IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 25, 2018, accepted December 10, 2018, date of publication December 18, 2018,
date of current version February 22, 2019.

Digital Object Identifier 10.1109/ACCESS.2018.2888568

Detecting Android Locker-Ransomware on Chinese Social Networks

DAN SU[1], **JIQIANG LIU**[1], **XIAOYANG WANG**[2], **AND WEI WANG**[1]
[1]Beijing Key Laboratory of Security and Privacy in Intelligent Transportation, Beijing Jiaotong University, Beijing 100044, China
[2]Beijing Key Laboratory of Traffic Data Analysis and Mining, Beijing Jiaotong University, Beijing 100044, China

Corresponding author: Wei Wang (wangwei1@bjtu.edu.cn)

This work was supported in part by the National Key R&D Program of China, under Grant 2017YFB0802805, in part by the Fundamental Research Funds for the Central Universities of China, under Grant 2018JBZ103, and in part by the Natural Science Foundation of China, under Grant U1736114 and Grant 61672092.

ABSTRACT In recent years, an increasing amount of locker-ransomware has been posing a great threat to the Android platform as well as users' properties. Locker-ransomware blackmails victims for ransom by compulsorily locking the devices. What is worse, a mature locker-ransomware transaction chain has taken shape on Chinese social networks. The effective detection of locker-ransomware is an emergent yet crucial issue. To deal with this issue, in this paper, we are motivated to propose a light-weight and automated method for the detection of locker-ransomware. First, we conduct a thorough survey of the locker-ransomware's transaction market and perform a comprehensive analysis of locker-ransomware's behaviors. Second, to cope with the code obfuscation problem, we extract features of both displayed texts and background operations based on the observed behaviors. The fine-grained features are extracted from multiple sources, which can profile locker-ransomware in different aspects. Finally, we employ the ensemble of four machine learning algorithms for detection. The experimental results show that our method outperforms VirusTotal. It achieves the best performance with the detection accuracy of 99.98%.

INDEX TERMS Android, locker-ransomware, malware detection.

I. INTRODUCTION

Ransomware is a type of malware that blackmails users for ransom by blocking access to devices or data. In general ransomware can be categorized as locker-ransomware and crypto-ransomware. More specifically, locker-ransomware blocks users' interactions with the device by resetting the PIN code or popping up a full-screen window. The window covers the screen, which makes it impossible for users to interact with the device. The window may disappear only after the victims pay for the password and input it, as promised. Crypto-ransomware encrypts users' data and demands payment for the decryption. Locker-ransomware and crypto-ransomware occasionally appear together. It should be noted, however, paying the ransom does not guarantee that users can get the password and regain access to the devices. Most ransomware attackers are driven by profit. To instigate users to pay the ransom without hesitation and suspicion, the attackers turn to psychological tactics. They often equip the ransomware with humiliating messages and pornographic images. Their tricky use of both technology and psychology make ransomware a severe problem to the public, which requires to be urgently addressed.

Ransomware first appears on computers. The first ransomware, known as AIDS, was created in the late 1980s [1]. It encrypted files and demanded ransom by mail. With the development of the Internet and cryptocurrency, ransomware has become formidable with multiple ways of propagation and payment, causing significant losses in the past few years. In 2017, after infecting over 200,000 computers across 150 countries with economic losses of 4 billion dollars [2], WannaCry earned its reputation as the most widespread ransomware attack to date.

In recent years, smart mobile devices have been widely favored. As the most popular mobile operating system, Android dominates the market with a global share of 85.9% [3]. The open nature of Android system and the readily-available application distribution mechanism attract lots of attackers. Android devices have become lucrative targets and ideal hosts for ransomware to propagate.

The associate editor coordinating the review of this manuscript and approving it for publication was Alessio Vecchio.

Sypeng [4], which appeared in 2014, is the first Android locker-ransomware sample. It infected devices via a counterfeit Adobe Flash update message. It locked the screen and popped up a fake FBI message with a demand of $200 ransom. Koler [5] is notable for being the first Android ransomware worm, on account of its self-replicated behaviors. It would automatically send messages enclosing a download link to the worm to everyone in an infected device's contact list. In 2015, Lockerpin [6] appeared as the first real example of ransomware that was capable of resetting the PIN code of a device. It left victims with a locked mobile screen, demanding for a $500 ransom. In 2017, Kaspersky Lab detected over 544 thousand mobile ransomware [7]. McAfee Lab [8] claimed that mobile malware authors have set their sights firmly on monetization. They have added ransomware capabilities to create a new threat on the mobile platform. Clearly the mobile ransomware targeting Android has posed a great threat to Android ecosystem and users' properties.

China is the largest smart phone market in the world. In 2018, 50.37% of the population in China used a smartphone [9]. In December 2017, Android held a share of 78% of the mobile operating system market in China [10]. However, according to Mobile Tencent Analytics [11], 42% of Chinese users' system versions are still below Android 6.0. The drawbacks in the early versions of platform, e.g., lacking flexible permission management, expose users to high risks of being attacked. Moreover, on Chinese social networks, Android ransomware transaction has become a mature industrial chain. QQ [12] developed by Tencent [13] is one of the largest online communities worldwide with over 899 million active accounts [14]. Its popularity and anonymous registration mechanism make it a hot market for ransomware transactions. When we search the keyword "Android locker" in Chinese, we can find hundreds of QQ groups aiming at trading ransomware. The detailed tutorials shared in the groups enable those with little Android knowledge to make their own ransomware. The low technical requirement and high profit tempt a great number of common people into developing and propagating ransomware, making the situation even worse. The significant growth of ransomware targeting Android increasingly requires efficient methods that can automatically profile and detect them.

One of the challenges of effectively detecting locker-ransomware is that most of locker-ransomware is built with code obfuscation. The names of functions are replaced by simple letters, e.g., reset_password() is replaced by a(), which increases the difficulty in analyzing the code. In addition, locker-ransomware normally appears along with shells. It masques as popular apps for download. After being installed, the shell will immediately release the inside locker-ransomware to perform malicious behaviors. Under the protection of shells, amounts of locker-ransomware can escape existing detection methods.

In this paper, we focus on the Android locker-ransomware that are widely spread on Chinese social networks. Android locker-ransomware shows new tricks in development, masquerade and propagation. Equipped with psychological tactics, locker-ransomware has become a troublesome issue. To resolve this issue, first, we join the ransomware-trading groups and conduct a thorough survey of the transaction market. We acquire detailed information about traders as well as the complete transaction process. Second, we collect 301 latest unique samples distributed in real world. We perform a comprehensive analysis of locker-ransomware's behaviors and their techniques. Some interesting points are revealed, e.g., locker-ransomware is more likely to masquerade as hot game cheaters or red pocket grabbers. Third, we propose a light-weight framework to effectively profile and detect locker-ransomware. Observing that widgets are normally not on the obfuscation list, we make use of the texts shown in the widgets. Unlike typical malware, which hides itself from victims, locker-ransomware tends to interact with victims by making a clear notification reminding victims to pay the ransom. Keywords such as "lock" and "unlock" frequently appear thus we build a text feature set to detect the lock-related texts. However, some benign locker apps also have the same keywords, so texts alone are inadequate to distinguish them. We enlarge the feature set with background behaviors. Finally, an ensemble detector which combines four kinds of machine learning approaches is applied to detect locker-ransomware. Our ensemble method achieves the accuracy as high as 99.98%.

We make the following contributions:

(1) We make a thorough survey of the transaction of locker-ransomware on Chinese social networks. We describe the details about this industrial chain, including ways of ransomware developed and distributed. To the best of our knowledge, this is the first systematic work on detecting Android locker-ransomware distributed on Chinese social networks.

(2) We perform a comprehensive analysis of locker-ransomware's behaviors and their techniques. We provide a detailed description of their malicious behaviors. Both technical and psychological tricks are exposed.

