Evaluation of the Reliability of Android Userland Memory Forensics

Sneha Sudhakaran¹, Aisha Ali-Gombe², Andrew Case³ and Golden G Richard III¹
¹Department of Computer Science, Louisiana State University, Baton Rouge, USA
²Department of Computer Science, Towson University, Towson, USA
³Board of Directors, Volatility Foundation, Reston, USA
ssudha1@lsu.edu
aaligombe@towson.edu
andrew@dfir.org
golden@cct.lsu.edu

Abstract: Memory Forensics is one of the most important emerging areas in computer forensics. In memory forensics, analysis of userland memory is a technique that analyses per-process runtime data structures and extracts significant evidence for application-specific investigations. In this research, our focus is to examine the critical challenges faced by process memory acquisition that can impact object and data recovery. Particularly, this research work seeks to address the issues of consistency and reliability in userland memory forensics on Android. In real-world investigations, memory acquisition tools record the information when the device is running. In such scenarios, each application’s memory content may be in flux due to updates that are in progress, garbage collection activities, changes in process states, etc. In this paper we focus on various runtime activities such as garbage collection and process states and the impact they have on object recovery in userland memory forensics. The outcome of the research objective is to assess the reliability of Android userland memory forensic tools by providing new research directions for efficiently developing a metric study to measure the reliability. We evaluated our research objective by analysing memory dumps acquired from 30 apps in different Process Acquisition Modes. The Process Acquisition Mode (PAM) is the memory dump of a process that is extracted while external runtime factors are triggered. Our research identified an inconsistency in the number of objects recovered from analysing the process memory dumps with runtime factors included. Particularly, the evaluation results revealed differences in the count of objects recovered in different acquisition modes. We utilized Euclidean distance and covariance as the metrics for our study. These two metrics enabled the authors to identify how the change in the number of recovered objects in PAM impact forensic analysis. Our conclusion revealed that runtime factors could on average result in about 20% data loss, thus revealing these factors can have an obvious impact on object recovery.

Keywords: Userland, Memory Dump Acquisition, Reliability, Metric Evaluation

1. Introduction

In a recent survey conducted, the Android operating system makes up 71.9% of the mobile operating system market share (Market Share, 2020). The survey results prove the popularity of the operating system among end-users. Over the last decade, there was a parallel increase in cybercrimes, causing damage that cost up to $6 trillion in February 2021 (Market Crime, 2021). Digital forensics is a branch of forensic science used for recovering, investigating, and examining digital devices to recover evidences (Auty et al., 2007). Memory forensics is one of the techniques in digital forensics for extracting evidence from digital media that can serve as evidence for solving such cyber-crimes (Sylve et al., 2012). This technique can be an efficient solution to extract evidence and solve cybercrime, thereby making end-users more secure. Among different memory forensics techniques, userland (process memory) memory forensics is one important research area for analyzing applications (app) and activities associated with the app is essential in today’s cyber world (Auty et al., 2007). One of the most critical components impacting evidence recovery is process memory acquisition (Pagani et al. 2019). The acquisition must be performed with utmost attention to conquer the challenges that persist. While some research has been done on challenges faced during the memory dump acquisition(Pagani et al., 2019), to the best of our knowledge, there is very little research that focuses on external runtime factors that can affect evidence recovery from process dumps that includes RecOOP (Pridgen et al.). The primary focus of this paper is to conduct a study on the impact of the external runtime factors like the Garbage Collection (GC) and the Process States and finally develop a metric evaluation to assess the reliability of userland memory forensic tools. The external factors like GC during process memory acquisition impact object recovery because the number of objects allocated and recovered without GC occurring is more than objects allocated and recovered after GC’s occurrence. The object count difference is primarily because some objects are collected and lost after a GC cycle. The difference in object count was observed in different process states and is described in detail. First, we present a methodology to identify the impact on process memory samples with combinations of external runtime factors on object recovery using some available and free Android userland memory forensic tools (Ali-Gombe et al., 2019) and (Sudhakaran et al., 2020). Next, we evaluated multiple apps and derived a metric
evaluation criterion for measuring the reliability of userland forensic tools on process memory capture acquired by incorporating the runtime factors during the dump acquisition. The study and conclusion deduced from this research were based on the count of objects recovered from process memory dumps with runtime factors included using current Android memory forensic tools. Developing a metric evaluation is needed because of the noticeable changes in the number of objects retrieved. The difference in the number of objects recovered helped us analyze the integrity, consistency, and data loss in different dumps acquired. Therefore, we could better understand userland memory forensic tools’ reliability using metrics like Euclidean distance and covariance.

1.1 Contribution
- Examining the impact of Garbage Collection and Process States on evidence recovery in userland memory.
- Quantifying the integrity, consistency, and damage in recovered data.
- Measuring the reliability of userland memory forensics techniques using metrics like Euclidean distance and Covariance.

