TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Martin Välbe 182500IVCM

BENCHMARKING OF ANDROID APPLICATIONS' SYSTEM CALLS BEHAVIOR: IMPLICATIONS FOR MALWARE DETECTION

Master's thesis

Supervisor: Alejandro Guerra Manzanares MSc

Co-supervisor: Tarmo Oja MSc

Tallinn 2021

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Martin Välbe 182500IVCM

ANDROIDI RAKENDUSTE SÜSTEEMIKUTSETE KÄITUMISE VÕRDLUSUURING: MÕJUD KAHJURVARA TUVASTAMISELE

Magistritöö

Juhendaja: Alejandro Guerra Manzanares MSc

Kaasjuhendaja: Tarmo Oja MSc

Tallinn 2021

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Martin Välbe

30.07.2021

Abstract

Android OS has been a market leader for mobile operating systems for nearly a decade now, thus being a long-time primary target for malicious actors. From year to year, various statistics display the merciless growth of mobile malware specifically developed against Android platforms. Meanwhile, the efforts among academic and professional researchers to counter malware are increasing as well, comprising various methods and approaches to detect and neutralize malicious software. These efforts include a wide range of dynamic features able to function as threat indicators. Among those features are system calls – programmatic routines that allow user applications to request privileged services from kernel. Within academic community, an imposing number of research concentrating on at least partial implementation of system calls in dynamic malware detection methods has been published. However, the studies seem to have been concentrating their effort on perfecting the malware detection models and algorithm trainings, while turning a blind eye to the multitude of platforms that the malware is targeting. Results of rigorous work based on a single Android device or emulation environment seem to be generalized to all Android devices without evaluation. The purpose of this thesis was to examine and analyze existing differences between system calls of malicious and benign applications on different Android platforms, including both emulators and real devices, in a cross-comparison. For that, the testing setup comprising of different platforms were utilized and limited sets of both malicious and benign applications were employed for extracting system calls data for comparative analysis. The results based on system call summaries indicated significant differences between real devices and emulators in terms of system call invocations. The differences between the real devices were less extensive, but in general still unexpectedly significant, suggesting that the studies employing system calls must consider with platform-specific differences in terms of general reliability.

This thesis is written in English and is 72 pages long, including 6 chapters, 11 figures and 14 tables.

Annotatsioon

Androidi rakenduste süsteemikutsete käitumise võrdlusuuring: mõjud kahjurvara tuvastamisele

Pikaajalise turuliidrina on Androidi operatioonisüsteem kujunenud kahjurvara arendajate ja kuritahtlike rühmituste üheks peamiseks sihtmärgiks. Samaaegselt on akadeemiliste ja kutseliste uurimisrühmade pingutused kahjurvara vastu võitlemisel pidevalt suurenenud. Kahjurvara tuvastamisel rakendatavad meetodid kasutavad ohuindikaatoritena erinevaid dünaamilisi karakteristikuid. Nende karakteristikute hulka kuuluvad ka süsteemikutsed ehk kindlad programmilised mehhanismid, mis võimaldavad rakendustel pöörduda operatsioonisüsteemi tuuma poole, käivitamaks erinevaid piiratud teenuseid. Viimastel aastatel on ilmunud hulk uuringuid, mis keskenduvad süsteemikutsete vähemalt osalisele rakendamisele dünaamilistes tuvastusmeetodites. Samas on taolised uuringud keskendunud tuvastusalgoritmidele ja mudelitreeningute täiustamisele, pöörates sealjuures vähe tähelepanu baasplatvormide omadustele. Reeglina üldistatakse üheainsa Androidi seadme või emulatsiooni põhjal tehtud töö tulemused täiendava valideerimiseta kogu laiale Androidi seadmete spektrile. Käesoleva magistritöö eesmärk oli uurida ja analüüsida erinevusi mobiilkahjurvara ja healoomuliste rakenduste süsteemikutsete vahel erinevatel Androidi platvormidel (reaalsed seadmed ja emulaatorid). Selleks rajatud testimiskeskkonnas käivitati valitud hulka pahatahtlikke ja healoomulisi rakendusi eesmärgiga koguda nende poolt teostatud süsteemikutseid, mille logiandmete summeerimise tulemusena viidi läbi võrdlev analüüs. Tulemused viitasid olulistele erinevustele süsteemikutsete rakendamisel reaalsete seadmete ja emulaatorite vahel. Reaalsete seadmete omavahelised erinevused olid küll väiksema ulatusega, kuid üldiselt siiski üle ootuste märkimisväärsed. Sellest järeldub, et süsteemikutseid käsitlevad uuringud peaksid laialdasema usaldusväärsuse saavutamiseks arvestama platvormispetsiifiliste erinevustega.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 72 leheküljel, 6 peatükki, 11 joonist, 14 tabelit.

List of abbreviations and terms

ABI	Application Binary Interface
ADB	Android Debug Bridge
AOT	Ahead-of-Time
API	Application Programming Interface
APK	Android Package
APT	Advanced Persistent Threat
ART	Android Runtime
AV	Anti-Virus
AVD	Android Virtual Device
CPU	Central Processing Unit
DL	Deep Learning
ENISA	European Union Agency for Cybersecurity
IPC	Inter-Process Communication
ISA	Instruction Set Architecture
JNI	Java Native Interface
MIPS	Microprocessor without Interlocked Pipelined Stages
ML	Machine Learning
NDK	Native Development Kit
OS	Operating System
PID	Process Identifier
ROM	Read-only Memory
SDK	Software Development Kit
SoC	System on a Chip
VM	Virtual Machine

Table of Contents

List of Figures

List of Tables

1 Introduction

The mobile device market is ever-growing and large numbers of applications and other software are constantly being developed and remodelled for mobile platforms. According to Statista [1], as of June 2021 there were roughly 6.4 billion existing smartphone subscriptions in the world. The most popular operating system (OS) for mobile platforms is Android, with an estimated worldwide market share of nearly 73% [2].

The advancement and widespread availability of smartphones has undoubtedly improved the communication possibilities. Yet alongside the numerous benefits, there are also several negative impacts, both social and technological. On the technological side, as many services that were once only reserved for personal computers have found their way into mobile platforms, malicious actors have also turned their eyes and mindset to that specific field, trying to find ways to exploit the potential weaknesses of mobile systems. There has been a massive influx of malicious software specifically targeting mobile devices during the last decade. The first reported malware designed for Android OS is known to be an SMS trojan named *Trojan-SMS.AndroidOS.FakePlayer.a* and it was discovered by Kaspersky Lab in August 2010 [3]. The following year of 2011 saw explosive growth of malicious programs for mobile platforms, driven by the surge in the number of threats exclusively targeting Android [4]. According to an assessment by AV-TEST, a prominent security software evaluation institute, there were more than 19 million different Android malware samples in existence worldwide as of April 2017, with almost half a million new samples discovered in that particular month alone [5]. The European Union Agency for Cybersecurity (ENISA) states in its recent threat report covering the years 2019 and 2020 [6] that malware has been ranked as #1 threat in the digital landscape of Europe as of 2020, maintaining the same position from 2018. This report also points out roughly 400 000 detections of pre-installed spyware and adware on mobile devices, indicating the increase of supply-chain types of attacks as well.

Detection of mobile malware through different means has thus become an important subject for many researchers trying to disrupt the spread of such malicious software. Because of several (physical) restrictions the mobile devices present, both development and research are more often than not done using different mobile operating system emulators on PC-s, that are by design duplicating both software and hardware aspects of the original device (the mobile phone hardware and its operating system). Emulators are considered to be a great testing platform due to being inexpensive, fairly fast, accessible and debugging-friendly.

There is however one aspect concerning both emulators as well as various original devices – the architectures of the mobile devices and the computers hosting the emulator are different. For example, the devices using the most popular mobile platform, Android, are very largely based on ARM architecture (although Android currently supports Intel and used to support MIPS architectures as well) [7]. At the same time, there is currently no *true* ARM-based emulation platform available and the emulators are through software means emulating an ARM processor on host computer which usually has an x86 or x64 processor. This could lead to certain runtime inconsistencies that distort the state of the original device in the emulation environment. In addition, it is probable that there might be an unknown number of inconsistencies that exist between different ARM-based devices as well.

A 2019 study [8] conducted in Tallinn University of Technology about machine learning-based mobile malware detection showed that the system calls triggered by the same malware on Android emulator and real phone are different. Those behavioral differences could possibly mean that the learning model created by emulator data may not be appropriate for detecting the malware obtained from real device. The exact reasons behind those system call differences triggered by malware as well as their extent on different platforms must be studied to bring awareness and clarity to the matter of how these differences could affect malware analysis processes. In addition to the evasion techniques employed for data extraction, the behavioral differences obtained from emulators and real devices might be an additional factor that weakens the practical advantage of using dynamic features over static ones on ML models [8].

The main research problem of this thesis consists of the fact that there is yet no solid research about the true extent of the system call differences on different platforms and the possible nature behind those differences. The purpose of this thesis is to examine and analyze existing differences between system calls of malicious and benign applications on different Android platforms, including both emulators and real devices in a cross-comparison. The results and conclusions of this research are meant to lead the way to examine the potential effects of system call differences on different platforms to malware detection development.

Research questions:

> **RQ1:** How significant are the potential differences of system calls invoked by a single application between different types of platforms?
>
> **RQ2:** What are the possible causes behind system call differences on different platforms?
>
> **RQ3:** How could the existing differences affect malware analysis and what are the possibilities to overcome this problem?

The novelty of this thesis constitutes in a fact that there are no known previous cross-platform comparative researches concerning the potential differences in system calls triggered by the same malware samples. Understanding and mapping the existence of those potential differences can hopefully assist the future researches in the field of mobile malware detection.

The object of this thesis would be to use both malware and benign applications for testing the potential system call differences on Android platforms. The selection of the testing platforms would include different popular models of Android phones and their emulated counterparts in two different popular emulation platforms. The scope of the thesis would be limited to compare the system call log summaries and not the call sequence alignments or patterns. The research would consist an experimental method, as the process would need to measure the impacts of independent variables (applications and different operational platforms) on dependent ones (system calls invoked by the malware and benign application sets). The final validation of the gathered data would be conducted in a form of a comparative analysis.

This document is structured as follows: Chapter 2 introduces the theoretical background and explains the main concepts behind the purpose of this thesis. Chapter 3 describes the selection of testing samples, setup of the collection platforms and the data collection procedure. Chapter 4 outlines the collected data, prepares and eventually undertakes the comparative analysis. Chapter 5 discusses and summarizes the main issues discovered, points out the threats to validity of this thesis and provides suggestions for future work.

2 Theoretical background

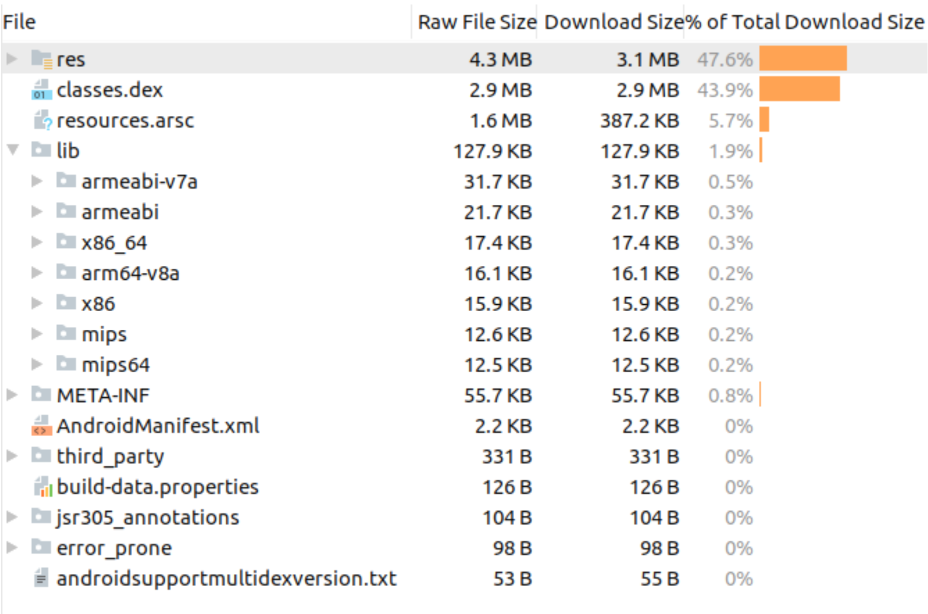
This chapter describes the relevant aspects of the Android mobile operating system and gives a short overview of the evolution of its malware and counter-malware measures. It also brings out the previous research examples bound to the modern approaches of system call-based dynamic malware detection (such as employing machine learning) and points to the certain practical issues that such modern detection techniques must consider with. The final part of this chapter also describes the particular issues addressed by this thesis and leads on to the practical part of the thesis.

2.1 Android OS – from application to kernel

Android is a mobile operating system based on a Linux kernel, which has been modified in order to efficiently perform on energy-constrained devices. It has been developed mainly for touchscreen mobile devices. Android OS has evolved significantly since its inaugural release in 2008 with version 1.0 [9, p. 168]. The latest stable release version available is Android 11, which was launched on September 8, 2020. However, according to [10], as of May 2021, the most popular release versions worldwide were still the two previous ones – Android 9 with 17.34% and Android 10 with 36.99%.

2.1.1 Android application fundamentals

The contents of an Android device that the user sees and is able to interact with (contacts, mail, camera, games, settings, etc.) comes in the form of applications, which form the topmost layer of the Android system. The majority of the Android applications available are written in programming languages such as Java or (more increasingly) Kotlin. One of the more common development suites available for this purpose is Android Studio Development Kit (SDK). The development tools compile the code along with any data and resource files into an Android Package (APK), which is an archive file with an *.apk* suffix. The typical structure and contents of an APK file are shown in Figure 1 below. One such APK file contains all the contents of an Android app and is the source that Android-powered devices use to install the app. However, using the Android Native Development Kit (NDK), modules of an application could also be written in C and C++ code, which would be compiled into a native library and packaged with the APK during the build [11], [12]. Such approach can be useful to achieve extra performance for running computationally intensive applications, such as games, or to reuse already existing C or C++ code libraries [13]. Most of the applications written in native code are either games or graphic simulations, because native code improves the performance of CPU intensive applications. Java code runs on specific runtime environment called Android Runtime (ART), while the native code runs outside the virtual machine (VM). Runtime environment is further explained in subsection 2.1.2.



File	Raw File Size	Download Size	% of Total Download Size
res	4.3 MB	3.1 MB	47.6%
classes.dex	2.9 MB	2.9 MB	43.9%
resources.arsc	1.6 MB	387.2 KB	5.7%
lib	127.9 KB	127.9 KB	1.9%
armeabi-v7a	31.7 KB	31.7 KB	0.5%
armeabi	21.7 KB	21.7 KB	0.3%
x86_64	17.4 KB	17.4 KB	0.3%
arm64-v8a	16.1 KB	16.1 KB	0.2%
x86	15.9 KB	15.9 KB	0.2%
mips	12.6 KB	12.6 KB	0.2%
mips64	12.5 KB	12.5 KB	0.2%
META-INF	55.7 KB	55.7 KB	0.8%
AndroidManifest.xml	2.2 KB	2.2 KB	0%
third_party	331 B	331 B	0%
build-data.properties	126 B	126 B	0%
jsr305_annotations	104 B	104 B	0%
error_prone	98 B	98 B	0%
androidsupportmultidexversion.txt	53 B	55 B	0%

Figure 1. Typical internal structure of an APK archive

The APK archive contains several subcomponents. Some of the common ones are described as follows:

- The **AndroidManifest.xml** file stores the basic information of Android applications, including package information like the package name, app ID, requested permissions, app components like activities, services, broadcast receivers, content providers, etc., and different hardware and software features [14].

- The **classes.dex** file contains the compiled bytecode composed of all Android classes which is compiled into single *.dex* file format. For the (now deprecated) Dalvik VM, the bytecode was optimised into an *.odex* file (which is pre-processed version of *.dex*) on first launch of the app [15]. The *.odex* files have been replaced by ELF files in more recent Android versions.

- The **lib/** folder holds compiled native code in its subfolders that are specific to the central processor unit (CPU) architecture (e.g. armeabi-v7a stores compiled code of all 32-bit ARM-based CPUs). The APK without such a folder is written entirely in Java or Kotlin and is able to run on each CPU architecture;

- The **resources.arsc** is an application resource table containing information of precompiled resources included in the application, such as their ID's, names and properties.

- The **res/** folder holds the application's resources, such as the image files, layouts, strings, sound files, styles etc.

- **META-INF**: This folder contains information about the application's signature and signed checksums for all the other files in the package [9, p. 304].

By default, every app runs in its own Linux process. The Android system starts the process when any of the app's components need to be executed, and then shuts down the process when it's no longer needed or when the system must recover memory for other apps. The Android system implements the *principle of least privilege*. That is, each app, by default, has access only to the components that it requires to do its work and no more. An app can request permission to access device data such as the device's location, camera, and Bluetooth connection. The user has to explicitly grant these permissions [11].

2.1.2 Basic structure of Android OS

Although there are several overlapping approaches for representing the system architecture of the Android OS, the overall structure incorporates different components that fall into several layers and sections, beginning from the topmost applications layer, descending to the application framework layer, libraries and runtime, finally reaching to the lowermost layer – the Linux-based kernel. Figure 2 depicts one perspective to visualize the concept of those layers as a software stack.

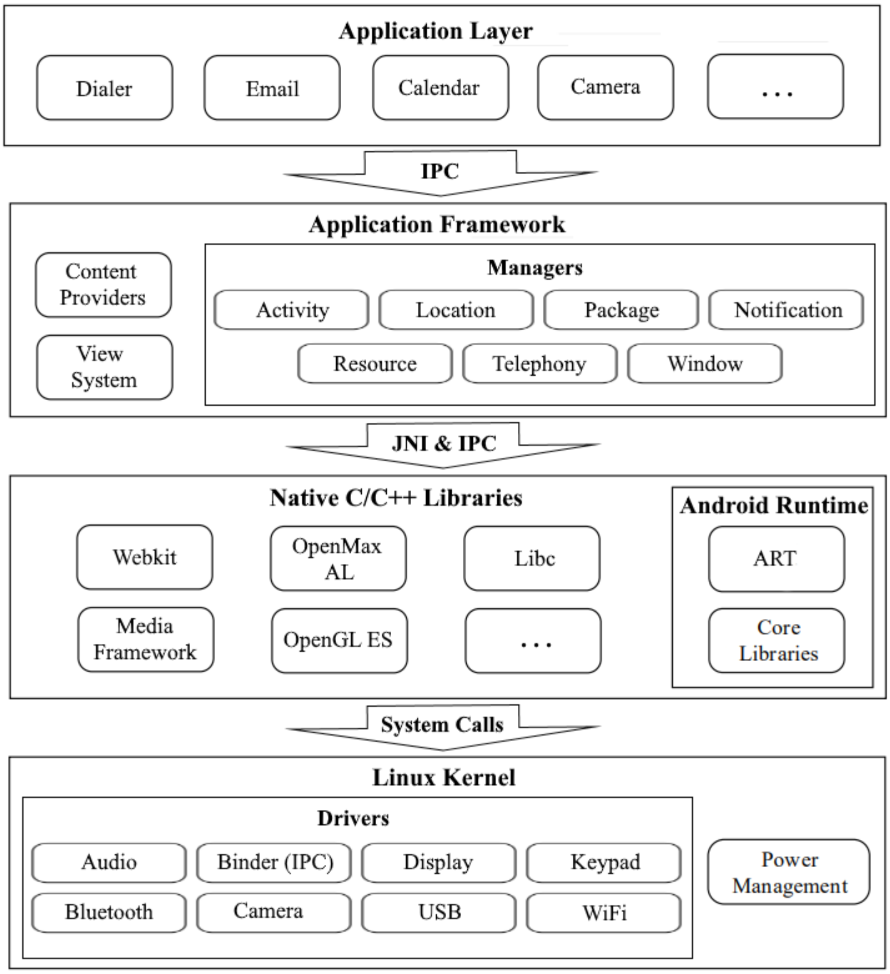


Figure 2. Android architecture

The topmost *applications* layer allows the user to interact directly with the device. It consists of both preinstalled and third-party apps. Data exchanges between apps and different system processes are handled by inter-process communication (IPC) mechanism, which is often misused by different Android malwares for communicating with system resources, as described in [16]. User-installed third-party apps can be downloaded from different digital distribution services (Google Play, Amazon App store etc.) or other repositories [9, p. 174].

Application framework is the layer responsible for handling the basic functioning of a phone, such as resource management, handling calls, and so on. This is the block through which the applications installed on the device directly talk to it [9, p. 174]. This framework layer includes the collection of application programming interfaces (API) written in Java. APIs are a set of coherent methods for apps to interact with device [15]. These APIs form the building blocks that developers need to create Android apps by simplifying the reuse of core, modular system components and a selection of services [17].

The next layer includes a set of native *libraries* in C/C++, used in various components of Android. Much of the low-level functionality relied upon by higher-level classes in the Android Framework is implemented by shared libraries and accessed via Java Native Interface (JNI), which allows both calling from Java code into native code and vice versa [18]. They are compiled in a native code processor and help devices to handle different kinds of data. Many of these libraries use open code projects and some of the examples of those libraries are Media Framework (supports recording of many video, audio and images formats), WebKit (web browser support), Surface Manager (manages graphical rendering of 2D and 3D representation), SQLite (a lightweight relational database engine), SSL (encryption services provider), and others [19, pp. 134-135].

In the same layer as libraries there is also *ART*, a distinctive section responsible for running applications on Android devices. ART is a runtime environment for each application providing the translation of application bytecode into native machine code instructions using a hybrid version of ahead-of-time (AOT) compilation. AOT compilation increases the device performance efficiency significantly by compiling entire applications into machine code upon their installation and not during their runtime. ART has replaced its less efficient predecessor – Dalvik VM – since the release of Android version 5.0 [9, pp. 172-174].

The lowermost layer of Android OS consists of a modified Linux-based *kernel*, which is described in the next subsection of this chapter.

2.1.3 Kernel and system calls

The *kernel* is a program that constitutes the central core of a computer operating system. It has complete control over everything that occurs in the system [20]. The Android OS is built on top of the Linux kernel, with some architectural changes made by Google. The primary reason for choosing the Linux kernel is the fact that it is a portable platform that can be compiled easily on different hardware. The kernel acts as an abstraction layer between the software and hardware present on the device [9, p. 171]. The Linux kernel contains code for all the different chip architectures and hardware drivers it supports [21].

The kernel is responsible for managing the core functionality of Android, such as process management, memory management, security and networking. Each version of Android has a different version of the underlying Linux kernel. Android 9 (Pie) version is known to use Linux Kernel 4.4, 4.9, or 4.14, whereas Android 10 targets Linux kernel 4.9, 4.14, or 4.19. The actual kernel depends on the individual device [9, p. 171]. The Android-specific kernel enhancement includes power management, shared memory drivers, alarm drivers, binders, kernel debugger and logger and low memory killers [22].

The android application takes the services of the kernel through the system calls. Whenever a user requests for services like call a phone in user mode through the phone call application, the request is forwarded to the Telephone Manager Service in the application framework. The Android runtime transforms the user request passed by the Telephone Manager Service to library calls, which results in multiple system calls to Android Kernel. While executing the system call, there is a switch from user mode to kernel mode to perform the sensitive operations. When the execution of operations requested by the system call is completed, the control is returned to the user mode [22].

In the Linux kernel, each machine architecture (*i.e*. x86-64 or ARMv8-A) can augment the standard system calls with its own. Consequently, the system calls available on one architecture may differ from those available on another. Nonetheless, a very large subset of system calls — more than 90 percent — is implemented by all architectures [23, p. 3]. Those architectures are described in the next section. Apparently, the number of Linux system calls available is not only dependent on machine architecture, but they seem to vary by kernel versions as well. For example, during this research, 335 different syscalls were identified for Linux kernel version 5.4 and 333 system calls for Linux kernel version 4.14 on x86_64 architecture. Those numbers were 277 and 276 for arm64 platform, respectively.

2.2 Hardware implementation and emulators

Different Android devices use different CPUs, which in turn support different instruction set architectures (ISA), which is the set of instructions that are written in machine code by each of the processor families. Each combination of CPU and ISA has its own Application Binary Interface (ABI) [7]. An ABI defines the binary interface between two or more pieces of software on a particular architecture. It defines how an application interacts with itself, how an application interacts with the kernel, and how an application interacts with libraries. ABIs are concerned with issues such as calling conventions, byte ordering, register use, system call invocation, linking, library behavior, and the binary object format. The calling convention, for example, defines how functions are invoked, how arguments are passed to functions, which registers are preserved and which are mangled, and how the caller retrieves the return value [23, p. 6]. Currently, the ABIs supported by Android are:

- **armeabi-v7a –** this ABI is for 32-bit ARM-based CPUs supporting *armeabi* instruction set;
- **arm64-v8a –** this ABI is for ARMv8-A based CPUs, which support the 64-bit *AArch64* architecture;
- **x86 –** this ABI is for 32-bit CPUs supporting the instruction set commonly known as *x86* or *i386*;
- **x86_64 –** this ABI is for CPUs supporting the instruction set commonly referred to as *x86-64* [7].

Android applications written in pure Java or Kotlin language have generic support by all of those ABIs. However, if the app is containing C/C++ code, then it must be compiled into a native library for each specific CPU architecture in order to be supported by targeted platform. The compiled architecture-specific native code is then packaged and stored in APK's *lib* folder, as described above in subsection 2.1.1.

In the near past, Android was also compatible with ISA called ‘Microprocessor without Interlocked Pipelined Stages’ (MIPS), but the competition is presently narrowed down to both 32-bit and 64-bit ARM and x86 technologies. It can also be concluded that ARM has prevailed as the primary CPU architecture used by all modern smartphones due to its optimal efficiency that is best suitable for mobile platforms. Despite that, Android apps are still sought-after for several x86_64 platforms such as Google’s Chromebooks [24]. Besides that, there is also another large technological segment, where Android continuously embraces the x86 architectures – the emulators.

An emulator is a software that mimics the hardware and software of the target device on a computer. They do this by translating the ISA of the target device to the one used by the computer to conduct testing using binary translation, thus mimicking the way how the target device works [25]. Although many people use emulators to play Android games on PC, the main purpose of the concept is intended for different developers and testers to aid and simplify their work. Emulators have also been widely used by malware analysts and researchers as a cheap, agile and adjustable platform – a convenient alternative to real mobile devices.

2.3 Android malware

This section recounts the general specifications and classification types of malicious applications targeting the Android OS. In addition, the latter part of the section briefly describes the employment of system calls as potential threat indicators in various malware detection methods available.

2.3.1 Taxonomy and installation methods

Throughout its evolution, malware has been labelled into different groups and subgroups according to their nature, function and origin. Such practice of classification, called taxonomy, is conducted mostly by different anti-virus (AV) vendors and cyber security companies. The following pages describe the relevant malware classification types while presenting several common examples.

Malware Categories. This embodies an overall categorization of malicious software based on the intentions and behaviours of the specimen. Well-known categories include [14]:

- **Ransomware**: Ransomware is a malware that locks the user's device to prevent the victim from accessing the data by using private key encryption until the victim pays a ransom.
- **Spyware**: A type of malware that covertly monitors user's personal information or activities and sends the collected information to the remote server without the user's awareness.
- **Trojan**: A type of malware that masquerades as a benign application but actually performs harmful activities.
- **Adware**: Adware presents unwanted advertisements to users.
- **Backdoor**: A type of malware that sets grounds for other malicious software by providing a backdoor in victimized devices.
- **Worm**: Worm is a piece of code that could be replicated and spread to other devices by the network [14].
- **Scareware**: Scareware is a malware that manipulates users into believing they need to download or buy malicious, sometimes useless, software [26].
- **SMS malware**: These Trojans use the text messaging services of a mobile device to send and intercept messages, which results in unexpected charges for users [27].

Malware Families. A malware family is a group of malware that shares common characteristics and behavioral traits. Adopting an attack or malicious behavior by inserting a payload (or more than one payload) might require using the same package names used for the attack. By frequent use of package names or other common characteristics, this becomes one identity (signature) of a group of malware (family) [28]. Family can be considered as a subcategory for malware types described above. A selection of families represented by the malware samples used for this research (*Mobok, WannaLocker, Plankton, etc.*) are briefly described in subsection 3.2.1.

Installation Methods. Android malware families can also be categorized by their installation method on victim's device (*i.e*. by different infiltration techniques). Following techniques are some of the more common examples of installation methods:

- **Repacking**: Repacking is the method of modifying and repackaging the APK file of registered benign application from Android application market and redistributing it. The modified APK contains code for stealing personal or financial information and causing damage to the device [22].
- **Update attack**: Instead of embedding the entire malware code, this technique includes just an update component in itself, which allows the entire malicious code to be downloaded and installed on the host device at the run time [22].
- **Drive-By-Download attack**: This approach refers to downloads that are launched automatically when a user is on a malicious website. The creators use spam or malvertising to bring users to the landing page that automatically launches a fake download [29].

2.3.2 Malware system calls as potential threat indicators

At the lowest level of the operating system, an application's functionality boils down to the tasks and services it requests the kernel to perform, through system calls [30]. All requests from the applications will pass through the System Call Interface before its execution through the hardware – a behavior which can give information about the intentions of the application [22]. Unless the malware itself is installed in the kernel of the operating system, the malware will have to use system calls to function. With this in mind, it is possible to trace system calls and analyze them for any discernible patterns [31].

Different methods and approaches have been established in order to determine the maliciousness of an application at least partially through its system calls. In a paper [31] for example, the authors separate malicious classification approaches (*signature-based detection, probabilistic detection, sequential detection*) and anomaly detection approaches (*specification-based detection* and *learning-based detection*) as well as countermeasures which the malware developers apply to bypass detections.

Despite the prevalent opinion that system calls *alone* are not sufficient enough to effectively detect the malicious behavior, as proposed in [30] and [32], they have a proven and eminent position in different detection approaches. The next section of this chapter describes the methods, where system calls have been utilized as features for malware detection.

2.4 Malware detection and system calls

Researching and countering malware is a decades old field of study for which the aspect of mobile malware has become just another, rapidly growing branch. Malware targeting Android OS is actually not as old as Android OS itself, since it took some time for the operating system to become popular and for malicious actors to adjust to it. Since the introduction of malware on Android OS, the effort to counter this threat has been forced to increase, and many academic and professional researchers are trying to keep the pace with malicious actors by developing new means and methods to detect and neutralize mobile malware threats.

The idea of using system call auditing as a means of malicious intrusion detection emerged in studies long before the rise and proliferation of personal mobile devices [33], [34]. Having become a common part of software behavioral studies, system call monitoring has also been carried over to research focusing on the detection of mobile malware during the last decade.

2.4.1 Conventional malware detection methods involving system calls

Traditionally, three different malware analysis methods are distinguished depending on the extraction and consideration of the features – static analysis, dynamic analysis and hybrid analysis. *Static analysis* researches properties of software that can be investigated by the inspection of the application's code to detect malicious intention without executing it (the latter rules out the use of pure static analysis for auditing the system calls, which are captured during the application's runtime).

In contrast to static analysis, *dynamic analysis* technique scrutinizes the behavior of the Android application during its execution by extracting and analyzing the dynamic features like process lists, system call traces, symbol table, a list of open files and network traffic [35]. In case of Android, when applied simplistically, it might provide limited coverage, which can be improved with stimulation by manually or automatically injecting events to trigger behaviors [15].

While the static analysis can be countered by code obfuscation, dynamic analysis could be circumvented by the methods of runtime sandbox detection. The *hybrid analysis* combines static and dynamic methods and analyzes the application to extract static and dynamic features, thereby improving recognition accuracy. However, the disadvantages are that the analysis and detection time is long, it takes up many system resources, and the calculation overhead is enormous [32]. System call traces are among the popular dynamic features for reasons described in subsection 2.3.2, and therefore used in studies including both dynamic and hybrid analysis methods.

2.5 Related works and previous research

The survey of literature described here involves system calls as features for malware detection-oriented research on Android OS. Most of the literature also reveals the collection platforms and methods used to gather system call data during the application runtime, although many of the studies remain rather vague in terms of their setup description. In regards of collection platforms, both real mobile devices and emulators have been used by researchers in the near past, but seldom in unison and typically not employing different real device models or different OS versions during a single study. Thus, the data gathered from a single source platform is largely generalized as accountable for all Android systems.

2.5.1 Related works

A research described in [36] proposed a malware detection system based on system calls performed during the boot process of the installed applications by monitoring and analyzing the sequences of system calls, using the methods based on Needleman–Wunsch global alignment approach and the Wilcoxon signed-rank test. It proved limited satisfactory results on the set of malware specimen that trigger its infection vector when the application is first started. The experiment used system call traces gathered from various unspecified real Android devices using different unspecified OS versions.

In the study [37], the authors present DroidRevealer – a real-time analysis system running on real devices using system call monitoring to detect malicious behavior. Although the experiments and evaluation of the tool were conducted on an unspecified emulator platform using Android version 4.4.2 and Linux kernel 3.4.0, the authors claimed that the tool performed equivalently on real devices with acceptable overhead.

In [38], the authors proposed an anomaly-based malware detection approach which analyzed relevant system calls in a unified manner, using a database of predetermined normal behavior for comparison. The study used Samsung Galaxy S real device and Android version 2.3 with Linux kernel 2.6.

In [39], the author proposed an approach to map system level behavior and Android APIs, based on the observation that system level behaviors cannot be avoided but sensitive Android APIs could be evaded. The mapping between system calls and Android APIs was intended to be used to detect malicious applications which try to evade Android APIs to conduct malicious actions. The study used Intel Xeon CPU host running Fedora 28, and an emulator (seemingly Android SDK) of Samsung S9 with Android 8.0.

A research described in [22] explored the behavior of malware samples from 10 prominent Android malware families based on system call patterns in comparison with benign applications. The study did not elaborate the specifications of the setup, besides the fact that an unspecified emulator was used.

In the paper [30], authors used an open-source ML software Weka with manually annotated behavior classes and system call features to analyze application behavior.

They reached to the conclusion that system calls were not sufficient features for mobile application behavior classification. Setup was well-described, as they used an Android SDK emulating Nexus 6, running Android 6.0.1 on a host PC with Intel Core-i7 CPU running Ubuntu 14.04.

In [40], the authors utilized the ML techniques on system calls accompanied by dynamic analysis to distinguish between malware and benign behaviors, showed the significance of system calls-based scanners in comparison to other attributes and concluded that carefully created adversarial samples are able to evade detection. The experiment was thoroughly described and it used both automated and manual event injections. Setup was also well-described (Android SDK emulator on Ubuntu 14.04 with Intel Core i7-4510U CPU) except for the details of the emulated platform itself.

In [41], three traditional feature-vector-based representations were implemented for Android system calls in order to propose a novel graph-based representation. The experiment concluded that the graph-based representations are able to improve the malware classification accuracy over the corresponding feature-vector-based representations from the same input. The study used Genymotion emulator on Ubuntu 14.04 desktop (AMD Opteron 6386 CPU) with no further specifications given about emulated platform.

In the study [42], using supervised ML on a system calls as features, a non-signature-based malware detector was proposed, that would not be vulnerable to mimicry attack typically used to defeat system-call based detectors. The experimental setup used emulator and was overall well-described (host with Ubuntu 14.04 with Intel Core i7 emulating Android virtual device, an armeabi-v7a image of Nexus 4 running Android 4.1.2).

In [43], the authors propose *MALINE*, a tool that uses frequency and dependency techniques based on system call tracking to perform automatic malware classification while applications are executed in a sandbox environment. The study used a set of host machines running Ubuntu 12.04 and employing Android SDK emulator using multiple x86 CPU/ABI images of unspecified Android device emulations. The authors admitted that the architectural differences between emulators and real Android devices might act as a potential threat to validity of the experiment.

In a research described in [44], a novel two-step feature selection approach based on highly relevant system calls is proposed to extract refined calls, which could discriminate malware from benign apps. To address the problem of higher dimensional attribute set, the authors derived suboptimal system call space by applying the proposed feature selection method to maximize the separability between malware and benign samples. The overall description of the experiment was very detailed apart from the experimental setup description, which only mentioned that an x86-based Android emulator was used for feature (system call) extraction.

In [45], the authors use two different feature models, the frequency vector and the co-occurrence matrix, to extract features from the system call sequence followed by appliance of different machine learning algorithms to identify Android malware and to measure the effectiveness of those distinct appliances. For the feature extraction, an unspecified real device running Android version 4.0.4 was used.

In the paper [16], the authors model the system call sequence generated by a malware application as a stationary first-order ergodic Markov chain and prove the existence of typical patterns which contain the malicious system call code of the application. They succeeded in finding the occurrence of common malicious system call codes in the system call sequence of several malware families. The only mention of the setup specification was the fact that an emulator having ARMv8 architecture was used for collecting the system call sequences.

In [46], the authors extracted features from malware and benign applications by sequencing the system calls and proposed a novel way of feature reduction using Gaussian dissimilarity to detect malware samples. The platform used for feature extraction was an unspecified real Android device with OS version 5.1.

In [47], the authors proposed a new feature selection mechanism that was named ‘selection of relevant attributes for improving locally extracted features using classical feature selectors’, or *SAILS*, which specifies at discovering prominent system calls from applications. They conducted an extensive analysis of ML and deep learning (DL) algorithms under diverse classifier parameters. SAILS resulted in improved values for evaluation metrics, compared to the conventional feature selection algorithms. The experiments were conducted on Ubuntu 18.04 platform with Intel Core i5-8250U CPU and an Android SDK emulator with an unspecified emulated platform, that was used for feature extraction.

2.5.2 Previous research

A study [8] conducted in Tallinn University of Technology in 2019 explored whether the selection of data source for the system calls may have an impact on the performance of the machine learning models. This study provided a comparative analysis of the data sets obtained from both an emulator-based and real device-based sources, as well as a demonstration of the impact of data source selection on detection models' performance.

At that time, the study showed that the system calls of 110 benign and 110 malware applications extracted from an emulator (Genymotion emulation of Samsung Galaxy S8 with Android 8.0) generated more distinguishable data in comparison with real device (Samsung Galaxy A6 with Android 8.0) and concluded that designers of detection models would have to pay attention to the data sources utilized in the various steps of the machine learning workflow.

This thesis research concentrates on the issue raised in the aforementioned study and tries to address the potential research gap in this field of study, as the possible impact of collection platforms on the applications' behavior seems to be insufficiently researched. In order to examine the extent of possible differences induced by different data sources in more detail, a wider selection of Android platforms has been introduced, while bringing the size of the dataset of benign and malicious applications down to a relatively limited scale (described in detail in chapter 3). The following chapter explains and describes the selection of application dataset, implemented platforms and the system call extraction procedure, while chapters 4 and 5 outline and discuss the extent of the results and their potential causes and effects.

3 Methodology

This chapter brings forth the data collection methodology and specifies the selection of the APK files, setup of the utilized devices and emulators (collection platforms) and the established criteria for different system settings. The latter part of the chapter expands upon the description of the implemented data collection procedure.

3.1 Goal setting

The related works examples described in chapter 2 were primarily focused on malware detection using ML, therefore they were distinguished by a great number of APK files (both benign and malicious) and mostly one (or in some cases several) Android OS-based testing platforms. This is quite understandable, as the main concern of those studies has been on ML models and on presenting the possibilities of training the ML model to detect malware on certain testing environment, whereas on most cases real Android device has not been the best choice for such large-scale research due to its limited functional resources.

In this study, the amount of both benign and malicious APK samples has been drastically reduced and the number of different testing platforms has been increased in order to approach the research questions in an optimal manner.

The general purpose was to put together a testing environment which includes widely used devices running modern and widespread Android versions, all which would also be emulated in more popular emulation platforms in order to collect the system calls of smaller set of various APK files (both malicious and benign, legacy and latest) for later comparison and analysis.

3.2 Selection of testing samples

For the task, 8 malicious and 8 benign APK files were used. The characteristics of the APK had to include:

- Each of the selected APK-s would be able to install and execute as an application on each testing platform;
- Both malicious and benign APK sets would also include natively compiled libraries compiled for different ABI-s;
- Malicious samples would represent both widely spread families and less known or custom-made specimen;
- Malicious samples would represent both legacy and recent specimen.

Malware APK samples were chosen from two different datasets that are publicly available for research purposes. Four samples of the most recent[1] malware specimen were collected from the *AndroidMalware_2020* dataset available in public GitHub repository [48]. The other four samples of somewhat older[2] malware samples were acquired from *CICAndMal2017* dataset made available by University of New Brunswick's Canadian Institute of Cybersecurity. *CICAndMal2017* dataset was originally established for Android malware research published in [49].

3.2.1 Malware

The selection of malware APK files from datasets was based on the four requirements described above. Samples were picked randomly and tested for the requirements. If any of the randomly chosen samples did not meet the requirements (*i.e.* was unable to install or execute on all the platforms) then it was discarded and other sample was chosen for testing. All final samples are representing different malware families, *i.e*. a subgroup of malware labelled and named by its malicious characteristics. Such labelling and naming of malware is mostly done by various AV vendors, who specialize in malware research and taxonomy. However, there are still numerous inconsistencies and ambiguities found in classification policies [50] which means those samples might also have different

1 Samples discovered as malware after January 2020

2 Samples discovered as malware between 2013 and 2017

names or labels given by different AV vendors or threat intelligence organizations. Here we are using the classification of the malware type and family name that has been used in respective datasets. Tables 1 and 2 show the main properties of used malware samples with short descriptions of each sample following after.

Table 1. Properties of malware APK files (type, family, package name, target ABI)

Code	Type	Family	Package name	Supported ABI
MW1	Backdoor	"Mobok"	com.awesome.fantasywallpaper	armeabi, armeabi-v7a, arm64-v8a, x86, x86_64
MW2	Spyware	"XploitSpy"	com.dotgears.flappybird	armeabi, armeabi-v7a, x86
MW3	Adware	"Hiddad"	com.CORONAVIRUS.OUTBREAK	all
MW4	Spyware	"Bahamut"	com.r.voiceofislam	all
MW5	Adware	"MobiDash"	fi.app4.fap	armeabi, armeabi-v7a, arm64-v8a, x86, x86_64, mips, mips64
MW6	Scareware	"Android.Spy.277"	com.os7.launcher.theme	armeabi, armeabi-v7a, x86, mips
MW7	Ransomware	"WannaLocker"	com.android.tencent.zdevs.bah	all
MW8	SMS malware	"Plankton"	com.badguys.japansound	all

Table 2. Properties of malware APK files (dataset, hash, first appearances, size)

Code	Dataset	MD5 hash	Upload to VT[1]	Upload to Koodous[2]	APK Size
MW1	AndroidMalware_2020	83763edd2d2e5d380df5c777cc9cdc24	2020-02-14	2020-02-15	5.26 MB
MW2	AndroidMalware_2020	117e1331306fec02b1ffe6b68d148cc9	2020-05-06	2020-05-06	1.34 MB
MW3	AndroidMalware_2020	ec2b4ad861c0dbef1404713d9eac48a4	2020-03-13	2020-03-13	11.27 MB
MW4	AndroidMalware_2020	9368dd657e410f8a9ba2b71c95cc0777	2020-08-26	2020-08-26	10.48 MB
MW5	CICAndMal2017	08d05f01671f788e9c17a9ffca0547b0	2016-02-09	2016-02-09	4.20 MB
MW6	CICAndMal2017	2c5f158e2be5b0a67fe7378d6cff0d2d	2015-12-10	2015-12-11	4.51 MB
MW7	CICAndMal2017	762138e933a681628ceab29d8e5a96a2	2017-07-25	2017-07-26	11.68 MB
MW8	CICAndMal2017	0378f0cf4e7241a4c0f5a0722e601638	2013-08-07	2019-07-15	17.27 MB

MW1 (Backdoor "Mobok"): The first variants of this malware were discovered in 2019. Its different versions have been masked as photo or image editing apps which have also known to have been available in Google Play market due to malware's evasive characteristics. Besides giving the attacker a potentially full control over the victim's device, this malware also steals user's personal data and uses it to unknowingly sign them up to paid subscription services [51]. This gives the exploiter a financial gain

1 Date of first submission to VirusTotal.com
2 Date of first submission to Koodous.com

in expense of the victim. Mobok was ranked as top 13th mobile malware by Kaspersky's Securelist in 2020 [52].

