ESCAPADE: Encryption-type-ransomware: System Call based Pattern Detection

Christopher Chew Jun Wen[1 [0000−0003−2149−2641]],
Vimal Kumar[1 [0000−0002−4955−3058]],
Panos Patros[2 [0000−0002−1366−9411]],
Robi Malik[2]

[1] Department of Computer Science, University of Waikato
[2] Department of Software Engineering, University of Waikato
Hamilton, Aotearoa New Zealand
cc246@students.waikato.ac.nz, vimal.kumar@waikato.ac.nz
panos.patros@waikato.ac.nz, robi@waikato.ac.nz

Abstract. Encryption-type ransomware has risen in prominence lately as the go-to malware for threat actors aiming to compromise Android devices. In this paper, we present a ransomware detection technique based on behaviours observed in the system calls performed by the malware. We identify and present some common high-level system call behavioural patterns targeted at encryption-type ransomware and evaluate these patterns. We further present our repeatable and extensible methodology for extracting the system call log and patterns.

Keywords: Android · Behaviour · Patterns · Encryption-ransomware

1 Introduction

As mobile phones become more pervasive and entangled in our lives, we amass more and more private information on them, such as personal photos, credit card information, contacts, and private messages. As a result of this, mobile phones have become more or less a portable identification card in the modern day. This obviously makes them a target of malware attacks.

One of the more frequent types of attacks on mobile are ransomware attacks. As noted by SecureList's 2019 *Mobile malware evolution report* [6], there were a total of 60,176 installed ransomware packages on mobile devices in 2018 and the number went up to 68,362 in 2019. Additionally, malware authors employ cunning techniques to adapt, evolve and take advantage of current events. For example, CovidLocker [40, 23], a locker-type ransomware variant poses as a Covid tracking application to lure people into downloading and installing it. At times, the evolution results into completely different types of malware. For example, Black Rose Lucy [17, 30] which was originally an information-stealing malware distributed as a Service (MaaS) has now evolved to a encryption-type ransomware variant due to its encryption capabilities.

The effects of ransomware are widespread, posing a threat to both consumers and organisations. In June 2019, there were 1,625,351 consumers targeted by ransomware [29]. For businesses, ransomware is even more destructive. According to Sophos *The State of Ransomware 2020* report [43], 82% of the organisations surveyed in India were affected by ransomware in 2019. Ransomware heavily impacts businesses monetarily. The report [43] further stated that an average of $761,106 USD, globally was required to remediate the effects of an attack for an organisation. In July, 2020, Garmin suffered a ransomware attack [1] and paid an undisclosed ransom amount after being asked for $10 million. In another recent ransomware attack, Blackbaud [39] also paid an undisclosed amount to secure their data from being made public. This clearly shows that current defences are insufficient and there is a need for better protection of users from ransomware attacks, especially as mobile devices continue to grow in popularity.

In recent years, researchers have looked at system calls for dynamically analysing malware as it offers a balance between user-level and kernel-level analysis. User-level analysis is often unable to capture the behaviour of more sophisticated malware variants. Kernel-level offers more depth and resilience, however, the devised approaches can often lead to a complex design, thus leading to an over-fitted solution. Hence, our decision to focus on system calls. By using system call-level dynamic analysis, we aim to address the following research objectives:

- **RO1: Identify system call level behavioural patterns for encryption-type ransomware** While there have been recent works on pattern detection on system call logs [19, 26], none has focused on patterns produced by specific malware types. In this paper we aim to discover a set of common behavioural patterns for encryption-type ransomware, such as file encryption and tampering with user files through the reliance on the system call logs.
- **RO2: Evaluate the effectiveness of the behavioural patterns** We also evaluated the viability and efficacy of these patterns at detecting encryption-type ransomware behaviours from different families, to discover the shared common behaviour among encryption-type ransomware.
- **RO3: Create and make available, a dataset of system call logs of malware activity** We believe behaviour detection using system calls can be a useful technique for malware detection and analysis therefore we have made our dataset available for researchers to utilise in malware research.

2 Background and Related Work

In this section, we detail the evolution and improvements of Android security and its current state; following with an overview of different types of ransomware, and conclude with the different types of malware analysis techniques used throughout the years, and how our proposed approach can contribute to the existing area.

2.1 Android Security History

Since the introduction of Android—a mobile operating system—in 2008, there have been many updates and improvements to its security. In 2012, Bouncer was

released in an effort to deter the upsurge of Android malware in the preceding year [31]. Bouncer targeted pre-existing applications as well as new applications. The approach that Bouncer took was sandboxing [28], where applications were executed, and scanned for malware in an isolated environment on a cloud infrastructure; this was devoid of any access to the users' real data.

However, individuals quickly detected the vulnerabilities of Bouncer. Oliva Hou from Trend Micro [18] noted that researchers were able to acquire specific details of the runtime environment, such as the duration of Bouncer's testing phase (which was five minutes), and the phone contents used in the simulated environment (two photos, one contact and the Google account). These details could easily be exploited by attackers through the use of simple obfuscation techniques to avoid detection by Bouncer.

Bouncer was, therefore, not a sustainable security mechanism. A few years later in May 2017, a more robust approach known as Play Protect was introduced. In addition to the introduction of Play Protect, a security Application Programming Interface (API) called SafetyNet Verify Apps was introduced in September of the same year. This API aimed to address three key ideas: to help further protect users from malicious applications, determine if a user's device is protected by Play Protect, and prompt users to enable Play Protect if it is disabled.

2.2 Ransomware

Ransomware, a type of malware that holds the users' data for ransom—often requesting monetary payment—has been one of the more prevalent malware types, with 61,132,338 ransomware related threats detected in 2019, which was a visible increase compared to 55,470,005 in 2018 [32]. In addition to its prevalence, newer variants and iterations have appeared throughout the years adopting more sophisticated techniques, such as self-propagation, stronger encryption, and alternative infection vectors [37, 36].