(3) We extract six categories of features that combine displayed texts and background operations. One category of features is extracted from multiple sources, aiming at profiling one typical characteristic of locker-ransomware. Our method can handle common obfuscation and root shells.

(4) We propose an ensemble approach to effectively detect locker-ransomware. The detection result is determined by four kinds of machine learning algorithms. It outperforms antivirus engines in VirusTotal and achieves the accuracy of 99.98%. The experimental results demonstrate the effectiveness of the features and detection method.

The rest of this paper is organized as follows. Section II introduces related work on Android malware detection. Section III describes the propagation and behaviors of locker-ransomware. Section IV describes the proposed method. Section V describes the evaluation. The conclusion follows in Section VI.

II. RELATED WORK

A. ANDROID PLATFORM

To better understand the behaviors of Android locker-ransomware, we first introduce the basics of Android applications (apps). An Android app is a compressed Android Package (APK) file that contains the app's manifest file, code, resources, assets, and certificates. The AndroidManifest.xml declares the components in the app, including Activity, Service, Broadcast Receiver and Content Provider [15]. Apps require specific permissions to perform sensitive operations. The requested permissions are also declared in the manifest file. The implemented classes and methods are compiled in the Dex file format which is understandable by the Dalvik virtual machine. We can disassemble the code to readable Smali files and extract APIs the app has invoked. The resources, e.g., images, strings, .xml files that control the layout of activities, are in res/ directory. The layout of a floating window is controlled by these layout files. The assets that can be retrieved by AssetManager [16] are in assets/ directory. Some ransomware conceals the malicious part as assets and releases it after installation.

Users can download apps from multiple sources, including but not limited to official or third-party markets, websites and social networks. The various sources facilitate the distribution of apps, however, bring great risks at the same time. Though most sources deploy antivirus engines to prevent malware from sneaking in, malware is becoming much more sophisticated and stealthier to break the defense. Attackers can evade the detection by multiple tricks, e.g., compress malware in encrypted zip files.

B. ANDROID MALWARE DETECTION

Android malware detection has been a widely studied topic, and some inspiring work has been published. Arp *et al.* [17] presented DREBIN, which extracted static features such as component names, permissions, intents, API, etc. They considered linear SVM for the training task. Shabtai *et al.* [18] extracted features from Android app files, such as Java bytecode and XML files. They tested several feature selection approaches to find the most representative sets of features. Zhu *et al.* [19] extracted four kinds of features including permissions, system events, APIs and permissions. Then an ensemble random forest classifier was learned to detect whether an app was potentially malicious or not. They achieved an accuracy of 89.91%. Some work [20-25] extracted HTTP header and statistic information (e.g., number of received packages) to profile malapps' network behaviors, and machine learning algorithms were applied for the classification. Arora *et al.* [26] combined static permissions and dynamic network traffic information as features. For the detection method, they combined Supervised Learning (KNN Algorithm) and Unsupervised Learning (K-Medoids Algorithm) and achieved overall detection accuracy of 91.98%. Martinelli *et al.* [27] proposed BRIDEMAID which matched static n-grams and monitored device, app and user behaviors. Their evaluation was based on 2794 malapps with a detection accuracy of 99.7%. Canfora *et al.* [28] also applied frequencies of opcodes n-grams as features. Du *et al.* [29] divided a function call graph into community structures and used features of these community structures to detect malware. They reduced the computation time by improving the Girvan-Newman algorithm and using machine learning classification instead of a similarity comparison of subgraphs.

In previous work, we developed an anomaly detection system called Anomadroid [30] to profile normal behaviors of normal apps. Apps whose behaviors deviated from the normal profile were identified as malicious. We also built a framework [31] to effectively manage a big app market in terms of detecting malware and categorizing benign apps.

In recent years, researchers have paid great attention to ransomware. For PC ransomware, Chen *et al.* [32] monitored dynamic behaviors of the app and generated API call flow graphs. They converted 2-sequences of APIs as features and the frequency of corresponding 2-sequences as feature values. Then they adopted four machine learning classifiers for the detection. To detect crypto-ransomware, Scaife *et al.* [33] proposed CryptoDrop, an early-warning detection system that alerted a user during suspicious file activity. They identified three primary indicators suited to detect malicious file changes. Kharraz *et al.* [34] proposed a dynamic analysis system called UNVEIL. It generated an artificial user environment, and detected when ransomware interacted with user data.

Only a few work has been released on detecting mobile ransomware. Andronio *et al.* [35] proposed HelDroid to monitor threatening texts, locking and encryption behaviors. Particularly, they applied texts extraction and static code analysis to detect locker-ransomware. Maiorca *et al.* [36] proposed R-PackDroid, a supervised machine learning approach to discriminate between ransomware, generic malware and benign apps. They applied the presence of API packages as features. Chen *et al.* [37] focused on crypto-ransomware and proposed RansomProber. Based on the observation that ransomware did not display the encryption process during the attack, they utilized UI analysis technique to judge the legality of encryption operations.

The existing work has left some problems unsolved. First, the code obfuscation has been a great challenge for analysis and detection. Some approaches in previous work is incapable to deal with it. Second, most ransomware is protected by shells. Some approaches ignore the shells and unable to extract the ransomware inside, thus become vulnerable. Effective and efficient features and detection approaches are in great demand.

III. ANDROID LOCKER-RANSOMWARE

In this section, we will describe the propagation and behaviors of locker-ransomware. We find hundreds of QQ groups that are related to transactions of locker-ransomware. The transaction of ransomware has become a mature industry chain on Chinese social networks. We observed the following strategies about the propagation and behaviors of Android locker-ransomware.

A. PROPAGATION

1) CHINESE MARKET

To elude regulations and enhance transaction probabilities, the trading environment requires anonymity as well as convenience for traders to negotiate and pay. QQ's anonymous registration mechanism and its digital wallet perfectly match the requirements. Moreover, the large number of QQ users are potential customers for ransomware sellers. To sell ransomware, the sellers can either join existing QQ groups or create their own groups. QQ groups that sell ransomware can easily be found by searching the keyword "Android locker" in Chinese. The groups normally require membership fees less than ¥10. The number of group members ranges from dozens to over one thousand. For buyers, they can pay to join the group and send a message to all group members to ask for ransomware. It is worth noting that not only the group owner but all other members in the group can be sellers. A group is an open market. After negotiation, one ransomware normally costs less than ¥10. The buyer can pay the seller by QQ wallet and the seller will send him the ransomware.

In some groups, the group owners share ransomware development tools, video tutorials and source code. Once paid about ¥100, they also provide one-on-one tutoring on how to develop ransomware. Therefore, besides selling ransomware, sellers can profit from membership fees, ransom and tutoring fees. Although the unit-price seems unsatisfying, given the large number of buyers, the profit is tempting. Many common people and even victims are engaging in ransomware dealing, making the ransomware market prosperous on social networks. The transaction process can be seen in Fig. 1.

To curb the diffusion of ransomware, QQ has been equipped with an online antivirus engine to scan uploaded files and delete the identified malware. But the sellers manage to avert the scanning simply by compressing the ransomware in an encrypted .rar file and sending the file with password to buyers. Without the password, the antivirus engine is unable to extract and detect ransomware. As a result, the sellers can easily break the defense and share group files.

2) DEVELOPMENT TOOLS

Android Studio is the official integrated development environment for Android which runs on PC. However, to accelerate the development process, attackers are more likely to use AIDE [38], which can be operated on Android devices. It can edit, debug, compile, sign and run APKs with simple operations. To generate their own ransomware, new learners can make use of existing ransomware's source code by only changing unlock password and contact information. With detailed tutorials, one can complete the repackage procedure even with little knowledge of Android. The low requirements of hardware and skills make AIDE the most popular tool in ransomware development.

