Ransomware as a Service: Demystifying Android Ransomware Generators

Can Tu[1], Liu Wang[2], Yang Xu[3], Yiping Zhao[2], Haitao Xu[4], and Haoyu Wang[1(✉)]

[1] Huazhong University of Science and Technology, Wuhan, China
haoyuwang@hust.edu.cn
[2] Beijing University of Posts and Telecommunications, Beijing, China
[3] China Mobile Information Security Center, Hong Kong, China
[4] Zhejiang University, Hangzhou, China

Abstract. Ransomware has become a pervasive and lucrative threat in the Android platform, prompting the emergence of Ransomware as a Service (RaaS) business model. Ransomware generators, as an outgrowth of this model, have been found to be readily available on the web. This has further fueled the proliferation of ransomware attacks by enabling individuals without programming skills to participate in the ransomware economy. Although the nuisance of ransomware generators has been mentioned by a few security reports, our community lacks an understanding of the characteristics of these Android ransomware generators. In this paper, we take the first step towards systematically studying Android ransomware generators. We analyze the RaaS business model from multiple perspectives including their behaviors, practices, generated apps, and ecosystem. We observe that deceptive tactics exist in some so-called ransomware generator apps, such as malware masquerading and developer spoofing. For the generated ransomware, we reveal their common locking mechanisms and a variety of unlocking mechanisms. We also provide an overview of the ecosystem by revealing the participating entities, propagation channels, and workflow. Our findings contribute to advancing our understanding of Android ransomware generators and their associated risks, and inform the development of effective countermeasures and strategies to combat ransomware threats.

Keywords: Ransomware · Ransomware generator · Ransomware as a Service · Android malware

1 Introduction

Android dominates the smartphone operating system market with a share of 71.8% in Q4 of 2022 [8]. However, Android's open-source nature makes it a popular target of malware and virus attacks [6]. Remarkably, as a type of malware

The full name of the affiliation is Hubei Key Laboratory of Distributed System Security, Hubei Engineering Research Center on Big Data Security, School of Cyber Science and Engineering, Huazhong University of Science and Technology.

H. Duan et al. (Eds.): SecureComm 2023, LNICST 568, pp. 426–447, 2025.
https://doi.org/10.1007/978-3-031-64954-7_22

that criminals use to extort victims to pay a ransom or risk losing access to their data, ransomware is particularly notorious in this regard, with an average of over 4,000 mobile ransomware attacks per month [12]. In 2016, ransomware attacks were at a rate of one every 40 s [7], while in 2022, ransomware attacks occur once every 11 s [2]. There even could be a ransomware attack every two seconds [11]. While ransomware attacks initially targeted individuals, large groups, such as financial services and governments, have become frequent targets in the past five years [18]. Ransomware attacks cause huge financial losses every year. In addition to ransom payments, service downtime and data leakage pose even greater damages. It is estimated that the costs associated with ransomware hacks are $7.5 billion in 2019 [10], and will reach $265 billion by 2031 [1].

Massive ransomware profits lead to the emergence of *Ransomware as a Service (RaaS)*. RaaS is a cybercrime business model in which a ransomware group sells its ransomware code to affiliates lacking programming skills, who then use it to carry out their own ransomware attacks. Practically, these services are mostly "advertised" on the dark web [23]. *Ransomware generators* (or *ransomware creators*) that come with this business model have grown more notorious. A ransomware creator is some application or tool used for creating custom ransomware simply by providing contact information and an unlock path. Ransomware generators are readily available to the public on the legitimate web. When querying "how to create our own ransomware" on search engines, it would return many step-by-step instructional videos and tutorials. Hence, the high availability of ransomware generators poses a significant potential risk to online users, which necessitates developing effective countermeasures against their proliferation.

Some existing security reports have revealed the nuisance of ransomware generators. For example, a report released by `360 Beaconlab` [3] illustrates the usage of ransomware generators and their propagation process via `Tencent QQ` groups. However, our research community still lacks a comprehensive understanding of the ecosystem of Android ransomware generators in the wild, and there is no accessible dataset for researchers to study ransomware generators.

This Work. In this paper, we present the first measurement study of Android ransomware generators in the wild. We make efforts to collect a valuable dataset including 4,216 Android ransomware generator apps (see Sect. 3.2), which, to the best of our knowledge, is the first Android ransomware generator app dataset in our community. We then perform a comprehensive analysis of ransomware generators in terms of their families, clustering, operational logic, and malicious actors behind them (see Sect. 4). Next, we study the ransomware apps generated by these generators, including their generation templates, family labeling, and their locking and unlocking mechanisms (see Sect. 5). Lastly, we provide an overview of the ecosystem of ransomware generators, including how they are developed and propagated, as well as their communication with remote servers (see Sect. 6).

Among the many interesting findings, the following are prominent:

- *Not all ransomware generators create ransomware properly.* Through sampling and manual analysis, some of our collected ransomware generator apps cannot generate ransomware properly. Interestingly, some ransomware generators are actually used as bait to seduce people (affiliates) and turn them into victims. For example, some malware developers spread ransomware in the guise of a ransomware generator to inflate their own profits.
- *The ransomware generators often use some templates for creating ransomware.* The striking similarity between many generated ransomware reveals that many ransomware generators actually use certain predetermined templates to create ransomware. More exactly, a single ransomware generator can contain more than one type of template, and one template can be used by multiple ransomware generators.
- *The generated ransomwares have similar locking mechanisms, but there are various unlocking mechanisms.* Through manual inspection, we observe that most generated ransomware achieves device locking by manipulating specific parameters of the `WindowsManager` class, while they use various unlocking methods such as password, remote control, phone answer, etc.
- *The ransomware generator ecosystem has formed a complete industrial chain.* The ecosystem is composed of three phases, i.e., development, propagation, and exploitation, which are usually associated with different participating entities. Social media platforms and online discs are the main distribution channels for ransomware generators.

To facilitate further study in this direction, we have released our dataset to the research community [5].

2 Background and Related Work

2.1 Ransomware and Ransomware Generator

Ransomware is a kind of malware that demands payment in exchange for a stolen functionality [22], causing considerable economic losses every year. Based on their apparent malicious behavior, they can generally be divided into two different types [19]: one is a screen locker that acts as a lock on the victim's device, and the other is a crypto that encrypts the victim's files, rendering them inaccessible. Both types of ransomware are created to intimidate victims to pay for ransom. However, they do not guarantee that the devices or files can be unlocked after paying for ransom. More importantly, there is an emerging business model called Ransomware as a Service (RaaS) that thinks of ransomware as a service [9], where malware developers rent out ransomware and its control infrastructure to other cybercriminals. This has spawned a product that can generate custom ransomware, called ransomware generator or ransomware creator. For convenience, this paper refers to them uniformly as ransomware generators (RGs). Most of the RGs that can be found are easy to use, and those who use them do not even need to master programming skills. The advent of RG has lowered the barrier to participation in the ransomware economy. Despite the lack of research, RG is the real existential threat to cybersecurity.