The rest of the paper is organized as follows: Section 2 presents the Background of this paper; Section 3 provides an overview of our Design and implementation; Section 4 presents the Evaluation of the proposed approach; Section 5 summarizes the Related Literature and finally section 6 presents the Conclusion

2. Background

2.1 Android Runtime Environment
Android executes apps in an application runtime environment called Android Runtime (ART) (Ali-Gombe et al., 2019) (Schwermer et al., 2018). ART introduced significant improvements like GC and better debugging (Schwermer et al., 2018) (Ali-Gombe et al., 2019) (ART Space, 2017). In Android 8, the developers improved ART by introducing features like Concurrent Copying GC (Ali-Gombe et al., 2019) enabling smaller heap sizes and faster object allocation and deallocation (Schwermer et al., 2018). First, the technique uses a concurrent and moving garbage collection algorithm. Utilizing region-based memory allocation (Ali-Gombe et al., 2019), allocated objects are evacuated from a region and subsequently destroyed if and only if the region has live objects whose count is less than some percentage threshold. Also, this algorithm creates a compacting heap by introducing short pauses during collection. It also utilizes a read barrier configuration to ensure mutators never see old versions of objects. This configuration allows threads to efficiently and concurrently access heap objects during collection. The algorithm uses the RegionSpace allocator, and if the use of TLAB is enabled, the system uses the RegionSpaceTlab allocator for movable objects. On newer Android versions, RegionSpaceTlab (Ali-Gombe et al., 2019) is the default for most small object allocations and LargeObjectSpace (Sudhakaran et al., 2020) for large object allocations. Second, the core Android system components and services like ART are built from native code that relies on native libraries written in C/C++ (ART Platform, 2017). Finally, ART's memory management does not provide a memory swap area but instead uses paging mechanisms and file mapping (Soares, A.M.M., de Sousa Jr, RT, 2017). Overall, for end-users, ART is more beneficial than its predecessor Dalvik by offering improved performance and faster application start-up time (ART Dalvik, 2017). In ART, improvements like Foreground and Background collectors are used when an app is in Foreground and Background process states (Ali-Gombe et al., 2019). This research intends to study in detail the GC and process state improvements made in ART and identify how they impact consistency, data loss/integrity, and reliability of forensic evidence recovery tools.

2.2 Object Allocation and Deallocation
In Android, object allocation utilizes memory management algorithms based on the size of the object (AndroidLOS, 2017) (Ali-Gombe et al., 2019) (Sudhakaran et al., 2020). Objects with an allocation size of fewer than 12KB are small objects like primitives, strings, arrays, and other complex objects such as InetAddress and are allocated using the Alloc() function (Ali-Gombe et al., 2019). On the other hand, the AllocLarge() function allocates large objects above a certain threshold of 12KB (Sudhakaran et al., 2020). The small objects are allocated using the region-based memory management algorithm, and the large objects are allocated using the large object space algorithm. In a region-based algorithm, objects get allocated in specific memory regions. During GC, an entire region is garbage collected if the objects in the corresponding region alive are below a certain threshold. The Large Object Space (LOS) gets allocated in a region in the process memory called Dalvik Large Object Allocation. In ART, LOS uses discontinuous memory mapping, where object allocation regions are not contiguous. LOS allocates objects in the form of arrays of types such as byte, char, string, float, and int
The objects and the associated references in the allocated spaces are freed when the allocated object is not alive with a `Free()` function (ART Free, 2017).

2.3 Garbage Collection
There are four major GC algorithms designed for ART. The algorithms are Semi-Space, Generational Semi-Space, Concurrent Mark Sweep, and Concurrent Copying (Ali-Gombe et al., 2019) (Jones et al., 2016). Beginning with Android 8, the default GC plan is Concurrent Copying (Ali-Gombe et al., 2019). The other GC plan used in ART is Concurrent Mark Sweep (ART Developers, 2017). When utilizing region-based memory allocation, allocated objects in the region having a live object count less than a certain threshold are evacuated from a region and subsequently destroyed (Ali-Gombe et al., 2019). Similarly, allocated objects having an object count greater than the threshold utilize LOS allocation and the objects are collected and destroyed if and only if the object and its reference are no longer alive. In such situations, the function `FinishGC()` (FinishGC, 2017) is enabled indicates that GC has been enabled. Otherwise, the GC is reset or disabled when the `ResetGCPerformanceInfo()` (ResetGC, 2017) function/method is triggered.

2.4 Process State
In an Android system, the process state is the highest-ranking active component within the app that it hosts (Android Process, 2017). Foreground processes have highest priority & empty processes have lowest priority as shown in Figure 1.

