# Automated Privacy Network Traffic Detection via Self-labeling and Learning

Yuejun Li*†, Huajun Cui*†, Jiyan Sun*, Yan Zhang*†, Yueqi Li*†, Guozhu Meng*, Weiping Wang*
*Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
†School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
‡Corresponding Author: cuihuajun@iie.ac.cn

***Abstract*—With the increasing popularity of mobile devices, privacy leakage has become more and more serious. The inappropriate behaviors of mobile APPs have brought substantial security risks to the public (e.g., location leakage). Existing solutions detect privacy leakage based on network traffic analysis. However, they can only detect unencrypted traffic, which leads to failures in the face of encrypted traffic. To solve this challenge, we designed an Automated Privacy Traffic Detection system (APTD). APTD can automatically generate self-labeling privacy traffic datasets, learn to identify the encrypted privacy traffic, and accurately assess the risk of privacy leakage. Due to its automation capability, APTD can directly support privacy leakage detection for newly-emerged applications without any system changes. To comprehensively evaluate APTD, we conducted an experiment on 2327 real-world mobile APPs. APTD automatically generated a labeled dataset containing 27343 real-world encrypted traffic traces. Based on the dataset, APTD identifies privacy traffic, and performs a privacy leakage risk assessment of APPs. The results show that APTD achieves 97% accuracy and 99% recall on our dataset and identifies 12 APPs that transmit high-risk privacy data.**

***Index Terms*—Privacy traffic, Encrypted traffic detection, Privacy leakage**

## I. Introduction

Smartphones have become critical auxiliary tools in people's daily life. Meanwhile, the problem of user privacy leakage is becoming more and more serious. The security report of Qihoo [1] shows that privacy stealing accounts for 66.2% among malicious APPs. Compared with PCs, smartphones store many user privacy data, such as passwords, short messages, address books, location information, etc. Leakage of such information may affect users' personal and property safety (e.g., bank account passwords leakage leads to property loss, location information leakage leads to being tracked).

There are two ways to detect privacy information leakage from Android APPs. The first one is the APP code detection methods [2–4] . And the other is the privacy network traffic detection [5].

The former methods mainly refer to analyzing the APP source code to determine whether it may transmit privacy data. Therefore, there is a severe limitation that they can only confirm that malicious code is executed, but they can not confirm whether privacy information has been transmitted to the remote server. However, existing privacy traffic detection methods can only analyze unencrypted traffic. As a result, existing methods based on unencrypted traffic are no longer effective, as more and more developers tend to transmit data by HTTPS requests.

With the advances of machine learning technologies, recent studies have shown the great potential of applying machine learning for encrypted traffic identification[6–12]. However, they focus on general application classification problems (e.g., identifying the type of an app is video or game) rather than identifying privacy traffic. There are some problems to be addressed before we can apply existing work to privacy traffic detection. First, the performance of machine learning methods usually relies on high-quality datasets, which are currently lacking for the user privacy detection problem and also difficult to obtain due to the complex behavior of real-world applications. Second, existing machine learning solutions can only work well for the APPs included in the training datasets. They are difficult to do self-adaption when facing new APPs.

Therefore, considering the popularity and the seriousness of privacy leakage in the Android system, we focus on the privacy information transmitted by Android APPs as follows:

- Device identification information, such as ICCID, IMEI, IMSI, MAC address, Android ID, etc.
- User identification information, such as name, date of birth, gender, mailing address, relationship status, etc.
- Location information, such as latitude and longitude, postal code, etc.
- APP privacy information, such as photos, chat messages, call records, contacts, audio files, etc.
- User login credential information, such as account number, password, etc.
- Network information, such as IP address, gateway address, etc.

To address the above challenges, we propose an Automated Privacy Detection system (APTD). APTD first enables the automated generation and self-labeling of high-quality privacy traffic datasets. Then it learns from the data to identify the encrypted privacy traffic and performs the privacy risk assessment for APPs. The contributions of our work are summarized as follows.

- First, we design a novel method to imitate user-application interactions to continuously trigger traffic and capture comprehensive and accurate traffic. In the experiment, we find that existing work of traffic triggering

978-1-6654-9792-3/22/$31.00 ©2022 IEEE

based on random click simulation is inaccurate and inefficient due to the randomly-changing invalid application behaviors, e.g., an accidental touch on the WiFi switch. In comparison, our method steadily imitates user-application interaction and triggers application traffic faithfully in a depth-first-search way, which increases the diversity of valid traffic. The evaluation results show that the traffic richness collected by our method is 1.85 times higher than existing solutions on average.

