I/O Behaviour Analysis on Android Targeted Ransomware

Beatriz Fernandes Oliveira
University of Minho
Braga, Portugal
pg50942@alunos.uminho.pt

Bruno Filipe Miranda Pereira
University of Minho
Braga, Portugal
pg50271@alunos.uminho.pt

Bruno Filipe Jardim Machado
University of Minho
Braga, Portugal
pg49997@alunos.uminho.pt

***Abstract*—The number of cryptographic ransomware attacks is increasing on a yearly basis. Similarly, this type of ransomware is evolving at an ever-increasing pace. By contrast, the current detection techniques are becoming less and less effective due to more sophisticated obfuscation techniques and are simply unable to keep up with the constant advances of ransomware. With more than three billion active users and an ever-increasing presence in our daily lives, Android devices are emerging as a new target for ransomware attacks. As such, understanding their behavior is crucial to designing new, more effective detection mechanisms. This paper proposes an extension to CRIBA [1] (an open-source framework for cryptographic ransomware analysis and comparison for the Linux operating system). This extension aims to enable CRIBA to operate seamlessly on Android devices. Our study provides insight into the I/0 patterns of cryptographic ransomware in an Android environment and how its different families compare to their Linux counterparts.**

***Index Terms*—android ransomware, cryptographic ransomware, I/O tracing and analysis**

I. Introduction

Ransomware attacks stand out as one of the most damaging malware attacks, directly compromising user data by holding it hostage and demanding a ransom. These attacks are not only increasing in frequency but also diversifying, targeting various operating systems, including Windows, Linux and Android devices.

Furthermore, ransomware continually evolves, avoiding the rules of detection tools. With this in mind, it becomes crucial and urgent to take a step back and prioritize ransomware behavior analysis to counteract this continuous threat. In this work, the focus will be on the impacts of these attacks on Android devices and the analysis of Android ransomware samples.

There are two types of ransomware attacks: **Locker** and **Cryptographic** ransomware. The Locker, as the name implies, locks the user out of their system, while the Cryptographic variant prevents users from accessing their data by encrypting files, rendering them unusable from the user's perspective. Our research will specifically concentrate on the Cryptographic variant due to its greater challenges in recovery from the attacks and its potential for more significant damage to user data.

Nowadays, android devices are an integral part of our daily activities, facilitating various tasks such as shopping, transportation, banking transactions, navigation, and sharing important documents and personal pictures. Smartphones have evolved into extensions of their owners, storing sensitive data like banking information, documents, pictures and others, making it essential to guarantee the safety of these devices and their data.

The prevalence of Android devices, coupled with the booming mobile device industry, creates a lucrative opportunity for cybercriminals. Since 2013, Android devices have been targeted by ransomware attacks. In one instance, the **ScarePackage** ransomware attack managed to infect over nine hundred thousand Android devices over thirty-day period [2].

With the rapid development of new ransomware families employing innovative techniques to avoid detection, researchers encounter difficulties in keeping pace with emerging attacks. Their response time to new attacks tends to be slower then the development speed of these new threats. Achieving quicker response times to new ransomware attacks implies a comprehensive understanding of how these ransomware families operate, the type of data that needs to be collected, and how they evolve over time.

Our primary objective, is to enhance the understanding of ransomware I/O behavior on Android devices, streamlining the analysis phase of ransomware to assist researchers in keeping up pace with new ransomware trends. Subsequently, our goal is to present a comprehensive characterization of Android ransomware families, gaining further insights into their I/O behavior interaction with the operating system.

Similar achievements have already been made on Linux systems using the CRIBA framework [1], which automates the steps involved in gathering relevant data. It correlates this data to extract crucial information about ransomware execution, ultimately creating visualizations that provide researchers with easier insights on ransomware behavior without the need to analyze large trace files.

We argue that the creation of a dedicated framework for the Android operating system is essential, given its relevance and recent vulnerability to ransomware attacks, which pose threats to user data.

However, as the CRIBA framework was originally designed for the Linux operating system, a direct application to Android devices is not feasible. Consequently our approach involves modifying the CRIBA framework to enable its operation on the android operating system.

This undertaking presents a number of challenges, particularly with the transition from Linux to Android. The tracing capabilities of CRIBA are not inherently compatible with Android, posing a significant obstacle. Moreover, as CRIBA cannot perform the required tracing on Android, the correlation algorithms present within CRIBA, which automate the data analysis, must be adjusted to accommodate a new tracer.

The created tool has allowed us to observe some similarities between the Linux and Android ransomware families. Moreover, this tool automates the analysis and visualization of Android ransomware samples behavior and, hopefully, it will aid in the development of new techniques, that are more accurate at detecting Android ransomware behavior.