![Android Process States](image)

Android's different process states are foreground, background, visible, service, and empty (Android Process, 2017). The different process states are explained below:

2.4.1 Foreground Process
Foreground processes (Android Process, 2017) are the process or the app that is currently an active process running in the Android system and is last to be terminated by the system. A process is in foreground state if it meets one or more of the following conditions given below:
- The process involves activities with user interaction.
- Process hosting a service-connected to user interacting activities.
- Service that is triggered by a function call to `startForeground()`.
- Process that holds services like `onCreate()`, `onResume()` or `onStart()`, `onReceive()` calls.

2.4.2 Visible Process
A process is classified as a 'visible process’ if it contains an activity visible to the user while the activity does not involve interaction with the user (Android Process, 2017).

2.4.3 Service Process
Processes that contain a service that has already been started and is currently in execution are classified as service process (Android Process, 2017).

2.4.4 Background Process
These processes contain activities neither visible to the user nor hosting a service. Android maintains a dynamic list of background processes, terminating processes such that processes that were the least recently in the foreground are killed first (Android Process, 2017).

2.4.5 Empty Process
Empty processes no longer contain any active applications but reserve memory space and serve as hosts for newly launched applications (Android Process, 2017).

This research focuses only on Foreground and Background process states.
3. Analysis Setup

3.1 Experimental Setup
In this work, we used the Genymotion Android emulator in the experimental setup as the execution environment (Genymotion, 2016) to evaluate the selected apps. We created Android Virtual Devices (AVD’s) for the Samsung S8 emulator running Android 8.0-API 26 and apps chosen from different categories like Browser, Entertainment, SMS, Social media, Vault, Gaming, and malicious apps from Google Playstore (GooglePlay, 2021) and VirusShare (VirusShare, 2021).

All the emulators had 4GB memory, and selected apps were installed and loaded with chat messages, multiple images, text files, videos to simulate a series of actions performed by a user on real devices. We interacted with all the apps selected manually, with a similar sequence of actions conducted on each app to generate consistent activities for evaluation. E.g., The app `com.appstalking.photoeditor` was installed on the Genymotion emulator, and we performed a sequence of actions in both Foreground and Background states. In the Foreground process state, the activities include opening the app and typing a text message, then uploading an image saved in the Genymotion emulator. Next, we uploaded and edited a pdf file, and finally uploaded a video and watched the video. For the background process state, we repeat the same sequence of actions performed on the `com.appstalking.photoeditor` when it was in the foreground, before putting it in the background with a Gmail app made as a foreground app.

3.2 Process Acquisition Modes
As mentioned above, this research leverages two external runtime factors GC and process states. In Figure 2, the memory layout called the vtype (Auty et al., 2007) is explained to understand the different parameters and their memory offsets in the automated script to confirm if the process dump acquired includes the runtime factors or not. The automated script used to check the existence of each runtime factor will be made open source when the paper is published. The heap in the process dump acquired has the vtypes on which this work focuses. The specific vtypes focussed in this work includes GC_collector at location 504; card_table_ at 56; last_gc_type_ at 224; next_gc_type_ at 228; desired_collector_ at 100; block_gc_count_ at 584; block_gc_time_ at 592; gc_plan_ at 440 an concurrent_copying_collector_ at 524 as shown in Figure 2.

Figure 2: Runtime Structure for External Runtime Factors in Memory
In the case of GC, we identified that when GC is enabled the variables blocking_gc_count is 1 and blocking_gc_time holds some integer(FinishGC, 2017). While the GC is not enabled, the variables blocking_gc_count and blocking_gc_time are set to 0 (ResetGC, 2017). In the case of Process State, the variable desired_collector_ holds a value that indicates if the process was running in the Foreground or Background. The desired_collector_ is 7 when the app is running in foreground and desired_collector_ is 8 when the app runs in background. On executing the automated script on all the selected app for analysis each PAM mode gives the corresponding output mentioned below

3.2.1 F-GC
The variable types with values for this memory dump are desired_collector_ = 0x07; block_gc_count_ = 1; block_gc_time_ = <value>.

3.2.2 F-NGC
The variable types with values for this memory dump are desired_collector_ = 0x07; block_gc_count_ =0; block_gc_time_ = 0.
3.2.3 B-GC
The variable types with values for this memory dump are _desired_collector_ = 0x08; _block_gc_count_ = 1; _block_gc_time_ = <value>.

3.2.4 B-NGC
The variable types with values for this memory dump are _desired_collector_ = 0x08; _block_gc_count_ = 0; _block_gc_time_ = 0.

3.2.5 Clean
This state of process dump acquisition is the fresh state when the app runs with no runtime state triggered. The combination would be a default condition when a user opens the app and interacts in the foreground. This state is acquired every time before acquiring the process memory with the runtime factor triggered. CleanFGC is the process memory dump acquired before acquiring the F-GC combination dump. Similarly, we acquired a clean dump before F-NGC, B-NGC, and B-GC, and the memory acquisition modes were called CleanFNGC, CleanBNGC, and CleanBGC, respectively.

