, not just to HTTP(S) traffic. Third, per-app classifiers obviate the need for DNS lookups, which are costly and inaccurate, but are necessary when using per-domain classifiers.

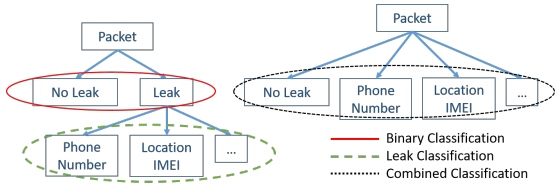


Fig. 2. Evaluation approaches: (1) Binary Classification: we assess how well we identify whether or not a packet contains a PII (Sec. V-1); (2) Leak Classification: we assess how well we infer the PII type from packets that already contain a PII, ignoring packets without PII (Sec. V-2); (3) Combined Classification - assess how well we identify the PII type *and* the No Leak label, considering all packets (Sec. V-3).

Per-app vs. Per-domain classifiers. For each method, we compare how well the per-domain, per-app, and general classifiers perform. We train specialized classifiers for those domains and apps that contain at least one positive sample (packet with a PII leaked), and one negative sample (packet with no PII leak). Results (Table II) show that per-app and per-domain classifiers perform similarly but per-app classifiers are able to cover more data than the per-domain classifiers.[6] The number of possible specialized classifiers that we can build in each case and the amount of traffic that they can cover in our dataset is as follows:

- All PII, per-app classifiers: 211 (93.3% of traffic, 99.5% of packets with PII).
- All PII, per-domain classifiers: 182 (63.6% of traffic, 95.0% of packets with PII).
- *unknown* PII, per-app classifiers: 47 (54.4% of traffic, 99.5% of packets with *unknown* PII).
- *unknown* PII, per-domain classifiers: 49 (24.5% of traffic, 87.4% of packets with *unknown* PII).

Evaluation Approaches. After classifying a packet, either a leak is detected with one or more PII types, or No Leak is detected. Depending on how one summarizes these numbers over classified packets, we may have different assessments. In particular, the majority of packets contain no leaks and if we consider them when computing F1 scores and other classification metrics, our results may look deceivingly good since non-leaky packets are easy to classify. Hence, we considered three evaluation schemes, summarized in Fig. 2. For each approach, we perform 5-fold cross-validation on the given

[5]We use the Recon code available here: https://github.com/Eyasics/recon

[6]This is expected since apps generally exhibit more diverse behavior by connecting to various domains, some of which collect PII and some of which do not. Thus, we are more likely to find apps that have sent at least one packet containing PII and one packet without PII, as opposed to domains that receive packets with and without PII.

	Method					
	(1) Recon on All PII	(2) Recon on *unknown*	(3) String Matching & Recon on *unknown*	(4) Multi-Label on All PII	(5) Multi-Label on *unknown*	(6) String Matching & Multi-Label
Per-Domain Avg	**73.8% ± 39.3**	69.5% ± 45.5	94.9% ± 20.7	99.2% ± 1.90	99.3% ± 2.88	**98.7% ± 10.6**
Per-App Avg	**74.6% ± 30.6**	69.0% ± 42.8	97.6% ± 13.0	98.8% ± 2.24	98.9% ± 3.23	**99.6% ± 3.05**
General	**55.6%**	50.3	97.3	77.4%	81.8%	**99.6%**

TABLE II
LEAK CLASSIFICATION (SEC. V-2) RESULTS: HOW WELL EACH METHODOLOGY DISTINGUISHES BETWEEN LEAK TYPES. F1 SCORE REPORTED BASED ON 5-FOLD CROSS-VALIDATION. PRECISION AND RECALL ARE DEFERRED TO [6].

model (when applicable), and calculate the average and the standard deviation across the trained specialized classifiers.

1) Binary Classification: This evaluation scheme evaluates how well the models classify a packet as containing a leak or not. In this case, both Recon's DT and our Multi-Label BR achieve similar F1 scores, over 95% on average. These results are consistent with Recon's own reports in [5], and their full details can be found in [6].

2) Leak Classification: Table II shows how well each model distinguishes PII types in packets that contain a leak, *i.e.,* packets without a PII are not taken into account. First, standard deviation is high because certain domains are easy to learn and get near perfect F1 scores, while a small set of domains are difficult (some even have a 0% F1 score). Recon's heuristic scores low when attempting to extract the PII type (column 1); see [5] for a description of the heuristic. Second, when we reduce the set of PII types to look for (column 2), the heuristic performs slightly worse, probably due to not having enough samples of *unknown* PII. Third, as expected, String Matching can find *predefined* PIIs with 100% accuracy, thus the overall F1 score improves by ~20% (column 3 vs. column 1), and the standard deviation decreases. Fourth, the Multi-Label approach shows significant improvement when compared to Recon's heuristic (column 4 vs. 1, and column 5 vs. 2); this is expected, since we do not need to estimate probabilities or calculate out thresholds as Recon does. Fifth, the complete AntShield achieves near perfect performance, and decreases the standard deviation (column 6 vs. 1-3). Finally, in most cases, the specialized classifiers outperform the general ones.

