Indian Journal of Science and Technology, Vol 9(21), DOI: 10.17485/ijst/2016/v9i21/90273, June 2016
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

System Call Analysis of Android Malware Families

Sapna Malik* and Kiran Khatter

School of Engineering and Technology, Ansal University, Gurgaon – 122003, Haryana, India; sapnadhankhar@gmail.com, kirankhatter@gmail.com

Abstract

Background/Objectives: Now a days, Android Malware is coded so wisely that it has become very difficult to detect them. The static analysis of malicious code is not enough for detection of malware as this malware hides its method call in encrypted form or it can install the method at runtime. The system call tracing is an effective dynamic analysis technique for detecting malware as it can analyze the malware at the run time. Moreover, this technique does not require the application code for malware detection. Thus, this can detect that android malware also which are difficult to detect with static analysis of code. As Android was launched in 2008, so there were fewer studies available regarding the behavior of Android Malware Families and their characteristics. The aim of this work is to explore the behavior of 10 popular Android Malware Families focused on System Call Pattern of these families. **Methods/Statistical Analysis:** For this purpose, the authors have extracted the system call trace of 345 malicious applications from 10 Android Malware Families named FakeInstaller, Opfake, Plankton, DroidKungFu, BaseBridge, Iconosys, Kmin, Adrd and Gappusin using strace android tool and compared it with the system calls pattern of 300 Benign Applications to justify the behavior of malicious application. **Findings:** During the experiment, it is observed that the malicious applications invoke some system calls more frequently than benign applications. Different Android malware invokes the different set of system calls with different frequency. **Applications/Improvements:** This analysis can prove helpful in designing intrusion-detection systems for an android mobile device with more accuracy.

Keywords: Android Kernal, Android Malware Installation Methods, Malware Families, System Call Analysis

1. Introduction

Being an open-source operating system, Android Operating System is more vulnerable to attacks. In the android market, the pervasion of malicious applications is rampant, and it is on the rise like never before. The intruders can develop malicious applications faster with a tool like App Inventor and launch their malwares in the market more quickly[1]. The Intrusion-Detection System for mobile device should opt for a mechanism for detecting the malicious and normal applications accurately. The Intrusion-detection system for mobile devices is based on two techniques of analysis – static and dynamic[2]. The static analysis technique is based on the reverse engineering of .apk(android application package) file of android applications. It includes the exploration of AndroidManifest.xml and classes.dex files for malicious codes without installing and executing the app. The dynamic analysis scrutinizes the behaviors of the application during its execution. In the work, the intrusion detection system for an android device proposed[3] which analyzed process lists, system call traces, symbol table, a list of open files and network traffic features for intrusion detection. The behavior graph of the malicious application created[4] by tracking dependencies among system calls of six malware families Allaple, Bagle, Mytob, Agent, Netsky. In research work System Call Short Sequence Birthmark and Input Dependent System Call Subsequence Birthmark, two systems call based software birthmarks proposed for detection of malicious application behavior of 1600 malicious applications[5]. The diverse nature of system calls invoked by different applications has been identified and the research work has proposed a system-centric approach for malware detection[6]. The analysis of the application log and system call log of 230 applications has been done in the Research Work[7]. CopperDroid[8] analyzed low-level OS-specific

*Author for correspondence

and high-level Android-specific behaviors. The authors[9] analyzed the behavior of two malware families- Super History Eraser and Task Killer Pro, in terms of the system call generated by simulating the application with user interfaces events. The approach[10] of malware analysis is based on two features- permission and system call. In this work, the dataset of 400 applications, 200 benign applications, and 200 malicious applications has been taken. The system calls features[11] have been used for detection of malicious web pages. In this proposed work, the individual system calls invoked and sequence of the system call invoked by malicious javascript have been analyzed. A technique of hook system calls and binder driver function based malware detection has been proposed in the work[12]. The comparison between security of Android OS and IOS has been done in the work and it proposes the UAS (User Access Security) framework for providing the mobile user more control over his resources at the execution time through permission[13]. Malware Detection and Elimination using Bayesian Technique and Nymble Algorithm has been proposed for securing the Delay Tolerant Network[14.] The proposed work was the integration of Honeypot Technology, Intrusion Detection Systems and Malware Analysis in Windows based platform for Botnet research[15]. The system call, a dynamic feature of the android application is an effective feature for intrusion detection in the android device. We need to analyze plenty of android applications to understand the behavior of malicious applications.