The paper is structured as follows: Section II presents an overview of relevant concepts to the knowledge field of this work. Section III presents a high-level description of our solution, describing all the steps performed to enable CRIBA workflow for the Android system. Section IV describes the test environment created to perform the tool evaluation. Section V presents the results obtained by running our tool with the samples used, highlighting the analysis carried out on each ransomware family and the conclusions obtained regarding their behavior. Section VI reviews related work, and Section VII closes the paper and outlines future work.

II. Background

Ransomware is a form of malware attack with the primary objective of holding the user's system hostage until a ransom amount is paid. The primary goal of this malicious software is to extort money from its victims.

As discussed in Section I, ransomware can be categorized into two main types, **Cryptographic** and **Locker**. This research specifically concentrates on **Cryptographic** ransomware as it is the main responsible for causing significant damages to user data.

A. Phases of a Ransomware Attack

A ransomware attack can be divided in four main phases, them being: Infection, Communication with C&C server, Destruction, and lastly, Extortion [19]. Initially, in the *Infection Phase*, the malicious actors exploit system vulnerabilities or rely on social engineering techniques to deploy the malicious software on the victim's system, which culminates with the installation of ransomware samples in this system.

After the successful deployment of ransomware on the victim's system, the *Communication Phase* follows, particularly in more sophisticated attacks. In this phase, the ransomware collects valuable information about the infected system and transmits that data to a Command and Control server (C&C). The C&C server is responsible for storing and presenting the gathered information to the attackers. Additionally, these servers play a crucial role in generating payment information and storing cryptographic keys.

Following the *Communication Phase* is the *Destruction Phase*, during which ransomware initiates malicious actions to damage the user's system. In the context of cryptographic ransomware, the *Destruction Phase* involves malware encrypting files, and either deleting the original files or renaming them.

Lastly, there is the *Extortion Phase* where the ransomware has finished encrypting the files, and the attacker asks the victim for a ransom payment to release (in this case decrypt) the victim's data. Attackers often resort to cryptocurrencies as a form of payment, making it even more challenging to track these malicious actors. Typically attackers leave ransom notes informing the victim that an attack has taken place, and provide instructions on how the payment should be made.

B. Ransomware Analysis Research

Ransomware analysis aims to better understand the behaviors associated with a ransomware attack. The research in this field can be categorized, according to the approach followed, as *Dynamic Analysis* and *Static Analysis* [19].

1) Static Analysis: This technique of ransomware analysis aims to understand ransomware behavior by extracting information from a ransomware sample without actually running it. This is achieved by disassembling ransomware binaries and observing how the code is supposed to run, attempting to find suspicious activities within the malicious code. Static analysis is safer then other forms of analysis since the ransomware is not executed. This allows the threat to be identified and stopped even before the malware initiates its malicious behaviors.

However, this analysis technique can be easily evaded by more sophisticated variants of ransomware attacks that employ obfuscation methods to hide their malicious activity in their binaries.

2) Dynamic Analysis: This analysis technique involves executing the ransomware sample in a controlled and isolated environment and observing how the malware interacts with this environment. Unlike *Static Analysis*, where traditional obfuscation techniques can conceal malicious behavior, *Dynamic Analysis* allows researchers to uncover malicious activities more effectively. By observing ransomware's operations in a real execution, attackers face greater challenges hiding their actions. However, this approach, while superior in finding malicious behavior, requires substantial resources, as researchers must establish an isolated environment and execute the ransomware sample to gather valuable data.

In our approach, we will perform *Dynamic Analysis* by collecting ransomware samples and executing them in an isolated Android environment. This process involves gathering data that describes the interaction of ransomware samples with the operating system, such as the execution of system calls, I/O operations, file system operations, among others.

III. Design

A. CRIBA and DIO

Our tool was built on top of CRIBA's [1] architecture, shown in Figure 1, which itself is built on top of DIO [20], a tool for diagnosing applications I/O behavior through system call observability, in POSIX storage systems.

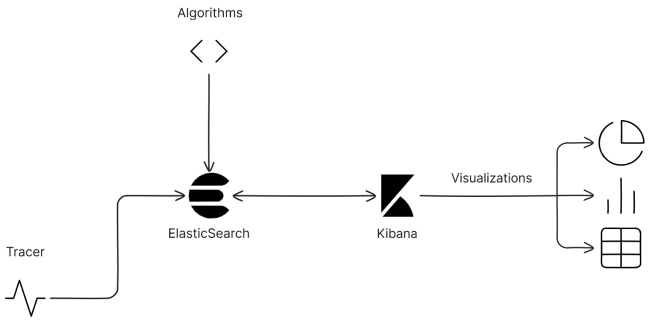

Fig. 1. CRIBA's Pipeline.

CRIBA automates the analysis of ransomware by leveraging the tracing capabilities of DIO to gather I/O data from ransomware execution and providing a set of correlation algorithms and a preset of dashboards designed for the purpose of analyzing cryptographic ransomware samples, specifically.