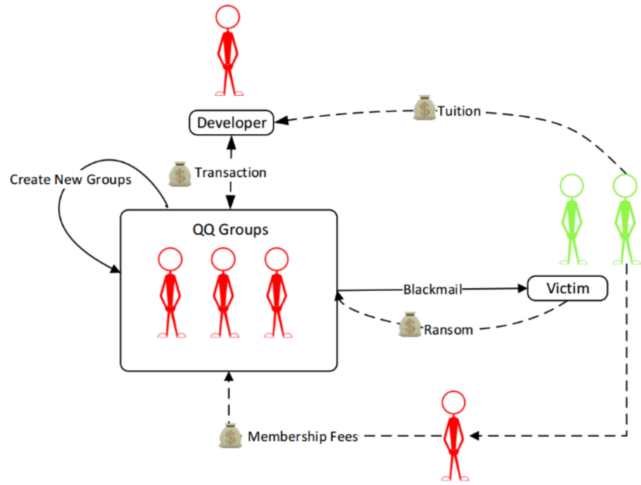


FIGURE 1. Transaction of locker-ransomware.

3) MASK

Ransomware always masquerades as popular apps to seduce downloading. According to the ransomware's names, locker-ransomware normally disguises as free cheaters of hot games, red pocket grabbers, QQ added-service providers and pornographic video players. These apps have some characteristics in common: (1) Most of the targeted apps are illegal and unavailable on official application markets. Therefore, whoever needs them have to turn to other sources, which makes opportunities for ransomware to distribute. (2) These apps tend to require more permissions and even the root permission. Victims who are familiar with these apps will not hesitate to grant the permissions during installation. (3) They have a broad user base. The ransomware targets on users with special needs but with little knowledge of Android. In disguise, ransomware seduces downloading without difficulty.

B. BEHAVIORS

1) LOCKING SCREENS

The most common behavior of locker-ransomware is to make the devices unavailable by creating a floating window on the screen or changing the PIN code. The floating window shows threatening messages, along with attacker's contact information and payment methods. The window does not respond to any touch event, and neither does the back or home button, making the device completely inaccessible. On Android platform, the placement and appearance of windows are controlled by WindowManager [39]. The ransomware creates the floating window by setting specific layout parameters of WindowManager. For example, the window is granted with top privilege when its type is set as TYPE_SYSTEM_ERROR. It can appear on top of everything. If its flag is set as FLAG_NOT_FOCUSABLE or FLAG_NOT_TOUCHABLE, the window cannot ever get key input focus or receive touch events, so the user is unable to send key or other button events to it. An example can be seen in Fig. 2.

To disable the physical or virtual key, e.g., home, back and menu, attackers overwrite the onKeyDown() method to

```
private void c()
{
    wmParams = new WindowManager.LayoutParams();
    mWindowManager = (WindowManager)getApplication()
                    .getSystemService(getApplication().WINDOW_SERVICE);
    wmParams.type = LayoutParams. TYPE_SYSTEM_ERROR;
    wmParams.format = PixelFormat.RGBA_8888;
    wmParams.flags = LayoutParams.FLAG_FULLSCREEN;
    wmParams.gravity = Gravity.CENTER_HORIZONTAL | Gravity.TOP;   Layout
    wmParams.x = 0;                                               Parameters
    wmParams.y = 0;
    wmParams.width = WindowManager.LayoutParams.FILL_PARENT;
    wmParams.height = WindowManager.LayoutParams.FILL_PARENT;
    LayoutInflater inflater = LayoutInflater.from(getApplication());
    mFloatLayout = inflater.inflate(R.layout.newone, null);
    mWindowManager.addView(mFloatLayout, wmParams);
}
```

FIGURE 2. Layout parameters.

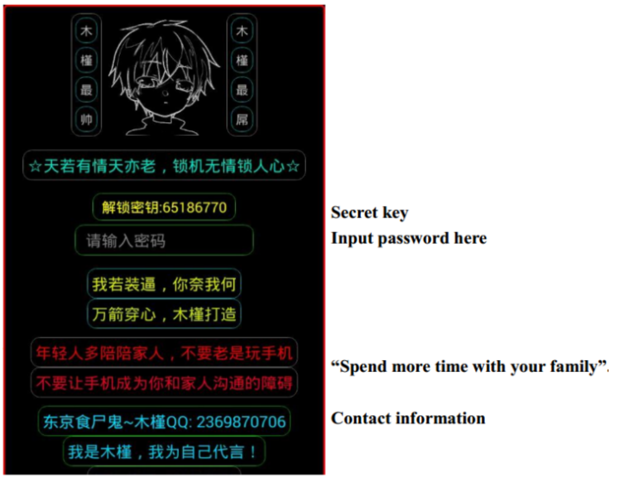


FIGURE 3. A screenshot of a locked phone.

ignore the press event. Some ransomware seduces users to grant the device administrator privilege. The ransomware creates an intent with the action of DevicePolicyManager.ACTION_ADD_DEVICE_ADMIN [40] to add itself as a new device administrator to the system. Once the administrator is active, the ransomware will have the permission to rewrite /data/system/password.key file and reset the PIN code to lock the screen. Even worse, the app with administrator permission cannot be uninstalled by being dragged to the trash, in the way common apps are. Thus it is difficult for users unfamiliar with system settings to get rid of ransomware.

Fig. 3 shows the screen of a locked phone. A text view is presented to the victim to insert a password. The generation of password is based on the secret key displayed on the screen. The attacker leaves his QQ number as contact information. Victims are expected to contact his QQ and pay the ransom in order to get the password.

2) BLOCKING PHONE CALLS

Some ransomware can block phone calls to make the device inaccessible. Whenever a call comes in, Android system will broadcast the event to all apps. The ransomware can listen to the event by registering a Broadcast Receiver [41]. It can extract the phone number from TelephonyManager [42]. If the number is not the one predefined by

```
<intent-filter
    android:priority="2147483647">
    <action android:name="android.intent.action.BOOT_COMPLETED"/>
</intent-filter>
```

FIGURE 4. An example of priority setting in the AndroidManifest.xml.

attackers, the ransomware will terminate the call. By setting the mode of ringer to RINGER_MODE_SILENT, the ransomware can mute the phone to make the blocking process underground.

3) HIGH PRIORITY

Most ransomware starts working as soon as the device completes booting. It registers a BroadcastReceiver and listens to BOOT_COMPLETED intent. In AndroidManifeat.xml, there is one attribute of intent-filter, named priority. It controls the order in which broadcast receivers are executed. Apps with higher priority values are called before those with lower values. Ransomware sets the priority as the largest value of integer, which gives it the highest priority. It will be the first app to know the device has completed booting, and then it aborts the broadcast and starts malicious services. An example can be seen in Fig. 4.

4) HIJACKING ACTIVITIES

After the device completes booting, the ransomware will detect the top activity at intervals by utilizing ActivityManager [43]. If the package name of the top activity does not belong to the ransomware, the ransomware can kill the top activity by calling killBackgroundProcesses() and restart its own activity.

5) ROOT SHELLS

Some ransomware utilizes root shells to masquerade as popular apps, as discussed in Section A. The root shell's task is to get root permission and release the ransomware in the system directory. The real ransomware that locks the screen hides in the /asset directory and disguises as a .so file, e.g., dalvik.so. After obtaining the root permission, the root shell will copy dalvik.so to /system/app/ directory and rename it as x.apk. After rebooting the system, the ransomware will be automatically installed on the device. Android regards it as a pre-installed system app which cannot be directly uninstalled. Although most ransomware can be removed by Android Debug Bridge (ADB) [44], it still bothers common users.