In this paper, we have explored the system call pattern of 10 malware families FakeInstaller, Opfake, Plankton, DroidKungFu, BaseBridge, GinMaster, Iconosys, Kmin, Adrd and Gappusin and compare it with the system call pattern of normal application by extracting the events of 300 normal apps and 345 Malicious apps from different malware families, using Android-based event analysis tool, Strace.

2. Android Operating System Architecture

Android, a popular Linux-based mobile operating system developed by Android Inc. in 2005 which was later bought by google. In 2007, Google, Open Handset Alliance (OHA) and other device manufacturers such as HTC, Sony and Samsung, wireless carriers and chip makers collaborated on Android design, development and distribution and launched the first android based smartphone HTC Dream in October 2008. Android Operating system is based on Linux 2.6 Kernel which was modified to run effectively and efficiently on computational and energy-constrained mobile device. Many Libraries and drivers of Linux 2.6 kernel were modified, and newer ones were added to enable android run on the mobile device. Android Community had developed its own C library named Bionic java and Android specific java runtime engine Dalvik Virtual Machine and made a complete software stack of operating system, middleware components and application framework for mobile devices. Figure 1 shows Android Architecture where the lower layer is an android operating system that provides all the operating system functionality like process management, memory management, network functionality and the device drivers. The middle layer comprises of Bionic C library and Android Runtime for providing most of the android specific functionality. The Android Runtime has Dalvik Virtual machine and core libraries for facilitating the execution of the android application by converting the java bytecode of the application to the dex executable, a DVM specific bytecode.

The application framework holds API (Application Programming Interface) interface. In this layer, the activity manager governs the activities of android application and monitors its life cycle. The Content Provider facilitates sharing of data among applications. The Resource Manager manages non-code services of the android application and Notification Manager raises the custom alerts. The topmost layer is User and Built-in Applications.

2.1 Android Kernel and System Calls

To reiterate, the Android has modified Linux 2.6 Kernel at the core. The modifications are done for adapting this operating system for the mobile devices. The Android Specific kernel enhancement includes power management, shared memory drivers, alarm drivers, binders, kernel debugger and logger and low memory killers.

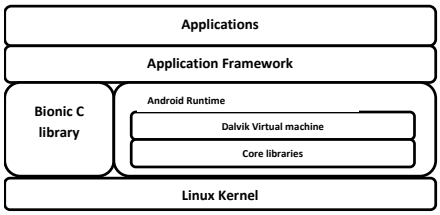

Figure 1. Android operating system architecture.

The android application takes the services of the kernel through the system calls. Whenever a user request 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 Dalvik Virtual Machine in 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. The Kernel Invocation calls are sub-grouped into three types of System Calls. 1) System Call- used to invoke native operations of the kernel. 2) Binder Call- for invocation of the binder drivers in the kernel. 3) The Socket Call - allows the read/write/send/receive operations of Linux socket. In this analysis, we will consider all these subgroups as system calls. There are 250 types of System Calls in Android Operating system for performing operations like allocating resources, performing read/write operations, protecting critical data, etc. As discussed above, the system calls are the interface between the user and the kernel. This means all requests from the applications will pass through the System Call Interface before its execution through the hardware. So capturing and analyzing the system call can give information about the behavior of the application.

3. Categorization of Android Malware

Android Malware is categories on the basis of how they install themselves on the victim's device, how they are triggered, and what type of risks associated with them. The following section will discuss their installation method, their activation triggers and types of malicious activity they perform.

3.1 Installation Method

Android malware families are categorized by their installation method on victim's mobile. The following are the methods used by the android malicious application for its successful installation.

- **Repacking** – Repacking is the method of modifying and repackaging the apk file of registered android benign application of Android application market and redistributing it. The modified code contains code for stealing personal or financial information and causing damage to the device, but this method is easy to be exposed just by doing static analysis of code. e.g. GoldDream.
- **Update attack**- The technique is inspired by the first technique but in this technique instead of embedding the entire malware code, it just includes update component and with this update component, the entire malicious code is downloaded and installed on the host mobile phone at the run time. This type of installation cannot be exposed by static analysis of the infected application. For e.g. Malware families like BaseBridge, DroidKungFuUpdate, AnserverBot, and Plankton are used this technique.
- **Drive-By-Download attack**- This is the traditional approach used by the malware families for their successful installation on the host mobile by luring the user through advertisements and announcing their application as an interesting and feature-rich application. e.g. GGTracker, Jifake, Spitmo and ZitMo are malware families, which use this installation method.
- **Standalone**- In this category the application announces itself as a spyware but does some malicious activities, or the application pretends to be a benign application but having some malicious code, or the application requires root authorization for doing malicious activities.