With all the existing similarities between the Linux and Android systems, mainly the eBPF technology and the same system calls, a large percentage of CRIBA's pipeline could be utilized for analyzing traces from an Android system while only needing a few adjustments to make a seamless integration.

Despite these similarities, the tracer used in CRIBA was not suited for tracing in an Android environment. Therefore an alternative was warranted, leading to the adoption of *BPFroid* [21].

B. BPFroid

This new tracer allows us to trace Android framework API, native libraries, system calls and other events using the eBPF technology.

By replacing the tracer used in CRIBA with BPFroid and integrating it with the CRIBA analysis pipeline, CRIBA was effectively repurposed to analyze cryptographic ransomware samples in Android devices.

Although with a few caveats, considering that the traces provided by BPFroid do not possess the same amount of data as the ones provided by CRIBA. This means that a few of the visualizations present in CRIBA's dashboards are not available when analyzing the I/O behavior of cryptographic ransomware samples in an Android environment.

C. The workflow of the tool

With all the similarities with CRIBA it is no surprise that our tool works in the same fashion as CRIBA does. That is, the execution of the tool is divided in two distinct phases:

1) Tracing: In this phase, there is the need to have a controlled environment, e.g., a virtual machine (VM) simulating an Android device, which executes a cryptographic ransomware sample. All the while, BPFroid is collecting all the system calls executed by this sample and producing a log that can be later processed and analyzed.

2) Analysis: This next stage needs to parse the traces obtained by BPFroid so they fit the same structure as the ones obtained by CRIBA before being forwarded to the analysis pipeline. This pipeline can store and index the data collected, so that the correlation algorithms can be executed. When executing these algorithms, the data is immediately shown in the preset dashboards to proceed with the exploration of the results obtained.

D. Correlation Algorithms

All of the algorithms present in CRIBA were able to be ported, with some adjustments, in order to work with the traces produced by BPFroid.

- **UNExt** Extracts the filename and extension of the files targeted by the syscalls.
- **DsetU** Compares the files targeted by the syscalls with the files contained in a given file dataset collection.
- **Transversals** Describes how the ransomware traverses the file system, e.g., Breadth First Search.
- **FnGram** Computes the n-grams colocations for the files that were accessed by a ransomware sample over time.
- **FSysSeq** Computes the sequences of consecutive unique system calls done by the ransomware threads to all files.
- **tfidFam** Eases the comparison of distinct ransomware families.

These algorithms, that are described with greater detail in CRIBA [1], allow us to have an in-depth look at how these cryptographic ransomware samples interact with the file system in an Android device to establish a profile for each one of these samples. It also makes it possible to compare these sample behaviors to the ones targeting Linux systems.

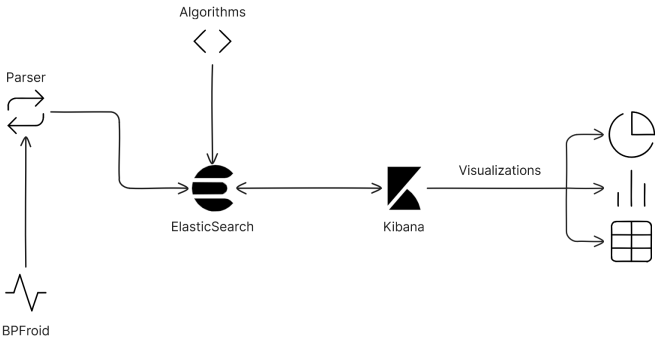

Fig. 2. Tool Architecture.

IV. Evaluation Methodology

A. Ransomware Families

Despite the first unsuccessfully attempts to gather Android ransomware samples, three Android ransomware families were chosen to carry out this analysis.

The first family, named **ATANK**, corresponds to a cryptographic ransomware found in 2018 on the GitHub platform. [11] It encrypts files using an AES key (composed of thirty-two random characters) and changes the file extension, after

being encrypted, to `.xdrop`. Another characteristic of this ransomware is that, it deletes all the files after twenty-four hours or after a reboot.

The **FileCoder** ransomware, which corresponds to the second family, can affect all Android devices with a version equal to or higher than 5.1. [13] This ransomware, discovered by the *ESET Mobile Security* in 2019, was distributed through malicious website posts and attempts to spread the victim's contacts. Moreover, it even uses this contact list to spread itself via SMS. [12], [14]

Finally, the **SLocker** family was discovered in 2016, as were more than four hundred variants in several large-scale attack campaigns distributed through fake applications. [17] The following year, a new variant was detected by Trend Micro and copied the *WannaCry* ransomware GUI. [15] More recently, another variant of this crypto-ransomware was found in an application that intended to provide users with more information about the coronavirus. [16] Instead, after being downloaded and executed, the users were locked out of their devices. Furthermore, it is considered the first Android ransomware that uses file encryption, is distributed through fake apps, and uses a TOR anonymity network to communicate with its controller. [18]

B. File System Dataset

In order to correctly assess this tool and to create an environment closest to reality for this evaluation, the team resorted to using the framework **Impressions** [22] to create an accurate file system with our operating system.