With the growing numbers of mobile devices, ransomware, such as WannaLocker, SimpleLocker, Filecoder, and Black Rose Lucy [17, 30], have found their way into the mobile ecosystem. Ransomware are generally of two types: locker ransomware and encryption ransomware [33]. Locker-type ransomware traditionally displays a persistent screen that prevents the user from interacting with the rest of the system. This screen will often display the ransom note demanding monetary payment. On mobile devices, specifically Android, locker-type ransomware makes the application persistent by displaying a perpetual alert dialog or activity, or disabling interactions with the navigation bar [3]. Another technique used is altering users' lock screens, thus preventing access to their devices [2, 20].

Encryption-type ransomware are more destructive where the user's files are encrypted to prevent the user from accessing any of their data [21, 2]. Similar to locker-type ransomware, a ransom note is often displayed after the encryption phase has been completed. Typically for encryption-type ransomware, the process begins by scanning the user's personal directories, such as *Documents*,

and *Pictures* for files. Once the scanning phase has completed, the ransomware often identifies files containing specific extensions, such as, *.docx*, *.png*, and *.jpg* to encrypt. This method is normally used to speed up the encryption process, and efficiently determine the important user files to encrypt (i.e., the files most important to a user) [13]. For the encryption process, the data of the identified files are read, and written to a new encrypted file with an unknown file extension. The original file is then removed or overwritten [7].

2.3 Static Analysis

In static analysis, a malware analyst, observes the code of the given application and tries to determine if it is malicious or benign, and gains insight on its functionality without the necessity of executing the application. Static Analysis, however, has limited effectiveness when more sophisticated malware utilises advanced techniques, such as binary/code/control flow obfuscation, and polymorphic coding [12, 34, 9] to avoid detection.

AndroSimilar [10] and DroidMoss [47] adopted the idea of fuzzy hashing which compared similarities between the signatures generated. This produced a percentage of similarity with 100% being an exact match. This approach aimed to counteract the issue of code obfuscation and application repacking. However, AndroSimilar [10] produced high false negative rates (28%) when detecting unknown malware and considerably higher false negatives for the various methods of code obfuscation which consisted of method renaming (45%), junk method insertion (44%), goto obfuscation (43%), and string encryption (24%). DroidMoss's false negative rates were lower (10.7%). All the tested applications however, came from third-party app stores, whereas AndroSimiliar focused on both official Play store and third-party app stores.

2.4 Dynamic Analysis

In dynamic analysis, rather than observing the code, malicious applications are directly executed in an isolated environment and observed over time for malicious behaviour. This mitigates the core limitations of static analysis. Obfuscation is not an issue as dynamic analysis only observes the behaviour of the application at run-time. As a result of this, dynamic analysis is also capable of discovering new malware.

One of the dynamic analysis technique used is taint analysis, a method of observing data flow and tainting sensitive data paths that could potentially be used maliciously. TaintDroid [8] utilised this approach along with variable-level tracking of native methods within the Dalvik VM interpreter, which contained taint markings in a taint map. These taint markings were propagated through the Android Inter-Process Communication Binder, based on the defined data flow rules on how the application used the tainted data, to the untrusted application's taint map. If the untrusted application made a library call deemed as a taint sink (e.g., *network send*), then the application was marked as malicious.

In contrast, under our method of detection, we observe high-level behavioural patterns at a system call-level with each pattern classified in different levels of severity. This allows for more precise details regarding an application's behaviour and more flexibility with our detection model.

One of the dynamic technique is pattern detection at a system call level, which has often been used for kernel-level malware analysis. Works in [44, 26, 19] apply system call analysis on mobile operating systems such as Android. This approach is useful because system calls are able to determine the precise operations that occurred during the execution of an application/program, which can help identify malicious activities or behaviours.

One drawback, however, with system call monitoring is the size of the log files generated. Due to background processes—such as `clock_gettime()` that periodically record the system clock time—occurring in parallel with the core operations, the raw log size created from monitoring an application, is large.

Isohara et.al. [19] addressed this issue by filtering out unnecessary system calls. They achieved this by grouping system calls into specific categories and filtered processes unrelated to the application through the use of a process tree. For their detection phase, Isohara et.al. created 16 different patterns represented as regular expressions. These regular expressions utilised assistant keywords, which relate to specific strings such as, file paths or commands such as `su`.

The work of Isohara provides a good insight into pattern detection in system call logs using regular expressions. Our proposed approach improves on this notion by introducing a formalised and reproducible methodology for safely collecting and extracting system call logs from Android applications. This methodology allows us to create a comprehensive dataset that will enable researchers to better analyse encryption-type ransomware, devise new behavioural patterns, and evaluate the efficacy of their own approaches. Furthermore, our approach focuses on the concept of extensibility where we adopt a customisable multi-level filtering process to allow the abstraction of information within the system call logs. This creates a more human readable log thus making it easier for analyses. In addition to the multi-level filtering, we utilise a token-based approach for our patterns where each token is represented as a smaller sub-pattern.

SCSDroid [26] is a thread-grained behavioural pattern detection method on the system call level leveraging the Longest Common Subsequence (LCS) algorithm to extract potentially malicious patterns from system calls. The Bayes theorem is then utilised with these patterns to determine if an application was a Maliciously Repackaged Application (MRA) or a benign application.

The proposed approach of SCSDroid gives a good perspective of the viability of pattern detection used in malware detection. However, as noted in their conclusion, one of the limitations is its inability to detect unknown families that have not been acquired (i.e., trained). In comparison, in our approach, we develop behavioural patterns to match high-level common behaviour based on a range of ransomware families. This allows us to capture a broader range of behavioural patterns as opposed to family-specific patterns. Furthermore, we demonstrate

a reproducible experimental testbed for identify malicious patterns on Android applications.