3.2 Activation

In this category the application activates itself for doing malicious activities. The authors[16] found in their analysis that the malicious applications activate themselves during boot completion events, package removing or package installation events, SMS received events, call phone, during network connectivity events, during system events and during the launch of some popular host applications' main activity.

3.3 Malicious Payload

This category elaborates purpose of the malicious application. There are four types of malicious payloads. The first is Privilege Escalation in which malicious application tries to take root access to exploit the root privilege. In

the second type, the malicious application uses victim's device as a bot by controlling it from a remote device. In the third type of malicious payload, the malicious application uses victim's device for sending premium messages without user's knowledge in order to harm him with financial loss. Collecting user's information like SMS received, phone call log, and device information is the purpose of the fourth kind of malicious payload.

4. System Call Analysis of Android Malware Families

For doing a system call analysis of android malware applications this experiment has taken the dataset of malicious applications from Drebin project[17] and from Androtracker Project[18] for the Normal application. First, we installed the application on the android emulator then we simulated the application with the monkey tool by passing the package name as a parameter. The monkey tool enables execution of the android application with raising the test events. After successful running of the monkey tool, we had recorded traces of the system call invoked by the malicious and normal application through strace. After executing strace command over the process of applications, we received log file of the system calls invoked by the application under consideration. The log file has the system call name, a time stamp at which it is invoked and the signals of the process. We further processed the log files to get the count of a particular system call invoked by the process of malicious code. We have analyzed 345 malicious applications from 10 android malware families named FakeInstaller, Opfake, Plankton, DroidKungFu, BaseBridge, Iconosys, Kmin, Adrd, Gappusin and 300 Normal Applications. Table 1 shows the characteristics of Malware Families and the no. of instances of each family used for the analysis. Now we will discuss each family one by one with its observation and findings during the experiment.

4.1 FakeInstaller

FakeInstaller is the most common and hard to detect malware family. FakeInstaller uses the method of Repackaging for its installation by inserting their random class and method names into malicious application in the encrypted form. During the run time, it decrypts the string that has the location of the method which it wants to call. Fakeinsataller is hard to detect with static analysis of infected application code. The intention of Fake installer is to harm the user with a financial loss by sending SMS to premium services owned by the malware authors. During the analysis of the system call trace, the behavior of its sending SMS to premium services comes forth prominently. During the analysis, it is observed that the system calls sendto(), recefrom() which are used for sending and receiving data from the socket are heavily invoked. Further, the process control related system call like ptrace() is used for process tracing and controlling the other processes, and the sigprocemask() is used for blocking signal to the process, wait4(), futex, getpid() for getting process id, getuid() for getting user id of the owner of the process, prctl() for controlling execution of the process, are also heavily used. In the observation, it is also found that the malware also executes the system call related to writing and reading data from the files stored on phone and SD memory like write(), read(), ioctl(), fcntl64(), stat64(), close(), open(), mmap(), munmap(), lseek(), dup() etc.

4.2 Opfake

OpFake is the second most popular Trojan-SMS named on Opera Mini Mobile Browser for being fake downloader of it. Opfake uses the Repacking installation method and sends SMS with SIM data, downloads other malicious applications and stores them on the SD Card. The attacker first creates a fake website to lure the customers to download Opera Mini Browser and names their apk files similar to opera. It heavily uses PTrace(), SigprocMask(), Futex(), Clock(), GetUID(), GetPID() system calls for affecting execution of other process and for successful execution of their process. It also uses Recvfrom(), SendTo() system calls for sending SMS to premium services owned by malware author. It also invokes Write(), Read(), Close(), Dup(), mkdir(), chmod() for writing and reading the data of SD card.