2.2 Ransomware Detection and Analysis

Ransomware is always a hot topic because of its threatening and negative impact. A number of researchers have focused on ransomware detection and analysis in our community. As there are many different types of ransomware, the detection methods are not set in stone. For example, RansomProber [20] is a real-time detection system that analyzes user interface widgets of related activities and the coordinates of users' finger movements to detect ransomware. Ju-Seong Ko et al. [25] proposed a method of four steps including "Monitoring, Encryption, Already Analyzed and Is Normal" to detect new/variant/unknown ransomware in real-time. Some detection methods are proposed based on machine learning. For example, Su et al. [31] proposed a light-weight method to automatically detect locker ransomware by extracting features from multiple sources, and used four kinds of machine learning algorithms including Support Vector Machine (SVM), Decision Tree (DT), Random Forest (RF) and Logistic Regression (LR) for classification. TaeGuen Kim et al. [24] used an existence-based and similarity-based method to generate feature vectors and proposed a multimodel deep neural network model to fit the features. Besides, Monika et al. [28] analyzed the evolution of different families in the Android environment and revealed that ransomware attack is related to requesting permissions. Sharma et al. [30] coined the term "RansomAnalysis" to analyze the AndroidManifest.xml file for the extraction of permissions. Alejandro Martín et al. [27] focused on a specific family feature `jisut` and provided a detailed analysis of this family.

3 Study Design

3.1 Research Questions

This study aims to demystify Android ransomware generators and their ecosystem by conducting comprehensive research. We seek to focus on the following research questions (RQs):

RQ1 *What are the characteristics of the Android ransomware generator apps?* No prior work characterizes Android ransomware generators in a comprehensive manner. Our community lacks a good understanding of the profiles, behaviors, and characteristics of the Android ransomware generators in the wild.

RQ2 *What are the characteristics of the generated ransomware from Android ransomware generators?* The Android ransomware generators are used to create a variety of Android ransomware, it is thus interesting to investigate how these ransomware apps are created and how they operate (lock and unlock) on victims' smartphones.

RQ3 *What are the characteristics of the ecosystem of Android ransomware generators?* We seek to provide an overview of the structure and participating entities of the ecosystem. The findings may be helpful in preventing the proliferation of ransomware generators.

3.2 Dataset

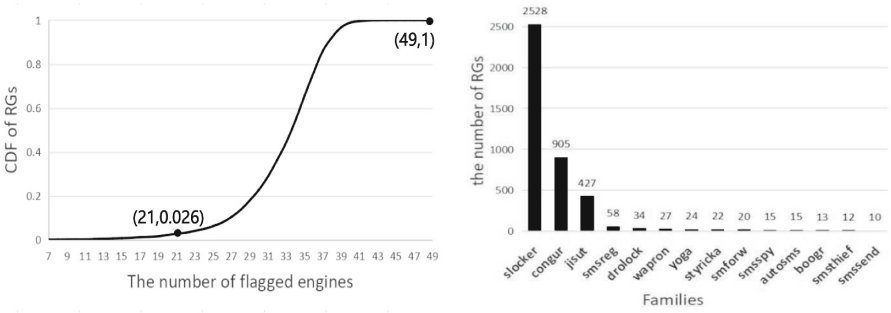
To answer the research questions, we need to harvest a valuable dataset of Android ransomware generators. While we sourced samples from various places, official app markets like Google Play rarely had ransomware generator samples. To the best of our knowledge, Koodous [13] stands out as the leading online platform for Android malware research that provides mobile malware downloading services. It is by far the largest Android app repository open to the public, hosting samples from various sources including app markets, webpages, and thousands of researchers. Thus, we purchased the premium services of Koodous for data collection. To specifically target ransomware generator apps, we relied on a keyword matching-based method utilizing some keywords such as "lockscreen generator", "ransomware generator" (in both English and Chinese languages) as the seeds to search related apps on Koodous. We got a total of 5,942 apps returned. However, not all of these apps are genuine ransomware generators that we expect. We first randomly selected a number of apps for manual inspection and found that some apps lock the device immediately after running, that is, they are actually ransomware. Thus, to further sanitize the dataset, we implemented a filtering process trying to retain the real ransomware generator apps. Specifically, the filtering is based on the observation that the genuine ransomware generators have an `assets` directory containing template files that enable them to call the `getAssets()` function to read these template files (see Sect. 5.1 for details). So we distinguish true ransomware generators from other malware by detecting whether or not they contain this `assets` directory. As a result, we obtained a total of 4,216 ransomware generator apps, with a size range from 0.06 MB to 31.80 MB. These apps are generally small, with 93.12% being less than 12MB in size.

4 Characterizing the Ransomware Generators

4.1 VirusTotal Detection and Family Labeling

For the collected ransomware generators, we seek to understand how they behave in antivirus detection, e.g., whether they can evade antivirus detection and whether they can be identified as ransomware-related. VirusTotal [14] is a widely-used online service that aggregates over 60 antivirus engines. We uploaded all the collected ransomware generator apps to VirusTotal (in July 2022) in order to examine their maliciousness. Figure 1(a) shows the detection results of VirusTotal. We found that all the ransomware generators in the dataset were flagged as malicious by at least 7 antivirus engines, with over 97.31% of them being labeled as malicious by more than 20 engines. This suggests that these ransomware generators are detected as malicious by quite a number of antivirus engines.

After obtaining the VirusTotal scan results, we utilized the widely used malware family tagging tool AVClass2 [29] to assign a family name to each ransomware generator. AVClass2 selects the highest voted known family (FAM) or unknown tag (UNK) as the likely family for the sample. If an app does not have



(a) Distribution of VirusTotal Flags (b) Family distribution of the RGs

Fig. 1. Distribution of VirusTotal results and families

any family or unknown tags, it will be tagged as `SINGLETON`. In our dataset, 4,160 (98.67%) of the ransomware generators were tagged with 32 unique family names, and 56 apps were left untagged (i.e., assigned a `SINGLETON` tag). However, we found that the BEH (which refers to an app's behavior) of 22 untagged ransomware generators was tagged as `lockscreen`. This indicates that although these ransomware generators are not assigned to any family, they still exhibit malicious screen-locking behavior. Figure 1(b) shows the distribution of the number of RGs for each family that contains no less than 10 RGs in the dataset. The family containing the largest number of RGs is `slocker` with 2,528 RGs, followed by `congur` (905) and `jisut` (427). Interestingly, we notice that some of these families do not belong to the ransomware category. For example, 20 RGs were identified as `smforw` which belongs to the spyware category. To investigate this further, we randomly selected several RGs that were identified as non-ransomware families such as `smforw` and `smsreg`, and manually installed them. We found that the interfaces and behaviors of these apps were similar to those of other ransomware generators. However, in addition to generating ransomware, they were also capable of generating other types of malware such as phishing apps. Thus, we conjecture that the reason they are marked as other categories may be due to their multifaceted behaviors.