3 Methodology

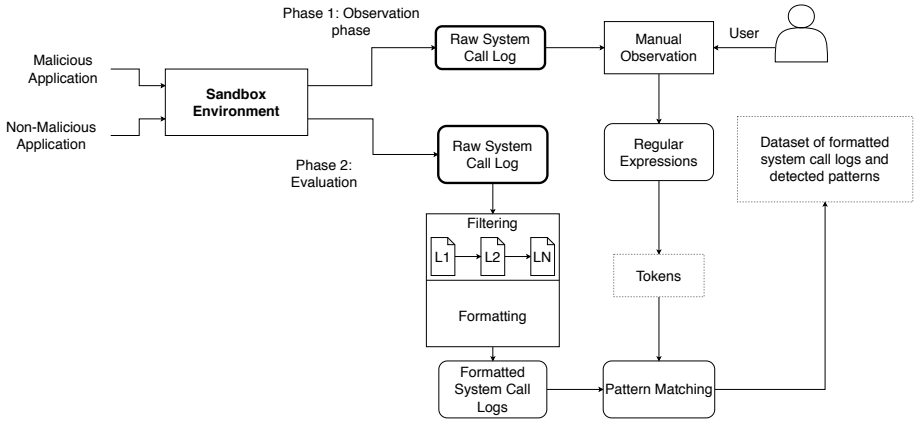

Fig. 1. Methodology process overview

Figure 1 provides an overview of the process followed in this work. The sandbox environment component is our run-time environment where applications are examined; this environment is described in more detail in section 3.1. The output of the sandbox environment splits into two phases. The first phase is the observation phase where applications are observed for their behaviour during runtime. After which, regular expressions are created based on the benign and malicious behaviours observed during that phase. These regular expressions are then converted into our token representation for pattern matching.

These tokens are used in our second phase, labelled as *Evaluation*. This phase starts with the extraction of the raw system call logs (similar to the observation phase), then applies multiple layers of filtering to abstract and remove repetitive or unrelated system calls. After which, the filtered log is formatted for pattern matching using our created tokens. This process is repeated for all unique variants containing a unique hash—also known as a sample—resulting into the final dataset, which contains the formatted system call logs and detected patterns.

The following subsections extensively describe our methodology of collecting and formatting system call logs for detection of malware in more detail. The methodology proposed enables researchers to utilise a streamlined and reproducible approach to safely extract system call logs for effective pattern-based malware detection.

3.1 System Call Log Collection

The first part of our approach is the collection of system call logs. To achieve this, we devised an automatic process of installing applications and tracing system call logs. The environment we used was a Google Pixel 2 emulator running API level 24, created from Android Studio. To automate the process of installing applications and starting applications, we used Android Debug Bridge [15] (ADB) and Android Monkey [16], a program used for generating events on an application. To acquire the system call logs, we ran Strace [25], a command line tool originally utilised in Linux, to extract and capture the system calls from each application during runtime. The parent process (`Zygote`) was traced to ensure we capture all behaviours produced by the applications.

During the observation phase, we noticed that Android ransomware often prompts for admin privileges. Hence, we automatically accepted the requested permissions for each application. Additionally, to simulate a real-user experience, we used Android Monkey to insert events periodically during the application's runtime. This is described in more detail in section 3.2.

3.2 Detection of Behavioural Patterns

To acquire a set of high-level common behavioural patterns for encryption-type ransomware, we conducted an evaluation with 10 encryption-type ransomware samples from five families obtained from CICAndMal2017 [24] and Koodous [22]. Each application was executed 10 times and manually observed during runtime to comprehensively acquire their malicious behaviour. Additionally, 10 benign samples were also analysed to observe the differences in behaviour.

The five ransomware families used for our pattern observation phase consisted of: WannaLocker, DoubleLocker, SimpleLocker, Filecoder, and Wipelocker. All samples were evaluated from each of these families to acquire our common high-level behaviours. The samples used within our pattern observations phase are excluded from our dataset of malicious applications to avoid any potential bias within our evaluation phase in Section 4. During the observation phase, we were able to discover 12 behavioural patterns. We classified the behavioural patterns in three categories, five of these patterns are classified as *Malicious*, four are classified as *Suspicious*, and three are *General* behavioural patterns.

Table 3 in the appendix shows the 12 patterns we identified and created. Within this table, the use of `>` is to concatenate each token. Additionally, Table 2 in the appendix provides each token's objective.

3.3 Pattern Acquisition and Classification

Our method of acquiring the patterns was based on our deduction in the observation phase. This was achieved by going through each application and identifying malicious (or potentially malicious) behaviour and its respective high-level system call counterpart via the captured log. For example, if an application encrypted the user's files then the high-level behaviour at a system call level would

translate to `openat` - open user file, one or multiple `read` system calls, `openat` - create new encrypted file, one or multiple `write` system calls.

We aim to observe common high-level behavioural patterns specifically focusing on encryption-type ransomware. However, not all captured behavioural patterns correlate to malicious behaviour.

For example, consider the creation of a socket to connect to an external URL to transfer specific resources. This type of behaviour occurs in both benign and malicious applications. However, the usage will differ. A malicious application often uses that connection to contact a Command and Control (C&C) server [38] to download the payload, whereas a benign application would use the connection to download resources; often occurring in applications requiring frequent updates, such as online mobile games, or linking accounts such as social media accounts. Therefore, to aid in distinguishing the behaviour of patterns, we created a classification to better represent the patterns detected.

Patterns in the *Malicious* category are explicitly classified as malicious behaviours. Applications that contain *Malicious* patterns contain malicious segments that resemble behaviour of encryption-type ransomware.

Behavioural patterns classified in the *Suspicious* category are deemed as potentially malicious. These types of patterns can lead to malicious behaviour. However, the behaviour by itself does not indicate any malice.

Patterns in the *General* category are common benign behaviours that exist in malicious and benign applications with low indication of malicious behaviour.

Note: *Suspicious* and *General* patterns are not used in our evaluations for this paper. These patterns were primarily identified and created to aid future detection systems that utilise common high-level behaviour. Furthermore, encryption-type ransomware exhibits distinct malicious behavioural patterns unlike other types of malware, such as Adware and Trojans, where the malicious behaviours are not always immediately evident. The inclusion of these two pattern categories will be more beneficial in those types of malware.