MW2 (Spyware "XploitSpy"): This new malware is basically a very capable Android monitoring kit which is available as an open-source downloadable toolset. It has a built-in APK builder which auto-builds the malicious APK file that could be apparently repackaged and modified to be disguised as a random benign-looking application. If successfully installed on a victim's device, it gives the exploiter the access to victim's logs, files, microphone and other features [53].

MW3 (Adware "Hiddad"): Hiddad is one of the many variants of adware out in the wild and it has been detected since 2016 [54]. Adware is usually not directly malicious but rather annoying and unethical due to its hidden and undeclared essence. The unwanted ad-code is embedded into a regular applications and distribution service providers could often remain oblivious to their existence for long periods of time. Different versions of Hiddad have resided in Google Play for several consecutive periods [55]. There seem to exist also other versions of malware named Hiddad which have been identified having features of a backdoor malware [56].

MW4 (Spyware "Bahamut"): Bahamut, which also gives name to this malware sample, is actually a threat actor or advanced persistent threat (APT) that has been labelled by researches as a sophisticated hack-for-hire group targeting individuals and organisations mainly in Middle-East and South Asia [57]. This particular malware APK sample was discovered in late 2020 and it contained a number of spyware components which aimed at extracting sensitive user related information (call logs, contacts, device info, media files etc.) and sending it back to the attackers' server [58].

MW5 (Adware "MobiDash"): This malware is another example of aggressive adware where ad-code is repackaged into some legitimate APK. It displays pop-up advertisements every time the user of the device unlocks the screen. To distract users and evade different dynamic analysis, this malware would wait days or even weeks after installation before executing its malicious ad-code [59], [60].

MW6 (Scareware "Android.Spy.277"): This malware trojan has been classified as scareware due to its features that besides stealing the victim's data as a regular spyware, it would also display advertisements or fake warnings (for example about an overpowered battery or malware infection etc.) and offers the victim to download the next (malware) application as a "solution". Already by the time of its discovery back in 2016 it was embedded into more than 100 different applications – a number that has been likely grown since then [61].

MW7 (Ransomware "WannaLocker"): First appearing in 2017, this file-encrypting malware reminds the famous Windows ransomware WannaCry made available for Android OS. Targeting Chinese users and originally spreading in Chinese forums, this malware changes the background wallpaper and actually encrypts small-sized (up to 10 KB) files stored on different locations on device's storage. The home page of the application then demands a small amount of money from the victim for the decryption keys [62].

MW8 (SMS malware "Plankton"): The CICAndMal2017 dataset and some previous research [63] classify Plankton as SMS-malware, while its properties and behaviour would suggest it to be actually a spyware. Plankton is an older example of a malware family repackaged in legitimate apps that would steal information and attempt to open a backdoor on Android devices. When executed, it would try to collect the device ID and permissions and send them to a remote server. This server would then push a backdoor payload onto the device, which would use the host application's permissions to collect additional information and send it back to the server [19, p. 17], [64].

3.2.2 Benign applications

All of the benign APK files were collected from CICAndMal2017 dataset and were tested in VirusTotal to confirm their non-malicious nature. Those files were picked in a random manner with only prerequisites that they would be able to correctly install and execute on each testing platform and half of them would include native libraries similarly to their malware counterparts. Their properties are described in Table 3 and Table 4.

Table 3. Properties of benign APK files (package name, target ABI)

Code	Category	Package name	Supported ABI
BN1	Audiobook player	ak.alizandro.smartaudiobookplayer	armeabi, armeabi-v7a, x86
BN2	Timesheet organizer	com.aadhk.time	armeabi, armeabi-v7a, x86, mips
BN3	Graphic design tool	cover.designer.maker.scopic	armeabi, armeabi-v7a, arm64-v8a, x86, x86_64, mips, mips64
BN4	PDF converter	pdfConversion.Droid	armeabi, armeabi-v7a, x86
BN5	QR code scanner	app.qrcode	all
BN6	Camping database	au.com.angryrobot.wikicamps	all
BN7	Diary	com.adpog.diary	all
BN8	Alarm Clock	com.alarmclock.xtreme.free	all

Table 4. Properties of benign APK files (dataset, hash, first appearances, size)

Code	Dataset	MD5 hash	Upload to VT[1]	Upload to Koodous[2]	APK Size
BN1	CICAndMal2017	ea30d7cc4c1dd7ad31bc32156fd2025b	2017-02-16	2017-02-17	4.65 MB
BN2	CICAndMal2017	0ea05f7634ac6b1003a774d3d7f22103	2016-09-07	2016-09-07	7.39 MB
BN3	CICAndMal2017	33b2fcb832c67a6c69a5cc05b0a44e3f	2017-02-07	2017-05-05	24.11 MB
BN4	CICAndMal2017	de76fdefa4a223d38162c8d349752720	2016-12-12	2016-12-12	23.84 MB
BN5	CICAndMal2017	90c81f6acc471d922fee136880eda641	2017-02-13	2017-02-13	3.02 MB
BN6	CICAndMal2017	1607aef3d413ddd619c0248b07dd0087	2016-12-17	2016-12-17	9.42 MB
BN7	CICAndMal2017	944761948baeddf0e503325bf5e41ca4	2016-07-19	2016-07-21	2.24 MB
BN8	CICAndMal2017	d93520ceee3ce2a3ff29a38cd7f6428c	2015-04-02	2015-08-08	7.61 MB

3.3 Selection of implementation platforms

To measure the possible differences of system calls on different platforms, three mobile phones running two different Android OS versions (9 and 10) were selected as benchmarks for this research. Those phones were also emulated as accurately as possible in both Android Studio 4.1.2 Android Virtual Device (AVD) Manager and Genymotion Desktop 3.2.0 emulator. All real devices were rooted with Magisk Manager, while emulated devices had root available through system images. All smartphones were using 4G connection (one SIM-card for each platform) and had 16 GB microSD card inserted as an additional internal storage unit.

1 Date of first submission to VirusTotal.com
2 Date of first submission to Koodous.com

3.3.1 Implementation platform specifications

The host platform for all the emulations was a PC with Intel Core i7-8665U x86_64 CPU and 32 GB RAM running Ubuntu 20.04 LTS with 5.4.0-70-generic kernel version. For internet connection during emulation testing, a 4G USB-modem was used. This section provides the overview about the parameters of given testing platforms, including the details of their central processing unit (CPU), system on a chip (SoC) model and kernel version. Most of this information was gathered by examining *build.prop* system file and using AIDA64 system information application.

Table 5, Table 6 and Table 7 reflect the specifications of used smartphones and their emulations. Since PH1 and PH2 are running a 32-bit OS on a 64-bit CPU, system images with x86 ABI were chosen for EA1 and EA2 emulations in AVD manager to better match the kernel architecture, while PH3 running 64-bit OS was emulated as EA3 with x86_64 system image[1]. Such distinction was not possible with Genymotion, which offered only 32-bit x86 system images.

Table 5. Smartphone hardware and OS specifications

Code	Device	System	CPU	Operating System
PH1	Samsung Galaxy A20e	**Model:** SM-A202F **RAM:** 3 GB **Storage:** 32 GB **Display:** 720 x 1560 5.8″ (296 dpi)	**SoC model:** Samsung Exynos 7 Octa (7885) **Instruction set:** 64-bit ARMv8-A **Supported ABIs:** arm64-v8a, armeabi-v7a, armeabi	**Android version:** 9 (Pie) **API level:** 28 **Kernel architecture:** armv8l (32-bit) **Kernel version:** 4.4.111-17594784
PH2	Samsung Galaxy A40	**Model:** SM-A405FN **RAM:** 4 GB **Storage:** 64 GB **Display:** 1080 x 2340 5.9″ (437 dpi)	**SoC model:** Samsung Exynos 7 Octa (7904) **Instruction set:** 64-bit ARMv8-A **Supported ABIs:** arm64-v8a, armeabi-v7a, armeabi	**Android version:** 10 **API level:** 29 **Kernel architecture:** armv8l (32-bit) **Kernel version:** 4.4.177-20196810
PH3	Xiaomi Redmi Note 8 Pro	**Model:** Note 8 Pro **RAM:** 6 GB **Storage:** 64 GB **Display:** 1080 x 23406.53″ (395 dpi)	**SoC model:** MediaTek Helio G90T (MT6785T) **Instruction set:** 64-bit ARMv8-A **Supported ABIs:** arm64-v8a, armeabi-v7a, armeabi	**Android version:** 10 **API level:** 29 **Kernel architecture:** aarch64 (64-bit) **Kernel version:** 4.14.141-g30b7a06

1 At the time of conducting this research, Android SDK did not offer ARM system images for Android 9 and Android 10. Android 7 version of ARM image was tested, but it failed to properly function on a x86_64 host PC.

Table 6. Android Studio emulation hardware and OS specifications

Code	Emulation	System	CPU	Operating System
EA1	Android Virtual Device Samsung Galaxy A20e	**Model:** AOSP on IA emulator **RAM:** 3 GB **Storage:** 32 GB **Display:** 720 x 1560 5.8″ (320 dpi)	**SoC model:** Android virtual processor **Instruction set:** 32-bit x86 **Supported ABIs:** x86, armeabi-v7a, armeabi	**Android version:** 9 (Pie) **API level:** 28 **Kernel architecture:** i686 (32-bit) **Kernel version:** 4.4.124+
EA2	Android Virtual Device Samsung Galaxy A40	**Model:** Android SDK built for x86 **RAM:** 4 GB **Storage:** 64 GB **Display:** 1080 x 2340 5.9″ (480 dpi)	**SoC model:** Android virtual processor **Instruction set:** 32-bit x86 **Supported ABIs:** x86	**Android version:** 10 **API level:** 29 **Kernel architecture:** i686 (32-bit) **Kernel version:** 4.14.175-g6f3fc9538452
EA3	Android Virtual Device Xiaomi Redmi Note 8 Pro	**Model:** Android SDK built for x86_64 **RAM:** 6 GB **Storage:** 64 GB **Display:** 1080 x 2340 6.53″ (320 dpi)	**SoC model:** Android virtual processor **Instruction set:** 64-bit x86 **Supported ABIs:** x86_64, x86	**Android version:** 10 **API level:** 29 **Kernel architecture:** x86_64 (64-bit) **Kernel version:** 4.14.175-g6f3fc9538452

Table 7. Genymotion emulation hardware and OS specifications

Code	Emulation	System	CPU	Operating System
EG1	Genymotion emulator Samsung Galaxy A20e	**Model:** Emulated A20e **RAM:** 3 GB **Storage:** 32 GB **Display:** 720 x 1560 5.8″ (300 dpi)	**SoC model:** Intel Core i7-8665U CPU @ 1.90 GHz (host) **Instruction set:** 32-bit x86 **Supported ABIs:** x86	**Android version:** 9 (Pie) **API level:** 28 **Kernel architecture:** i686 (32-bit) **Kernel version:** 4.4.157-genymotion-gbca5a41
EG2	Genymotion emulator Samsung Galaxy A40	**Model:** Emulated A40 **RAM:** 4 GB **Storage:** 32 GB **Display:** 1080 x 2340 5.9″ (440 dpi)	**SoC model:** Intel Core i7-8665U CPU @ 1.90 GHz (host) **Instruction set:** 32-bit x86 **Supported ABIs:** x86	**Android version:** 10 **API level:** 29 **Kernel architecture:** i686 (32-bit) **Kernel version:** 4.4.157-genymotion-gbca5a41
EG3	Genymotion emulator Xiaomi Redmi Note 8 Pro	**Model:** Emulated Note 8 Pro **RAM:** 6 GB **Storage:** 32 GB **Display:** 1080 x 2340 6.53″ (400 dpi)	**SoC model:** Intel Core i7-8665U CPU @ 1.90 GHz (host) **Instruction set:** 32-bit x86 **Supported ABIs:** x86	**Android version:** 10 **API level:** 29 **Kernel architecture:** i686 (32-bit) **Kernel version:** 4.4.157-genymotion-gbca5a41

3.3.2 Implementation platform settings

Emulations were created to imitate the real smartphone devices as accurately as possible using the setup mechanisms made available by the emulation programs. This means that all emulated specifications (like the OS image, number of CPU cores, RAM size, display resolutions and dpi, storage and SD card size etc.) were all selected according to the real devices' specifications, if applicable. Some minor deviations remained, however, since some features like storage size (in Genymotion) or display dpi sizes were offered in a fixed configurations. In such fixed cases, if selectable, the nearest possible value to the real specification was used.

Within operating system, each testing platforms were also given as similar settings as possible to create a setting similar to 'average' user. This means that consent was given to provide diagnostics and usage data, location services were enabled, WiFi was enabled (although only 4G connection was used), Google Play Services were enabled and logged in to with a Google account. The exceptions here were Android Studio emulations, as the system images only permitted Google APIs but not Google Play Services, while Genymotion allowed to use Google Play through Open Gapps widget. On every platform, Play Protect was disabled to allow untampered installation and execution of malware applications.

3.4 Data collection procedure

The purpose of the data collection in this thesis research was to trace and log system calls for each installed and executed application from each Android platform under the same conditions for further analysis. For this task the ADB (Android debug bridge) tool was used in concert with the Monkey and *strace* tools. The details of the collection process are described below.

3.4.1 Android Debug Bridge

Android Debug Bridge or ADB is a versatile command-line tool that lets the user to communicate with a device. The *adb* command facilitates a variety of device actions, such as installing and debugging apps, and it provides access to a Unix shell to run a variety of commands on a device. It is included in the Android SDK Platform-Tools package [65]. All interactions with testing platforms during the data collection process were conducted using this tool. This required enabling the ‘developer options’ mode on each device’s settings.

3.4.2 Application Exerciser Monkey

The Monkey tool was used to execute the chosen application packages through ADB. This program runs on the emulator or device and generates pseudo-random streams of user events such as clicks, touches, or gestures, as well as a number of system-level events. This tool is specifically used by developers to stress-test applications in a random yet repeatable manner [66]. The exact usage of the Monkey tool in this research is demonstrated in subsection 3.4.4.

3.4.3 *Strace* tool

Being a potent tool, *strace* is a diagnostic, debugging and instructional userspace utility for Linux. It is used to monitor and tamper with interactions between processes and the Linux kernel, which include system calls, signal deliveries, and changes of process state [67]. In essence, *strace* helps the user to trace the interactions between user process and system kernel. Its usefulness in malware detection lies in the fact that unless the malware itself is installed in the kernel of the operating system, it needs to use system calls to function, which in turn can be tracked (and logged) with *strace* [31]. The exact usage of *strace* jointly with Monkey tool in this research is demonstrated in subsection 3.4.4.

There was no *strace* tool originally available on the real devices’ system binaries. After rooting, it became possible to add *strace* to their /*system*/*bin* folders. This required the system to be temporarily mounted as ‘writable’ and after successful installation of *strace* to be re-mounted again as ‘read-only’. An *strace* binary precompiled for ARM platforms was implemented by adding it into the /*system*/*bin* folder.

3.4.4 Collection process

After implementing the predetermined OS settings on a specific device or emulation, the platform was ready to be used for system call data collection. It must be noted that after each collection of malware sample's system calls, the read-only memory (ROM) on the real device was re-flashed with clean system image. For benign samples, system factory reset was implemented after each collection round. The devices were re-rooted and the predetermined OS settings were then again applied. This policy ensured that the original unvaried system state was restored before each collection, but made the overall collection process exceedingly time-consuming. In case of emulators, snapshots (Android Studio) and cloning (Genymotion) were implemented.

Table 8. Modes of application execution

Code	**Mode of execution**	**Description**
1E	Execution only	Plain execution of application. No interaction added to ongoing process.
50E	50 events injected	Execution of an application with 50 additional pseudo-random events injected through Monkey tool.

The purpose of the collection was to install the testing APK, execute the application with the Monkey tool while attaching the *strace* to the launched application process, let the application run for 5 minutes while collecting and saving the system call logs into an output file and finally detach the *strace*, close the application and pull the output file to host computer. Such procedure was employed twice for each sample – first time with only application execution and secondly with 50 pseudo-random events generated by Monkey tool during the collection process (explained in Table 8). For both installation and collection procedures, two simple bash scripts with necessary commands were generated, which are presented below.

Installation:

```
#!/bin/bash
# Command: ./scriptname.sh apkname.apk

# Install the given application to connected device.
adb install $1

# Print the application’s package name.
aapt dump badging $1 | awk -v FS="'" '/package: name=/{print $2}'
```

The last line was used to print the application package name on command line interface which would then be used in conjunction with the following collection script.

Execution and collection[1]:

```
#!/bin/bash
# Command: ./scriptname.sh app.package.name

# Create log collection folder.
adb shell su -c mkdir -m 777 /sdcard/Download/stracelog

# Launch app (event 1) for 300 sec & log the syscalls.
adb shell su -c monkey -p $1 1 && adb shell su -c timeout 300 strace -o
/sdcard/Download/stracelog/syscall-$1.txt -Cittr -p $(adb shell ps -A | grep
$1 | awk '{print $2}')
sleep 3

# Pull the log to host.
adb pull /sdcard/Download/stracelog/syscall-$1.txt /destination/path/

# Close & clear user data.
adb shell su -c pm clear $1
```

The collection script created the log collection folder, executed the application with the Monkey tool including the given pseudo-random events (1 shown in this example) and attached *strace* for 5 minutes onto that application’s parent process identifier (PID) while saving the collected log to a given file with the corresponding package name. When *strace* was detached after 5 minutes, the script waited for 3 seconds and pulled

1 Using the ADB shell, the ‘su -c *command*’ was required when interacting with real devices, while the emulations required ‘su 0 *command*’.

the saved log file to host PC, closed the application and deleted its user data, but kept the application. The process was then repeated for the second time with the script that had 50 pseudo-random events injected by Monkey. Any other manipulation with the device or emulator was not implemented during the collection process. There were rare occurrences when the application crashed or was closed by a random Monkey event during the collection process. On those cases, the whole procedure was repeated according to the predefined criteria.

However, there was one forced deviation from this policy – prior to executing several tested applications in any Android 10 platform, a non-skippable permissions screen had to be passed by tapping/clicking the 'Continue' button (see Figure 3 for an example). The reason for this factor was one of the several privacy improvements added to Android 10 compared to Android 9. As described in [68], if the particular application targets Android 5.1 (API level 22) or lower, users would see a permissions screen when using that app on a device that runs Android 10 or higher for the first time.

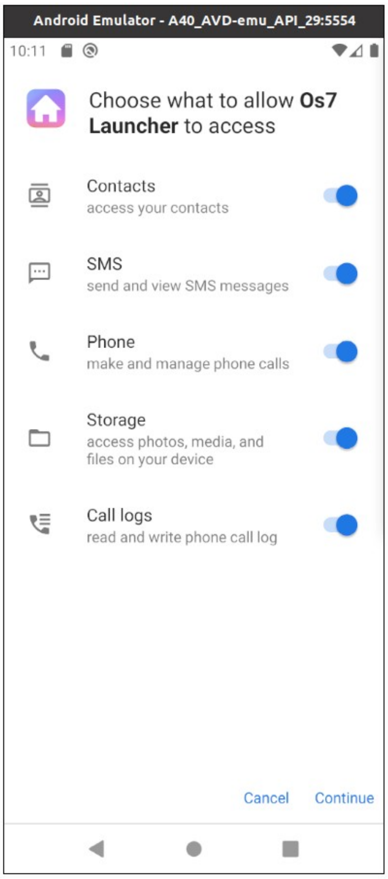


Figure 3. Android 10 legacy apps' permission check (MW6 example)

In order to collect the syscall data from legacy applications, the 'Continue' button was pressed with no other manipulations. The legacy applications affected on Android 10 platforms by this feature were MW2, MW5, MW6, MW7, MW8, BN2, BN6, BN7 and BN8. The possible impact of this provision is discussed in chapter 5.

As mentioned, due to the policy of ROM re-flashing (in case of malware samples) or system resetting (in case of benign samples) and readjusting the OS settings according to the predefined criteria, the system calls extraction from the real devices was an exceedingly time-consuming endeavor compared to the snapshot restoring or cloning possibilities available for emulated platforms. The result of the extraction process was a collection of system call logs of each malicious and benign application from each platform with both 1 event (execution only) and 50 events (pseudo-random injections by monkey tool). Table 9 shown below depicts the collection matrix in a simplified way with both types of collected logs[1] included for each extraction combination.

Table 9. Collection matrix

		REAL ANDROID PHONES			ANDROID SDK EMULATOR			GENYMOTION EMULATOR		
APK sample	**APK type**	**PH1**	**PH2**	**PH3**	**EA1**	**EA2**	**EA3**	**EG1**	**EG2**	**EG3**
MALWARE:										
MW1	native libs	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E
MW2	native libs	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E
MW3	plain	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E
MW4	plain	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E
MW5	native libs	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E
MW6	native libs	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E
MW7	plain	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E
MW8	plain	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E
BENIGN:										
BN1	native libs	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E
BN2	native libs	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E
BN3	native libs	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E
BN4	native libs	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E
BN5	plain	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E
BN6	plain	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E
BN7	plain	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E
BN8	plain	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E	1E + 50E

After the collection process, there were 144 logs from malicious applications and 144 from benign applications gathered from both plain execution and 50 pseudo-random

1 *1E* = system calls log with plain execution; *50E* = system calls log with 50 pseudo-random events

event injection collections. In the following phase, all collected logs were examined and analyzed. The gathered data, analysis process and the results of the analysis are described in chapter 4.

4 Results

The gathered 288 log files resulted altogether in a large amount of raw data, with a variety of options to examine and analyse it. The results indicated differences in varying proportions in their length of sequences and the amount of different system calls initiated by application during the 5-minute period. In general, this was predictable, as different applications perform in different manners. However, as the purpose of the study was to discover and report possible behavioral differences between different Android platforms, the data originating from malicious and benign applications was to be compared in relation to each real device and their emulated counterparts. Due to the extensive amount of data, it was not feasible to compare the differences in the level of individual unique system calls. In order to limit the scope of the study, the gathered data was examined from the perspective of call summaries, while also considering the amounts of unique system calls initiated on different platforms. Other possible methods, such as comparing the sequence alignments of various system calls, were not applied.

4.1 Initial examination of raw data

The typical output sample of the log files is shown as a fragment from a larger output in Figure 4 below. This example originates from MW3 application collected on PH1 platform (timestamps have been removed from the example).

```
...
getuid() = 10168
epoll_pwait(42, [{EPOLLIN, {u32=74, u64=74}}], 16, 0, NULL, 8) = 1
read(74, "\0\0\0\0\1\0\0\0", 8) = 8
timerfd_settime(75, TFD_TIMER_ABSTIME, {it_interval={tv_sec=0, tv_nsec=0}, it_value={tv_sec=706, tv_nsec=555045000}}, NULL) = 0
ioctl(44, BINDER_WRITE_READ, 0x7ff290e6a8) = 0
getuid() = 10168
epoll_pwait(42, [{EPOLLIN, {u32=74, u64=74}}], 16, 0, NULL, 8) = 1
read(74, "\2\0\0\0\0\0\0\0", 8) = 8
write(74, "\0\0\0\0\1\0\0\0", 8) = 8
getuid() = 10168
openat(AT_FDCWD, "/dev/ashmem", O_RDWR|O_CLOEXEC) = 172
fstat(172, {st_mode=S_IFCHR|0666, st_rdev=makedev(10, 67), ...}) = 0
ioctl(172, ASHMEM_SET_NAME, 0x7ff290d8b8) = 0
ioctl(172, ASHMEM_SET_SIZE, 0x2000) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE, 172, 0) = 0x7184cb9000
close(172) = 0
mmap(NULL, 1073152, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_NORESERVE, -1, 0) = 0x70ce8f6000
mprotect(0x70ce8f6000, 4096, PROT_NONE) = 0
prctl(PR_SET_VMA, PR_SET_VMA_ANON_NAME, 0x70ce8f6000, 4096, "thread stack guard") = 0
mmap(NULL, 20480, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x70fb3f6000
prctl(PR_SET_VMA, PR_SET_VMA_ANON_NAME, 0x70fb3f6000, 20480, "bionic TLS guard") = 0
mprotect(0x70fb3f7000, 12288, PROT_READ|PROT_WRITE) = 0
...
```

Figure 4. Example of a system call log (fragment)

The *strace* tool gives the user an option to display a summary of system calls at the end of the log output using the -C flag as part of the command (or -c for summary only). An example of the summary of the same extracted log as above is shown in Figure 5.

```
...
ioctl(44, BINDER_WRITE_READ, 0x7ff290f648) = 0
epoll_pwait(42,  <detached ...>
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 50.19    0.528000         771       684           epoll_pwait
 13.31    0.140000         286       489           ioctl
 12.17    0.128000         603       212        42 futex
  7.60    0.080000         197       405         1 read
  5.70    0.060000         163       367           write
  4.94    0.052000         151       344           getuid
  1.90    0.020000         106       187       131 recvfrom
  1.52    0.016000         122       131           sendto
  1.14    0.012000        1000        12           timerfd_settime
  0.76    0.008000         533        15           mmap
  0.38    0.004000        1333         3           clone
  0.38    0.004000         333        12           prctl
  0.00    0.000000           0         4           openat
  0.00    0.000000           0         5           close
  0.00    0.000000           0         1           writev
  0.00    0.000000           0         9           fstat
  0.00    0.000000           0         5           munmap
  0.00    0.000000           0        10           mprotect
  0.00    0.000000           0        27           madvise
------ ----------- ----------- --------- --------- ----------------
100.00    1.052000                  2922       174 total
```

Figure 5. Example of a single *strace* log summary

These summaries include, among other data, the list of recorded unique system calls, the amount of each unique calls initiated, and also the total sum of all recorded system calls during the tracing process. These total sums and the number of unique system calls from each application were used as primary data for further analysis. The total sums and data was extracted from each of the logs and arranged into separate matrices. Those figures, representing each application recorded on each platform, are shown for both 1 event and 50 event collections in Table 10 and Table 11, respectively[1].

Table 10. Extraction results – general overview (execution only)

1 EVENT	**REAL ANDROID PHONES**						**ANDROID SDK EMULATOR**						**GENYMOTION EMULATOR**					
	PH1		PH2		PH3		EA1		EA2		EA3		EG1		EG2		EG3	
Sample	*Total*	*n*	*Total*	*n*	*Total*	*n*	*Total*	*n*	*Total*	*n*	*Total*	*n*	*Total*	*n*	*Total*	*n*	*Total*	*n*
MALWARE:																		
MW1	1631	**13**	1819	**30**	3580	**38**	4434	**37**	8881	**36**	7674	**36**	3018	**36**	2385	**34**	2619	**34**
MW2	7793	**50**	6604	**51**	15044	**50**	3463	**43**	5427	**46**	3485	**45**	2931	**47**	2235	**47**	2206	**46**
MW3	3549	**19**	5916	**44**	16147	**46**	9208	**43**	1204022	**47**	12155	**43**	10693	**49**	10254	**47**	10361	**47**
MW4	617	**27**	672	**26**	1094	**33**	2020	**28**	3280	**29**	1876	**25**	866	**29**	920	**28**	971	**28**
MW5	6243	**30**	6825	**34**	6113	**30**	4539	**48**	846	**20**	313	**12**	12937	**53**	8646	**33**	8088	**32**
MW6	594055	**36**	490404	**32**	4330	**19**	265683	**38**	581141	**30**	256816	**30**	186062	**45**	197392	**29**	171947	**28**
MW7	17732	**38**	16900	**38**	17646	**34**	19297	**30**	34388	**36**	14457	**29**	14289	**12**	15845	**36**	13668	**33**
MW8	4981	**28**	5049	**35**	6971	**26**	9459	**50**	49116	**26**	10238	**24**	7217	**52**	87740	**50**	249006	**26**
BENIGN:																		
BN1	2417	**32**	2912	**36**	9199	**38**	3569	**30**	7553	**32**	4664	**32**	1710	**36**	1710	**34**	1420	**34**
BN2	627	**19**	1497	**25**	4278	**16**	4585	**49**	515	**17**	263	**16**	3742	**56**	3031	**56**	401	**18**
BN3	553	**24**	4420	**49**	7575	**51**	6598	**48**	10889	**48**	6667	**45**	6357	**50**	4989	**48**	4892	**48**
BN4	2287	**21**	206184	**54**	224182	**56**	23004	**52**	189200	**52**	9478	**49**	20044	**51**	6523	**50**	7424	**50**
BN5	61	**8**	4602	**51**	7387	**52**	7650	**50**	13342	**51**	7150	**49**	6263	**52**	4676	**51**	4827	**51**
BN6	2709	**24**	1886	**24**	1477	**24**	3325	**36**	5247	**24**	1740	**23**	3947	**40**	1096	**22**	1326	**22**
BN7	2477	**37**	2096	**37**	2206	**37**	3815	**37**	4924	**24**	3642	**37**	2173	**40**	1745	**36**	1624	**36**
BN8	147	**16**	1121	**27**	248	**17**	7382	**53**	402	**18**	237	**17**	3908	**55**	373	**21**	796	**24**

Table 11. Extraction results – general overview (50 events injected)

50 EVENTS	**REAL ANDROID PHONES**						**ANDROID SDK EMULATOR**						**GENYMOTION EMULATOR**					
	PH1		PH2		PH3		EA1		EA2		EA3		EG1		EG2		EG3	
Sample	*Total*	*n*	*Total*	*n*	*Total*	*n*	*Total*	*n*	*Total*	*n*	*Total*	*n*	*Total*	*n*	*Total*	*n*	*Total*	*n*
MALWARE:																		
MW1	4481	**15**	1910	**30**	5845	**31**	6766	**31**	10716	**11**	6626	**35**	5148	**31**	6199	**30**	2966	**11**
MW2	8010	**50**	7002	**51**	6164	**21**	3635	**46**	6783	**47**	4145	**46**	3144	**47**	2523	**47**	2508	**47**
MW3	2922	**19**	2334	**22**	5155	**29**	154294	**27**	11299	**26**	8283	**26**	7065	**26**	5666	**26**	3047	**33**
MW4	576	**26**	603	**26**	615	**26**	1834	**26**	2497	**26**	1806	**26**	361	**24**	642	**25**	670	**25**
MW5	6119	**25**	6188	**30**	5386	**22**	342	**10**	357	**12**	458	**16**	10161	**30**	8859	**34**	7225	**24**
MW6	598843	**35**	502948	**36**	1437	**12**	259148	**29**	584115	**30**	128754	**14**	192390	**29**	210356	**29**	170547	**28**
MW7	16988	**38**	16157	**36**	13642	**36**	16232	**37**	28584	**34**	14956	**28**	14041	**11**	13862	**28**	13305	**12**
MW8	4543	**29**	4295	**26**	72619	**22**	4623	**25**	400113	**26**	44368	**23**	102560	**29**	76949	**50**	247626	**27**
BENIGN:																		
BN1	2622	**32**	26	**5**	449	**13**	3065	**31**	4470	**31**	3163	**30**	1041	**30**	998	**31**	279	**12**
BN2	1107	**19**	171	**15**	44	**12**	392	**18**	1583	**18**	67	**14**	148	**18**	527	**22**	215	**13**
BN3	2942	**26**	830	**31**	6828	**32**	7409	**32**	6309	**28**	255788	**34**	1694	**30**	2551	**31**	3992	**33**
BN4	2146	**20**	1676	**25**	2889	**18**	2640	**22**	2503	**21**	1318	**19**	1921	**21**	1553	**20**	1390	**20**
BN5	106	**8**	256	**20**	725	**9**	440	**14**	1790	**27**	708	**16**	496	**20**	392	**19**	641	**15**
BN6	2524	**24**	1380	**24**	1468	**24**	2533	**23**	5378	**24**	1380	**24**	1713	**22**	808	**22**	1543	**22**
BN7	2486	**37**	2060	**37**	874	**25**	3706	**37**	4907	**24**	3762	**37**	1890	**37**	1850	**37**	1490	**36**
BN8	169	**17**	547	**23**	168	**15**	141	**16**	412	**18**	236	**17**	143	**17**	170	**18**	160	**17**

1 *Total* = total sum of different system calls; *n* = number of unique system calls

4.2 Determining the comparison sets

As there were 16 distinct applications involved (8 malicious and 8 benign), there were obviously also 16 distinct behaviors expected on a single platform, since every application differs from another by its nature. The purpose of this study was to examine and report dissimilarities of system calls between different Android platforms, which implied it was necessary to compare the data extracted from each application across the platforms.

To establish comprehensible scope for that purpose, all 3 real phones were employed as the basis for 4 separate platform comparison subgroups. Such division provided distinctive sets of data for comparison. Subgroup A concentrates exclusively on differences between the real devices, while subgroups B, C and D examine the differences between each real device and its corresponding emulation platforms.

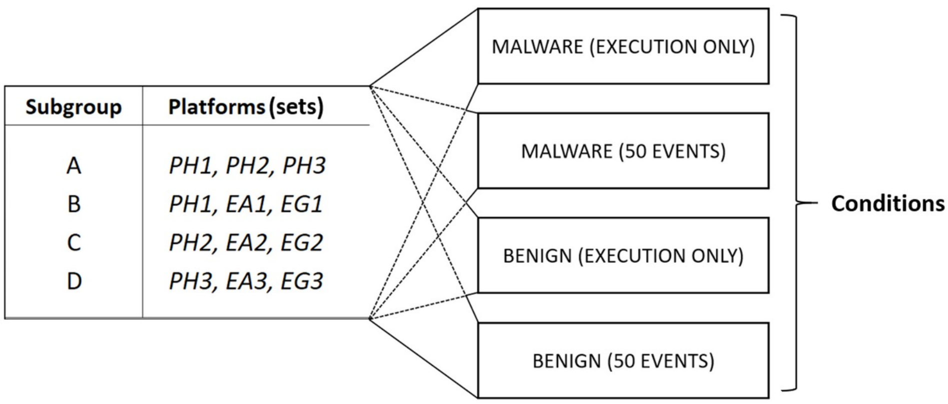


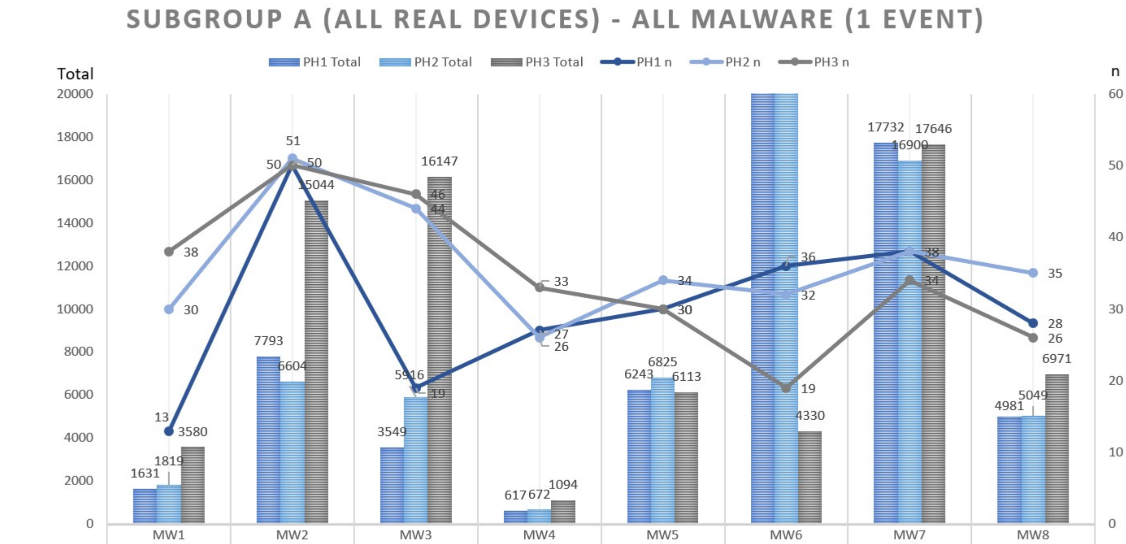
Figure 6. Comparison sets and conditions

Since the extraction of the system call logs was carried out on each platform with two limited sets of applications (malicious and benign) in two modes of execution (the first approach being solely a plain application execution and the second approach having 50 pseudo-random event injections) implied that there were eventually 4 different perspectives for examining potential contrasts within each subgroup. The four subgroups and the concept of conditions based on application types and modes of execution are presented in Figure 6. The next section of this chapter outlines the implementation of data analysis through an example of one of those perspectives, while section 4.4 describes the results gained from all given combinations described above.

4.3 Data structuring and comparison

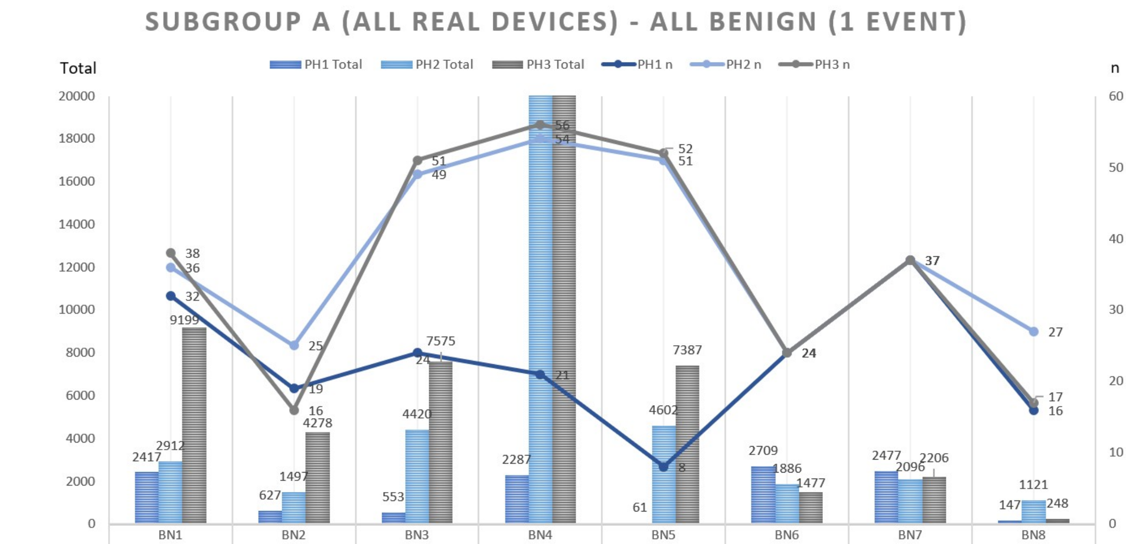
This section describes the data processing through visualization, set comparison and scoring. To adequately evaluate the principal differences or similarities within the data gathered from *strace* log summaries (presented above in Table 10 and Table 11) in regards to the selected comparison sets and conditions (shown in Figure 6), an experimental approach was implemented. Products regarding the subgroup A (real device comparison) under two of four conditions (both malware samples and benign samples with application execution only) are introduced in this section as descriptive examples. All resulting content obtained from this approach is fully presented in Appendix 2.

To achieve a more clarified overview of the log summaries' output, the data was visualized into multiple graphs, each representing 1 subgroup under 1 condition. As there were occasional anomalies present within the data (which will be discussed below in chapter 5.1), the figures tended to vary substantially at times. The total sum of system calls from each application was limited to optimal amount of 20000 in primary vertical axis of the graphical representation (the unfitting figures have been presented in charts' data tables). The maximum amount of unique system calls from logs never exceeded 60, which is the uppermost limit in secondary vertical axis. Figure 7 below visualizes the results extracted from real devices while running malware without additional interactions (execution only), while Figure 8 shows the same for benign applications.



	MW1	MW2	MW3	MW4	MW5	MW6	MW7	MW8
PH1 Total	1631	7793	3549	617	6243	594055	17732	4981
PH2 Total	1819	6604	5916	672	6825	490404	16900	5049
PH3 Total	3580	15044	16147	1094	6113	4330	17646	6971
PH1 n	13	50	19	27	30	36	38	28
PH2 n	30	51	44	26	34	32	38	35
PH3 n	38	50	46	33	30	19	34	26

Figure 7. Malware system call summaries – subgroup A (execution only)



	BN1	BN2	BN3	BN4	BN5	BN6	BN7	BN8
PH1 Total	2417	627	553	2287	61	2709	2477	147
PH2 Total	2912	1497	4420	206184	4602	1886	2096	1121
PH3 Total	9199	4278	7575	224182	7387	1477	2206	248
PH1 n	32	19	24	21	8	24	37	16
PH2 n	36	25	49	54	51	24	37	27
PH3 n	38	16	51	56	52	24	37	17

Figure 8. Benign apps' system call summaries – subgroup A (execution only)

Each of the 4 subgroups incorporated 3 platforms, ready to be compared to each other under 4 distinct conditions in relation to 2 selected attributes – total sum of all recorded system calls and the amount of unique systems calls. In order to convert the outcomes of the attributes' comparison into a unified evaluation method to score similarities, the differences of those attributes were treated as follows:

- For *total sums*, simply a *ratio* of increase between two sets of sums was calculated (i.e. when comparing the sums 1500 and 1000, the ratio would be 1,5 showing a 50% increase);

- For amounts of *unique system calls*, which might hold substantially different unique values among them, each set of contents were juxtaposed to find the intersecting unique system calls and calculate their *Jaccard coefficient* (indicated as a value ranging between 0 and 1). The Jaccard coefficient (or Jaccard index) is a measure of similarity between sample sets defined as the size of the intersection divided by the size of the union:

$$J(S_1, S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = \frac{|S_1 \cap S_2|}{|S_1| + |S_2| - |S_1 \cap S_2|}.$$

Applying those measures in the comparison process resulted in a two-part comparison indexes, with both portions representing a percentage of difference. In order to consider the comparisons of system call summaries originating from the execution of the same application on two different platforms as similar, a similarity threshold of 0,75 was applied for both attributes. This meant that the comparison results were regarded as similar, if both the total sum ratios did not exceed 1,25 and the unique system calls' coefficient did not fall below 0,75. The example results for the similarity thresholds are shown in Table 12, applied on the same data as displayed in Figure 7 and Figure 8 above.

Table 12. Similarity comparison – subgroup A (execution only)

	PH1 v PH2			PH1 v PH3			PH2 v PH3		
APK	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.
MW1	**1.12**	0.43	13(13;30)	2.19	0.34	13(13;38)	1.97	**0.79**	30(30;38)
MW2	**1.18**	**0.94**	49(50;51)	1.93	**0.92**	48(50;50)	2.28	**0.98**	50(51;50)
MW3	1.67	0.43	19(19;44)	4.55	0.41	19(19;46)	2.73	**0.96**	44(44;46)
MW4	**1.09**	**0.96**	26(27;26)	1.77	**0.76**	26(27;33)	1.63	0.74	25(26;33)
MW5	**1.09**	**0.83**	29(30;34)	**1.02**	**0.94**	29(30;30)	**1.12**	**0.88**	30(34;30)
MW6	**1.21**	**0.84**	31(36;32)	137.20	0.53	19(36;19)	113.26	0.59	19(32;19)
MW7	**1.05**	**1.00**	38(38;38)	**1.00**	0.71	30(38;34)	**1.04**	0.71	30(38;34)
MW8	**1.01**	0.70	26(28;35)	1.40	**0.86**	25(28;26)	1.38	0.74	26(35;26)
	PH1 v PH2			PH1 v PH3			PH2 v PH3		
APK	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.
BN1	**1.20**	**0.89**	32(32;36)	3.81	**0.79**	31(32;38)	3.16	**0.85**	34(36;38)
BN2	2.39	**0.76**	19(19;25)	6.82	**0.75**	15(19;16)	2.86	0.64	16(25;16)
BN3	7.99	0.49	24(24;49)	13.70	0.47	24(24;51)	1.71	**0.92**	48(49;51)
BN4	90.15	0.39	21(21;54)	98.02	0.38	21(21;56)	**1.09**	**0.96**	54(54;56)
BN5	75.44	0.16	8(8;51)	121.10	0.15	8(8;52)	1.61	**0.98**	51(51;52)
BN6	1.44	**1.00**	24(24;24)	1.83	**0.85**	22(24;24)	1.28	**0.85**	22(24;24)
BN7	**1.18**	**1.00**	37(37;37)	**1.12**	**0.95**	36(37;37)	**1.05**	**0.95**	36(37;37)
BN8	7.63	0.59	16(16;27)	1.69	0.74	14(16;17)	4.52	0.63	17(27;17)

Note: **Bold** – threshold reached; – overall similarity at least 0.75 (good); – overall similarity at least 0.90 (great)

The result tables also include the number of intersecting unique system calls on both compared platforms (with the total number of unique system calls from both platforms shown in parentheses, respectively). Additionally, the comparisons where the sets displayed remarkable similarity (maximum ratio of total sums between 1,0 and 1,10 plus Jaccard index greater than 0,90), have been distinguished. All resulting content obtained from this implementation is also fully presented in Appendix 2.