4.2 Clustering of Ransomware Generators

To investigate the behaviors and characteristics of different ransomware generators, we cluster them based on similarities in resources and code. Firstly, we used an open-source tool called FSquaDRA2 [21] to cluster apps by resource similarity. FSquaDRA2 measures the bytecode of two apps based on a feature set of resource names and asset signatures using Jaccard distance/similarity. Based on the calculated results, we empirically set the similarity threshold as 0.8, i.e., for the apps with similarity scores higher than 0.8, we group them into the same cluster. This results in roughly 300 clusters covering 2,794 apps, with over 1,400 apps remaining ungrouped. We then used SimiDroid [26] to perform code-level

Table 1. Characteristics of each cluster

cluster	# APKs	# Pkg	Normal RG	Malfunctional RG	Fake RG	Tricking RG	# Developer Signature
com.bangbangtang.lock	942	40	✓				17
com.qqmagic	587	16			✓		4
com.mzbahkbb	563	36	✓				10
com.binge.shuazan	293	8			✓		4
com.qqmagic*	173	9			✓		3
com.js.jianshang	152	36	✓				13
com.lesj.ll	112	36		✓			5
com.jiwu88	79	30		✓			6
com.lock.scq	58	5	✓				3
com.patriot666	57	11	✓				4
com.lock.xiaoyu	48	2			✓		2
com.yhsjscq	48	7	✓				6
com.suojishengchengqi.cx	47	1				✓	3
com.xd.scq	44	2				✓	1
com.phone.locksc.yc	41	4	✓				4
com.ppopp	37	9	✓				3
v.Vike	30	2			✓		1
com.xian	30	4	✓				4
com.w	21	1		✓			2
com.binge.shuazan*	20	9			✓		4
com.bangbangtang	19	3	✓				3
quanxia.suojiqi	18	6			✓		2
com.mycompany.myapp	17	1				✓	1
com.newapp	16	5		✓			10
com.yhsjscq*	14	12	✓				4
ku.look	13	8	✓				2
ay.suojilock	11	3	✓				4
com.qqmagid	11	1			✓		1
com.suoji.scum	10	2	✓				3
com.suoji	10	1				✓	1
android.support.v7	10	1		✓			2
com.dump.iapp	9	4		✓			2
com.moshang.scq	7	1		✓			1
com.QUN.didi	5	1		✓			1

similarity measurements among apps. On the basis of FSquaDRA2 clustering results, we randomly selected one app from each cluster as the central app of that cluster, then used SimiDroid to compare the "simiScore" values between every two central apps. If the score exceeds 0.8, we merge the corresponding clusters. Next, we used SimiDroid to measure the simiScore of each isolated app to the central app of each cluster, and added the isolated app to the corresponding cluster when the simiScore exceeds 0.8. In this way, we end up grouping 3,552 apps into 34 clusters and the remaining 664 apps are isolated.

Table 1 shows the attributes of RGs for each cluster. We extract the package name of each RG and take the most frequent package name in each cluster as the cluster name. If there are two clusters with duplicate names (note that different apps can have the same package name), one of them is marked with a '*' suffix. Next, we randomly selected 5 RGs in each cluster and manually installed and ran them on smartphones to examine their behaviors. Unfortunately, we found that not all the RGs could generate ransomware normally, e.g., some of them are

actually malware in the guise of ransomware generators. We categorized these RGs into four categories based on their specific behaviors. (*i*) *Normal RG*, which are the real RGs that can generate custom ransomware normally as intended. (*ii*) *Malfunctional RG*, which fail to perform their intended function properly (e.g., they cannot be installed or run successfully), yet they also do not display any obviously malicious behavior. (*iii*) *Fake RG*, which have similar interfaces to normal RGs but directly lock the screen when the "generate" button is pressed. They are usually actual ransomware under the guise of RG. (*iv*) *Tricking RG*, which can generate ransomware normally but does not allow for customization of the unlocking way, resulting in the original creator being the sole profit maker. We next describe these four categories of RGs in more detail and provide the corresponding number of clusters for each category based on our tested apps, but note that different categories of RGs may appear in a cluster.

(*i*) **Normal RG: Generating custom ransomware.** There are 14 clusters (with 1,965 RGs, 55.32%) that can generate custom ransomware normally. These RGs have the ability to generate one or multiple types of ransomware, and users can choose the type they want and provide custom information (such as contact details and unlocking passwords) to generate it. Some of these RGs require registration or sharing the app with others for a certain number of times before they can be used. To register, users must contact the RG's producer to get a special passcode, which usually involves a fee. There is always a text prompt in the app showing the producer's contact information (usually his social media accounts) and the amount required as an electronic red envelope payment. We also find that a small number of RGs connect to specific websites where producers can verify user registration details before enabling their use (producers write the user's registration number to the website and the RG visits the website before running to determine if the user is available). Besides, some RGs have a trial period when users can use them for free. Besides, we also observe that some RGs can even generate other RGs that are capable of generating custom ransomware.

(*ii*) **Malfunctional RG: Unable to generate ransomware properly.** There are 8 clusters (with 259 apps, 7.29%) found to be incapable of generating ransomware normally. These apps display various issues such as generating a blank APK, no response when clicking the "generate" button, and absence of input boxes. The underlying reasons behind these issues are program exceptions and server exceptions. Additionally, some of these apps lack a Manifest or signature, which prevents them from being installed successfully. Nevertheless, despite the malfunction, there are no apparent signs of any malicious behavior exhibited by these apps. Thus, these RGs were simply categorized as malfunctioning, and incapable of generating ransomware properly.

(*iii*) **Fake RG: Locking the device directly.** We identified 8 clusters (with 1,180 apps, 33.22%) that exhibit malicious behavior that involves locking the device directly, i.e., they are essentially ransomware disguised as RGs. These fake RGs appear like normal RGs and also require users to input relevant details such as contact information and unlock password. They also provide different options to convince users that they can generate various kinds of ransomware.

To perpetrate their malicious schemes, the apps induce users to grant root permission and click the "generate" button, but in fact, they do not generate any new app after users complete the operation. Instead, they reboot the device and lock it directly. Since the ransomware is written to the system partition, it is not possible to prevent the locking behavior by entering the recovery mode to restore the factory settings.

(*iv*) **Tricking RG: Generating non-custom ransomware.** This is a type of fraudulent RG with nasty behaviors. We identified only 4 clusters (with 118 apps, 3.32%) belonging to this category, which is significantly lower than other categories. This category of RGs has an interface closely resembling that of the normal RGs, allowing users to input their own information to generate ransomware. After clicking the "generate" button, a new ransomware appears to be created successfully. However, the generated ransomware is actually not user-customized. Its unlocking password and contact information on the interface are predetermined by the RG's original creator and cannot be authentically changed. When running the generated ransomware, the victim's device will be locked. The victim must contact the RG's original creator to pay the ransom (as the contact information displayed is still attributed to the RG's original creator). This deprives the RG's users (i.e., would-be attackers) of any benefit from the generated ransomware, and ironically, the would-be attackers turn out to be the victims (imagine a scenario where they test their own generated ransomware and their device cannot be unlocked unless they pay the ransom). We suspect that this is a ploy by the creators of the RGs to hook more victims.