3.3 Process Memory Acquisition For Analysis
Figure 3 depicts the system architecture of this analysis study. The app memory dump is acquired with an automated tool called Memfetch (Memfetch, 2009) with different runtime factors included, and the process dumps acquired are called the process acquisition mode (PAM). The Memfetch tool is more beneficial as this research mainly focuses on userland memory analysis. Memfetch extracts all anonymous memory blocks like the stack and heap used for object allocation (mem pages) and pages used for files and executables mappings (map pages) into distinct binary files called the mem*.bin and the map*.bin respectively (Memfetch, 2009). Memfetch further provides a statistical metadata file that shows the range, size, segment of each acquired memory block in a file called the mfetch.lst. These files are then analyzed by forensic tools to recover all the objects. In our research, we focus specifically on two runtime factors, the GC and Process States.

In the automated script (https://github.com/ssudha1/metrics/tree/main), before we used Memfetch for acquiring process dumps, we initiated the two runtime factors. The GC was forced from the Android shell using 'adb shell kill -10 PID' where 'kill -10' signals SIGUSR1 (ForceGC, 2013), and PID is the process id of the application. However, the process states during memory acquisition were based on the manual setting by the end-user. We focus on the Foreground and Background process states which predominantly relate to user input. These states occur when the user acquires the process dump by running the automated script. The Foreground state is where the user interacts with the application, and the Foreground Collector gets triggered. The Background process state is where the application under execution is not visible to the user, and the Background Collector is triggered. E.g. When the user starts the app in the Foreground process state and then executes the automated script (https://github.com/ssudha1/metrics/tree/main), it generates a process dump without GC enabled called Foreground and No GC enabled (F-NGC) followed by a dump with GC enabled called Foreground and GC triggered (F-GC). Similarly, when the user starts the app in the Background state and the automated script(https://github.com/ssudha1/metrics/tree/main) is executed, the process dump acquired would be initially the dump without GC enabled called Background and No GC enabled (B-NGC) followed by a dump with GC called as the Background and GC enabled (B-GC).
Figure 3: Design Architecture for Evaluation

3.4 Analysis Tools

3.4.1 DroidScraper

Ali-Gombe et al. proposed DroidScraper (Ali-Gombe et. al, 2019), a tool for analyzing the ART RegionSpace memory allocation to extract small objects from process dumps. This tool recovers runtime data structures by focusing on the recovery of small objects allocated in the RegionSpace. DroidScraper is a userland in-memory object recovery and reconstruction tool to extract runtime artifacts from Android process memory space. The evaluation of DroidScraper has shown that it can recover in-memory data allocated using Android’s RegionSpace by using RegionSpaceTab allocator for recovering small objects with a recovery percentage of almost 90%. Also, DroidScraper can reconstruct and recover objects, thereby detecting evidence of file and network activities, database accesses, and recovery of cryptographic keys.

3.4.2 AmpleDroid

Sudhakaran et al. proposed a tool AmpleDroid (Sudhakaran et. al, 2020) a tool to analyze an Android app dump to extract the large objects allocated in the specific memory region called LargeObjectSpace. The LargeObjectSpace allocation is studied to identify how the objects above a certain threshold are stored in userland memory. AmpleDroid performs a complete process memory analysis on Android version 8 large object memory allocation and extracts multimedia and text files that other tools cannot currently retrieve. This tool is used in forensic investigations to provide an overall idea of how large object files (text, video, image, etc.) are allocated in an app’s memory and a high recovery percentage of approximately 91%.

Lastly, in this experimental setup the forensic tools - DroidScraper (Ali-Gombe et al., 2019) and AmpleDroid (Sudhakaran et al., 2020) are used to recover the objects allocated in each acquisition mode. The final objects extracted from all the dumps will include a set of objects extracted from an Android app memory. Finally, the authors were able to identify the objects recovered in each dump and determine how reliable these forensic tools are in recovering objects from process memory.