4.4 Outlining the comparison results

The general overview of the results covering the full spectrum of comparison sets and conditions has been displayed in Figure 9.

Real devices compared to each other. From the overall viewpoint of the real device platforms comparison, it becomes evident that although the similarities within this subgroup were the most prevalent, they were present only under certain conditions. Firmly consistent similar results were present between PH1 and PH2 with malware samples, both with execution only and event injection categories. However, those similarities have largely diminished if the aforementioned devices are compared with

PH3, with only a few malware samples continuing to display firm consistency (such as MW4, MW5 and MW7). In addition, it was also evident that the malware samples displayed slightly stronger similarities across platforms, when interaction with the application was mimicked through 50 event injections.

The benign samples did not display such abundance of similar activity. Only 1 application out of 8 reached the threshold on every comparison with plain execution only. There were no significant general similarities existent between platforms as it was with malware. A minor exception was present in comparison between PH2 and PH3 only with plain execution condition, where BN4 and BN7 fit into the 0,90 (great similarity) threshold. The same condition had also a noticeable ‘semi-similarity’ present in the category of unique system calls.

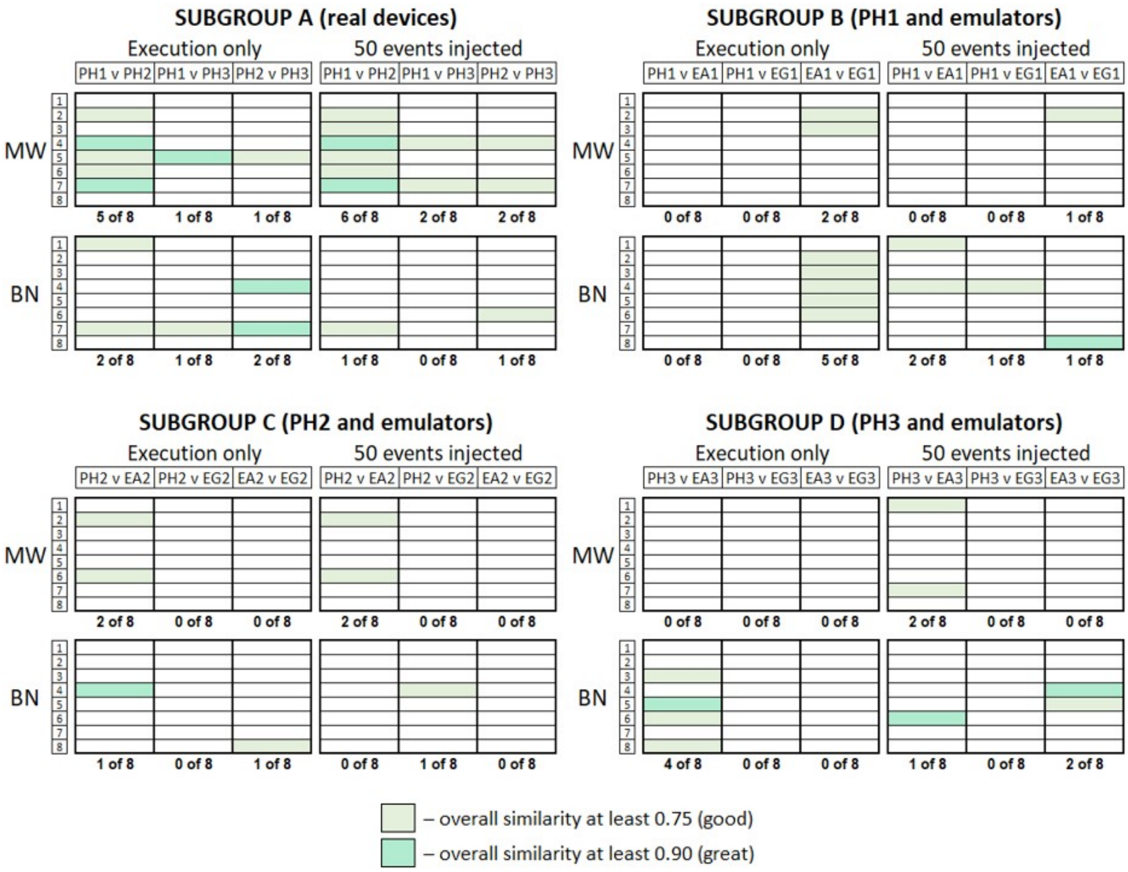


Figure 9. General overview of comparison results

Real devices compared to their emulations. The measurement results between real devices and emulators showed mostly chaotic results. Although there were some solitary matches, no significant similarity patterns were present in overwhelming majority of comparison sets and conditions. The only remarkable exception was present in subgroup D where PH3 and its SDK emulation (EA3) reached both similarity thresholds with 4 benign samples in execution only condition. Even so, it must be emphasized that between those two comparison platforms, all 8 benign applications displayed remarkable similarities in the unique system call category, reaching well above the minimal similarity thresholds. Those levels of similarities were however mostly fallen off under condition with 50 injected events.

This chapter described the raw results, formulation of the comparison subgroups and the data structuring. The latter part described the comparative analysis and outlined the general overview of the comparison results. Chapter 5 will further discuss the more relevant issues and finally concludes the main discoveries.

5 Discussion

This chapter discusses the several anomalous results and makes an effort to explain the causes and implications of the general findings. The final parts of this chapter focus on possible threats to validity, points out the possible directions for future research and summarizes the main findings of this thesis.

5.1 Outlier cases

Before addressing and contemplating the potential causes of the similarities and differences in system calls' summaries discovered in previous chapter, it is necessary to examine the certain anomalies that were present among those findings. As shown in the previous chapter, the similarities in system call summaries between different platforms were mostly a rarity, and more present only under certain conditions. The majority of comparisons indicated rather significant differences in both total sum of system calls and amounts of unique system calls. As it can be seen above in Table 10 and Table 11, the total amounts of system calls originating from the single source application varied significantly on different testing platforms. Certain levels of variances, differing several or occasionally even dozens of times in comparison, were occurring in the cases of every application sample and therefore can be considered normality. However, there were applications, which showed inconsistencies of total system call sums in levels that differed even hundreds of times on different testing platforms (as shown in Table 13 and Table 14). In the context of present study, such levels of differences can be considered as outliers but were still purposely added to comparison, as they represent the real behavior of the real samples on the real platforms. This section presents those anomalies and contemplates over the possible causes of such behavior.

Table 13. Outliers among the results (execution only)

1 EVENT	REAL ANDROID PHONES						ANDROID SDK EMULATOR						GENYMOTION EMULATOR					
	PH1		PH2		PH3		EA1		EA2		EA3		EG1		EG2		EG3	
Sample	*Total*	*n*	*Total*	*n*	*Total*	*n*	*Total*	*n*	*Total*	*n*	*Total*	*n*	*Total*	*n*	*Total*	*n*	*Total*	*n*
MALWARE:																		
MW1	1631	**13**	1819	**30**	3580	**38**	4434	**37**	8881	**36**	7674	**36**	3018	**36**	2385	**34**	2619	**34**
MW2	7793	**50**	6604	**51**	15044	**50**	3463	**43**	5427	**46**	3485	**45**	2931	**47**	2235	**47**	2206	**46**
MW3	3549	**19**	5916	**44**	16147	**46**	9208	**43**	1204022	**47**	12155	**43**	10693	**49**	10254	**47**	10361	**47**
MW4	617	**27**	672	**26**	1094	**33**	2020	**28**	3280	**29**	1876	**25**	866	**29**	920	**28**	971	**28**
MW5	6243	**30**	6825	**34**	6113	**30**	4539	**48**	846	**20**	313	**12**	12937	**53**	8646	**33**	8088	**32**
MW6	594055	**36**	490404	**32**	4330	**19**	265683	**38**	581141	**30**	256816	**30**	186062	**45**	197392	**29**	171947	**28**
MW7	17732	**38**	16900	**38**	17646	**34**	19297	**30**	34388	**36**	14457	**29**	14289	**12**	15845	**36**	13668	**33**
MW8	4981	**28**	5049	**35**	6971	**26**	9459	**50**	49116	**26**	10238	**24**	7217	**52**	87740	**50**	249006	**26**
BENIGN:																		
BN1	2417	**32**	2912	**36**	9199	**38**	3569	**30**	7553	**32**	4664	**32**	1710	**36**	1710	**34**	1420	**34**
BN2	627	**19**	1497	**25**	4278	**16**	4585	**49**	515	**17**	263	**16**	3742	**56**	3031	**56**	401	**18**
BN3	553	**24**	4420	**49**	7575	**51**	6598	**48**	10889	**48**	6667	**45**	6357	**50**	4989	**48**	4892	**48**
BN4	2287	**21**	206184	**54**	224182	**56**	23004	**52**	189200	**52**	9478	**49**	20044	**51**	6523	**50**	7424	**50**
BN5	61	**8**	4602	**51**	7387	**52**	7650	**50**	13342	**51**	7150	**49**	6263	**52**	4676	**51**	4827	**51**
BN6	2709	**24**	1886	**24**	1477	**24**	3325	**36**	5247	**24**	1740	**23**	3947	**40**	1096	**22**	1326	**22**
BN7	2477	**37**	2096	**37**	2206	**37**	3815	**37**	4924	**24**	3642	**37**	2173	**40**	1745	**36**	1624	**36**
BN8	147	**16**	1121	**27**	248	**17**	7382	**53**	402	**18**	237	**17**	3908	**55**	373	**21**	796	**24**

Note: *Total* = total sum of different system calls; *n* = number of unique system calls

Table 14. Outliers among the results (50 events injected)

50 EVENTS	REAL ANDROID PHONES						ANDROID SDK EMULATOR						GENYMOTION EMULATOR					
	PH1		PH2		PH3		EA1		EA2		EA3		EG1		EG2		EG3	
Sample	*Total*	*n*	*Total*	*n*	*Total*	*n*	*Total*	*n*	*Total*	*n*	*Total*	*n*	*Total*	*n*	*Total*	*n*	*Total*	*n*
MALWARE:																		
MW1	4481	**15**	1910	**30**	5845	**31**	6766	**31**	10716	**11**	6626	**35**	5148	**31**	6199	**30**	2966	**11**
MW2	8010	**50**	7002	**51**	6164	**21**	3635	**46**	6783	**47**	4145	**46**	3144	**47**	2523	**47**	2508	**47**
MW3	2922	**19**	2334	**22**	5155	**29**	154294	**27**	11299	**26**	8283	**26**	7065	**26**	5666	**26**	3047	**33**
MW4	576	**26**	603	**26**	615	**26**	1834	**26**	2497	**26**	1806	**26**	361	**24**	642	**25**	670	**25**
MW5	6119	**25**	6188	**30**	5386	**22**	342	**10**	357	**12**	458	**16**	10161	**30**	8859	**34**	7225	**24**
MW6	598843	**35**	502948	**36**	1437	**12**	259148	**29**	584115	**30**	128754	**14**	192390	**29**	210356	**29**	170547	**28**
MW7	16988	**38**	16157	**36**	13642	**36**	16232	**37**	28584	**34**	14956	**28**	14041	**11**	13862	**28**	13305	**12**
MW8	4543	**29**	4295	**26**	72619	**22**	4623	**25**	400113	**26**	44368	**23**	102560	**29**	76949	**50**	247626	**27**
BENIGN:																		
BN1	2622	**32**	26	**5**	449	**13**	3065	**31**	4470	**31**	3163	**30**	1041	**30**	998	**31**	279	**12**
BN2	1107	**19**	171	**15**	44	**12**	392	**18**	1583	**18**	67	**14**	148	**18**	527	**22**	215	**13**
BN3	2942	**26**	830	**31**	6828	**32**	7409	**32**	6309	**28**	255788	**34**	1694	**30**	2551	**31**	3992	**33**
BN4	2146	**20**	1676	**25**	2889	**18**	2640	**22**	2503	**21**	1318	**19**	1921	**21**	1553	**20**	1390	**20**
BN5	106	**8**	256	**20**	725	**9**	440	**14**	1790	**27**	708	**16**	496	**20**	392	**19**	641	**15**
BN6	2524	**24**	1380	**24**	1468	**24**	2533	**23**	5378	**24**	1380	**24**	1713	**22**	808	**22**	1543	**22**
BN7	2486	**37**	2060	**37**	874	**25**	3706	**37**	4907	**24**	3762	**37**	1890	**37**	1850	**37**	1490	**36**
BN8	169	**17**	547	**23**	168	**15**	141	**16**	412	**18**	236	**17**	143	**17**	170	**18**	160	**17**

Note: *Total* = total sum of different system calls; *n* = number of unique system calls

There was a one-time occasion, where an application (MW6) which consistently never stopped invoking system calls on during the collection time frame, drastically slowed down its regular behavior on a single platform (shown in Figure 10). This remarkable difference occurred only on PH3 under both 1 event and 50 event conditions. On other real devices and every emulation (including the emulations of PH3 itself – namely EA3 and EG3), the iterations (albeit mostly dissimilar according to established comparison standards) never stopped.

PH1		PH2		PH3	
calls	syscall	calls	syscall	calls	syscall
---------	----------------	---------	----------------	---------	----------------
308217	clock_gettime	269900	clock_gettime	801	clock_gettime
68572	recvfrom	58488	recvfrom	165	gettimeofday
67425	write	49628	write	99	write
50516	epoll_pwait	35140	epoll_pwait	99	recvfrom
34412	getuid32	29407	getuid32	81	epoll_pwait
17654	futex	15094	futex	75	ioctl
17359	ioctl	14990	ioctl	47	getuid32
17151	read	14645	gettimeofday	33	read
17139	gettimeofday	14618	read	32	futex
61	mprotect	229	mprotect	3	mprotect
...	...	...	...	...	...

Figure 10. Top 10 system calls from MW6 summaries (Subgroup A)

All other outlier cases had the opposite nature – on limited occasions, some applications that previously had shown consistent activities after usual boot sequences or injected input events, and eventually stopping with system call *epoll_pwait()* to wait for inputs, began to instrument unstoppable sequences of system calls on certain platform(s). Such anomalous occurrences were present in cases of two malware samples (MW3, MW8) and two benign application samples (BN3, BN4). Most of the outlier cases produced extreme amounts of *clock_gettime()* calls, with reasons likely related to CPU context switches or interruptions related to graphics processing, while logging the issues. If the model training would happen to be implemented on such an exaggerated behavior, the results might lead to model deviations and eventually to inaccuracies in malware detection.

The issue related to large differences in MW8 system call summaries seemed to be directly related to graphics incompatibilities on certain platforms (on PH3 and especially Genymotion emulations), as the main screen tended to flicker for certain period of time after application execution (displayed as a screenshot in Figure 11). In order to confirm this flicker as the issue behind excessive amounts of system calls, several re-runs with MW8 were made on those platforms, that confirmed the relation between the length of the flickering and the total sum of syscalls made. In comparison described in chapter 4, only the results from the first runs from each platform were used.

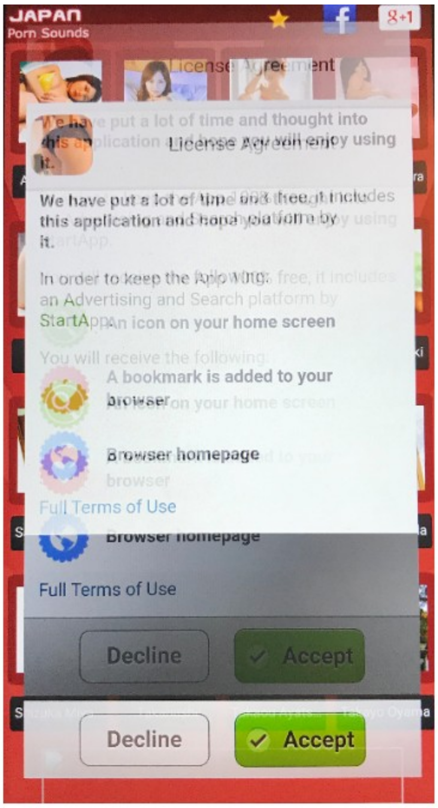


Figure 11. Example of flickering on application's main page (MW8 on PH3)

Based on the case description above, it would be feasible to generalize the software-related graphical incompatibility issues to other outlier cases, however the validation of such a theory would need additional research.

5.2 Causes and implications

Despite the existence of several outliers, the majority of results did not display such extreme levels of system call iterations. Still, the differences between the testing platforms were evident, as shown in chapter 4. This section contemplates these differences by observing the data and known specifications, while section 5.3 points out the possible limitations of the approach and gives suggestions for future research.

Real devices compared to their emulations. When comparing the summaries between the real devices and their emulations (both Android Studio and Genymotion) it is evident that the use and distribution of the system calls diverge significantly. The system call differences with real devices are more prevalent in their Genymotion emulations, where only 2,08% (2 out of 96) total comparisons fitted within the applied minimum similarity threshold (both originating from sample BN4). It must be noted that Genymotion only allowed the usage of 32-bit x86 system images, which might have seriously affected the comparison of EG3 to its reference model PH3 with 64-bit OS. Android SDK emulations fared slightly better with 14,58% (14 out of 96) comparisons fitting into 0,75 similarity threshold and showing even a certain consistency with only benign applications, when comparing PH3 with its emulation EA3.

The reasons behind these divergences are probably multifaceted with their roots in both hardware and software-related aspects. All emulations in this study were based on Intel architecture (as described in Table 6 and Table 7) while most of the real devices in today's market (including the phones used in this research) implement ARM technology. Although system calls in Linux are mostly universal for userspace processes on different Linux devices, their numbers and availability in separate architectures differ from one another [69], as do the calling conventions in x86 and ARM platforms [70]. These principal differences are also reflecting in the tracing logs, which in turn means that while the emulators are fast and convenient platforms for feature extraction and model training, the acquired malware detection abilities might not perform accurately in the real-life situation on actual Android devices. In order to prove or disapprove this statement, the data from the extracted logs gathered by this research would need to be validated by trained detection algorithms in the future research. On the other hand, the latest versions of Android SDK indicate that Google seems to be bringing back ARM-based system images to SDK emulator with the latest Android 12, which might bring new levels of architectural compatibility for emulator-based researchers.

The architectural differences in system call handling are however not the only potential reason between the overall differences between real devices and their emulations. The modern malware tends to implement anti-emulation and sandbox detection techniques in various levels of efficiency, which is a rather well-explored area in different forensic studies [71], [72]. Even a well-prepared emulation setup could never mimic a real device with uncanny accuracy, which means that the more sophisticated malware might never reveal its true nature in an emulated environment.

Similarities and differences between real devices. This thesis study employed three Android phone models (specifications are described in Table 5). While the comparison of system calls' summaries in this subgroup showed the most consistent results, the similarities were still only present under certain conditions. Most of the malware samples displayed fundamentally similar results on PH1 and PH2 with both modes of execution (similarities with the condition of 50 injected events were the closest). However, such similarities in malware system calls were much less present when PH3 was added to comparison. Additionally, the benign applications (except for BN6 and BN7) did not manifest fully consistent cross-platform similarities, although the unique calls category between PH2 and PH3 displayed good parity when executed without event injection.

The platforms with the most number of similarities – PH1 and PH2 – were running different Android versions (9 and 10), but were sharing the same kernel architecture (armv8l – a 32-bit version of ARMv8), which was their main difference with PH3 having Aarch64 architecture and slightly newer version of kernel. In short, although sharing the same ARM technology, PH1 and PH2 were both employing 32-bit operating systems on a 64-bit chipset, while PH3 had a privilege of running a 64-bit OS, due to having a higher amount of RAM. The noticeable differences in system call summaries indicate that due to its improved instruction sets, PH3 implements different calling conventions for most of the applications than less capable PH1 and PH2. The future research to compare the possible system call patterns and sequence alignments within extracted logs originating from Aarch32 and Aarch64 platforms might find some similarities, where implementing the direct comparison of summaries was not enough. On the other hand, the near future of Android seems to be 64-bit only, as ARM has already declared to terminate the support for 32-bit operations from 2023 onwards [73].

5.3 Limitations and future research

There were a number of limitations present in this study, that may impose negative effects to the validity of the outcome of the thesis. The following subsections describe the possible threats to validity and make an attempt to offer suggestions for future researchers to further expand and improve the topic in order to better understand the causes and effects of malware system calls on different mobile platforms.

5.3.1 Threats to validity

Missing initial boot sequences. Hooking *strace* to a newly started application's process through ADB shell in the manner implemented in this study involves a delay, during which a varying amount of initial system call sequences will be not traced. The length of this delay seemed to vary from several dozens of milliseconds up to a 1000 or even more milliseconds (those lengths were not measured or taken into account in this thesis). During such timespan, hundreds of initial system calls would remain unrecorded, and the variations of *strace* connection delay might therefore distort the final result amounts gathered from different platforms.

In addition to this limitation, there was also an issue regarding the legacy applications in Android 10 described at the end of chapter 3.4.4. A security permissions screen was needed to be bypassed, before Android 10 would execute the legacy application as a process on the first time on that particular device. In order to keep the policy of extracting the system calls only from the initial execution of the application, the collection script was temporarily modified for legacy applications on Android 10 by adding 1 second sleep delay between *monkey* and *strace* commands in order to manually press the 'Continue' button on screen, which would execute the app for the first time. Such diversion added at least several hundred milliseconds to the already existing delay. The legacy applications affected on Android 10 platforms by this feature were MW2, MW5, MW6, MW7, MW8, BN2, BN6, BN7 and BN8.

Unweighted unique system call values. In the similarity assessment phase, each registered unique system calls on each platform were given the same value. Since the amount of their appearances in the sequences differed, an introduction of the weighing or ranking system of unique system calls would likely make the similarity assessment more accurate.

Possible emulation detection. As noted in the previous subsection, some of the malware might be able to detect the emulation environment and hide their main activities. During this study, the applications were not reverse-engineered in order to examine their potential abilities of this kind. This leaves at least a theoretical possibility, that some of the malware could have expressed different behavior in real device and emulation platforms.

5.3.2 Suggestions for future research

Although the limitations described above might impose certain distorting effects on the results, it is unlikely that those effects would significantly threaten the final outcomes of similarity comparison. For the future researches incorporating the methods used in this thesis, it would nevertheless be advisable to implement measures to negate the effects of those limitations in order to achieve better accuracy.

In order to fully validate the outcomes of this study and estimate the potential effects of platform-related system call differences on malware detection models, the *strace* logs extracted during the collection process in this study should be tested with an existing detection algorithms trained on either real device and emulation environments. It must be noted that some of the malware samples used in this thesis were quite recent and unique – a trait which might render them immune to the detection models trained on older datasets (the families and overall characteristics of used samples are described in chapter 3.2.1).

Although the effects of the acquired results remain to be validated by detection models trained on emulated environments, it would likely be advisable to prefer real device platforms to emulated ones for decisive research results, in order to achieve outcomes accordingly to the malware behaviors on real-life production environments. However, such comparative study should be repeated with a *true* ARM emulation platform, incorporating an ARM-based system image and a host with ARM-based CPU. If not already possible, the establishment of such setup combinations should be available in a very near future.

Additionally, in order to acquire more detailed information about the causes and possible relations of system call differences on various real devices, a subsequent research on larger collection of real devices (with 64-bit operation systems) should be implemented. Such study should consist of both sequence alignment and summary comparison and could also involve custom-made malware. Furthermore, in order to examine possible statistical deviations, a larger amount of applications should be tested with several repetitions for each execution.

5.4 Conclusion

This thesis compared certain aspects concerning the behavior of malicious and benign applications, derived from summaries of applications' system call recordings, on a selection of different platforms. The scope of the thesis was limited to a summational approach to system call implementations, which provided good basis for comparative analysis, but restricted the reasoning over technical causes to a level of observational interpretation.

Answers to the research questions:

RQ1: How significant are the potential differences of system calls invoked by a single application between different types of platforms? – The results based on system call summaries indicated significant differences between real devices and emulators in terms of system call invocations. On both emulation environments, 96 separate system call extractions were conducted (total 192 from emulators). Each sample's summaries were compared for similarity with corresponding sample summaries from respective real devices. As a result, only 2,08% of the system call logs extracted from Genymotion emulator and 14,58% from Android SDK emulator were found to be within the similarity limits with their counterparts collected from real devices.

The differences between the real devices were less extensive, but in general still unexpectedly significant. The only consistency in similarity comparison was between two devices sharing the same version of 32-bit kernel architecture, and even for those two, the consistency was strong only with malware samples and yet disappointingly weak with benign applications. Their separate comparisons with the device possessing different kernel architecture was largely inconsistent but also controversial, with few applications exhibiting unusual consistencies.

RQ2: What are the possible causes behind system call differences on different platforms? – The reasoning based on observational analysis would conclude that the main reason behind the differences comes from contrasting instruction sets and kernel architectures of compared platforms. Despite that being the potential main factor, the issue of system call differences is most likely tied not only to a single aspect, but is more a cluster of different causes, including hardware performance, memory availability, connectivity issues, etc.

RQ3: How could the existing differences affect malware analysis and what are the possibilities to overcome this problem? – Although the comparison results must yet be validated on an existing detection models, it is likely the emulator platforms are not the appropriate means for feature extraction and model training if the main purpose is the effective countering of malware in real-life environments. The results gathered in this study also indicated less significant but notable differences between real devices employing different extensions of ARM architecture. To fully evaluate and overcome the potential effects of those differences on malware detection models, additional research is needed.

6 Summary

The objective of this thesis was to examine and analyze existing differences between system calls of malicious and benign applications on different android platforms, including both emulators and real devices in a cross-comparison.

In order to achieve that purpose, a testing setup comprising of different platforms was employed and documented. For raw data acquisition, strict criteria describing the settings, conditions and restrictions was implemented. Limited sets of both malicious and benign applications were installed and executed on testing environment according to the predefined criteria for extracting system calls data to be evaluated in a comparative analysis.

From the gathered raw data files, summaries representing both total amounts of system calls and individual unique system calls were selected and categorized into a comparison sets and subgroups according to respective testing platform and modus of execution. The categorized sets were compared within subgroups and evaluated according to the experimental approach described in thesis.

The results indicated significant differences between real devices and emulators in terms of system call invocations. The differences between the real devices were less extensive, but in general still unexpectedly significant. Although the results displayed in this thesis would need additional validation by trained detection algorithms, the current findings are suggesting that the studies employing system calls must consider with platform-specific differences in terms of general reliability.

References

[1] “Smartphone users 2021,” *Statista*. https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/ (accessed Jun. 28, 2021).

[2] “Mobile Operating System Market Share Worldwide,” *StatCounter Global Stats*. https://gs.statcounter.com/os-market-share/mobile/worldwide/ (accessed Jun. 28, 2021).

[3] “First SMS Trojan for Android,” *Securelist*, Aug. 10, 2010. https://securelist.com/first-sms-trojan-for-android/29731/ (accessed May 22, 2021).

[4] “Mobile Malware Evolution, Part 5,” *Securelist*, Feb. 28, 2012. https://securelist.com/mobile-malware-evolution-part-5/36485/ (accessed May 25, 2021).

[5] “AV-TEST: Security Report 2016/2017.” Accessed: Jun. 29, 2021. [Online]. Available: https://www.av-test.org/fileadmin/pdf/security_report/AV-TEST_Security_Report_2016-2017.pdf

[6] “ENISA Threat Landscape 2020: Cyber Attacks Becoming More Sophisticated, Targeted, Widespread and Undetected.” https://www.enisa.europa.eu/news/enisa-news/enisa-threat-landscape-2020 (accessed Jul. 12, 2021).

[7] “Android ABIs | Android NDK,” *Android Developers*. https://developer.android.com/ndk/guides/abis (accessed Apr. 24, 2021).

[8] A. Guerra-Manzanares, H. Bahsi, and S. Nomm, “Differences in Android Behavior Between Real Device and Emulator: A Malware Detection Perspective,” in *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, Granada, Spain, Oct. 2019, pp. 399–404. doi: 10.1109/IOTSMS48152.2019.8939268.

[9] R. Tamma, O. Skulkin, H. Mahalik, and S. Bommisetty, *Practical Mobile Forensics - Fourth Edition*. Packt Publishing, 2020. Accessed: Jun. 29, 2021. [Online]. Available: https://go.oreilly.com/university-college-london/library/view/-/9781838647520/

[10] “Mobile & Tablet Android Version Market Share Worldwide,” *StatCounter Global Stats*. https://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide (accessed Jun. 12, 2021).

[11] “Application Fundamentals,” *Android Developers*. https://developer.android.com/guide/components/fundamentals (accessed Apr. 17, 2021).

[12] “Add C and C++ code to your project,” *Android Developers*. https://developer.android.com/studio/projects/add-native-code (accessed Apr. 17, 2021).

[13] “Get started with the NDK | Android NDK,” *Android Developers*. https://developer.android.com/ndk/guides (accessed Apr. 17, 2021).

[14] Y. Liu, C. Tantithamthavorn, L. Li, and Y. Liu, “Deep Learning for Android Malware Defenses: a Systematic Literature Review,” *arXiv:2103.05292 [cs]*, Mar. 2021, Accessed: Jun. 15, 2021. [Online]. Available: http://arxiv.org/abs/2103.05292

[15] K. Tam, A. Feizollah, N. B. Anuar, R. Salleh, and L. Cavallaro, “The Evolution of Android Malware and Android Analysis Techniques,” *ACM Comput. Surv.*, vol. 49, no. 4, pp. 1–41, Feb. 2017, doi: 10.1145/3017427.

[16] R. Surendran, T. Thomas, and S. Emmanuel, “On Existence of Common *Malicious System Call Codes* in Android Malware Families,” *IEEE Trans. Rel.*, vol. 70, no. 1, pp. 248–260, Mar. 2021, doi: 10.1109/TR.2020.2982537.

[17] “Platform Architecture,” *Android Developers*. https://developer.android.com/guide/platform (accessed May 18, 2021).

[18] J. J. Drake, *Android hacker’s handbook*. Indianapolis, IN: John Wiley & Sons, 2014.

[19] K. Dunham, S. Hartman, J. A. Morales, M. Quintans, and T. Strazzere, *Android malware and analysis*. Boca Raton: CRC Press, Taylor & Francis Group, 2014.

[20] “Kernel Definition,” *The Linux Information Project*. http://www.linfo.org/kernel.html (accessed Jun. 12, 2021).

[21] “Kernel,” *Android Open Source Project*. https://source.android.com/devices/architecture/kernel (accessed Jun. 12, 2021).

[22] S. Malik and K. Khatter, “System Call Analysis of Android Malware Families,” *Indian Journal of Science and Technology*, vol. 9, no. 21, Jun. 2016, doi: 10.17485/ijst/2016/v9i21/90273.

[23] R. Love, *Linux system programming*, Second edition. Beijing: O’Reilly Media, 2013.

[24] “Arm vs x86: Instruction sets, architecture, and more differences explained,” *Android Authority*, Jun. 05, 2021. https://www.androidauthority.com/arm-vs-x86-key-differences-explained-568718/ (accessed Jun. 07, 2021).

[25] “Emulators vs Simulators vs Real Device for Testing,” *BrowserStack*. https://www.browserstack.com/guide/testing-on-emulators-simulators-real-devices-comparison (accessed Jun. 06, 2021).

[26] “What is Scareware?,” *Forcepoint*, Apr. 01, 2019. https://www.forcepoint.com/cyber-edu/scareware (accessed Jun. 18, 2021).

[27] “SMS Trojan,” *Malwarebytes Labs*. https://blog.malwarebytes.com/threats/sms-trojan/ (accessed Jun. 15, 2021).

[28] F. Alswaina and K. Elleithy, “Android Malware Family Classification and Analysis: Current Status and Future Directions,” *Electronics*, vol. 9, no. 6, p. 942, Jun. 2020, doi: 10.3390/electronics9060942.

[29] F. Tchakounté and P. Dayang, “System Calls Analysis of Malwares on Android,” *Maejo International Journal of Science and Technology*, vol. 2, no. 9, pp. 669–674, Sep. 2013.

[30] P. K. Das, A. Joshi, and T. Finin, “App behavioral analysis using system calls,” in *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, Atlanta, GA, May 2017, pp. 487–492. doi: 10.1109/INFCOMW.2017.8116425.

[31] K. Denney, C. Kaygusuz, and J. Zuluaga, “A Survey of Malware Detection Using System Call Tracing Techniques,” 2018.

[32] Z. Wang, Q. Liu, and Y. Chi, “Review of Android Malware Detection Based on Deep Learning,” *IEEE Access*, vol. 8, pp. 181102–181126, 2020, doi: 10.1109/ACCESS.2020.3028370.

[33] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, “A sense of self for Unix processes,” in *Proceedings 1996 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, 1996, pp. 120–128. doi: 10.1109/SECPRI.1996.502675.

[34] A. P. Kosoresow and S. A. Hofmeyer, “Intrusion detection via system call traces,” *IEEE Softw.*, vol. 14, no. 5, pp. 35–42, Oct. 1997, doi: 10.1109/52.605929.

[35] S. Malik, “Anomaly based Intrusion Detection in Android Mobiles: A Review,” *International Journal of Engineering Research*, vol. 8, no. 10, pp. 698–710, Oct. 2019.

[36] J. M. Vidal, M. A. S. Monge, and L. J. G. Villalba, “A novel pattern recognition system for detecting Android malware by analyzing suspicious boot sequences,” *Knowledge-Based Systems*, vol. 150, pp. 198–217, Jun. 2018, doi: 10.1016/j.knosys.2018.03.018.

[37] H. Ruan, X. Fu, X. Liu, X. Du, and B. Luo, “Analyzing Android Application in Real-Time at Kernel Level,” in *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, Vancouver, BC, Canada, Jul. 2017, pp. 1–9. doi: 10.1109/ICCCN.2017.8038362.

[38] A. Amamra, J.-M. Robert, and C. Talhi, “Enhancing malware detection for Android systems using a system call filtering and abstraction process: Security and communication networks,” *Security Comm. Networks*, vol. 8, no. 7, pp. 1179–1192, May 2015, doi: 10.1002/sec.1073.

[39] B. Zhao, "Mapping System Level Behaviors with Android APIs via System Call Dependence Graphs," in *Computer Science & Information Technology (CS & IT)*, May 2019, pp. 139–152. doi: 10.5121/csit.2019.90612.

[40] V. P., A. Zemmari, and M. Conti, "A machine learning based approach to detect malicious android apps using discriminant system calls," *Future Generation Computer Systems*, vol. 94, pp. 333–350, May 2019, doi: 10.1016/j.future.2018.11.021.

[41] L. Xu, D. Zhang, M. A. Alvarez, J. A. Morales, X. Ma, and J. Cavazos, "Dynamic Android Malware Classification Using Graph-Based Representations," in *2016 IEEE 3rd International Conference on Cyber Security and Cloud Computing (CSCloud)*, Beijing, China, Jun. 2016, pp. 220–231. doi: 10.1109/CSCloud.2016.27.

[42] P. Vinod and P. Viswalakshmi, "Empirical Evaluation of a System Call-Based Android Malware Detector," *Arab J Sci Eng*, vol. 43, no. 12, pp. 6751–6770, Dec. 2018, doi: 10.1007/s13369-017-2828-0.

[43] M. Dimjašević, S. Atzeni, I. Ugrina, and Z. Rakamaric, "Evaluation of Android Malware Detection Based on System Calls," in *Proceedings of the 2016 ACM on International Workshop on Security And Privacy Analytics*, New Orleans Louisiana USA, Mar. 2016, pp. 1–8. doi: 10.1145/2875475.2875487.

[44] K. Deepa, G. Radhamani, P. Vinod, M. Shojafar, N. Kumar, and M. Conti, "Identification of Android malware using refined system calls," *Concurrency Computat Pract Exper*, vol. 31, no. 20, Oct. 2019, doi: 10.1002/cpe.5311.

[45] X. Xiao, X. Xiao, Y. Jiang, X. Liu, and R. Ye, "Identifying Android malware with system call co-occurrence matrices," *Trans. Emerging Tel. Tech.*, vol. 27, no. 5, pp. 675–684, May 2016, doi: 10.1002/ett.3016.

[46] A. S. M. Ahsan-Ul-Haque, Md. S. Hossain, and M. Atiquzzaman, "Sequencing System Calls for Effective Malware Detection in Android," in *2018 IEEE Global Communications Conference (GLOBECOM)*, Abu Dhabi, United Arab Emirates, Dec. 2018, pp. 1–7. doi: 10.1109/GLOCOM.2018.8647967.

[47] A. Ananya, A. Aswathy, T. R. Amal, P. G. Swathy, P. Vinod, and S. Mohammad, "SysDroid: a dynamic ML-based android malware analyzer using system call traces," *Cluster Comput*, vol. 23, no. 4, pp. 2789–2808, Dec. 2020, doi: 10.1007/s10586-019-03045-6.

[48] Sk3ptre, "Popular Android malware seen in 2020." https://github.com/sk3ptre/AndroidMalware_2020 (accessed Mar. 09, 2021).

[49] A. H. Lashkari, A. F. A. Kadir, L. Taheri, and A. A. Ghorbani, "Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification," in *2018 International Carnahan Conference on Security Technology (ICCST)*, Montreal, QC, Oct. 2018, pp. 1–7. doi: 10.1109/CCST.2018.8585560.

[50] M. Hurier *et al.*, “Euphony: Harmonious Unification of Cacophonous Anti-Virus Vendor Labels for Android Malware,” in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, Buenos Aires, Argentina, May 2017, pp. 425–435. doi: 10.1109/MSR.2017.57.

[51] “MobOk Malware Hides in Photo Editors on Google Play, Siphons Cash.” https://threatpost.com/mobok-malware-google-photo-editor/145932/ (accessed Apr. 03, 2021).

[52] “Mobile malware evolution 2020.” https://securelist.com/mobile-malware-evolution-2020/101029/ (accessed Apr. 03, 2021).

[53] R. Chauhan, “How To Hack Android Mobile With XploitSPY From Android And Computer,” *DarkHunts*, Apr. 23, 2020. http://darkhunts.com/hack-android-using-xploitspy/ (accessed Apr. 03, 2021).

[54] “Adware/Hiddad!Android,” *FortiGuard*. https://fortiguard.com/encyclopedia/virus/7079783 (accessed Apr. 03, 2021).

[55] P. Kohli, “Icon-hiding Android adware returns to the Play Market,” *Sophos News*, Oct. 08, 2019. https://news.sophos.com/en-us/2019/10/08/icon-hiding-android-adware-returns-to-the-play-market/ (accessed Apr. 03, 2021).

[56] “Android/Hiddad.AKH!tr,” *FortiGuard*. https://fortiguard.com/encyclopedia/virus/8211482 (accessed Apr. 03, 2021).

[57] G. Belding, “BlackBerry exposes threat actor group BAHAMUT: Cyberespionage, phishing and other APTs,” *Infosec Resources*, Mar. 09, 2021. https://resources.infosecinstitute.com/topic/blackberry-exposes-threat-actor-group-bahamut-cyberespionage-phishing-and-other-apts/ (accessed Apr. 03, 2021).

[58] Sonicwall News, “Android spyware Bahamut spreads disguised as Voice of Islam app,” Nov. 20, 2020. https://securitynews.sonicwall.com/xmlpost/android-spyware-bahamut-spreads-disguised-as-voice-of-islam-app/ (accessed Apr. 03, 2021).

[59] “Android/Adware.MobiDash,” *Malwarebytes Labs*. https://blog.malwarebytes.com/detections/android-adware-mobidash/ (accessed Apr. 03, 2021).

[60] “Family MobiDash,” *Kharon Project - Studying Android malware behaviors*, Jan. 2015. http://kharon.gforge.inria.fr/dataset/malware_MobiDash.html (accessed Apr. 03, 2021).

[61] “More than 100 Google Play apps found to contain advertising spyware,” *Dr.Web*, Mar. 31, 2016. https://news.drweb.com/show?i=9902&c=5&lng=en (accessed Apr. 03, 2021).

[62] “WannaCry WannaBe targeting Android smartphones,” *Avast Blog*, Jun. 07, 2017. https://blog.avast.com/wannacry-wannabe-targeting-android-smartphones (accessed Apr. 04, 2021).

[63] A. F. A. Kadir, N. Stakhanova, and A. A. Ghorbani, “Understanding Android Financial Malware Attacks: Taxonomy, Characterization, and Challenges,” *JCSM*, vol. 7, no. 3, pp. 1–52, 2018, doi: 10.13052/jcsm2245-1439.732.

[64] X. Jiang, “Security Alert: New Stealthy Android Spyware -- Plankton -- Found in Official Android Market,” Jun. 09, 2011. https://www.csc2.ncsu.edu/faculty/xjiang4/Plankton/ (accessed Apr. 04, 2021).

[65] “Android Debug Bridge (adb),” *Android Developers*. https://developer.android.com/studio/command-line/adb (accessed Apr. 07, 2021).

[66] “UI/Application Exerciser Monkey,” *Android Developers*. https://developer.android.com/studio/test/monkey (accessed Apr. 07, 2021).

[67] “Strace - Linux Syscall Tracer,” *Strace*. https://strace.io/ (accessed Apr. 07, 2021).

[68] “Privacy changes in Android 10,” *Android Developers*. https://developer.android.com/about/versions/10/privacy/changes (accessed Apr. 08, 2021).

[69] “Linux kernel system calls table,” *Marcin Juszkiewicz*. https://marcin.juszkiewicz.com.pl/download/tables/syscalls.html (accessed Jul. 28, 2021).

[70] “How System Call is works in ARM/x86 and System Call Implementation in ARM/x86.” http://vivenembedded.blogspot.com/2013/08/how-system-call-is-works-in-armx86-and.html (accessed Jul. 28, 2021).

[71] T. Vidas and N. Christin, “Evading android runtime analysis via sandbox detection,” in *Proceedings of the 9th ACM symposium on Information, computer and communications security*, Kyoto Japan, Jun. 2014, pp. 447–458. doi: 10.1145/2590296.2590325.

[72] T. Petsas, G. Voyatzis, E. Athanasopoulos, M. Polychronakis, and S. Ioannidis, “Rage against the virtual machine: hindering dynamic analysis of Android malware,” in *Proceedings of the Seventh European Workshop on System Security - EuroSec ’14*, Amsterdam, The Netherlands, 2014, pp. 1–6. doi: 10.1145/2592791.2592796.

[73] D. Whaley, Director, S. S. Solutions, and Arm, “Why Arm Is Making All Cortex-A Mobile Cores 64-bit Only,” *Arm Blueprint*, May 25, 2021. https://www.arm.com/blogs/blueprint/64-bit (accessed Jul. 28, 2021).

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

I, Martin Välbe

1 Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "Benchmarking of Android Applications' System Calls Behavior: Implications for Malware Detection", supervised by Alejandro Guerra Manzanares, MSc and co-supervised by Tarmo Oja, MSc

1.1 to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;

1.2 to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.