Malicious Patterns Our first malicious pattern observed from the logs was related to file renaming and unlinking within the user's main directory (*Rename & Unlink File*). This behaviour was observed in the WannaLocker/Slocker sample, which renamed the initial encrypted file using an unknown file extension. Once the file extension has changed, the ransomware proceeded to unlink the user's original file that was related to the encrypted file. We only looked for this pattern in files within the user directory or external directory (SDcard) as these directories are the points of interest for encryption-type ransomware due to the importance of the files residing within them (often important to the users, such as photos, notes, and other important documents, but not required for the system to work) [41]. Additionally, during our observation phase, the folder *Android* was also within the user directory. Hence, we added an additional condition to exclude that specific directory. The main system call sequences observed, began with `renameat`, followed by an `fstat`, which always occurred before an `unlinkat` operation.

The next malicious pattern from our observations was unlinking of users' files. From our analysis, we were able to find consistent occurrences of this pattern in the ransomware samples and there were no traces of this pattern occurring in the 10 benign samples during our observation phase. The sequence for this pattern began with an `unlinkat` system call followed by the location of the user directory, and the type of file removed.

Another malicious behavioural pattern discovered was the creation of files with unknown file extensions within the user's main directory (*Unknown File Ext Created*). From the different samples observed, this was a prevalent behaviour for encryption-type ransomware where a new file was created to hold the encrypted data of the original user's file. This encrypted file was in a nonstandard file extension and the file name consisted of the original file's name including its original file extension. The main sequence of tokens for this pattern started with an `openat` system call followed by the user directory token, then searched for any files created not matching a regular file extension type.

The last two common malicious patterns discovered were reading of user files and writing to a file with an unknown file extension. These two behavioural patterns represented the encryption segment of a encryption-type ransomware. This was a common behaviour that occurred in all of our ransomware logs.

The first pattern that represents the encryption component is *Read User File*. This pattern focuses on capturing the behaviour of applications reading three times from a file within the user directory. From our observation phase, some of the malicious variants observed read the contents of files within the user directory over multiple `read` operations in a specific block size, unlike the benign samples, which read the file contents in one single block. Hence, the inclusion of three read operations; this is to filter out apparent benign applications. The sequence of this pattern begins with an `openat` system call followed by the location of the user directory then three `read` operations.

The second pattern of the encryption component is *Write File Unknown Extension*. This pattern observed the behaviour of applications writing data to a newly created file with an unknown file extension. This pattern, together with *Read User File*, represented the encryption behaviour seen from the various encryption-type ransomware in our observation phase. The sequence of tokens for this pattern starts with an `openat` system call with the user directory specified, followed by a file created with an unknown file extension and a `write` operation.

Suspicious Patterns The first suspicious pattern we noted was applications making connections to an external IPv4 address. This could mean the malicious app making connection to a C&C server, however, this can also just be a non-malicious app connecting to the outside internet. We, therefore classified as suspicious but not malicious. The sequence of this pattern observes any `connect` system call followed by an IPv4 address.

Another suspicious behavioural pattern was directory searching. This behaviour is traditionally exhibited by encryption-type ransomware, which searches for user files within the device to encrypt. However, this behaviour does not in-

herently signify malicious behaviour as there are benign applications that can exhibit the same behaviour, such as cache-cleaning applications. The sequence consists of an `openat` system call and a directory name, then a sequence of `getdents64` (system call for getting directory entries), ending with a `close`.

The next notable suspicious pattern discovered in some ransomware samples, was the creation of an obfuscated file. This file had no file extension and the content contained an external URL. Similar to the first suspicious pattern, we were unable to validate the legitimacy of the URL address. However, many of the ransomware logs observed, contained URL addresses that were related to C&C servers. The sequence of tokens for this pattern comprised an `openat` system call, then any obfuscated file name with no file extension, followed by a `pwrite64` operation with the contents matching any URL address.

The last suspicious pattern was the acquisition of network information via `getaddrinfo`. From our observations, the majority of the ransomware logs attempted to acquire network information, such as socket addresses, and socket types from unknown domains via `getaddrinfo`. However, this does not necessarily indicate malice as we discovered legitimate trusted domains in benign applications such as, `googleadservices`. This pattern began by matching a `socket` system call followed by the subsequent sequence of system calls: `setsockopt`, `connect`, `fnctl64`, `fstat64`, and concluding with a match for a URL address.

General Patterns There are three patterns in the *General* category. These patterns consist of simple file I/O operations, read and write file behaviour, and generic file unlinking (targets known file extensions in any directory location), such as temporary files (`.tmp`, `_tmp`), backup files (`.bak`), or File locks (`.flock`).

The patterns in the *General* category aim to provide more detailed information regarding an application's behaviour regardless of whether the application is malicious or benign.

For *File Read*, and *File Write*, the sequence started with an `openat` system call, then a read/write operation. The last pattern *Generic File Unlink* matches any *unlinkat* system call with any file matching *.flock*, *.xml*, *.bak*, or *.db-wal*.

Our first research objective was to identify common high-level behavioural patterns for encryption-type ransomware at a system call level. To satisfy this requirement, we identified 12 different behavioural patterns, represented as tokens, and categorised them into different severity levels based on our observations. By utilising these patterns with our methodology for collecting and extracting system calls, we were able to devise a novel meta language for detecting malicious encryption-type ransomware behavioural patterns. This approach presents an easily reproducible testbed for researchers to evaluate potentially malicious applications, and create behavioural patterns based on system call logs.

4 Evaluation

This section details the method of evaluation used for our proposed method, which includes our testing environment, dataset acquisition, evaluation method,

evaluation of detected patterns identified in a set of encryption-type ransomware and benign applications. These evaluations are conducted to identify shared commonalities that exists between different encryption-type ransomware families as well as assessing the viability against a benign set of applications. Thus, validating our second research objective.

4.1 Dataset Acquisition

To acquire our dataset of encryption-type ransomware samples, we retrieved the hash or package name publicised from established anti-virus vendors, such as Avast [5] and ESET [46], and relevant search tags, such as family name from Koodous [22]; then we manually verified each malicious application against VirusTotal [42] before downloading the APK from Koodous [22].

As our focus was encryption-type Android ransomware, it was difficult to acquire a large sample size due to the distinctive category. However, we managed to acquire 500 distinct samples to assess our behavioural patterns. Out of that sample size, 213 applications exhibited encryption-type ransomware behaviours. Applications that did not encrypt our files were manually re-evaluated to thoroughly examine the potential cause of failure. From the re-evaluation, we discovered 18 samples that required manual interaction to enable the encryption component. These 18 samples were included in the 213 malicious samples.