3) Combined Classification: This scheme evaluates how well each model distinguishes among PII types and "no leak," *i.e.,* packets without a PII are taken into account. In this case, all methodologies achieve F1 scores close to 90% and the difference between the performance of different classification methods is less pronounced (see [6]). This is because the majority of packets do not contain a leak, the binary classifiers work well (see Sec. V-1) and classify the "no leak" packets correctly, making the results look deceivingly good. This is why we emphasize the *Leak Classification* performance, as it provides deeper insight into the classifiers' performance.

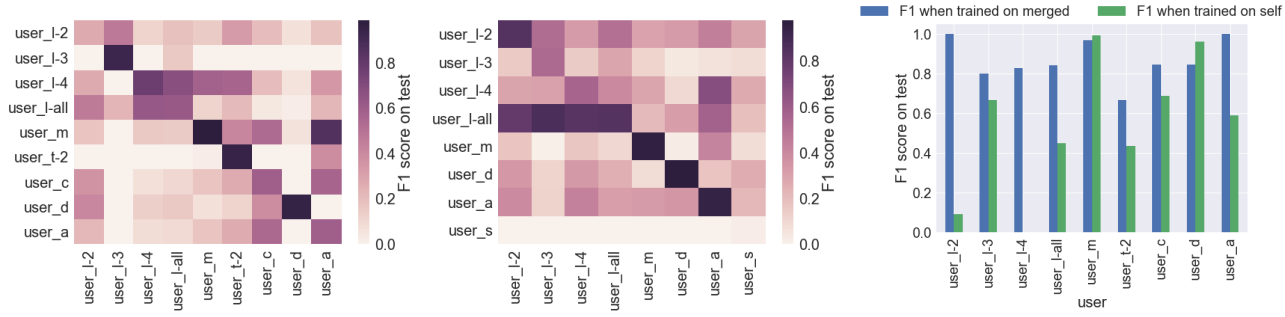
VI. COLLABORATION AMONG USERS

The classification approach we described above can detect PII leaks in any packet trace (on the device or at a remote server, online or offline), although our intended application was on the mobile device and in real-time. However, there was no distinction among different *users* or *devices*, so far, in training and testing those classifiers. An interesting question is whether sharing information (training data, classifiers, or other information) among different mobile users, directly or through a crowdsourcing entity, can help improve the classification accuracy, as opposed to each user training and testing on their own data. Preliminary results below show that the answer is *yes*, which motivates further investigation of collaboration and distributed vs. centralized learning of privacy leaks.

Dataset. To answer that question, we utilized another dataset: 10 real users (members of our group at UC Irvine) used the AntMonitor app [7] to contribute packet traces (in PCAPNG format) from their Android phones for a period of 7 months. We analyzed the packets generated only by two apps: Facebook and Chrome, which were used by all users. For some users we also have logs of the same user over a different time period or using a different phone, which we refer to as "alter-egos"; this increases the number of distinct users to 19 and 8 for Facebook and Chrome, respectively. For our experiments, we decided to include the users that have a significant number of (HTTP) packets and leaks for both Facebook and Chrome.

Q1: Can one user's classifier accurately detect leaks on another user's packets? As a first step, we train (with 10-fold cross validation) binary DTs on each user's data separately and we test on every user individually, and we report the results in Fig. 3. Each row indicates a user's classifier and each column indicates another user's data used for prediction. In Fig. 3(a), the diagonal has very high F1 scores, which is expected since the same user's logs were used for training and testing. user_l2 is an exception: her classifier is weak and cannot predict well even on the same data. The weak classifier is the result of having too few positive (leaking) examples to learn from. On the other hand, user_m's classifier can achieve $F1 > 0.8$ when predicting on user_a's data; however user_e's classifier cannot predict well user_m's leaks. We can observe a symmetry between two other users: user_c and user_a: user_c's classifier can predict well user_a's leaks and vice versa. Moreover, there are some users (user_l-3, user_d) that can be predicted well only by their own classifiers. Interestingly, the heatmap for the Chrome app (Fig. 3(b)), shows different patterns: (i) when we train on user_l-all (data from all alter-egos of user_l), we achieve good scores on all the alter egos of user_l; and (ii) user_l-4's classifier can predict quite well on user_a's data.

To understand when some users' classifiers can predict other users' leaks, we looked at the common PII types between users. For the Facebook app, we observed that user_m has a similar number of common PII types with all the users except for user_l-3. Similarly, user_d has a similar number of common PII types with user_m and user_a. Regardless, user_m's classifier cannot predict well on user_d's data, indicating that the number of common PII types is not a significant metric for predicting performance. Digging deeper, we found that the types of features that end up in the DTs themselves are a stronger indication of predictive power. In Fig. 4, we show the classifier trained on user_m's data. "?aid=" (a URL parameter



(a) F1 Heatmap for Facebook App (b) F1 Heatmap for Chrome Browser App (c) Facebook App

Fig. 3. **(a-b) Q1: sharing one user's data can help another user predict.** A classifier is trained on each user on the y-axis and is tested on each user on the x-axis, for packets belonging to Facebook **(a)** and Chrome **(b)** apps. **(c): Q2: sharing training data among all users helps.** The prediction power of a classifier trained on "merged" (80% of all users' data) is larger than a classifer trained only on "self" (that user's (80%) data collected on that user's device). In both cases, the classifier is tested on the mentioned user's 20% remaining data.

that is often associated with Advertiser ID) appears closer to the root of the tree (indicating highest information gain); and it is responsible for the predictive power of user_m's classifier on several other users in our dataset.

