applied sciences

MDPI

Article

KRDroid: Ransomware-Oriented Detector for Mobile Devices Based on Behaviors

Senmiao Wang [1], Sujuan Qin [1,*], Jiawei Qin [1,*], Hua Zhang [1], Tengfei Tu [1], Zhengping Jin [1,*] and Jing Guo [2]

[1] State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China; wangsenmiao@bupt.edu.cn (S.W.); zhanghua_288@bupt.edu.cn (H.Z.); tutengfei.kevin@bupt.edu.cn (T.T.)

[2] National Computer Network Emergency Response Technical Team/Coordination Center of China (CNCERT/CC), Beijing 100029, China; guojing@cert.org.cn

* Correspondence: qsujuan@bupt.edu.cn (S.Q.); qinjiawei@bupt.edu.cn (J.Q.); zhpjin@bupt.edu.cn (Z.J.)

Abstract: Ransomware has become a serious threat on Android and new cases of ransomware are continuously growing. Most existing ransomware detectors use sensitive text or APIs to detect ransomware. Some goodware applications with the functionalities of locking screen and encrypting files have similar behaviors with ransomware. It is difficult for ransomware detectors to identity them. In this paper, we made detailed analyses of three kinds of active ransomware. We proposed a behavior-based ransomware detector on Android, called KRDroid. KRDroid deploys on servers or PCs, that is, ransomware cannot be activated and cause any loss during testing. Experiments showed that our ransomware-oriented detector can find 1809 of 1862 unseen ransomware. It can also distinguish goodware with similar ransom behaviors to ransomware with an accuracy of 97.5%.

Keywords: ransomware detector; behavior-pattern-based detection; ransomware analysis; Android

Citation: Wang, S.; Qin, S.; Qin, J.; Zhang, H.; Tu, T.; Jin, Z.; Guo, J. KRDroid: Ransomware-Oriented Detector for Mobile Devices Based on Behaviors. *Appl. Sci.* **2021**, *11*, 6557. https://doi.org/10.3390/app11146557

Academic Editor: Eui-Nam Huh

Received: 1 June 2021
Accepted: 14 July 2021
Published: 16 July 2021

1. Introduction

With the unprecedented outbreak of different kinds of ransomware in recent years, devices and files from all walks of life have been locked. It has brought economic losses to both individuals and enterprises. Ransomware have been growing from the last few years since 2017, and it has become a key threat to mobile devices [1]. There are at least 150 countries with 300,000 users are attacked by the WannaCry (a kind of ransomware) according to the statistics. It causes economic losses as high as USD 8,000,000,000. According to the new report released by Precise Security, WannaCry remains one of the most influential ransomware in 2019. In 2019, a new kind of ransomware, Silex, was found by researchers. The spread of these types of ransomware is rapid. Silex first affected 350 devices and then quickly expanded to more than 1500 devices. According to the statistics released by Coveware, the payment ransomware require in the second quarter of 2020 is four times higher than in 2019 [2].

It is reported that the number of mobile devices based on the Android platform has sharply increased [3–6]. It is worth noting that the number of Android devices will be approximately 6.1 billion by the end of 2020 [6–9]. At present, ransomware running on Android is still a threat to mobile devices. In this work, we mainly focus on detecting ransomware based on the Android platform for mobile devices.

Ransomware detection on Windows has been relatively well established. For instance, 2entFOX can detect highly survivable ransomware with high detection accuracy and low false-positive rate [10]. UNVEIL uses filesystem to monitor and OCR to detect locking devices and encrypting files ransomware [11]. ShieldFS [12] and reference [13] can identify ransomware by I/O request packets. EldeRan uses dynamic analysis to distinguish ransomware from goodware [14]. Some works [15–18] focus on encrypting ransomware detection by using traffic characteristics or sensitive APIs.

The methods of detecting ransomware on other platforms could not be directly applied on Android. On the one hand, detectors [15–18] use traffic to identify ransomware. This means detected ransomware should have network access, while most ransomware on Android can ransom without network access. On the other hand, Android has its own security mechanism, meaning that there are many different files and features that can be used for Android ransomware detection.

For Android, the approach for ransomware-oriented detection is incomplete. In 2016, N.Andronio et al. [19] first proposed a ransomware detector based on machine learning. To our best knowledge, HelDroid [19] and GreatEatlon [20] are the earliest ransomware-oriented detectors based on static analysis with machine learning. They detect ransomware based on threatening text detectors, lock detectors, and encryption detectors. If the ransomware uses unseen language, it may cause many misjudgments. The execution time is nearly seconds per sample on average [19]. There are also some detectors that use dynamic analysis to identify ransomware. DNA-Droid [21] combines static and dynamic analysis to detect ransomware. R-PackDroid [22] is a practical on-device detector of Android ransomware. Azmoodeh et al. [23] focus on files encryption ransomware in IoT and detect them by using energy consumption. If users need to detect large-scale samples by using detectors with dynamic analysis, it may be time consuming.

Many ransomware detectors identify ransomware based on sensitive APIs. However, there are some ransomware that use insensitive API callings to ransom. For example, a ransomware application can make its interface be the top-level interface suspending on the screen though users press Home buttons or Back buttons. Detectors may misjudge them as goodware behaviors. Some goodware applications that have the functions of locking devices and encrypting files have behaviors similar to ransomware. For example, some goodware applications such as time management applications lock the devices according to the time users have set. It is difficult for ransomware detectors to identity them.

Contributions. In the light of this, we made detailed analyses of three kinds of active ransomware, including the different runtime behaviors, ransom codes and the differences between ransomware and goodware with similar behaviors, for example, screen beautification applications with lock function and files management applications with an encryption function. Then, we constructed a multidimensional behavior pattern based on ransom behaviors. Finally, we proposed a behavior-based Android ransomware detector for mobile devices, called KRDroid. It retains the relational behavior patterns of ransomware. The main contributions of this paper are as follows.

The analyses of three kinds of active ransomware. We collected three kinds of active IoT Android ransomware from VirusTotal [24], AMD [25], and from open source databases [26]. According to their runtime behaviors, we sorted out ransomware into three groups: device lock ransomware, files encryption ransomware, and screen resource control ransomware. We analyzed them from multiple dimensions for their extortion behaviors and source code.

The construction of a ransomware-behavior-pattern-based multidimensional feature set. We extracted features from API callings, permissions, intents, and other dimensions to construct different kinds of ransom behavior patterns. In this way, the feature set can be seen as a formal expression set that retains the relational behaviors of ransomware.

A behavior-based ransomware-oriented detector. We proposed a behavior-based ransomware-oriented detector, KRDroid, to find Android ransomwares. KRDroid deploys on servers or PCs, that is, ransomware cannot be activated and cause any loss during testing. Experiments results show that KRDroid can detect unseen ransomware with the accuracy of 97.5%.

2. Related Research

With the increase of threats of ransomware, ransomware-oriented detectors for IoT devices have attracted more and more attention. In terms of related research, we mainly review the ransomware detectors based on I/O, dynamic analysis, and static analysis.

2.1. Ransomware Detection Based on I/O

Song et al. [23] proposed a method to detect ransomware using I/O rate, CPU usage, and memory usage. It discriminates between normal processes and ransomware by means of monitoring file events and computing resources. The method can protect users from the damage caused by ransomware applications without any information about ransomware codes.

Continella et al. [12] proposed ShieldFS, a ransomware detection file system. ShieldFS detects ransomware by means of the I/O usage and the change of IRP loggers (I/O request package logger). This method mainly detects files encryption ransomware, and it also can recover files that have already been encrypted by ransomware.

Feng et al. [13] proposed a method to detect files encryption ransomware based on deception and behavior monitoring. They created decoy files in the device at the very beginning to induct ransomware encrypting decoy files. In this way, abnormal processes can be detected.

Ko et al. [27] proposed a real-time ransomware detection with the help of intercepting requests from APIs to read or write to a file and judges whether the file is encrypted based on Shannon entropy.