From our observations via manual reevaluation, we noticed several factors that caused the failure of encryption. This was likely due to some of the samples requiring a connection to a C&C server that was no longer active. Additionally, some of the applications were installed and crashed upon start-up; thus, preventing the malicious code from executing. Furthermore, there were applications that failed to install on the emulator due to issues, such as a missing manifest file.

As part of our contribution in this paper, we produced a dataset of system call logs collected from our evaluation of 213 encryption-type ransomware and made it publicly available online[3]. This can also enable other works surrounding system call pattern detection to evaluate their own approaches, or expand and develop new behavioural patterns from their own observations.

Alongside our malicious dataset of encryption-type ransomware, we acquired 502 benign applications from APKPure [4] to evaluate the efficacy of our approach. Two of these samples were cache cleaning applications. These two special samples were included as these types of applications closely resembled the high-level behaviours of encryption-type ransomware, specifically the behaviour of removing user files. These two applications were tested separately with manual interaction to ensure we captured the cleaning process.

4.2 Evaluation Method

We ran each application for two minutes using our automation script. Once all the system calls were extracted, we put them through our detection program, and calculated the number of all detected patterns for the different severity levels.

[3] https://crow.org.nz/tools/ransomwaresystemcalldataset

For our ransomware dataset, we identified different malicious patterns for all six ransomware families. Any application whose log contained a match for at least one malicious pattern was classified as malicious. Any falsely identified malicious patterns were noted within this evaluation.

4.3 Detected Patterns

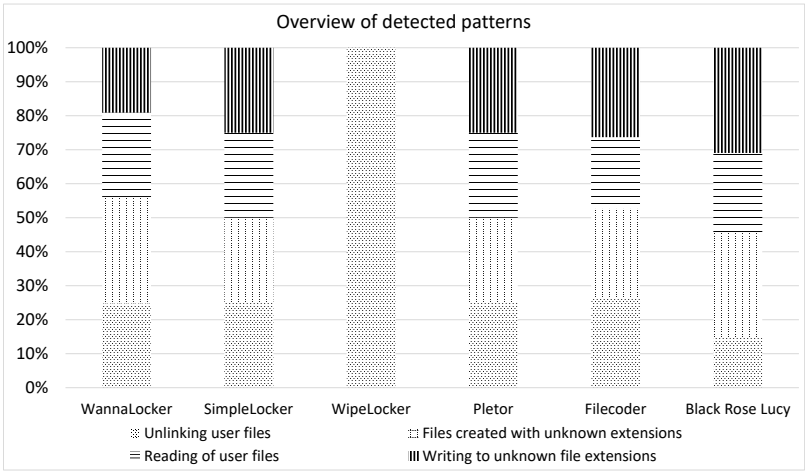

Fig. 2. Overall results of each detected pattern

This section details our evaluation of the six different encryption-type ransomware families. Figure 2 illustrates the results of our evaluation for the malicious dataset. The following paragraphs provides a thorough elaboration of each family and their discovered patterns.

WannaLocker: For WannaLocker we acquired 51 samples; from these 51 samples, we detected 850 malicious patterns. *Unlinking User Files* and *Read User File* were detected 211 times each, *Write File Unknown Extension* was detected 162 times, and 266 patterns were *Unknown File Ext Created*.

SimpleLocker: We acquired 64 encryption-type ransomware samples of SimpleLocker, and out of these 64 samples, we were able to discover 1280 malicious patterns. Within the 1280, we detected an even split of 320 *Unlinking User Files*, *Unknown File Ext Created*, *Read User File*, and *Write File Unknown Extension*. However, we were unable to detect any *Rename & Unlink File* as this behaviour did not occur in any of the samples.

WipeLocker: For WipeLocker, 70 samples were acquired for evaluation. All 70 samples detected 5 *Unlinking User Files* with no other malicious patterns detected. This led to a total of 350 malicious patterns detected. Although WipeLocker did not indicate any behaviour of file encryption (even after re-evaluating

the applications manually), the attributes this family exhibited were similar to encryption-type ransomware such as, the detected pattern of unlinking user files, and directory searching (a suspicious pattern, which we were able to detect a total of 733 occurrences); hence, the inclusion of this family within our evaluation.
Pletor: We were able to acquire six samples from Pletor and from those six samples, a total of 120 malicious patterns were discovered; with even split of 30 between *Unlinking User Files*, *Read User File*, and *Write File Unknown Extension*.
FileCoder: For Filecoder, we were only able to acquire five samples. However, out of these five samples, we were able to discover 95 malicious samples. From those 95 patterns, *Unlinking User Files* and *Write File Unknown Extension* were split evenly with 25 total detected samples each, whilst the remaining 20 were classified as *Read User File*.
Black Rose Lucy: For Black Rose Lucy, we acquired 17 samples for our evaluation. Out of these 17 samples, 307 malicious patterns were detected. Out of these, we identified 45 instances of *Unlinking User Files*, 95 *Unknown File Ext Created*, 72 *Read User File*, and 95 *Write File Unknown Extension*.

Unlike other encryption-type ransomware, we noticed that Black Rose Lucy specifically targeted the user's external storage directory (`/sdcard/`) rather than the user's internal directory during our evaluation. Additionally, we required manual interaction with each of the samples as Android Monkey was unable to detect the package name of the application.

One of the research objective was to evaluate the viability of the devised patterns for behavioural pattern detection against a set of encryption-type ransomware. Within our evaluation we were unable to discover any patterns for *Rename & Unlink File* as this behaviour was likely tied to a specific variant of WannaLocker or SimpleLocker. However, from the overall results of our evaluation, seen in Figure 2, there is clear indication of shared common behaviour among encryption-type ransomware regardless of the family, with the only exception of WipeLocker, which is known to only remove user files. Through the patterns detected and shared commonalities identified, we have validated the viability of these common high-level behavioural patterns for detection of encryption-type ransomware. This further reinforces the conclusion of the first research objective.