4.3 Plankton

This malware family uses the update methodology for its propagation. The malware uses the dynamic code loading thus, they are difficult to trace with static code analysis and manual introspection of code. This malware sends device information to remote services enabling the malware author to control the device remotely. The malware also installs supporting malicious codes by downloading it through the HTTP and stores them on the SD card. During dynamic analysis of the system calls it is observed

Table 1. No. of malware family samples and their Characteristics

Malware Type	No. of Sample Application	Type	year	Installation Method	Activation methods	Privilege Escalation	Remote Control	Privacy Leaks	Malicious Activities
FakeInstaller	80	Trojan SMS	2012	Repackaging	-	No	Yes	No	Send SMS, Process SMS, Delete SMS
Opfake	67	Trojan SMS	2011	Repackaging	Boot Complete	No	Yes	No	Send SMS, Delete SMS, Process SMS, Send Device Data to remote Server, Download, install, delete package
Plankton	65	Trojan	2011	Update, Repackaging	OnCreate()	No	Yes	Yes	Send device information, Check User Internet Browsing activity
DroidKungFu	57	Trojan Spy	2011	Repackaging	Boot Complete , Bettery Status, System Events	Yes	Yes	Yes	Send Device Information ,Network information, Phone data, SD card Data to Remote Server
BaseBridge	30	Trojan spy	2011	Repackeging, Update	Boot Complete	Yes	Yes	Yes	Send SMS, Delete SMS, Process SMS, Send Device Data to remote Server, Download, install, delete package, Dial Phone Numbers, terminate process
GinMaster	28	Trojan Spy	2011	Repackaging	Boot Complete	Yes	Yes	Yes	Send Device Information, Installed Apps Information and Network Information to remote server.

Iconosys	13	Trojan spy	2012	Standalone	Boot Complete	No	Yes	Yes	Send Confidential Data to Remote Server and do SMS on Designated No.
Kmin	13	Trojan		Standalone	Boot Complete	No	Yes	Yes	Send Device Information to Remote Server.
Adrd	10	Trojan Spy	2011	Repackaging	Boot Complete, Network Configuration Change, Call phone	No	Yes	Yes	Send Device Information, Network Status to the Remote Device, can Install other apps
Gappusin	10	Trojan	2012	Update	-	No	Yes	Yes	Send Wi-Fi ,network related Confidential Data to Remote Device to enable the device easy for hacking.

that this malware family invokes the system calls socket(), connect(), sendto(), recvfrom(), socketpair() heavily for establishing a network connection and sending device information to remote server and downloading malicious code. It is also found that the malware of this family also uses access(), umask(), chmod(), rename(), mkdir(), read(), write() file operation related system calls for storing its malicious code on SD Card.

4.4 DroidKungFu

DroidKungFu is a dangerous malware family that aims to take root privilege for controlling user device and sends its confidential data to the remote server causing privacy leaks of the user. The malware author takes control of the device and uses it for malicious activities by making it a BOT. This malware family bundles its code in the popular android application for spreading itself. The malware of this family activates itself at boot complete and at battery status intents to do malicious activities craftily. The malware of this family poses medium risk[19]. This malware family was detected in 2011 and targeted the Chinese market. The hazardous behavior is depicted in the system call trace also as this malware can invoke the root authorized system calls like fchown32() for changing the owner of the files having the critical data, umask() for changing the read, write and execute permission on executable files, flock() for tinkering with lock on files, fork() for creating new process and pipe() for doing inter-process communication. The experiment shows the heavy use of system calls like ptrace(), write(), read(), futex(), recevfrom(), sendto() etc. for controlling process, sending and receiving messages to remote server and reading and writing data on the memory.

4.5 BaseBridge

BaseBridge, discovered in 2011 is one of the most hazardous and difficult to control malware family. BaseBridge damages the user mobile device from all possible directions and has a number of variants. This malware family takes control of the device by taking root privileges and sends critical data like IMSI, IMEI, OS version, phone number, device information by invoking the read() system call for reading critical files and sending it to the remote server through HTTP connection by using sendto() system call and acts as a scout for malware author. These also have the permission of sending SMS, Blocking SMS, deleting SMS and can do phone calls, thus harms the user financially also. This malware silently installs a BOTNET application for SMS charging. The different variants of this malware family use both repackaging and update for their

installation and get activated on Boot Complete, Battery Status and on system events. In the experiment, this malware has been found spying by changing the permission of memory protected region by invoking the mprotect() system call and by changing the permission of critical files by chmod() and umask() system call and by performing read() and write() operation on the files of user. This malware also tries to get memory status and information about other processes by invoking madvise(), getpid() and getuid(). This malware also invokes the system calls mmap(), munmap(), fdatasysc() for updating data of the memory.