Having this in mind, the team created a file system with 1GB, that includes 3547 files with sizes between 0MB to 29MB, as it is possible to see in Figure 3.

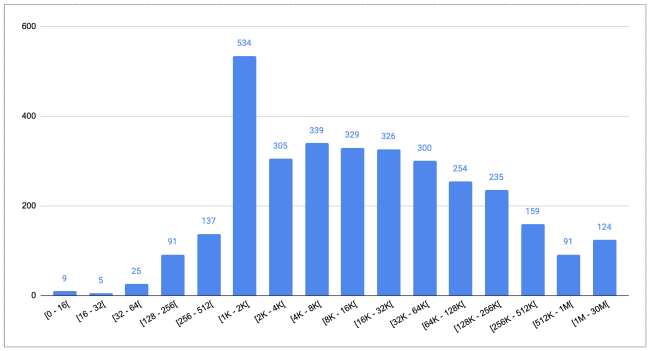

Fig. 3. File size distribution in the created file system.

C. Setup

In order to run all ransomware samples safely, our setup included two virtual environments, a Linux virtual machine, and an Android operating system emulator.

The Linux virtual machine corresponds to an Ubuntu 64-bit 22.04.3 with a disk size of 48.5 GB, 8GB of memory, and four cores from an Intel Core i9 2,3 GHz with eight cores. This machine created with the help of **VMWare Fusion** also includes the **Android-SDK** tools. Since this tool is based on the CRIBA pipeline it was necessary to set up our modified version of the pipeline, ready for analysis.

TABLE I
TOP 3 SYSCALL TYPES ISSUED PER RANSOMWARE FAMILY.

Syscall	*Family* Atank	Filcoder	Slocker
#1	`read` (66.16%)	`read` (26.79%)	`read` (49.43%)
#2	`write` (33.24%)	`write` (18.49%)	`write` (49.21%)
#3	`openat` (0.14%)	`openat` (16.64%)	`close` (0.33%)

TABLE II
EXECUTION TIME, PROCESS CREATION, ACCESSED FILES AND ISSUED SYSCALLS STATISTICS FOR THE RANSOMWARE FAMILIES.

Family	*Execution time (min)*	*Process* *PIDs*	*TIDs*	*Accesses* *Paths*	*Extensions*	*Syscalls* *Events*
Atank	14.53	2	30	418	109	4 763 662
Filecoder	2.12	1	40	2012	26	35 492
Slocker	1.13	1	27	438	121	252 153

```
   Name: device-root
 Device: pixel (Google)
   Path: /home/pi-test/.android/avd/device-root.avd
 Target: Default Android System Image
         Based on: Android API 34 Tag/ABI: default/x86_64
 Sdcard: 512 MB
```

Fig. 4. Android Virtual Device (AVD) specification.

Regarding the Android emulator, it was necessary to create an **Android Virtual Device** (AVD). To create this AVD, we focused on choosing an android version which allows us to have system root permissions. The characteristics of the AVD used are shown in Figure 4.

Once the AVD was working, a compiled version of *BPFroid* was copied into it, using the Android Debug Bridge (ADB) , which enabled the execution of this tracer in the emulator to capture of all events specified in a configuration file, **hooks.js**. After every ransomware sample execution, this emulator is restored to a clean state.

Therefore, the setup for the execution of our tool is complete.

V. EVALUATION RESULTS

In this section, we will discuss the analysis of the three ransomware variants *ATANK*, *Filecoder* and *Slocker*. We will begin with a brief overview, presenting generic statistics for each family. Subsequently, we are going to dive into more detail about specific features of a ransomware attack, such as data encryption.

Finally, we will provide conclusions on how our proposed tool contributed to these findings and compare the different ransomware families.