- Second, we develop an efficient real-time framework to generate self-labeled high-quality encrypted privacy traffic datasets automatically. Specifically, we first propose methods to capture both the encrypted traffic and unencrypted traffic at the same time and then perform the exact association between them. Rather than using proxy as existing methods apply, our method eliminates the use of proxy and thus avoids the risk of being detected and bypassed by applications. Next, the captured encrypted traffic traces are self-labeled by the corresponding associated unencrypted traffic in an automated way.
- Third, we propose efficient machine learning methods for encrypted privacy traffic identification and accurate assessment methods for the evaluation of privacy leakage risk levels. For the machine learning module, we derive a set of key statistic traffic features that works most efficiently for privacy traffic identification. Next, we design a self-defined sample balancing method to optimize the random-forest algorithm for attaining higher accuracy. Finally, we propose a novel privacy risk metric that jointly quantifies the risk level of the APP categories, APP permissions, and the actual privacy information transmitted, providing accurate risk evaluation for different APPs.

All the above processes of our APTD system are completely automated without any human interventions and can run in real-time (less than 8 minutes for each application) for any new mobile application. Among 2327 most-popular real-world APPs, our APTD system finds 12 APPs that transmit high-risk privacy, 54 APPs that transmit medium-risk privacy, and 157 APPs that transmit low-risk privacy, and all these results are further confirmed by the manual analysis. The evaluation results show that our approach achieves 97% accuracy, 97% precision, 99% recall, and 98% F1-score, which validates its high performance for identifying real-world privacy traffic.

## II. Related work

As mentioned before, the existing network-side privacy traffic detection methods[5] can not identify encrypted privacy traffic. And the machine-learning-based methods in encrypted privacy traffic identification[6–12] are just concerned about application classification and are difficult to generalize. And we can not apply them in identifying encrypted privacy traffic directly due to the lack of high-quality datasets. To build a high-quality dataset, we need to imitate user-application interactions to trigger traffic, and then capture encrypted traffic and unencrypted traffic at the same time. By privacy keyword matching in visible unencrypted traffic, we can label the corresponding encrypted traffic.

### A. Traffic identification

**Unencrypted privacy traffic identification.** The basic method of unencrypted privacy traffic identification performs privacy information keyword matching on the plaintext in the traffic. For scenarios where plaintext text or privacy information keywords are difficult to identify, the ReCon system proposed by Ren et al.[5] uses the Weka data mining tool [13] to train a C4.5 decision tree classifier for predicting privacy leaks, and uses a bag-of-words model to sub-phrase unencrypted traffic, then filter low-frequency words and known common non-privacy information fields to determine possible privacy information leakage fields. And keyword matching can not be performed on encrypted traffic due to the invisible loads.

**Encrypted traffic identification.** Maciej et al.[6] exploited the sequence of message types of a given application in the SSL/TLS header to build a first-order homogeneous Markov chain as an application-specific statistical fingerprint. Further, Meng et al.[7] proposed a second-order Markov chain based on the length of certificate packets in SSL/TLS sessions for encrypted traffic classification. The results demonstrate an average improvement of 30% in classification accuracy compared to the state-of-the-art methods. Zhong et al.[8] proposed a novel traffic classification model based on Gaussian mixture model and hidden Markov model named MGHMM, and experimental results on protocol-level and application-level traffic classification showed that the scheme has good performance. Cong et al.[9] proposed a comprehensive effective traffic information analysis (CETAnalytics) framework and a substructure network, Attract, to achieve payload content analysis by matching traffic structures. Giuseppe et al.[10] proposed a multimodal multitasking deep learning approach (Distiller classifier) to perform traffic classification. The results show that their approach is better than both multi-task extensions of single-task baselines and local multi-task architectures. Thijs et al.[11] proposed FLOWPRINT, a semi-supervised approach for encrypted traffic identification. The application to which the encrypted traffic belongs is identified by the temporal correlation between network traffic destinations and can reach an accuracy of 89.2%. Similarly, Tal et al.[12] transforms basic flow data into intuitive pictures and then uses CNN to identify traffic categories (browsing, chat, video, etc.) and applications in use. These approaches just focus on the application classification, and we can not apply them in encrypted privacy traffic directly.

### B. Traffic triggering and capturing

**Traffic triggering.** The most popular tool to imitate user-application interactions and trigger traffic is Monkey[14]. However, the Monkey's operation is completely random and it is difficult to guarantee that all interactable layouts are triggered, e.g., all clicks do not occur on clickable layouts,

or click occurs on network component results in closing the network connection and no traffic is captured.

**Traffic capturing.** As for traffic capturing, there are two kinds of tools to capture traffic. The first one is Network Interface Card(NIC) tools that capture traffic from NIC, such as TCPDUMP and Wireshark[15]. This approach is simple and easy to apply. However, it can not look insight into the encrypted traffic. The second approach is based on the man-in-the-middle technique. The middleman acts as the server to the client and the client to the server. So the middleman can capture requests sent by both ends and can look insight into the encrypted traffic. Typical man-in-the-middle traffic capture tools are Charles and Fiddler[16].