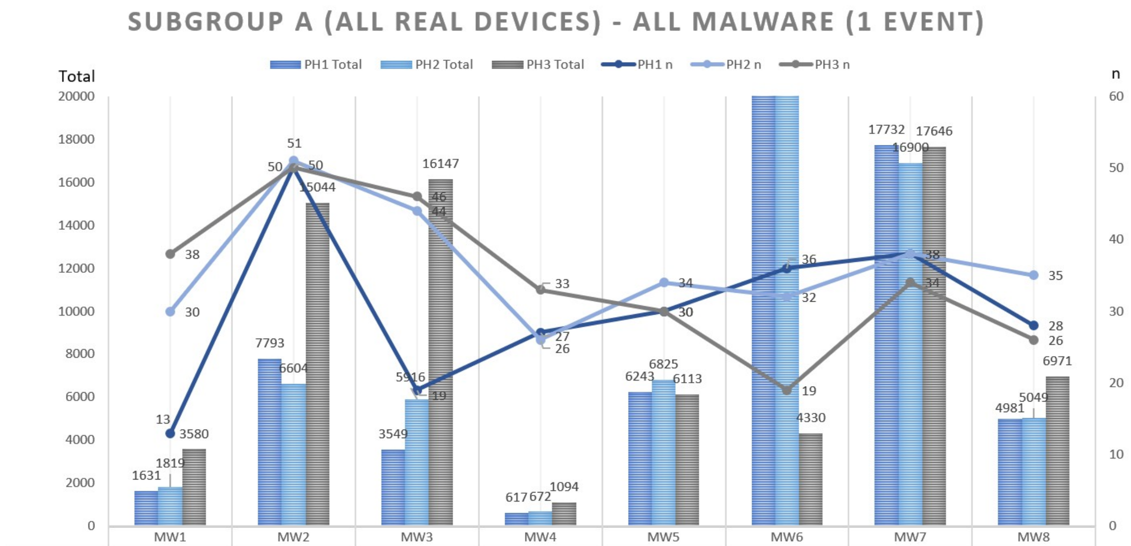
2 I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.

3 I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

30.07.2021

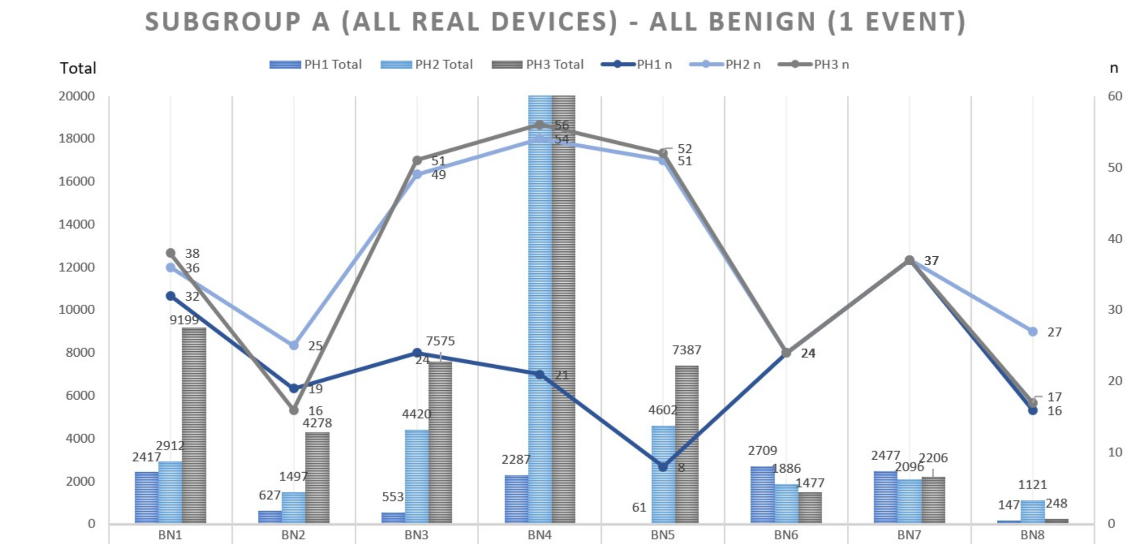
1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 – Detailed comparison results



	MW1	MW2	MW3	MW4	MW5	MW6	MW7	MW8
PH1 Total	1631	7793	3549	617	6243	594055	17732	4981
PH2 Total	1819	6604	5916	672	6825	490404	16900	5049
PH3 Total	3580	15044	16147	1094	6113	4330	17646	6971
PH1 n	13	50	19	27	30	36	38	28
PH2 n	30	51	44	26	34	32	38	35
PH3 n	38	50	46	33	30	19	34	26

Figure A2-1. Malware system call summaries – subgroup A (execution only)



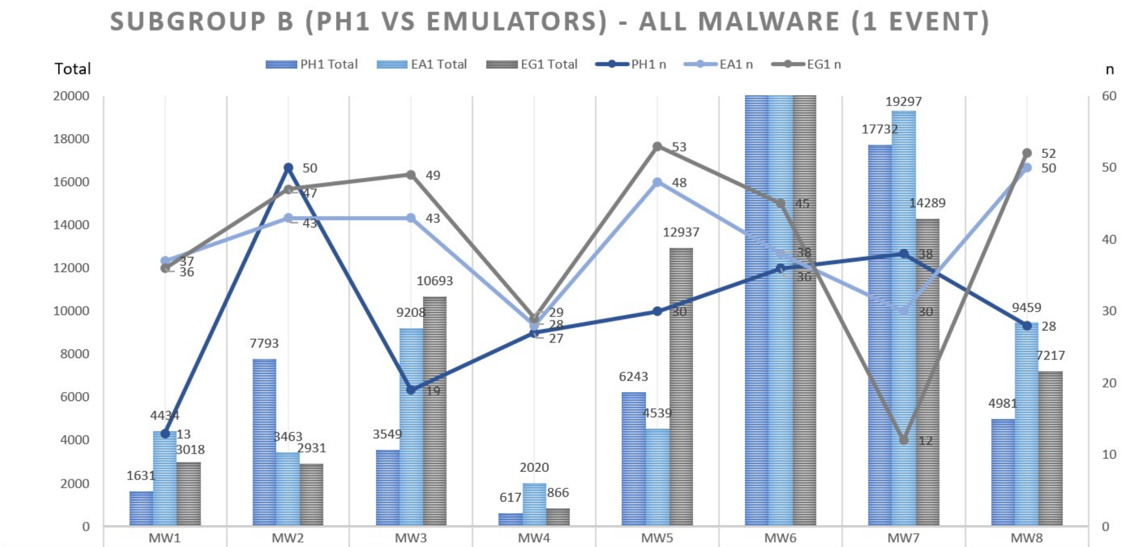
	BN1	BN2	BN3	BN4	BN5	BN6	BN7	BN8
PH1 Total	2417	627	553	2287	61	2709	2477	147
PH2 Total	2912	1497	4420	206184	4602	1886	2096	1121
PH3 Total	9199	4278	7575	224182	7387	1477	2206	248
PH1 n	32	19	24	21	8	24	37	16
PH2 n	36	25	49	54	51	24	37	27
PH3 n	38	16	51	56	52	24	37	17

Figure A2-2. Benign apps' system call summaries – subgroup A (execution only)

Table A2-1. Similarity comparison – subgroup A (execution only)

	PH1 v PH2			PH1 v PH3			PH2 v PH3		
APK	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.
MW1	**1.12**	0.43	13(13;30)	2.19	0.34	13(13;38)	1.97	**0.79**	30(30;38)
MW2	**1.18**	**0.94**	49(50;51)	1.93	**0.92**	48(50;50)	2.28	**0.98**	50(51;50)
MW3	1.67	0.43	19(19;44)	4.55	0.41	19(19;46)	2.73	**0.96**	44(44;46)
MW4	**1.09**	**0.96**	26(27;26)	1.77	**0.76**	26(27;33)	1.63	0.74	25(26;33)
MW5	**1.09**	**0.83**	29(30;34)	**1.02**	**0.94**	29(30;30)	**1.12**	**0.88**	30(34;30)
MW6	**1.21**	**0.84**	31(36;32)	137.20	0.53	19(36;19)	113.26	0.59	19(32;19)
MW7	**1.05**	**1.00**	38(38;38)	**1.00**	0.71	30(38;34)	**1.04**	0.71	30(38;34)
MW8	**1.01**	0.70	26(28;35)	1.40	**0.86**	25(28;26)	1.38	0.74	26(35;26)
	PH1 v PH2			PH1 v PH3			PH2 v PH3		
APK	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.
BN1	**1.20**	**0.89**	32(32;36)	3.81	**0.79**	31(32;38)	3.16	**0.85**	34(36;38)
BN2	2.39	**0.76**	19(19;25)	6.82	**0.75**	15(19;16)	2.86	0.64	16(25;16)
BN3	7.99	0.49	24(24;49)	13.70	0.47	24(24;51)	1.71	**0.92**	48(49;51)
BN4	90.15	0.39	21(21;54)	98.02	0.38	21(21;56)	**1.09**	**0.96**	54(54;56)
BN5	75.44	0.16	8(8;51)	121.10	0.15	8(8;52)	1.61	**0.98**	51(51;52)
BN6	1.44	**1.00**	24(24;24)	1.83	**0.85**	22(24;24)	1.28	**0.85**	22(24;24)
BN7	**1.18**	**1.00**	37(37;37)	**1.12**	**0.95**	36(37;37)	**1.05**	**0.95**	36(37;37)
BN8	7.63	0.59	16(16;27)	1.69	0.74	14(16;17)	4.52	0.63	17(27;17)

Note: **Bold** – threshold reached; [|] – overall similarity at least 0.75 (good); [|] – overall similarity at least 0.90 (great)



	MW1	MW2	MW3	MW4	MW5	MW6	MW7	MW8
PH1 Total	1631	7793	3549	617	6243	594055	17732	4981
EA1 Total	4434	3463	9208	2020	4539	265683	19297	9459
EG1 Total	3018	2931	10693	866	12937	186062	14289	7217
PH1 n	13	50	19	27	30	36	38	28
EA1 n	37	43	43	28	48	38	30	50
EG1 n	36	47	49	29	53	45	12	52

Figure A2-3. Malware system call summaries – subgroup B (execution only)

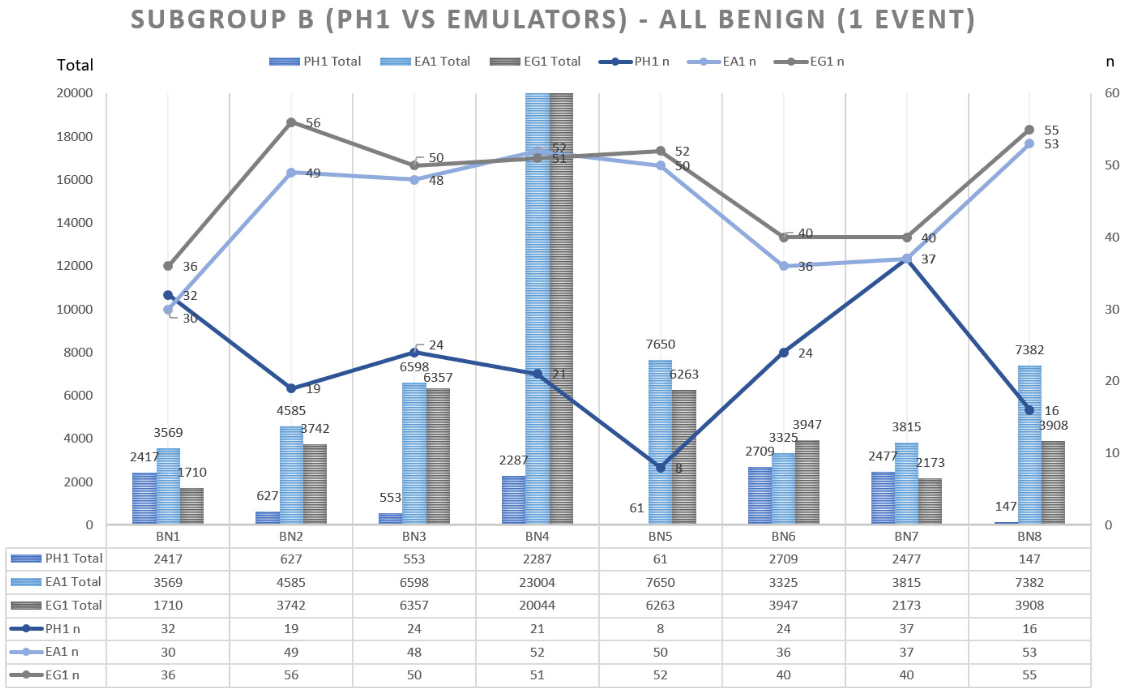
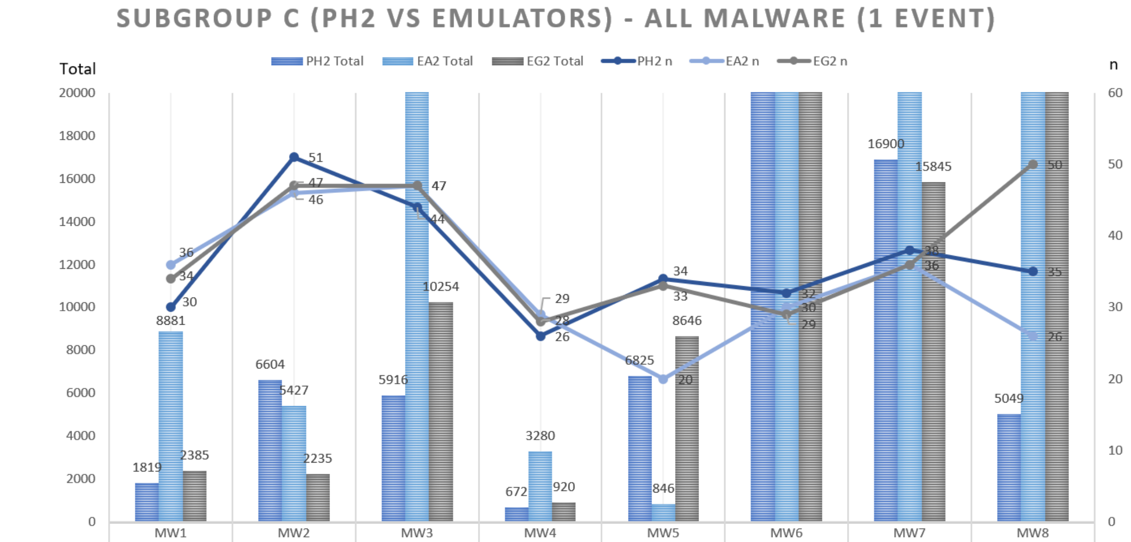


Figure A2-4. Benign apps' system call summaries – subgroup B (execution only)

Table A2-2. Similarity comparison – subgroup B (execution only)

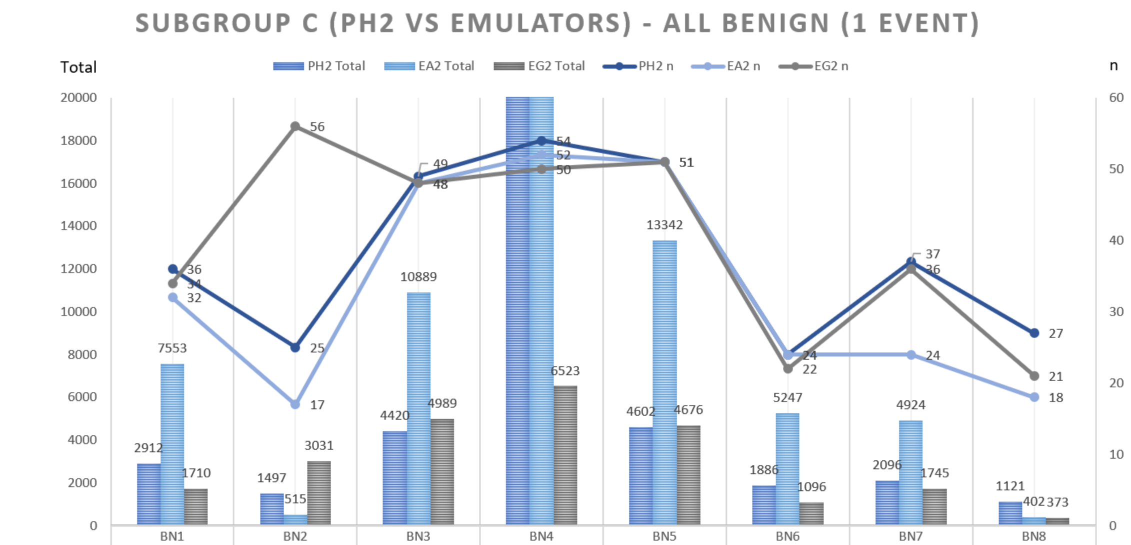
	PH1 v EA1			PH1 v EG1			EA1 v EG1		
APK	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.
MW1	2.72	0.22	9(13;37)	1.85	0.23	9(13;36)	1.47	**0.97**	36(37;36)
MW2	2.25	**0.75**	40(50;43)	2.66	**0.83**	44(50;47)	**1.18**	**0.84**	41(43;47)
MW3	2.59	0.35	16(19;43)	3.01	0.31	16(19;49)	**1.16**	**0.80**	41(43;49)
MW4	3.27	0.57	20(27;28)	1.40	0.60	21(27;29)	2.33	**0.97**	28(28;29)
MW5	1.38	0.37	21(30;48)	2.07	0.43	25(30;53)	2.85	**0.84**	46(48;53)
MW6	2.24	0.72	31(36;38)	3.19	**0.76**	35(36;45)	1.43	**0.84**	38(38;45)
MW7	**1.09**	0.62	26(38;30)	**1.24**	0.28	11(38;12)	1.35	0.40	12(30;12)
MW8	1.90	0.42	23(28;50)	1.45	0.40	23(28;52)	1.31	**0.92**	49(50;52)
	PH1 v EA1			PH1 v EG1			EA1 v EG1		
APK	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.
BN1	1.48	**0.94**	30(32;30)	1.41	**0.79**	30(32;36)	2.09	**0.83**	30(30;36)
BN2	7.31	0.33	17(19;49)	5.97	0.32	18(19;56)	**1.23**	**0.84**	48(49;56)
BN3	11.93	0.36	19(24;48)	11.50	0.35	19(24;50)	**1.04**	**0.85**	45(48;50)
BN4	10.06	0.38	20(21;52)	8.76	0.38	20(21;51)	**1.15**	**0.91**	49(52;51)
BN5	125.41	0.14	7(8;50)	102.67	0.13	7(8;52)	**1.22**	**0.85**	47(50;52)
BN6	**1.23**	0.40	17(24;36)	1.46	0.42	19(24;40)	**1.19**	**0.85**	35(36;40)
BN7	1.54	0.64	29(37;37)	**1.14**	0.57	28(37;40)	1.76	**0.88**	36(37;40)
BN8	50.22	0.23	13(16;53)	26.59	0.22	13(16;55)	1.89	**0.89**	51(53;55)

Note: **Bold** – threshold reached; – overall similarity at least 0.75 (good); – overall similarity at least 0.90 (great)



	MW1	MW2	MW3	MW4	MW5	MW6	MW7	MW8
PH2 Total	1819	6604	5916	672	6825	490404	16900	5049
EA2 Total	8881	5427	1204022	3280	846	581141	34388	49116
EG2 Total	2385	2235	10254	920	8646	197392	15845	87740
PH2 n	30	51	44	26	34	32	38	35
EA2 n	36	46	47	29	20	30	36	26
EG2 n	34	47	47	28	33	29	36	50

Figure A2-5. Malware system call summaries – subgroup C (execution only)



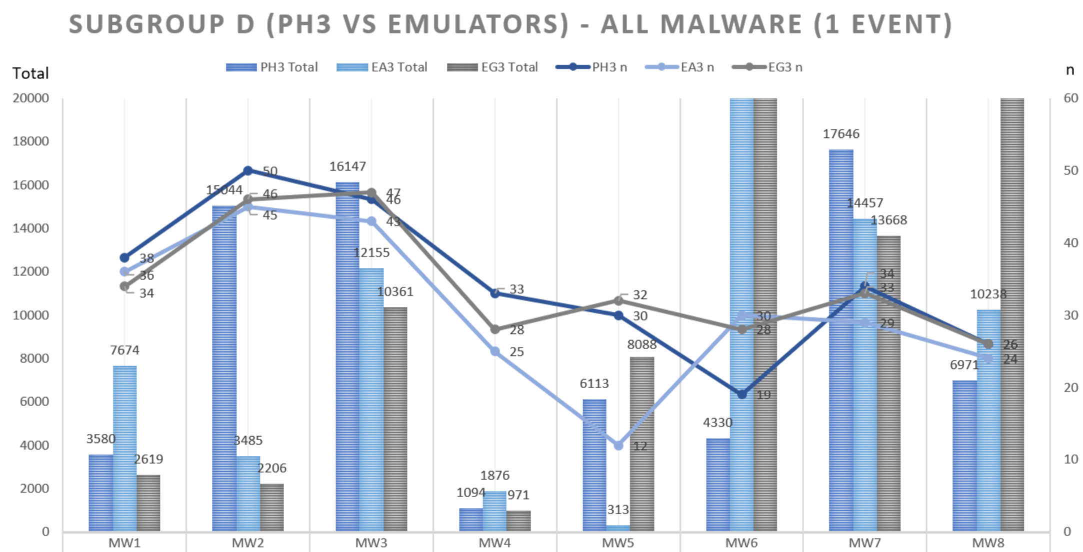
	BN1	BN2	BN3	BN4	BN5	BN6	BN7	BN8
PH2 Total	2912	1497	4420	206184	4602	1886	2096	1121
EA2 Total	7553	515	10889	189200	13342	5247	4924	402
EG2 Total	1710	3031	4989	6523	4676	1096	1745	373
PH2 n	36	25	49	54	51	24	37	27
EA2 n	32	17	48	52	51	24	24	18
EG2 n	34	56	48	50	51	22	36	21

Figure A2-6. Benign apps' system call summaries – subgroup C (execution only)

Table A2-3. Similarity comparison – subgroup C (execution only)

	PH2 v EA2			PH2 v EG2			EA2 v EG2		
APK	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.
MW1	4.88	0.61	25(30;36)	1.31	0.64	25(30;34)	3.72	**0.94**	34(36;34)
MW2	**1.22**	**0.87**	45(51;46)	2.95	**0.88**	46(51;47)	2.43	**0.94**	45(46;47)
MW3	203.52	0.65	36(44;47)	1.73	0.65	36(44;47)	117.42	**0.92**	45(47;47)
MW4	4.88	0.57	20(26;29)	1.37	0.59	20(26;28)	3.57	**0.97**	28(29;28)
MW5	8.07	0.35	14(34;20)	1.27	0.72	28(34;33)	10.22	0.56	19(20;33)
MW6	**1.19**	**0.94**	30(32;30)	2.48	**0.85**	28(32;29)	2.94	**0.90**	28(30;29)
MW7	2.03	0.68	30(38;36)	**1.07**	0.72	31(38;36)	2.17	**0.95**	35(36;36)
MW8	9.73	0.49	20(35;26)	17.38	0.55	30(35;50)	1.79	0.49	25(26;50)
	PH2 v EA2			**PH2 v EG2**			**EA2 v EG2**		
APK	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.
BN1	2.59	**0.84**	31(36;32)	1.70	**0.89**	33(36;34)	4.42	**0.89**	31(32;34)
BN2	2.91	0.68	17(25;17)	2.02	0.42	24(25;56)	5.89	0.28	16(17;56)
BN3	2.46	0.67	39(49;48)	**1.13**	0.67	39(49;48)	2.18	**0.96**	47(48;48)
BN4	**1.09**	**0.93**	51(54;52)	31.61	**0.89**	49(54;50)	29.01	**0.96**	50(52;50)
BN5	2.90	0.67	41(51;51)	**1.02**	0.70	42(51;51)	2.85	**0.96**	50(51;51)
BN6	2.78	0.55	17(24;24)	1.72	0.59	17(24;22)	4.79	**0.92**	22(24;22)
BN7	2.35	0.42	18(37;24)	**1.20**	0.62	28(37;36)	2.82	0.58	22(24;36)
BN8	2.79	0.41	13(27;18)	3.01	0.50	16(27;21)	**1.08**	**0.77**	17(18;21)

Note: **Bold** – threshold reached; [light green] – overall similarity at least 0.75 (good); [green] – overall similarity at least 0.90 (great)



	MW1	MW2	MW3	MW4	MW5	MW6	MW7	MW8
PH3 Total	3580	15044	16147	1094	6113	4330	17646	6971
EA3 Total	7674	3485	12155	1876	313	256816	14457	10238
EG3 Total	2619	2206	10361	971	8088	171947	13668	249006
PH3 n	38	50	46	33	30	19	34	26
EA3 n	36	45	43	25	12	30	29	24
EG3 n	34	46	47	28	32	28	33	26

Figure A2-7. Malware system call summaries – subgroup D (execution only)

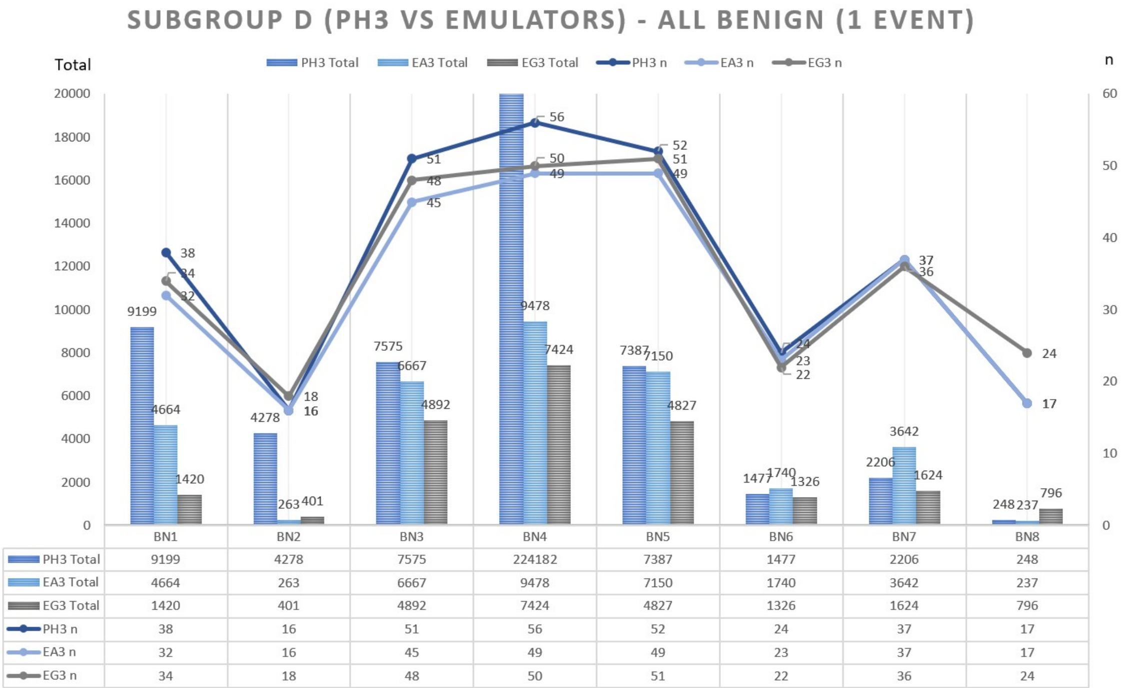
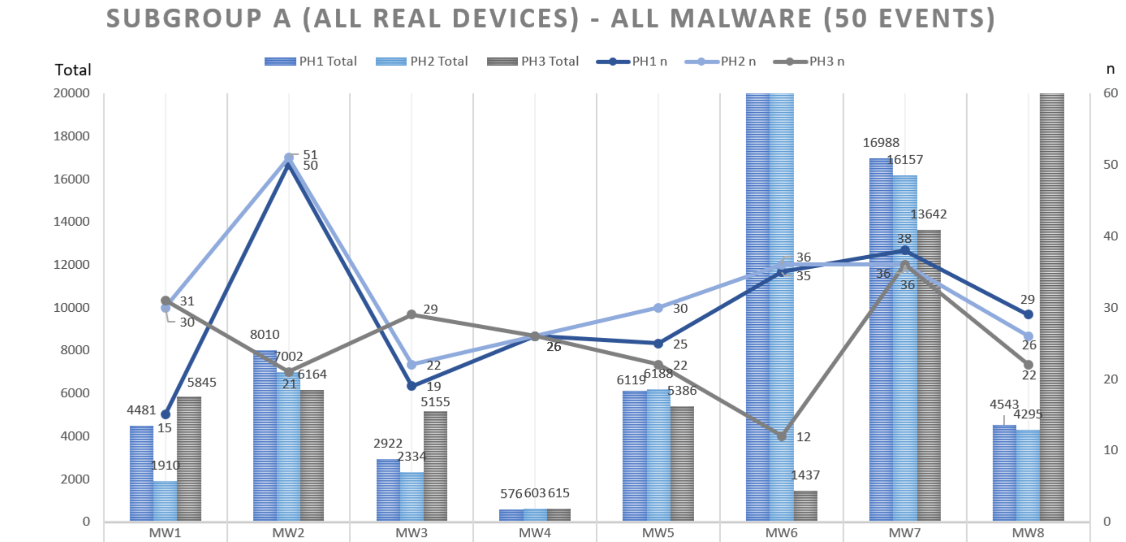


Figure A2-8. Benign apps' system call summaries – subgroup D (execution only)

Table A2-4. Similarity comparison – subgroup D (execution only)

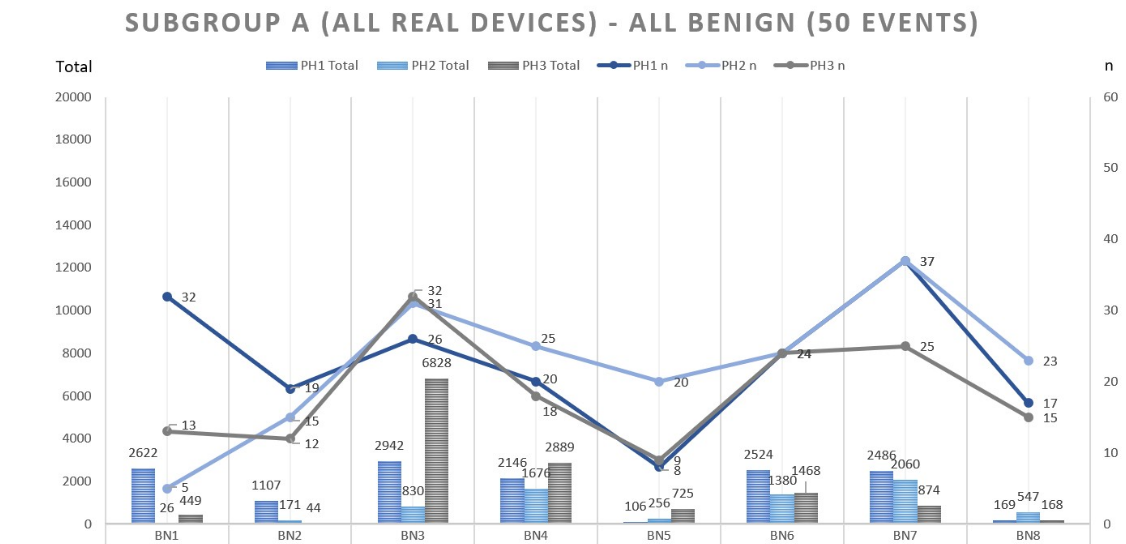
	PH3 v EA3			PH3 v EG3			EA3 v EG3		
APK	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.
MW1	2.14	**0.90**	35(38;36)	1.37	0.64	28(38;34)	2.93	0.67	28(36;34)
MW2	4.32	**0.86**	44(50;45)	6.82	**0.88**	45(50;46)	1.58	**0.98**	45(45;46)
MW3	1.33	**0.93**	43(46;43)	1.56	0.63	36(46;47)	**1.17**	0.67	36(43;47)
MW4	1.71	**0.75**	25(33;25)	**1.13**	0.61	23(33;28)	1.93	0.61	20(25;28)
MW5	19.53	0.40	12(30;12)	1.32	0.68	25(30;32)	25.84	0.33	11(12;32)
MW6	59.31	0.53	17(19;30)	39.71	0.52	16(19;28)	1.49	**0.93**	28(30;28)
MW7	**1.22**	0.57	23(34;29)	1.29	0.52	23(34;33)	**1.06**	0.59	23(29;33)
MW8	1.47	**0.92**	24(26;24)	35.72	0.68	21(26;26)	24.32	0.67	20(24;26)
	PH3 v EA3			**PH3 v EG3**			**EA3 v EG3**		
APK	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.
BN1	1.97	**0.79**	31(38;32)	6.48	**0.89**	34(38;34)	3.28	**0.89**	31(32;34)
BN2	16.27	**0.88**	15(16;16)	10.67	0.70	14(16;18)	1.52	**0.79**	15(16;18)
BN3	**1.14**	**0.88**	45(51;45)	1.55	0.65	39(51;48)	1.36	0.69	38(45;48)
BN4	23.65	**0.84**	48(56;49)	30.20	**0.86**	49(56;50)	1.28	**0.98**	49(49;50)
BN5	**1.03**	**0.94**	49(52;49)	1.53	0.72	43(52;51)	1.48	0.72	42(49;51)
BN6	**1.18**	**0.96**	23(24;23)	**1.11**	0.59	17(24;22)	1.31	0.61	17(23;22)
BN7	1.65	**1.00**	37(37;37)	1.36	0.62	28(37;36)	2.24	0.62	28(37;36)
BN8	**1.05**	**0.89**	16(17;17)	3.21	0.46	13(17;24)	3.36	0.46	13(17;24)

Note: **Bold** – threshold reached; [light green] – overall similarity at least 0.75 (good); [green] – overall similarity at least 0.90 (great)



	MW1	MW2	MW3	MW4	MW5	MW6	MW7	MW8
PH1 Total	4481	8010	2922	576	6119	598843	16988	4543
PH2 Total	1910	7002	2334	603	6188	502948	16157	4295
PH3 Total	5845	6164	5155	615	5386	1437	13642	72619
PH1 n	15	50	19	26	25	35	38	29
PH2 n	30	51	22	26	30	36	36	26
PH3 n	31	21	29	26	22	12	36	22

Figure A2-9. Malware system call summaries – subgroup A (50 events injected)



	BN1	BN2	BN3	BN4	BN5	BN6	BN7	BN8
PH1 Total	2622	1107	2942	2146	106	2524	2486	169
PH2 Total	26	171	830	1676	256	1380	2060	547
PH3 Total	449	44	6828	2889	725	1468	874	168
PH1 n	32	19	26	20	8	24	37	17
PH2 n	5	15	31	25	20	24	37	23
PH3 n	13	12	32	18	9	24	25	15

Figure A2-10. Benign apps' system call summaries – subgroup A (50 events injected)

Table A2-5. Similarity comparison – subgroup A (50 events injected)

	PH1 v PH2			PH1 v PH3			PH2 v PH3		
APK	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.
MW1	2.35	0.36	12(15;30)	1.30	0.39	13(15;31)	3.06	**0.91**	29(30;31)
MW2	**1.14**	**0.94**	49(50;51)	1.30	0.42	21(50;21)	**1.14**	0.41	21(51;21)
MW3	**1.25**	**0.86**	19(19;22)	1.76	0.60	18(19;29)	2.21	0.65	20(22;29)
MW4	**1.05**	**0.93**	25(26;26)	**1.07**	**0.86**	24(26;26)	**1.02**	**0.86**	24(26;26)
MW5	**1.01**	**0.83**	25(25;30)	**1.14**	0.74	20(25;22)	**1.15**	0.73	22(30;22)
MW6	**1.19**	**0.92**	34(35;36)	416.73	0.34	12(35;12)	350.00	0.33	12(36;12)
MW7	**1.05**	**0.95**	36(38;36)	**1.25**	**0.76**	32(38;36)	**1.18**	**0.80**	32(36;36)
MW8	**1.06**	0.72	23(29;26)	15.98	**0.76**	22(29;22)	16.91	**0.85**	22(26;22)
	PH1 v PH2			PH1 v PH3			PH2 v PH3		
APK	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.
BN1	100.85	0.16	5(32;5)	5.84	0.41	13(32;13)	17.27	0.38	5(5;13)
BN2	6.47	0.70	14(19;15)	25.16	0.55	11(19;12)	3.89	**0.80**	12(15;12)
BN3	3.54	0.73	24(26;31)	2.32	**0.76**	25(26;32)	8.23	**0.91**	30(31;32)
BN4	1.28	**0.80**	20(20;25)	1.35	0.73	16(20;18)	1.72	0.72	18(25;18)
BN5	2.42	0.40	8(8;20)	6.84	0.70	7(8;9)	2.83	0.45	9(20;9)
BN6	1.83	**1.00**	24(24;24)	1.72	**0.85**	22(24;24)	**1.06**	**0.85**	22(24;24)
BN7	**1.21**	**1.00**	37(37;37)	2.84	0.68	25(37;25)	2.36	0.68	25(37;25)
BN8	3.24	0.74	17(17;23)	**1.01**	0.68	13(17;15)	3.26	0.58	14(23;15)

Note: **Bold** – threshold reached; (light green) – overall similarity at least 0.75 (good); (green) – overall similarity at least 0.90 (great)

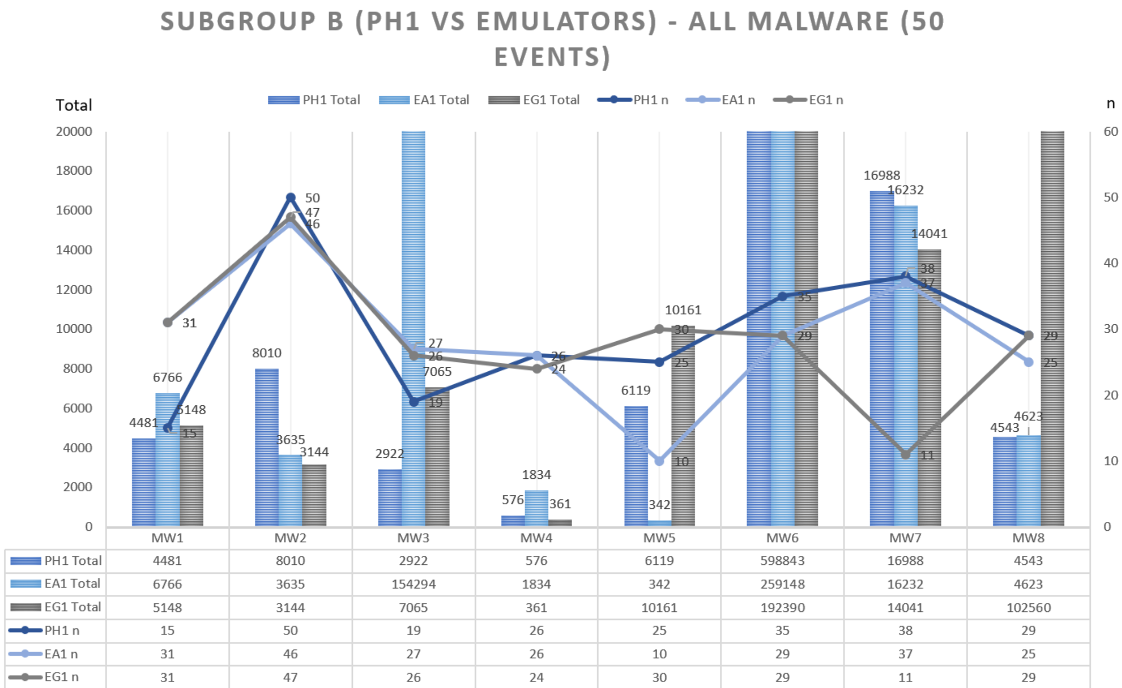


Figure A2-11. Malware system call summaries – subgroup B (50 events injected)

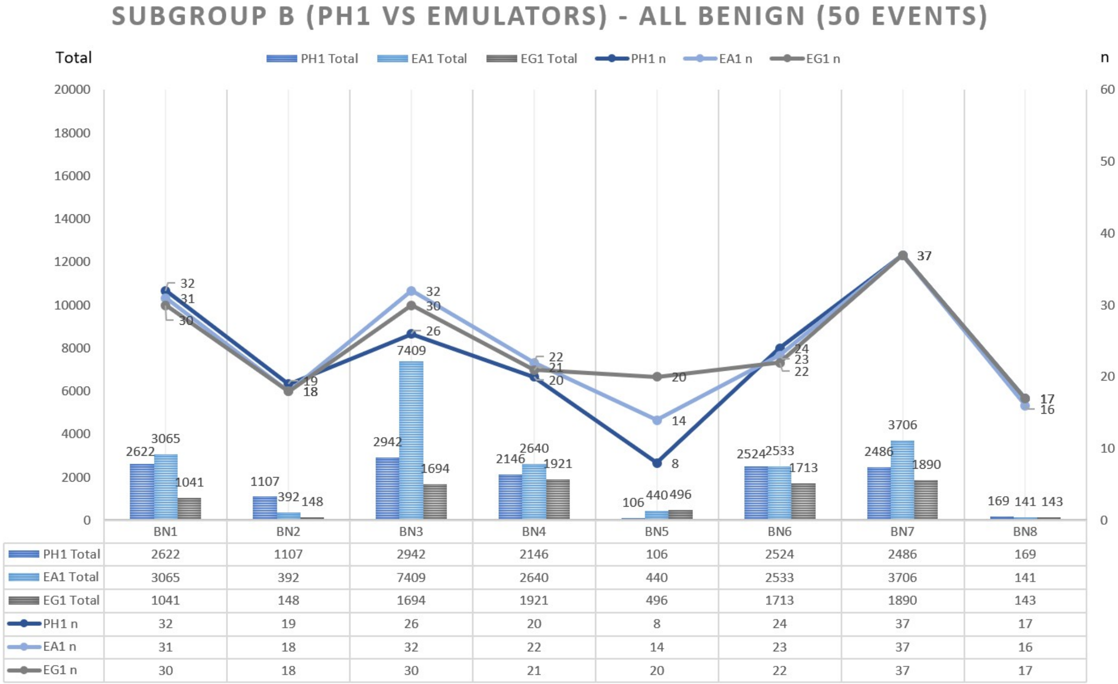
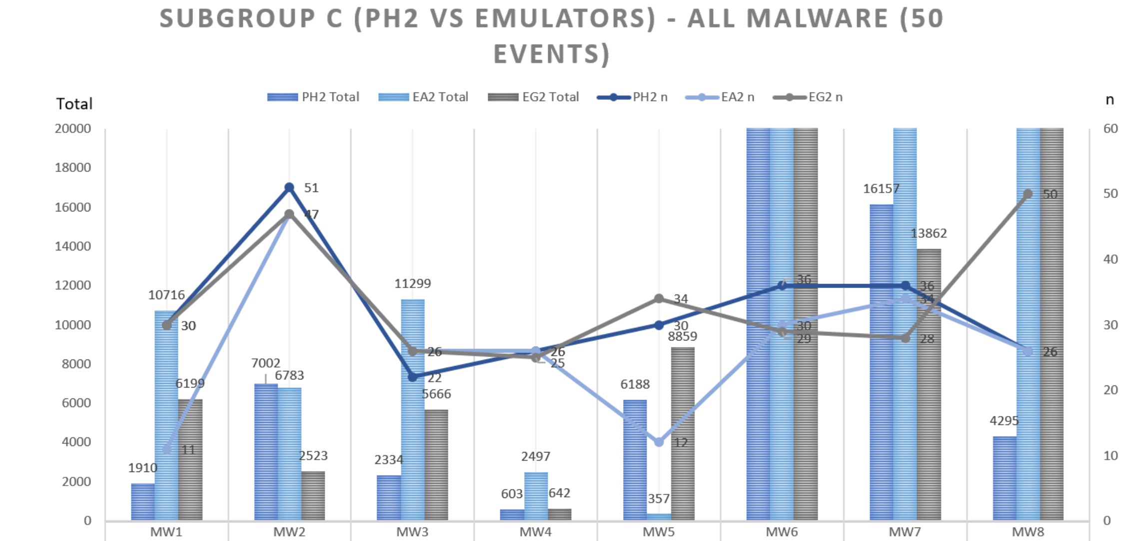


Figure A2-12. Benign apps' system call summaries – subgroup B (50 events injected)

Table A2-6. Similarity comparison – subgroup B (50 events injected)