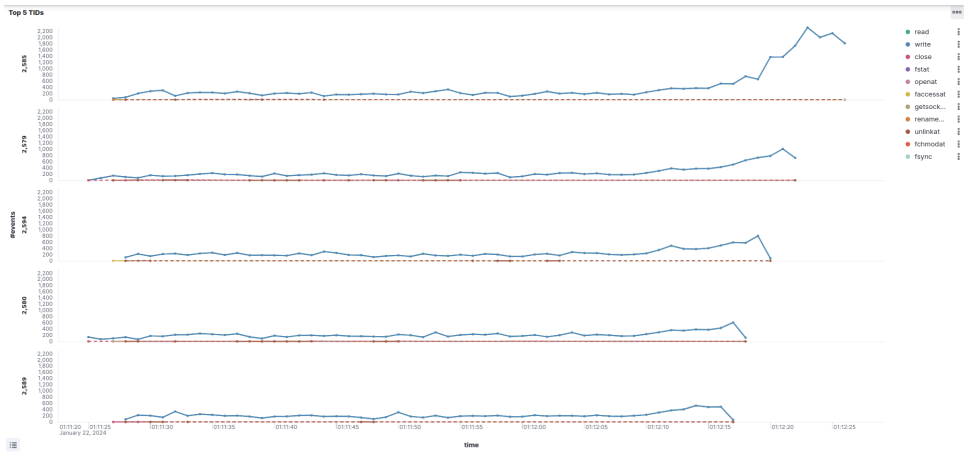

Fig. 5. Aggregated number of operations, separated by system call type, for the top five threads lauched by *Slocker*.

A. Overview

Given that we are employing dynamic analysis to understand ransomware I/O behavior and its interactions with the *Android* operating system, one of the first metrics to have in consideration is the amount and the type of system calls issued by the malicious software.

The tables II and I reveal some general metrics collected with our tool.

From table II, we can notice that there is a significant difference between the execution times of *ATANK* and the other two ransomware families analyzed. *ATANK* execution took $\approx$ 14.5 minutes while *Filecoder* and *Slocker* did in $\approx$ 2.1 and 1.1 minutes, respectively.

Another important observation to make about the general statistics present in table II is that across the three ransomware families there is a similarity in the number of processes and threads. All three ransomware families present a lot more threads than processes, only *ATANK* has two processes with the others having only one.

Table II also gives insights about the number of file paths and file extensions accessed by each ransomware family. Here we can see that the *Filecoder* accessed a large number of files (2012 files) when compared with the other two, however *Filecoder* did that while interacting with fewer extensions (26 file extensions).

Table I gives valuable information about the distribution of the majority of system calls executed by each ransomware family. Across all three families of ransomware the *read* system call is the most executed, and the *write* is the second most executed system call. The third most executed system call varies, for *ATANK* and *Filecoder* it is the *openat* and for the *Slocker* is the *close* system call. All of these system calls are closely related to the encryption employed by these malicious applications, since to encrypt a file one has to open it, read its contents, write the encrypted contents and close the file.

One final consideration is the distribution of the system calls executed through the threads of each ransomware. For the *ATANK* ransomware, when comparing the top 5 main threads in terms of number of system calls executed, one of the main five threads executed $\approx$ 99% of the total number of system calls distributed by these five threads. Similarly, for the *Filecoder* family $\approx$ 60% of the system calls are executed by one of the main five threads.

However, for the *Slocker* ransomware family a significant different distribution is observed, with 10 of the available 27 threads being involved in the encryption of files. The distribution of system calls executed among the top five threads is between $\approx$ 20% and $\approx$ 30% of system calls being executed by each thread. Figure 5 illustrates the similar distribution of workload among the top five threads of *Slocker*, as mentioned earlier.

In summary, these general statistics illustrate that ransomware families have different patterns of execution time, create a different amount of threads and processes, and access a different number of files.

All of these statistics were automatically generated using the *Unext* script, and explored through the *General Overview* and *Directory Transversal* dashboards.

TABLE III
TOP 2 SYSCALL SEQUENCES ISSUED PER FAMILY OVER A FILE.

Family	*Syscall sequence*	
	Main Sequence	*Second Sequence*
ATANK	OP→ST→RD→CL	OP→ST→WR→CL
Filecoder	OP→ST→RD→CL→OP→ST→CL	OP→ST→WR→CL
Slocker	OP→ST→RD→CL	OP→ST→CL→OP→ST→WR→CL

B. Encryption of Files

Table III shows the two most frequent sequences of system calls executed over a file by each analyzed ransomware family. To understand these sequences of system calls it is important to note, that if one system call is repeated consecutively it only appears as one system call in the represented sequences.

From Table III we can extract a distinctive pattern: the main sequence always includes a read system call (or multiple consecutive reads), and the second sequence always includes a write system call (or multiple consecutive writes).

TABLE IV
SYSCALL SEQUENCES ISSUED BY THE *Filecoder* FAMILY OVER TWO FILES.

Family	*Syscall sequence*	
	F189.cpp	*F189.cpp.sevem*
Filecoder	OP→ST→RD→CL→OP→ST→CL	OP→ST→WR→CL

These sequences of system calls are linked to the encryption of data, the main sequence is employed by each ransomware to read the user data from a file, and the second sequence is used to write the encrypted data.

Upon further investigation, we were able to find these system call sequences being executed by ransomware samples to encrypt user data, and another common pattern was found across all three ransomware variants: to encrypt a user file, the ransomware creates a new file with a distinctive extension, and starts reading data from the user file and writing the encrypted form of that data to the new file. When the encryption process is done all three ransomware samples delete the original file with an *unlinkat* system call.