However, this approach can not capture comprehensive and accurate traffic. Because there are some ways of detecting proxies that can be used to change the communication behavior of the APP. For example, some APPs will use System.getProperty("http.proxyHost") or System.getProperty("http.proxyPort") these two APIs to check whether the current system has set up an HTTP proxy, and once a proxy is found, measures can be taken to disguise their behavior or change their communication behavior, resulting in this method being less comprehensive and accurate in the traffic collection process.

To build a high-quality dataset, we need to capture encrypted traffic and unencrypted traffic at the same time. Therefore, we propose a new method to capture unencrypted traffic and encrypted traffic simultaneously.

## III. System Design

### A. System overview

As shown in Figure 1, APTD can automatically generate self-labeling privacy traffic datasets, adaptively learn to identify the encrypted traffic of user privacy data, and accurately assess the risk of user privacy leakage. It proceeds in five phases:

- *Starting up*. Crawl APPs from the APP store and perform static analysis on APK files. Install and launch the app, by matching the layout keywords and attributions to initialize the APPs, including click pop-up windows, etc. Then, use our proposed AutoClick to trigger APP traffic.
- *Traffic capturing*. Use our proposed AutoCapture to capture encrypted traffic and use TCPDUMP to unencrypted traffic at the same time.
- *Datasets generation*. Parse pcap files captured by *Traffic capturing* to traffic traces, associate encrypted traffic traces, and unencrypted traffic traces. Then, label unencrypted traffic traces by privacy keywords matching, and tag the corresponding encrypted traffic traces to build the dataset.
- *Encrypted privacy traffic identification*. Based on the dataset created by *Datasets generation*, extract statistical features, and use the random forest to identify encrypted privacy traffic trace.
- *Risk of encrypted privacy traffic assessment*. As for the encrypted privacy traffic trace detected by *Encrypted privacy traffic identification*, calculate the risk metric to assess the risk of encrypted privacy traffic trace.

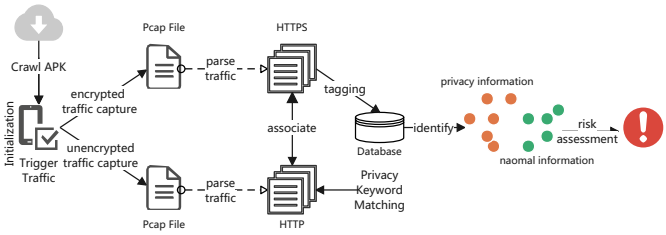



Fig. 1: System overview.

### B. Starting up

**Crawler:** Execute crawler scripts to crawl popular APK files in 360 APP Market, perform static analysis on APK files by AAPT tool, and write APK file location, package name, etc. into the database.

**APP initialization and traffic triggering:** Before capturing traffic, we need to initialize the APP, such as authorize the permissions required by the APP, agree to the privacy policy, slide the pop-up window, etc., and trigger APP traffic. As mentioned before, Monkey is inaccurate and inefficient due to the randomly-changing invalid application behaviors, it is necessary to propose an automated APP initialization and traffic triggering method. We need to address two challenges: 1) imitate users to authorize the permissions, agree to the privacy policy, and turn off ads. 2) trigger as much as possible interactive layouts of the APP to create sufficiently comprehensive and abundant traffic.

Through our research, we found that Android has an important component - Activity, which is responsible for managing the user interface(UI)) of the application. An application may contain multiple Activities, and each interface is an Activity, which means that the interface switching is actually a transition between different Activities. And, Android implements UI content and layout (collection of components) through View associated with Activity. The view is the base class of Android UI components, and ViewGroup also inherits from the View class. ViewGroup and View are in the parent-child relationship, and ViewGroup contains some View or ViewGroup, which makes the UI structure like a tree, that is, the view tree.

Therefore, we propose an Initialization program to initialize the APPs. It can install and activate the APPs, click pop-up windows, agree to the privacy policy, turn off ads, etc. automatically by matching the layout keywords and attributions. Similarly, we proposed AutoClick to traverse the APP layout view tree[17] hierarchically and click all clickable layouts to trigger traffic. The process is shown in the Algorithm 1. We save each new layout of the running app. And find all clickable layouts and click them. We will stop the AutoClick program when the depth of the traversing tree is up to 6.