4.4 Benign Applications Results

Table 1 contains a summary of our results where we evaluated the efficacy of our patterns on our benign dataset. The *Percentage* column provides the percentages of true negatives and false positives detected for all benign samples evaluated. The *Sample Size* column denotes the numerical value of true negatives and false positive samples detected.

To evaluate the efficacy and viability of our patterns, we tested our approach on a dataset consisting of 502 benign applications. Two of those are the cache-cleaning applications discussed separately below. Out of the other 500 benign applications, we encountered six falsely classified applications. This was due to

Table 1. Summary of all benign applications evaluated

Benign Samples	Percentage	Sample Size
True Negative	98.6%	495
False Positive	1.4%	7

a mismatch of four different patterns, specifically, *Unlinking User Files*, *Read User File*, *Unknown File Ext Created*, and *Write File Unknown Extension*.

For our pattern matching results, two applications incorrectly matched *Read User File*; this was due to the applications creating and reading application related files within the user directory, such as `dslv_state.txt`. To mitigate this issue, `openat` system calls with the flag `O_CREAT` could be excluded. This would ensure that only user created files were captured within this pattern.

The third benign application that was falsely classified incorrectly matched the patterns *Unlinking User Files* and *Read User File*, due to the application creating and utilising temporary files within the user directory. This was one of the drawbacks of capturing high-level behaviour. For most cases, these patterns would capture unlinking of user created files and existing user file access and reads, which is a behaviour often exhibited by encryption-type ransomware as part of the file encryption process. However, in the case of an application creating and utilising a file within the user directory, it would be classified as a false positive. A potential solution is to exclude files created by the application within the user directory, as previously suggested, or reduce and combine the behavioural patterns related to file encryption.

The last three benign applications falsely classified were incorrectly matching two behavioural patterns: *Unknown File Ext Created* and *Write File Unknown Extension*. These patterns were falsely classified due to the applications creating an application folder within the user directory and a file with an unknown file extension within the application folder.

Similar to the proposed solution for the aforementioned third application, combining behavioural patterns related to file encryption could provide a more accurate representation. Alternatively, the pattern could be altered to only check for primary directories (i.e., directories not created by the application), such as *Photos*, *Documents*, and *Downloads*.

Cache Cleaning Applications For the two cache cleaning applications, one of them resulted in a false positive. There were four total malicious patterns matched and all four of those patterns were linked to *Read User File*.

From the examination of the patterns file and system call log file, these four patterns were deemed as irregular behaviour as it was unusual for a benign application to be reading the contents of user created files (i.e., pre-existing files, not created by the application).

5 Conclusions

In this paper, we identified and explored three core research objectives. The first research objective was to identify system call level behavioural patterns for encryption-type ransomware. To achieve this, we presented an extensive methodology for collecting and identifying behavioural patterns at a system call level. Using this methodology, we were able to discover a set of common high-level behavioural patterns at a system call level.

Our second research objective evaluates the effectiveness of the behavioural patterns identified. This was achieved by creating 12 behavioural patterns for detecting encryption-type ransomware. Consequently, we were able to evaluate these patterns against a set of encryption-type ransomware to identify shared commonalities between different families using pattern matching.

By utilising our methodology and behavioural patterns, we developed a publicly available dataset of formatted system call logs of encryption-type ransomware, which satisfied our third research objective. This dataset was created to contribute to the area of system call pattern detection that can be utilised in the future for purposes, such as evaluation or discovery of additional patterns.

5.1 Discussion and Future Work

There are limitations to the work presented in this paper. One of the limitations relates to the generation of regular expressions. Currently, we require manual observation and interaction to create regular expressions. This process can often be tedious and difficult. As we continue to develop our approach, we intend to automate this process.

Another limitation is our approach of identifying behavioural patterns. The patterns identified were based on our own observations from various applications. As a result of this, there may have been some behaviours that were not captured. In future, we would like to introduce a more formalised and robust methodology of identifying behavioural patterns at a system call level ensuring that all behaviours are captured without any uncertainty.

Additionally, we intend to introduce more behavioural patterns capable of detecting other types of malware, such as Backdoors and Trojans, which were identified as two of the most prominent types of infections for third-party apps [14]. This enables us to expand our dataset and evaluate the efficacy of our methodology on a larger sample size consisting of different types of malware.

Similar to static analysis, dynamic analysis suffers in keeping up with malware developers' avoidance techniques. Dynamic analysis generally relies on an isolated environment, most commonly through the use of Virtual Machines (VMs) to observe malicious applications. However, more sophisticated malware have processes in place to detect virtual environments [11, 45]. There are several tests that a malware may perform to check if the environment is emulated. One of which is registry checks; whenever a virtual machine is spawned in, a new registry entry is inserted. Another possible way how malware can detect

virtual environments is by checking the MAC address as certain virtual environment software, like VMware produces distinct MAC address prefixes for the VM. Specifically, for VMware, the following MAC address prefix can be 00-05-69, 00-0c-29, 00-1c-14 or 00-50-56 [35]. In practice, this holds true as the authors of ANDRUBIS [27] identified this as one of their limitations. Similarly, CopperDroid [44] also noted in their related works section that DroidBox, a malware detection tool using taint analysis, also suffers from the same issue.

The aforementioned issue highlights the inadequacy of heavily relying on conventional VM-based dynamic analysis. Thus, to prevent this issue we adopted a deeper level of analysis through the utilisation of system call log data to identify malicious behaviour. Although we evaluated our approach on an emulated environment, our approach is still applicable as long as the system call log is producible. As part of our future work, we are working on a system that enables efficient capturing of system calls in real time on a real user device utilising the technique shown in this paper. This alleviates the reliance of a virtualised environment, which most dynamic analysis techniques utilise.

As previously mentioned in Section 3.3, *Suspicious* and *General* patterns were not utilised in our evaluations. However, these patterns were still identified and created to lead into future work. These patterns can be expanded to create a more robust real-time malware detection model for Android devices or aid current and future anti-malware applications in detecting and deterring malware.