C. PASSWORD AND UNLOCKING

1) PASSWORD

a: NO PASSWORD

The ransomware pops up a full-screen floating window with only contact information. After receiving the ransom, the attacker will trigger the self-destructive program by calling or texting to the victim.

b: CONSTANT PASSWORD

Some passwords are constant values, e.g., birthdays or sentences that can be easily remembered. The floating window will disappear only after the correct password is entered. Constant passwords can be effortlessly located by professionals in the decompiled code.

c: ADVANCED PASSWORD

This kind of password adopts complicated logical calculations or encryption algorithms, e.g. AES, DES. To generate numerous different passwords, the ransomware can create a random number and take it into calculation. In some cases, the encryption progress includes several rounds to make the password unbreakable.

2) UNLOCKING

a: BY PASSWORD

The most common way to unlock the device is to pay for password. After the password is entered in the floating window, the window will disappear and the victim regains control of the device.

b: BY PHONE CALLS OR SMS

Another way is to trigger self-destructive program by calling or texting to the victim. The attacker predefines his phone number in the code of ransomware and registers a listener to monitor the phone state. When a call or SMS comes in, the ransomware will check whether it is from the attacker. If the calling number is the same as the predefined phone number, the ransomware will kill its service. But in this way, the attacker's phone number is exposed.

c: BY UNLOCKING TOOLS

To decrypt advanced password, attackers usually develop an unlocking tool to automatically make a reverse calculation of encryption algorithms. They will send victims the tool after receiving the ransom. The victims can generate the password and unlock the phone by themselves.

d: BY THE INTERNET

In order to protect their personal information, attackers apply an anonymous communication technology, e.g., Onion Network, to remotely unlock the device. This technology was originally designed to protect the privacy of sender and recipient, but was abused by amounts of malware.

D. PAYMENT

Most attackers leave their QQ numbers as contact information and the victims are instructed to make payments via QQ wallet. Unlike the high ransom in other countries, ransom of Chinese locker-ransomware is affordable, normally ¥20 - ¥50, so victims will not call the police with the mentality of "the less trouble, the better". Besides, QQ is pseudonymous, the victim is unable to find personal information about the attacker. Some attackers hide their QQ numbers by leaving a QR code instead. The QRcode links to the payment interface of attacker's QQ wallet. Therefore, the victims pay the ransom without any knowledge about the attackers. Though some international ransomware supports Bitcoin, we have not observed any samples on Chinese social networks.

E. LOW LEVEL OF API

We find that most locker-ransomware is based on early versions of API. 86% of our samples are targeted on API 21 which corresponds to Android 5.0. The early versions lack flexible permission control mechanism. Users need to grant the app install-time permissions, otherwise the app will not be successfully installed. If the users indiscreetly install ransomware and grant them dangerous permissions, the permissions are irrevocable, which gives ransomware chances to damage the device. However, 42% of Chinese users' system versions are still blow Android 6.0. They are more likely to subject to ransomware than those with higher versions of Android.

F. PSYCHOLOGIC TRICKS

Ransomware is motivated by profit. Attackers take full advantage of psychologic tricks to achieve the goal. They make use of users' greed and seduce them install the ransomware by offering free apps. To prevent victims from seeking help, they turn to fear and shame. Unlike most malware's underground and sneaky behaviors, ransomware tends to show off the victory. Ransomware displays threatening messages and pornographic images, along with loud music and high-frequent vibration to compel victims to pay the ransom. The victims will be too humiliated and scared to ask for professionals' assistance, and have to make the payments obediently.

IV. METHOD

A. OVERVIEW

In this section, we propose a framework to profile and detect locker-ransomware based on the behaviors discovered. As described in Fig. 5, after the apps are fed into the framework, text and behavior modules work in parallel to extract texts displayed in the UI and behaviors in the background. Six categories of features are extracted. With the ensemble of four classifiers, the apps are finally identified as benign apps or locker-ransomware.

B. FEATURE EXTRACTION

The extraction and selection of features are important in effectively detecting locker-ransomware. In previous work, one type of features is usually from one single source. For example, permissions are only extracted from the manifest file and APIs are from decompiled code. However, during the development of locker-ransomware, multiple existing ways serve to the same function, e.g., the texts in widgets may appear in layout files, string resources or decompiled code. Therefore, in our work, a type of features in our feature

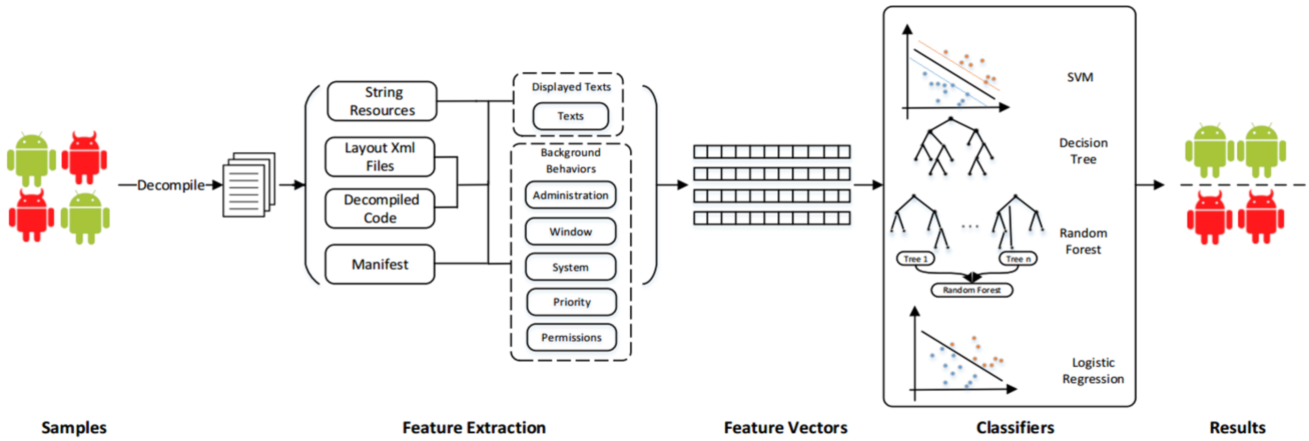


FIGURE 5. **Framework of our method.**

set may come from different sources. Based on the locker-ransomware's behaviors discussed in Section III, we have two observations:

(1) Unlike other malware, which hide its malicious behaviors, locker-ransomware makes clear statements that it is responsible for the locking behaviors.

(2) Unlike benign apps' various functions, locker-ransomware's function is straightforward, e.g., creating a floating window or resetting the PIN code to lock the screen.

Based on the observations, we take both displayed texts and background behaviors into consideration. We extract the following six categories of features. The summary of features can be seen in Table 1.

1) TEXT

Since locker-ransomware aims for ransom, the texts displayed on UI widgets are much different from that of benign apps. For example, most of ransomware's texts contain keywords of "lock", "unlock" and "pay", which is rare in benign apps. We define 22 keywords related to locking and unlocking behaviors. The texts in widgets can be set in multiple ways: (a) layout xml files, (b) string resources and (c) decompiled code. We take a comprehensive analysis of these files to get the texts.

2) ADMINISTRATOR

Amounts of locker-ransomware request device administrator privilege to reset the PIN code. In order to use the device administrator API, (a) the app's manifest should include the BIND_DEVICE_ADMIN permission and an intent filter to respond to the ACTION_DEVICE_ADMIN_ENABLED intent. (b) In the XML files, the app needs to declare the relevant security policies, e.g.," reset-password", "wipe-data", etc. An example can be seen in Fig. 6. (c) In the code, the attackers overwrite the APIs in DeviceAdminReceiver class to implement locking behaviors. We extract these administrator-related operations as a feature set.