### C. Traffic capturing

In this section, we introduce how to capture encrypted traffic and unencrypted traffic simultaneously. As mentioned in

**Algorithm 1** Hierarchical Traversal View Tree

**Input:** APP Layout $Current\_layout$
**Output:** End Of Traversal Flag $depth = 6$

```
depth = 0
Layout_list = []
while depth ≤ 5 do
  if Current_layout not in Layout_list then
    Layout_list.append(Current_layout)
    depth = depth + 1
    Click_component = []
    for componentinCurrent_layout do
      if component is clickable then
        Click_component.append(component)
        while Click_component ≠ [] do
          click the Click_component[−1]
          Click_component.pop()
          read current layout
          Current_layout = layout
        end while
      else
        continue
      end if
    end for
  else
    back to previous layout
  end if
end while
```

Section II-B, we use TCPDUMP to capture encrypted traffic and need to propose a new method to capture unencrypted traffic. Because the MITM tools such as Fiddler may be detected and bypassed by APPs so that it can not capture comprehensive and accurate traffic. We need to address the challenge avoid the risk of being detected and bypassed by APPs and capture comprehensive and accurate traffic.

We found that the dynamic instrumentation[14, 15] technique can hook our code to any APP native functions, including the interface for reading and sending TLS data in the default TLS protocol library of Android, which is SSL_read and SSL_write. As shown in Figure 2, by hooking these two APIs, the data can be acquired before the plaintext traffic is encrypted and sent, and after the ciphertext, traffic is unencrypted and read.

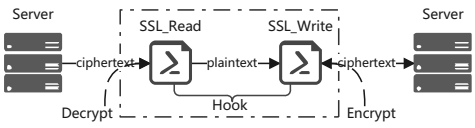



Fig. 2: Hook function diagram

Therefore, we propose AutoCapture to capture unencrypted traffic. AutoCapture hooks the traffic collection code into SSL_read and SSL_write in the Android native layer. The advantage is that the unencrypted traffic can be collected comprehensively and accurately, and it also avoids the risk of being detected and bypassed by APPs. In addition, to capture encrypted traffic and unencrypted traffic at the same time, we control two programs by reading changes of key fields in the database.

### *D. Datasets generation*

**Traffic parsing:** We parse pcap files from the network layer, getting the source address, source port, and other information, and stream reorganization, restore the TCP stream, according to the HTTP 1.1 protocol-RFC2616 standard, reorganize and parse the HTTP(S) information carried by TCP, output the HTTP(S) request and store it in the database.

**Associate encrypted traffic with unencrypted traffic:** After parsing, we need to correlate the HTTP requests and HTTPS requests because the AutoCapture collects the runtime traffic of a specific APP and TCPDUMP collects all the traffic passing through the NIC. By comparing the same five tuples (source address, destination address, source port, destination port, and timestamp), we associate the corresponding HTTP requests and HTTPS requests to exclude the irrelevant noise traffic and improve the dataset's accuracy and availability.

**Tagging datasets:** Based on the user privacy information transmitted by Android APPs mentioned in Section I, we apply keyword matching on the content of HTTP requests, such as IMEI, WiFi, etc... Then we label HTTP requests and the corresponding HTTPS requests if they contain privacy information. Finally, we build an encrypted privacy traffic dataset.

### *E. Encrypted privacy traffic identification*

**Feature extraction:** We use the CICflowmeter[20, 21] to extract encrypted traffic statistics features, such as flow duration, bytes of the first packet transmitted in the forward flow, etc. CICflowmeter distinguishes the statistical features from forwarding and backward directions and also constructs a flag for each flow consisting of source address, destination address, and protocol number. The features of each flow are one-dimensional arrays in the form of $80 \times 1$. Then we preprocess the feature dataset.

- Remove the irrelevant contents from the feature dataset, such as timestamp, flow_id, etc.
- Convert the strings into integers, such as source address, destination address, etc.

**Encrypted privacy traffic identification:** We use the random forest[22] algorithm to identify encrypted privacy traffic, like Figure 3.

For training, we use GridSearchCV for hyperparameter optimization:

- Adjust the number of iterations to test and print the optimal parameters.
- Adjust the maximum depth and the minimum number of samples required for internal node repartitioning.
- Since the minimum number of samples required for internal node repartitioning is related to the minimum number of samples in the leaf nodes, we associate the two parameters and continue the optimization.

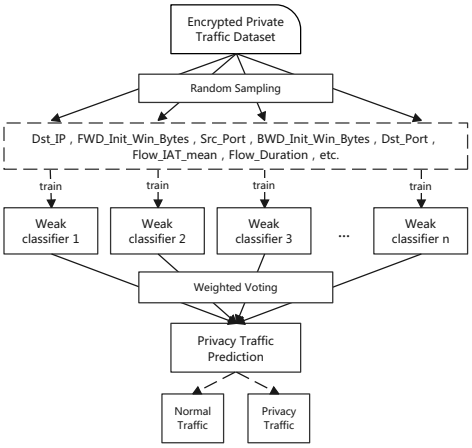



Fig. 3: Random Forest overview.

- Modify the corresponding parameters and optimize the maximum number of features.

Before inputting the random forest, the 30% feature dataset is randomly selected as the test set. The final identification achieves an accuracy of 94%.