4. Evaluation

We evaluated 30 apps in different process acquisition modes (cleanFGC, F-GC, cleanFNGC, F-NGC, cleanBGC, B-GC, cleanBNGC, B-NGC), and the results are shown in Figure 4. We evaluated 15 benign apps from different app categories: Browser, Editing, Entertainment, Gaming, SMS, Social Media and 15 malicious apps. Upon analysis, we identified that the process dumps with the acquisition modes - no GC and both process states (Foreground and background) extracted more small and large objects than the process dumps with acquisition modes GC and both process states. We used two metrics, Euclidean distance and covariance, to evaluate the data loss. The Euclidean distance calculated here gives the percentage of dissimilarity between two acquisition modes, e.g. (Data recovered in a clean state, data retrieved with runtime factor triggered). We used the covariance metric to examine the variability of data retrieved in different process acquisition modes, e.g., the variability of data recovered in F-NGC, F-GC, B-NGC, and B-GC. The results in Figure 4 proves a greater probability for objects and their references get collected in the case of process acquisition modes F-GC and B-GC. On analysis with
DroidScraper and AmpleDroid, we identified that the number of recovered objects in modes F-GC and B-GC was lower than F-NGC and B-NGC. Figure 4 shows a significant difference in the small and large objects from all acquisition modes in certain benign entertainment apps like com.vid007.videobuddy, com.redbox.tv and com.ezscreenrecorder. The range of data loss calculated using Euclidean distance is approximately 10-20% in the case of GC acquisition modes while 0-10% in the case of No-GC acquisition modes. The data loss in terms of covariance ranges between .06 to .09 in (6-9% varying) both small and large objects. Therefore, we identified the change to be consistent in the case of entertainment apps. In the case of SMS apps, we identified the Euclidean distance as 7-17% and 0-7% for GC and No-GC process acquisition modes, respectively. We also determined that the covariance value was between .04 to .09 (4-9% varying). However, this consistency was not observed in Browser and Editing apps as we observed a Euclidean metric ranging from 0-19% in the case of all the acquisition modes and covariance variability from 0 to 11 (0 to 11% varying). Therefore, we could not identify a consistent range of percentage data loss but can conclude that the loss of data is restricted to 20% in the case of all benign and malicious applications. Thus, the garbage collected objects removed from memory might or might not serve as a loss of a critical piece of evidence during object recovery. In any case, if such pieces of evidence are not recovered, it eventually leads to issues that may question the reliability of forensic investigation tools. Such an object recovery can miss out on evidence that can also cause innocent users to be convicted. Therefore, on analyzing the difference in the count of objects recovered it is clear from in Figure 4 that the objects recovered using B-GC acquisition mode were fewer and had greater Euclidean distance compared to other modes. This also made us understand that among the process states lesser objects were recovered from Background compared to objects recovered from dumps in foreground. For E.g, on comparing F-GC and B-GC, more objects were recovered in F-GC compared to B-GC.
4.1 Recovery Consistency and Data Loss

On analysis, we identified that the results in Figure 5 reveal that we retrieved more non-Gui objects (Class name, text, etc.) in case of benign (apps selected from each category) and malicious applications than Gui and network objects. When we compared the count of Gui objects recovered from dumps with GC triggered and dumps with No-GC triggered, we identified that in the Foreground process dumps, the Editing, Entertainment, and SMS apps had an average of 19% reduction while the malware apps had an average of 24% reduction. When we compared the Gui object count for B-NGC and B-GC, in the case of Editing, Entertainment and SMS, and Malware apps, the reduction in object recovered was 19%, 19.5%, 24.5%, and 21%, respectively. When we compared the count of Non-Gui objects recovered from dumps with GC triggered and dumps with No-GC triggered, we identified that in the Foreground process dumps, the Editing, Entertainment, and SMS apps had an average of 20% reduction. In comparison, the malware apps had an average of 22% reduction. When we compared the Gui object count for B-NGC and B-GC, in the case of Editing, Entertainment, and SMS, and Malware apps, the reduction in object recovered was 20%, 20.5%, 22%, and 24%, respectively.