In summary, ransomware detectors based on I/O usage are sensitive to files encryption ransomware used for encryption needs with much input and output file stream. Ransomware that lock devices or control screen resources may not applicable for these methods.

2.2. Ransomware Detection Based on Dynamic Analysis

Sgandurra et al. [14] proposed EldeRan, a ransomware-oriented detector based on dynamic analysis and machine learning classification. EldeRan focuses on the installation of applications to check for characteristics signs of ransomware by means of monitoring the selected APIs [14].

Abdullah et al. [28] proposed an Android ransomware detector based on dynamic analysis. It extracts system calls with the help of dynamic analysis and uses them as features. Algorithms such as Random Forest, J48, and Naïve Bayes are used to train the model.

Considering some ransomware may use complicated packing techniques, Chen et al. [29] proposed RansomProber, a real-time ransomware detection system with dynamic analysis. Instead of monitoring APIs, RansomProber uses information entropy to measure the degree of data transformation in sensitive directories [29]. To some extent, it can detect files encryption ransomware with customized cryptosystems.

Detectors with dynamic analysis can detect ransomware in real time. The analysis time of detectors for an application is approximately 5 seconds [29]. When detecting the large-scale samples, it will be time consuming.

2.3. Ransomware Detection Based on Static Analysis

Bibi et al. [30] proposed an effective Android ransomware detector. It extracts features from traffic with the help of 8 different feature filtration techniques and chosen 19 important features. Karimi et al. [31] proposed a method for Android ransomware detection based on transforming the sequence of executable instructions into a grayscale image and exploited valuable features by means of using LDA.

HelDroid [19] is a ransomware-oriented detector, which identifies ransomware by means of sensitive text based on NLP, lock-device function, and file-encrypt function based on FlowDroid [32]. According to the judge logic of the detector, a ransomware behavior must have ransom text. It requires the training corpus to be all-inclusive of the keywords of the ransom, as well as the language. When facing applications with unseen language, the detector will not identify the ransomware even if it has ransom behaviors. The execution time is nearly seconds per sample on average [19].

After approximately one year, some researchers improved HelDroid [19] and proposed GreatEatlon [20], a new ransomware-oriented detector. It extends FlowDroid [32] to track

encryption-related information flows to improve the encryption detector. It also adds a lightweight prefilter to filter goodware behaviors from the analysis queue to shorten execution time [20]. When facing ransomware with unseen language, the detector cannot identify ransomware either.

In order to detect ransomware with confusion, R-PackDroid [22] was proposed in 2018. Different from HelDroid [19] and GreatEatlon [20], it is designed as an application that can be installed on mobile phones. This detector uses static detection and extract API packages to represent the application and uses random forest for classification. R-PackDroid has the resilience of the related information against obfuscation [22]. Due to the detection mode of R-PackDroid [22] being "install–detect", large-scale samples detection may be time consuming.

3. Characterization of Ransomware

In order to have a better knowledge of ransomware, we collected 754 ransomware from the AMD dataset [25] and VirusTotal [24]. This section will analyze the characterization of different kinds of ransomware.

3.1. Analysis of Different Kinds of Ransomware

To our best knowledge, according to the behaviors, ransomware can be divided into three groups. The $\boldsymbol{R}$ represents the set of ransomware. As shown in formula (1), R contains three kinds of ransomware R_{DL}, R_{SRC}, and R_{FE}. R_{DL} represents device lock ransomware, which ransom users by automatically modifying the passwords of devices. R_{SRC} represents screen resource control ransomware, which ransom users by constantly holding the screen resource. R_{FE} represents files encryption ransomware, which ransom users by encrypting private files.

$$\boldsymbol{R} = \{\boldsymbol{R_{DL}}, \boldsymbol{R_{SRC}}, \boldsymbol{R_{FE}}\} \tag{1}$$

3.1.1. Device Lock Ransomware

Device lock ransomware behaviors are the most common and easy-to-implement ransomware. After they are activated, they can automatically modify the passwords, PINsm or gesture passwords. There were 461 device lock ransomware behaviors in the collected data, and we summarized 186 features of device lock ransomware.

A typical device lock ransomware can be represented as R_{DL}. As shown in formula (2), r_{dl} contains permission of *BIND_DEVICE_ADMIN*, typical API callings and sensitive strings. p_{dl} represents the permission of ransomware, such as *android.permission.BIND_DEVICE_ADMIN*. After applying this permission, a ransomware application can obtain super administrator rights. Android set the ransomware as the device manager to prevent being accidentally uninstalled. s_l represents sensitive or threaten strings in applications. A ransomware application usually uses threaten strings to call for payment.

The tuple $\langle A_{sub}, R_r \rangle$ represents API calling sequences. As shown in formula (3), R_r is the subset of $\{\varphi, \&, \|\}$. φ represents the relationship between each API is *none*, & represents the relationship between each API is *and* and $\|$ represents the relationship between each API is *or*. As shown in formula (4), A_{sub} is the subset of A_k and A_k represents the universal set of a_k. a_k represents APIs related to device lock. As shown in Table 1, *resetPassword()* is used to reset the password of the device, *resetView()* is used to reset the gesture view of the device, *setParameter(SpeechConstant.SAMPLE_RATE, "8000")* is used to set the voice password of the device, and *lockNow()* is used to lock the device. A device lock ransomware may first call *resetPassword()* to modify the password and then call *lockNow()* to lock the device. Both API callings are indispensable.

$$\boldsymbol{R_{DL}} = \{\boldsymbol{r_{dl}} = (\boldsymbol{p_{dl}}, \langle \boldsymbol{A_{sub}}, \boldsymbol{R_r} >, \boldsymbol{s_l}) \mid \boldsymbol{l} = \boldsymbol{1}, \ldots, \|\boldsymbol{s_l}\|\} \tag{2}$$

$$\boldsymbol{R_r} \subseteq \{\boldsymbol{\varphi}, \&, \|\} \tag{3}$$

$$\boldsymbol{A_{sub}} \subseteq \boldsymbol{A_k} = \{\boldsymbol{a_1}, \boldsymbol{a_2}, \ldots, \boldsymbol{a_k} \mid \boldsymbol{k} = \boldsymbol{1}, \ldots, \|\boldsymbol{a_k}\|\} \tag{4}$$

Table 1. Typical features of device lock ransomware applications.

Feature	Meaning
resetPassword()	Reset the password.
resetView()	Reset the gesture view.
setParameter(SpeechConstant.SAMPLE_RATE,"8000")	Set the voice password.
lockNow()	Lock the device.
android.permission.BIND_DEVICE_ADMIN	Apply and get super administrator rights.
Ladrt/R/ADRTLogCatReader	AIDE (IDE in Android) feature.

3.1.2. Files Encryption Ransomware

Files encryption ransomware behaviors are also a kind of common ransomware. After they are activated, ransomware applications automatically encrypt the privacy files on the device, including photos, txt files, etc. There were 223 files encryption ransomware behaviors in the collected data, and we summarized 411 features of files encryption ransomware.

A typical files encryption ransomware can be represented as R_{FE}. As shown in formula (5), r_{fe} contains permissions related to read or write, typical attack mode, and sensitive strings. p_{fe} represents related permissions such as *android.permission.WRITE_EXTERNAL_STORAGE*, which allows ransomware writing files on storage. s_l represents sensitive or threaten strings in applications.

att_k represents the attack mode of files encryption ransomware. As shown in formula (6), attack mode contains attack time, attack target, encryption method, attack order, and attack flow. The subset of att_k contains the typical API calling sequences. att_{time} represents the attack time. It includes encrypting files immediately and waiting for commands. att_{target} represents the encryption folder. att_{order} represents the attack order of the ransomware, i.e., the ransomware application encrypts files after obtaining the complete file list or encrypting each file when it is discovered by the ransomware. $att_{enmethod}$ represents the encryption method, including calling *AES()*, *DES()* or other methods. att_{fea} represents the attack flow of the ransomware. As shown in Table 2, the ransomware loops the storage structure of the device to find the target type of the files by *addCatefory()*and *createChooser()*; Once it finds the eligible files, it obtains data by calling *read()* or *FileInputStream()*, then calls encrypt API, such as *Ljava/crypto/spec/IvParameterSpec* to encrypt data; finally, it uses *write()* or *FileOutputStream()* to write the encrypted file in the storage.

$$R_{FE} = \{r_{fe} = (p_{fe}, att_k, s_l) \mid k = 1, \ldots, \|att_k\|, l = 1, \ldots, \|s_l\|\} \quad (5)$$

$$att_k =< att_{time}, a\ tt_{target}, a\ tt_{enmethod}, a\ tt_{order}, a\ tt_{fea} > \quad (6)$$

Table 2. Typical features of files encryption ransomware applications.