Considering the imbalance of the privacy samples and normal samples, we balanced the feature dataset - copying a set of privacy samples into the dataset. Then repeat the hyperparameter optimization steps and retrain the model. Finally, our random forest reaches 97% accuracy in the identification of encrypted privacy traffic. The main parameters and values of the random forest are shown in Table I.

TABLE I: Parameters of RF

| Parameters | Values | Descriptions |
|---|---|---|
| n_estimators | 70 | Number of decision trees |
| criterion | "gini" | Evaluation criteria for features |
| max_features | "log2" | Maximum number of features used in a single decision tree |
| max_depth | "None" | Maximum depth of decision tree |
| min_samples_split | 2 | Minimum number of samples required to split internal nodes |
| min_samples_leaf | 1 | Minimum number of samples that should be available on a leaf node |

### *F. Risk of encrypted privacy traffic assessment*

To provide clear risk warnings for traffic monitors, we propose an encrypted privacy traffic transmission risk metric. Firstly, considering the possible impact of privacy information leakage, we classify the basic level of privacy information into four categories, as detailed in Table II. For example, an encrypted traffic trace contains device information that belongs to low risk; requesting location permission belongs to high risk.

Secondly, we jointly quantify the risk level of the APP categories, APP permissions, and the actual encrypted privacy traffic. It helps to strike a balance between the privacy leakage risk level and the alarm priority. Because it is normal for a mapping APP to request permissions and transmit traffic for location information, but is not to a drawing APP.

Finally, We propose the metric. We use $C_i$ to indicate the privacy risk level to which the APP category belongs, $R_i$ to indicate the privacy risk level included in the APP permission, and $T_i$ to indicate the risk level of the information transmitted by real traffic. Thus we have the actual APP category risk level $A_i$ and risk of encrypted privacy traffic $P_i$ :

TABLE II: Risk level definitions

| Risk Level | Information transimited by traffic | APP category | APP permission |
|---|---|---|---|
| High risk | Location information, User identification information, User login credential information | communication, business, shopping, travel, finance, health | CALL_PHONE, READ_CONTACTS, ACCESS_COARSE_LOCATION, ACCESS_FINE_LOCATION, ... |
| Medium Risk | Network information, APP privacy information | media, news, life, themes, photography, education | CAMERA, INTERNET, ACCESS_NETWORK_STATE, ACCESS_WIFI_STATE, ... |
| Low Risk | Device identification information | system, tools | MOUNT_UNMOUNT_FILESYSTEMS, WRITE_EXTERNAL_STORAGE, WRITE_SETTINGS, ... |
| No Risk | No privacy information | None | MODIFY_AUDIO_SETTINGS, SET_ALARM, ... |

$$A_i = \mathbb{I}(C_i = R_i) \cdot 1 + \mathbb{I}(C_i \neq R_i) \cdot [\mathbb{I}(C_i > R_i) \cdot C_i + \mathbb{I}(C_i < R_i) \cdot (R_i + 2)] \quad (1)$$

$$P_i = \mathbb{I}(T_i = 0) \cdot 0 + \mathbb{I}(T_i \neq 0) \cdot [\mathbb{I}(T_i = A_i) \cdot 1 + \mathbb{I}(T_i \neq A_i) \cdot T_i] \quad (2)$$

$\mathbb{I}$ indicates the indicator function, which is 1 if the statement is true and 0 otherwise. $i \in [0, 4]$ corresponds to four levels of risk. As a result, $P_i$ is the real risk level of an encrypted privacy traffic trace. For example, as shown in Figure 4, an encrypted traffic trace was transmitted by $com.adroid.kuaisuqidong$, this tool APP belongs to low risk($C_i = 1$), but its permissions include location request which belongs to high risk($R_i = 3$), so real risk level of the APP $A_i = 5$. The result indicates that it deserves to be noticed because it has requested excessive permissions. In addition, the traffic trace contains location information($T_i = 3$), and we can know that $P_i = 3$ by calculating the risk metric, which means this traffic trace belongs to high risk.

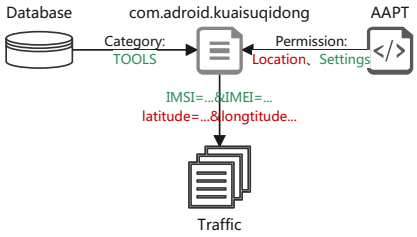



Fig. 4: Assess the risk of an encrypted privacy traffic.

Based on the metric, among 2879 encrypted privacy traffic traces, 88 traffic trace contains high risk privacy information transmitted by 12 APPs, 502 traffic trace contains medium risk privacy information transmitted by 54 APPs, and 2289 traffic trace contains low risk privacy information transmitted by 157 APPs.