Table IV shows the system call sequences present in Table III , being used by the *Filecoder* ransomware family to encrypt a file from our file dataset.

As mentioned above, all three ransomware families analyzed add an extension to the encrypted file. The characteristic extensions for *ATANK*, *Filecoder* and *Slocker* are respectively: `.xdrop`, `.seven` and `.勿卸 件解密加QQ1951275599bahk10018998`.

Apart from the general encryption behavior that is common across all three ransomware families, some distinctive patterns were discovered for each ransomware family in the encryption phase.

1) Atank: After reading all the contents of the original file and creating a new file to write the encrypted data, the original file is deleted with the following sequence of system calls (`close→faccessat→unlinkat→openat`). The ransomware tries to open the original file after it's been deleted, possibly to verify that if it was deleted correctly.

With more detailed analysis of the encryption process of *ATANK* a strange behavior was found, two threads were found opening the same file, both creating two files with the `.xdrop` extension, and both trying to read from the original file and writing the encrypted data to the newly created file. After closing the files, both threads try to remove the original user file using the `unlinkat` system call. One of the threads fails to delete the file (`unlinkat` returns a negative value) since the other had already removed that file.

These two threads try to encrypt multiple files simultaneously, but by the time the ransomware reaches our file dataset only one thread is encrypting files. This strange behavior on some of the files might produce irreversible damage to the files, since the two threads are accessing and writing to the same file concurrently.

Another interesting finding on the encryption process of the *ATANK* is that the thread encrypting a file reads 8 bytes at a time from the original file, and writes 16 bytes at a time to the encrypted data file.

2) Filecoder: This family follows some of the same patterns found in the *ATANK* ransomware. It starts by reading the contents of the original file then creates a new file with the `.seven` extension where the encrypted data is written. The difference here is found once the encryption is done, the *Filecoder* variant closes the original file, and then opens it again, issues an `fstat` system call and closes the file again.

Another observation of the *Filecoder*'s encryption process is that, during the dataset transversal, this ransomware opens a directory before it opens the files inside this directory. So for a directory `dir/` including two files `a.txt` and `b.txt` we will see an `open(dir)` system call and only after the `open(a.txt)` and `open(b.txt)` system calls that open the files within the directory.

3) Slocker: This is the only family of the three analyzed, where the encryption of files is done by multiple threads. Although the encryption is done by multiple threads, contrasting with the previous analyzed families, the encryption process is very similar to the pattern found in *ATANK* and *Filecoder*.

However, the *Slocker* family deviates from the typical encryption pattern once the encryption is done, which means the data is totally written to the file created by the ransomware. As the other two families, *Slocker* creates a new file to write the encrypted content, starts reading data from the original file and writes the encrypted form of this data to the created file. The files created by *Slocker* to write encrypted data have the extension `".!! 勿卸 件解密加QQ1951275599bahk10018998"` until the encryption process ends. When the encryption process ends, the file with the extension `".!! 勿卸 件解密加QQ1951275599bahk10018998"` is renamed and the extension changes to `".勿卸 件解密加QQ1951275599bahk10018998"`, possibly so the ransomware knows what files have been fully encrypted.

In summary, the actions applied by these ransomware families are very similar to each other, however they have distinctive features that distinguishes the three families. In general the encryption process involves reading the original user file, creating a new file with the same name but with a different extension and writing the encrypted version of the original data in this new file.

All of the observations discussed in this section were obtained with *FSysSeq*, *UNext* algorithms and observed with the *Syscall sequences*, and *File Names and Extensions* dashboards.

C. Dataset's File Selection

In some cases ransomware tries to select the files it will encrypt to accelerate the encryption process and cause the most damage possible to the user data. With that in mind, here we will present and discuss the observations made on the files selected by ransomware for encryption.

Starting by the *ATANK* sample, one interesting pattern found during analysis is that it never accessed any files deeper than the first folder of the dataset directory. Despite taking the

longest time to execute, it encrypted less files than the other two samples, accessing 181 files from our dataset.

Considering the *Filecoder* family, we start to observe more interesting patterns, that demonstrate a ransomware choosing specific files to encrypt. Upon analyzing the encrypted files by this variant of ransomware, we noticed that from all the extensions present in the our file dataset, this sample only accessed the following 10 extensions: `.cpp, .gif, .mml, .mp3, myd, .odb, .pdf, .pst, .txt, .wma`.

These file extensions are known to be associated with documents, images, audio and video files, *Microsoft* files, *MySQL* database files and others. So, by targeting this specific files on our file dataset, the ransomware is trying to provoke the most damage possible to sensitive and important user data.

On the *Slocker* variant, we observed that it only encrypts files on the dataset that are within a depth of three directories starting from the dataset root directory.