Feature	Meaning
android.permission.WRITE_EXTERNAL_STORAGE	Apply for reading and writing access to SDCard.
setCancelable()	Applying for creating and deleting file permissions in SDCard.
e Landroid/content/Intent->addCategory() Landroid/content/Intent->createChooser() Ljavax/crypto/Cipher->getInstance() Ljavax/crypto/Cipher-><init> Ljavax/crypto/Cipher->doFinal() Ljavax/crypto/spec/SecretKeySpec-><init>	Traversing and encrypting the specified file.

3.1.3. Screen Resource Control Ransomware

The screen resource control ransomware applications are uncommon ransomware. There were only 70 screen resource control ransomware behaviors in the collected data. After they are activated, there are 25 ransomware applications that make their interfaces as the top-level interfaces suspending on the top of devices and disable the Home and Back buttons. That is to say, other applications or other system functions cannot be used. Although another 45 ransomware applications make their interfaces the top-level interfaces, the user can press the Home and Back buttons to exit. However, this kind of exit is temporary, and the interface of the ransomware will suspend on the screen in a very short time to prevent users from using their phones normally. After analyzing these applications, we summarized 378 features of screen resource control ransomware.

A typical screen resource control ransomware can be represented as R_{SRC}. As shown in formula (7), r_{src} contains related permissions, intents, typical API callings, and sensitive strings. p_{src}, in_{src} represent the related permissions and intents. s_l represents sensitive or threaten strings in applications.

The tuple $\langle A_{sub}, R_r \rangle$ represents API calling sequences. As shown in formula (8) and formula (9), R_r is the subset of $\{\varphi, \&, \|\}$. A_{sub} is the subset of A_k and A_k represents the universal set of a_k. a_k represents APIs related to the screen resource control. As shown in Table 3, *LayoutParams->FLAG_FULLSCREEN* is used to suspend the interface as full screen. *setCancelable()* and *setFlags()* are used to suspend the interface as well, for the parameters of them have different meanings. Modifying the default parameter from *True* to *False* in *setCancelable()* means that users cannot press the external area of the dialog, using parameter *1024* in *setFlags()* means that the system window will be set as a full-screen window.

Table 3. Typical features of screen resource control ransomware applications.

Feature	Meaning
LayoutParams->FLAG_FULLSCREEN	Suspend the interface.
Window->setFlags(I,I),v2,v4,v4	Set the top-level window by modifying parameter.
android.intent.category.HOME onWindowFocusChanged() sendBroadcast()	Monitor the Home button and disable the Home button.
Dialog->setCancelable()	Set the current window cannot be cancelled or make it constant appearing.

OnkeyDown() and *OnAttachWindow()* are used to disable *Home* and *Back* buttons; for the *Home* button that is the system button, the KeyEvent barely captures the click events; thus, developers need to rewrite the *OnAttachWindow()*. If the version of Android is version 2.3 and below, the method can be rewritten similar to Listing 1. If the version of Android is version 4.0 and above, the method can be rewritten similar to Listing 2.

Listing 1. An example of *OnAttachWindow*.

```
1 public void onAttachedToWindow(){
2       this.getWindow().setType(WindowManager.LayoutParams.TYPE_KEYGUARD);
3       super.onAttachedToWindow();
4    }
```

If the version of Android is version 4.0 and above, the method can be rewritten similar to Listing 2.

Listing 2. An example of *OnAttachWindow.*

```
1 public static final int FLAG_HOMEKEY_DISPATCHED = 0x80000000;
2 public void onCreate(Bundle savedInstanceState){
3       super.onCreate(savedInstanceState);
4      this.getWindow().setFlags(FLAG_HOMEKEY_DISPATCHED,
           FLAG_HOMEKEY_DISPATCHED);
5       setContentView(R.layout.main);}
```

The *OnKeyDown()* will be rewritten similar to Listing 3.

Listing 3. An example of *OnKeyDown.*

```
1 public boolean onKeyDown(int keyCode, KeyEvent event){
2      if (keyCode == event.KEYCODE_HOME){
3         return true;
4      }
5      return super.o0nKeyDown(keyCode,event);
6    }
7 }
```

The API calling sequences shown in Listing 3 are used to disable the *Home* buttons. android.intent.category. *Home* is used to register the monitor of the *Home* button. *Landroid/app/Activity->onWindowFocusChanged()* is used to monitor whether the *Home* button is being clicked or not. *sendBroadcast()* is used to send the fake click broadcast. The button can be disabled by means of calling these API sequences.

$$R_{SRC} = \{r_{src} = (p_{src}, in_{src}, < A_{sub}, R_r >, s_l) \mid l = 1, \ldots, \|s_l\|\} \tag{7}$$

$$R_r \subseteq \{\varphi, \&, \|\} \tag{8}$$

$$A_{sub} \subseteq A_k = \{a_1, a_2, \ldots, a_k \mid k = 1, \ldots, \|a_k\|\} \tag{9}$$

3.2. Differences Between Ransomware and Goodware

In our research, we found that some ransomware and goodware applications have similar runtime behaviors. Some typical behaviors such as device lock and files encryption also exist in goodware applications. For example, as shown in Figure 1, screen beautification applications and time management applications have the function of locking devices. Furthermore, files management applications have the function of encrypting files.

Figure 1. Goodware applications have the function of locking devices.

We randomly selected 50 screen beautification applications, time management applications, and 50 files management applications from the internet [33] and uploaded them to VirusTotal [24]. The result showed that 10% of screen beautification and time management applications were misjudged as ransomware, and 19% of the files management applications were misjudged as ransomware. That is, the similar behaviors between the two may make detectors identify some goodware applications as ransomware.

In order to have a better knowledge of the differences between ransomware and goodware applications, we analyzed the differences between device lock ransomware, files encryption ransomware, and goodware applications.

3.2.1. Device Lock and Screen Resource Control Ransomware vs. Goodware Applications

As shown in Figure 2, though both ransomware and goodware applications apply the permission of *BIND_DEVICE_ADMIN* to obtain super administrator rights and use *lockNow()* to lock the device, there are some differences between them in runtime behaviors and source code.