<table>
<thead>
<tr>
<th>Pedagogy</th>
<th>App Category</th>
<th>Class Name</th>
<th>Foreground</th>
<th>Back Ground</th>
<th>Foreground</th>
<th>Back Ground</th>
<th>Class Name</th>
<th>Foreground</th>
<th>Back Ground</th>
<th>Class Name</th>
<th>Foreground</th>
<th>Back Ground</th>
</tr>
</thead>
<tbody>
<tr>
<td>snehastudakaran</td>
<td>Browser</td>
<td>18</td>
<td>18</td>
<td>0.00%</td>
<td>19</td>
<td>16</td>
<td>16.7%</td>
<td>18</td>
<td>17</td>
<td>10.0%</td>
<td>19</td>
<td>16</td>
</tr>
<tr>
<td>snehastudakaran</td>
<td>Android</td>
<td>18</td>
<td>18</td>
<td>0.00%</td>
<td>19</td>
<td>16</td>
<td>16.7%</td>
<td>18</td>
<td>17</td>
<td>10.0%</td>
<td>19</td>
<td>16</td>
</tr>
<tr>
<td>snehastudakaran</td>
<td>Mobile</td>
<td>18</td>
<td>18</td>
<td>0.00%</td>
<td>19</td>
<td>16</td>
<td>16.7%</td>
<td>18</td>
<td>17</td>
<td>10.0%</td>
<td>19</td>
<td>16</td>
</tr>
<tr>
<td>snehastudakaran</td>
<td>Desktop</td>
<td>18</td>
<td>18</td>
<td>0.00%</td>
<td>19</td>
<td>16</td>
<td>16.7%</td>
<td>18</td>
<td>17</td>
<td>10.0%</td>
<td>19</td>
<td>16</td>
</tr>
<tr>
<td>snehastudakaran</td>
<td>Internet</td>
<td>18</td>
<td>18</td>
<td>0.00%</td>
<td>19</td>
<td>16</td>
<td>16.7%</td>
<td>18</td>
<td>17</td>
<td>10.0%</td>
<td>19</td>
<td>16</td>
</tr>
<tr>
<td>snehastudakaran</td>
<td>Email</td>
<td>18</td>
<td>18</td>
<td>0.00%</td>
<td>19</td>
<td>16</td>
<td>16.7%</td>
<td>18</td>
<td>17</td>
<td>10.0%</td>
<td>19</td>
<td>16</td>
</tr>
<tr>
<td>snehastudakaran</td>
<td>Social</td>
<td>18</td>
<td>18</td>
<td>0.00%</td>
<td>19</td>
<td>16</td>
<td>16.7%</td>
<td>18</td>
<td>17</td>
<td>10.0%</td>
<td>19</td>
<td>16</td>
</tr>
<tr>
<td>snehastudakaran</td>
<td>Media</td>
<td>18</td>
<td>18</td>
<td>0.00%</td>
<td>19</td>
<td>16</td>
<td>16.7%</td>
<td>18</td>
<td>17</td>
<td>10.0%</td>
<td>19</td>
<td>16</td>
</tr>
<tr>
<td>snehastudakaran</td>
<td>Games</td>
<td>18</td>
<td>18</td>
<td>0.00%</td>
<td>19</td>
<td>16</td>
<td>16.7%</td>
<td>18</td>
<td>17</td>
<td>10.0%</td>
<td>19</td>
<td>16</td>
</tr>
<tr>
<td>snehastudakaran</td>
<td>Tools</td>
<td>18</td>
<td>18</td>
<td>0.00%</td>
<td>19</td>
<td>16</td>
<td>16.7%</td>
<td>18</td>
<td>17</td>
<td>10.0%</td>
<td>19</td>
<td>16</td>
</tr>
<tr>
<td>snehastudakaran</td>
<td>Finance</td>
<td>18</td>
<td>18</td>
<td>0.00%</td>
<td>19</td>
<td>16</td>
<td>16.7%</td>
<td>18</td>
<td>17</td>
<td>10.0%</td>
<td>19</td>
<td>16</td>
</tr>
<tr>
<td>snehastudakaran</td>
<td>Health</td>
<td>18</td>
<td>18</td>
<td>0.00%</td>
<td>19</td>
<td>16</td>
<td>16.7%</td>
<td>18</td>
<td>17</td>
<td>10.0%</td>
<td>19</td>
<td>16</td>
</tr>
<tr>
<td>snehastudakaran</td>
<td>Travel</td>
<td>18</td>
<td>18</td>
<td>0.00%</td>
<td>19</td>
<td>16</td>
<td>16.7%</td>
<td>18</td>
<td>17</td>
<td>10.0%</td>
<td>19</td>
<td>16</td>
</tr>
<tr>
<td>snehastudakaran</td>
<td>Messages</td>
<td>18</td>
<td>18</td>
<td>0.00%</td>
<td>19</td>
<td>16</td>
<td>16.7%</td>
<td>18</td>
<td>17</td>
<td>10.0%</td>
<td>19</td>
<td>16</td>
</tr>
</tbody>
</table>

*Figure 4: Evaluation Result*
for B-NGC and B-GC, in the case of Editing, Entertainment and SMS, and Malware apps, the reduction in object recovery count was 20%.

Similarly, the reduction in Network objects recovered ranges between 18% to 23% in all the apps studied in this research. The histograms in Figure 6 include component-based recovery from a few apps selected from each app category. Figure 6 depicts that we retrieved more Android Services than Android activity components and resource files on plotting apps from each category like editing, malware, SMS, entertainment.