4.6 Ginger Master

Ginger Master is a Trojan spy malware which forges SQLite database having information about user phone number, IMEI number, Android version, List of installed applications and Network information, and sends that to the remote server. It has root access capability and can install and uninstall the malicious application without user permission. It uses the method of repackaging by bundling it code into the popular application. During the experiment, the spy nature of Ginmaster has prominently surfaced. There are multiple calls to the system calls like getuid(), getpid(), futex(), wait4(), ioctrl(), read(), epoll(), cacheflush(), write() for the purpose of gathering information about user process and files. There are also multiple calls to the system calls sendto() and recefrom() for sending and receiving data to/from malware remote server.

4.7 Iconosys

Iconosys is another standalone Trojan spy discovered in 2012. This malware family triggers its methods on boot complete and runs in the background silently to sniff the online activities of the user, device location, text messages, user contact list, credit card details and sends this confidential information to malware author. It has the permissions to read, write and send sms to the designated number. This also has phone call capability and can harm the user financially also. This can write and read the device storage. The analysis of system calls shows the malicious activities of controlling user process by invoking ptrace(), sigpromask(), futex(), ioctrl(), epoll() and wait4() multiple times. This also shows the multiple invocations of system calls recevfrom(), sendto() for sending data to the remote server.

4.8 Kmin, Adrd and Gappusin Malware Families

Kmin, Adrd and Gappusin malware families were discovered in 2011. Kmin is a standalone installation based malware family which is activated at boot complete event. This type of malware takes control of user device and sends SMS to premium services, blocks user SMS and sends user private data to the remote servers. Adrd, a Trojan spy bundles its code in the benign application for breaching the privacy of user by sending confidential data to the remote server through HTTP connection. This is activated on Boot complete and also has the capability of calling phone. Gappusin is also an SMS Trojan who sends device information like IMEI number, IMSI, OS to the remote server and gives the control of the device to a remote server. It also sends SMS, Blocks SMS. During the experiment, this malware also shows the multiple times invocations of system calls from the process-related calls, file structure related calls and communication-related system calls. Table 2 shows the frequency of different system calls invoked by these malware families.

In this experiment, we have also observed the system call traces of 300 benign applications of Androtracker[13] project. In the experiment, we have observed that no. of Malicious Applications that invoke a system, the call is much more than the no. of benign applications. Figure 2 shows the comparison of system calls invoked by the benign applications and malicious applications. Moreover, there is a huge difference between the maximum frequency of invoking a system call in the benign application and malicious application. The malicious applications are used to invoke the system call more frequently and multiple times than benign applications. Figure 3 shows the maximum frequency of invoking a system call invoked by the benign and malicious application.

5. Conclusion

System call analysis is an effective technique for detecting the Android Malware as in this technique; we do not require bytecode of the malicious application for doing the analysis. In this paper, we capture the system call traces of malicious applications during execution of the malicious application and analyze it. It is observed that the malware application invokes the system calls more frequently than the benign android application. The most frequently invoked system calls are trace(), sigproc-

Table 2. Frequency of invoking system call by malware families

Type of system call	Description	Fake Installer	Opfake	Plankton	DroidKungFu	BaseBridge	GinMaster	Iconosys	Kmin	Adrd	Gappusin
PTRACE	process trace	39384	43561	1915	12015	2957	308	3148	5086	188	0
SIGPROCMASK	examine and change blocked signals	27942	24915	1207	7544	2259	341	2519	3650	502	350
CLOCK	determine processor time	6617	4753	10349	4650	823	8388	8180	12608	12608	1760
RECVFROM	receive a message from a socket	4566	3640	7357	3856	567	6785	6189	5000	5000	1405
WRITE	write to a file descriptor	4167	1973	1807	3796	1221	2514	1899	4006	2159	594
WAIT4	wait for process to change state, BSD style	4034	1825	604	3764	1066	108	1135	1825	64	0
SENDTO	send a short message on a socket	2762	2053	4467	2451	308	4214	3726	2682	2682	863
MPROTECT	set protection on a region of memory	1684	1511	1498	2191	699	834	1342	1678	1678	815
FUTEX	fast user-space locking	5643	14534	2905	2157	5125	2899	1330	5807	5807	4576
IOCTL	control device	2147	1721	3811	1738	360	2496	2643	2321	2321	778
FCNTL64	manipulate file descriptor	108	70	136	1525	177	100	254	67	937	25
GETPID	get process identification	1796	1181	1221	1387	272	2061	1204	2546	2546	548
GETUID32	get user identity	1371	1187	1871	1114	208	1472	1042	1719	1719	322
EPOLL	I/O event notification facility	1314	1190	1543	1006	216	1474	1171	1661	1633	291
CACHEFLUSH	flush contents of instruction and/or data cache	718	632	576	932	258	340	469	487	487	271
READ	read from a file descriptor	4517	1261	1331	904	173	1411	971	1347	1154	333
STAT64	get file status	86	142	200	778	70	269	113	160	222	54
FSTAT64	get file status	12	43	99	770	76	45	141	28	384	22
GETTIMEOFDAY	get / set time	32	96	465	765	140	50	261	47	354	5