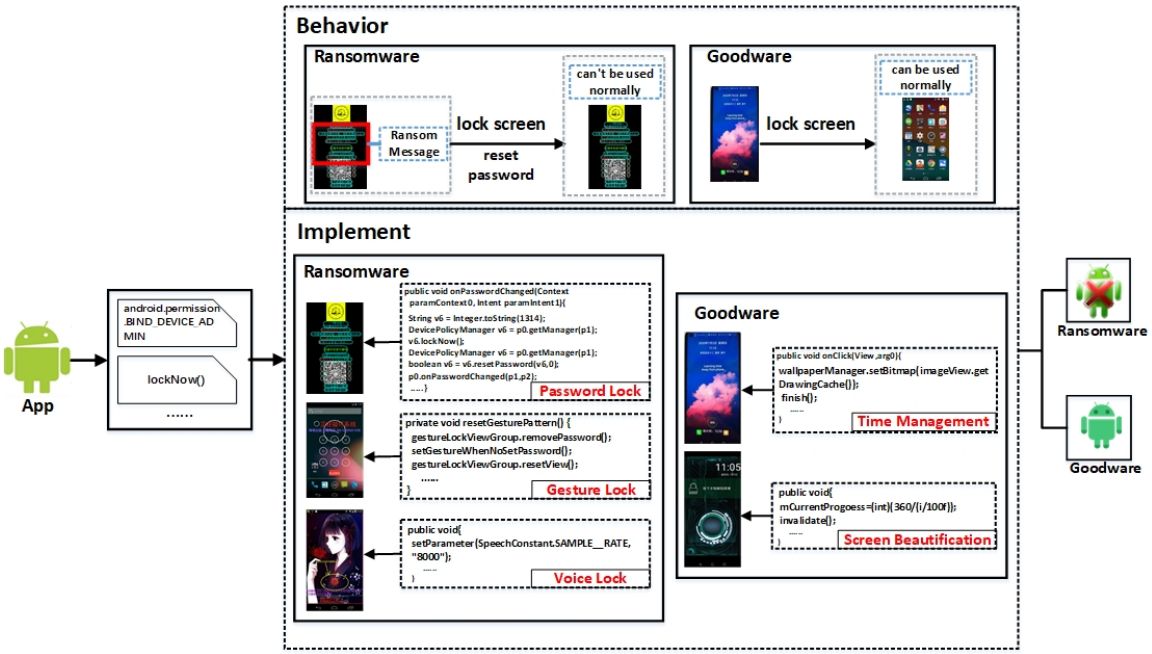

Figure 2. Differences between device lock ransomware and goodware applications.

As shown in formula (10) and formula (11), goodware applications with similar behaviors to ransomware can be represented as $G_{D\&S}$, $g_{D\&S}$ contains related permissions $p_{D\&S}$. and typical API callings a_k. The feature intersection of goodware applications and the union of device lock ransomware and screen resource control ransomware applications include *android.permission.BIND_DEVICE_ADMIN*, *lockNow()*, etc. For these goodware applications, only reset the wrappers or extend the device unlock time according to the settings of users. Though goodware applications monitor the Power Off buttons and lock the devices, they do not reset the PINs, gesture passwords, or voiceprints of the devices, that is, users can unlock their devices with their own passwords and use their devices normally.

$$G_{D\&S} = \{g_{D\&S} = (p_{D\&S}, a_k) \mid k = 1, \ldots, \|a_k\|\} \tag{10}$$

$$G_{D\&S} \cap (R_{DL} \cup R_{SRC}) = \{\text{LockNow(),permission.BIND\_DEVICE\_ADMIN,} \ldots\} \tag{11}$$

The device-locking ransomware applications lock the device and modify the original passwords. The device cannot be returned to the Home menu by clicking Home buttons or *Back* Buttons. When the user presses the *Power Off* button, it can be hibernated as normal. However, when the user tries to reset the device again, the device is still locked by the ransomware application. In this way, the user has to pay ransom to receive the correct password.

The screen resource control ransomware applications set their own activities as the top-level activities by setting particular parameters in the bytecode. The ransomware disables *Home* buttons and *Back* buttons, in addition to disabling *Power Off* buttons. In this way, the ransomware forces the device to constantly operate without being hibernated and forces users to pay ransom for the exit password. Some ransomware applications continue

to suspend the interfaces although the users click the *Home* or *Back* buttons. Moreover, some researchers also found some ransomware applications disable the USB of devices to prevent users from uninstalling the application by ADB commands. The detailed differences between device lock and screen resource ransomware and goodware applications are shown in Table 4.

Table 4. Differences between device lock and screen resource control ransomware and goodware applications.

	Ransomwares		Goodwares
	Device Lock	Screen Resource Control	
lock screen	✓ [1]	× [2]	✓
reset password	✓	×	×
top-level interface	×	- [3]	×
constant appear	×	-	×
disable Home Button	✓	-	×
disable Back Button	✓	-	×
disable USB interface	-	-	×
lockNow()	✓	×	✓
resetPassword()	✓	×	×
setCancelable()	×	-	×
Window->setFlags(I,I),v2,v4,v4	✓	✓	×
OnkeyDown()/OnkeyUp()	-	✓	×
onAttachedToWindow()	-	✓	×

1 ✓ means this kind of applications have the feature. 2 × means this kind of applications don't have the feature. 3 - means this kind of applications may have the feature.

3.2.2. Files Encryption Ransomware vs. Goodware Applications

As shown in Figure 3, both files encryption ransomware and files management applications can encrypt privacy files of devices, but there is still some differences between them in encrypt–decrypt mode.

Files management applications are a kind of privacy protection application. They give the users encryption options and wait orders to encrypt the customized files. These goodware applications show progress indicator bars to remind users of the current encryption progresses, and give corresponding prompts after the encryption operation is completed. Users can decrypt the files by the passwords they set.

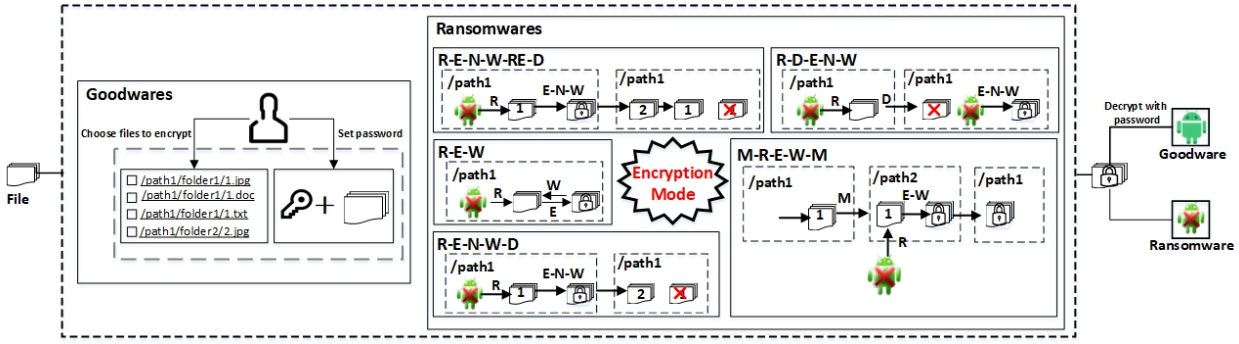

Figure 3. Differences between files encryption ransomware and goodware applications.

Ransomware applications first loop the target files and automatically encrypt these files in devices without any information. As shown in formula (12) and formula(13), E_{mode} represents the encryption mode of ransomware and e_i represents the encryption

process. *R* represents read operation, *E* represents encrypt operation, *W* represents write operation, *N* represents new operation, *D* represents delete operation, *RE* represents rename operation, and *M* represents move operation. In this paper, we mainly introduce five encryption modes.

$$E_{mode} = \{e_1, e_2, e_3, e_4, e_5\} \tag{12}$$

$$E_{mode} = \begin{bmatrix} & R & E & W & N & D & M & RE \\ e_1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ e_2 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ e_3 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ e_4 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ e_5 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{13}$$

e_1 represents the encryption mode that is reading files, encrypting data, and then writing them back to the original files.

e_2 represents the encryption mode that is reading files, encrypting data, creating new files, writing encrypted data to the new files, and deleting original files.

e_3 represents the encryption mode that is reading files, encrypting data, creating new files, writing encrypted data to the new files, renaming the new files, and deleting original files.

e_4 represents the encryption mode that is reading files, deleting original files, encrypting data, creating new files, and writing encrypted data to the new files.

e_5 represents the encryption mode that is moving original files to other folders, reading files, encrypting data, writing the encrypted data back to the original files, and moving the files back to the original location.

The detailed differences between files encryption ransomware and files management applications are shown in Table 5.

Table 5. Differences between files encryption ransomware and goodware applications.

	Files Encryption Ransomwares	Goodwares
Encrypt file	✓	✓
User can choose which file to encrypt	×	✓
Can recover encrypted with password user set	×	✓
Backstage encrypt files automatically	✓	×
Has target default encryption type of file	✓	×
EndecodeUtils.deCrypto()	×	✓
Landroid/content/Intent->addCategory Landroid/content/Intent->createChooser Ljavax/crypto/Cipher->getInstance Ljavax/crypto/Cipher-><init> Ljavax/crypto/Cipher->doFinal Ljavax/crypto/spec/SecretKeySpec-><init>	×	✓

4. A Ransomware-Oriented Detector

In this section, we introduce a ransomware-behavior-pattern-based, multidimensional, ransomware-oriented detection approach for mobile devices. It uses static analysis to analyze the source code and extract features based on behavior patterns; it also uses the form of binary feature to represent the feature information of samples and XGBoost to classify samples.