<table>
<thead>
<tr>
<th>Acquisition Mode</th>
<th>Package Name</th>
<th>Package Type</th>
<th>Large</th>
<th>Gui</th>
<th>Non-Gui</th>
<th>Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>F-NGC</td>
<td>com.appstore.playeeditor</td>
<td>Editing</td>
<td>18</td>
<td>1728</td>
<td>24723</td>
<td>205</td>
</tr>
<tr>
<td></td>
<td>de-zerymedia.imageline</td>
<td>Editing</td>
<td>16</td>
<td>3553</td>
<td>38119</td>
<td>479</td>
</tr>
<tr>
<td></td>
<td>com.redbox.tv</td>
<td>Entertainment</td>
<td>10</td>
<td>1271</td>
<td>18883</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>com.experimenter</td>
<td>Entertainment</td>
<td>28</td>
<td>2181</td>
<td>30449</td>
<td>398</td>
</tr>
<tr>
<td></td>
<td>com.dieplusspam</td>
<td>SMS</td>
<td>14</td>
<td>2003</td>
<td>28771</td>
<td>361</td>
</tr>
<tr>
<td></td>
<td>com.malware.bc</td>
<td>Malware</td>
<td>17</td>
<td>1549</td>
<td>23563</td>
<td>318</td>
</tr>
<tr>
<td></td>
<td>com.androidbattery</td>
<td>Malware</td>
<td>10</td>
<td>1151</td>
<td>10001</td>
<td>102</td>
</tr>
<tr>
<td></td>
<td>com.malware.mountap</td>
<td>Malware</td>
<td>8</td>
<td>1004</td>
<td>9883</td>
<td>111</td>
</tr>
<tr>
<td>F-GC</td>
<td>com.appstore.playeeditor</td>
<td>Editing</td>
<td>16</td>
<td>1400</td>
<td>15281</td>
<td>134</td>
</tr>
<tr>
<td></td>
<td>de-zerymedia.imageline</td>
<td>Editing</td>
<td>16</td>
<td>1994</td>
<td>23108</td>
<td>317</td>
</tr>
<tr>
<td></td>
<td>com.redbox.tv</td>
<td>Entertainment</td>
<td>9</td>
<td>991</td>
<td>8329</td>
<td>99</td>
</tr>
<tr>
<td></td>
<td>com.experimenter</td>
<td>Entertainment</td>
<td>27</td>
<td>1771</td>
<td>25771</td>
<td>253</td>
</tr>
<tr>
<td></td>
<td>com.dieplusspam</td>
<td>SMS</td>
<td>10</td>
<td>1558</td>
<td>23165</td>
<td>321</td>
</tr>
<tr>
<td></td>
<td>com.malware.bc</td>
<td>Malware</td>
<td>13</td>
<td>1287</td>
<td>30459</td>
<td>276</td>
</tr>
<tr>
<td></td>
<td>com.androidbattery</td>
<td>Malware</td>
<td>9</td>
<td>792</td>
<td>7648</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td>com.malware.mountap</td>
<td>Malware</td>
<td>7</td>
<td>839</td>
<td>7805</td>
<td>87</td>
</tr>
<tr>
<td>B-NGC</td>
<td>com.appstore.playeeditor</td>
<td>Editing</td>
<td>15</td>
<td>1806</td>
<td>25904</td>
<td>191</td>
</tr>
<tr>
<td></td>
<td>de-zerymedia.imageline</td>
<td>Editing</td>
<td>15</td>
<td>2489</td>
<td>34381</td>
<td>491</td>
</tr>
<tr>
<td></td>
<td>com.redbox.tv</td>
<td>Entertainment</td>
<td>10</td>
<td>1309</td>
<td>12315</td>
<td>133</td>
</tr>
<tr>
<td></td>
<td>com.experimenter</td>
<td>Entertainment</td>
<td>28</td>
<td>2471</td>
<td>33585</td>
<td>405</td>
</tr>
<tr>
<td></td>
<td>com.dieplusspam</td>
<td>SMS</td>
<td>14</td>
<td>2096</td>
<td>29001</td>
<td>380</td>
</tr>
<tr>
<td></td>
<td>com.malware.bc</td>
<td>Malware</td>
<td>16</td>
<td>1600</td>
<td>20998</td>
<td>338</td>
</tr>
<tr>
<td></td>
<td>com.androidbattery</td>
<td>Malware</td>
<td>9</td>
<td>1301</td>
<td>11598</td>
<td>119</td>
</tr>
<tr>
<td></td>
<td>com.malware.mountap</td>
<td>Malware</td>
<td>8</td>
<td>1188</td>
<td>12006</td>
<td>97</td>
</tr>
<tr>
<td>B-GC</td>
<td>com.appstore.playeeditor</td>
<td>Editing</td>
<td>15</td>
<td>1489</td>
<td>16008</td>
<td>148</td>
</tr>
<tr>
<td></td>
<td>de-zerymedia.imageline</td>
<td>Editing</td>
<td>13</td>
<td>1952</td>
<td>31236</td>
<td>338</td>
</tr>
<tr>
<td></td>
<td>com.redbox.tv</td>
<td>Entertainment</td>
<td>9</td>
<td>997</td>
<td>9003</td>
<td>102</td>
</tr>
<tr>
<td></td>
<td>com.experimenter</td>
<td>Entertainment</td>
<td>20</td>
<td>2097</td>
<td>25993</td>
<td>297</td>
</tr>
<tr>
<td></td>
<td>com.dieplusspam</td>
<td>SMS</td>
<td>12</td>
<td>1496</td>
<td>25669</td>
<td>319</td>
</tr>
<tr>
<td></td>
<td>com.malware.bc</td>
<td>Malware</td>
<td>15</td>
<td>1274</td>
<td>26889</td>
<td>269</td>
</tr>
<tr>
<td></td>
<td>com.androidbattery</td>
<td>Malware</td>
<td>9</td>
<td>992</td>
<td>8303</td>
<td>97</td>
</tr>
<tr>
<td></td>
<td>com.malware.mountap</td>
<td>Malware</td>
<td>7</td>
<td>907</td>
<td>8251</td>
<td>83</td>
</tr>
</tbody>
</table>