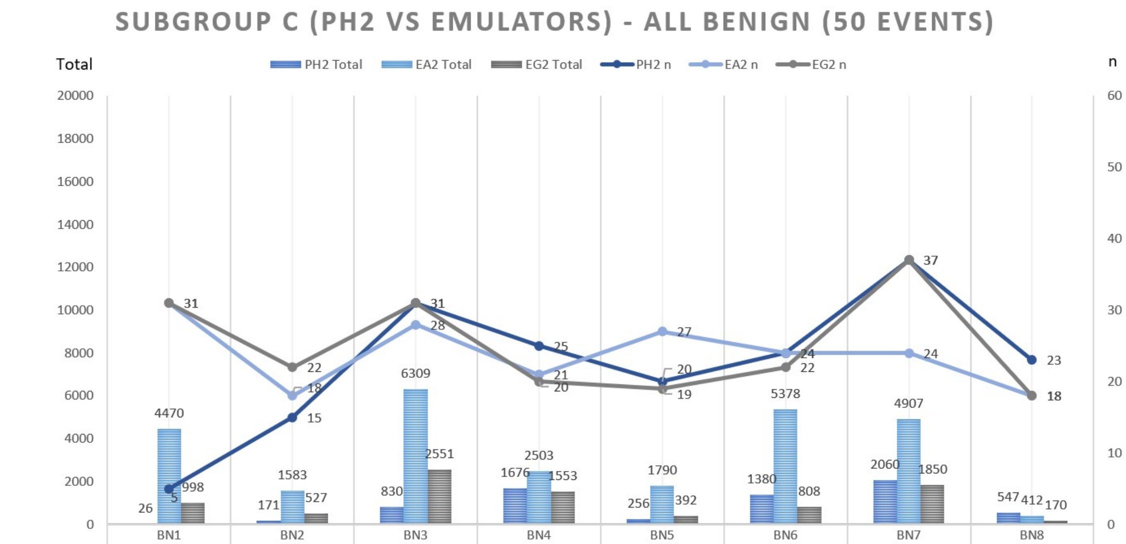
	PH1 v EA1			PH1 v EG1			EA1 v EG1		
APK	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.
MW1	1.51	0.35	12(15;31)	**1.15**	0.31	11(15;31)	1.31	**0.94**	30(31;31)
MW2	2.20	**0.81**	43(50;46)	2.55	**0.90**	46(50;47)	**1.16**	**0.90**	44(46;47)
MW3	52.80	0.53	16(19;27)	2.42	0.55	16(19;26)	21.84	**0.89**	25(27;26)
MW4	3.18	0.63	20(26;26)	1.60	0.61	19(26;24)	5.08	**0.92**	24(26;24)
MW5	17.89	0.30	8(25;10)	1.66	0.62	21(25;30)	29.71	0.33	10(10;30)
MW6	2.31	**0.83**	29(35;29)	3.11	**0.83**	29(35;29)	1.35	**0.93**	28(29;29)
MW7	**1.05**	0.70	31(38;37)	**1.21**	0.26	10(38;11)	**1.16**	0.30	11(37;11)
MW8	**1.02**	0.64	21(29;25)	22.58	0.71	24(29;29)	22.18	**0.86**	25(25;29)
	PH1 v EA1			**PH1 v EG1**			**EA1 v EG1**		
APK	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.
BN1	**1.17**	**0.91**	30(32;31)	2.52	**0.94**	30(32;30)	2.94	**0.97**	30(31;30)
BN2	2.82	**0.95**	18(19;18)	7.48	**0.85**	17(19;18)	2.65	**0.89**	17(18;18)
BN3	2.52	0.57	21(26;32)	1.74	0.60	21(26;30)	4.37	**0.82**	28(32;30)
BN4	**1.23**	**0.75**	18(20;22)	**1.12**	**0.86**	19(20;21)	1.37	**0.79**	19(22;21)
BN5	4.15	0.47	7(8;14)	4.68	0.33	7(8;20)	**1.13**	0.70	14(14;20)
BN6	**1.00**	0.62	18(24;23)	1.47	0.64	18(24;22)	1.48	**0.96**	22(23;22)
BN7	1.49	0.64	29(37;37)	1.32	0.64	29(37;37)	1.96	**1.00**	37(37;37)
BN8	**1.20**	0.65	13(17;16)	**1.18**	0.70	14(17;17)	**1.01**	**0.94**	16(16;17)

Note: **Bold** – threshold reached; [|] – overall similarity at least 0.75 (good); [|] – overall similarity at least 0.90 (great)



	MW1	MW2	MW3	MW4	MW5	MW6	MW7	MW8
PH2 Total	1910	7002	2334	603	6188	502948	16157	4295
EA2 Total	10716	6783	11299	2497	357	584115	28584	400113
EG2 Total	6199	2523	5666	642	8859	210356	13862	76949
PH2 n	30	51	22	26	30	36	36	26
EA2 n	11	47	26	26	12	30	34	26
EG2 n	30	47	26	25	34	29	28	50

Figure A2-13. Malware system call summaries – subgroup C (50 events injected)



	BN1	BN2	BN3	BN4	BN5	BN6	BN7	BN8
PH2 Total	26	171	830	1676	256	1380	2060	547
EA2 Total	4470	1583	6309	2503	1790	5378	4907	412
EG2 Total	998	527	2551	1553	392	808	1850	170
PH2 n	5	15	31	25	20	24	37	23
EA2 n	31	18	28	21	27	24	24	18
EG2 n	31	22	31	20	19	22	37	18

Figure A2-14. Benign apps' system call summaries – subgroup C (50 events injected)

Table A2-7. Similarity comparison – subgroup C (50 events injected)

	PH2 v EA2			PH2 v EG2			EA2 v EG2		
APK	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.
MW1	5.61	0.24	8(30;11)	3.25	0.67	24(30;30)	1.73	0.32	10(11;30)
MW2	**1.03**	**0.88**	46(51;47)	2.78	**0.88**	46(51;47)	2.69	**0.96**	46(47;47)
MW3	4.84	0.50	16(22;26)	2.43	0.50	16(22;26)	1.99	0.73	22(26;26)
MW4	4.14	0.58	19(26;26)	**1.06**	0.59	19(26;25)	3.89	**0.96**	25(26;25)
MW5	17.33	0.35	11(30;12)	1.43	0.68	26(30;34)	24.82	0.35	12(12;34)
MW6	**1.16**	**0.83**	30(36;30)	2.39	**0.76**	28(36;29)	2.78	**0.90**	28(30;29)
MW7	1.77	0.67	28(36;34)	**1.17**	0.56	23(36;28)	2.06	**0.82**	28(34;28)
MW8	93.16	0.49	17(26;26)	17.92	0.38	21(26;50)	5.20	0.49	25(26;50)
	PH2 v EA2			**PH2 v EG2**			**EA2 v EG2**		
APK	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.
BN1	171.92	0.16	5(5;31)	38.38	0.16	5(5;31)	4.48	**0.94**	30(31;31)
BN2	9.26	**0.79**	15(15;18)	3.08	0.54	13(15;22)	3.00	0.71	17(18;22)
BN3	7.60	0.47	19(31;28)	3.07	0.55	22(31;31)	2.47	**0.84**	27(28;31)
BN4	1.49	**0.77**	20(25;21)	**1.08**	**0.80**	20(25;20)	1.61	**0.78**	18(21;20)
BN5	6.99	0.38	13(20;27)	1.53	0.50	13(20;19)	4.57	0.59	17(27;19)
BN6	3.90	0.55	17(24;24)	1.71	0.59	17(24;22)	6.66	**0.92**	22(24;22)
BN7	2.38	0.42	18(37;24)	**1.11**	0.61	28(37;37)	2.65	0.56	22(24;37)
BN8	1.33	0.46	13(23;18)	3.22	0.52	14(23;18)	2.42	**0.89**	17(18;18)

Note: **Bold** – threshold reached; [ ] – overall similarity at least 0.75 (good); [ ] – overall similarity at least 0.90 (great)

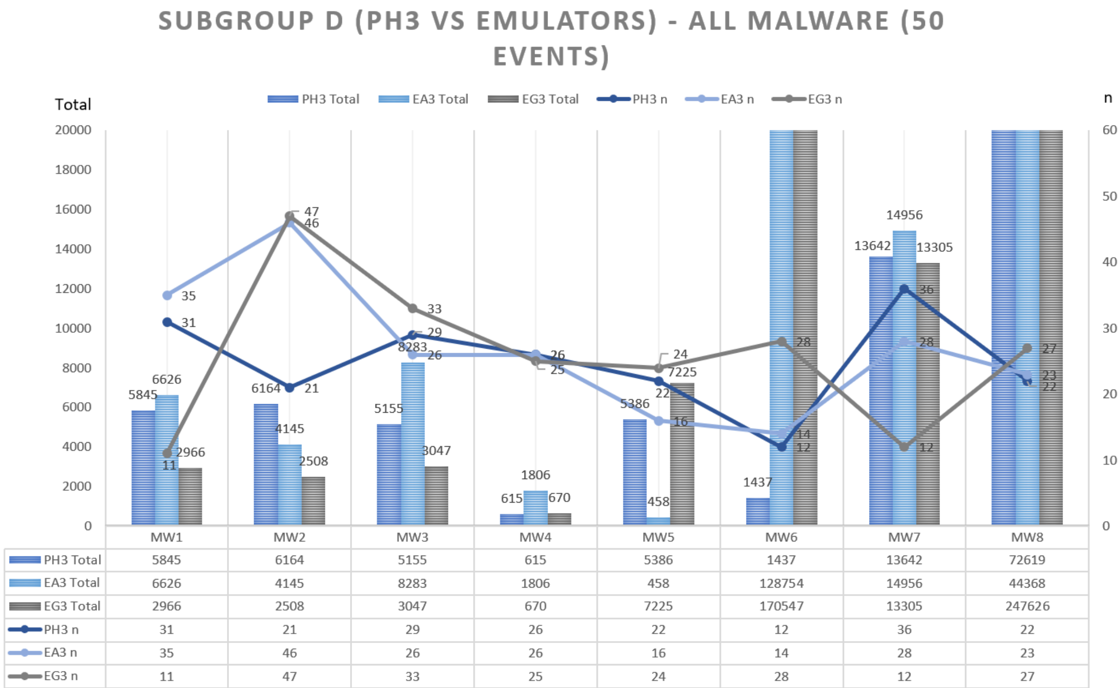
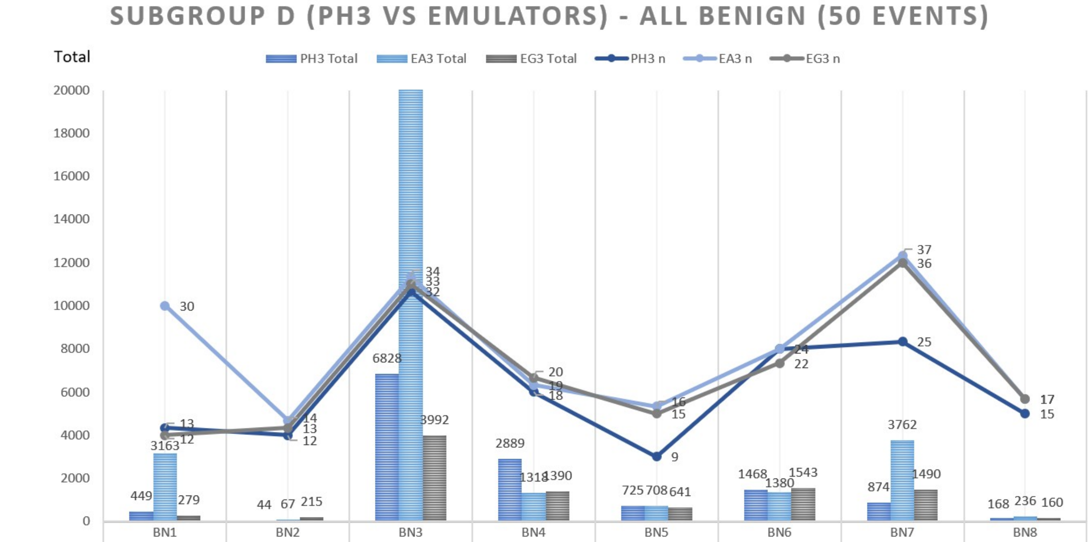


Figure A2-15. Malware system call summaries – subgroup D (50 events injected)



	BN1	BN2	BN3	BN4	BN5	BN6	BN7	BN8
PH3 Total	449	44	6828	2889	725	1468	874	168
EA3 Total	3163	67	255788	1318	708	1380	3762	236
EG3 Total	279	215	3992	1390	641	1543	1490	160
PH3 n	13	12	32	18	9	24	25	15
EA3 n	30	14	34	19	16	24	37	17
EG3 n	12	13	33	20	15	22	36	17

Figure A2-16. Benign apps' system call summaries – subgroup D (50 events injected)

Table A2-8. Similarity comparison – subgroup D (50 events injected)

	PH3 v EA3			PH3 v EG3			EA3 v EG3		
APK	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.
MW1	**1.13**	**0.83**	30(31;35)	1.97	0.27	9(31;11)	2.23	0.24	9(35;11)
MW2	1.49	0.43	20(21;46)	2.46	0.42	20(21;47)	1.65	**0.98**	46(46;47)
MW3	1.61	**0.83**	25(29;26)	1.69	0.55	22(29;33)	2.72	0.51	20(26;33)
MW4	2.94	**0.93**	25(26;26)	**1.09**	0.65	20(26;25)	2.70	0.65	20(26;25)
MW5	11.76	0.46	12(22;16)	1.34	0.64	18(22;24)	15.78	0.38	11(16;24)
MW6	89.60	0.73	11(12;14)	118.68	0.33	10(12;28)	1.32	0.45	13(14;28)
MW7	**1.10**	**0.78**	28(36;28)	**1.03**	0.30	11(36;12)	**1.12**	0.38	11(28;12)
MW8	1.64	**0.96**	22(22;23)	3.41	0.58	18(22;27)	5.58	0.61	19(23;27)
	PH3 v EA3			**PH3 v EG3**			**EA3 v EG3**		
APK	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.	Total sum	Uniques	Intersect.
BN1	7.04	0.39	12(13;30)	1.61	0.67	10(13;12)	11.34	0.40	12(30;12)
BN2	1.52	0.73	11(12;14)	4.89	0.67	10(12;13)	3.21	**0.80**	12(14;13)
BN3	37.46	**0.94**	32(32;34)	1.71	0.67	26(32;33)	64.08	0.72	28(34;33)
BN4	2.19	**0.76**	16(18;19)	2.08	**0.81**	17(18;20)	**1.05**	**0.95**	19(19;20)
BN5	**1.02**	0.56	9(9;16)	**1.13**	0.50	8(9;15)	**1.10**	**0.82**	14(16;15)
BN6	**1.06**	**0.92**	23(24;24)	**1.05**	0.59	17(24;22)	**1.12**	0.59	17(24;22)
BN7	4.30	0.68	25(25;37)	1.70	0.49	20(25;36)	2.52	0.62	28(37;36)
BN8	1.40	**0.78**	14(15;17)	**1.05**	0.60	12(15;17)	1.48	0.55	12(17;17)

Note: **Bold** – threshold reached; – overall similarity at least 0.75 (good); – overall similarity at least 0.90 (great)

Appendix 3 – Log summaries

PH1 calls syscall	PH2 calls syscall	PH3 calls syscall	EA1 calls syscall	EA2 calls syscall	EA3 calls syscall	EG1 calls syscall	EG2 calls syscall	EG3 calls syscall
1105 clock_gettime	1621 clock_gettime	3449 gettimeofday	2394 clock_gettime	5877 clock_gettime	3298 clock_gettime	234 pread64	226 pread64	182 ioctl
239 fstat64	216 getuid32	3388 clock_gettime	164 ioctl	228 pread64	179 ioctl	201 ioctl	192 ioctl	162 pread64
211 ioctl	139 ioctl	303 pread64	116 pread64	196 ioctl	152 pread64	173 rt_sigprocmask	186 rt_sigprocmask	151 getuid32
186 getuid32	116 pread64	269 getuid32	114 getuid32	184 rt_sigprocmask	141 getuid32	166 mmap2	152 getuid32	107 mmap2
114 pread64	102 rt_sigprocmask	268 ioctl	89 fstatat64	141 getuid32	119 fstatat64	135 munmap	142 mmap2	101 rt_sigprocmask
80 rt_sigprocmask	84 epoll_pwait	203 rt_sigprocmask	83 mmap2	137 mmap2	97 rt_sigprocmask	116 getuid32	118 munmap	92 epoll_pwait
73 mmap2	80 mmap2	196 mmap2	83 rt_sigprocmask	131 fstatat64	93 mmap2	74 futex	106 madvise	86 munmap
61 munmap	69 fstatat64	166 munmap	77 epoll_pwait	118 munmap	90 epoll_pwait	73 epoll_pwait	95 epoll_pwait	72 fstatat64
57 futex	63 munmap	146 madvise	62 munmap	81 epoll_pwait	80 munmap	65 write	74 fstatat64	61 madvise
46 epoll_pwait	56 writev	145 writev	53 write	72 futex	56 write	62 fstatat64	57 write	60 write
39 writev	55 futex	140 fstatat64	44 futex	49 write	52 madvise	42 mprotect	45 futex	47 futex
33 write	52 madvise	81 epoll_pwait	44 faccessat	47 faccessat	50 futex	40 prctl	41 mprotect	38 faccessat
28 fstatat64	43 write	71 futex	35 recvfrom	43 mprotect	44 faccessat	38 faccessat	38 faccessat	36 recvfrom
22 faccessat	35 faccessat	64 mprotect	28 mprotect	42 madvise	36 recvfrom	38 close	36 recvfrom	31 mprotect
20 recvfrom	30 recvfrom	49 faccessat	26 writev	29 recvfrom	29 mprotect	36 openat	27 clock_gettime	24 clock_gettime
19 close	23 mprotect	47 write	26 fstat64	27 close	21 close	36 recvfrom	24 close	24 close
18 openat	19 fstat64	31 close	25 openat	24 fstat64	20 writev	35 fstat64	21 writev	21 fstat64
12 mprotect	18 gettimeofday	28 fstat64	24 close	20 writev	19 fstat64	30 madvise	21 fstat64	21 writev
11 prctl	16 close	27 prctl	21 prctl	18 openat	17 openat	24 clock_gettime	19 openat	19 openat
8 madvise	15 openat	26 openat	13 read	13 gettimeofday	12 read	19 writev	16 prctl	13 prctl
8 read	13 prctl	25 recvfrom	11 mkdirat	12 mkdirat	11 mkdirat	12 read	13 read	11 read
5 gettimeofday	7 clone	10 mkdirat	7 clone	10 prctl	8 getsockopt	10 clone	10 getsockopt	10 getsockopt
4 getdents64	6 getsockopt	10 clone	7 madvise	9 dup	8 dup	10 getsockopt	9 clone	9 clone
4 getsockopt	6 read	10 getsockopt	7 getsockopt	8 clone	7 clone	9 dup	8 dup	8 dup
3 clone	5 mkdirat	9 dup	5 dup	8 getsockopt	7 prctl	8 mkdirat	8 mkdirat	8 mkdirat
2 _llseek	4 getdents64	9 fcntl64	3 epoll_ctl	8 read	5 fcntl64	5 epoll_ctl	5 fcntl64	5 fcntl64
2 epoll_ctl	4 fchmodat	6 fchmodat	3 fchmodat	8 fcntl64	4 epoll_ctl	5 fchmodat	5 fchmodat	5 fchmodat
2 mkdirat	3 fcntl64	6 read	2 _llseek	4 fchmodat	3 fchmodat	2 lseek	4 lseek	4 lseek
2 fchmodat	2 fsync	4 epoll_ctl	2 fcntl64	4 epoll_ctl	2 _llseek	2 fsync	4 epoll_ctl	4 epoll_ctl
1 dup	2 _llseek	2 fsync	1 pwrite64	2 lseek	2 lseek	2 _llseek	2 fsync	2 fsync
1 pwrite64	2 epoll_ctl	2 _llseek	3569 total	2 _llseek	1 sched_yield	2 fcntl64	2 _llseek	2 _llseek
1 fcntl64	2 unlinkat	2 lseek		1 pwrite64	1 pwrite64	2 unlinkat	2 unlinkat	2 unlinkat
2417 total	1 renameat	2 unlinkat		7553 total	4664 total	1 pwrite64	1 pwrite64	1 pwrite64
	1 dup	1 readlinkat				1 renameat	1 renameat	1 renameat
	1 sched_yield	1 pwrite64				1 eventfd2	1710 total	1420 total
	1 pwrite64	1 fstatfs64				1 epoll_create1		
	2912 total	1 sysinfo				1710 total		
		1 renameat						
		9199 total						

Figure A3-1. BN1 system call summaries (execution only)

PH1 calls syscall	PH2 calls syscall	PH3 calls syscall	EA1 calls syscall	EA2 calls syscall	EA3 calls syscall	EG1 calls syscall	EG2 calls syscall	EG3 calls syscall
1205 clock_gettime	11 clock_gettime	278 clock_gettime	2166 clock_gettime	3471 clock_gettime	2184 clock_gettime	120 ioctl	110 pread64	45 recvfrom
231 fstat64	7 epoll_pwait	36 gettimeofday	108 pread64	110 pread64	108 pread64	116 pread64	107 getuid32	44 epoll_pwait
215 ioctl	4 getuid32	32 epoll_pwait	99 getuid32	97 ioctl	95 ioctl	90 rt_sigprocmask	104 ioctl	42 ioctl
190 getuid32	3 ioctl	27 recvfrom	95 ioctl	93 getuid32	90 getuid32	83 getuid32	94 epoll_pwait	42 write
118 pread64	1 mprotect	24 getuid32	75 rt_sigprocmask	84 write	79 rt_sigprocmask	81 recvfrom	81 rt_sigprocmask	35 getuid32
90 rt_sigprocmask	26 total	22 write	65 mmap2	79 epoll_pwait	77 write	79 write	70 recvfrom	27 madvise
77 mmap2		14 ioctl	62 epoll_pwait	79 rt_sigprocmask	74 epoll_pwait	78 mmap2	65 mmap2	18 futex
68 munmap		8 read	58 munmap	75 recvfrom	63 recvfrom	75 epoll_pwait	59 munmap	13 read
66 futex		3 futex	55 write	62 mmap2	61 mmap2	70 munmap	53 madvise	5 close
59 epoll_pwait		2 writev	51 recvfrom	59 munmap	58 munmap	50 futex	47 write	3 munmap
51 write		1 mprotect	38 writev	51 futex	52 futex	24 read	33 futex	3 writev
47 recvfrom		1 epoll_ctl	27 fstatat64	51 fstatat64	51 fstatat64	23 fstat64	29 fstatat64	2 epoll_ctl
39 writev		1 close	26 futex	24 madvise	41 madvise	21 fstatat64	25 read	279 total
28 fstatat64		449 total	22 fstat64	21 read	20 faccessat	19 faccessat	19 faccessat	
22 faccessat			20 faccessat	20 faccessat	18 read	18 close	17 fstat64	
19 close			16 close	17 fstat64	17 fstat64	16 openat	14 mprotect	
18 openat			16 mprotect	15 mprotect	16 mprotect	15 prctl	14 writev	
15 read			15 openat	13 writev	13 writev	13 mprotect	12 close	
13 mprotect			15 read	12 close	13 close	13 madvise	9 clock_gettime	
13 prctl			10 madvise	9 openat	9 openat	11 writev	9 openat	
10 madvise			6 prctl	5 gettimeofday	4 epoll_ctl	6 clock_gettime	5 prctl	
5 gettimeofday			4 getsockopt	4 getsockopt	4 getsockopt	4 epoll_ctl	4 getsockopt	
4 getdents64			3 epoll_ctl	3 prctl	3 prctl	4 getsockopt	3 clone	
4 getsockopt			2 clone	3 fcntl64	3 fcntl64	3 clone	3 fcntl64	
3 clone			2 _llseek	3 epoll_ctl	2 clone	2 _llseek	3 epoll_ctl	
3 epoll_ctl			2 mkdirat	2 clone	2 fchmodat	2 mkdirat	2 _llseek	
2 mkdirat			2 fchmodat	2 fchmodat	2 _llseek	2 fchmodat	2 mkdirat	
2 _llseek			2 sendto	2 _llseek	2 mkdirat	1 dup	2 fchmodat	
2 fchmodat			1 dup	2 mkdirat	1 pwrite64	1 pwrite64	1 dup	
1 dup			1 pwrite64	1 dup	1 dup	1 fcntl64	1 pwrite64	
1 pwrite64			1 fcntl64	1 pwrite64	3163 total	1041 total	1 sendto	
1 fcntl64			3065 total	4470 total			998 total	
2622 total								

Figure A3-2. BN1 system call summaries (50 events injected)

PH1	PH2	PH3	EA1	EA2	EA3	EG1	EG2	EG3
calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall
351 clock_gettime	894 clock_gettime	2182 clock_gettime	1568 clock_gettime	304 clock_gettime	69 recvfrom	615 mprotect	402 prctl	52 write
60 recvfrom	76 write	494 epoll_pwait	567 mprotect	46 recvfrom	38 futex	433 prctl	293 futex	43 epoll_pwait
50 futex	68 futex	462 getuid32	265 ioctl	32 epoll_pwait	37 write	375 futex	269 ioctl	42 getuid32
44 write	65 epoll_pwait	396 gettimeofday	256 futex	32 write	33 epoll_pwait	331 ioctl	205 rt_sigprocmask	41 recvfrom
30 epoll_pwait	65 recvfrom	318 ioctl	227 prctl	27 futex	23 ioctl	217 mmap2	184 mmap2	38 futex
22 ioctl	63 getuid32	178 read	211 mmap2	17 ioctl	19 getuid32	198 write	181 write	35 ioctl
21 read	59 ioctl	80 recvfrom	195 write	16 getuid32	14 sendto	163 pread64	166 pread64	32 rt_sigprocmask
17 getuid32	42 read	72 write	147 rt_sigprocmask	15 read	12 read	157 rt_sigprocmask	158 fstatat64	22 read
12 sendto	34 rt_sigprocmask	64 futex	146 pread64	10 sendto	4 clock_gettime	131 read	142 madvise	22 pread64
5 timerfd_settim	22 pread64	16 sendto	131 getuid32	4 timerfd_settim	3 mprotect	129 fstatat64	131 getuid32	19 sendto
3 prctl	20 sendto	6 timerfd_settim	114 read	3 mprotect	3 timerfd_settim	128 getuid32	125 mprotect	13 mmap2
3 mmap2	18 madvise	4 mprotect	99 fstatat64	2 rt_sigprocmask	2 rt_sigprocmask	114 munmap	120 read	12 madvise
2 mprotect	14 mmap2	2 mmap2	93 munmap	2 mmap2	2 mmap2	113 fstat64	98 munmap	11 munmap
2 madvise	11 munmap	2 rt_sigprocmask	85 epoll_pwait	2 madvise	2 madvise	87 madvise	60 epoll_pwait	10 prctl
1 gettimeofday	10 timerfd_settim	1 clone	71 recvfrom	1 gettimeofday	1 clone	73 epoll_pwait	55 fstat64	5 timerfd_settim
1 clone	6 openat	1 prctl	71 fstat64	1 clone	1 prctl	71 openat	52 close	2 writev
1 fstat64	5 mprotect		68 openat	1 prctl		69 close	49 recvfrom	1 clone
1 openat	4 gettimeofday		66 close			66 recvfrom	41 openat	1 mprotect
1 close	4 writev		29 clone			35 clock_gettime	38 clock_gettime	
	4 prctl		25 sendto			28 faccessat	35 fcntl64	
	4 close		24 writev			26 sendto	28 faccessat	
	3 fstat64		19 faccessat			24 fcntl64	25 sendto	
	3 fstatat64		11 dup			24 clone	24 clone	
	2 clone		11 rt_sigaction			15 pwrite64	16 writev	
	1 lseek		10 fcntl64			14 writev	15 pwrite64	
			7 _llseek			11 dup	12 dup	
			7 getsockopt			8 getsockopt	10 fdatasync	
			6 epoll_ctl			8 mkdirat	8 _llseek	
			6 readlinkat			7 _llseek	8 mkdirat	
			6 timerfd_settime			7 unlinkat	8 readlinkat	
			5 mkdirat			6 readlinkat	8 getsockopt	
			4 getdents64			6 epoll_ctl	6 lseek	
			4 unlinkat			6 timerfd_settim	6 epoll_ctl	
			3 lseek			5 fchmodat	6 timerfd_settime	
			3 fsync			4 fsync	5 fchmodat	
			3 ugetrlimit			4 getdents64	4 fsync	
			3 fchmodat			3 ugetrlimit	4 getdents64	
			3 getrandom			3 getrandom	4 unlinkat	
			2 getpriority			3 lseek	3 fstatfs64	
			2 mremap			2 getpriority	3 getrandom	
			2 fstatfs64			2 fdatasync	2 mremap	
			2 renameat			2 sched_yield	2 getpriority	
			2 eventfd2			2 mremap	2 rt_sigaction	
			1 setrlimit			2 fstatfs64	2 ugetrlimit	
			1 sysinfo			2 renameat	2 ftruncate64	
			1 sigaltstack			2 eventfd2	2 renameat	
			1 timerfd_create			2 socketpair	2 eventfd2	
			1 epoll_create1			1 setrlimit	2 socketpair	
			1 socketpair			1 sysinfo	1 sysinfo	
						1 uname	1 uname	
						1 rt_sigaction	1 sched_yield	
						1 ftruncate64	1 geteuid32	
						1 timerfd_create	1 timerfd_create	
						1 epoll_create1	1 epoll_create1	
						1 setsockopt	1 setsockopt	
						1 sendmsg	1 sendmsg	
627 total	1497 total	4278 total	4585 total	515 total	263 total	3742 total	3031 total	401 total

Figure A3-3. BN2 system call summaries (execution only)

PH1	PH2	PH3	EA1	EA2	EA3	EG1	EG2	EG3
calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall
592 clock_gettime	96 clock_gettime	15 clock_gettime	85 recvfrom	1162 clock_gettime	15 epoll_pwait	37 futex	82 epoll_pwait	60 recvfrom
116 recvfrom	16 epoll_pwait	12 futex	63 futex	52 epoll_pwait	10 read	20 recvfrom	65 recvfrom	37 write
113 write	16 futex	3 ioctl	60 write	52 write	9 ioctl	19 epoll_pwait	59 write	27 epoll_pwait
71 epoll_pwait	12 read	2 epoll_pwait	53 epoll_pwait	46 getuid32	8 futex	17 write	51 futex	25 futex
64 futex	8 ioctl	2 mmap2	31 ioctl	44 recvfrom	4 write	14 read	49 getuid32	20 getuid32
41 read	8 timerfd_settim	2 mprotect	27 read	43 futex	4 timerfd_settim	9 ioctl	45 read	18 ioctl
39 ioctl	3 getuid32	2 rt_sigprocmask	25 getuid32	39 ioctl	4 clock_gettime	7 getuid32	40 ioctl	17 read
34 getuid32	2 write	2 gettimeofday	17 sendto	32 rt_sigprocmask	3 getuid32	4 mprotect	32 rt_sigprocmask	3 mprotect
16 sendto	2 rt_sigprocmask	1 getuid32	8 clock_gettime	22 pread64	2 mprotect	4 timerfd_settim	22 pread64	2 rt_sigprocmask
6 timerfd_settim	2 mmap2	1 clone	7 madvise	22 read	2 rt_sigprocmask	4 sendto	22 sendto	2 mmap2
3 prctl	2 madvise	1 read	4 prctl	19 sendto	2 mmap2	3 prctl	13 mmap2	2 madvise
3 mmap2	1 gettimeofday	1 prctl	3 mmap2	13 mmap2	2 madvise	3 mmap2	13 madvise	1 clone
2 mprotect	1 clone		3 timerfd_settim	13 prctl	1 clone	2 madvise	11 munmap	1 prctl
2 madvise	1 mprotect		2 mprotect	11 munmap	1 prctl	1 close	10 prctl	
1 close	1 prctl		1 close	5 timerfd_settim		1 clone	5 timerfd_settim	
1 gettimeofday			1 clone	3 madvise		1 sched_yield	2 mprotect	
1 clone			1 fstat64	2 mprotect		1 fstat64	1 close	
1 fstat64			1 openat	2 gettimeofday		1 openat	1 dup	
1 openat				1 clone			1 clone	
							1 writev	
							1 fstat64	
							1 sendmsg	
1107 total	171 total	44 total	392 total	1583 total	67 total	148 total	527 total	215 total

Figure A3-4. BN2 system call summaries (50 events injected)

PH1

calls	syscall
102	ioctl
89	futex
71	getuid
70	fstat
34	epoll_pwait
27	mmap
26	write
24	pread64
15	writev
13	munmap
12	prctl
12	recvfrom
11	read
11	mprotect
10	close
5	openat
5	newfstatat
4	clone
3	dup
2	fcntl
2	timerfd_settime
2	clock_gettime
2	getrandom
1	epoll_ctl
553	total

PH2

calls	syscall
596	newfstatat
502	futex
351	mprotect
338	ioctl
305	mmap
283	prctl
240	getuid
232	read
229	write
222	pread64
151	munmap
119	epoll_pwait
93	close
92	openat
86	rt_sigprocmask
70	clock_gettime
69	fstat
59	faccessat
43	writev
43	madvise
40	fcntl
39	lseek
32	recvfrom
30	sendto
27	clone
20	readlinkat
14	timerfd_settime
11	dup
11	fstatfs
11	getrandom
10	mkdirat
8	getsockopt
7	rt_sigaction
6	mremap
5	epoll_ctl
5	uname
4	getdents64
4	sched_yield
2	getpriority
2	socketpair
1	eventfd2
1	ftruncate
1	fchmodat
1	timerfd_create
1	setpriority
1	getrlimit
1	sysinfo
1	setsockopt
1	sendmsg
4420	total

PH3

calls	syscall
1014	futex
851	getuid
767	ioctl
724	newfstatat
704	epoll_pwait
444	prctl
412	read
393	mprotect
335	mmap
254	pread64
249	clock_gettime
227	write
158	rt_sigprocmask
145	munmap
95	close
93	writev
92	openat
82	fstat
79	lseek
73	faccessat
60	sched_yield
51	madvise
45	fcntl
44	clone
32	recvfrom
25	sendto
19	readlinkat
12	mkdirat
12	dup
10	fstatfs
10	getsockopt
10	getrandom
9	timerfd_settime
7	rt_sigaction
6	epoll_ctl
6	mremap
5	uname
4	getdents64
2	socketpair
2	sysinfo
2	eventfd2
2	getpriority
1	sendmsg
1	epoll_create1
1	fchmodat
1	timerfd_create
1	getrlimit
1	socket
1	connect
1	setsockopt
1	ftruncate
7575	total

EA1

calls	syscall
1701	clock_gettime
1201	mprotect
462	fstatat64
390	futex
363	ioctl
330	mmap2
296	prctl
228	getuid32
215	write
175	pread64
170	epoll_pwait
163	read
133	munmap
131	openat
125	close
93	fstat64
71	recvfrom
62	writev
46	rt_sigprocmask
36	faccessat
32	clone
24	fcntl64
16	_llseek
15	dup
13	readlinkat
13	sendto
10	lseek
10	rt_sigaction
9	mkdirat
8	getrandom
7	mremap
7	fstatfs64
7	timerfd_settime
6	unlinkat
6	epoll_ctl
5	getsockopt
4	getdents64
3	ugetrlimit
2	getpriority
2	eventfd2
1	setrlimit
1	sysinfo
1	sched_yield
1	sigaltstack
1	fchmodat
1	timerfd_create
1	epoll_create1
1	socketpair
6598	total

EA2

calls	syscall
6610	clock_gettime
728	fstatat64
452	prctl
343	ioctl
321	futex
298	mprotect
285	mmap2
223	pread64
198	write
195	getuid32
159	read
140	munmap
134	epoll_pwait
125	madvise
118	rt_sigprocmask
79	openat
76	close
57	fstat64
40	faccessat
38	fcntl64
34	writev
32	recvfrom
32	readlinkat
24	clone
22	_llseek
17	gettimeofday
13	fstatfs64
12	sendto
12	lseek
11	mkdirat
11	dup
8	getrandom
7	timerfd_settime
7	mremap
5	getsockopt
5	epoll_ctl
4	getdents64
2	socketpair
2	getpriority
2	ugetrlimit
1	sched_yield
1	fchmodat
1	setpriority
1	sysinfo
1	uname
1	timerfd_create
1	eventfd2
1	setsockopt
10889	total

EA3

calls	syscall
2396	clock_gettime
725	newfstatat
457	prctl
353	ioctl
323	futex
313	mprotect
290	mmap
221	pread64
198	getuid
184	read
168	write
166	epoll_pwait
141	munmap
120	rt_sigprocmask
84	close
79	openat
62	fstat
41	faccessat
41	fcntl
37	madvise
37	lseek
35	writev
32	readlinkat
32	recvfrom
25	clone
13	fstatfs
12	sendto
12	dup
11	mkdirat
11	sched_yield
8	getrandom
7	mremap
7	timerfd_settime
5	epoll_ctl
5	getsockopt
4	getdents64
2	socketpair
2	getrlimit
2	getpriority
1	fchmodat
1	uname
1	sysinfo
1	timerfd_create
1	eventfd2
1	setsockopt
6667	total

EG1

calls	syscall
1506	futex
1112	mprotect
476	prctl
474	fstatat64
408	ioctl
306	mmap2
240	getuid32
227	write
206	read
205	epoll_pwait
140	openat
134	fstat64
132	pread64
131	close
126	madvise
105	munmap
78	clock_gettime
42	writev
40	recvfrom
40	faccessat
36	clone
26	fcntl64
16	_llseek
15	rt_sigprocmask
14	dup
14	sendto
11	lseek
11	readlinkat
11	timerfd_settime
10	getrandom
9	mkdirat
8	getdents64
6	mremap
6	epoll_ctl
6	fstatfs64
6	unlinkat
6	getsockopt
2	getpriority
2	sched_yield
2	ugetrlimit
2	eventfd2
2	socketpair
1	setpriority
1	sysinfo
1	uname
1	fchmodat
1	timerfd_create
1	epoll_create1
1	setsockopt
1	sendmsg
6357	total

EG2

calls	syscall
770	futex
555	fstatat64
429	prctl
386	ioctl
273	mmap2
271	mprotect
269	clock_gettime
257	getuid32
205	pread64
198	write
185	epoll_pwait
178	read
170	madvise
146	rt_sigprocmask
116	munmap
73	close
66	fstat64
62	openat
44	faccessat
42	writev
40	fcntl64
38	clone
32	recvfrom
26	sendto
18	_llseek
17	readlinkat
16	sched_yield
14	lseek
12	dup
11	mkdirat
11	getrandom
10	timerfd_settime
9	fstatfs64
8	getdents64
6	epoll_ctl
6	getsockopt
5	mremap
2	getpriority
2	ugetrlimit
2	eventfd2
2	socketpair
1	setpriority
1	sysinfo
1	uname
1	fchmodat
1	timerfd_create
1	epoll_create1
1	setsockopt
4989	total

EG3

calls	syscall
746	futex
553	fstatat64
428	prctl
363	ioctl
269	mmap2
265	mprotect
263	clock_gettime
249	getuid32
207	write
203	pread64
197	epoll_pwait
185	read
170	madvise
144	rt_sigprocmask
115	munmap
69	close
62	openat
62	fstat64
43	faccessat
40	writev
37	clone
37	fcntl64
32	recvfrom
18	_llseek
17	readlinkat
14	lseek
13	sendto
11	dup
11	mkdirat
11	getrandom
10	timerfd_settime
9	fstatfs64
8	getdents64
6	getsockopt
5	mremap
5	epoll_ctl
2	getpriority
2	ugetrlimit
2	socketpair
1	setpriority
1	sysinfo
1	uname
1	sched_yield
1	fchmodat
1	timerfd_create
1	eventfd2
1	setsockopt
1	sendmsg
4892	total

Figure A3-5. BN3 system call summaries (execution only)

PH1

calls	syscall
1769	futex
260	getuid
215	ioctl
143	fstat
127	epoll_pwait
71	writev
54	clock_gettime
47	write
41	mmap
40	pread64
30	read
24	recvfrom
21	prctl
19	munmap
17	mprotect
13	close
9	getrandom
7	openat
7	clone
6	rt_sigprocmask
5	timerfd_settime
4	dup
4	fcntl
4	sched_yield
3	epoll_ctl
2	sendto
2942	total

PH2

calls	syscall
154	epoll_pwait
98	getuid
77	recvfrom
73	futex
71	write
66	ioctl
61	read
58	pread64
32	rt_sigprocmask
31	mmap
23	clock_gettime
19	munmap
18	writev
9	timerfd_settime
7	openat
7	close
4	newfstatat
4	fstat
2	epoll_ctl
2	dup
2	fcntl
2	sendto
2	mprotect
1	unlinkat
1	faccessat
1	fchmodat
1	lseek
1	fsync
1	prctl
1	getsockopt
1	clone
830	total

PH3

calls	syscall
1463	write
1428	recvfrom
1019	epoll_pwait
711	futex
647	getuid
566	ioctl
439	read
116	clock_gettime
82	pread64
64	rt_sigprocmask
63	mmap
57	writev
40	newfstatat
27	munmap
23	mprotect
22	prctl
9	fstat
8	clone
7	timerfd_settime
7	sendto
7	close
4	faccessat
4	fcntl
3	dup
3	getrandom
3	epoll_ctl
1	fsync
1	renameat
1	openat
1	unlinkat
1	fchmodat
1	getsockopt
6828	total

EA1

calls	syscall
1407	recvfrom
1299	clock_gettime
1294	write
1076	futex
812	epoll_pwait
454	getuid32
442	ioctl
376	read
42	pread64
41	mmap2
24	writev
23	munmap
21	fstat64
18	prctl
15	close
15	mprotect
9	openat
7	fstatat64
5	clone
4	dup
4	sendto
3	rt_sigprocmask
3	madvise
3	timerfd_settime
3	getrandom
2	fcntl64
2	epoll_ctl
1	fsync
1	unlinkat
1	fchmodat
1	faccessat
1	getsockopt
7409	total

EA2

calls	syscall
4270	clock_gettime
584	futex
224	epoll_pwait
178	getuid32
175	ioctl
154	write
114	recvfrom
78	read
76	mmap2
73	rt_sigprocmask
68	pread64
40	madvise
38	sched_yield
35	writev
30	mprotect
29	prctl
26	munmap
25	fstatat64
20	clone
18	gettimeofday
11	getrandom
10	close
8	timerfd_settime
7	sendto
6	fcntl64
4	fstat64
4	dup
4	epoll_ctl
6309	total

EA3

calls	syscall
56257	recvfrom
45191	epoll_pwait
27447	write
25078	read
22867	sendto
22760	ioctl
20220	futex
19833	getuid
14778	clock_gettime
398	madvise
140	prctl
136	mmap
112	mprotect
96	newfstatat
77	writev
64	rt_sigprocmask
58	pread64
50	sched_yield
48	timerfd_settime
37	close
35	munmap
27	fstat
19	fcntl
17	clone
17	faccessat
14	getrandom
3	dup
3	epoll_ctl
1	openat
1	fsync
1	unlinkat
1	renameat
1	fchmodat
1	getsockopt
255788	total

EG1

calls	syscall
507	futex
233	epoll_pwait
137	getuid32
135	write
129	ioctl
121	recvfrom
86	read
50	mmap2
42	pread64
36	clock_gettime
35	prctl
29	fstat64
25	munmap
22	mprotect
19	madvise
19	close
13	openat
11	writev
8	clone
8	timerfd_settime
4	dup
4	getdents64
4	fstatat64
4	sendto
3	fcntl64
3	epoll_ctl
3	getrandom
2	faccessat
1	sched_yield
1	getsockopt
1694	total

EG2

calls	syscall
948	futex
406	clock_gettime
215	getuid32
188	epoll_pwait
166	ioctl
87	write
65	rt_sigprocmask
65	mmap2
58	pread64
47	read
37	writev
37	recvfrom
35	madvise
28	prctl
28	sched_yield
25	fstat64
21	mprotect
20	munmap
16	clone
11	fstatat64
11	getrandom
10	close
5	fcntl64
5	timerfd_settime
4	dup
4	sendto
2	getdents64
2	epoll_ctl
2	openat
2	faccessat
1	getsockopt
2551	total