References

1. Abrams, L.: Confirmed: Garmin received decryptor for WastedLocker ransomware (2020), `https://www.bleepingcomputer.com/news/security/confirmed-garmin-received-decryptor-for-wastedlocker-ransomware/`
2. Al-rimy, B.A.S., Maarof, M.A., Shaid, S.Z.M.: Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions. Computers & Security **74**, 144–166 (2018)
3. Andronio, N., Zanero, S., Maggi, F.: Heldroid: Dissecting and detecting mobile ransomware. In: International Symposium on Recent Advances in Intrusion Detection. pp. 382–404. Springer (2015)
4. APKPure: Benign dataset (nd), `https://apkpure.com/`
5. Avast: Avast blog (2020), `https://blog.avast.com/`
6. Chebyshev, V.: Mobile malware evolution 2018. SecureList (2019, March 16), `https://securelist.com/mobile-malware-evolution-2018/89689/statistics`
7. Chen, J., Wang, C., Zhao, Z., Chen, K., Du, R., Ahn, G.J.: Uncovering the face of Android ransomware: Characterization and real-time detection. IEEE Transactions on Information Forensics and Security **13**(5), 1286–1300 (2017)
8. Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B.G., Cox, L.P., Jung, J., McDaniel, P., Sheth, A.N.: TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. ACM Transactions on Computer Systems (TOCS) **32**(2), 5 (2014)
9. Faruki, P., Bharmal, A., Laxmi, V., Ganmoor, V., Gaur, M.S., Conti, M., Rajarajan, M.: Android security: a survey of issues, malware penetration, and defenses. IEEE communications surveys & tutorials **17**(2), 998–1022 (2014)

10. Faruki, P., Laxmi, V., Bharmal, A., Gaur, M.S., Ganmoor, V.: AndroSimilar: Robust signature for detecting variants of Android malware. Journal of Information Security and Applications **22**, 66–80 (2015)
11. Gadhiya, S., Bhavsar, K.: Techniques for malware analysis. International Journal of Advanced Research in Computer Science and Software Engineering **3**(4) (2013)
12. Gandotra, E., Bansal, D., Sofat, S.: Malware analysis and classification: A survey. Journal of Information Security **5**(02), 56 (2014)
13. Gazet, A.: Comparative analysis of various ransomware virii. Journal in computer virology **6**(1), 77–90 (2010)
14. Google: Android security 2018 year in review (2019), `https://source.android.com/security/reports/Google_Android_Security\2018_Report_Final.pdf`
15. Google: Android Debug Bridge (adb) (2020), `https://developer.android.com/studio/command-line/adb`
16. Google: UI/application exerciser monkey (2020), `https://developer.android.com/studio/test/monkey`
17. Goud, N., Goud, N.G., Goud, N., Goud, N., Insiders, C., Acquisitions, M.., Insiders, C.: Black Rose Lucy ransomware attack on Android devices (Apr 2020), `https://www.cybersecurity-insiders.com/black-rose-lucy-ransomware-attack-on-android-devices/`
18. Hou, O.: A look at Google Bouncer [blog post] (2012, July 20), `https://blog.trendmicro.com/trendlabs-security-intelligence/a-look-at-google-bouncer/`
19. Isohara, T., Takemori, K., Kubota, A.: Kernel-based behavior analysis for Android malware detection. In: 2011 Seventh International Conference on Computational Intelligence and Security. pp. 1011–1015. IEEE (2011)
20. Kanwal, M., Thakur, S.: An app based on static analysis for Android ransomware. In: 2017 International Conference on Computing, Communication and Automation (ICCCA). pp. 813–818. IEEE (2017)
21. Kok, S., Abdullah, A., Jhanjhi, N., Supramaniam, M.: Ransomware, threat and detection techniques: A review. Int. J. Computer Science and Network Security **19**(2), 136 (2019)
22. Koodous: Malicious dataset (nd), `https://koodous.com/`
23. Lance, W.: CovidLock ransomware exploits coronavirus with malicious Android app. TechRepublic (2020, March 17), `https://www.techrepublic.com/article/covidlock-ransomware-exploits-coronavirus-with-malicious-android-app/`
24. Lashkari, A.H., Kadir, A.F.A., Taheri, L., Ghorbani, A.A.: Toward developing a systematic approach to generate benchmark Android malware datasets and classification. In: 2018 International Carnahan Conference on Security Technology (ICCST). pp. 1–7. IEEE (2018)
25. Levin, D.V.: Strace (2020), `https://strace.io/`
26. Lin, Y.D., Lai, Y.C., Chen, C.H., Tsai, H.C.: Identifying Android malicious repackaged applications by thread-grained system call sequences. computers & security **39**, 340–350 (2013)
27. Lindorfer, M., Neugschwandtner, M., Weichselbaum, L., Fratantonio, Y., Van Der Veen, V., Platzer, C.: Andrubis–1,000,000 apps later: A view on current Android malware behaviors. In: 2014 third international workshop on building analysis datasets and gathering experience returns for security (BADGERS). pp. 3–17. IEEE (2014)
28. Lockheimer, H.: Android and security [blog post] (2012, February 2), `https://googlemobile.blogspot.com/2012/02/android-and-security.html`