## IV. Evaluation

### *A. Implementation*

We use BeautifulSoup[23], a popular library for HTML crawling, to crawl APPs from the 360 APP store[24]. We crawled a total of 2327 popular APPs. APP initialization and our AutoClick are based on the uiautomator2[25], a library

that can get the properties of any control of any APP on the screen and perform any operation on it. AutoCapture is based on the dynamic instrumentation technique. In the experiment, each APP runs for a total of 7 minutes, the APP initialization is completed in the first 2 minutes, and the traffic collection is completed in the remaining time. 962MB unencrypted traffic was captured by the AutoCapture, and 6.3GB encrypted traffic was captured by the TCPDUMP. We make use of pypacpkit[26] a stream pcap file extractor, to parse traffic to HTTP(S) requests. And We associate the HTTP and HTTPS requests, use keyword matching to tagging datasets. The dataset contains 27343 traffic traces, of which 2879 traffic traces contain privacy information. Then, we use CICFlowmeter to extract features and random forest to identify encrypted privacy traffic. Finally, we propose a metric to assess the risk of encrypted privacy traffic. All the methods are implemented in Python 3.9.

### B. Effectiveness evaluation

**Effectiveness of AutoClick.** We conducted a comparison experiment between the AutoClick and the widely adopted testing tool Monkey.

1) Select 20 APPs randomly and install them on two identically configured Google Pixel3 phones.
2) Both of them run AutoCapture to capture traffic.
3) Launch the same APP, use Monkey to create 1000 random events on one of the phones, while the other phone runs the AutoClick program.
4) Test each APP 3 times for 5 minutes each time.
5) Divide both traffic into different categories according to the method, protocol, path of access URL, host, keywords, and values.
6) Compare the number of different categories of traffic contains.

The results show that among the 20 APPs tested, the traffic richness created by the AutoClick in 5 minutes is significantly higher than that created by Monkey, and the traffic richness collected AutoClick is 1.85 times higher than that of Monkey on average. In addition, Monkey failed to collect traffic due to touching the WiFi component switch several times during the experiment, while the APPs using the AutoClick worked normally.

**Effectiveness of AutoCapture.** Similarly, we conducted a comparison experiment between the AutoCapture and the widely adopted man-in-the-middle tool Fiddler everywhere.

1) Select 20 APPs randomly, and install them on two identically configured Google Pixel3 phones.
2) Set the Fiddler proxy on the WiFi of one phone, while the WiFi of the other phone is left untouched.
3) Launch the same APP, while one is captured by the Fiddler, the other is captured by the AutoCapture program.
4) Observe whether the behavior of the APP has changed.

The results show that 11 APPs performed abnormally on the phone with Fiddler proxy, such as Talking Tom2, in the case of setting Fiddler agent, the pop-up horizontal ads could not be seen when opening the APP, and the advertising traffic could not be captured, while the APPs on the phone using AutoCapture to collect traffic were all running normally.

### C. Efficiency evaluation

As mentioned before, each APP runs for 7 minutes, the APP initialization is completed in the first 2 minutes, and the traffic collection is completed in the remaining time. Then, it costs 45 minutes on the traffic parsing for 27343 traffic traces, on average, each traffic trace cost 0.09875s. Association and dataset tagging cost 18 minutes, each traffic trace costs 0.03950s on average. And training and identification cost 15s totally. Finally, it costs less 1 second to assess the risk level of encrypted privacy traffic. As a result, the APTD is a lightweight system, it can identify an encrypted privacy traffic trace and assess the risk of it quickly.

### D. Important features

We evaluate the importance of the features in the input feature dataset by determining the contribution of each feature to each decision tree in the random forest, the average is taken and finally, the level of contribution of each feature to the classification is compared, the top 34 contributions of features are shown in the Figure 5. And the descriptions of top 10 features are in Table III.

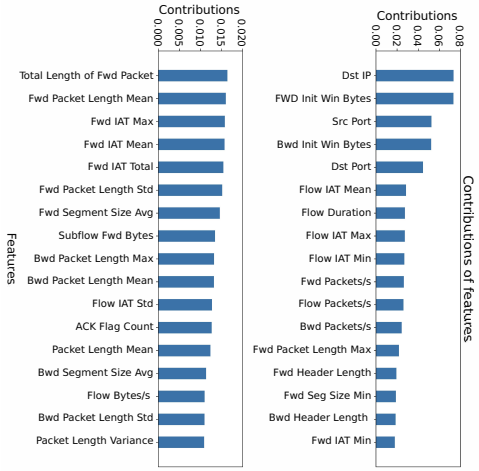



Fig. 5: Contributions of features.