EG3

calls	syscall
1056	recvfrom
760	write
463	epoll_pwait
435	futex
336	getuid32
298	ioctl
297	read
79	clock_gettime
72	pread64
48	rt_sigprocmask
42	mmap2
21	munmap
12	prctl
9	madvise
7	writev
7	fstat64
6	faccessat
6	timerfd_settime
5	close
3	fsync
3	openat
3	fstatat64
3	unlinkat
3	fchmodat
3	sendto
3	getsockopt
2	dup
2	fcntl64
2	epoll_ctl
2	renameat
2	getrandom
1	mprotect
1	sched_yield
3992	total

Figure A3-6. BN3 system call summaries (50 events injected)

PH1	PH2	PH3	EA1	EA2	EA3	EG1	EG2	EG3
calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall
827 futex	161629 gettimeofday	175901 gettimeofday	14297 mprotect	172932 gettimeofday	2491 clock_gettime	12115 mprotect	842 read	980 futex
624 clock_gettime	29381 cacheflush	22934 cacheflush	1739 clock_gettime	6980 clock_gettime	864 read	1576 futex	619 _llseek	856 read
232 gettimeofday	7935 clock_gettime	15519 clock_gettime	831 read	1501 mprotect	722 fstatat64	836 read	581 mprotect	619 _llseek
118 epoll_pwait	866 read	1064 futex	737 futex	877 prctl	720 futex	781 prctl	576 prctl	614 mprotect
100 ioctl	845 futex	1019 read	657 prctl	824 read	620 _llseek	637 _llseek	564 futex	591 prctl
84 getuid32	624 _llseek	991 mprotect	637 _llseek	734 fstatat64	576 prctl	494 openat	547 fstatat64	547 fstatat64
61 write	603 fstatat64	837 getuid32	517 openat	686 futex	477 mprotect	486 fstatat64	339 ioctl	385 ioctl
53 recvfrom	475 mprotect	717 fstatat64	512 mmap2	642 _llseek	429 ioctl	461 mmap2	321 mmap2	342 mmap2
51 read	463 ioctl	698 ioctl	470 ioctl	556 mmap2	362 mmap2	448 ioctl	258 write	274 getuid32
23 fstat64	449 madvise	680 prctl	451 fstatat64	468 madvise	285 getuid32	262 getuid32	228 madvise	266 madvise
22 prctl	390 getuid32	646 epoll_pwait	345 getuid32	440 ioctl	250 epoll_pwait	234 madvise	217 getuid32	257 clock_gettime
21 mmap2	379 mmap2	635 _llseek	244 munmap	415 openat	222 write	233 write	188 epoll_pwait	244 write
15 mprotect	350 prctl	475 mmap2	243 write	362 munmap	222 madvise	211 munmap	173 pread64	235 epoll_pwait
14 madvise	260 write	275 write	233 epoll_pwait	293 getuid32	189 pread64	205 pread64	168 munmap	173 pread64
8 writev	244 pread64	273 openat	212 close	253 write	169 munmap	194 close	161 openat	171 munmap
7 openat	173 munmap	272 pread64	205 pread64	237 pread64	139 openat	184 epoll_pwait	110 clock_gettime	162 openat
7 clone	165 epoll_pwait	271 munmap	134 writev	204 epoll_pwait	114 rt_sigprocmask	154 fstat64	91 recvfrom	108 rt_sigprocmask
7 close	133 openat	120 close	131 fstat64	142 close	103 close	86 recvfrom	90 close	102 recvfrom
6 timerfd_settim	125 rt_sigprocmask	118 rt_sigprocmask	89 recvfrom	112 rt_sigprocmask	92 recvfrom	74 clock_gettime	88 rt_sigprocmask	87 close
5 getrandom	113 close	116 writev	49 clone	92 recvfrom	68 writev	48 faccessat	63 fstat64	71 fstat64
2 sendto	86 writev	96 madvise	39 faccessat	70 writev	66 fstat64	45 writev	47 faccessat	47 faccessat
	82 fstat64	87 recvfrom	27 getcwd	66 fstat64	44 clone	40 clone	39 fcntl64	41 clone
	71 recvfrom	81 fstat64	26 rt_sigprocmask	44 fcntl64	44 fcntl64	28 sendto	33 writev	41 writev
	69 faccessat	77 faccessat	24 fcntl64	44 faccessat	44 faccessat	26 rt_sigprocmask	31 clone	39 fcntl64
	47 clone	46 clone	19 dup	43 clone	33 readlinkat	26 fcntl64	18 readlinkat	27 sendto
	46 fcntl64	44 fcntl64	14 readlinkat	33 readlinkat	17 lseek	25 getcwd	14 sendto	18 readlinkat
	26 sendto	21 sendto	12 getrandom	24 getcwd	14 sendto	14 dup	14 lseek	12 timerfd_settime
	20 readlinkat	20 readlinkat	12 sendto	15 sendto	13 fstatfs64	13 readlinkat	11 dup	11 dup
	14 lseek	14 lseek	10 lseek	13 fstatfs64	12 getrandom	11 lseek	11 timerfd_settim	11 getrandom
	12 getdents64	12 getcwd	10 rt_sigaction	12 lseek	11 dup	10 getrandom	9 fstatfs64	10 lseek
	12 getrandom	11 dup	9 timerfd_settim	12 getrandom	8 timerfd_settim	9 timerfd_settim	8 getdents64	10 sched_yield
	11 dup	10 getsockopt	7 mremap	11 dup	7 mremap	8 getdents64	7 mkdirat	9 fstatfs64
	11 fstatfs64	10 fstatfs64	7 fstatfs64	7 mkdirat	7 mkdirat	7 mremap	7 getrandom	8 getdents64
	11 timerfd_settim	10 getrandom	6 geteuid32	7 mremap	6 geteuid32	7 fstatfs64	6 getcwd	7 mkdirat
	8 getsockopt	9 sched_yield	6 getegid32	6 timerfd_settim	6 getegid32	7 mkdirat	6 geteuid32	6 getcwd
	8 mkdirat	8 timerfd_settim	6 unlinkat	6 geteuid32	5 getsockopt	7 unlinkat	6 getegid32	6 geteuid32
	7 rt_sigaction	8 mkdirat	5 epoll_ctl	6 getegid32	5 epoll_ctl	6 geteuid32	6 getsockopt	6 getegid32
	6 geteuid32	7 rt_sigaction	5 mkdirat	5 epoll_ctl	4 getdents64	6 getegid32	5 mremap	6 getsockopt
	6 getegid32	6 getdents64	5 getsockopt	5 getsockopt	3 getcwd	6 getsockopt	5 epoll_ctl	5 mremap
	5 uname	6 getegid32	4 getdents64	4 getdents64	2 socketpair	5 epoll_ctl	2 getpriority	5 epoll_ctl
	5 epoll_ctl	6 geteuid32	3 ugetrlimit	2 socketpair	2 eventfd2	4 sched_yield	2 ugetrlimit	2 socketpair
	2 getpriority	5 epoll_ctl	2 getpriority	2 fchmodat	2 fchmodat	2 getpriority	2 fchmodat	2 getpriority
	2 sched_yield	5 uname	2 madvise	2 eventfd2	2 getpriority	2 ugetrlimit	2 eventfd2	2 ugetrlimit
	2 getcwd	3 connect	2 fchmodat	2 ugetrlimit	2 ugetrlimit	2 fchmodat	2 socketpair	2 fchmodat
	2 socketpair	3 socket	2 eventfd2	2 getpriority	1 timerfd_create	2 eventfd2	1 sysinfo	2 eventfd2
	2 fchmodat	2 fchmodat	1 setrlimit	1 sendmsg	1 sysinfo	2 socketpair	1 uname	1 sysinfo
	2 eventfd2	2 sysinfo	1 sysinfo	1 epoll_create1	1 uname	1 sysinfo	1 timerfd_create	1 uname
	1 sysinfo	2 eventfd2	1 sched_yield	1 sysinfo	1 epoll_create1	1 uname	1 epoll_create1	1 timerfd_create
	1 ugetrlimit	2 getpriority	1 sigaltstack	1 setsockopt	1 setsockopt	1 timerfd_create	1 setsockopt	1 epoll_create1
	1 ftruncate64	2 socketpair	1 timerfd_create	1 timerfd_create		1 epoll_create1	1 sendmsg	1 setsockopt
	1 setsockopt	1 sendmsg	1 epoll_create1	1 uname		1 setsockopt		
	1 sendmsg	1 timerfd_create	1 socketpair	1 sched_yield				
	1 timerfd_create	1 ugetrlimit						
	1 epoll_create1	1 epoll_create1						
		1 ftruncate64						
		1 setsockopt						
2287 total	206184 total	224182 total	23004 total	189200 total	9478 total	20044 total	6523 total	7424 total

Figure A3-7. BN4 system call summaries (execution only)

PH1	PH2	PH3	EA1	EA2	EA3	EG1	EG2	EG3
calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall
763 futex	657 clock_gettime	1399 clock_gettime	1133 futex	1165 clock_gettime	283 futex	946 futex	450 futex	372 futex
535 clock_gettime	423 futex	393 futex	318 clock_gettime	228 futex	252 clock_gettime	206 epoll_pwait	220 epoll_pwait	209 epoll_pwait
231 gettimeofday	109 epoll_pwait	246 gettimeofday	230 epoll_pwait	209 gettimeofday	182 epoll_pwait	133 getuid32	170 clock_gettime	171 clock_gettime
118 epoll_pwait	92 ioctl	157 recvfrom	188 getuid32	185 epoll_pwait	98 write	125 write	155 getuid32	127 getuid32
103 ioctl	86 getuid32	153 epoll_pwait	180 ioctl	118 recvfrom	95 getuid32	115 ioctl	100 ioctl	108 write
84 getuid32	54 madvise	153 write	155 write	107 ioctl	92 ioctl	107 recvfrom	91 write	94 ioctl
74 write	51 write	114 getuid32	110 recvfrom	106 getuid32	77 recvfrom	92 read	91 recvfrom	91 read
66 recvfrom	48 recvfrom	112 ioctl	100 read	103 write	74 read	34 clock_gettime	90 read	71 recvfrom
40 read	46 read	45 read	49 prctl	77 madvise	28 madvise	27 prctl	26 madvise	21 prctl
23 fstat64	13 sched_yield	29 sched_yield	32 mmap2	64 read	28 mmap2	22 fstat64	26 rt_sigprocmask	18 rt_sigprocmask
21 prctl	12 writev	18 rt_sigprocmask	31 fstat64	32 mmap2	28 rt_sigprocmask	19 sched_yield	26 mmap2	18 mmap2
21 mmap2	12 rt_sigprocmask	18 mmap2	28 writev	32 rt_sigprocmask	20 prctl	18 mmap2	25 prctl	18 madvise
15 mprotect	11 mmap2	12 writev	23 mprotect	19 prctl	15 mprotect	14 mprotect	15 mprotect	14 writev
14 madvise	11 fstat64	10 mprotect	14 close	16 clone	14 writev	13 writev	13 clone	11 mprotect
8 writev	9 timerfd_settim	9 prctl	12 openat	16 mprotect	14 clone	12 madvise	13 writev	11 timerfd_settime
7 clone	7 mprotect	9 clone	11 sendto	13 writev	7 timerfd_settim	9 timerfd_settim	11 timerfd_settim	9 clone
7 openat	6 clone	8 fstat64	10 clone	5 getrandom	5 getrandom	6 openat	9 sched_yield	9 sendto
7 close	6 openat	4 getrandom	7 timerfd_settim	3 sched_yield	4 sendto	6 clone	9 sendto	8 fstat64
5 timerfd_settim	5 prctl		5 getrandom	3 timerfd_settim	2 sched_yield	6 sendto	8 fstat64	5 sched_yield
4 getrandom	4 gettimeofday		2 dup	1 epoll_ctl		6 close	5 getrandom	5 getrandom
	4 getrandom		1 munmap	1 close		5 getrandom		
	4 close		1 epoll_ctl					
	3 fstatat64							
	2 sendto							
	1 lseek							
2146 total	1676 total	2889 total	2640 total	2503 total	1318 total	1921 total	1553 total	1390 total

Figure A3-8. BN4 system call summaries (50 events injected)

PH1

calls	syscall
20	futex
17	epoll_pwait
7	read
7	getuid
6	ioctl
2	timerfd_settim
1	write
1	clock_gettime
61	total

PH2

calls	syscall
608	newfstatat
489	futex
363	ioctl
362	mprotect
319	mmap
287	prctl
274	getuid
238	pread64
216	read
210	write
166	munmap
140	rt_sigprocmask
115	epoll_pwait
110	clock_gettime
94	openat
90	close
71	fstat
64	faccessat
49	writev
41	madvise
41	lseek
39	fcntl
32	clone
28	sendto
20	readlinkat
19	recvfrom
13	timerfd_settim
12	getsockopt
11	fstatfs
10	dup
8	getdents64
7	rt_sigaction
6	mkdirat
6	mremap
5	uname
5	getrandom
4	sched_yield
4	fsync
4	epoll_ctl
4	unlinkat
4	fchmodat
3	renameat
2	getpriority
2	socketpair
1	ftruncate
1	timerfd_create
1	getrlimit
1	sysinfo
1	setsockopt
1	sendmsg
1	eventfd2
4602	total

PH3

calls	syscall
906	getuid
828	ioctl
748	newfstatat
696	epoll_pwait
567	futex
435	prctl
397	mprotect
393	read
345	mmap
302	pread64
246	rt_sigprocmask
220	write
184	munmap
154	clock_gettime
122	madvise
101	writev
96	openat
94	close
82	faccessat
82	fstat
81	lseek
44	fcntl
40	clone
40	sched_yield
25	sendto
19	readlinkat
16	getsockopt
15	recvfrom
10	fstatfs
10	dup
9	timerfd_settim
8	fsync
8	unlinkat
8	fchmodat
7	rt_sigaction
6	mkdirat
6	renameat
6	mremap
5	epoll_ctl
5	uname
4	getdents64
4	getrandom
2	socketpair
2	eventfd2
2	getpriority
1	timerfd_create
1	sysinfo
1	epoll_create1
1	getrlimit
1	ftruncate
1	sendmsg
1	setsockopt
7387	total

EA1

calls	syscall
2013	clock_gettime
1212	mprotect
484	ioctl
483	futex
464	fstatat64
399	mmap2
320	prctl
307	getuid32
233	pread64
185	write
171	read
170	openat
167	epoll_pwait
166	rt_sigprocmask
158	close
155	munmap
133	fstat64
106	writev
52	faccessat
42	clone
27	fcntl64
21	recvfrom
17	_llseek
16	unlinkat
14	readlinkat
13	getsockopt
12	lseek
12	dup
12	sendto
10	rt_sigaction
9	timerfd_settim
8	fsync
8	fchmodat
7	mremap
7	fstatfs64
6	renameat
5	epoll_ctl
5	getrandom
4	getdents64
3	ugetrlimit
3	mkdirat
2	getpriority
2	eventfd2
1	setrlimit
1	sysinfo
1	sched_yield
1	sigaltstack
1	timerfd_create
1	epoll_create1
1	socketpair
7650	total

EA2

calls	syscall
8135	clock_gettime
759	fstatat64
489	prctl
443	ioctl
401	futex
367	mmap2
337	mprotect
278	getuid32
268	rt_sigprocmask
263	pread64
204	write
202	madvise
170	munmap
166	read
142	epoll_pwait
88	openat
82	close
69	fstat64
67	gettimeofday
62	writev
51	clone
49	faccessat
39	fcntl64
32	readlinkat
23	_llseek
21	recvfrom
15	sendto
14	lseek
13	fstatfs64
11	getsockopt
10	dup
9	getrandom
8	getdents64
7	mremap
6	timerfd_settim
5	unlinkat
5	fsync
5	fchmodat
5	mkdirat
5	epoll_ctl
3	renameat
2	socketpair
2	eventfd2
2	ugetrlimit
2	getpriority
1	sendmsg
1	sysinfo
1	setsockopt
1	epoll_create1
1	uname
1	timerfd_create
13342	total

EA3

calls	syscall
2459	clock_gettime
742	newfstatat
481	prctl
385	ioctl
330	futex
328	mmap
324	mprotect
249	pread64
246	getuid
216	rt_sigprocmask
194	read
177	epoll_pwait
174	write
165	munmap
87	openat
82	close
68	fstat
52	faccessat
50	sched_yield
43	writev
40	lseek
37	fcntl
32	readlinkat
30	clone
21	recvfrom
15	sendto
13	getsockopt
13	madvise
13	fstatfs
10	dup
8	fsync
8	unlinkat
8	fchmodat
7	mremap
7	timerfd_settim
6	renameat
5	mkdirat
5	getrandom
4	epoll_ctl
4	getdents64
2	getrlimit
2	getpriority
2	socketpair
1	sendmsg
1	uname
1	sysinfo
1	timerfd_create
1	eventfd2
1	setsockopt
7150	total

EG1

calls	syscall
1285	mprotect
541	futex
531	fstatat64
504	prctl
502	ioctl
400	mmap2
253	getuid32
234	pread64
206	write
197	read
178	munmap
168	openat
166	rt_sigprocmask
164	epoll_pwait
153	close
146	fstat64
141	madvise
131	clock_gettime
55	writev
49	faccessat
36	clone
29	fcntl64
21	recvfrom
18	_llseek
16	sendto
13	readlinkat
13	unlinkat
13	lseek
12	dup
12	timerfd_settim
10	getsockopt
8	mremap
8	fstatfs64
6	getdents64
5	fchmodat
5	fsync
5	epoll_ctl
5	mkdirat
4	getrandom
3	renameat
2	getpriority
2	sched_yield
2	ugetrlimit
2	eventfd2
2	socketpair
1	restart_syscall
1	sysinfo
1	uname
1	timerfd_create
1	epoll_create1
1	setsockopt
1	sendmsg
6263	total

EG2

calls	syscall
563	fstatat64
444	prctl
422	ioctl
372	futex
306	mmap2
275	mprotect
259	getuid32
245	pread64
242	rt_sigprocmask
204	madvise
170	write
162	clock_gettime
148	munmap
147	read
113	epoll_pwait
74	close
69	openat
67	fstat64
58	faccessat
53	writev
39	fcntl64
38	clone
28	sendto
19	_llseek
17	readlinkat
16	lseek
15	recvfrom
13	getsockopt
10	dup
9	fstatfs64
8	fsync
8	unlinkat
8	fchmodat
8	timerfd_settim
6	getdents64
6	renameat
5	mremap
5	epoll_ctl
5	mkdirat
5	getrandom
2	socketpair
2	getpriority
2	ugetrlimit
2	eventfd2
1	sysinfo
1	uname
1	sched_yield
1	timerfd_create
1	epoll_create1
1	setsockopt
1	sendmsg
4676	total

EG3

calls	syscall
564	fstatat64
444	prctl
422	ioctl
398	futex
304	mmap2
272	mprotect
263	getuid32
245	pread64
238	rt_sigprocmask
211	write
208	madvise
180	read
162	clock_gettime
155	epoll_pwait
147	munmap
75	close
70	openat
68	fstat64
59	faccessat
52	writev
39	fcntl64
36	clone
22	sendto
21	recvfrom
19	_llseek
17	readlinkat
16	lseek
14	getsockopt
11	timerfd_settim
10	dup
9	fsync
9	fstatfs64
9	unlinkat
9	fchmodat
7	renameat
6	getdents64
5	mremap
5	epoll_ctl
5	mkdirat
5	getrandom
2	eventfd2
2	getpriority
2	sched_yield
2	ugetrlimit
2	socketpair
1	sysinfo
1	uname
1	timerfd_create
1	epoll_create1
1	setsockopt
1	sendmsg
4827	total

Figure A3-9. BN5 system call summaries (execution only)

PH1

calls	syscall
43	futex
27	epoll_pwait
12	getuid
11	ioctl
9	read
2	timerfd_settim
1	write
1	clock_gettime
106	total

PH2

calls	syscall
48	futex
46	epoll_pwait
38	read
26	madvise
19	getuid
12	write
11	ioctl
11	openat
11	timerfd_settim
8	close
6	newfstatat
6	fstat
4	sendto
2	lseek
2	writev
2	rt_sigprocmask
1	clock_gettime
1	clone
1	mmap
1	mprotect
256	total

PH3

calls	syscall
235	epoll_pwait
212	getuid
141	ioctl
84	read
34	futex
8	write
6	timerfd_settim
4	writev
1	mprotect
725	total

EA1

calls	syscall
131	epoll_pwait
74	futex
67	read
50	getuid32
47	write
21	clock_gettime
17	ioctl
9	recvfrom
7	timerfd_settim
6	writev
5	sendto
4	prctl
1	mprotect
1	getrandom
440	total

EA2

calls	syscall
1171	clock_gettime
136	epoll_pwait
103	write
76	futex
73	recvfrom
67	read
61	getuid32
32	ioctl
14	gettimeofday
13	madvise
11	prctl
6	timerfd_settim
6	sendto
4	sched_yield
3	writev
2	mprotect
2	mmap2
1	fsync
1	openat
1	faccessat
1	fchmodat
1	unlinkat
1	fstat64
1	getsockopt
1	fstatat64
1	close
1	getrandom
1790	total

EA3

calls	syscall
151	epoll_pwait
120	sched_yield
93	write
74	read
73	recvfrom
61	getuid
50	futex
30	ioctl
17	clock_gettime
12	madvise
9	prctl
7	timerfd_settim
6	sendto
3	writev
1	mprotect
1	getrandom
708	total

EG1

calls	syscall
127	epoll_pwait
95	futex
70	read
59	getuid32
57	write
22	ioctl
15	recvfrom
14	prctl
9	sendto
8	timerfd_settim
3	munmap
3	writev
3	mmap2
3	clock_gettime
2	mprotect
2	madvise
1	fstat64
1	openat
1	getrandom
1	close
496	total

EG2

calls	syscall
122	epoll_pwait
54	getuid32
54	read
29	futex
27	write
26	madvise
21	ioctl
9	prctl
9	recvfrom
7	timerfd_settim
6	mmap2
6	clock_gettime
6	rt_sigprocmask
4	mprotect
3	clone
3	writev
3	sendto
2	faccessat
1	getrandom
392	total

EG3

calls	syscall
141	epoll_pwait
96	getuid32
94	write
79	read
73	recvfrom
59	futex
28	ioctl
28	writev
11	timerfd_settim
9	prctl
8	madvise
7	clock_gettime
6	sendto
1	mprotect
1	getrandom
641	total

Figure A3-10. BN5 system call summaries (50 events injected)

PH1	PH2	PH3	EA1	EA2	EA3	EG1	EG2	EG3
calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall
645 recvfrom	533 recvfrom	352 recvfrom	586 clock_gettime	2973 clock_gettime	384 recvfrom	611 fstatat64	284 recvfrom	380 recvfrom
644 write	346 write	265 write	381 recvfrom	600 recvfrom	308 write	492 recvfrom	177 write	224 write
340 epoll_pwait	203 getuid	168 epoll_pwait	365 write	557 write	215 epoll_pwait	443 write	126 getuid32	145 getuid32
274 futex	191 epoll_pwait	157 getuid	318 fcntl64	300 epoll_pwait	214 clock_gettime	318 fcntl64	119 epoll_pwait	138 epoll_pwait
256 getuid	166 ioctl	138 ioctl	228 epoll_pwait	197 getuid32	142 getuid	266 epoll_pwait	108 ioctl	123 ioctl
219 ioctl	159 futex	104 futex	183 pwrite64	177 futex	134 futex	213 ioctl	91 futex	117 futex
162 read	131 read	88 read	178 ioctl	177 ioctl	122 ioctl	213 faccessat	74 read	86 read
33 fstat	32 madvise	70 madvise	177 getuid32	148 read	96 read	189 getuid32	37 madvise	33 madvise
31 writev	22 newfstatat	36 newfstatat	142 futex	26 madvise	26 madvise	184 futex	15 rt_sigprocmask	15 rt_sigprocmask
16 pread64	20 writev	26 writev	120 fstat64	18 fstatat64	20 rt_sigprocmask	183 pwrite64	12 pread64	12 pread64
15 mmap	16 pread64	15 rt_sigprocmask	100 read	15 rt_sigprocmask	18 newfstatat	123 fstat64	9 mprotect	10 writev
15 mprotect	15 rt_sigprocmask	14 pread64	75 pread64	12 pread64	14 pread64	122 read	9 writev	9 mprotect
11 rt_sigprocmask	12 mmap	11 mmap	67 fstatat64	9 mmap2	10 mmap	75 pread64	9 mmap2	9 mmap2
10 close	9 munmap	8 mprotect	57 close	9 writev	9 writev	73 mkdirat	7 munmap	6 munmap
8 munmap	9 mprotect	7 munmap	48 fdatasync	8 mprotect	8 mprotect	61 close	6 fstatat64	6 fstatat64
6 prctl	4 fstat	4 fstat	47 openat	6 munmap	7 munmap	53 openat	4 fstat64	4 fstat64
5 madvise	4 close	3 clock_gettime	37 mmap2	4 fstat64	4 fstat	48 fdatasync	2 clone	2 clone
4 openat	4 getdents64	2 close	34 writev	2 clone	2 clone	40 mmap2	2 epoll_ctl	2 epoll_ctl
4 getdents64	2 epoll_ctl	2 epoll_ctl	27 faccessat	2 gettimeofday	2 close	40 unlinkat	2 close	2 close
3 dup	2 openat	2 clone	23 munmap	2 epoll_ctl	2 epoll_ctl	29 madvise	1 dup	1 dup
3 fcntl	2 prctl	2 prctl	21 unlinkat	2 close	1 fcntl	26 munmap	1 prctl	1 prctl
2 epoll_ctl	2 clone	1 faccessat	18 geteuid32	1 prctl	1 dup	19 mprotect	1 fcntl64	1 fcntl64
2 clone	1 dup	1 fcntl	17 prctl	1 fcntl64	1 prctl	18 prctl	1096 total	1326 total
1 newfstatat	1 fcntl	1 dup	16 mprotect	1 dup	1740 total	18 geteuid32		
2709 total	1886 total	1477 total	16 rt_sigprocmask	5247 total		17 writev		
			12 ftruncate64			16 rt_sigprocmask		
			8 dup			12 ftruncate64		
			5 epoll_ctl			8 dup		
			4 clone			8 readlinkat		
			4 fchmodat			5 epoll_ctl		
			3 mkdirat			4 clone		
			3 getsockopt			4 getdents64		
			2 lseek			4 fchmodat		
			1 statfs64			3 statfs64		
			1 eventfd2			3 getsockopt		
			1 epoll_create1			2 lseek		
			3325 total			1 fsync		
						1 renameat		
						1 eventfd2		
						1 epoll_create1		
						3947 total		

Figure A3-11. BN6 system call summaries (execution only)

PH1	PH2	PH3	EA1	EA2	EA3	EG1	EG2	EG3
calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall
587 write	313 recvfrom	327 write	551 write	3030 clock_gettime	260 recvfrom	458 recvfrom	184 recvfrom	383 recvfrom
585 recvfrom	288 write	316 recvfrom	545 recvfrom	608 recvfrom	233 write	355 write	140 write	346 write
306 epoll_pwait	162 epoll_pwait	182 epoll_pwait	295 epoll_pwait	599 write	216 clock_gettime	242 epoll_pwait	97 epoll_pwait	198 epoll_pwait
263 futex	145 getuid	144 getuid	261 futex	321 epoll_pwait	152 epoll_pwait	157 getuid32	93 getuid32	152 getuid32
241 getuid	113 futex	125 ioctl	244 clock_gettime	200 getuid32	112 getuid	151 ioctl	74 ioctl	129 futex
202 ioctl	110 ioctl	100 futex	181 getuid32	180 ioctl	94 futex	146 futex	61 futex	127 ioctl
146 read	78 read	79 read	166 ioctl	175 futex	92 ioctl	113 read	42 read	92 read
31 writev	51 madvise	60 madvise	137 read	150 read	65 read	13 mmap2	38 madvise	36 madvise
30 fstat	22 newfstatat	36 newfstatat	31 rt_sigprocmask	24 madvise	36 madvise	10 pread64	15 rt_sigprocmask	15 rt_sigprocmask
26 rt_sigprocmask	19 writev	26 writev	24 pread64	18 fstatat64	21 sched_yield	8 munmap	12 pread64	12 pread64
22 pread64	16 pread64	15 rt_sigprocmask	18 mmap2	15 rt_sigprocmask	20 rt_sigprocmask	8 writev	9 writev	10 writev
17 mmap	15 rt_sigprocmask	14 pread64	16 madvise	12 pread64	18 newfstatat	8 prctl	9 mmap2	9 mmap2
13 mprotect	11 mmap	11 mmap	14 writev	9 mmap2	14 pread64	7 mprotect	8 mprotect	8 mprotect
12 madvise	8 munmap	8 mprotect	12 munmap	9 writev	10 mmap	7 close	7 munmap	7 munmap
11 munmap	8 mprotect	7 munmap	9 mprotect	7 mprotect	9 writev	6 rt_sigprocmask	6 fstatat64	6 fstatat64
8 close	4 close	4 fstat	6 prctl	6 munmap	8 mprotect	6 fstat64	4 fstat64	4 fstat64
6 prctl	4 getdents64	3 clock_gettime	6 fstat64	4 fstat64	7 munmap	6 madvise	2 close	2 epoll_ctl
4 openat	4 fstat	2 clone	6 close	2 clone	4 fstat	4 epoll_ctl	2 clone	2 clone
4 getdents64	2 epoll_ctl	2 prctl	3 epoll_ctl	2 gettimeofday	2 clone	2 dup	2 epoll_ctl	2 close
3 epoll_ctl	2 openat	2 close	2 dup	2 close	2 close	2 clone	1 dup	1 dup
2 dup	2 clone	2 epoll_ctl	2 clone	2 epoll_ctl	2 epoll_ctl	2 fcntl64	1 prctl	1 prctl
2 fcntl	1 dup	1 faccessat	2 fcntl64	1 prctl	1 prctl	2 openat	1 fcntl64	1 fcntl64
2 clone	1 fcntl	1 dup	2 openat	1 dup	1 dup	1713 total	808 total	1543 total
1 newfstatat	1 prctl	1 fcntl	2533 total	1 fcntl64	1 fcntl			
2524 total	1380 total	1468 total		5378 total	1380 total			

Figure A3-12. BN6 system call summaries (50 events injected)

PH1	PH2	PH3	EA1	EA2	EA3	EG1	EG2	EG3
calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall
332 fstat	260 write	278 write	1856 clock_gettime	2973 clock_gettime	1894 clock_gettime	261 recvfrom	211 recvfrom	216 recvfrom
273 ioctl	247 recvfrom	262 recvfrom	275 write	600 recvfrom	246 recvfrom	255 write	166 write	180 write
256 write	156 epoll_pwait	249 getuid	264 recvfrom	557 write	214 write	203 ioctl	144 ioctl	131 ioctl
249 recvfrom	147 ioctl	174 ioctl	181 epoll_pwait	300 epoll_pwait	138 epoll_pwait	175 epoll_pwait	137 epoll_pwait	118 epoll_pwait
241 getuid	145 getuid	167 epoll_pwait	162 ioctl	197 getuid32	138 ioctl	156 futex	125 getuid32	116 pread64
149 epoll_pwait	143 pread64	137 writev	132 getuid32	177 futex	118 getuid	136 getuid32	125 rt_sigprocmask	115 rt_sigprocmask
143 pread64	120 rt_sigprocmask	124 pread64	126 futex	177 ioctl	116 pread64	121 rt_sigprocmask	119 futex	112 getuid32
118 futex	116 fcntl	115 rt_sigprocmask	113 fcntl64	148 read	115 rt_sigprocmask	115 pread64	116 pread64	87 fcntl64
115 fcntl	108 madvise	95 madvise	110 rt_sigprocmask	26 madvise	88 futex	113 fcntl64	87 fcntl64	81 futex
114 rt_sigprocmask	93 futex	92 futex	109 pread64	18 fstatat64	87 fcntl	90 mmap2	86 madvise	72 madvise
72 newfstatat	93 newfstatat	87 fcntl	73 read	15 rt_sigprocmask	78 madvise	80 fstat64	66 read	56 mmap2
69 read	77 fstat	85 newfstatat	70 fstat64	12 pread64	67 newfstatat	71 read	66 mmap2	55 read
63 mmap	64 read	69 read	62 mmap2	9 mmap2	63 read	55 munmap	55 fstatat64	55 fstatat64
55 munmap	58 mmap	58 mmap	52 pwrite64	9 writev	55 mmap	52 pwrite64	48 munmap	49 munmap
40 close	52 munmap	52 munmap	50 munmap	8 mprotect	49 munmap	45 close	39 fstat64	39 fstat64
39 writev	39 close	39 fstat	30 close	6 munmap	39 fstat	35 openat	28 pwrite64	28 pwrite64
34 openat	28 openat	28 pwrite64	24 fstatat64	4 fstat64	28 pwrite64	30 prctl	22 close	22 close
19 pwrite64	21 writev	19 close	23 openat	2 clone	22 close	29 mprotect	19 mprotect	17 mprotect
19 geteuid	19 pwrite64	14 mprotect	19 mprotect	2 gettimeofday	15 fdatasync	26 fstatat64	15 fdatasync	15 fdatasync
16 mprotect	19 geteuid	12 fdatasync	14 writev	2 epoll_ctl	13 mprotect	22 madvise	11 openat	11 openat
11 prctl	13 mprotect	9 faccessat	9 prctl	2 close	11 openat	13 faccessat	10 writev	9 writev
9 fdatasync	9 fdatasync	8 openat	8 fdatasync	1 prctl	9 writev	11 writev	10 prctl	8 faccessat
8 faccessat	8 faccessat	5 fchmodat	8 unlinkat	1 fcntl64	8 faccessat	10 unlinkat	8 clone	5 prctl
6 clock_gettime	6 clock_gettime	4 dup	8 faccessat	1 dup	5 fchmodat	8 fdatasync	8 faccessat	5 fchmodat
5 fchmodat	5 fchmodat	3 clone	7 geteuid32	5247 total	4 dup	8 clone	5 fchmodat	4 dup
4 getdents64	4 dup	3 ftruncate	5 fchmodat		4 prctl	8 dup	4 dup	3 clone
3 dup	4 getdents64	3 prctl	4 dup		3 clone	8 fchmodat	3 ftruncate64	3 ftruncate64
3 clone	3 prctl	3 geteuid	4 fchmod		3 ftruncate	7 geteuid32	3 geteuid32	3 geteuid32
3 madvise	3 clone	3 clock_gettime	4 ftruncate64		3 geteuid	5 epoll_ctl	2 epoll_ctl	2 epoll_ctl
2 epoll_ctl	2 epoll_ctl	2 epoll_ctl	3 clone		2 epoll_ctl	4 fchmod	1 fchmod	1 fchmod
1 mkdirat	1 mkdirat	1 mkdirat	3 epoll_ctl		1 mkdirat	4 ftruncate64	1 fsync	1 fsync
1 unlinkat	1 unlinkat	1 unlinkat	2 madvise		1 fchmod	4 getsockopt	1 statfs64	1 statfs64
1 renameat	1 renameat	1 statfs	1 fsync		1 statfs	3 fsync	1 mkdirat	1 mkdirat
1 statfs	1 statfs	1 getsockopt	1 statfs64		1 getsockopt	2 lseek	1 unlinkat	1 unlinkat
1 fchmod	1 fchmod	1 fchmod	1 mkdirat		1 fsync	2 mkdirat	1 renameat	1 renameat
1 fsync	1 fsync	1 fsync	1 renameat		1 unlinkat	2 renameat	1 getsockopt	1 getsockopt
1 getsockopt	1 getsockopt	1 renameat	1 getsockopt		1 renameat	1 sched_yield	1745 total	1624 total
2477 total	2069 total	2206 total	3815 total		3642 total	1 statfs64		
						1 eventfd2		
						1 epoll_create1		
						2173 total		

Figure A3-13. BN7 system call summaries (execution only)

PH1	PH2	PH3	EA1	EA2	EA3	EG1	EG2	EG3
calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall
332 fstat	255 write	200 write	1808 clock_gettime	3030 clock_gettime	1896 clock_gettime	261 recvfrom	191 recvfrom	178 recvfrom
272 ioctl	246 recvfrom	197 recvfrom	300 write	608 recvfrom	264 write	233 write	177 ioctl	146 write
255 write	157 epoll_pwait	120 epoll_pwait	266 recvfrom	599 write	261 recvfrom	153 epoll_pwait	167 write	121 ioctl
242 recvfrom	146 ioctl	80 ioctl	172 epoll_pwait	321 epoll_pwait	167 epoll_pwait	151 ioctl	140 rt_sigprocmask	116 pread64
239 getuid	146 getuid	71 getuid	156 ioctl	200 getuid32	145 ioctl	132 futex	134 pread64	115 rt_sigprocmask
150 epoll_pwait	143 pread64	64 futex	127 getuid32	180 ioctl	123 getuid	117 getuid32	129 getuid32	104 getuid32
143 pread64	120 rt_sigprocmask	52 read	117 futex	175 futex	116 pread64	113 pread64	122 epoll_pwait	103 epoll_pwait
134 futex	116 fcntl	21 fcntl	112 fcntl64	150 read	115 rt_sigprocmask	112 fcntl64	97 madvise	87 fcntl64
115 fcntl	101 madvise	13 newfstatat	95 pread64	24 madvise	98 futex	109 rt_sigprocmask	92 fcntl64	70 futex
114 rt_sigprocmask	93 futex	9 fstat	79 rt_sigprocmask	18 fstatat64	87 fcntl	71 mmap2	88 futex	70 madvise
72 newfstatat	93 newfstatat	8 pread64	71 read	15 rt_sigprocmask	76 madvise	70 fstat64	82 mmap2	55 fstatat64
66 read	77 fstat	7 madvise	70 fstat64	12 pread64	68 read	67 read	59 fstatat64	55 mmap2
63 mmap	67 read	5 mmap	55 mmap2	9 mmap2	67 newfstatat	58 munmap	56 munmap	49 munmap
55 munmap	58 mmap	5 close	52 pwrite64	9 writev	55 mmap	52 pwrite64	53 read	45 read
40 close	52 munmap	4 writev	43 munmap	7 mprotect	48 munmap	29 close	47 fstat64	39 fstat64
39 writev	39 close	4 rt_sigprocmask	30 close	6 munmap	39 fstat	24 fstatat64	36 close	28 pwrite64
34 openat	28 openat	3 mprotect	24 fstatat64	4 fstat64	28 pwrite64	23 openat	28 pwrite64	22 close
19 pwrite64	21 writev	2 clone	23 openat	2 clone	22 close	16 mprotect	25 mprotect	15 fdatasync
19 geteuid	19 pwrite64	2 faccessat	19 mprotect	2 gettimeofday	15 fdatasync	16 prctl	15 fdatasync	13 mprotect
17 mprotect	19 geteuid	2 prctl	16 writev	2 close	13 mprotect	15 madvise	15 prctl	11 openat
11 prctl	13 mprotect	1 epoll_ctl	9 prctl	2 epoll_ctl	11 openat	8 fdatasync	15 openat	9 writev
9 fdatasync	9 fdatasync	1 openat	8 fdatasync	1 prctl	9 writev	8 unlinkat	14 writev	8 faccessat
8 faccessat	8 faccessat	1 fchmodat	8 unlinkat	1 dup	8 faccessat	8 faccessat	14 faccessat	5 fchmodat
7 madvise	6 clock_gettime	1 munmap	8 faccessat	1 fcntl64	5 fchmodat	7 writev	9 dup	4 dup
6 clock_gettime	5 fchmodat	1 dup	7 geteuid32	5378 total	4 dup	7 geteuid32	8 fchmodat	4 prctl
5 fchmodat	4 dup	874 total	5 fchmodat		4 prctl	5 fchmodat	7 clone	3 clone
4 getdents64	4 getdents64		4 fchmod		3 clone	4 fchmod	4 lseek	3 ftruncate64
3 epoll_ctl	3 prctl		4 ftruncate64		3 ftruncate	4 ftruncate64	4 epoll_ctl	3 geteuid32
3 dup	3 clone		4 epoll_ctl		3 geteuid	3 dup	4 getsockopt	2 epoll_ctl
3 clone	2 epoll_ctl		3 dup		2 epoll_ctl	3 clone	3 fsync	1 fchmod
1 mkdirat	1 mkdirat		3 clone		1 fsync	3 epoll_ctl	3 ftruncate64	1 fsync
1 unlinkat	1 unlinkat		3 madvise		1 renameat	3 clock_gettime	3 geteuid32	1 statfs64
1 renameat	1 renameat		1 fsync		1 unlinkat	1 fsync	3 unlinkat	1 mkdirat
1 statfs	1 statfs		1 statfs64		1 getsockopt	1 statfs64	2 mkdirat	1 unlinkat
1 fchmod	1 fchmod		1 mkdirat		1 fchmod	1 mkdirat	2 renameat	1 renameat
1 fsync	1 fsync		1 renameat		1 statfs	1 renameat	1 fchmod	1 getsockopt
1 getsockopt	1 getsockopt		1 getsockopt		1 mkdirat	1 getsockopt	1 statfs64	1490 total
2486 total	2060 total		3706 total		3762 total	1890 total	1850 total	

Figure A3-14. BN7 system call summaries (50 events injected)

PH1

calls	syscall
31	futex
25	ioctl
23	epoll_pwait
18	getuid
9	read
9	fstat
9	clock_gettime
5	write
4	recvfrom
3	timerfd_settim
3	prctl
3	mmap
2	mprotect
1	openat
1	close
1	clone
147	total

PH2

calls	syscall
147	ioctl
110	pread64
98	getuid
85	madvise
83	futex
68	rt_sigprocmask
67	epoll_pwait
58	mmap
55	munmap
46	write
43	read
42	newfstatat
38	fstat
36	fcntl
32	close
30	clock_gettime
17	openat
12	writev
12	recvfrom
12	mprotect
8	timerfd_settime
6	dup
6	geteuid
5	prctl
2	lseek
2	clone
1	faccessat
1121	total

PH3

calls	syscall
41	epoll_pwait
38	ioctl
31	futex
28	newfstatat
28	getuid
25	write
21	read
9	clock_gettime
7	prctl
4	timerfd_settim
4	recvfrom
4	fstat
2	mmap
2	rt_sigprocmask
2	mprotect
1	writev
1	clone
248	total

EA1

calls	syscall
3693	clock_gettime
578	mprotect
409	ioctl
296	mmap2
291	futex
269	pread64
216	prctl
198	rt_sigprocmask
172	getuid32
155	munmap
147	fstat64
130	write
122	fstatat64
101	openat
95	close
89	read
84	epoll_pwait
75	fcntl64
32	clone
30	pwrite64
26	writev
26	faccessat
19	recvfrom
12	getsockopt
11	rt_sigaction
10	dup
10	unlinkat
10	fchmodat
9	_llseek
8	fsync
6	fdatasync
6	epoll_ctl
5	renameat
5	readlinkat
4	getdents64
4	mkdirat
3	ugetrlimit
3	geteuid32
3	timerfd_settime
3	getrandom
2	getpriority
2	mremap
2	fstatfs64
2	eventfd2
1	lseek
1	setrlimit
1	sysinfo
1	sched_yield
1	sigaltstack
1	ftruncate64
1	statfs64
1	timerfd_create
1	epoll_create1
7382	total

EA2

calls	syscall
232	clock_gettime
28	ioctl
26	futex
24	epoll_pwait
24	write
18	getuid32
9	read
8	gettimeofday
8	fstatat64
7	prctl
4	fstat64
4	recvfrom
2	mmap2
2	madvise
2	rt_sigprocmask
2	mprotect
1	clone
1	timerfd_settim
402	total

EA3

calls	syscall
79	clock_gettime
27	futex
27	ioctl
25	epoll_pwait
19	write
18	getuid
10	read
8	newfstatat
7	prctl
4	recvfrom
4	fstat
2	rt_sigprocmask
2	mmap
2	mprotect
1	madvise
1	clone
1	timerfd_settim
237	total