Figure 5: Component based Analysis

![Component based recovery for editing apps in 0-200 scale](image1)

![Component based recovery for SMS apps in 0-200 scale](image2)

![Component based recovery for entertainment apps in 0-200 scale](image3)

![Component based recovery for malware apps in 0-200 scale](image4)

Figure 6: Histogram analysis of various files in selected apps
5. Related Literature

The analysis of software tools' reliability is inevitable in today's digital world (Ivanov, et. al, 2018)). The reliability assessment is more critical if the analyzed tool is used for solving cyber-related cases. Reliability assessment of such tools is required to analyze if the tool is unreliable, then the evidence extracted by the tool also becomes unreliable, thereby ending up generating incorrect evidence. Hence the reliability assessment of such tools is mandatory, and this research provides a methodology for one such assessment. We identified multiple published works on reliability assessment; however, only a few were related to reliability testing of memory dump acquisition for forensic analysis. Most of the reliability measurement works were based on multiple approaches to develop software reliability growth models using techniques like fuzzy models, regression analysis, neural networks, and machine learning (Ivanov et al., 2018). Another technique for reliability testing is by methodologies like training and testing neural networks (Fisch et. al, 2010) (Park et. al, 2013). However, in this work, we focus on issues about software that focus on the extraction of all types of objects (small and large) from smartphone app memory. This is difficult due to the challenges faced during process dump acquisition (Sylve et. al, 2012) (Pagani et. al, 2019) (Schatz et. al, 2007). There are many works associated with the recovery of forensic data. However, only a few works address the challenges faced by forensic investigators with memory acquisition (Pagani et. al, 2019), are more useful in this work. Among different memory dump acquisition techniques mentioned in works like Volatility (Aute et. al, 2007), (Schatz et al., 2007) - Schatz, in his work, explained his technique for reliable volatile memory dump recovery. Also, Pagani et al., in their work, explained how memory forensics should consider the time in which each memory dump was acquired (Pagani et. al, 2019). Pagani et al. in their work provided a way to assess the reliability of a result obtained thereby minimizing the effect of the acquisition time or detect inconsistencies in the data. While our research closely relates to the idea of reliability (Huelsbergen et. al, 1993) but focuses primarily on external runtime factors, we could not find any work that mentioned runtime factors as critical during memory dump acquisition. Therefore, our research provides a new research dimension of focusing on the critical memory dump acquisition. Hasanbadi et al. proposed an approach to determine approximately how much sequential memory acquisition at a designated time-intervals can mitigate the current challenges in memory forensics (Hasanbadi et al., 2018).

6. Conclusion

In this paper, the methodology identified two runtime factors that can impact Android application dump acquisitions. We presented the analysis results highlighting the changes in the count of objects recovered in every app dump analyzed using userland memory forensic tools. We evaluated multiple apps and identified a difference in object recovery rate during the analysis of app dumps with the runtime factors included during memory acquisition. Finally, we calculated the reliability of userland memory forensic tools using Euclidean distance and covariance metrics. Our evaluation of 30 apps (benign and malicious apps) shows these process states can impose data loss of approximately 20% with a metric Euclidean distance and less than 18% data loss with covariance metric. Furthermore, our comparative analysis found that the count of objects recovered from Foreground acquisitions modes are greater than objects recovered from Background acquisition modes in most of the apps analyzed. The userland forensic analysis’s reliability study was conducted on both small and large object recovery tools. The result will be more reliable in object recovery for a forensic investigator who can extract all the objects allocated in memory from the app startup. Also, this research highlights runtime factors and helps investigators explain the environment during memory acquisition, thereby providing a better understanding of reliability factors when dealing with forensic tools during memory dumps acquisition.

References

Market Crime, 2021 “300 terrifying cybercrime and cybersecurity statistics trends” Online: https://www.comparitech.com/vpn/cybersecurity
AndroidLOS, 2017 http://android xref/art/runtime/gc/space/large_object_space.h


Schwermer, P., 2018, “Performance evaluation of Kotlin and Java on Android Runtime”