Summarizing, we can observe different behaviors of ransomware when choosing files to encrypt, with *Filecoder* trying to achieve more damage by targeting specific files and *Slocker* and *ATANK* only encrypting files up to a depth of directories, possibly to accelerate the encryption process.

D. Families Similarity

Figure 6 provides valuable information about the similarity and dissimilarity across the three ransomware families. Starting on the system calls, we can observe a significant similarity between the *ATANK* and *Slocker* families, this can be explained by the common system calls used by ransomware families (`openat, read, write, close, etc`) and because, as discussed in Section V-B, these two families have very similar patterns for reading data using the same system call sequence to read data (illustrated on Table III). In general, all three ransomware families share a significant amount of system call types and sequences, explaining the similarities above 50% across all families, with *Filecoder* being more dissimilar from the other two because it executes a bigger diversity of system calls.

The similarities between the three families when comparing the accessed file names and extensions are less significant. This is possibly linked to the encryption process, explained in Section V-B, where all three families create a new file with a new extension to write encrypted data, effectively accessing a new file name and a new extension, and given that each ransomware family has its unique extension that can lower the similarities found by the *tfidFam* algorithm.

E. Linux and Android Families

In this section we are going to give a brief comparison between the families analyzed with CRIBA [1] in the Linux Operating System (OS) , and the ones analyzed with our tool in the Android OS.

Starting with the creation of *Ransom Notes* which is described in CRIBA as a way for attacker to communicate to the victim an attack has occurred, every family analyzed in [1] created this files throughout the file system as files were encrypted. However, none of the ransomware families analyzed in this research displayed this kind of behavior, instead they would display a message on screen or change the device's background to inform the victim of the attack.

Moving to the encryption behavior, there's another significant difference between the *Android* and the *Linux* families. The *Linux* targeting families would read a user file and write the encrypted data to the same file and, when the encryption process was done, these families would change the name of the file adding a characteristic extension with the system call `rename`.

On the other hand, the *Android* analyzed families read the contents of a user file and create a new file, with a specific extension, writing the encrypted data to this new file and when the encryption process is completed they delete the original file with the system call `unlinkat`

Additionally, the *Filecoder*, *Slocker*, *AVOSLOCKER*, *RANSOMEXX*, and *DARKSIDE* use a single process, although the number of threads created is very distinct between *Linux* and *Android* families (higher in the Android families), as the file path accessed number (lower in the Android families).

VI. Related Work

As shown in Table V and VI, most work related to ransomware in Android devices focuses on the **detection** of ransomware, mostly through static and dynamic code analysis, [3]-[5], [9], [10], although others [6]-[8] use the tracing of system calls. These results are then used to classify a process as harmless or as a ransomware. Since the main purpose of these tools is the detection of ransomware not much is known about the behavior of the ransomware itself.

The reports generated by the system call tracers while a ransomware sample is being executed are too extensive to be analyzed without aid. The tool presented in this paper aims to automate the process of analyzing ransomware samples.

Besides that, it simply focuses on the tracing and visualizing I/O patterns generated by ransomware samples along with making the comparison of different ransomware families more efficient through the correlation algorithms. As such, this tool **does not** have the purpose of ransomware detection, it is instead a tool for **analyzing** the behavior ransomware samples. As stated many times above this tool is an extension for Android devices of CRIBA [1], as such it follows in the same principles.

VII. Conclusion

With the above research we present an extension to *CRIBA* [1], allowing it to perform analysis of *Android* ransomware samples, by extracting similar features as does extracted by CRIBA on *Linux* ransomware samples.

Our modified version of *CRIBA* is capable of non-intrusive collection of relevant I/O data from ransomware sample activity and is able to accommodate *CRIBA*'s correlation algorithms for the *Android* operating system, expanding the capabilities proposed by *CRIBA* to perform analysis on ransomware that targets a different operating system.

Fig. 6. Heatmaps comparing the families regarding the type of issued system calls, and accessed file extension and names.

TABLE V
RELATED WORK FOCUS.

Article	*Tracing*	*Detection*
[1]	✓	
[3]		✓
[4]		✓
[5]		✓
[6]	✓	
[7]		✓
[8]		✓
[9]		✓
[10]		✓

TABLE VI
RELATED WORK TYPE OF ANALYSIS PERFORMED.

Article	*System Calls*	*Static/Dynamic Code Analysis*
[1]	✓	
[3]		✓
[4]		✓
[5]		✓
[6]	✓	
[7]	✓	
[8]	✓	
[9]		✓
[10]		✓

We were also able to accommodate most of *CRIBA*'s visualization component to the *Android* ransomware samples extracted data, excluding the *Resource Usage* and *File Offset* dashboards

Our modified version of CRIBA to accommodate *Android* targeting ransomware is publicly available at **https://github.com/dsrhaslab/criba-android**.