4.1. Workflow

The detailed workflow of the ransomware-oriented detector is shown in Figure 4.

When an application needs to be tested, the AndroidManifest.xml and classes.dex are first extracted from the apk file. Second, Androguard [34], a static analysis tool, is used to extract features. Then, features are divided into two parts. For the features that do

not need to be counted for their frequency, we use **1** to represent their existence and **0** to represent the opposite. Next, all the features are combined to form the feature vectors and use XGBoost to classify them. Lastly, the detector outputs the results of the detection.

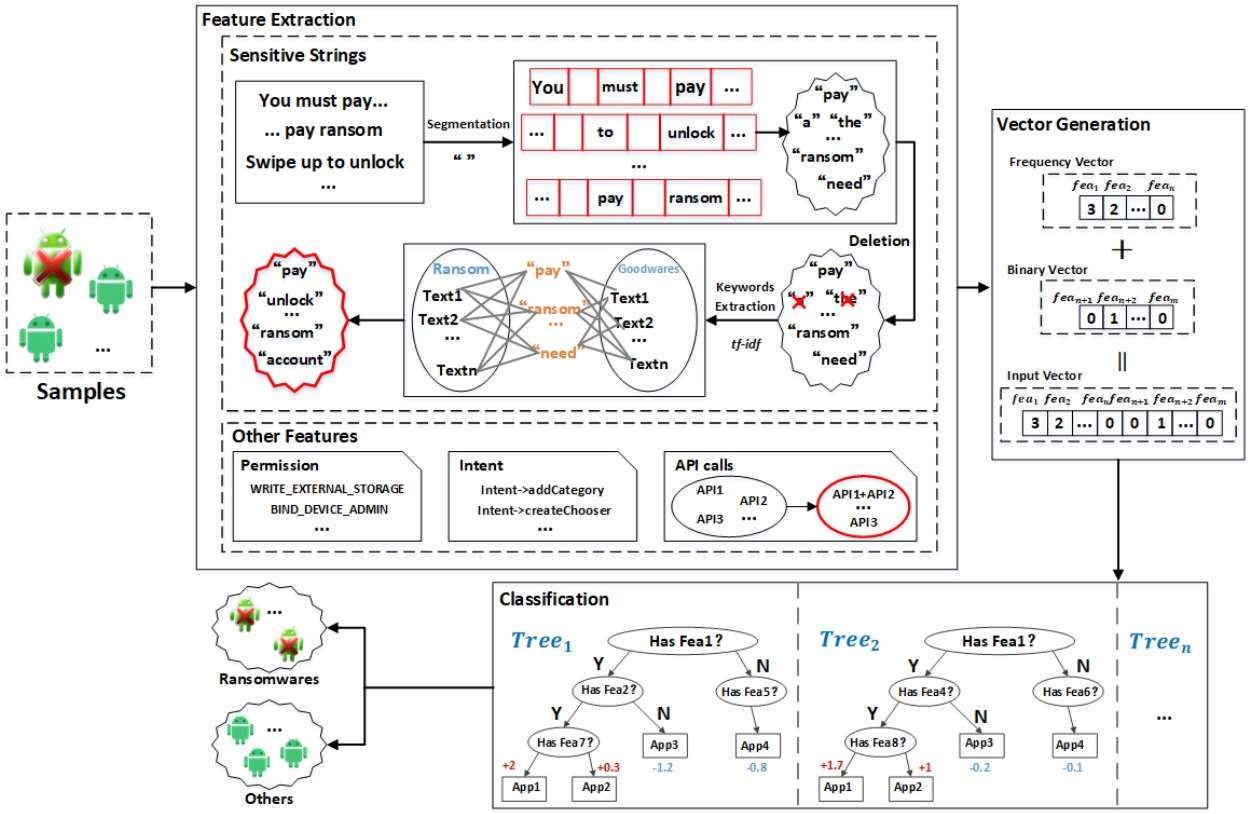

Figure 4. Workflow of the behavior-based, ransomware-oriented detector.

4.2. Feature Extraction

With the help of Androguard [34], a tool that can read the binary format of Android XML files(AXML) and decompile DEXfiles [35], we extracted features from AndroidManifest.xml and classes.dex. The feature set contains sensitive strings set and other features set.

Sensitive Strings Set. The sensitive strings mentioned in this paper mean constant strings declared in the Dalvik bytecode. In order to better distinguish ransomware from other applications, we segmented the constant strings based on the word segmentation method in NLP. As shown in Algorithm 1, the steps of building sensitive strings set are as follows.

(1) **Segmentation**. We used special characters such as " " for the baseline of the segmentation. T_r represents the text set of ransomware after segmentation, and T_o represents the text set of other applications after segmentation.

(2) **Deletion**. We removed some meaningless words from T_r and T_o. The meaningless words include stop words such as *a*, *the*, and some obvious common words. We used T'_r to represent the ransom text set after deletion and used T'_o to represent other text sets.

(3) **Keywords Extraction**. We used *tf-idf* to calculate the weight of each word in T'_r and T'_o. The result of *tf-idf* refers to whether the word has the discrimination between ransomware and other applications. The weight can be expressed similar to formula (14). The $t_{i,j}$ represents the number of the word t appears in T'_r and in T'_o. The $\sum_i t'_{ri} + \sum_j t'_{oj}$ represents the total words in both T'_r and T'_o. The $\sum_i \text{label}_{ri}$ represents the number of

ransomware, and the $\sum_j \text{label}_{oj}$ represents the number of other applications. The $\sum_{i,j} \text{label}_{t_{ij}}$ represents the number of applications containing the word $\boldsymbol{t}$.

$$\boldsymbol{weight} = \frac{\boldsymbol{t_{i,j}}}{\sum_i \boldsymbol{t_{ri}} + \sum_j \boldsymbol{t_{oj}}} \boldsymbol{\log}\left(\frac{\sum_i \boldsymbol{label_{ri}} + \sum_j \boldsymbol{label_{oj}}}{\sum_{i,j} \boldsymbol{label_{t_i}} + \boldsymbol{1}}\right) \tag{14}$$

Algorithm 1 The algorithm of building sensitive strings set

Input: apks, label
Output: S
1: $T_r \leftarrow$ **Segment**$(apks, label_r)$
2: $T_o \leftarrow$ **Segment**$(apks, label_o)$
3: $T_r' \leftarrow$ **Deletion**$(T_r, meaningless\_word)$
4: $T_o' \leftarrow$ **Deletion**$(T_o, meaningless\_word)$
5: **for** $t \in T_r' \cup T_o'$ **do**
6: $\quad weight \leftarrow \frac{t_{i,j}}{\sum_i t_{ri} + \sum_j t_{oj}} \log\left(\frac{\sum_i label_{ri} + \sum_j label_{oj}}{\sum_{i,j} label_{t_i} + 1}\right)$
7: $\quad$ **if** $weight > threshold$ **then**
8: $\quad\quad S \leftarrow S \cup t$
9: $\quad$ **end if**
10: **end for**
return S

Other Features Set. The algorithm of building other features set is shown in Algorithm 2. The other features set can be represented as set F. As shown in formula (15), f_m contains permissions, intents, API callings, and sensitive strings. The p_i represents permissions, a kind of the security model of Android. Permissions need to be declared before calling sensitive APIs. The in_i represents intents, the runtime binding mechanism of Android. Intents are responsible for internal communication. The s_l represents sensitive strings related to ransom, which we obtained based on *tf-idf*.