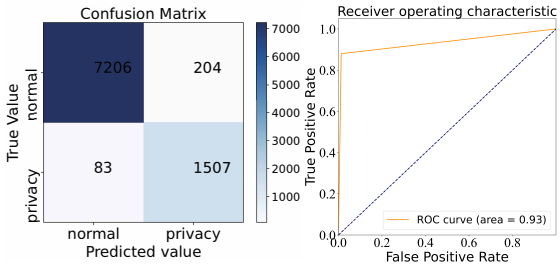
TABLE III: Descriptions of features.

| Features | Descriptions |
|---|---|
| Dst IP | Destination address |
| FWD Init Win Bytes | Bytes of the first packet transmitted in the forward flow |
| Src Port | Source address |
| Bwd Init Win Bytes | Bytes of the first packet transmitted in the backward flow |
| Dst Port | Destination Port |
| Flow IAT Mean | Average of the interval between two flows |
| Flow Duration | Flow duration |
| Flow IAT Max | Maximum of the interval between two flows |
| Flow IAT Min | Minimum of the interval between two flows |
| Fwd Packets/s | Packets transmitted forward per second |

### E. Identification Analysis

We evaluate the effectiveness of the model in four dimensions, accuracy, precision, recall and F-score. Therefore, $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}, Precision = \frac{TP}{TP+FP}, F1 -$

$score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$, $Recall = \frac{TP}{TP+FN}$. The confusion matrix and ROC curve of the random forest algorithm are shown in Figure 6. As can be seen, the accuracy of using the random forest algorithm to identify encrypted privacy traffic is 0.9681, the precision is 0.9725, the recall is 0.9886, and the F1-score is 0.9805. To more clearly illustrate the superiority of the method proposed in this paper, we selected two models to reproduce, namely Fs-net and ETC-CNN.




(a) Matrix of RF (b) ROC of RF

Fig. 6: Matrix and ROC

**Fs-net:** Fs-net is a flow-sequence encrypted traffic classification model proposed by Liu et al.[27]. It uses the bidirectional LSTM algorithm. In the original paper, the model uses the traffic sequence as the only original traffic information to classify encrypted traffic into specific APPs, 99.14% TPR, 0.05% FPR, and 99.06% FTF can be achieved.

The process of reproducing Fs-net is as follows:

- According to the feature format of the Fs-net, take the length of the first 512 data packets for the pcap files and fill in the lengths with 0 if it is not enough.
- Integer all processed txt datasets into a txt.
- Divide the txt into a test set and a training set.
- Modify the model to 2 classifications, train and test the Fs-net.

The results show that the accuracy of the Fs-net classification is 0.90.

**ETC-CNN:** ETC-CNN is a popular open-source model for CNN-based encrypted traffic classification on Github. This CNN contains a total of six convolutional layers, two pooling layers, and one fully connected layer.

The process of reproducing Fs-net is as follows:

- Convert the pcap file into JSON format and label the JSON file.
- Extract the flow statistical features from JSON file.
- Preprocess the features and divide them into train set and test set.
- Modify the model to 2 classifications, train and test the ETC-CNN.

The results show that the accuracy of the model's classification is 0.89.

The comprehensive comparison results of the three models are shown in Figure 7 In summary, the classification results of applying our encrypted privacy traffic dataset in the two models replicated were less accurate than the random forest model. Because the features used in the Fs-net model only include packet length, and the statistical features used in the ETC-CNN model do not include important features such as destination address and forward/backward first packet length, which contributes significantly to the classification. The traffic features used in these two models are not very suitable for the identification of encrypted privacy traffic. Therefore, the accuracy of encrypted privacy traffic identification is much lower than that of the random forest.

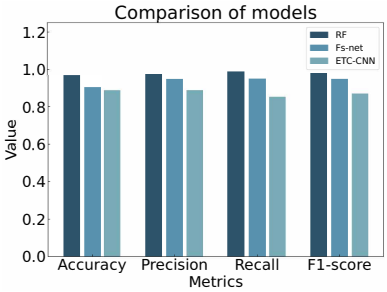



Fig. 7: Comparison and evaluation.

## V. Conclusion

The problem of user privacy leakage caused by Android mobile devices is becoming more and more serious, and the existing network-side solutions to Android privacy leakage have a severe limitation that they can not identify encrypted privacy traffic. Therefore, we propose APTD, a system that can automatically generate self-labeling privacy traffic datasets, adaptively learn to identify the encrypted traffic of user privacy data, and accurately assess the risk of user privacy leakage. In addition, APTD can directly support privacy leakage detection for newly-emerging unknown applications without any system changes. For a comprehensive evaluation, we test the APTD system performance on 27343 real traffic traces generated from the most popular 2327 real-world mobile applications. The evaluation results show that APTD achieves 97% accuracy and 99% recall for the tested traffic traces and identifies 0.52% popular applications transmit high risk privacy information, which validates its high effectiveness on detecting the user privacy leakage in real-world traffic. To the best of our knowledge, this is the first work that manages to address the problem of encrypted privacy traffic detection for mobile applications. We will optimize our system in future work to achieve better recognition results.

## VI. Acknowledgement

This work is supported by the National Key Research and Development Program of China (No.2019YFB1005201).