EG1

calls	syscall
612	mprotect
396	ioctl
346	prctl
320	futex
269	pread64
267	mmap2
218	rt_sigprocmask
164	munmap
150	getuid32
140	fstatat64
138	fstat64
131	write
99	read
86	openat
77	close
75	madvise
74	fcntl64
67	epoll_pwait
58	clock_gettime
30	pwrite64
25	faccessat
22	clone
19	recvfrom
15	writev
10	getsockopt
9	dup
8	unlinkat
8	fchmodat
7	_llseek
6	fsync
6	fdatasync
6	epoll_ctl
6	mkdirat
5	renameat
5	readlinkat
4	getdents64
3	geteuid32
3	getrandom
2	getpriority
2	sched_yield
2	mremap
2	ugetrlimit
2	fstatfs64
2	timerfd_settime
2	eventfd2
1	lseek
1	sysinfo
1	uname
1	rt_sigaction
1	ftruncate64
1	statfs64
1	timerfd_create
1	epoll_create1
1	socketpair
1	setsockopt
3908	total

EG2

calls	syscall
59	getuid32
50	epoll_pwait
49	ioctl
40	futex
34	write
27	rt_sigprocmask
20	pread64
16	madvise
13	mmap2
12	recvfrom
10	munmap
9	clock_gettime
9	read
8	prctl
6	fstatat64
4	fstat64
3	mprotect
1	timerfd_settim
1	clone
1	writev
1	sched_yield
373	total

EG3

calls	syscall
119	ioctl
100	pread64
68	madvise
66	getuid32
61	rt_sigprocmask
56	mmap2
53	munmap
46	epoll_pwait
42	write
41	futex
30	fcntl64
20	fstat64
18	fstatat64
18	close
12	recvfrom
9	mprotect
9	clock_gettime
9	read
8	prctl
6	dup
2	writev
1	clone
1	faccessat
1	timerfd_settim
796	total

Figure A3-15. BN8 system call summaries (execution only)

PH1

calls	syscall
41	futex
23	madvise
19	ioctl
18	epoll_pwait
18	write
13	getuid
10	read
9	clock_gettime
3	timerfd_settim
3	prctl
3	mmap
3	mprotect
2	writev
1	openat
1	close
1	fstat
1	clone
169	total

PH2

calls	syscall
77	getuid
67	ioctl
63	epoll_pwait
62	futex
36	write
35	madvise
31	read
30	clock_gettime
29	rt_sigprocmask
21	newfstatat
20	pread64
13	mmap
12	recvfrom
10	munmap
7	fstat
6	openat
6	timerfd_settim
5	writev
5	prctl
5	mprotect
4	close
2	clone
1	lseek
547	total

PH3

calls	syscall
31	epoll_pwait
27	ioctl
25	futex
23	getuid
17	write
15	read
9	clock_gettime
7	prctl
4	timerfd_settim
2	mprotect
2	rt_sigprocmask
2	mmap
2	madvise
1	sched_yield
1	clone
168	total

EA1

calls	syscall
37	clock_gettime
23	epoll_pwait
22	ioctl
18	getuid32
15	futex
8	read
4	prctl
3	mmap2
2	write
2	mprotect
2	writev
1	close
1	clone
1	fstat64
1	openat
1	timerfd_settim
141	total

EA2

calls	syscall
237	clock_gettime
40	futex
27	ioctl
23	epoll_pwait
18	getuid32
18	write
8	read
8	fstatat64
8	gettimeofday
7	prctl
4	fstat64
4	recvfrom
2	mmap2
2	madvise
2	mprotect
2	rt_sigprocmask
1	timerfd_settim
1	clone
412	total

EA3

calls	syscall
79	clock_gettime
28	ioctl
24	epoll_pwait
24	write
22	futex
18	getuid
9	read
8	newfstatat
7	prctl
4	fstat
4	recvfrom
2	mprotect
2	mmap
2	rt_sigprocmask
1	sched_yield
1	timerfd_settim
1	clone
236	total

EG1

calls	syscall
42	futex
19	epoll_pwait
15	ioctl
13	write
13	getuid32
9	prctl
9	clock_gettime
8	read
3	mprotect
3	mmap2
3	madvise
1	close
1	clone
1	writev
1	fstat64
1	openat
1	timerfd_settim
143	total

EG2

calls	syscall
28	futex
25	epoll_pwait
25	write
22	ioctl
22	getuid32
10	read
9	clock_gettime
7	prctl
4	fstat64
3	fstatat64
2	mprotect
2	rt_sigprocmask
2	mmap2
2	madvise
1	clone
1	writev
1	timerfd_settim
4	recvfrom
170	total

EG3

calls	syscall
33	futex
24	write
21	ioctl
20	epoll_pwait
17	getuid32
9	clock_gettime
8	read
7	prctl
4	fstat64
4	recvfrom
3	fstatat64
2	mprotect
2	rt_sigprocmask
2	mmap2
2	madvise
1	clone
1	timerfd_settim
160	total

Figure A3-16. BN8 system call summaries (50 events injected)

PH1	PH2	PH3	EA1	EA2	EA3	EG1	EG2	EG3
calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall
515 recvfrom	828 madvise	1591 madvise	1134 recvfrom	4195 clock_gettime	3711 clock_gettime	774 recvfrom	1057 madvise	1142 madvise
385 write	239 recvfrom	435 recvfrom	938 write	2216 madvise	1840 madvise	534 write	274 recvfrom	306 recvfrom
192 futex	198 futex	337 write	445 epoll_pwait	618 recvfrom	510 recvfrom	397 futex	216 write	239 write
134 ioctl	132 write	288 futex	442 futex	474 write	393 write	266 ioctl	199 futex	233 futex
131 epoll_pwait	108 getuid	171 ioctl	357 ioctl	349 futex	303 futex	258 epoll_pwait	114 getuid32	123 getuid32
131 getuid	96 ioctl	163 getuid	329 getuid32	200 ioctl	173 ioctl	227 getuid32	113 ioctl	121 ioctl
129 read	76 epoll_pwait	126 epoll_pwait	284 read	199 getuid32	172 getuid	192 read	86 epoll_pwait	94 epoll_pwait
7 fstat	24 mprotect	104 read	158 clock_gettime	172 epoll_pwait	145 epoll_pwait	44 mmap2	64 read	71 read
2 close	13 writev	74 mprotect	60 fstatat64	152 read	123 read	37 fstatat64	37 fstatat64	39 mprotect
2 mprotect	12 prctl	39 mmap	33 mmap2	60 fstatat64	60 newfstatat	35 madvise	32 mprotect	37 fstatat64
1 dup	11 close	37 newfstatat	28 prctl	52 mprotect	52 mprotect	34 prctl	29 mmap2	34 mmap2
1 fcntl	11 rt_sigprocmas	30 prctl	27 close	26 mmap2	25 mmap	27 close	25 prctl	25 prctl
1 mmap	11 mmap	23 rt_sigprocmas	26 fstat64	24 faccessat	24 faccessat	26 fstat64	20 rt_sigprocmas	20 rt_sigprocmas
1631 total	8 fstat	21 writev	24 writev	24 prctl	23 prctl	23 mprotect	17 faccessat	17 faccessat
	8 getrandom	19 faccessat	24 faccessat	15 rt_sigprocmas	15 rt_sigprocmas	22 openat	14 writev	16 pread64
	5 openat	15 munmap	22 openat	13 writev	13 writev	18 munmap	14 close	15 writev
	4 dup	15 close	21 mprotect	13 close	13 close	17 faccessat	10 fstat64	14 close
	4 fcntl	14 openat	15 madvise	10 fstat64	10 fstat	14 writev	8 openat	10 fstat64
	4 faccessat	12 fstat	7 mkdirat	8 openat	8 openat	13 pread64	6 pread64	8 openat
	4 getsockopt	9 clone	6 dup	7 mkdirat	7 mkdirat	10 rt_sigprocmas	5 clone	7 munmap
	3 unlinkat	6 clock_gettime	6 clone	6 gettimeofday	6 gettimeofday	6 dup	5 getrandom	5 clone
	3 fchmodat	5 mkdirat	5 munmap	5 clone	5 getsockopt	6 clone	5 getsockopt	5 getrandom
	3 newfstatat	5 getrandom	5 rt_sigprocmas	5 getrandom	5 getrandom	5 getrandom	4 lseek	5 getsockopt
	3 fsync	5 pread64	5 getrandom	5 getsockopt	5 clone	5 getsockopt	4 dup	4 lseek
	3 clone	5 getsockopt	5 getsockopt	4 unlinkat	4 dup	4 mkdirat	4 fcntl64	4 dup
	2 epoll_ctl	5 fcntl	4 pread64	4 fcntl64	4 unlinkat	4 unlinkat	4 mkdirat	4 fcntl64
	2 lseek	4 unlinkat	4 unlinkat	4 pread64	4 fcntl	3 fsync	4 unlinkat	4 mkdirat
	2 pread64	4 dup	3 fsync	4 dup	4 pread64	3 epoll_ctl	3 fsync	4 unlinkat
	1 renameat	3 fsync	3 epoll_ctl	3 fsync	3 fsync	3 fchmodat	3 fchmodat	3 fsync
	1 read	3 fchmodat	3 fchmodat	3 fchmodat	3 fchmodat	2 lseek	2 munmap	3 fchmodat
	1819 total	2 renameat	2 lseek	2 getdents64	2 epoll_ctl	2 getdents64	2 getdents64	2 getdents64
		2 getdents64	2 getdents64	2 renameat	2 renameat	2 fcntl64	2 epoll_ctl	2 epoll_ctl
		2 epoll_ctl	2 fcntl64	2 munmap	2 lseek	2 renameat	2 renameat	2 renameat
		2 lseek	2 renameat	2 lseek	2 munmap	1 statfs64	1 statfs64	1 statfs64
		1 statfs	1 statfs64	2 epoll_ctl	2 getdents64	1 eventfd2	2385 total	2619 total
		1 readlinkat	1 eventfd2	1 statfs64	1 statfs	1 epoll_create1		
		1 mremap	1 epoll_create1	8881 total	7674 total	3018 total		
		1 fstatfs	4434 total					
		3580 total						

Figure A3-17. MW1 system call summaries (execution only)

PH1	PH2	PH3	EA1	EA2	EA3	EG1	EG2	EG3
calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall
1462 recvfrom	862 madvise	1407 madvise	1972 recvfrom	5746 clock_gettime	2801 clock_gettime	1543 recvfrom	1555 recvfrom	737 recvfrom
978 epoll_pwait	238 recvfrom	1286 recvfrom	1381 write	1201 recvfrom	1828 madvise	803 write	1173 madvise	632 madvise
527 getuid	197 futex	756 write	803 epoll_pwait	1196 madvise	494 recvfrom	730 epoll_pwait	929 epoll_pwait	401 epoll_pwait
492 ioctl	137 write	665 epoll_pwait	552 futex	656 write	381 write	511 getuid32	655 write	349 write
491 write	126 getuid	428 getuid	546 ioctl	616 epoll_pwait	302 futex	488 ioctl	549 getuid32	224 getuid32
484 read	101 ioctl	419 ioctl	545 getuid32	350 getuid32	166 ioctl	436 read	527 ioctl	223 ioctl
18 futex	81 epoll_pwait	365 read	501 read	350 ioctl	166 getuid	395 futex	487 read	218 read
16 writev	31 mprotect	310 futex	246 clock_gettime	345 read	141 epoll_pwait	40 writev	171 futex	168 futex
3 clock_gettime	19 writev	38 newfstatat	39 rt_sigprocmas	237 futex	120 read	23 mmap2	20 pread64	11 mprotect
3 process_vm_read	14 prctl	33 mprotect	30 pread64	17 mprotect	50 mprotect	20 madvise	18 mmap2	2 fstat64
2 epoll_ctl	14 mmap	25 writev	24 mmap2	2 fstat64	22 mmap	19 rt_sigprocmas	17 writev	1 writev
2 close	13 rt_sigprocmas	21 rt_sigprocmas	18 fstat64	10716 total	21 prctl	19 fstat64	15 mprotect	2966 total
1 rt_sigprocmask	11 close	20 pread64	16 writev		15 rt_sigprocmas	18 pread64	14 rt_sigprocmas	
1 rt_sigreturn	11 getrandom	11 mmap	15 munmap		14 faccessat	15 prctl	10 fstat64	
1 mprotect	10 fstat	10 munmap	13 close		13 close	14 close	9 munmap	
4481 total	5 openat	8 fstat	10 prctl		12 writev	12 munmap	8 fstatat64	
	4 clone	7 prctl	10 openat		10 fstat	12 sendto	8 faccessat	
	4 dup	5 faccessat	8 madvise		9 newfstatat	11 openat	6 prctl	
	4 fcntl	5 close	5 mprotect		8 openat	6 mprotect	5 close	
	4 faccessat	4 getrandom	5 getrandom		6 gettimeofday	5 getrandom	4 openat	
	4 getsockopt	3 clock_gettime	4 faccessat		5 getsockopt	4 faccessat	3 getsockopt	
	3 unlinkat	2 openat	3 fsync		5 clone	3 fsync	2 fsync	
	3 fchmodat	2 fsync	3 fstatat64		5 getrandom	3 fstatat64	2 clone	
	3 newfstatat	2 unlinkat	3 unlinkat		4 unlinkat	3 unlinkat	2 epoll_ctl	
	3 fsync	2 fchmodat	3 fchmodat		4 dup	3 fchmodat	2 unlinkat	
	2 epoll_ctl	2 renameat	3 getsockopt		4 fcntl	3 getsockopt	2 renameat	
	2 lseek	2 getsockopt	2 dup		3 fsync	2 dup	2 fchmodat	
	2 pread64	2 fcntl	2 fcntl64		3 mkdirat	2 clone	2 getrandom	
	1 renameat	2 dup	2 epoll_ctl		3 fchmodat	2 fcntl64	1 dup	
	1 read	2 epoll_ctl	1 clone		2 renameat	2 epoll_ctl	1 fcntl64	
	1910 total	1 clone	1 renameat		2 pread64	1 renameat	6199 total	
		5845 total	6766 total		2 getdents64	5148 total		
					2 epoll_ctl			
					2 lseek			
					1 statfs			
					6626 total			

Figure A3-18. MW1 system call summaries (50 events injected)

PH1

calls	syscall
3249	clock_gettime
984	mprotect
567	ioctl
451	futex
326	getuid32
294	write
285	read
230	epoll_pwait
216	prctl
181	gettimeofday
170	mmap2
119	fstat64
103	fstatat64
93	madvise
73	openat
64	close
59	writev
59	pread64
55	munmap
41	sendto
32	faccessat
27	clone
22	recvfrom
9	fcntl64
9	timerfd_settime
8	getdents64
7	dup
7	_llseek
7	rt_sigprocmask
7	mkdirat
7	readlinkat
4	epoll_ctl
4	getsockopt
3	fstatfs64
2	getpriority
2	ugetrlimit
2	socketpair
2	fchmodat
2	getrandom
1	lseek
1	sysinfo
1	uname
1	rt_sigaction
1	socket
1	connect
1	setsockopt
1	sendmsg
1	unlinkat
1	timerfd_create
1	eventfd2
7793	total

PH2

calls	syscall
3153	clock_gettime
437	futex
355	ioctl
275	write
242	read
240	fstatat64
233	getuid32
190	prctl
186	mprotect
182	mmap2
147	madvise
130	epoll_pwait
92	pread64
84	munmap
80	gettimeofday
65	rt_sigprocmask
65	openat
60	fstat64
58	sendto
57	close
46	faccessat
42	writev
29	clone
22	recvfrom
15	_llseek
15	fcntl64
14	timerfd_settime
12	readlinkat
9	dup
9	rt_sigaction
8	getdents64
8	mkdirat
7	getsockopt
6	fstatfs64
5	uname
4	lseek
4	epoll_ctl
2	getpriority
2	socketpair
2	fchmodat
2	getrandom
1	sysinfo
1	sched_yield
1	ugetrlimit
1	ftruncate64
1	socket
1	connect
1	setsockopt
1	sendmsg
1	timerfd_create
1	eventfd2
6604	total

PH3

calls	syscall
7795	clock_gettime
1036	getuid32
1004	gettimeofday
994	epoll_pwait
957	ioctl
591	futex
535	read
364	prctl
296	fstatat64
295	write
187	mprotect
173	mmap2
92	pread64
83	madvise
79	munmap
60	rt_sigprocmask
58	openat
57	fstat64
53	writev
51	close
49	sendto
48	faccessat
32	recvfrom
29	clone
17	fcntl64
15	_llseek
12	readlinkat
10	timerfd_settime
9	dup
9	rt_sigaction
8	mkdirat
7	getsockopt
6	fstatfs64
5	uname
4	getdents64
4	epoll_ctl
3	lseek
2	fchmodat
2	getpriority
2	socketpair
2	getrandom
1	eventfd2
1	sysinfo
1	socket
1	connect
1	sched_yield
1	timerfd_create
1	ugetrlimit
1	ftruncate64
1	setsockopt
15044	total

EA1

calls	syscall
1012	clock_gettime
652	mprotect
276	ioctl
209	prctl
182	futex
169	mmap2
130	getuid32
113	write
96	fstatat64
96	read
68	fstat64
64	openat
61	close
55	pread64
51	epoll_pwait
46	munmap
34	writev
31	clone
23	recvfrom
14	faccessat
11	rt_sigaction
8	dup
8	fcntl64
7	_llseek
6	readlinkat
5	mkdirat
5	getsockopt
4	getdents64
4	epoll_ctl
3	mremap
3	ugetrlimit
3	fstatfs64
2	getpriority
2	fchmodat
2	timerfd_settime
2	getrandom
1	lseek
1	setrlimit
1	sysinfo
1	sigaltstack
1	unlinkat
1	timerfd_create
1	eventfd2
3464	total

EA2

calls	syscall
3001	clock_gettime
369	prctl
278	fstatat64
248	futex
230	ioctl
162	mprotect
160	mmap2
128	write
124	read
107	getuid32
93	madvise
76	pread64
68	munmap
54	rt_sigprocmask
50	openat
38	close
36	fstat64
26	epoll_pwait
26	clone
21	writev
19	faccessat
15	gettimeofday
14	fcntl64
13	readlinkat
12	_llseek
7	mkdirat
6	dup
6	fstatfs64
5	recvfrom
4	getsockopt
4	getdents64
4	mremap
3	epoll_ctl
2	socketpair
2	rt_sigaction
2	fchmodat
2	lseek
2	getpriority
2	ugetrlimit
2	getrandom
1	timerfd_settime
1	eventfd2
1	sysinfo
1	timerfd_create
1	uname
1	setsockopt
5427	total

EA3

calls	syscall
1058	clock_gettime
367	prctl
272	fstatat64
256	futex
230	ioctl
162	mprotect
159	mmap2
132	read
120	write
107	getuid32
92	madvise
78	pread64
68	munmap
57	openat
54	rt_sigprocmask
42	close
36	fstat64
26	clone
26	epoll_pwait
21	writev
19	faccessat
14	fcntl64
13	readlinkat
12	_llseek
7	mkdirat
7	lseek
6	dup
6	fstatfs64
5	recvfrom
4	getsockopt
4	mremap
4	getdents64
3	epoll_ctl
2	socketpair
2	fchmodat
2	ugetrlimit
2	getpriority
2	rt_sigaction
2	getrandom
1	timerfd_settime
1	eventfd2
1	sysinfo
1	timerfd_create
1	setsockopt
1	uname
3485	total

EG1

calls	syscall
694	mprotect
360	prctl
292	ioctl
261	futex
160	mmap2
140	write
125	getuid32
124	fstatat64
111	read
81	madvise
76	fstat64
63	clock_gettime
63	openat
58	close
53	pread64
50	munmap
48	epoll_pwait
26	clone
23	writev
23	recvfrom
18	faccessat
10	dup
8	fcntl64
7	mkdirat
7	_llseek
6	epoll_ctl
6	readlinkat
5	getsockopt
4	getdents64
3	lseek
3	mremap
3	fstatfs64
2	getpriority
2	ugetrlimit
2	fchmodat
2	eventfd2
2	getrandom
1	sysinfo
1	uname
1	sched_yield
1	rt_sigaction
1	unlinkat
1	timerfd_create
1	timerfd_settime
1	epoll_create1
1	socketpair
1	setsockopt
2931	total

EG2

calls	syscall
348	prctl
255	futex
223	ioctl
154	write
151	fstatat64
132	read
130	mmap2
126	mprotect
116	getuid32
75	madvise
60	clock_gettime
59	pread64
55	munmap
50	rt_sigprocmask
40	epoll_pwait
36	fstat64
34	openat
32	close
24	clone
20	writev
18	faccessat
14	fcntl64
9	readlinkat
9	recvfrom
8	_llseek
7	dup
7	mkdirat
4	getdents64
4	fstatfs64
4	getsockopt
3	sched_yield
3	mremap
3	epoll_ctl
2	lseek
2	getpriority
2	rt_sigaction
2	ugetrlimit
2	fchmodat
2	getrandom
2	socketpair
2	sendto
1	sysinfo
1	uname
1	timerfd_create
1	timerfd_settime
1	eventfd2
1	setsockopt
2235	total

EG3

calls	syscall
348	prctl
250	futex
220	ioctl
151	fstatat64
145	write
130	mmap2
130	read
128	mprotect
112	getuid32
76	madvise
60	clock_gettime
59	pread64
55	munmap
50	rt_sigprocmask
35	epoll_pwait
34	openat
34	fstat64
32	close
24	clone
20	writev
18	faccessat
14	fcntl64
9	readlinkat
9	recvfrom
8	_llseek
7	dup
7	mkdirat
4	getdents64
4	fstatfs64
4	getsockopt
3	sched_yield
3	mremap
3	epoll_ctl
2	lseek
2	getpriority
2	rt_sigaction
2	ugetrlimit
2	fchmodat
2	getrandom
2	socketpair
1	sysinfo
1	uname
1	timerfd_create
1	timerfd_settime
1	eventfd2
1	setsockopt
2206	total

Figure A3-19. MW2 system call summaries (execution only)

PH1

calls	syscall
3365	clock_gettime
986	mprotect
600	ioctl
435	futex
338	getuid32
302	write
295	read
251	epoll_pwait
220	prctl
207	gettimeofday
172	mmap2
117	fstat64
103	fstatat64
92	madvise
74	openat
63	close
59	pread64
58	writev
55	munmap
45	sendto
32	faccessat
28	clone
21	recvfrom
9	timerfd_settime
8	getdents64
8	fcntl64
7	_llseek
7	rt_sigprocmask
7	mkdirat
7	readlinkat
6	dup
5	epoll_ctl
4	getsockopt
3	fstatfs64
2	socketpair
2	getpriority
2	ugetrlimit
2	fchmodat
2	getrandom
1	lseek
1	sysinfo
1	uname
1	rt_sigaction
1	socket
1	connect
1	setsockopt
1	sendmsg
1	unlinkat
1	timerfd_create
1	eventfd2
8010	total

PH2

calls	syscall
3412	clock_gettime
488	futex
342	ioctl
284	write
267	read
249	fstatat64
226	getuid32
190	prctl
186	mprotect
182	mmap2
170	madvise
133	epoll_pwait
92	pread64
84	munmap
80	openat
70	gettimeofday
69	close
69	fstat64
65	sendto
65	rt_sigprocmask
46	faccessat
40	writev
29	clone
21	recvfrom
18	timerfd_settime
15	_llseek
15	fcntl64
12	readlinkat
9	dup
9	rt_sigaction
8	getdents64
8	mkdirat
7	lseek
7	getsockopt
6	fstatfs64
5	uname
4	epoll_ctl
3	sched_yield
2	getpriority
2	socketpair
2	fchmodat
2	getrandom
1	sysinfo
1	ugetrlimit
1	ftruncate64
1	socket
1	connect
1	setsockopt
1	sendmsg
1	timerfd_create
1	eventfd2
7002	total

PH3

calls	syscall
3522	clock_gettime
581	epoll_pwait
541	gettimeofday
495	getuid32
449	ioctl
261	read
85	write
84	futex
40	madvise
37	sendto
22	recvfrom
12	prctl
12	timerfd_settime
9	writev
4	faccessat
4	mprotect
2	close
1	fstat64
1	mmap2
1	munmap
1	epoll_ctl
6164	total

EA1

calls	syscall
831	clock_gettime
654	mprotect
299	ioctl
258	futex
224	prctl
193	getuid32
176	write
173	mmap2
126	read
96	fstatat64
76	fstat64
75	epoll_pwait
68	writev
66	openat
56	close
55	pread64
46	munmap
33	clone
24	recvfrom
14	faccessat
12	sendto
11	rt_sigaction
7	readlinkat
7	_llseek
7	fcntl64
5	mkdirat
4	dup
4	getdents64
4	getsockopt
3	mremap
3	ugetrlimit
3	epoll_ctl
3	fstatfs64
3	timerfd_settime
2	getpriority
2	rt_sigprocmask
2	fchmodat
2	getrandom
1	lseek
1	setrlimit
1	sysinfo
1	sigaltstack
1	unlinkat
1	timerfd_create
1	eventfd2
1	socketpair
3635	total

EA2

calls	syscall
3986	clock_gettime
386	prctl
300	futex
287	ioctl
278	fstatat64
188	write
165	mprotect
164	mmap2
161	getuid32
144	read
104	madvise
76	pread64
68	munmap
64	epoll_pwait
58	rt_sigprocmask
50	openat
45	fstat64
38	close
30	writev
28	clone
22	gettimeofday
19	faccessat
17	recvfrom
14	fcntl64
13	readlinkat
12	_llseek
9	sendto
7	mkdirat
7	dup
6	fstatfs64
4	mremap
4	getsockopt
4	getdents64
3	timerfd_settime
3	epoll_ctl
2	socketpair
2	rt_sigaction
2	fchmodat
2	lseek
2	getpriority
2	ugetrlimit
2	getrandom
1	sysinfo
1	setsockopt
1	uname
1	timerfd_create
1	eventfd2
6783	total

EA3

calls	syscall
1362	clock_gettime
381	prctl
287	ioctl
272	futex
272	fstatat64
180	write
165	mprotect
164	read
163	mmap2
160	getuid32
105	madvise
79	epoll_pwait
78	pread64
68	munmap
58	rt_sigprocmask
57	openat
45	fstat64
42	close
30	writev
28	clone
20	recvfrom
19	faccessat
14	fcntl64
13	readlinkat
12	_llseek
9	sendto
7	mkdirat
7	dup
7	lseek
6	fstatfs64
4	mremap
4	getdents64
4	getsockopt
3	timerfd_settime
3	epoll_ctl
2	ugetrlimit
2	rt_sigaction
2	fchmodat
2	getrandom
2	getpriority
2	socketpair
1	eventfd2
1	timerfd_create
1	uname
1	sysinfo
1	setsockopt
4145	total

EG1

calls	syscall
689	mprotect
387	prctl
316	ioctl
306	futex
187	write
159	mmap2
151	read
142	getuid32
124	fstatat64
90	fstat64
81	madvise
70	epoll_pwait
62	openat
60	clock_gettime
53	pread64
50	munmap
50	close
27	writev
26	clone
18	recvfrom
18	faccessat
8	fcntl64
7	mkdirat
7	_llseek
7	readlinkat
5	dup
5	sendto
4	getsockopt
4	getdents64
3	mremap
3	epoll_ctl
3	fstatfs64
2	getpriority
2	rt_sigprocmask
2	ugetrlimit
2	fchmodat
2	timerfd_settime
2	getrandom
2	socketpair
1	lseek
1	sysinfo
1	uname
1	rt_sigaction
1	unlinkat
1	timerfd_create
1	eventfd2
1	setsockopt
3144	total

EG2

calls	syscall
362	prctl
302	futex
266	ioctl
204	write
155	getuid32
151	fstatat64
145	read
137	mmap2
130	mprotect
83	madvise
64	epoll_pwait
60	clock_gettime
59	pread64
55	munmap
54	rt_sigprocmask
42	fstat64
34	openat
32	close
29	writev
26	clone
20	recvfrom
18	faccessat
14	fcntl64
9	readlinkat
9	sendto
8	_llseek
7	mkdirat
7	dup
4	getdents64
4	fstatfs64
4	getsockopt
3	timerfd_settime
3	mremap
3	epoll_ctl
2	lseek
2	getpriority
2	rt_sigaction
2	ugetrlimit
2	fchmodat
2	getrandom
2	socketpair
1	sysinfo
1	uname
1	sched_yield
1	timerfd_create
1	eventfd2
1	setsockopt
2523	total

EG3

calls	syscall
359	prctl
301	futex
266	ioctl
192	write
156	getuid32
151	fstatat64
146	read
134	mmap2
129	mprotect
81	madvise
61	epoll_pwait
60	clock_gettime
59	pread64
56	munmap
54	rt_sigprocmask
42	fstat64
34	openat
33	close
30	writev
26	clone
20	recvfrom
18	faccessat
14	fcntl64
14	sendto
9	readlinkat
8	_llseek
7	dup
7	mkdirat
4	getdents64
4	fstatfs64
4	getsockopt
3	mremap
3	epoll_ctl
3	timerfd_settime
2	lseek
2	getpriority
2	rt_sigaction
2	ugetrlimit
2	fchmodat
2	getrandom
2	socketpair
1	sysinfo
1	uname
1	timerfd_create
1	eventfd2
1	setsockopt
1	sendmsg
2508	total

Figure A3-20. MW2 system call summaries (50 events injected)

PH1

calls	syscall
807	epoll_pwait
537	ioctl
484	read
447	write
392	getuid
332	recvfrom
292	futex
177	sendto
16	mmap
14	prctl
11	timerfd_settim
10	mprotect
8	fstat
8	munmap
4	close
3	openat
3	clone
3	madvise
1	writev
3549	total

PH2

calls	syscall
884	prctl
857	epoll_pwait
759	ioctl
572	futex
551	getuid
473	read
448	write
336	recvfrom
139	sendto
109	madvise
104	mprotect
101	mmap
77	newfstatat
64	pread64
55	rt_sigprocmask
54	munmap
43	close
39	fstat
38	clock_gettime
33	openat
32	faccessat
31	writev
24	clone
13	timerfd_settim
13	fcntl
10	dup
8	lseek
7	rt_sigaction
7	getsockopt
6	mkdirat
5	uname
4	getdents64
4	readlinkat
4	epoll_ctl
2	getpriority
2	socketpair
1	ftruncate
1	timerfd_create
1	getrlimit
1	sysinfo
1	setsockopt
1	mremap
1	getrandom
1	eventfd2
5916	total

PH3

calls	syscall
2995	epoll_pwait
2153	read
1828	write
1602	getuid
1500	ioctl
1148	recvfrom
1129	prctl
1089	futex
617	sendto
503	madvise
298	newfstatat
225	mprotect
201	mmap
146	pread64
110	munmap
74	openat
72	close
65	fstat
59	rt_sigprocmask
58	lseek
52	writev
35	faccessat
29	clock_gettime
26	clone
23	sched_yield
21	fcntl
12	timerfd_settim
12	readlinkat
10	dup
7	getsockopt
7	rt_sigaction
6	mkdirat
6	fstatfs
5	uname
4	getdents64
4	mremap
4	epoll_ctl
2	socketpair
2	getpriority
2	getrandom
1	sysinfo
1	ftruncate
1	setsockopt
1	timerfd_create
1	getrlimit
1	eventfd2
16147	total

EA1

calls	syscall
1309	epoll_pwait
1262	clock_gettime
923	write
883	futex
803	read
795	ioctl
734	prctl
705	recvfrom
666	getuid32
430	sendto
220	mprotect
98	writev
96	mmap2
45	fstat64
38	pread64
34	close
32	munmap
30	openat
21	clone
10	dup
10	rt_sigaction
9	fstatat64
7	rt_sigprocmask
7	fcntl64
6	timerfd_settim
5	epoll_ctl
4	madvise
3	ugetrlimit
3	getsockopt
2	lseek
2	getpriority
2	getdents64
2	mkdirat
2	eventfd2
2	getrandom
1	setrlimit
1	sysinfo
1	sigaltstack
1	readlinkat
1	faccessat
1	timerfd_create
1	epoll_create1
1	socketpair
9208	total

EA2

calls	syscall
563722	clock_gettime
325679	futex
116063	epoll_pwait
114975	getuid32
26646	recvfrom
18569	sendto
13210	read
12954	write
7421	ioctl
2230	madvise
1101	prctl
300	fstatat64
203	mprotect
177	mmap2
123	pread64
122	sched_yield
85	munmap
59	writev
59	rt_sigprocmask
53	openat
51	close
45	fstat64
29	gettimeofday
26	clone
20	fcntl64
12	readlinkat
12	_llseek
12	faccessat
11	dup
8	timerfd_settim
5	mkdirat
5	fstatfs64
5	epoll_ctl
4	lseek
4	getsockopt
4	getdents64
3	mremap
2	socketpair
2	eventfd2
2	getpriority
2	ugetrlimit
2	getrandom
1	sysinfo
1	epoll_create1
1	setsockopt
1	uname
1	timerfd_create
1204022	total

EA3

calls	syscall
2120	epoll_pwait
1913	clock_gettime
1277	read
1170	write
991	futex
958	recvfrom
905	ioctl
735	prctl
706	getuid
504	sendto
128	mprotect
117	madvise
110	newfstatat
104	mmap
59	pread64
49	writev
48	munmap
47	rt_sigprocmask
37	close
35	openat
34	fstat
20	clone
13	fcntl
11	lseek
10	dup
9	faccessat
7	timerfd_settim
5	readlinkat
5	mkdirat
4	getdents64
4	epoll_ctl
4	getsockopt
3	getrandom
2	socketpair
2	getrlimit
2	getpriority
1	sysinfo
1	fstatfs
1	eventfd2
1	mremap
1	timerfd_create
1	uname
1	setsockopt
12155	total

EG1

calls	syscall
1659	epoll_pwait
1193	recvfrom
1102	prctl
983	ioctl
975	futex
876	read
790	write
777	getuid32
768	mprotect
578	sendto
170	mmap2
128	madvise
124	fstatat64
96	pread64
83	munmap
78	fstat64
54	openat
51	writev
48	close
36	clock_gettime
21	clone
11	dup
10	fcntl64
10	faccessat
7	_llseek
7	rt_sigprocmask
7	timerfd_settim
6	readlinkat
5	epoll_ctl
5	mkdirat
4	getdents64
4	getsockopt
3	ugetrlimit
3	lseek
2	getpriority
2	mremap
2	fstatfs64
2	eventfd2
2	getrandom
2	socketpair
1	setrlimit
1	sysinfo
1	uname
1	unlinkat
1	timerfd_create
1	epoll_create1
1	socket
1	connect
1	setsockopt
10693	total

EG2

calls	syscall
1766	epoll_pwait
1251	recvfrom
1069	prctl
964	ioctl
942	futex
912	read
807	getuid32
799	write
618	sendto
200	madvise
153	fstatat64
130	mprotect
130	mmap2
84	pread64
62	munmap
47	writev
47	rt_sigprocmask
39	close
37	fstat64
36	clock_gettime
36	openat
20	clone
17	fcntl64
10	dup
10	faccessat
8	_llseek
8	readlinkat
6	timerfd_settim
6	lseek
5	mkdirat
4	getdents64
4	epoll_ctl
4	getsockopt
3	sched_yield
3	fstatfs64
2	getpriority
2	mremap
2	ugetrlimit
2	getrandom
2	socketpair
1	sysinfo
1	uname
1	timerfd_create
1	eventfd2
1	socket
1	connect
1	setsockopt
10254	total

EG3

calls	syscall
1695	epoll_pwait
1255	recvfrom
1070	prctl
976	ioctl
952	futex
902	read
882	write
810	getuid32
635	sendto
193	madvise
157	fstatat64
141	mmap2
139	mprotect
102	pread64
71	munmap
48	writev
47	rt_sigprocmask
45	close
41	fstat64
36	clock_gettime
36	openat
20	clone
20	fcntl64
11	dup
11	faccessat
8	_llseek
8	readlinkat
6	timerfd_settime
6	lseek
5	mkdirat
4	getdents64
4	epoll_ctl
4	getsockopt
3	fstatfs64
2	getpriority
2	mremap
2	ugetrlimit
2	getrandom
2	socketpair
1	sysinfo
1	uname
1	timerfd_create
1	eventfd2
1	socket
1	connect
1	setsockopt
1	sendmsg
10361	total

Figure A3-21. MW3 system call summaries (execution only)

PH1

calls	syscall
684	epoll_pwait
489	ioctl
405	read
367	write
344	getuid
212	futex
187	recvfrom
131	sendto
27	madvise
15	mmap
12	timerfd_settim
12	prctl
10	mprotect
9	fstat
5	close
5	munmap
4	openat
3	clone
1	writev
2922	total

PH2

calls	syscall
600	epoll_pwait
458	ioctl
325	getuid
200	read
170	write
157	futex
138	madvise
102	recvfrom
65	sendto
33	newfstatat
19	timerfd_settim
11	openat
9	mmap
9	close
8	fstat
8	rt_sigprocmask
8	mprotect
5	prctl
4	clone
2	lseek
2	munmap
1	writev
2334	total

PH3

calls	syscall
1498	epoll_pwait
887	getuid
866	read
802	ioctl
505	write
249	futex
100	sendto
84	madvise
42	recvfrom
35	prctl
14	newfstatat
11	timerfd_settim
9	mprotect
7	mmap
7	writev
6	faccessat
6	fstat
6	close
4	rt_sigprocmask
3	fcntl
3	munmap
2	sched_yield
2	dup
2	uname
1	clone
1	socketpair
1	epoll_ctl
1	sendmsg
1	readlinkat
5155	total

EA1

calls	syscall
31109	recvfrom
23338	futex
22411	epoll_pwait
19493	sendto
16461	write
16338	read
16045	getuid32
8455	ioctl
454	clock_gettime
72	writev
21	mprotect
21	prctl
17	madvise
15	mmap2
8	sched_yield
5	timerfd_settim
5	close
4	pread64
4	clone
4	fstat64
4	openat
3	munmap
2	rt_sigprocmask
2	fcntl64
1	dup
1	readlinkat
1	socketpair
154294	total

EA2

calls	syscall
8320	clock_gettime
895	epoll_pwait
462	read
433	ioctl
378	getuid32
273	futex
231	write
67	sendto
63	madvise
40	writev
32	fstatat64
25	prctl
14	mprotect
13	gettimeofday
10	mmap2
10	rt_sigprocmask
9	timerfd_settim
7	recvfrom
5	clone
3	faccessat
2	fcntl64
2	close
2	fstat64
1	munmap
1	epoll_ctl
1	dup
11299	total

EA3

calls	syscall
2078	epoll_pwait
1048	write
1032	read
952	recvfrom
689	ioctl
662	futex
604	getuid
478	sendto
460	clock_gettime
123	madvise
35	writev
24	newfstatat
24	prctl
21	sched_yield
10	rt_sigprocmask
9	mprotect
9	mmap
6	timerfd_settim
4	clone
4	fstat
3	close
3	fcntl
2	dup
1	socketpair
1	readlinkat
1	munmap
8283	total

EG1

calls	syscall
1646	epoll_pwait
1127	recvfrom
814	read
752	write
751	ioctl
681	getuid32
582	futex
524	sendto
42	madvise
40	prctl
35	writev
17	mmap2
13	mprotect
9	fstat64
5	timerfd_settim
5	close
4	munmap
4	openat
4	clone
2	sched_yield
2	rt_sigprocmask
2	fcntl64
1	dup
1	readlinkat
1	socketpair
1	sendmsg
7065	total

EG2

calls	syscall
1312	epoll_pwait
775	recvfrom
641	ioctl
627	write
619	read
586	getuid32
434	futex
405	sendto
110	madvise
37	writev
30	sched_yield
27	prctl
10	rt_sigprocmask
9	mprotect
9	mmap2
9	fstatat64
5	timerfd_settim
4	clone
4	fstat64
4	close
3	fcntl64
2	dup
1	munmap
1	readlinkat
1	socketpair
1	sendmsg
5666	total

EG3

calls	syscall
1046	epoll_pwait
436	read
434	ioctl
383	getuid32
249	write
139	futex
72	sendto
65	madvise
47	recvfrom
36	writev
36	prctl
13	sched_yield
13	mmap2
13	fstatat64
12	rt_sigprocmask
11	mprotect
7	faccessat
5	close
5	clone
5	fstat64
5	timerfd_settime
3	fcntl64
2	dup
1	munmap
1	fsync
1	epoll_ctl
1	openat
1	unlinkat
1	renameat
1	readlinkat
1	fchmodat
1	socketpair
1	getsockopt
3047	total

Figure A3-22. MW3 system call summaries (50 events injected)

PH1 calls syscall	PH2 calls syscall	PH3 calls syscall	EA1 calls syscall	EA2 calls syscall	EA3 calls syscall	EG1 calls syscall	EG2 calls syscall	EG3 calls syscall
124 getuid	146 getuid	163 getuid	1091 clock_gettime	2265 clock_gettime	1025 clock_gettime	114 ioctl	134 getuid32	136 getuid32
99 ioctl	68 ioctl	136 ioctl	123 write	135 getuid32	120 getuid	101 getuid32	119 ioctl	122 ioctl
65 fstat	58 epoll_pwait	91 madvise	123 getuid32	131 ioctl	107 epoll_pwait	94 recvfrom	98 recvfrom	106 write
46 pread64	56 madvise	72 newfstatat	114 ioctl	115 write	105 recvfrom	87 write	93 write	106 recvfrom
40 rt_sigprocmask	46 pread64	69 writev	110 recvfrom	110 epoll_pwait	105 write	76 epoll_pwait	91 epoll_pwait	104 epoll_pwait
35 epoll_pwait	44 rt_sigprocmask	69 pread64	102 epoll_pwait	110 recvfrom	91 ioctl	66 pread64	64 pread64	64 pread64
32 write	43 write	64 epoll_pwait	79 futex	66 pread64	46 pread64	50 futex	49 rt_sigprocmask	49 rt_sigprocmask
32 writev	39 writev	62 mmap	46 pread64	49 rt_sigprocmask	44 rt_sigprocmask	48 mmap2	40 mmap2	43 futex
32 mmap	36 newfstatat	57 rt_sigprocmask	40 rt_sigprocmask	44 futex	43 futex	46 rt_sigprocmask	39 madvise	42 madvise
23 munmap	26 recvfrom	51 write	35 mmap2	42 fstatat64	36 newfstatat	39 munmap	38 futex	40 mmap2
19 recvfrom	26 mmap	46 futex	32 read	39 mmap2	33 read	27 read	34 munmap	33 munmap
16 futex	23 munmap	44 munmap	27 munmap	33 munmap	31 madvise	16 mprotect	28 read	33 read
9 close	21 futex	39 mprotect	24 writev	32 read	29 mmap	15 prctl	15 writev	16 writev
8 prctl	10 read	27 recvfrom	14 mprotect	22 madvise	23 munmap	14 madvise	14 mprotect	14 fstatat64
7 read	5 close	18 prctl	12 prctl	15 writev	11 writev	12 writev	14 fstatat64	13 mprotect
5 mprotect	4 getdents64	15 clock_gettime	11 close	14 mprotect	5 process_vm_readv	11 close	10 close	10 close
4 openat	4 fstat	13 read	6 dup	13 gettimeofday	4 mprotect	10 process_vm_readv	6 fstat64	6 fstat64
4 getdents64	4 mprotect	11 close	6 fstat64	9 close	4 prctl	10 fstat64	5 dup	5 dup
3 clock_gettime	3 process_vm_readv	8 openat	5 epoll_ctl	8 fstat64	4 fstat	6 dup	5 process_vm_readv	5 process_vm_readv
3 process_vm_readv	2 epoll_ctl	8 fstat	5 process_vm_readv	5 dup	3 close	5 epoll_ctl	4 lseek	4 lseek
2 epoll_ctl	2 openat	6 clone	4 openat	5 process_vm_readv	2 epoll_ctl	4 openat	4 prctl	4 prctl
2 dup	2 clone	5 dup	3 clone	4 epoll_ctl	2 clone	3 clone	4 epoll_ctl	4 epoll_ctl
2 fcntl	1 dup	4 epoll_ctl	2 lseek	3 fcntl64	1 dup	3 clock_gettime	3 fcntl64	3 fcntl64
2 clone	1 fcntl	4 fcntl	2 fcntl64	3 prctl	1 fcntl	2 lseek	3 clock_gettime	3 clock_gettime
1 newfstatat	1 rt_sigreturn	3 process_vm_readv	1 rt_sigreturn	2 clone	1 rt_sigreturn	2 rt_sigreturn	2 clone	2 clone
1 rt_sigreturn	1 prctl	2 lseek	1 eventfd2	2 openat		2 fcntl64	2 openat	2 openat
1 madvise		1 faccessat	1 epoll_create1	2 lseek		1 eventfd2	1 rt_sigreturn	1 rt_sigreturn
		1 rt_sigreturn	1 getsockopt	1 rt_sigreturn		1 epoll_create1	1 getsockopt	1 getsockopt
		1 getsockopt		1 getsockopt		1 getsockopt		
		1 fstatfs						
		1 readlinkat						
		1 sysinfo						
		1 mremap						
617 total	672 total	1094 total	2020 total	3280 total	1876 total	866 total	920 total	971 total