Algorithm 2 The algorithm of building feature set

Input: apks
Output: F
1: $S =$ **Sensitive Strings Set Aggregation**$(apks, label)$
2: **for** *each apk* **do**
3: $\quad F \leftarrow$ **Extract**(p_i, in_j, a_k, s_l)
4: $\quad < A_{sub}, R_r > \leftarrow$ **Extract**$(a_i, \ldots, a_j, \varphi, \&, \|)$
5: $\quad$ **if** $F = \varphi, < A_{sub}, R_r > = \varphi$ **then**
6: $\quad\quad$ ***continue***
7: $\quad$ **else**
8: $\quad\quad F \cup < A_{sub}, R_r >$
9: $\quad$ **end if**
10: **end for**
return F

As shown in formula (16) and formula (17), R_r is the subset of $\{\varphi, \&, \|\}$. A_{sub} is the subset of A_k, and A_k represents the universal set of a_k. The a_k represents API callings, which provide certain functions for developers to access a set of routines based on Android. Developers can use different API calling sequences to implement different functions.

$$\boldsymbol{F} = \left\{ \begin{array}{l} \boldsymbol{f_m} = (\boldsymbol{p_i}, \boldsymbol{in_j}, < \boldsymbol{A_{sub}}, \boldsymbol{R_r} >, \boldsymbol{s_l}) \mid \\ \quad \boldsymbol{i = 1, \ldots, \|p_i\|, j = 1, \ldots, \|in_j\|, l = 1, \ldots, \|s_l\|} \end{array} \right\} \tag{15}$$

$$\boldsymbol{R_r} \subseteq \{\boldsymbol{\varphi}, \boldsymbol{\&}, \|\} \tag{16}$$

$$\boldsymbol{A_{sub}} \subseteq \boldsymbol{A_k} = \{\boldsymbol{a_1}, \boldsymbol{a_2}, \ldots, \boldsymbol{a_k} \mid \boldsymbol{k = 1, \ldots, \|a_k\|}\} \tag{17}$$

4.3. Classification

In this paper, we transfer the extracted features to vectors. As shown in formulas (18) and (19), *Vec* represents the vector set, containing a binary vector set and a value vector set. Vec_{value} represents the value vector set. The value of each dimension of the vector is float. Vec_{binary} represents the binary vector set. The value of each dimension of the vector is int. If Vec_i exists in the feature set, no matter how many times it appears in the application, the value of Vec_i is **1**. Otherwise, the value of Vec_i is **0**.

$$\boldsymbol{Vec} = \boldsymbol{Vec_{binary}} \cup \boldsymbol{Vec_{value}} \tag{18}$$

$$\boldsymbol{Vec}_{\text{binary}} = \left\{ \boldsymbol{vec_i} = \begin{cases} \mathbf{0}, \boldsymbol{notinfeatureset} \\ \mathbf{1}, \boldsymbol{inthefeatureset} \end{cases} \mid \boldsymbol{i} = \mathbf{1}, \ldots, \|\boldsymbol{vec_i}\| \right\} \tag{19}$$

Next, we combined the two groups of features as a whole vector, which represents the information of the application. Then, we used XGBoost, a supervised approach, to train the ransomware-oriented detector. We divided the ransomware and goodware applications into two parts, randomly used 80 percent of them to train, and used 20 percent of them to test.

5. Evaluation

We conducted three experiments to evaluate its detection capability and efficiency. To test the detection performance of KRDroid, we first evaluated it on a dataset with ransomware and other samples. Then, we compared the ransomware detection capability with HelDroid [19], a well-known ransomware detector and R-PackDroid [22], an on-device ransomware detector.

5.1. Dataset

$\boldsymbol{D}$ represents the dataset we used in our experiment. As shown in formula (20), $\boldsymbol{D}$ contains three datasets, $\boldsymbol{D_1}$, $\boldsymbol{D_2}$, and $\boldsymbol{D_3}$. $\boldsymbol{D_1}$ contains 1862 different kinds of ransomware in the period of 2014–2021 collected from reference [24–26,36,37], including *Koler*, *Locker*, *PronDroid*, *Simplocker*, *Svpeng*, *Congur*, *Fusob*, *Jisut*, *Pigetrl*, *Rkor*, *Piom*, and other types of ransomware. As shown in Figure 5, $\boldsymbol{D_1}$ contains 425 ransomware applications in the period of 2014–2015, 767 ransomware applications in the period of 2015–2016, 240 ransomware applications in the period of 2017–2018, and 430 latest ransomware applications in the period of 2021.1–2021.6. We used $\boldsymbol{D_1}$ to test the capability of KRDroid and evaluate whether KRDroid can still identify unseen ransomware when facing the latest samples.

$\boldsymbol{D_2}$ contains 1000 different kinds of malware (except ransomware), including *Smsreg*, a malware family that makes users register to premium services unknowingly, *Windadware*, an adware family that delivers adwares to devices, *Emial*, a malware family that monitors SMS messages on devices, *Agentspy*, a malware family that steals privacy information on devices, *DroidKungFu*, a kind of remote command and control (C&C) servers Trojans and other types of malware. We used $\boldsymbol{D_2}$ to evaluate whether KRDroid misjudges malware as ransomware.

$\boldsymbol{D_3}$ contains 1697 goodware applications, including screen beautification applications, files management applications, and other goodware applications. We used $\boldsymbol{D_3}$ to evaluate whether KRDroid misjudges goodware applications as ransomware or misjudges ransomware as goodware applications.

$$\boldsymbol{D} = \{\boldsymbol{D_1}, \boldsymbol{D_2}, \boldsymbol{D_3}\} \tag{20}$$

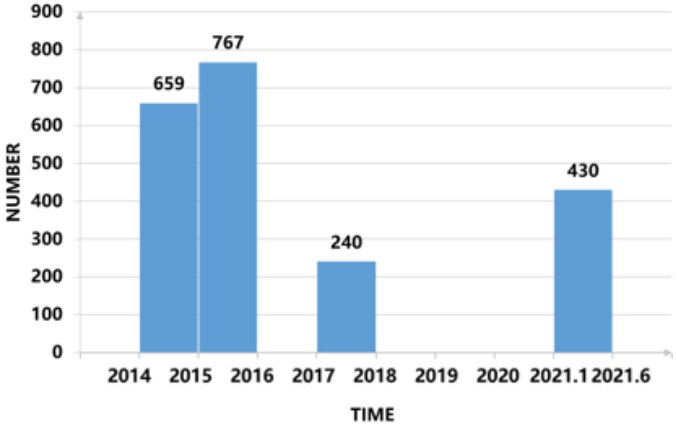

Figure 5. The composition of dataset.

5.2. Evaluation Metrics

In order to give a better evaluation of experiment results, we calculated accuracy, precision, recall, F1-score, false-positive rate, and false-negative rate for ransomware-oriented detector. As shown in formula (21)–(26), accuracy represents the total number of correct ransomware and other applications divided by the total number of classifications. Precision represents the accuracy of the detector in terms of data. The recall represents the sensitivity of the detector. F1-score represents the combination of precision and recall. False-positive rate represents the rate by which the detector misjudges negative ones as positive ones. False-negative rate represents the rate by which the detector misjudges positive ones as negative ones. In formula (21)–(26), the following are included:

(1) TP: The number of true positives, which means the classification of the detector is correct, and the application is ransomware;

(2)FP: The number of false positives, which means the classification of the detector is incorrect, and the application is not ransomware;

(3) FN: The number of false negatives, which means the classification of the detector is incorrect, and the application is ransomware;

(4) TN: The number of true negatives, which means the classification of the detector is correct, and the application is not ransomware.

$$\textbf{accuracy} = \frac{\boldsymbol{TP + TN}}{\boldsymbol{TP + TN + FP + FN}} \tag{21}$$

$$\textbf{precision} = \frac{\boldsymbol{TP}}{\boldsymbol{TP + FP}} \tag{22}$$

$$\textbf{recall} = \frac{\boldsymbol{TP}}{\boldsymbol{TP + FN}} \tag{23}$$

$$\boldsymbol{F1score} = \mathbf{2} \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{24}$$

$$\text{false positive rate} = \frac{FP}{TN + FP} \tag{25}$$

$$\text{false negative rate} = \frac{FN}{TP + FN} \tag{26}$$

5.3. Experiments

In this work, we will answer the following three questions to evaluate the detection performance of KRDroid. For each question, we first describe an experiment and give the corresponding results. Then, we provide a brief insight to summarize. The training dataset of all the experiments is the same.