TABLE 1. **Features.**

	Feature	Source	Number of features
1	Text	A, B, C	22
2	Administrator	B, C, D	7
3	Window property	C, D	9
4	System operation	C	6
5	Priority	C, D	2
6	Permission	D	1791

A. String resources B. Layout xml files
C. Decompiled code D. Manifest

```
<device-admin xmlns:android="http://schemas.android.com/apk/res/android" >
    <uses-policies>
        <limit-password />
        <watch-login />
        <reset-password />
        <force-lock />
        <wipe-data />
    </uses-policies>
</device-admin>
```

FIGURE 6. **The layout of a device administrator.**

3) WINDOW PROPERTY

A great amount of locker-ransomware implements the locking behavior by creating a floating window. The ransomware manipulates properties of the window to keep it from responding to users. By setting the layout parameters in WindowManager class, it creates a floating window which displays on top and ignores all the touch events. We extract the APIs related to window properties to form a feature set.

4) SYSTEM OPERATION

For those protected by shells, the root shell can remount the system directory and release the locker-ransomware after obtaining the root permission. The commands of "remount", "chmod", "cp" and "mv" targeting system directory are regarded as suspicious behaviors.

5) PRIORITY

To achieve the goal of locking the screen as soon as the device completes booting, locker-ransomware will register an intent-filter to listen to the BOOT_COMPETED broadcast. In the meanwhile, it sets the priority of intent-filter as the highest value in order to be the first one to receive the message. Some locker-ransomware even aborts broadcast afterwards. Setting priority in the manifest file and aborting broadcast in the code are considered to be suspicious.

6) PERMISSION

Android controls access to system resources with permissions. Specific permissions are required when an app interacts with system APIs or databases. The permissions are declared in AndroidManidest.xml. Both developer-defined and system-defined permissions are taken into consideration.

High dimensional feature vectors are required for classification. A feature vector, which represents a malware sample, is defined as $F = (f_1, f_2, \ldots f_n)$, where n is the number of features. The features we extract are all binary features. If the app has the i-th feature, f_i will be set to 1, otherwise 0.

C. THE ALGORITHMS FOR DETECTION

We employ four machine learning approaches as classifiers, namely, Support Vector Machine (SVM) [45], Decision Tree (DT) [46], Random Forest (RF) [47] and Logistic Regression (LR) [48]. Since each algorithm has two sides, to compensate for the disadvantages, we apply ensemble learning to obtain the final detection result. First, the classifiers work in parallel to predict the label of the test app. Then the final label is determined by majority opinions of the four classifiers, e.g., the test app's final label will be "benign" if two or more classifiers identify it as a benign app, and vice versa. The ensemble decision outperforms single machine learning algorithm in locker-ransomware detection.

V. EVALUATION

In this section, we conduct a series of experiments to evaluate the features and the detection algorithms.

A. DATA SET

The benign apps are downloaded from Anzhi Market [49], one of the largest third-party markets in China. We collect 9 categories of popular apps. Specifically, we download many benign wallpaper apps. We intend to test whether our features can distinguish locker-ransomware from benign wallpaper apps. The categories of apps can be seen in Table 2.

Since we focus on the locker-ransomware on Chinese social networks, our ransomware samples are collected in real world. We join 30 most popular ransomware-transaction QQ groups and download 664 locker-ransomware from the shared files. We remove the repeated ones according to their SHA1 values and finally get 301 locker-ransomware in total.

We employ Apktool [50] to decompile locker-ransomware APKs and obtain targeted files. To deal with the problem of root shells, we try to decompile each file in /assets/ directory, regardless of its extension name. As a result, in the 301 APKs, 226 more APKs are found. We regard the outside shell and inside APK as one APK.

TABLE 2. Benign dataset.

	Category	Number of samples
1	Photography	248
2	Communication	354
3	Weather	371
4	Music	394
5	Finance	884
6	News	983
7	Social	1052
8	Shopping	1796
9	Wallpaper	9669
Total		15751

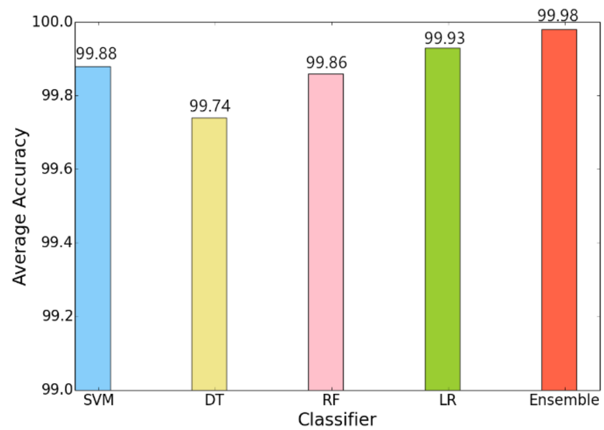


FIGURE 7. Average accuracy of classifiers.

The experiments are conducted on a PC with a quad-core 3.4 GHz i7 processors and 16G memory. Our method is implemented in Python language. The whole detection process can automatically conduct by shell commands without manual intervention.

B. DETECTION RESULTS

We test the effectiveness of extracted features on Support Vector Machine (SVM), Decision Tree (DT), Random Forest (RF) and Logistic Regression (LR) respectively. Then we apply the ensemble learning to make the final decision. We use n-fold cross-validation in our experiments to reduce overfitting. The average accuracy of each classifier is shown in Fig. 7.

From Table 3 we can see that all the detection accuracy is above 99%. It demonstrates that our features can effectively distinguish locker-ransomware from benign apps, regardless of the detection algorithms. For single algorithm, DT achieves the highest TPR which is 98.92%. LR achieves the highest accuracy of 99.93% and the lowest FPR of 0.02%. It is seen that after employing an ensemble of all classifiers with majority voting mechanism, we achieve the highest accuracy of 99.98%. Our ensemble method performs better than the single machine learning method.

To evaluate the time consumption of our method, we take an analysis of the execution time of each phase on

TABLE 3. Detection results.

Classifier	TPR (%)	FPR (%)	F-score(%)	Accuracy (%)
SVM	97.85	0.09	94.79	99.88
DT	98.92	0.24	91.92	99.74
RF	94.62	0.03	97.80	99.86
LR	96.77	0.02	97.83	99.93
Ensemble	98.92	0.00	99.46	99.98

TABLE 4. Time consumption of each phase.

	A	B	C				D
			SVM	DT	LR	RF	
(s)	3250	31	0.15	0.07	0.75	0.74	0.026
Total: 3282.74 s			Average: 10.91 s/sample				

A. Decompilation B. Feature extraction
C. Detection of single classifiers D. Ensemble

our 301 ransomware dataset, which is shown in Table 4. Decompiling is the most time consuming step, accounting for 99% of the total time. It takes 3250 seconds to decompile 301 ransomware samples and their inner APKs, which is 10.8 seconds per sample. Decompiling is time consuming because one ransomware sample may contain several inner hidden APKs. It would take much less time to decompile apps without root shells. Feature extraction is light-weighted, taking 0.1 second on each sample. Then on classification step, four classifiers work in parallel. The prediction process does not take much time. Finally, we apply the ensemble results of the four classifiers. In general, it takes an average of 10.91 seconds to analyze each sample. The experimental results indicate that our method is efficient to detect locker-ransomware.

C. FEATURE COMPARISON

To figure out the behaviors that locker-ransomware prefers, we rank the features according to their frequency. The top frequently appeared features are shown in Table 5. Up to 96.25% of locker-ransomware requests receive_boot_completed permission. It is understandable that most locking behaviors are triggered by the completed-booting broadcast. System_alert_window is a signature-level permission. It allows an app to create a window shown on top of all other apps. Although the Developer Guide [51] suggests that very few apps should use this permission, 92.49% of locker-ransomware requests it. In addition, 81.91% of locker-ransomware intends to obtain the highest priority of filter intents. And as expected, over 80% of ransomware has keywords of ''lock'' and ''unlock''. The displayed texts remind victims that their devices have been locked and they have to pay the ransom to unlock the phones.