Q2. Can sharing among more users help? Next, we trained on 80% of all users' merged data and tested on the remaining 20% of each user data separately. We compared to the case when we train on 80% each individual user's data and test on the same remaining 20%. We report the F1 scores in Fig. 3(c), which shows significant improvements when data is shared among users. For instance, when we trained and tested (with cross-validation) on user_l-4's data, the $F1$ score was 0. However, when we trained on all users' training data, user_l-4 achieved $F1 = 0.83$. That is, if the data of other users were not used for training, this particular user could not predict any PII leaks with his/her existing training data.

Q3. Can classifiers themselves leak private information? Interestingly, we observe one particular feature " /my_gen_x_hillary_problem..." that appears since the data was collected close to the US election in the summer of 2016. This feature contains sensitive information (*e.g.,* political interests and browsing history of user_m), which is in itself a privacy leak. This motivates the need for privacy-preserving techniques for sharing training data and/or classifiers from individual mobile devices with other users or with a crowdsourcing entity.

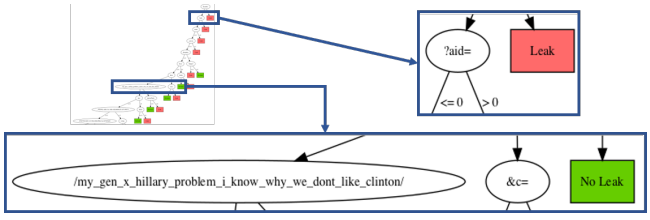


Fig. 4. Partial view of the DT for the Facebook app of user_m.

Future Directions. Our preliminary findings show that sharing information among users has the potential to train more accurate classifiers, which motivates future work on collaboration among users for learning PII leaks, either in a centralized or decentralized way [3]. However, there are a number of open questions that need to be addressed to achieve that potential. First, there are different patterns across users, apps and domains and one needs to identify which users should collaborate to share data and train classifiers. An automated approach is needed to identify similar users or cross-association of users and other features (*e.g.,* applications, domains), as demonstrated in Q1, Q2 above. Second, there is the concern of privacy: how to share information (training data, classifiers, or other information [3]) between pairs or groups of users and/or with a crowdsourcing entity in a privacy-preserving way, since sharing training data and/or classifiers compromises privacy as well, as demonstrated in Q3.

VII. Conclusion

This paper provides a classification methodology for detecting privacy leaks, which we define as transmission of personal information in network packets generated by a mobile device. This work is the first step towards truly distributed detection of PII leaks because (i) AntShield is the first such system that can run on the device itself and (ii) we demonstrate the potential and challenges of collaboration among different mobile devices. Future work will focus on (ii) identifying similar users that should collaborate to train the classifiers and (ii) distributed learning of PII leaks within the Federated Learning framework [3].

References

[1] App annie. https://www.appannie.com.
[2] Ui/application exerciser monkey. https://developer.android.com/studio/test/monkey.html.
[3] B. McMahan and D. Ramage. Federated learning: Collaborative machine learning without centralized training data. https://research.googleblog.com/2017/04/federated-learning-collaborative.html, April 2017.
[4] A. Razaghpanah, N. Vallina-Rodriguez, S. Sundaresan, C. Kreibich, P. Gill, M. Allman, and V. Paxson. Haystack: A multi-purpose mobile vantage point in user space. *arXiv:1510.01419v3*, Oct. 2016.
[5] J. Ren, A. Rao, M. Lindorfer, A. Legout, and D. Choffnes. Recon: Revealing and controlling pii leaks in mobile network traffic. In *Proc. of the 13th Annual Int. Conf. on Mobile Systems, Applications, and Services (MobiSys)*, volume 16, New York, NY, USA, 2016.
[6] A. Shuba, E. Bakopoulou, M. M. Asgari, H. Le, D. Choffnes, and A. Markopoulou. Antshield: On-device detection of personal information exposure. *arXiv preprint arXiv:1803.01261*, 2018.
[7] A. Shuba, A. Le, E. Alimpertis, M. Gjoka, and A. Markopoulou. Antmonitor: System and applications. *arXiv:1611.04268*, 2016.
[8] G. Tsoumakas and I. Katakis. Multi-label classification: An overview. *Int'l Journal of Data Warehousing and Mining*, 3(3), 2006.
[9] G. Tsoumakas, E. Spyromitros-Xioufis, J. Vilcek, and I. Vlahavas. Mulan: A java library for multi-label learning. *Journal of Machine Learning Research*, 12:2411–2414, 2011.