4.3 Operational Logic of Ransomware Generators

In this subsection, we seek to comprehend the operational mechanism of these ransomware generators and their ability to create a ransomware sample. Our primary focus is on the analysis of those ransomware generators that are capable of generating ransomware properly (i.e., Normal RG). Typically, these RGs set up a class for each type of ransomware they produce. When a user selects a specific ransomware type to generate, the RGs invoke `Class.forName()` function to retrieve a class object corresponding to the selected ransomware type. An `Intent` is then created and activated using `startActivity()` function. A user is presented with an input screen where he can enter his contact information and unlock password. Next, after a user verifies what he inputs and chooses to generate, the RGs call `getAssets()` function to read resource data including templates and other resource files relevant to the generated ransomware from their assets directory. The RGs then encrypt the contact information and unlock password entered by the user and embed it into the generated ransomware (the encryption method differs among RGs). At last, the results of whether RGs successfully generate ransomware can be returned by the `handleMessage()` method.

Regarding the type of ransomware generated, some RGs can only generate one kind of ransomware that can be unlocked in one way, some RGs can generate different kinds of ransomware with different unlocking means, and some can generate not only ransomware but also ransomware generators and other malware

such as phishing app. For RGs that can generate different kinds of ransomware, they usually set a button for each type of malware and listen to the button click event to confirm which type of ransomware is generated.

4.4 Certificates of Ransomware Generators

A unique developer certificate can generally be a means of developer identification. To identify the developers of the analyzed RGs, we used `keytool` [16], a key and certificate management utility, to collect the developer certificates for each RG. We discovered a total of 97 unique certificates, including 93 private keys and 4 generic/public keys. These 4 generic keys are publicly known private keys in the AOSP project, consisting of *test* key, *platform* key, *shared* key, and *media* key. For example, the test key is the default key used for packages that do not specify a private key. In our dataset, 3,584 (85.01%) RG apps use these four types of public keys, and the most widely used one is the test key, with 3,556 (84.34%) RG apps. This indicates that a large number of developers tend to hide themselves without using their private certificates. Among those that used their own private keys, we observed that the most used private key[1] was employed by 199 RG apps. Furthermore, 27 of the private keys had created more than one RG in our dataset. Additionally, 16 private keys were found in more than one cluster. For example, the private key[2] was present in 12 clusters and was used by 92 RG apps. We suspect that these private keys correspond to habitual malicious developers who engage in many ransomware-related campaigns.

As some private keys were used in many apps and found in different clusters, we further want to know whether these private keys are commonly used to make ransomware. We selected the frequent private keys that were used in more than 5 apps or more than 5 clusters, resulting in a total of 8 keys. Then we searched these keys in Koodous to find their related apps and we collected the latest 50 apps for each key according to the submission time (if there are less than 50 apps, we collect all of them), resulting in a total of 387 apps. Then we submitted these apps to VirusTotal and used AVClass2 to get the tagging results. As a result, all private keys had apps tagged as belonging to the ransomware family. However, it is noted that many apps associated with these keys were also tagged as other types of malware. For example, among apps using the key[3], 7 apps were classified to ransomware family while 30 were tagged as Trojan types, including `smssend`, `smsspy` and `smforw`. This suggests that the actors behind these RG apps are not solely focused on ransomware generation but are also involved in the creation of other forms of malware.

[1] SHA1: BD1C65A339E6D133C3C5ADB0A42205BE90F36CCD.
[2] SHA1: 5CBC9CFAD34A8E13B39863A0EC3F4EB48416F012.
[3] SHA1: 8BCA1D7A29BD40BE8562AA6F795BA1C6A2F46D38.

5 Characterizing the Generated Ransomware

5.1 Identification of Generation Templates

Upon experimentation with RGs to create ransomware, we discovered a striking resemblance between many of the resulting apps. We randomly selected several RGs from each cluster to examine the files stored in their `assets` directory and found that the files were largely similar as well. Thus, we conjecture that these RGs create ransomware by using some predetermined templates.

We further seek to uncover the templates utilized by the RGs in creating ransomware. As the templates are usually stored in the `assets` directory, which the RGs access through the `getAssets()` function, we focus on the files in `assets` directory of each RG. Of the 1,965 RGs capable of creating ransomware normally, there are a total of 16,449 files in their `assets` directories. Obviously, most of the files are not relevant to our search for templates. To accurately pinpoint the template files for each RG, we devised a 5-step search strategy that includes: (*i*) *Filter by file size.* We manually examined 100 RGs and found that the smallest size of their template files is 17 KB. Thus we first conservatively eliminated the files that are smaller than 10 KB to narrow the candidate files down to 13,138. (*ii*) *Deduplicate the files.* We then deduplicated the files, as some identical files are present in different RGs. Using a hashing function, we obtained a total of 1,714 unique files with SHA256 values. (*iii*) *Sanitize through VirusTotal.* The 1,714 unique files can have various purposes while the true template files are supposed to have ransomware-related properties. We then scanned these files in VirusTotal and kept only those that were tagged as ransomware families (e.g., `jisut, congur, slocker, ransomkd`) or had lock screen behavior. This resulted in 546 files. (*iv*) *Cluster the similar files.* Although these 546 files are unique with separate hashes, some of them may differ very slightly (e.g., only a different string or number). Thus, we next tried to group similar files into clusters. We also used FSquaDRA2 to cluster the files based on their resource similarity, and we obtained 47 clusters. We then selected one file from each cluster for further analysis. (*v*) *Manual verification.* At last, we manually validated the files by static analysis to ensure that the real template files were identified. In the end, we obtained 20 templates in total and 2 of them were the templates of RG that can be used to generate RG. Therefore, we obtained 18 templates used by these RGs to create ransomware.

It is worth noting that a single RG can contain more than one type of template, and one template can be used by multiple RGs. Among the 18 templates we identified, 14 of them are used by more than 500 RGs. Besides, we discovered that the majority of these templates do not have a '.apk' file extension. Instead, they often lack a file extension or use extensions such as '.zip', '.png', '.mp3', '.txt', '.html', '.jar', etc., in an attempt to disguise themselves.