We used 1526 ransomware in the period of 2014–2015 from reference [36], including *Koler*, *Locker*, *PronDroid*, *Simplocker*, *Svpeng*, and unlabeled ransomware applications as positive samples to train KRDroid. We used 400 malware and 1200 goodware applications in the period of 2014–2015 as negative samples to train KRDroid; in KRDroid, the issue is

not to only distinguish ransomware from malware applications but rather to distinguish ransomware from goodware applications.

In addition, we compared the MD5 of each sample in the test dataset with the training dataset before we started experiments to make sure that all the samples in the test dataset of the following experiments are different from samples used for training.

***Q*1**: What is the ransomware detection capability of KRDroid?
***Q*2**: Will KRDroid misjudge other malware applications as ransomware?
***Q*3**: Is the efficiency of KRDroid acceptable?

5.3.1. RQ1: What Is the Detection Effect of KRDroid?

In this experiment, we took D_1 and D_3 as the dataset for testing. The test dataset contains 1862 ransomware and 1697 goodware applications from reference [24–26,36–38]. In order to better evaluate the capability of KRDroid, we compared KRDroid with two ransomware-oriented detectors, HelDroid [36] and R-PackDroid [38]. HelDroid is a well-known ransomware-oriented detector, and we reproduced HelDroid from reference [36]. R-PackDroid is an on-device Android ransomware-oriented detector, and it can be download from reference [38]. The detailed result of this experiment is shown in Table 6.

Table 6. The comparison results of KRDroid, R-PackDroid, and HELDORID.

Model	Positive(TP+TN)		Negative(FP+FN)	Accuracy	Precision	Recall	F1-Score
	Ransomware	Goodware					
HelDroid	1558	1397	604	83.03%	83.67%	83.85%	83.76%
R-PackDroid	1692	1613	254	92.86%	90.87%	95.27%	93.02%
KRDroid	1809	1665	95	97.33%	97.15%	97.73%	97.44%

HelDroid correctly identified 1558 ransomware and 1397 goodware applications. The accuracy of HelDroid is 83.03%. R-PackDroid correctly identified 1692 ransomware and identified 1613 goodware applications. The accuracy of R-PackDroid is 92.86%. KRDroid correctly identified 1809 ransomware and 1655 goodware applications. The accuracy of KRDroid is 97.33%. The precision, recall, and F1-score of KRDroid are also higher than the two detectors.

We randomly sampled 46 true negatives and further analyzed the result of HelDroid. After the real machine test and decompile analysis, we found that there were 28 samples in 46 true negatives cannot be detected because of the unseen languages. Nine ransomware applications in the rest of the true negatives cannot be detected because of the unsuccessful lock detection. All of these samples had already been detected as sensitive text. In addition, we found that there were four samples in these nine ransomware applications that belong to screen resource control ransomware. As we mentioned before, this kind of ransomware does not need some real *lock APIs* such as *lockNow()* to reach their goals. The last nine ransomware applications that are misjudged are true negatives, which had not been detected.

The goal of R-PackDroid is to use a compact set of information more than enough to detect a wide variety of samples [22]. When building the detector, it uses the system API package list to represent the application rather than building multidimensional attack-pattern-based features. To some extent, it may cause some misjudgments because of the lack of effective information.

In addition, as is aforementioned, ransomware in D_1 is in the period of 2014–2021.6. KRDroid has good performance on identifying unseen ransomware in this experiment, which means that KRDroid is still valid when facing the latest samples in 2021.

Insight. Due to the accurate characterization and comprehensive behavior-based features build of ransomware applications, KRDroid can detect ransomware by analyzing source code. It detects ransomware by means of detecting ransom behaviors. In this way, national languages requirements do not need to be taken into consideration during detection.

KRDoid has good generalization. It can identify unseen ransomware similar to the training samples and can also identify unseen ransomware applications after they have already evolved. To some extent, it can also show that our analysis and behavior-based feature extraction of ransomware applications is valuable.

5.3.2. RQ2: Will KRDroid misjudge other malware applications as ransomware?

Since a ransomware application is a kind of malware, we still need to test that the accuracy of KRDroid is independent of malware classification. We randomly sampled 1000 ransomware in D_1 and randomly sampled 1000 goodware in D_3. These samples are collected from reference [24,26]. We used these samples and 1000 malware in D_2 as the dataset for test in this experiment. As mentioned above, all the applications in D_2 are malware, which is different from ransomware.

As shown in Figure 6, we found that there are 981 samples that can be correctly identified as ransomware, and only 19 ransomware misjudged as non-ransomware. There are 1986 samples that can be correctly identified as non-ransomware, and only 14 non-ransomware misjudged as ransomware. The false-positive rate of KRDroid is 1.94%, and the false negative rate is 0.7%.

Actual / Predict	Ransomware	Non-Ransomware (malware+goodware)
Ransomware	**981**	**14**
Non-Ransomware (malware+goodware)	**19**	**1986**

Figure 6. The confusion matrix of experiment 2.

Insight. KRDroid is a ransomware-oriented detector rather than a malware detector. It does not misjudge other malware applications as ransomware because other malware applications do not have typical ransom behaviors.

5.3.3. Is the Efficiency of KRDroid Acceptable?

We measured the efficiency of KRDroid on 450 samples collected from Virustotal [24]. Meanwhile, we used the same test dataset to test the HelDroid. Because R-PackDroid is an Android on-device detector, we did not take it into consideration. We assessed the execution time of HelDroid and KRDroid by running it on six cores of a MacBook Pro laptop containing an Intel Core i7 CPU 0.6 GHz processor.

The execution time of HelDroid was nearly 4 h 30 min, and the main bottleneck is the locking strategies detection [19]. The average CPU usage of Heldroid is nearly 90%, and memory usage is 18%. The execution time of KRDroid was nearly 5 s. The CPU usage of KRDroid is 1.6%, and the memory usage is less than 1%.

Insight. The efficiency of KRDroid is acceptable for detecting large-scale applications. It can detect a number of applications with fewer resources.

6. Limitations and Future Work

KRDroid is an Android ransomware-oriented detector that deploys on servers or PCs. KRDroid detects ransomware applications based on behavior patterns with the help of static analysis. Though KRDroid can identify most ransomware applications with less time and high accuracy, and it can identify ransomware even if evolved, there is still some ransomware applications that may be misjudged. Because these ransomware applications are implemented with the help of obfuscation, steganography, reflection, and reinforcement as goodware for these methods can prevent applications from being totally decompiled

and KRDroid could not obtain some core codes of ransomware. In the future, we will pay more attention to the detection of ransomware with code protection methods with the help of dynamic analysis. In addition, our research only focused on ransomware applications on Android. In the future, we will also turn our attention to the ransomware appoications in other platforms.

In addition, how to stop or prevent ransomware on Android devices is very essential for users. In our future work, we will pay our attention to on-device ransomware detectors and real-time files and devices protection against ransomware on Android devices.

7. Conclusions

In this paper, we made a detailed analysis of three kinds of active ransomware applications for mobile devices, including the different runtime behaviors and ransom code. To ensure the extracted features have discrimination, we made a comparative analysis to find out the differences between ransomware and goodware applications with similar behaviors. Then, we proposed a ransomware-oriented detector with a behavior-pattern-based multidimensional feature set. The detection can successfully identify more ransomware applications and can also distinguish ransomware from goodware with similar behaviors. It has a low false-positive rate and takes less time for detection.