To figure out the differences between locker-ransomware and benign apps, we also make a comparison of their features. It is worth pointing out a category of benign apps named

TABLE 5. Top features of locker-ransomware.

Feature	% in locker-ransomware
Receive_boot_completed	96.25
System_alert_window	92.49
Window_proprity	83.62
Priority	81.91
Keyword_lock	80.55
Keyword_unlock	80.34
Administrator	70.65
Mount_unmount_filesystems	68.94

TABLE 6. Average number of permissions an app request in different datasets.

Dataset	A	B	C	D
Average number of permissions	7	11	8	16

A. Locker-ransomware dataset B. All benign app dataset
C. Wallpaper apps D. Benign apps exclude wallpaper apps

wallpaper, has similar functions to manipulate the screen. Some wallpaper apps can not only change wallpapers, but can also turn off and lock the screen or reset password. On one hand, the functional similarities lead to the similarities in permission request. Due to their purposive functions, locker-ransomware and wallpaper apps tend to request less permissions than other categories of benign apps which provide more complicated services. We compare the average number of permissions an app request in different datasets: (A) locker-ransomware dataset, (B) benign app dataset, (C) wallpaper apps, (D) benign apps exclude wallpaper apps, which is shown in Table 6. The average number of permissions that locker-ransomware and wallpaper apps request is around 8, while other kinds of benign apps request 16 permissions. One the other hand, locker-ransomware and wallpaper apps do have differences. The most intuitive difference is that benign wallpaper apps' behaviors follow users' commands, while locker-ransomware compulsorily controls phones, which are irrelevant to users' expectations. We rank he features by their frequency and list the most different features in ranking, which is shown in Fig. 8. We can see that 70.65% of locker-ransomware asks for administrator privilege, while only 2.23% wallpaper apps have this feature. 65.53% of locker-ransomware aborts the broadcast after they received completed-booting signal, while only 5.94% of benign apps do. In contrast, most locker-ransomware does not request the location information, leading to the percentage of 2.05%, compared with 52.41% of all benign apps.

D. COMPARISON WITH OTHER WORK

1) COMPARISON WITH VIRUSTOTAL

VirusTotal is a state-of-the-art tool that provides free checking of files for viruses. It contains over sixty widely used antivirus engines e.g., McAfee and Symantec. It analyzes

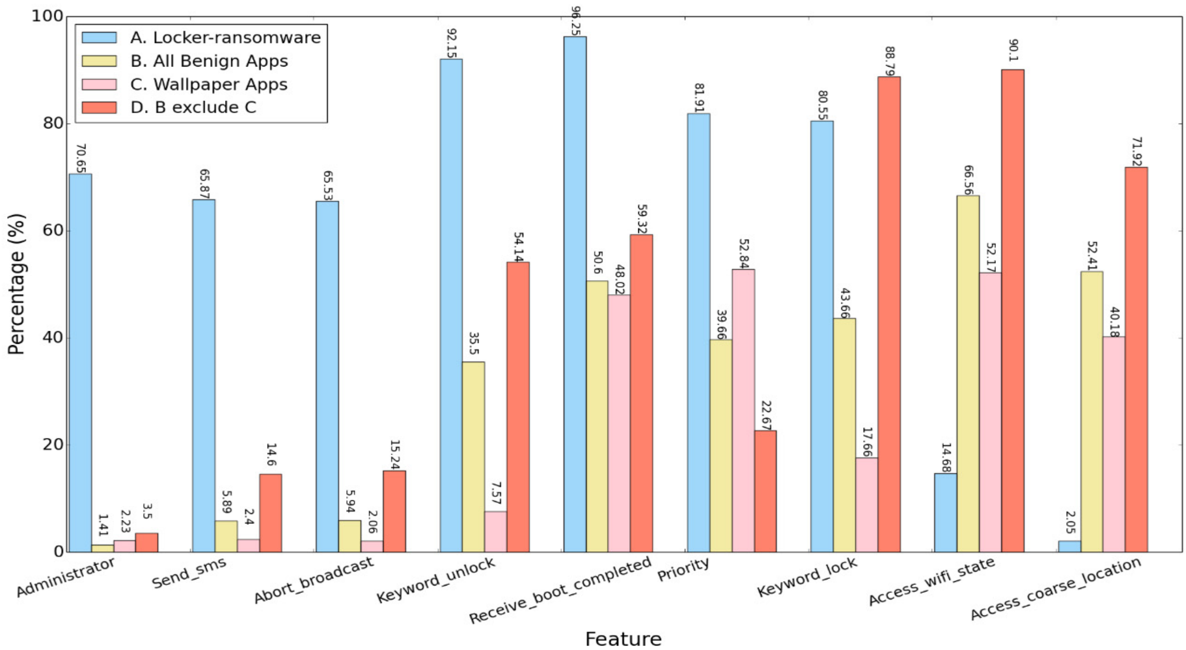


FIGURE 8. Top features in different datasets.

the submitted apps and returns the detection results of each antivirus engine. The app is labelled as "True" or "False" which respectively means malicious or benign. Much previous work employed the detecting results of VirusTotal to label their datasets. We upload the 301 locker-ransomware samples to VirusTotal to test the detecting performance of online antivirus engines.

Among the 68 antivirus engines, ESET-NOD32 wins the first place with average detection accuracy of 98.98%, followed by McAfee with accuracy of 95.90%. For Symantec, the accuracy is 84.30%. Fig. 9 shows the average detection accuracy of antivirus engines in VirusTotal. Each point represents an antivirus engine. The overall detection result is not desirable. Only 20 engines' accuracy is above 90%. 36 engines' accuracy is below 50%. Microsoft only reaches 13.65%.

Based on the detection result of each locker-ransomware sample, we find that only 50%-60% of antivirus engines can detect them. None of samples triggers 70% or more antivirus engines, which can be seen in Fig. 10. We notice that a sample has the lowest detection rate of 25.42%, which means only 25.42% of antivirus engines regard it as a malware sample. This sample is covered up by a root shell. It masquerades as an automated red pocket grabber, and seduces user to grant the root permission. Then it releases the locker-ransomware in system directory and reboots the device. After the device has completed booting, the screen is locked by a floating window. It is clear that most antivirus engines are not able to handle shells. They can only analyze the outer layer. If nothing is abnormal, they will regard the sample as a benign app. However, the inner layer which is ignored by most

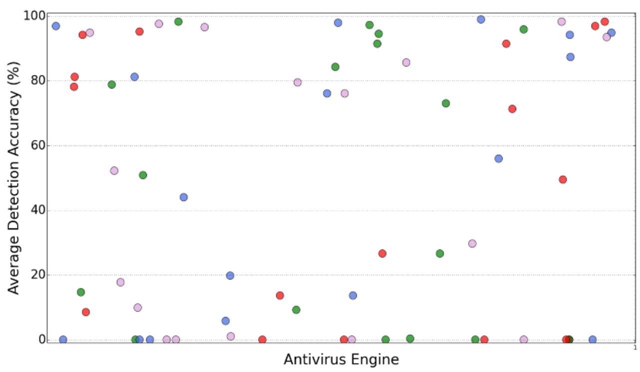


FIGURE 9. The average detection accuracy of antivirus engines in VirusTotal. The nodes represent antivirus engines. The y-axis represents their detection accuracy on our locker-ransomware dataset.

antivirus engines is the real malicious part. In our approach, we decompile each file in /assets/ directory, regardless of its extension name. Features of both the outside shell and the inside hidden app are taken into consideration. Thus our approach achieves better performance than antivirus engines in VirusTotal.