5.2 Families of Generated Ransomware

We use each of the templates to generate a ransomware app, resulting in 18 distinct ransomware apps. We first discuss the families and behaviors of the

Table 2. Characteristics of generated ransomware

Generated Ransomware (MD5)	Family	Unlocking Method
72dafc98a0073c40ba0d518bf9912a6b	slocker	Password unlock
f121118290e32255c5597712f4df5fa2	slocker	Password unlock
57254192f0dd966f9bf77e09c144209c	slocker	Password unlock
e5321c621b092e74a30ed2e1cea51d8f	congur	Password unlock
a294526801e4535e95f1685881540142	slocker	Serial number unlock
c7880d5a2c42a410f57011480e9b7df5	slocker	Answer the phone
fb60e94a7bc9da4f263a2943dd614806	congur	Password unlock
0e12e6a8ffc3ae2e65bbd8b0cb4aac11	congur	Remote control
ed476949368459346a3cb4b88632a647	jisut	Speech Recognition
b1373249a27913c4648bc4907ae6d359	congur	Serial number unlock
d7ee2fedb12daa3064bf9569188366a9	slocker	Serial number unlock
c236e6b022ef7e2c60c768e2b72e8fdf	slocker	Remote control
fe32225445a183f2922daec95dbf7d1a	congur	Remote control
4378139261daab3b22a0d1785e1fad66	slocker	Serial number unlock
fcc493f563b09ebb84a00fa7f4dc6441	Congur	Password unlock
fe33ae94fba3ffca02a3aa5f4cd93e08	jisut	Password unlock
a71bb0247a98203e3fb32f3b1b844ea1	congur	Volume key unlock
e8b8936ef367062381b0e9a962540151	slocker	Password unlock

generated ransomware. Note that all the ransomware generated in this study behaves to lock the device. We uploaded them to VirusTotal and found that all of them were detected by over 20 antivirus engines. Further, AVClass2 was used to label their families, and the results are presented in Table 2. Each ransomware app was tagged with a family name, including `congur`, `slocker`, `jisut`.

5.3 Locking Mechanisms

We performed a static analysis on the generated ransomware apps and found that they achieve screen locking in a common way, i.e., by manipulating specific parameters of `WindowManager`. To be specific, `WindowManager` in Android is responsible for managing the windows and views of the app, including the positioning, sizing, and layering of those windows. It has some configurable parameters that allow for the manipulation of the user interface. For example, `type` parameter is an integer value that is typically used to specify the type of window and determine its behavior. And `flag` parameter is a bitmask that is used to control various aspects of the window behavior, such as whether it should be shown when the device is locked and whether it should be kept on top of other windows. By our observation, almost all of the templates manipulate the WindowManager's type and flag parameters to make the locked screen appear at the topmost level, i.e., achieve screen locking. Most of them set the type parameter as 2010,

which is associated with `TYPE_SYSTEM_ERROR` that has the potential to disrupt the user's experience and cause confusion or frustration, making the system-level error messages or alerts appear on top of everything. Meanwhile, they also set the flag parameter (`FLAG_FULLSCREEN=1024` and `FLAG_LAYOUT_IN_SCREEN=256` in combination) to create a fullscreen immersive mode for an app's window. In this way, the locked window is allowed to be displayed in full screen at the top-most level. Besides, several templates set the type parameter as 2002, which is associated with `TYPE_PHONE` that creates a phone call UI that floats above all other windows, or 2005, which is associated with `TYPE_TOAST` that creates simple and lightweight messages that appear on top of all other windows. In addition, the flag parameter can also be set to `FLAG_NOT_FOCUSABLE=8` which specifies that a window should not receive focus or input events so that the victim will not get any response after clicking on the screen. After setting the two parameters, the ransomware then calls the `addview()` function to add the "view" to the currently locked window. The *Home*, *Menu* and *Back* buttons become unusable so that the victim is unable to close the currently locked window. Some ransomware even requests `RECEIVE_BOOT_COMPLETED` permission to prevent the victim from rebooting the device and escaping the locked screen. Worse still, 9 of the templates induced victims to grant the permission of `device_admin`, which allows them to reset the device's PIN. These RGs create a new Intent to call the admin activation interface and use the `intent.putExtra()` method to add their own `MyAdmin` class to the intent. Once the intent is started, it jumps to the `MyAdmin` class and adds `getManager(context).resetPassword()` function, which can reset the victim's PIN. By changing the PIN, the victim may not be able to guess or brute-force their way past the lock screen, and may be more likely to pay the ransom to the attacker in order to regain access to their device.

5.4 Unlocking Mechanisms

Despite the common locking mechanism, the ransomware apps use various methods to unlock the devices. As shown in Table 2, we identified 6 different methods used by the apps to achieve device unlocking. We next describe each in detail.
(*i*) **Password unlocking.** The most common way to unlock ransomware-locked devices is through a fixed password. Victims are instructed to contact the ransomware creator, usually via the contact information displayed on the screen, to get the password. Once the correct password is entered, the device can be successfully unlocked. There are also some ransomware that require shaking the phone to unlock it after entering the password. In order to create this kind of ransomware via an RG, would-be attackers need to provide the unlock password and contact information (displayed on the screen).
(*ii*) **Serial number unlocking.** In this method, a string of random numbers will appear on the screen (known as serial numbers) once the device is locked. The victim sends this serial number to the attacker who performs the calculation and sends the result back to the victim after the ransom is paid. The victim can successfully unlock the device by entering the calculation result obtained. To generate this kind of ransomware using an RG, users need to provide contact

information and basic mathematical operation orders (add, subtract, multiply, divide) to instruct the generated ransomware to complete the same calculation and thus determine if the victim input is correct. There are other variations of serial number-based unlocking methods as well. For example, there are several serial numbers displayed on the screen, but only one of them that satisfies specific rules can be computed correctly. The specific rules are set in RGs (e.g., among all the serial numbers, only the one whose second number is '1' is correct to be computed), and victims must get the rules from the attackers. In addition, there is also a form where victims must perform specific actions that are predefined in code, such as clicking a specific button at a specific time in a preset order, before seeing the serial number. For attackers, this is a more secure way to unlock the device as compared to fixed password unlocking, compensating for the danger of password leakage.