29. Malwarebytes: CTNT report cybercrime tactics and techniques: Ransomware retrospective (2020), https://resources.malwarebytes.com/files/2019/08/CTNT-2019-Ransomware_August_FINAL.pdf
30. Mana, O., Hazum, A., Melnykov, B., Kuperman, L.: Lucy's back: Ransomware goes mobile (Apr 2020), https://research.checkpoint.com/2020/lucys-back-ransomware-goes-mobile/
31. Micro, T.: Behind the Android menace: Malicious apps—TrendLabs security intelligence blog, https://blog.trendmicro.com/trendlabs-security-intelligence/infographic-behind-the-android-menace-malicious-apps
32. Micro, T.: The sprawling reach of complex threats (2020), https://www.trendmicro.com/vinfo/us/security/research-and-analysis/threat-reports/roundup/the-sprawling-reach-of-complex-threats
33. Mohammad, A.H.: Ransomware evolution, growth and recommendation for detection. Modern Applied Science **14**(3) (2020)
34. Moser, A., Kruegel, C., Kirda, E.: Limits of static analysis for malware detection. In: Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007). pp. 421–430. IEEE (2007)
35. Ninja, S.: How malware detects virtualized environment (and its countermeasures) (2016), https://resources.infosecinstitute.com/how-malware-detects-virtualized-environment
36. O'Kane, P., Sezer, S., Carlin, D.: Evolution of ransomware. IET Networks **7**(5), 321–327 (2018)
37. Richardson, R., North, M.M.: Ransomware: Evolution, mitigation and prevention. International Management Review **13**(1), 10 (2017)
38. Robert Lipovský, Lukáš Štefanko, G.B.: Labour party is latest victim of Blackbaud ransomware attack (2016), https://www.welivesecurity.com/wp-content/uploads/2016/02/Rise_of_Android_Ransomware.pdf
39. Scroxton, A.: Labour party is latest victim of Blackbaud ransomware attack (2020), https://www.computerweekly.com/news/252487002/Labour-Party-is-latest-victim-of-Blackbaud-ransomware-attack
40. Shivang, D.: CovidLock: Android ransomware walkthrough and unlocking routine (2020, March 16), https://www.zscaler.com/blogs/research/covidlock-android-ransomware-walkthrough-and-unlocking-routine
41. Song, S., Kim, B., Lee, S.: The effective ransomware prevention technique using process monitoring on Android platform. Mobile Information Systems **2016** (2016)
42. Sood, G.: virustotal: R Client for the virustotal API (2017), r package version 0.2.1
43. Sophos: The state of ransomware 2020 (2020), https://www.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/sophos-the-state-of-ransomware-2020-wp.pdf
44. Tam, K., Khan, S.J., Fattori, A., Cavallaro, L.: CopperDroid: Automatic reconstruction of Android malware behaviors. In: NDSS (2015)
45. Uppal, D., Mehra, V., Verma, V.: Basic survey on malware analysis, tools and techniques. International Journal on Computational Sciences & Applications (IJCSA) **4**(1), 103 (2014)
46. WeLiveSecurity: WeLiveSecurity (2020), https://www.welivesecurity.com/
47. Zhou, W., Zhou, Y., Jiang, X., Ning, P.: Detecting repackaged smartphone applications in third-party Android marketplaces. In: Proceedings of the second ACM conference on Data and Application Security and Privacy. pp. 317–326. ACM (2012)

6 Appendix

Listing 1.1. Filecoder match for Write To File Unknown Extension

```
4179;23:18:22;openat;(AT_FDCWD, "U_DIR/large_text.txt.seven",
    O_WRONLY|O_CREAT|O_TRUNC|O_LARGEFILE, 0666 <unfinished ...>
4179;23:18:22;openat;( ) = 36
4179;23:18:22;fstat64;(36, <unfinished ...>
4179;23:18:22;fstat64;( {st_mode=0, st_size=1, ...}) = 0
4179;23:18:22;write;(36,"\10\261{H|\254\226\32\202\342\322\222\230
    \376c\256h\347\253\347v\271\"\303\265W\203\"\203\244\265T"...,
    148720 <unfinished ...>
```

Listing 1.2. Benign cache-cleaning application pattern mismatch for Read User File

```
4436;21:30:03;openat;(AT_FDCWD, "U_DIR/large_image.jpg",
    O_RDONLY|O_LARGEFILE) = 119
4436;21:30:03;fstat64;(119, {st_mode=0, st_size=1, ...}) = 0
4436;21:30:03;read;(119, <unfinished ...>
4436;21:30:03;read;( "\377\330\377\340\0\20JFIF\0\1\1\1\0d\0d\0\0
    \377\376\0LFile sou"..., 8192) pow2
4436;21:30:03;read;(119, "\344\6Q,\24\266\325j\333\244\312N\371#
    \2\247\236*\244\363Bx\2\356\f\235\205(\266\360.\7"
    ..., 8192) pow2
4436;21:30:03;read;(119, <unfinished ...>
4436;21:30:03;read;( "l\214\254\223\10\250\222H\356\304\366\2\275\
    r-\251S\342\273t\357\177\336\306\376\33G\315\225p
    \272\276"..., 8192) pow2
```

Table 2. List of token names and their respective pattern

Token	Pattern Purpose
OP	System call operation
AL	All including newline
UD	User directory
N	Newline
ON	Optional match newline
A	Match all
UFC	Unknown file creations
DQ	Dotted quad formats (i.e. IPv4)
AD	URL address
OF	Obfuscated file
SF	Socket flags
GA	Get address info
MD	Match directory
MF	Match file (regular file with one extension)

Table 3. List of common behavioural patterns discovered and their token representation

Pattern Name	Pattern Combination
Rename & Unlink File	OP(renameat)>UD>\\(?!Android)>N>ON >OP(fstatat64) >N>ON>OP(unlinkat)>UD>A
Unlinking User Files	OP(unlinkat)>UD>MF
Unknown File Ext Created	OP(openat)>UD>UFC>A
Read User File	OP(openat)>UD>MF>(>AL>OP(read)>N>){3}
Write File Unknown Extension	OP(openat)>UD>UFC>AL>OP(write)>A
IPv4 Connections	OP(connect)>DQ
Directory Search	OP(openat)>MD>N>N>(>OP(getdents64)>N>)* >OP(close)>A
URL to Obfuscated Filename	OP(openat)>OF>(>OP(openat>)?>AL >OP(pwrite64)>AD
Socket Create and Connect	OP(socket)>SF>N>(>OP(socket)>N>)?>A >OP(setsockopt) >N>(>OP(setsockopt) >N>)>?>OP(connect) >N>(>OP(connect) >N>)>?>OP(fcntl64) >N>(>OP(fcntl64)>N>) >?>OP(fstat64) >N>(>OP(fstat64)>N>) >?>OP(write)>GA
File Write	OP(openat)>AL>OP(write)>A
File Read	OP(openat)>AL>OP(read)>A
Generic File Unlink	OP(unlinkat) (.*?(.(\w+)(\bflock|xml|bak|db-wal\b)\").+)

Note: Some sub-patterns were retained as a regular expression as certain parts are too specific to be represented as tokens.