## References

[1] 360 Internet Security Center. "China's cell phone security status report in the first half of 2019" [EB/OL]. 2019. https://zt.360.cn/1101061855.php?dtid=1101061451&did=210935462.

[2] Arzt, Steven, et al. "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps." Acm Sigplan Notices 49.6 (2014): 259-269.

[3] Li, Li, et al. "Iccta: Detecting inter-component privacy leaks in android apps." 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. Vol. 1. IEEE, 2015.

[4] Octeau, Damien, et al. "Effective Inter-Component Communication Mapping in Android: An Essential Step Towards Holistic Security Analysis." 22nd USENIX Security Symposium (USENIX Security 13). 2013.

[5] Ren, Jingjing, et al. "Recon: Revealing and controlling pii leaks in mobile network traffic." Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services. 2016.

[6] Korczyński, Maciej, and Andrzej Duda. "Markov chain fingerprinting to classify encrypted traffic." IEEE INFOCOM 2014-IEEE Conference on Computer Communications. IEEE, 2014.

[7] Shen, Meng, et al. "Certificate-aware encrypted traffic classification using second-order markov chain." 2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS). IEEE, 2016.

[8] Yao, Zhongjiang, et al. "Encrypted traffic classification based on Gaussian mixture models and Hidden Markov Models." Journal of Network and Computer Applications 166 (2020): 102711.

[9] Dong, Cong, et al. "CETAnalytics: Comprehensive effective traffic information analytics for encrypted traffic classification." Computer Networks 176 (2020): 107258.

[10] Aceto, Giuseppe, et al. "DISTILLER: Encrypted traffic classification via multimodal multitask deep learning." Journal of Network and Computer Applications 183 (2021): 102985.

[11] van Ede, Thijs, et al. "Flowprint: Semi-supervised mobile-app fingerprinting on encrypted network traffic." Network and Distributed System Security Symposium (NDSS). Vol. 27. 2020.

[12] Shapira, Tal, and Yuval Shavitt. "FlowPic: A generic representation for encrypted traffic classification and applications identification." IEEE Transactions on Network and Service Management 18.2 (2021): 1218-1232.

[13] Hall, Mark, et al. "The WEKA data mining software: an update." ACM SIGKDD explorations newsletter 11.1 (2009): 10-18.

[14] Paydar, Samad. "An Empirical Study on the Effectiveness of Monkey Testing for Android Applications." Iranian Journal of Science and Technology, Transactions of Electrical Engineering 44.2 (2020): 1013-1029.

[15] Goyal, Piyush, and Anurag Goyal. "Comparative study of two most popular packet sniffing tools-Tcpdump and Wireshark." 2017 9th International Conference on Computational Intelligence and Communication Networks (CICN). IEEE, 2017.

[16] Crane, Jocelyn. Fiddler crabs of the world: Ocypodidae: genus Uca. Vol. 1276. Princeton University Press, 2015.

[17] Wang, Peng, et al. "Automatic Android GUI traversal with high coverage." 2014 Fourth International Conference on Communication Systems and Network Technologies. IEEE, 2014.

[18] Luk, Chi-Keung, et al. "Pin: building customized program analysis tools with dynamic instrumentation." Acm sigplan notices 40.6 (2005): 190-200.

[19] Maebe, Jonas, Michiel Ronsse, and Koen De Bosschere. "DIOTA: Dynamic instrumentation, optimization and transformation of applications." Compendium of Workshops and Tutorials held in conjunction with PACT'02. 2002.

[20] Canadian institute for cybersecurity (cic). CICFlowMeter (2017).

[21] Sharafaldin, Iman, Arash Habibi Lashkari, and Ali A. Ghorbani. "Toward generating a new intrusion detection dataset and intrusion traffic characterization." ICISSp 1 (2018): 108-116.

[22] Belgiu, Mariana, and Lucian Drăguţ. "Random forest in remote sensing: A review of applications and future directions." ISPRS journal of photogrammetry and remote sensing 114 (2016): 24-31.

[23] BeautifulSoup, "a Python library for extracting html files," https://www.crummy.com/software/BeautifulSoup/bs4/doc/, March 2022.

[24] 360 APP store, "360 APP store," https://ext.se.360.cn/webstore/category/, March 2022.

[25] uiautomator2, "a Python library to do Android automation testing," https://github.com/openatx/uiautomator2#, March 2022.

[26] Sergeevich, Kollerov Andrey, and Fartushnyy Andrey Vladimirovich. "Analysis of Node Interaction Using the TLS Protocol by Means of Machine Learning Tools." 2020 Ural Symposium on Biomedical Engineering, Radioelectronics and Information Technology (USBEREIT). IEEE, 2020.

[27] Liu, Chang, et al. "Fs-net: A flow sequence network for encrypted traffic classification." IEEE INFOCOM 2019-IEEE Conference On Computer Communications. IEEE, 2019.