(*iii*) **Remote control unlocking.** This method incorporates a website acting as an intermediary between the attackers and the victims. We found two forms of such remote control unlocking. The first one is to use a website as a password transmitter, allowing an attacker to easily change the device's unlock password. The password is often written between specific notations (e.g., "【", "】") on the web page and can be changed easily. The victim needs to contact the attacker, pay for ransom to get access to the specific website, and thereby obtain the password string. Once entering the password, the ransomware verifies whether the input password is correct. As shown in Fig. 2, it compares the string between specific notations that get from the website with the input string from the victim to see if they match so as to determine if the input password is valid. This method involves web page interaction, and the unlock password is dynamic and easy to reset. Note that the entire unlocking process requires the victim's device to be connected to the Internet. To generate this type of ransomware, attackers need to provide their contact information and the address of the remote control website.

```
// java.lang.Thread, java.lang.Runnable
01 public void run() {
02   HttpURLConnection HUC = new URL (decrypt(openRawResource(*)).openConnection();
     //decrypt predefined website from resource file and try to connect
03   message.obj = HUC. getInputStream();// send message to another thread
04 }
05 public void handleMessage(Message message) {// android.os.Handler
06   String str2 = (str.subSequence(str.indexOf("【") , str.indexOf("】 "));
    //str refers to message.obj
   //get the string  between "【" and "】 " as password
07   if (*.getText().toString().equals(str2)) { //get inputting string from editText and compare
08       mWindowManager.removeView(*); //remove the window to unlock
09 }}
```

Fig. 2. Remote control unlocking

The second form of remote control unlocking is to use a website as a gatekeeper to make decisions about whether to approve the unlock. Victims are required to enter the serial number of the locking interface into a predetermined website. To get the website, victims must contact the attackers and pay for it. After that, the victim presses the unlock button, and then the ransomware connects with the website and checks whether the serial number is on the web page. If so, the device can be unlocked. When generating this kind of ransomware using an RG, attackers need to provide contact information and website addresses.
(*iv*) **Phone answering unlocking.** This way to unlock the device is by answering a call from a specified phone number. Attackers pre-set a specific phone number in the code when generating the ransomware. Once the device is locked, the victim contacts the attacker according to the contact information displayed on the screen, pays for ransom, and gives his own phone number to the attacker. Upon receiving the ransom, the attacker dials the victim's mobile phone with the pre-set phone number. The ransomware continuously monitors the device and records every incoming phone number, as demonstrated in Fig. 3. If the number matches the one set in advance by the attacker, the device can be unlocked.

```
// android.content.BroadcastReceiver
01 public void onReceive(Context context, Intent intent) {
02   if ("android.intent.action.PHONE_STATE".equals(intent.getAction())) {
     //whether the action is incoming call
03    if(decrypt(openRawResource (*)).equals(intent.getStringExtra ("incoming_number"))) {
     // decrypt the predefined phone number from the resource file
04       mWindowManager.removeView(*); //remove the window to unlock
05  }}}
```

Fig. 3. Answer the phone unlocking

(*v*) **Speech recognition unlocking.** This kind of unlocking method requires the victim to speak a pre-defined password (obtained after paying the ransom) to unlock the device. It also requires the granting of the recording permission. As shown in Fig. 4, the ransomware uses the speech recognition method in `DialogRecognitionListener` class to convert the spoken password into a string and remove any garbled characters. The ransomware then determines the similarity between the resulting string and the pre-defined password. If the similarity exceeds a certain threshold, the device will be unlocked successfully. To generate this kind of ransomware, attackers need to provide the unlock password.
(*vi*) **Volume key unlocking.** This kind of unlocking method requires the repeated pushing of a device's volume keys, such as two pushes of the volume up key followed by two pushes of the volume down key in quick succession. As demonstrated in Fig. 5, the ransomware uses `onkeydown()` function to confirm the correct key sequence, uses `System.currentTimeMillis()` function to ensure

```
// com.baidu.voicerecognition.android.ui.DialogRecognitionListener
01 public void onResults(Bundle bundle2) {
02  ArrayList<String> sAL =  bundle2.getStringArrayList(DialogRecognitionListener.
RESULTS_RECOGNITION) ; //get recognition results
03  String str1 = decrypt(openRaw Resource(*));
//decrypt predefined password from resource file
04  String str3 = sAL.get(0).replaceAll ("str2", "");//str2 refers to some garbled characters
//remove garbled characters from recognition results
05  if (SimilarDegree(str2, str3) >= 0.7d){//compute similarity degree
06   mWindowManager.removeView(*);//remove the window to unlock
07 }}
```

Fig. 4. Speech recognition unlocking

```
// android.app.Activity, android.view.KeyEvent.Callback
01 public boolean onKeyDown(int i, KeyEvent keyEvent) {
02   if (i == 25) {//25 stands for volume down key,24 stands for volume up key
03     if (System.currentTimeMillis() - this.mExitTime > 2000) {
       //the interval between two pressing activities should be less than 2s
04       this.mExitTime = System.currentTimeMillis();}
05     else {
06       this.down = true;
07       if (this.down && this.up) {//the value of "up" is decided by volume up key
08         System.exit(0);}}// use exit(0) to unlock
09    keytouch(this.usedTime, this.keyTouthInt, 2);}
     //in the volume up key part,the third parameter is 3;
```

Fig. 5. Volume key unlocking

the operation is completed within a specific time frame, and employs an integer to track the number of times the key is pushed. Unlike other methods, no prior setup is necessary to produce this kind of ransomware.

6 The Ecosystem of Ransomware Generators

6.1 The Structure of the Ecosystem

We uncover that the RG's ecosystem is a diverse and sophisticated system with different participating entities. These entities have established an underground industrial chain to create and distribute nasty RGs and ransomware for financial gain. Figure 6 illustrates the operational pipeline of the ecosystem, which is composed of the following three phases.

(i) **The development phase.** In the first phase (❶ in Fig. 6), the developers are responsible for creating various ransomware generators. Typically, they first create ransomware apps and then transform them into templates from which they generate the ransomware generators. Even more, a ransomware generator can also be used as a template for creating RGs as well.

(ii) **The propagation phase.** In the second phase (❷ in Fig. 6), the developers or agents start to distribute the created RGs and guide the potential attackers to

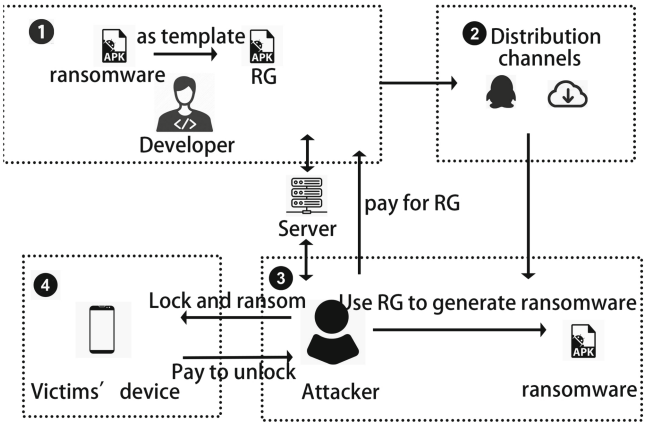


Fig. 6. An overview of RG's ecosystem

use them. The distribution channels are mainly through social apps (e.g., Tencent QQ) and online disks (e.g., Baidu Netdisk). The potential attackers usually pay for access to the RG and are provided with instructions such as a tutorial video on how to use it. However, we also find that there are some free RGs circulating in the wild. We suspect that this may be facilitated by the advanced techniques and shared ideologies within the community. For example, an attacker can obtain the source code of an RG using methods such as decompiling, and then with minor adjustments such as updating the contact information or resource files, a new RG can be obtained and shared.