As future work, the group considers that it would be important and meaningful to search for and test more ransomware samples and possibly introduce specific dashboards for the Android operating system.

REFERENCES

[1] T. Esteves, B. Pereira, R. Oliveira, J. Paulo, J. Marco, CRIBA: A Tool for Comprehensive Analysis of Cryptographic Ransomware's I/O Behavior. In: Symposium on Reliable Distributed Systems (SRDS 2023)

[2] "Mobile Ransomware" [Online]. Available: https://www.mimecast.com/content/mobile-ransomware/

[3] Andronio, N., Zanero, S., and Maggi, F., HelDroid: Dissecting and Detecting Mobile Ransomware. In: Research in Attacks, Intrusions, and Defenses. Ed. by Herbert Bos, Fabian Monrose, and Gregory Blanc. Cham: Springer International Publishing, 2015, pp. 382–404. isbn: 978-3-319-26362-5.

[4] Fereidooni, H. et al, ANASTASIA: ANdroid mAlware detection using STatic analySIs of Applications. In: 2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS). 2016, pp. 1–5. doi:10.1109/NTMS.2016.7792435.

[5] Gharib, A.and Ghorbani, A.,. DNA-Droid: A Real-Time Android Ransomware Detection Framework. In: Network and System Security. Ed. by Zheng Yan et al. Cham: Springer International Publishing, 2017, pp. 184–198. isbn:978-3-319-64701-2.

[6] Lin, Y. et al, Identifying android malicious repackaged applications by thread-grained system call sequences. In: Computers Security 39 (2013), pp. 340–350. issn: 0167-4048. doi: 10.1016/j.cose.2013.08.010 url: https://www.sciencedirect.com/science/article/pii/S0167404813001272

[7] Wu, D. et al, DroidMat: Android Malware Detection through Manifest and API Calls Tracing. In: 2012 Seventh Asia Joint Conference on Information Security. 2012, pp. 62–69. doi: 10.1109/AsiaJCIS.2012.18.

[8] Xiao, X. et al, Android malware detection based on system call sequences and LSTM. In: Multimedia Tools and Applications 78 (2019), pp. 3979–3999.

[9] Yang, T. et al, Automated Detection and Analysis for Android Ransomware. In: 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems. 2015, pp. 1338–1343. doi:10.1109/HPCC-CSS-ICESS.2015.39.

[10] Zhu, H. et al, DroidDet: Effective and robust detection of android malware using static analysis along with rotation forest model. In: Neurocomputing 272 (2018), pp. 638–646. issn: 0925-2312. doi:10.1016/j.neucom.2017.07.030. url https://www.sciencedirect.com/science/article/pii/S0925231217312870.

[11] "ATANK," 2024. [Online]. Available: https://malpedia.caad.fkie.fraunhofer.de/details/apk.atank

[12] "Investigadores da ESET descobrem nova família de ransomware Android," 2024. [Online]. Available: https://www.eset.com/pt/about/imprensa/artigos/investigadores-da-eset-descobrem-nova-familia-de-ransomware-android/

[13] "FileCoder," 2024. [Online].Available: https://malpedia.caad.fkie.fraunhofer.de/details/apk.filecoder

[14] "Android ransomware is back," 2024. [Online]. Available: https://www.welivesecurity.com/2019/07/29/android-ransomware-back/

[15] "SLocker Mobile Ransomware Starts Mimicking WannaCry," 2024. [Online]. Available: https://www.trendmicro.com/en_us/research/17/g/slocker-mobile-ransomware-starts-mimicking-wannacry.html

[16] "Android SLocker Variant Uses Coronavirus Scare to Take Android Hostage," 2024. [Online]. Available: https://www.bitdefender.com/blog/labs/android-slocker-variant-uses-coronavirus-scare-to-take-android-hostage/

[17] "SLocker Android Ransomware – In-Depth Analysis and Prevention Tips," 2024. [Online]. Available: https://sensorstechforum.com/slocker-android-ransomware/

[18] “SLocker Android Ransomware Resurfaces in Undetectable Form,” 2024. [Online]. Available: https://www.infosecurity-magazine.com/news/slocker-android-ransomware/

[19] Oz, Harun and Arış, Ahmet and Levi, Albert and Uluagac, Selcuk, A Survey on Ransomware: Evolution, Taxonomy, and Defense Solutions, 2021

[20] Esteves, T and Macedo, R and Oliveira, R and Paulo, J Diagnosing applications I/O bahavior through system call observability, 2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), Porto, Portugal 2023, pp. 1-8.

[21] Agman, Yaniv and Danny, Hendler BPFroid: Robust Real Time Android Malware Detection Framework, CoRR 2021

[22] Agrawal, Nitin and Arpaci-Dusseau, Andrea C. and Arpaci-Dusseau, Remzi H. Generating realistic impressions for file-system benchmarking, December 2009, Association for Computing Machinery