Author Contributions: Conceptualization, S.W.; Data curation, S.W. and Z.J.; Funding acquisition, J.G.; Investigation, T.T.; Methodology, S.Q.; Project administration, J.Q.; Resources, H.Z.; Software, S.W.; Writing–original draft, S.W.; Writing—review and editing, S.W. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the National Key R&D Program of China under Grant No. 2018YFB0804703.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: This work was supported in part by the National Key R&D Program of China under Grant No. 2018YFB0804703.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. McAfee Labs 2017 Threats Predictions. 2017. Available online: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-threats-predictions-2017.pdf (accessed on 12 July 2019).
2. Available online: https://www.coveware.com/blog/q2-2020-ransomware-marketplace-report (accessed on 1 December 2020).
3. Fake Super Mario Run App Steals Credit Card Information. 2017. Available online: https://blog.trendmicro.com/trendlabs-security-intelligence/fake-super-mario-run-app-steals-credit-card-information/ (accessed on 21 April 2017).
4. McAfee Labs Threats Report. 2019. Available online: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-mobile-threat-report-2019.pdf (accessed on 5 April 2020).
5. Avast Highlights the Threat Landscape for 2019. 2019. Available online: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-dec-2018.pdf (accessed on 5 April 2020).
6. Wu, P.; Liu, D.; Wang, J.; Yuan, B.; Kuang, W. Detection of Fake IoT App Based on Multidimensional Similarity. *IEEE Internet Things J.* **2020**, *7*, 7021–7031. [CrossRef]
7. Fake Alexa Setup App Is Topping Apple's App Store Charts. 2018. Available online: https://www.engadget.com/2018/12/27/fake-alexa-app-topping-apple-app-store-charts/ (accessed on 2 March 2019).
8. Scam iOS Apps Promise Fitness, Steal MONEY instead. 2018. Available online: https://www.welivesecurity.com/2018/12/03/scam-ios-apps-promise-fitness-steal-money-instead/ (accessed on 3 March 2019).
9. Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, up 31 Percent from 2016. 2017. Available online: Https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016 (accessed on 9 July 2017).
10. Mohammad Mehdi, A.; Shahriari, H.R. 2entFOX: A framework for high survivable ransomwares detection. In Proceedings of the 2016 13th International Iranian Society of Cryptology Conference on Information Security and Cryptology (ISCISC), Tehran, Iran, 7–8 September 2016.

11. Kharraz, A.; Arshad, S.; Mulliner, C.; Robertson, W.; Kirda, E. UNVEIL: A Large-Scale, Automated Approach to Detecting Ransomware. *Usenix Secur. Symp.* **2016**, *16*, 757–772.
12. Continella, A.; Guagnelli, A.; Zingaro, G.; De Pasquale, G.; Barenghi, A.; Zanero, S.; Maggi, F. ShieldFS: A self-healing, ransomware-aware filesystem. In Proceedings of the 32nd Annual Conference ACM, Los Angeles, CA, USA, 5–8 December 2016.
13. Song, S.; Bongjoon, K.; Sangjun, L. The Effective Ransomware Prevention Technique Using Process Monitoring on Android Platform. *Mob. Inf. Syst.* **2016**, *2016*, 1–9. [CrossRef]
14. Sgandurra, D.; Munoz-Gonzalez, L.; Mohsen, R.; Lupu, E.C. Automated Dynamic Analysis of Ransomware: Benefits, Limitations and use for Detection. *arXiv* **2016**, arXiv:1609.03020.
15. Aurélien, P.; Le Bouder, H.; Lanet, J.L.; Le Guernic, C.; Legay, A. Ransomware and the Legacy Crypto API. In Proceedings of the International Conference on Risks and Security of Internet and Systems 2017, Dinard, France, 19–21 September 2017.
16. Moore, C. Detecting Ransomware with Honeypot Techniques. In Proceedings of the Cybersecurity & Cyberforensics Conference IEEE, Amman, Jordan, 2–4 August 2016
17. Cabaj, K.; Mazurczyk, W. Using Software-Defined Networking for Ransomware Mitigation: The Case of CryptoWall. *IEEE Netw.* **2016**, *30*, 14–20. [CrossRef]
18. Manabu, H.; Kobayashi, R. Machine Learning Based Ransomware Detection Using Storage Access Patterns Obtained From Live-forensic Hypervisor. In Proceedings of the 2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS) IEEE, Granada, Spain, 22–25 October 2019.
19. Andronio, N.; Zanero, S.; Maggi, F. Heldroid: Dissecting and detecting mobile ransomware. In *Recent Advances in Intrusion Detection (RAID)*; Springer: Berlin/Heidelberg, Germany, 2015; pp. 382–404.
20. Zheng, C.; Dellarocca, N.; Andronio, N.; Zanero, S.; Maggi, F. Greateatlon: Fast, static detection of mobile ransomware. In *SecureComm, volume 198 of Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 617–636.
21. Gharib, A.; Ghorbani, A. DNA-Droid: A real-time android ransomware detection framework. In *NSS 2017*; LNCS; Yan, Z., Molva, R., Mazurczyk, W., Kantola, R., Eds.; Springer: Cham, The Natherland, 2017; Volume 10394, pp. 184–198.
22. Michele, S.; Davide, M.; Francesco, M.; Corrado, V.A.; Fabio, M.; Giorgio, G. R-PackDroid: Practical On-Device Detection of Android Ransomware. In *SAC 2017*; ACM: Marrakech, Morocco, 2017.
23. Azmoodeh, A.; Dehghantanha, A.; Conti, M.; Choo, K.K.R. Detecting crypto-ransomware in IoT networks based on energy consumption footprint. *J. Ambient. Intell. Humaniz. Comput.* **2018**, *9*, 1141–1152. [CrossRef]
24. Available online: https://www.virustotal.com/gui/contact-us/technical-support (accessed on 2 August 2019).
25. Available online: http://amd.arguslab.org (accessed on 20 December 2020).
26. Available online: https://koodous.com/ (accessed on 20 December 2020).
27. Ko, J.; Jo, J.; Kim, D.; Choi, S.; Kwak, J. Real Time Android Ransomware Detection by Analyzed Android Applications. In Proceedings of the 2019 International Conference on Electronics, Information, and Communication (ICEIC), Auckland, New Zealand, 22–25 January 2019; pp. 1–5. [CrossRef]
28. Abdullah, Z.; Muhadi, F.W.; Saudi, M.M.; Hamid, I.R.A.; Foozy, C.F.M. Android Ransomware Detection Based on Dynamic Obtained Features. In *Recent Advances on Soft Computing and Data Mining*; SCDM 2020; Advances in Intelligent Systems and Computing; Ghazali, R., Nawi, N., Deris, M., Abawajy, J., Eds.; Springer: Cham, The Natherlaand, 2020; Volume 978.
29. Chen, J.; Wang, C.; Zhao, Z.; Chen, K.; Du, R.; Ahn, G.J. Uncovering the Face of Android Ransomware: Characterization and Real-time Detection. IEEE *Trans. Inf. Forensics Secur.* **2017**, *13*, 1286–1300. [CrossRef]
30. Bibi, I.; Akhunzada, A.; Malik, J.; Ahmed, G.; Raza, M. An Effective Android Ransomware Detection Through Multi-Factor Feature Filtration and Recurrent Neural Network. In Proceedings of the 2019 UK/ China Emerging Technologies (UCET), Glasgow, UK, 21–22 August 2019; pp. 1–4. [CrossRef]
31. Karimi, A.; Moattar, M.H. Android ransomware detection using reduced opcode sequence and image similarity. In Proceedings of the 2017 7th International Conference on Computer and Knowledge Engineering (ICCKE), Mashhad, Iran, 26–27 October 2017; pp. 229–234. [CrossRef]
32. Available online: https://blogs.uni-paderborn.de/sse/tools/flowdroid/ (accessed on 3 February 2020).
33. Available online: https://developer.android.google.cn/ (accessed on 3 February 2020).
34. Available online: https://github.com/androguard/androguard (accessed on 11 December 2019).
35. Available online: https://developer.android.google.cn/reference/dalvik/system/DexFile (accessed on 3 February 2020).
36. Available online: https://github.com/necst/heldroidlynomials (accessed on 5 August 2020)
37. Available online: https://appstore.anva.org.cn/homePage/webinfoCommonList/1 (accessed on 30 June 2021).
38. Available online: http://prag.diee.unica.it/it/RPackDroid (accessed on 1 January 2020).