(*iii*) **The exploitation phase.** In the third phase (❸ and ❹ in Fig. 6), potential attackers can easily create customized ransomware using the RGs. The attackers then distribute their ransomware apps to potential victims and use some tricks (e.g., disguise the generated ransomware as a plugin) to lure victims into installing. Once the victims are hooked, they are required to pay extortion money to have their devices unlocked.

6.2 The Development of RGs

Our investigation has uncovered two primary ways for the development of RGs. The first involves the original producer creating the RGs and then selling or sharing them to spread. The second way involves someone who can reverse-engineer the RGs, secondary process them, and distribute them. In our dataset, we find that most of the RGs are indeed easily modifiable, lacking adequate security enhancements. This comes from the observation that many RGs use the same generation templates, differing only in the contact information and app name. And when we analyzed these RGs using static analysis techniques, they were often easily decompiled. Furthermore, we found that the interface of one type of RG that generates various kinds of ransomware showed technical support from the same individual called 'bi an hua'. This led us to believe that not all RGs

are developed independently. We then used APKid [4], a tool that can examine an Android APK or DEX file and detect the fingerprints of several different compilers, to know whether these RGs are repackaged. In general, the dexlib families can disassemble and compile DEX files without the source code. Thus, if an RG uses these libraries to create a DEX file, we consider that it is likely to be cracked and repackaged. We feed all the normal RGs to APKid, which will decompose apps and try to identify compilers used in the app. As a result, we found that 96.16% of files were compiled with `dexlib1.x` or `dexlib2.x`, Therefore, we suspect that most of these apps have been disassembled and recompiled. This suggests that most of the RGs in our dataset come from the second way.

6.3 The Propagation of RGs

The main propagation channels of RGs we find are QQ social platform and various online discs. When manually running some RG apps, we found that the majority of developers provided contact information in the form of a QQ number. We next tried to identify the QQ numbers to find the malicious actors behind them. We scanned all files in each RG that can generate ransomware normally, performing string matching to find contact information. However, this approach is not always successful in identifying contact information. For example, some RGs give contact information that is encrypted, some RGs post contact information on remote servers, and some RGs have contact information present in dynamic loading behavior. In the end, we collected 177 unique QQ numbers (including personal and group accounts) from over 400 RGs. Among these, we selected the most frequent QQ number in our dataset (which is a group account), and successfully joined its QQ group. We found that this group is used to share RG app source code and other reverse-cracked apps, with over 750 QQ members. The source code and apps are often shared as .zip files with accompanying instructions or videos on how to use them. We also discovered some QQ groups where applicants had to pay before joining. Overall, most of the QQ groups were established in 2017 and only a small percentage of them are still active.

In our investigation of the QQ groups, we noticed that some members uploaded the source code of RGs to some netdisks, including Baidu Netdisk, wodemo, lanzous, etc. To figure out the specific content, we next used soPandas [17], an online tool that can search resources from Baidu netdisk public link automatically, to search detailed resources corresponding with ransomware generators. We find the source code of the same type of RGs in our dataset, and some compressed package even includes teaching videos on how to use the RG.

6.4 Communication with Remote Servers

During our experience with RGs, we found that some RGs have interactions with some remote servers. To know it better, we used fiddler [15], a software that acts as a proxy-server and captures Android traffic, to capture the packet data from several RGs in each cluster that can generate custom ransomware normally. We also collected domains connected with all the RGs using VirusTotal. Through

our analysis of the captured traffic and the related domains, we found that a number of RGs are connected with advertising sites, such as CDNs (content delivery networks) like 'Cloudfront.net', 'jomodns.com', and 'tencent-cloud.net'. Besides, most of RG's related domains are linked to app packing services, which protect apps from being hacked or analyzed by reverse engineering tools. For example, over 30 RGs connect with 'jiagu.360.cn', a website that provides app security enhancement services to encrypt and pack apps, making them difficult to decompile. Based on the file creation time (one of the metadata we collected), we found that RG apps created later (after 2017) were mostly obfuscated and enhanced. We suspect that later developers pay more attention to the protection of their apps. In addition, we found that a number of RGs communicate with remote servers, which can be categorized into 2 types based on their purposes.

The first one is used to verify whether users can access the RGs. For instance, as aforementioned, some RGs require users to register before they can be used. After installing the RG, the user can see a registration number on the RG's interface and needs to pay a fee to the producer to activate the registration number. The producer writes the paid registration number to his website, and the RG interacts with the specific website to check whether the user's registration number is on the website to confirm the user's eligibility to use it. The second type of remote server communication involves showing attacker's contact information. Some apps interact with remote servers to retrieve this information, which is then displayed on the interface of RGs. This approach ensures that attacker contact details remain hidden during static analysis of these RGs. Almost all the web pages belong to 'www.wodemo.net', which is a netdisk where users can register to upload files, APKs, images, and other files for free.

7 Discussion

7.1 Implication

To the best of our knowledge, we are the first to systematically analyze the ransomware generator apps in the research community. This work has several important observations that contribute to ransomware-related cybersecurity research. First, we find that not all the so-called ransomware generators can generate ransomware in a proper way. Some are actual malware that spread in the guise of ransomware generators, and some are ill-intentioned RGs that trick potential attackers into their puppets. This observation sheds light on the disorderly nature and hidden dangers of the ransomware ecosystem and exposes the deceitful tactics employed by malicious developers. It serves as a cautionary tale for those considering engaging in this illicit activity, highlighting the potential risks and pitfalls that they may encounter. Moreover, we find that RGs often use some fixed generation templates to generate ransomware. This suggests that ransomware generators may be easier to achieve than previously thought, leading to a potentially more widespread proliferation of ransomware generators. However, the use of fixed templates also makes it easier for security researchers to identify and track new strains of ransomware, as well as develop more effective methods

of detection and prevention. Besides, we reveal the common locking practices and various unlocking methods. By analyzing the commonalities in locking mechanisms, security professionals can develop more robust defenses and mitigation strategies to prevent devices from being locked in the first place. Also, the diversity of unlocking methods highlights the need for multifaceted defense strategies. Furthermore, we investigate the ecosystem of ransomware generator apps and reveal their industrial chain. We find that RGs are easy to find on the Internet, such as QQ groups and web drives. This underscores the significance of public awareness and education about the risks associated with RGs.

7.2 Limitation

Our investigation is primarily limited by the apps we identified. We collect samples from Koodous by searching keywords such as 'ransomware generator', 'lockscreen generator' (in both English and Chinese). Although Koodous hosts Android apps from various sources, there is quite a possibility that we overlooked certain ransomware generator apps in the wild. Moreover, most of the RGs we collected were searched in Chinese, which indirectly narrows our study scope to China. This is because when we searched in English, we got very few apps returned. We suspect that it may be because there are relatively few English-language ransomware generator apps in Koodous, or the terms we used in our search do not fit very well the app names of the relevant English apps. Accordingly, the propagation channels of the RGs we found are the social media platform (Tencent QQ) and online disks, which are predominantly popular in China. We would like to remark that these channels may not necessarily represent other regions accurately. In addition, all the identified RGs belong to locker-ransomware-generators that generate lockers with the ability to lock victims' devices. This is probably because this behavior is more easily encapsulated into generators. We are still not clear whether there are any crypto-ransomware-generators that can generate encryption ransomware and if so, what are their characteristics. Besides, some of the processes and conclusions seem to be empirical decision-making (e.g., the choice of some tools and some thresholds). In future work, we plan to address these limitations by expanding our sample collection from diverse sources to ensure a more scientific analysis and enhance our understanding of ransomware generators.