2) COMPARISON WITH R-PACKDROID

To facilitate comparison with existing work, we conduct experiments with R-PackDroid that was proposed by Canfora *et al.* [28]. R-PackDroid is a supervised machine learning system for the detection of Android ransomware. It characterizes apps by a list of system API packages and employs Random Forest for the classification task. We randomly select 1500 benign apps from our benign dataset,

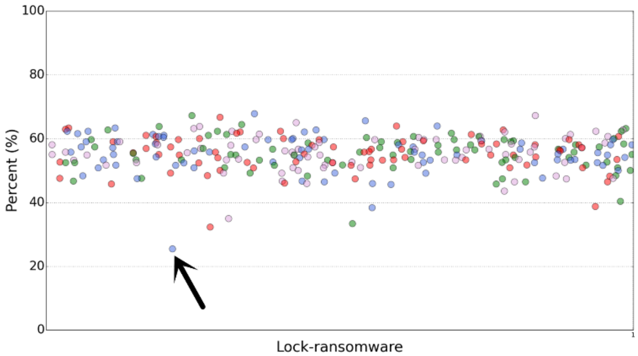


FIGURE 10. Percent of antivirus engines that can detect locker-ransomware samples. The nodes represent locker-ransomware samples. The y-axis represents the percentage of antivirus engines that identify it as malware. Only 25.42% of antivirus engines regard the pointed sample as malware.

TABLE 7. Detection results of R-droid and our ensemble method.

Classifier	TPR (%)	FPR (%)	F-score(%)	Accuracy (%)
R-Droid	86.00	3.42	79.63	95.49
Ensemble Method	96.00	0.00	97.96	99.16

together with 300 locker-ransomware samples, based on which we implement and evaluate R-PackDroid. We use n-fold cross-validation in our experiments and the detection result is shown in Table 7. Our method achieves better detection results than R-PackDroid. The TPR of our method is 10% higher than R-Droid. It verifies that our features can better profile locker-ransomware and distinguish it from benign apps.

E. DISCUSSION

To reduce the size of APKs and evade the detection, ransomware applies obfuscation which transforms the code without affecting the functionality to make it obscure and unintelligible. Much previous work [17, 52] that uses API-name-based features to detect malapps may become less effective since obfuscation can rename classes or functions. Based on our observations, however, both benign and locker-ransomware normally do not obfuscate widgets. Thus even though the information from APIs is invalid, we can still figure out its attempt according to the texts shown in the widgets. We thoroughly extract texts from three sources that configure widgets: layout files, string resources and decompiled code. Along with other representative features, e.g., window properties, we comprehensively profile the locker-ransomware behaviors.

The main reason we applied static analysis is that most informative features can be efficiently extracted by static code analysis. On the contrary, an emulator is locked after each execution of the locker-ransomware sample during the dynamic analysis. Although we can fix it with ADB commands, static features can better meet the demands of being efficient and effective when faced with large scale of ransomware. Dynamic payloads may affect the detection performance. However, dynamic payloads and root shells both can achieve the aim of hiding malicious components and evading detection. Based on our survey, locker-ransomware prefers root shells because of the lower technical requirements. The root shell's task is to get the root permission and release the locker-ransomware in the system directory. In this way, we dig out the inner hidden APK to overcome the root shell problem and achieve better performance than VirusTotal.

VI. CONCLUSION

Locker-ransomware applies various techniques to seduce users to download, makes the devices inaccessible and compels victims to pay ransom, thus poses a great threat to users' properties. In addition, Code obfuscation and shells enable ransomware to bypass the detection of many antivirus engines, which makes Android ecosystem insecure. Moreover, the transaction of locker-ransomware has become a severe problem on Chinese social networks. However, it does not attract much attention. To the best of our knowledge, this is the first systematic work on analyzing Android locker-ransomware distributed on Chinese social networks.

In this work, we conduct a comprehensive analysis of locker-ransomware's transaction process and behaviors. To protect users from locker-ransomware, we are motivated to provide a light-weight and effective detection framework. To overcome the challenges of code obfuscation and root shells, we extract six types of features from multiple sources. The features can highly summarize locker-ransomware's behaviors. We employ an ensemble of four classifiers by means of majority voting to have the final detection results. The experimental results demonstrate the effectiveness of features and the method. We achieve the best detection result with the accuracy of 99.98%.

In the future, we will continue to closely watch the latest development of ransomware and explore more informative features to better profile locker-ransomware in the aim to effectively monitor and detect them.

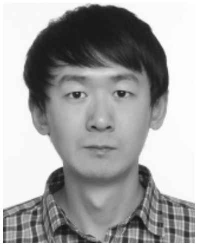
REFERENCES

[1] *Introduction of Ransomware*. Accessed: May 14, 2018. [Online]. Available: https://www.malwarebytes.com/ransomware/
[2] *Wannacry*. Accessed: May 14, 2018. [Online]. Available: https://www.cbsnews.com/news/wannacry-ransomware-attacks-wannacry-virus-losses/
[3] *Report of Gartner*. Accessed: May 14, 2018. [Online]. Available: https://www.gartner.com/newsroom/id/3859963
[4] *Sypeng*. Accessed: May 14, 2018. [Online]. Available: https://www.carbonite.com/blog/article/2017/08/the-evolution-of-a-cybercrime-a-timeline-of-ransomware-advances/
[5] *Koler*. Accessed: May 14, 2018. [Online]. Available: https://www.vpnmentor.com/blog/history-ransomware-threat-past-present-and-future/
[6] *Lockerpin*. Accessed: May 14, 2018. [Online]. Available: https://thehackernews.com/2015/09/android-lock-ransomware.html
[7] (2017). *Mobile Malware Evolution*. [Online]. Available: https://securelist.com/mobile-malware-review-2017/84139/
[8] *2018. McAfee Mobile Threat Report Q1*. [Online]. Available: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-mobile-threat-report-2018.pdf