Figure A3-23. MW4 system call summaries (execution only)

PH1 calls syscall	PH2 calls syscall	PH3 calls syscall	EA1 calls syscall	EA2 calls syscall	EA3 calls syscall	EG1 calls syscall	EG2 calls syscall	EG3 calls syscall
99 getuid	114 getuid	101 getuid	1005 clock_gettime	1720 clock_gettime	1007 clock_gettime	46 ioctl	101 getuid32	103 getuid32
95 ioctl	64 ioctl	70 ioctl	115 getuid32	107 getuid32	107 getuid	46 pread64	79 ioctl	81 ioctl
62 fstat	50 madvise	60 newfstatat	112 write	94 write	105 write	43 getuid32	70 recvfrom	77 recvfrom
46 pread64	46 epoll_pwait	50 pread64	105 recvfrom	94 recvfrom	96 recvfrom	40 rt_sigprocmask	64 epoll_pwait	69 epoll_pwait
40 rt_sigprocmask	46 pread64	46 writev	87 ioctl	87 epoll_pwait	89 epoll_pwait	30 mmap2	59 write	66 write
32 epoll_pwait	44 rt_sigprocmask	42 epoll_pwait	83 epoll_pwait	85 ioctl	86 ioctl	29 munmap	46 pread64	46 pread64
32 write	39 write	42 rt_sigprocmask	73 futex	46 pread64	46 pread64	26 epoll_pwait	44 rt_sigprocmask	44 rt_sigprocmask
31 mmap	36 newfstatat	38 write	46 pread64	44 rt_sigprocmask	44 rt_sigprocmask	25 write	29 futex	32 madvise
25 futex	28 writev	34 recvfrom	40 rt_sigprocmask	39 futex	42 futex	15 recvfrom	29 madvise	29 futex
23 munmap	27 mmap	29 mmap	34 writev	36 fstatat64	36 newfstatat	14 futex	27 mmap2	27 mmap2
21 writev	26 recvfrom	28 munmap	32 mmap2	27 mmap2	30 madvise	7 prctl	24 munmap	23 munmap
19 recvfrom	24 munmap	20 futex	28 read	26 read	29 mmap	7 madvise	18 read	21 read
8 prctl	17 futex	20 madvise	26 munmap	23 munmap	26 read	5 mprotect	12 fstatat64	12 fstatat64
7 close	10 read	12 read	9 mprotect	21 madvise	23 munmap	5 writev	11 writev	11 writev
7 read	5 close	4 fstat	9 prctl	11 writev	11 writev	5 fstat64	5 process_vm_readv	5 process_vm_readv
6 mprotect	4 getdents64	3 mprotect	6 fstat64	11 gettimeofday	5 process_vm_readv	5 process_vm_readv	4 mprotect	4 mprotect
4 openat	4 fstat	3 process_vm_readv	5 close	5 process_vm_readv	4 mprotect	5 read	4 fstat64	4 fstat64
4 getdents64	4 mprotect	3 clock_gettime	5 process_vm_readv	4 fstat64	4 prctl	2 close	3 close	3 epoll_ctl
3 epoll_ctl	3 epoll_ctl	3 prctl	3 epoll_ctl	4 mprotect	4 fstat	1 dup	3 epoll_ctl	3 clock_gettime
3 clock_gettime	3 process_vm_readv	1 clone	2 clone	3 close	3 close	1 clone	3 clock_gettime	3 close
3 process_vm_readv	2 openat	1 faccessat	2 madvise	3 epoll_ctl	3 epoll_ctl	1 rt_sigreturn	2 clone	2 clone
2 clone	2 prctl	1 rt_sigreturn	2 openat	2 clone	2 clone	1 fcntl64	2 prctl	2 prctl
1 dup	2 clone	1 dup	2 sendto	2 prctl	1 rt_sigreturn	1 epoll_ctl	1 dup	1 dup
1 fcntl	1 dup	1 fcntl	1 dup	1 dup	1 sendto	1 openat	1 rt_sigreturn	1 rt_sigreturn
1 newfstatat	1 fcntl	1 close	1 rt_sigreturn	1 fcntl64	1 fcntl		1 fcntl64	1 fcntl64
1 rt_sigreturn	1 rt_sigreturn	1 epoll_ctl	1 fcntl64	1 rt_sigreturn	1 dup			
576 total	603 total	615 total	1834 total	2497 total	1806 total	361 total	642 total	670 total

Figure A3-24. MW4 system call summaries (50 events injected)

PH1	PH2	PH3	EA1	EA2	EA3	EG1	EG2	EG3
calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall
910 clock_gettime	1100 clock_gettime	905 clock_gettime	1810 clock_gettime	465 clock_gettime	67 recvfrom	1518 ioctl	906 clock_gettime	896 clock_gettime
667 epoll_pwait	720 epoll_pwait	667 epoll_pwait	302 mprotect	74 epoll_pwait	65 epoll_pwait	1031 fstat64	726 epoll_pwait	687 epoll_pwait
634 faccessat	634 faccessat	635 faccessat	278 futex	64 recvfrom	58 write	1012 openat	716 madvise	648 madvise
383 write	467 getuid	411 write	270 ioctl	60 write	40 getuid	998 close	671 mmap2	637 mmap2
362 getuid	414 write	341 getuid	192 mmap2	54 getuid32	27 ioctl	909 clock_gettime	634 faccessat	630 faccessat
319 close	319 openat	315 close	159 write	42 ioctl	22 read	838 mmap2	634 prctl	627 prctl
317 openat	319 close	313 openat	157 getuid32	26 futex	20 futex	694 epoll_pwait	419 getuid32	385 write
317 fstat	316 fstat	313 fsync	156 prctl	26 read	10 clock_gettime	678 mprotect	418 write	380 getuid32
313 fsync	316 newfstatat	313 unlinkat	150 epoll_pwait	6 fcntl64	1 timerfd_settim	651 faccessat	332 close	315 close
313 unlinkat	313 unlinkat	313 fchmodat	118 pread64	5 gettimeofday	1 close	564 futex	323 fstat64	313 fstatat64
313 fchmodat	313 fsync	313 fstat	117 recvfrom	5 pread64	1 mprotect	477 fstatat64	319 fstatat64	313 fstat64
313 getsockopt	313 fchmodat	313 newfstatat	116 fcntl64	4 close	1 epoll_ctl	459 write	315 openat	311 fsync
313 newfstatat	313 getsockopt	313 getsockopt	93 fstat64	3 mmap2	313 total	428 getuid32	315 getsockopt	311 openat
312 renameat	312 renameat	312 renameat	89 munmap	2 fstat64		314 getsockopt	313 fsync	311 unlinkat
167 futex	196 ioctl	79 recvfrom	79 rt_sigprocmask	2 epoll_ctl		313 unlinkat	313 unlinkat	311 fchmodat
71 mprotect	181 futex	65 futex	79 read	2 dup		313 fchmodat	313 fchmodat	311 getsockopt
70 ioctl	90 recvfrom	62 mprotect	72 openat	2 fstatat64		310 fsync	312 renameat	310 renameat
63 recvfrom	73 mprotect	56 ioctl	68 close	2 munmap		308 renameat	120 ioctl	103 futex
31 read	49 read	32 read	56 pwrite64	1 timerfd_settime		240 prctl	118 futex	91 recvfrom
12 prctl	15 rt_sigprocmask	8 mmap	39 fstatat64	1 mprotect		136 pread64	111 recvfrom	65 mprotect
12 mmap	12 mmap	8 rt_sigprocmask	33 clone	846 total		118 fcntl64	78 mprotect	56 ioctl
8 madvise	7 writev	5 madvise	16 writev			107 madvise	73 rt_sigprocmask	29 read
7 writev	6 pread64	4 clone	12 fdatasync			96 munmap	53 pread64	13 rt_sigprocmask
6 timerfd_settim	6 prctl	4 prctl	10 rt_sigaction			82 recvfrom	37 read	11 pread64
4 clone	5 clone	4 writev	8 dup			79 rt_sigprocmask	27 munmap	6 fcntl64
2 epoll_ctl	3 timerfd_settim	3 timerfd_settim	8 faccessat			72 read	19 fcntl64	5 munmap
1 dup	3 munmap	2 epoll_ctl	7 epoll_ctl			56 pwrite64	8 dup	4 clone
1 fcntl	2 epoll_ctl	2 getrandom	6 geteuid32			32 clone	7 clone	2 dup
1 uname	2 sched_yield	1 dup	4 getdents64			15 writev	4 writev	2 writev
1 getrandom	2 madvise	1 fcntl	4 getsockopt			12 fdatasync	4 epoll_ctl	2 epoll_ctl
6243 total	1 dup	6113 total	3 madvise			9 mkdirat	4 lseek	2 getrandom
	1 fcntl		3 mkdirat			8 dup	2 timerfd_settim	1 timerfd_settim
	1 lseek		2 lseek			8 readlinkat	2 getrandom	8088 total
	1 getrandom		2 getpriority			7 epoll_ctl	8646 total	
	6825 total		2 _llseek			6 _llseek		
			2 ugetrlimit			6 geteuid32		
			2 ftruncate64			4 sched_yield		
			2 unlinkat			4 getdents64		
			2 readlinkat			3 lseek		
			2 fchmodat			2 getpriority		
			2 eventfd2			2 mremap		
			1 sysinfo			2 ugetrlimit		
			1 sched_yield			2 ftruncate64		
			1 mremap			2 fstatfs64		
			1 sigaltstack			2 timerfd_settime		
			1 timerfd_create			2 eventfd2		
			1 timerfd_settime			2 getrandom		
			1 epoll_create1			1 sysinfo		
			4539 total			1 uname		
						1 timerfd_create		
						1 epoll_create1		
						1 socketpair		
						1 setsockopt		
						12937 total		

Figure A3-25. MW5 system call summaries (execution only)

PH1	PH2	PH3	EA1	EA2	EA3	EG1	EG2	EG3
calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall
908 clock_gettime	911 clock_gettime	867 clock_gettime	69 write	186 clock_gettime	81 epoll_pwait	1296 ioctl	900 clock_gettime	891 clock_gettime
649 epoll_pwait	672 epoll_pwait	598 epoll_pwait	67 recvfrom	31 epoll_pwait	76 write	931 close	737 madvise	601 epoll_pwait
634 faccessat	632 faccessat	578 faccessat	65 epoll_pwait	31 recvfrom	67 clock_gettime	928 openat	713 epoll_pwait	595 madvise
366 write	392 write	300 write	42 getuid32	27 write	67 recvfrom	928 fstat64	700 mmap2	594 faccessat
339 getuid	352 getuid	296 getuid	33 futex	26 futex	48 ioctl	887 clock_gettime	638 prctl	594 mmap2
317 openat	324 openat	290 close	29 ioctl	22 getuid32	45 getuid	712 epoll_pwait	630 faccessat	594 prctl
317 close	322 close	289 fsync	21 read	18 ioctl	32 read	628 mmap2	416 getuid32	302 write
317 fstat	319 fstat	289 openat	14 clock_gettime	12 read	28 futex	624 faccessat	407 write	300 getuid32
313 unlinkat	319 newfstatat	289 unlinkat	1 close	1 epoll_ctl	5 timerfd_settim	423 write	335 close	298 close
313 fsync	313 fsync	289 renameat	1 epoll_ctl	1 close	3 close	391 getuid32	328 fstat64	297 fsync
313 fchmodat	313 unlinkat	289 fchmodat	342 total	1 mprotect	1 mprotect	308 fsync	322 fstatat64	297 openat
313 newfstatat	313 fchmodat	289 newfstatat		1 timerfd_settim	1 uname	308 fstatat64	313 openat	297 unlinkat
313 getsockopt	313 getsockopt	289 fstat		357 total	1 epoll_ctl	308 unlinkat	313 getsockopt	297 renameat
312 renameat	312 renameat	289 getsockopt			1 mmap	308 fchmodat	311 fsync	297 getsockopt
168 futex	143 futex	84 futex			1 munmap	308 getsockopt	311 unlinkat	297 fstatat64
73 mprotect	71 recvfrom	28 mprotect			1 fcntl	307 renameat	311 fchmodat	297 fchmodat
56 recvfrom	45 read	14 read			458 total	245 futex	310 renameat	297 fstat64
42 ioctl	42 ioctl	11 ioctl				161 recvfrom	191 futex	58 futex
18 read	37 mprotect	3 timerfd_settime				78 mprotect	139 ioctl	8 mprotect
12 prctl	10 rt_sigprocmask	3 recvfrom				43 read	103 recvfrom	5 ioctl
12 mmap	9 mmap	1 writev				14 madvise	101 rt_sigprocmask	4 read
7 madvise	7 timerfd_settim	1 epoll_ctl				12 prctl	93 mprotect	3 recvfrom
4 clone	5 clone	5386 total				4 clone	92 pread64	1 epoll_ctl
2 timerfd_settim	4 prctl					2 sched_yield	47 munmap	1 timerfd_settim
1 getrandom	2 lseek					2 getrandom	35 read	7225 total
6119 total	2 writev					1 dup	26 fcntl64	
	1 epoll_ctl					1 writev	11 clone	
	1 sched_yield					1 fcntl64	9 dup	
	1 madvise					1 epoll_ctl	4 writev	
	1 getrandom					1 timerfd_settim	4 epoll_ctl	
	6188 total					10161 total	4 lseek	
							2 timerfd_settime	
							2 getrandom	
							1 sched_yield	
							8859 total	

Figure A3-26. MW5 system call summaries (50 events injected)

PH1

calls	syscall
305793	clock_gettime
68061	recvfrom
66651	write
49861	epoll_pwait
34195	getuid32
17566	futex
17379	ioctl
17015	read
17014	gettimeofday
69	fstat64
68	mprotect
55	close
50	rt_sigprocmask
49	mmap2
42	madvise
25	openat
22	prctl
20	munmap
20	fcntl64
16	pwrite64
15	dup
13	fstatat64
11	pread64
10	writev
7	clone
6	faccessat
4	fdatasync
4	geteuid32
2	getpriority
2	setpriority
2	fsync
2	getsockopt
2	unlinkat
2	fchmodat
1	sched_yield
1	renameat
594055	total

PH2

calls	syscall
263671	clock_gettime
57239	recvfrom
48156	write
33966	epoll_pwait
28779	getuid32
14806	futex
14648	ioctl
14303	read
14303	gettimeofday
198	mprotect
81	madvise
56	rt_sigprocmask
37	mmap2
32	close
24	fstatat64
18	munmap
16	prctl
15	fcntl64
12	fstat64
10	pread64
7	writev
6	faccessat
4	clone
3	openat
2	getpriority
2	setpriority
2	fsync
2	getsockopt
2	unlinkat
2	fchmodat
1	_llseek
1	renameat
490404	total

PH3

calls	syscall
2367	clock_gettime
521	gettimeofday
291	recvfrom
289	write
230	ioctl
222	epoll_pwait
137	getuid32
93	futex
87	read
37	rt_sigprocmask
16	close
8	fcntl64
8	munmap
8	mmap2
6	mprotect
4	writev
2	setpriority
2	getpriority
2	madvise
4330	total

EA1

calls	syscall
68092	recvfrom
55670	write
50915	epoll_pwait
34110	getuid32
21046	futex
17172	ioctl
17031	read
1045	clock_gettime
144	mprotect
97	rt_sigprocmask
70	mmap2
39	pread64
36	fstat64
33	munmap
33	close
32	prctl
29	openat
21	madvise
14	writev
7	clone
7	faccessat
5	dup
5	getsockopt
3	fsync
3	fcntl64
3	epoll_ctl
3	fstatat64
3	unlinkat
3	fchmodat
2	lseek
2	getpriority
2	setpriority
1	mremap
1	fstatfs64
1	renameat
1	readlinkat
1	eventfd2
1	epoll_create1
265683	total

EA2

calls	syscall
306794	clock_gettime
64580	recvfrom
63709	write
47677	epoll_pwait
32318	getuid32
17284	futex
16175	ioctl
16147	read
16096	gettimeofday
207	mprotect
41	madvise
38	rt_sigprocmask
15	mmap2
14	fstatat64
10	prctl
6	faccessat
4	clone
4	fstat64
2	pread64
2	unlinkat
2	writev
2	fchmodat
2	setpriority
2	getsockopt
2	close
2	getpriority
2	fsync
2	openat
1	munmap
1	renameat
581141	total

EA3

calls	syscall
69436	recvfrom
49895	write
49072	epoll_pwait
34748	getuid32
17676	futex
17389	ioctl
17364	read
554	clock_gettime
463	mprotect
63	madvise
44	sched_yield
39	rt_sigprocmask
14	mmap2
14	fstatat64
9	prctl
6	faccessat
4	clone
4	fstat64
2	openat
2	unlinkat
2	fsync
2	writev
2	getpriority
2	fchmodat
2	close
2	pread64
2	setpriority
2	getsockopt
1	renameat
1	munmap
256816	total

EG1

calls	syscall
50108	recvfrom
36502	write
31131	epoll_pwait
25169	getuid32
17024	futex
12685	ioctl
12528	read
131	mprotect
100	rt_sigprocmask
89	mmap2
68	writev
58	fcntl64
56	fstat64
54	madvise
48	pread64
48	prctl
42	pwrite64
40	munmap
40	close
38	openat
14	fstatat64
12	clock_gettime
11	faccessat
9	clone
6	fdatasync
6	fchmodat
6	getsockopt
5	dup
5	sched_yield
5	unlinkat
3	fsync
3	geteuid32
3	epoll_ctl
2	lseek
2	getpriority
2	setpriority
1	mremap
1	ftruncate64
1	statfs64
1	fstatfs64
1	mkdirat
1	renameat
1	readlinkat
1	eventfd2
1	epoll_create1
186062	total

EG2

calls	syscall
54884	recvfrom
37029	write
34296	epoll_pwait
27495	getuid32
13834	futex
13750	ioctl
13720	read
2167	mprotect
88	madvise
38	rt_sigprocmask
27	writev
14	mmap2
9	prctl
6	fstatat64
6	faccessat
4	clone
4	fstat64
2	getpriority
2	setpriority
2	fsync
2	pread64
2	openat
2	unlinkat
2	fchmodat
2	getsockopt
2	close
1	munmap
1	sched_yield
1	renameat
197392	total

EG3

calls	syscall
48044	recvfrom
34168	write
27317	epoll_pwait
24075	getuid32
12194	futex
12040	ioctl
12012	read
1900	mprotect
70	madvise
38	rt_sigprocmask
26	writev
14	mmap2
9	prctl
6	fstatat64
6	faccessat
4	clone
4	fstat64
2	getpriority
2	setpriority
2	fsync
2	pread64
2	openat
2	unlinkat
2	fchmodat
2	getsockopt
2	close
1	munmap
1	renameat
171947	total

Figure A3-27. MW6 system call summaries (execution only)

PH1

calls	syscall
308217	clock_gettime
68572	recvfrom
67425	write
50516	epoll_pwait
34412	getuid32
17654	futex
17359	ioctl
17151	read
17139	gettimeofday
61	mprotect
45	rt_sigprocmask
40	madvise
32	fstat64
30	mmap2
27	sched_yield
22	prctl
20	fcntl64
18	openat
18	close
16	pwrite64
14	munmap
13	fstatat64
9	pread64
6	faccessat
4	clone
4	fdatasync
4	geteuid32
2	fsync
2	getpriority
2	setpriority
2	writev
2	getsockopt
2	unlinkat
2	fchmodat
1	renameat
598843	total

PH2

calls	syscall
269900	clock_gettime
58488	recvfrom
49628	write
35140	epoll_pwait
29407	getuid32
15094	futex
14990	ioctl
14645	gettimeofday
14618	read
229	mprotect
167	madvise
96	pread64
89	fcntl64
71	fstatat64
68	mmap2
63	rt_sigprocmask
56	close
46	fstat64
28	openat
22	munmap
16	prctl
14	_llseek
12	faccessat
9	geteuid32
8	pwrite64
5	fsync
5	clone
5	getsockopt
5	unlinkat
5	fchmodat
4	dup
4	fdatasync
4	renameat
3	writev
2	getpriority
2	setpriority
502948	total

PH3

calls	syscall
801	clock_gettime
165	gettimeofday
99	write
99	recvfrom
81	epoll_pwait
75	ioctl
47	getuid32
33	read
32	futex
3	mprotect
1	getpriority
1	setpriority
1437	total

EA1

calls	syscall
68596	recvfrom
52265	write
51480	epoll_pwait
34334	getuid32
17383	futex
17197	ioctl
17157	read
516	clock_gettime
52	mprotect
36	rt_sigprocmask
23	mmap2
20	madvise
17	prctl
15	fstat64
11	openat
11	close
6	faccessat
4	munmap
4	clone
4	writev
2	getpriority
2	setpriority
2	fsync
2	pread64
2	fstatat64
2	unlinkat
2	fchmodat
2	getsockopt
1	renameat
259148	total

EA2

calls	syscall
308780	clock_gettime
65003	recvfrom
64109	write
47979	epoll_pwait
32530	getuid32
16600	futex
16280	ioctl
16253	read
16201	gettimeofday
207	mprotect
60	madvise
38	rt_sigprocmask
15	mmap2
14	fstatat64
10	prctl
6	faccessat
4	clone
4	fstat64
2	setpriority
2	pread64
2	writev
2	getpriority
2	openat
2	fsync
2	fchmodat
2	unlinkat
2	getsockopt
2	close
1	munmap
1	renameat
584115	total

EA3

calls	syscall
33668	recvfrom
26521	write
25286	epoll_pwait
16865	getuid32
8932	futex
8455	ioctl
8422	read
266	mprotect
260	clock_gettime
47	madvise
26	rt_sigprocmask
2	writev
2	getpriority
2	setpriority
128754	total

EG1

calls	syscall
51710	recvfrom
38371	write
32274	epoll_pwait
25926	getuid32
17948	futex
12975	ioctl
12925	read
48	madvise
44	writev
40	mprotect
33	rt_sigprocmask
18	mmap2
15	fstat64
12	prctl
11	openat
11	close
6	faccessat
4	clone
2	fstatat64
2	getpriority
2	setpriority
2	fsync
2	pread64
2	unlinkat
2	fchmodat
2	getsockopt
1	munmap
1	sched_yield
1	renameat
192390	total

EG2

calls	syscall
56424	recvfrom
42941	write
37886	epoll_pwait
28283	getuid32
14238	futex
14135	ioctl
14106	read
2120	mprotect
77	madvise
44	writev
38	rt_sigprocmask
14	mmap2
9	prctl
6	fstatat64
6	faccessat
4	clone
4	fstat64
2	getsockopt
2	getpriority
2	setpriority
2	fsync
2	pread64
2	openat
2	unlinkat
2	fchmodat
2	close
1	munmap
1	sched_yield
1	renameat
210356	total

EG3

calls	syscall
47483	recvfrom
34201	write
27169	epoll_pwait
23794	getuid32
12042	futex
11899	ioctl
11870	read
1875	mprotect
85	madvise
38	rt_sigprocmask
28	writev
14	mmap2
9	prctl
6	fstatat64
6	faccessat
4	clone
4	fstat64
2	getpriority
2	setpriority
2	fsync
2	pread64
2	openat
2	unlinkat
2	fchmodat
2	getsockopt
2	close
1	munmap
1	renameat
170547	total

Figure A3-28. MW6 system call summaries (50 events injected)

PH1	PH2	PH3	EA1	EA2	EA3	EG1	EG2	EG3
calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall
3885 clock_gettime	3711 clock_gettime	4550 clock_gettime	4993 clock_gettime	20592 clock_gettime	3027 epoll_pwait	3037 epoll_pwait	3371 clock_gettime	2846 epoll_pwait
2850 epoll_pwait	2957 epoll_pwait	3182 epoll_pwait	3820 epoll_pwait	3616 epoll_pwait	2336 write	2391 recvfrom	2982 epoll_pwait	2329 recvfrom
2429 recvfrom	2421 write	2471 recvfrom	2569 getuid32	2545 getuid32	2316 recvfrom	2267 write	2382 write	2144 write
2281 write	2417 recvfrom	2458 write	2461 write	2440 recvfrom	2059 clock_gettime	1890 getuid32	2323 recvfrom	1881 getuid32
2166 getuid	2044 getuid	1997 getuid	2407 recvfrom	2311 write	1891 getuid	1788 clock_gettim	1902 getuid32	1855 clock_gettime
1567 futex	1385 futex	1283 futex	1516 futex	1321 futex	1455 futex	1621 futex	1251 futex	1236 futex
969 ioctl	748 ioctl	716 read	739 read	697 read	679 read	695 read	662 read	661 read
698 read	699 read	676 ioctl	641 ioctl	642 ioctl	609 ioctl	601 ioctl	641 ioctl	607 ioctl
487 fstat	113 madvise	84 madvise	20 madvise	51 madvise	16 madvise	3 mprotect	55 madvise	29 madvise
68 pread64	60 pread64	52 writev	18 pread64	31 rt_sigprocmask	11 sched_yield	3 madvise	48 pread64	10 mprotect
57 writev	60 fstat	35 mprotect	18 mmap2	24 gettimeofday	10 mprotect	1 epoll_ctl	43 mmap2	10 rt_sigprocmask
55 mmap	44 writev	25 mmap	14 mprotect	16 mprotect	8 rt_sigprocmask	1 close	35 rt_sigprocmask	8 writev
38 rt_sigprocmask	44 mmap	21 rt_sigprocmask	10 close	16 mmap2	7 writev		24 munmap	8 mmap2
33 munmap	40 rt_sigprocmask	16 munmap	10 writev	16 pread64	6 mmap		18 mprotect	7 close
23 close	26 munmap	15 pread64	9 munmap	8 close	4 close		13 close	5 prctl
22 mprotect	25 newfstatat	14 prctl	9 prctl	7 munmap	4 prctl		12 prctl	3 connect
21 prctl	18 close	12 newfstatat	8 rt_sigprocmask	7 writev	3 epoll_ctl		12 fstat64	3 clone
14 madvise	18 mprotect	6 close	8 fstat64	6 connect	3 fstat		11 writev	3 fstat64
13 openat	8 fcntl	5 openat	4 connect	6 socket	2 clone		8 fstatat64	3 epoll_ctl
6 clone	7 prctl	4 connect	4 socket	5 prctl	2 faccessat		6 clone	3 socket
6 faccessat	6 faccessat	4 socket	3 clone	5 getsockopt	1 dup		6 fcntl64	3 getsockopt
4 epoll_ctl	6 openat	3 fstat	3 fcntl64	4 sendmsg	1 getsockopt		6 faccessat	2 faccessat
4 dup	6 clone	2 clone	3 openat	3 clone	1 fcntl		5 getsockopt	2 sendmsg
4 fcntl	5 getsockopt	2 epoll_ctl	2 dup	3 fstat64	1 fsync		4 epoll_ctl	1 dup
4 getdents64	4 connect	2 fcntl	2 epoll_ctl	2 sendto	1 openat		4 socket	1 fsync
4 socket	4 epoll_ctl	2 getsockopt	2 sendmsg	2 epoll_ctl	1 newfstatat		4 connect	1 fcntl64
4 connect	4 getdents64	2 sendmsg	1 getsockname	2 faccessat	1 unlinkat		2 dup	1 openat
3 newfstatat	4 socket	1 mremap	1 sendto	2 getsockname	1 renameat		2 fsync	1 fstatat64
3 getsockopt	2 dup	1 fstatfs	1 setsockopt	1 openat	1 fchmodat		2 openat	1 unlinkat
2 unlinkat	2 unlinkat	1 readlinkat	1 getsockopt	1 fsync			2 unlinkat	1 renameat
2 renameat	2 renameat	1 getsockname		1 unlinkat			2 renameat	1 fchmodat
2 fchmodat	2 fchmodat	1 sendto		1 renameat			2 fchmodat	1 getsockname
2 fsync	2 fsync	1 getrandom		1 fchmodat			2 sendmsg	1 sendto
2 sendmsg	2 sendmsg	1 dup		1 fstatat64			1 getsockname	
1 lseek	1 lseek			1 dup			1 sendto	
1 getsockname	1 getsockname			1 fcntl64			1 setsockopt	
1 sendto	1 sendto							
1 setsockopt	1 setsockopt							
17732 total	16900 total	17646 total	19297 total	34388 total	14457 total	14298 total	15845 total	13668 total

Figure A3-29. MW7 system call summaries (execution only)

PH1	PH2	PH3	EA1	EA2	EA3	EG1	EG2	EG3
calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall	calls syscall
3577 clock_gettime	3343 clock_gettime	2523 epoll_pwait	3105 epoll_pwait	15773 clock_gettime	3168 epoll_pwait	2966 epoll_pwait	2971 epoll_pwait	2774 epoll_pwait
2853 epoll_pwait	2943 epoll_pwait	2468 recvfrom	3036 clock_gettime	3074 epoll_pwait	2472 recvfrom	2332 recvfrom	2292 recvfrom	2319 recvfrom
2413 recvfrom	2429 recvfrom	2445 write	2469 write	2464 recvfrom	2467 write	2223 write	2236 write	2127 write
2277 write	2312 write	1964 clock_gettime	2442 recvfrom	2410 write	2033 clock_gettime	1856 getuid32	1875 getuid32	1852 getuid32
2099 getuid	2023 getuid	1366 getuid	1950 getuid32	1959 getuid32	1957 getuid	1779 clock_gettim	1873 clock_gettime	1782 clock_gettime
1523 futex	1351 futex	1262 futex	1628 futex	1411 futex	1427 futex	1617 futex	1274 futex	1181 futex
837 ioctl	710 ioctl	717 read	709 read	722 read	717 read	671 read	661 read	671 read
701 read	701 read	672 ioctl	669 ioctl	644 ioctl	641 ioctl	583 ioctl	596 ioctl	584 ioctl
294 fstat	97 madvise	79 madvise	34 mmap2	33 madvise	16 rt_sigprocmask	10 madvise	28 madvise	10 madvise
66 pread64	53 fstat	30 mprotect	22 pread64	14 gettimeofday	14 madvise	3 mprotect	9 mprotect	3 mprotect
54 writev	34 writev	20 mmap	20 mprotect	11 mprotect	9 mprotect	1 writev	8 rt_sigprocmask	1 epoll_ctl
52 mmap	30 pread64	19 writev	19 close	10 rt_sigprocmask	5 writev		6 writev	1 close
42 madvise	26 mmap	14 prctl	18 prctl	8 mmap2	4 mmap		6 mmap2	
38 rt_sigprocmask	25 rt_sigprocmask	12 munmap	17 fstat64	7 writev	4 close		4 prctl	
29 munmap	14 munmap	8 rt_sigprocmask	15 rt_sigprocmask	7 close	3 prctl		4 close	
22 close	13 mprotect	6 close	14 writev	5 prctl	3 fstat		3 epoll_ctl	
22 mprotect	10 close	6 openat	12 openat	3 clone	2 dup		3 fstat64	
20 prctl	5 prctl	5 pread64	10 munmap	3 socket	2 getsockopt		2 clone	
12 openat	5 clone	4 fstat	8 madvise	3 fstat64	2 fcntl		2 faccessat	
6 faccessat	4 socket	3 epoll_ctl	6 clone	3 epoll_ctl	2 faccessat		1 dup	
6 clone	4 fcntl	2 clone	4 connect	3 connect	1 clone		1 fsync	
5 epoll_ctl	4 connect	2 faccessat	4 socket	3 getsockopt	1 epoll_ctl		1 fcntl64	
4 dup	4 getsockopt	2 fcntl	3 epoll_ctl	2 sendmsg	1 fsync		1 openat	
4 fcntl	3 epoll_ctl	1 fsync	2 fcntl64	2 faccessat	1 openat		1 fstatat64	
4 getdents64	2 faccessat	1 readlinkat	2 faccessat	1 dup	1 newfstatat		1 unlinkat	
4 socket	2 sendmsg	1 unlinkat	2 getsockopt	1 fcntl64	1 unlinkat		1 renameat	
4 connect	1 sendto	1 fchmodat	2 sendmsg	1 fsync	1 renameat		1 fchmodat	
3 newfstatat	1 dup	1 newfstatat	1 dup	1 openat	1 fchmodat		1 getsockopt	
3 getsockopt	1 unlinkat	1 fstatfs	1 fsync	1 fstatat64				
2 unlinkat	1 renameat	1 mremap	1 sched_yield	1 unlinkat				
2 renameat	1 fchmodat	1 getsockopt	1 fstatat64	1 renameat				
2 fchmodat	1 openat	1 getrandom	1 unlinkat	1 fchmodat				
2 fsync	1 newfstatat	1 renameat	1 renameat	1 getsockname				
2 sendmsg	1 fsync	1 socket	1 fchmodat	1 sendto				
1 lseek	1 getsockname	1 connect	1 getsockname					
1 getsockname	1 setsockopt	1 dup	1 sendto					
1 sendto			1 setsockopt					
1 setsockopt								
16988 total	16157 total	13642 total	16232 total	28584 total	14956 total	14041 total	13862 total	13305 total

Figure A3-30. MW7 system call summaries (50 events injected)

PH1

calls	syscall
1146	write
964	recvfrom
650	read
642	epoll_pwait
542	futex
371	getuid
306	ioctl
206	sendto
33	fstat
28	prctl
20	timerfd_settim
17	mmap
7	clock_gettime
7	rt_sigprocmask
6	munmap
5	openat
5	mprotect
4	close
4	pread64
4	madvise
3	dup
3	writev
2	fcntl
2	clone
1	readlinkat
1	getrlimit
1	setrlimit
1	socketpair
4981	total

PH2

calls	syscall
1103	write
836	recvfrom
607	read
578	epoll_pwait
498	futex
446	getuid
386	ioctl
175	sendto
89	madvise
42	fstat
40	rt_sigprocmask
35	prctl
33	clock_gettime
26	mmap
25	timerfd_settim
22	close
16	pread64
14	writev
13	munmap
13	newfstatat
10	fcntl
10	mprotect
9	dup
7	openat
4	clone
2	uname
2	getsockopt
1	fsync
1	epoll_ctl
1	unlinkat
1	faccessat
1	fchmodat
1	lseek
1	readlinkat
1	socketpair
5049	total

PH3

calls	syscall
1349	epoll_pwait
1141	write
992	getuid
936	recvfrom
885	read
737	ioctl
482	futex
184	sendto
84	madvise
73	prctl
19	timerfd_settim
19	fstat
13	mmap
8	close
7	clock_gettime
7	writev
6	munmap
6	dup
5	mprotect
5	fcntl
4	pread64
4	rt_sigprocmask
2	uname
1	socketpair
1	clone
1	readlinkat
6971	total

EA1

calls	syscall
1693	clock_gettime
1126	write
1063	recvfrom
856	epoll_pwait
697	futex
637	ioctl
581	mprotect
573	getuid32
571	read
254	prctl
227	mmap2
197	rt_sigprocmask
158	sendto
150	pread64
98	fstatat64
95	munmap
91	fstat64
68	openat
66	close
51	sched_yield
38	writev
36	clone
15	timerfd_settim
15	faccessat
11	dup
11	rt_sigaction
10	fcntl64
7	_llseek
6	epoll_ctl
6	readlinkat
6	getsockopt
5	madvise
5	mkdirat
4	getdents64
4	fchmodat
3	lseek
3	ugetrlimit
3	unlinkat
3	getrandom
2	getpriority
2	fsync
2	mremap
2	fstatfs64
2	eventfd2
1	setrlimit
1	sysinfo
1	sigaltstack
1	timerfd_create
1	epoll_create1
1	socketpair
9459	total

EA2

calls	syscall
35380	clock_gettime
3821	write
2263	futex
2251	recvfrom
1568	sendto
1212	epoll_pwait
1012	read
700	getuid32
612	ioctl
70	madvise
70	prctl
50	sched_yield
31	rt_sigprocmask
14	timerfd_settim
12	pread64
11	gettimeofday
10	mmap2
7	fstat64
6	munmap
5	mprotect
3	fcntl64
2	clone
2	close
2	dup
1	socketpair
1	readlinkat
49116	total

EA3

calls	syscall
2469	write
1986	recvfrom
1261	sendto
1098	epoll_pwait
1081	futex
800	read
624	getuid
547	ioctl
155	clock_gettime
94	madvise
70	prctl
16	timerfd_settim
7	fstat
6	rt_sigprocmask
5	mmap
5	mprotect
3	fcntl
2	clone
2	pread64
2	close
2	dup
1	socketpair
1	readlinkat
1	munmap
10238	total

EG1

calls	syscall
926	write
810	recvfrom
666	epoll_pwait
615	mprotect
588	futex
564	ioctl
535	read
492	prctl
480	getuid32
235	mmap2
182	rt_sigprocmask
132	sendto
132	pread64
132	madvise
126	fstatat64
123	munmap
99	fstat64
82	clock_gettime
64	openat
57	close
28	clone
23	writev
19	faccessat
15	timerfd_settim
10	dup
9	fcntl64
7	_llseek
7	mkdirat
6	epoll_ctl
6	readlinkat
6	getsockopt
4	getdents64
4	fchmodat
3	ugetrlimit
3	unlinkat
3	getrandom
3	lseek
2	fsync
2	getpriority
2	mremap
2	fstatfs64
2	eventfd2
2	socketpair
1	setrlimit
1	sysinfo
1	uname
1	sched_yield
1	rt_sigaction
1	timerfd_create
1	epoll_create1
1	setsockopt
1	sendmsg
7217	total

EG2

calls	syscall
24226	write
19051	sendto
15019	recvfrom
11045	futex
4049	getuid32
4026	read
4023	ioctl
3999	epoll_pwait
851	madvise
403	prctl
155	fstatat64
129	mmap2
112	rt_sigprocmask
107	mprotect
94	pread64
82	clock_gettime
64	munmap
58	writev
47	fstat64
34	close
33	openat
20	clone
16	fcntl64
14	faccessat
14	timerfd_settim
8	_llseek
8	readlinkat
7	dup
6	mkdirat
4	getdents64
4	getsockopt
3	epoll_ctl
3	fstatfs64
3	getrandom
2	lseek
2	getpriority
2	mremap
2	rt_sigaction
2	ugetrlimit
2	fchmodat
2	socketpair
1	sysinfo
1	fsync
1	uname
1	sched_yield
1	unlinkat
1	timerfd_create
1	eventfd2
1	setsockopt
1	sendmsg
87740	total

EG3

calls	syscall
190458	clock_gettime
18618	write
11269	sendto
10110	recvfrom
7651	futex
2667	read
2666	epoll_pwait
2651	getuid32
2566	ioctl
219	madvise
73	prctl
11	gettimeofday
10	timerfd_settim
7	fstat64
6	rt_sigprocmask
5	mmap2
3	mprotect
3	fcntl64
2	pread64
2	dup
2	clone
2	writev
2	close
1	munmap
1	readlinkat
1	socketpair
249006	total

Figure A3-31. MW8 system call summaries (execution only)

PH1

calls	syscall
1084	write
842	recvfrom
552	epoll_pwait
551	read
506	futex
336	getuid
274	ioctl
178	sendto
41	fstat
32	rt_sigprocmask
30	prctl
24	mmap
19	timerfd_settim
18	pread64
13	munmap
7	clock_gettime
7	mprotect
5	openat
5	close
4	madvise
3	dup
3	writev
2	fcntl
2	clone
1	readlinkat
1	getrlimit
1	setrlimit
1	socketpair
1	sendmsg
4543	total

PH2

calls	syscall
1011	write
844	recvfrom
533	epoll_pwait
524	read
429	futex
320	getuid
247	ioctl
172	sendto
79	madvise
29	timerfd_settim
18	fstat
16	openat
16	prctl
15	close
9	mmap
9	newfstatat
4	clock_gettime
4	rt_sigprocmask
3	lseek
3	mprotect
2	dup
2	writev
2	munmap
2	clone
1	fcntl
1	uname
4295	total

PH3

calls	syscall
18110	write
10591	sendto
9880	ioctl
9001	epoll_pwait
8941	read
8214	recvfrom
4977	futex
2514	getuid
321	madvise
16	prctl
15	timerfd_settim
9	fstat
7	mmap
6	mprotect
4	writev
3	close
2	clock_gettime
2	rt_sigprocmask
2	dup
2	munmap
1	clone
1	fcntl
72619	total

EA1

calls	syscall
956	write
906	recvfrom
676	epoll_pwait
438	futex
429	read
364	getuid32
289	ioctl
233	clock_gettime
185	sendto
31	prctl
27	rt_sigprocmask
16	pread64
16	mmap2
13	timerfd_settim
10	fstat64
8	munmap
7	mprotect
7	madvise
3	clone
2	fcntl64
2	openat
2	close
1	dup
1	readlinkat
1	socketpair
4623	total

EA2

calls	syscall
304034	clock_gettime
35469	write
16410	recvfrom
15780	sendto
7254	futex
6244	epoll_pwait
6078	read
4224	getuid32
4148	ioctl
290	madvise
70	prctl
50	sched_yield
14	timerfd_settim
11	gettimeofday
7	fstat64
6	rt_sigprocmask
5	mmap2
5	mprotect
3	fcntl64
2	close
2	clone
2	pread64
2	dup
1	readlinkat
1	socketpair
1	munmap
400113	total

EA3

calls	syscall
11688	write
8685	recvfrom
8112	sendto
4762	futex
3231	epoll_pwait
2669	read
2297	getuid
2202	ioctl
374	madvise
221	clock_gettime
35	prctl
29	rt_sigprocmask
11	timerfd_settim
10	pread64
10	writev
9	mmap
6	fstat
5	munmap
4	close
2	clone
2	mprotect
2	dup
2	fcntl
44368	total

EG1

calls	syscall
29095	write
23022	sendto
18122	recvfrom
13188	futex
4752	getuid32
4707	epoll_pwait
4700	read
4571	ioctl
104	writev
88	prctl
57	madvise
27	rt_sigprocmask
25	clock_gettime
21	mmap2
16	pread64
15	fstat64
14	timerfd_settim
10	mprotect
9	munmap
3	clone
3	openat
3	close
2	fcntl64
1	dup
1	setrlimit
1	ugetrlimit
1	readlinkat
1	socketpair
1	sendmsg
102560	total

EG2

calls	syscall
22476	write
14942	sendto
13766	recvfrom
8440	futex
3773	getuid32
3725	read
3713	ioctl
3670	epoll_pwait
870	madvise
404	prctl
157	rt_sigprocmask
155	fstatat64
139	mmap2
112	pread64
107	mprotect
96	writev
82	clock_gettime
74	munmap
47	fstat64
34	close
33	openat
20	clone
16	fcntl64
15	timerfd_settim
14	faccessat
8	_llseek
8	readlinkat
7	dup
6	mkdirat
4	getdents64
4	getsockopt
3	epoll_ctl
3	fstatfs64
3	getrandom
2	getpriority
2	mremap
2	rt_sigaction
2	ugetrlimit
2	fchmodat
2	socketpair
2	lseek
1	eventfd2
1	sysinfo
1	fsync
1	uname
1	sched_yield
1	unlinkat
1	timerfd_create
1	setsockopt
1	sendmsg
76949	total

EG3

calls	syscall
189375	clock_gettime
18432	write
11207	sendto
10055	recvfrom
7681	futex
2667	read
2662	epoll_pwait
2631	getuid32
2552	ioctl
232	madvise
73	prctl
11	gettimeofday
7	fstat64
7	timerfd_settim
6	rt_sigprocmask
5	mmap2
4	mprotect
3	writev
3	fcntl64
3	close
2	dup
2	clone
2	pread64
1	munmap
1	readlinkat
1	socketpair
1	sendmsg
247626	total

Figure A3-32. MW8 system call summaries (50 events injected)