8 Conclusion

In this paper, we present the first measurement study of Android ransomware generator apps. We first make efforts to collect a dataset of over 4,000 Android ransomware generators, then present a comprehensive analysis of these apps from multiple perspectives including their behaviors, generated apps, and ecosystem. We reveal a number of interesting findings about ransomware generators. Our study emphasizes the importance of ongoing research into ransomware generator apps to improve our understanding of their behavior and characteristics,

which can help in the development of effective countermeasures to prevent the proliferation of ransomware.

Acknowledgments.. This work was supported in part by the National Natural Science Foundation of China (grant No.62072046), the Key R&D Program of Hubei Province (2023BAB017, 2023BAB079), and the Knowledge Innovation Program of Wuhan-Basic Research.

References

1. Cybersecurity Predictions. PHOTO: Cybercrime Magazine. 2022 Cybersecurity Almanac: 100 Facts, Figures, Predictions And Statistics.The past, present, and future of cybercrime. Sponsored by Cisco - Steve Morgan, Editor-in-Chief Sausalito, Calif. - Jan. 19, 2022. https://cybersecurityventures.com/cybersecurity-almanac-2022/
2. 15+ SHOCKING RANSOMWARE STATISTICS [2023]: TRENDS + FACTS 7 Feb 2023. https://www.zippia.com/advice/ransomware-statistics/
3. analysis of mobile malware factories. https://blogs.360.net/post/analysis_of_mobile_malware_factories.html
4. APKiD. https://github.com/rednaga/APKiD
5. Dataset of Android Ransomware Generator Apps. https://github.com/CanTu05/Android-Ransomware-Generator
6. Development of new Android malware worldwide from June 2016 to March 2020. https://www.statista.com/statistics/680705/global-android-malware-volume/
7. Kaspersky: Kaspersky security bulletin 2016 (2016). Accessed 29 June 2018. https://securelist.com/kaspersky-security-bulletin-2016-story-of-the-year/76757/
8. Mobile operating systems' market share worldwide from 1st quarter 2009 to 4th quarter 2022. https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/
9. RANSOMWARE AS A SERVICE (RAAS) EXPLAINED HOW IT WORKS & EXAMPLES. https://www.crowdstrike.com/cybersecurity-101/ransomware/ransomware-as-a-service-raas/
10. Ransomware attacks are on the rise. These are the industries most a risk Nov 26, 2021. https://www.weforum.org/agenda/2021/11/industries-affected-ransomware-cybersecurity-cybercrime/
11. Ransomware Is the Greatest Business Threat in 2022. https://www.nasdaq.com/articles/ransomware-is-the-greatest-business-threat-in-2022
12. Symantec, Internet Security Threat Report (2019). https://www.symantec.com/content/dam/symantec/docs/reports/istr-24-2019-en.pdf
13.) Koodous (2021). https://koodous.com/
14. Virustotal (2021). https://www.virustotal.com/
15. fiddler (2022). https://www.telerik.com/fiddler
16. keytool (2022). https://docs.oracle.com/en/java/javase/12/tools/keytool.html
17. SoPanda (2022). https://www.sopandas.cn/
18. Alshaikh, H., Ramadan, N., Hefny, H.A.: Ransomware prevention and mitigation techniques. Int. J. Comput. Appl. **117**(40), 31–39 (2020)
19. Cabaj, K., Mazurczyk, W.: Using software-defined networking for ransomware mitigation: the case of cryptowall. IEEE Network **30**(6), 14–20 (2016)

20. Chen, J., Wang, C., Zhao, Z., Chen, K., Ruiying, D., Ahn, G.-J.: Uncovering the face of android ransomware: characterization and real-time detection. IEEE Trans. Inf. Forensics Secur. **13**(5), 1286–1300 (2018)
21. Gadyatskaya, O., Lezza, A.-L., Zhauniarovich, Y.: Evaluation of resource-based app repackaging detection in android. In: Brumley, B.B., Röning, J. (eds.) NordSec 2016. LNCS, vol. 10014, pp. 135–151. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47560-8_9
22. Gazet, A.: Comparative analysis of various ransomware virii. J. Comput. Virol. **6**, 77–90 (2010)
23. Karapapas, C., Pittaras, I., Fotiou, N., Polyzos, G.C.: Ransomware as a service using smart contracts and IPFS. In 2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), pp. 1–5 (2020)
24. Kim, T.G., Kang, B.J., Rho, M., Sezer, S., Gyu Im, E.: A multimodal deep learning method for android malware detection using various features. IEEE Trans. Inform. Forensics Secur. **14**(3), 773–788 (2019)
25. Ko, J.-S., Jo, J.-S., Kim, D.-H., Choi, S.-K., Kwak, J.: Real time android ransomware detection by analyzed android applications. In: 2019 International Conference on Electronics, Information, and Communication (ICEIC), pp. 1–5 (2019)
26. Li, L., Bissyandé, T.F., Klein, J.: Simidroid: identifying and explaining similarities in android apps. In: 2017 IEEE Trustcom/BigDataSE/ICESS, pp. 136–143. IEEE (2017)
27. Martín, A., Hernandez-Castro, J., Camacho, D.: An in-depth study of the Jisut family of android ransomware. IEEE Access **6**, 57205–57218 (2018)
28. Monika, P.Z., Lindskog, D.: Experimental analysis of ransomware on windows and android platforms: Evolution and characterization. Procedia Comp. Sci. **94**, 465–472 (2016). The 11th International Conference on Future Networks and Communications (FNC 2016) / The 13th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2016) / Affiliated Workshops
29. Sebastián, S., Caballero, J.: AVclass2: massive malware tag extraction from av labels (2020)
30. Sharma, S., Kumar, R., Krishna, C.R.: RansomAnalysis: the evolution and investigation of android ransomware. In: Dutta, M., Krishna, C.R., Kumar, R., Kalra, M. (eds.) Proceedings of International Conference on IoT Inclusive Life (ICIIL 2019), NITTTR Chandigarh, India. LNNS, vol. 116, pp. 33–41. Springer, Singapore (2020). https://doi.org/10.1007/978-981-15-3020-3_4
31. Dan, S., Liu, J., Wang, X., Wang, W.: Detecting android locker-ransomware on Chinese social networks. IEEE Access **7**, 20381–20393 (2019)