[9] *Smartphone Penetration Rate as Share of the Population in China*. Accessed: May 14, 2018. [Online]. Available: https://www.statista.com/statistics/321482/smartphone-user-penetration-in-china/
[10] *Market share in China*. Accessed: May 14, 2018. [Online]. Available: https://www.statista.com/statistics/262176/market-share-held-by-mobile-operating-systems-in-china/
[11] *Percentage of Android Versions in China*. Accessed: May 14, 2018. [Online]. Available: https://mta.qq.com/mta/data/device/os
[12] *Introduction of QQ*. Accessed: May 14, 2018. [Online]. Available: https://en.wikipedia.org/wiki/Tencent_QQ
[13] *Tencent Homepage*. Accessed: May 14, 2018. [Online]. Available: https://www.tencent.com/en-us/index.html
[14] *Active Accounts of QQ*. Accessed: May 14, 2018. [Online]. Available: https://en.wikipedia.org/wiki/List_of_virtual_communities_with_more_than_100_million_active_users
[15] *Fundamentals of Android*. Accessed: Jun. 16, 2018. [Online]. Available: https://developer.android.com/guide/components/fundamentals
[16] *Introduction of Asset Manager*. Accessed: Jun. 16, 2018. [Online]. Available: https://developer.android.com/reference/android/content/res/AssetManager
[17] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "DREBIN: Effective and explainable detection of Android malware in your pocket," in *Proc. NDSS*, Aug. 2014, pp. 23–26.
[18] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, "'Andromaly': A behavioral malware detection framework for Android devices," *J. Intell. Inf. Syst.*, vol. 38, no. 1, pp. 161–190, 2012.
[19] H.-J. Zhu, T.-H. Jiang, B. Ma, Z.-H. You, W.-L. Shi, and L. Cheng, "HEMD: A highly efficient random forest-based malware detection framework for Android," *Neural Comput. Appl.*, vol. 30, no. 11, pp. 3353–3361, 2018.
[20] C. Yang, J. Zhang, and G. Gu, "Understanding the market-level and network-level behaviors of the Android malware ecosystem," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, Atlanta, GA, USA, Jun. 2017, pp. 2452–2457.
[21] M. R. Amin, M. Zaman, M. S. Hossain, and M. Atiquzzaman, "Behavioral malware detection approaches for Android," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2016, pp. 1–6.
[22] J. Li, L. Zhai, X. Zhang, and D. Quan, "Research of Android malware detection based on network traffic monitoring," in *Proc. Ind. Electron. Appl. (ICIEA)*, Hangzhou, China, Jun. 2014, pp. 1739–1744.
[23] S. Garg, S. Peddoju, and A. K. Sarje, "Network-based detection of Android malicious apps," *Int. J. Inf. Secur.*, vol. 16, no. 4, pp. 385–400, 2017.
[24] Z. Li, L. Sun, Q. Yan, W. Srisa-An, and Z. Chen, "DroidClassifier: Efficient adaptive mining of application-layer header for classifying Android malware," in *Proc. Int. Conf. Secur. Privacy Commun. Syst.*, Guangzhou, China, 2016, pp. 597–616.
[25] A. Arora and S. K. Peddoju, "Minimizing network traffic features for Android mobile malware detection," in *Proc. Int. Conf. Distrib. Comput. Netw.*, Hyderabad, India, 2017, Art. no. 32.
[26] A. Arora, S. K. Peddoju, V. Chouhan, and A. Chaudhary, "Hybrid Android malware detection by combining supervised and unsupervised learning," in *Proc. 24th Annu. Int. Conf. Mobile Comput. Netw.*, New Delhi, India, 2018, pp. 798–800.
[27] F. Martinelli, F. Mercaldo, and A. Saracino, "BRIDEMAID: An hybrid tool for accurate detection of Android malware," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, New York, NY, USA, 2017, pp. 899–901.
[28] G. Canfora, A. De Lorenzo, E. Medvet, F. Mercaldo, and C. A. Visaggio, "Effectiveness of opcode ngrams for detection of multi family Android malware," in *Proc. Int. Conf. Availability, Rel. Secur. (ARES)*, Toulouse, France, Aug. 2015, pp. 333–340.
[29] Y. Du, J. Wang, and Q. Li, "An Android malware detection approach using community structures of weighted function call graphs," *IEEE Access*, vol. 5, pp. 17478–17486, 2017.
[30] D. Su, W. Wang, X. Wang, and J. Liu, "Anomadroid: Profiling Android applications' behaviors for identifying unknown malapps," in *Proc. Trustcom*, Tianjin, China, Aug. 2016, pp. 691–698.
[31] W. Wang, Y. Li, X. Wang, J. Liu, and X. Zhang, "Detecting Android malicious apps and categorizing benign apps with ensemble of classifiers," *Future Gener. Comput. Syst.*, vol. 78, pp. 987–994, Jan. 2018.
[32] Z.-G. Chen, H.-S. Kang, S.-N. Yin, and S.-R. Kim, "Automatic ransomware detection and analysis based on dynamic API calls flow graph," in *Proc. RACS*, Krakow, Poland, 2017, pp. 196–201.
[33] N. Scaife, H. Carter, P. Traynor, and K. R. B. Butler, "CryptoLock (and drop it): Stopping ransomware attacks on user data," in *Proc. Int. Conf. Distrib. Comput. Syst.*, Jun. 2016, pp. 303–312.
[34] A. Kharaz, S. Arshad, C. Mulliner, W. Robertson, and E. Kirda, "UNVEIL: A large-scale, automated approach to detecting ransomware," in *Proc. USENIX SecurSymp*, 2016, pp. 1–17.
[35] N. Andronio, S. Zanero, and F. Maggi, "HelDroid: Dissecting and detecting mobile ransomware," in *Proc. Int. Workshop Recent Adv. Intrusion Detection*, 2015, pp. 382–404.
[36] D. Maiorca, F. Mercaldo, G. Giacinto, C. A. Visaggio, and F. Martinelli, "R-PackDroid: API package-based characterization and detection of mobile ransomware," in *Proc. Symp. Appl. Comput.*, Marrakech, Morocco, 2017, pp. 1718–1723.
[37] J. Chen, C. Wang, Z. Zhao, K. Chen, R. Du, and G.-J. Ahn, "Uncovering the face of Android ransomware: Characterization and real-time detection," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 5, pp. 1286–1300, May 2018.
[38] *AIDE Homepage*. Accessed: Jun. 16, 2018. [Online]. Available: http://www.android-ide.com/
[39] *WindowManager*. Accessed: Jun. 16, 2018. [Online]. Available: https://developer.android.com/reference/android/view/WindowManager
[40] *DevicePolicyManager*. Accessed: Jun. 16, 2018. [Online]. Available: https://developer.android.com/reference/android/app/admin/DevicePolicyManager
[41] *BroadcastReceiver*. Accessed: Jun. 16, 2018. [Online]. Available: https://developer.android.com/reference/android/content/BroadcastReceiver
[42] *TelephonyManager*. Accessed: Jun. 16, 2018. [Online]. Available: https://developer.android.com/reference/android/telephony/TelephonyManager
[43] *ActivityManager*. Accessed: Jun. 16, 2018. [Online]. Available: https://developer.android.com/reference/android/app/ActivityManager
[44] *Android Debug Bridge (ADB)*. Accessed: Jun. 16, 2018. [Online]. Available: https://developer.android.com/studio/command-line/adb
[45] V. Vapnik, *The Nature of Statistical Learning Theory*, vol. 8. New York, NY, USA: Springer, 1995, no. 6, pp. 988–999.
[46] J. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA, USA: Morgan Kaufmann, 1992.
[47] L. Breiman, "Random forest," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.
[48] *Scikit-Learn Library*. Accessed: Jun. 16, 2018. [Online]. Available: http://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
[49] *Anzhi Market*. Accessed: Jun. 16, 2018. [Online]. Available: http://www.anzhi.com/
[50] *Apktool*. Accessed: Jun. 16, 2018. [Online]. Available: https://ibotpeaches.github.io/Apktool/
[51] *Developer Guide*. Accessed: Jun. 16, 2018. [Online]. Available: https://developer.android.com/reference/android/Manifest.permission#SYSTEM_ALERT_WINDOW
[52] M. Qiao, A. H. Sung, and Q. Liu, "Merging permission and API features for Android malware detection," in *Proc. Int. Congr. Adv. Appl. Inform.*, Jul. 2016, pp. 566–571.

DAN SU received the B.S. degree from Beijing Jiaotong University, China, in 2014, where she is currently pursuing the Ph.D. degree with the School of Computer and Information Technology. Her main research interest includes mobile security.

JIQIANG LIU received the B.S. and Ph.D. degrees from Beijing Normal University, in 1994 and 1999, respectively. He is currently a Professor with the School of Computer and Information Technology, Beijing Jiaotong University. He has published over 70 scientific papers in various journals and international conferences. His main research interests include trusted computing, cryptographic protocols, privacy preserving, and network security.



XIAOYANG WANG received the B.S. degree from Beijing Jiaotong University, China, in 2014, where he is currently pursuing the Ph.D. degree with the School of Computer and Information Technology. His main research interests include complex networks and data mining.

WEI WANG received the Ph.D. degree in control science and engineering from Xi'an Jiaotong University, China, in 2006. He is currently a Professor with the School of Computer and Information Technology, Beijing Jiaotong University, China. He was a Postdoctoral Researcher with the University of Trento, Italy, from 2005 to 2006. He was a Postdoctoral Researcher with TELECOM Bretagne and INRIA, France, from 2007 to 2008. He was a European ERCIM Fellow with the Norwegian University of Science and Technology (NTNU), Norway, and also with the Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg, from 2009 to 2011. He visited INRIA, ETH, NTNU, CNR, and New York University Polytechnic. He has authored or co-authored over 60 peer-reviewed papers in various journals and international conferences. His main research interests include mobile, computer, and network security. He is a young AE of the *Frontiers of Computer Science* Journal.