ACCESS	check user's permissions for a file	26	42	93	426	45	42	71	51	176	37
PREAD	read from or write to a file descriptor at a given offset	106	108	259	408	74	157	134	86	237	62
WRITEV	read or write data into multiple buffers	443	479	648	384	52	431	517	519	519	271
UMASK	set file mode creation mask	0	0	6	382	38	16	62	2	244	0
CLOSE	close a file descriptor	326	219	346	352	92	200	122	187	196	115
OPEN	open and possibly create a file	12	51	74	347	46	51	63	61	195	20
MMAP2	map files or devices into memory	234	363	320	298	98	308	112	349	349	239
MUNMAP	map or unmap files or devices into memory	221	331	263	227	68	217	87	310	310	199
MADVISE	give advice about use of memory	159	239	236	226	44	141	42	272	226	163
FCHOWN32	change ownership of a file	0	1	3	191	19	8	31	1	122	0
PRCTL	operations on a process	151	186	153	151	20	25	19	266	179	102
BRK	change the amount of space allocated for the calling process's data segment	135	76	170	139	126	38	171	230	230	141
LSEEK	reposition read/write file offset	348	18	116	118	0	6	0	18	2	0
DUP	duplicate a file descriptor	166	107	137	71	33	89	61	123	123	51
GETPRIORITY	get/set program scheduling priority	41	17	26	52	4	15	3	14	50	4
PIPE	create pipe	0	0	0	40	0	0	0	0	0	0
CLONE	create a child process	8	16	41	36	15	45	13	14	17	17
FSYNC	synchronize a file's in-core state with storage device	2	0	43	23	5	8	0	1	9	0

GETDENTS64	get directory entries	2	12	302	20	0	8	2	14	0	0
GETTID	get thread identification	4	15	9	14	10	3	13	3	4	0
LSTAT64	get file status	2	9	45	13	12	7	7	10	9	2
FORK	create a child process	0	0	0	10	0	0	0	0	0	0
NANOSLEEP	high-resolution sleep	10	24	9	9	30	21	2	62	3	1
RECVMSG	receive a message from a socket	2	6	0	8	0	2	0	8	0	0
CHMOD	change permissions of a file	3	3	43	7	10	8	8	0	10	2
SENDMSG	send a message on a socket	1	8	4	7	0	5	0	9	0	0
FLOCK	apply or remove an advisory lock on an open file	0	5	0	6	0	2	0	6	0	0
MKDIR	create a directory	1	6	1	6	3	3	1	6	1	2
CONNECT	initiate a connection on a socket	1	1	1	4	0	1	0	3	0	0
POLL	wait for some event on a file descriptor	0	2	1	4	0	1	0	3	0	0
RENAME	change the name or location of a file	1	0	43	4	5	8	0	2	9	0
SETPRIORITY	get/set program scheduling priority	0	3	3	4	2	0	4	0	0	0
SETSOCKOPT	get and set options on sockets	20	4	1	4	0	4	0	7	12	0
SOCKET	create an endpoint for communication	1	1	1	4	0	1	0	3	0	0
UNLINK	call the unlink function to remove the specified file	1	0	43	4	7	7	2	1	9	0
GETSOCKOPT	get and set options on sockets	0	0	0	3	0	0	0	0	0	0
BIND	bind a name to a socket	0	0	0	2	0	0	0	0	0	0

FTRUNCATE	truncate a file to a specified length	0	0	0	2	0	0	0	0	0	0
GETSOCKNAME	get socket name	0	0	0	2	0	0	0	0	0	0
INOTIFY	monitoring filesystem events	0	1	1	2	0	1	0	1	0	0
RESTART	restart a system call after interruption by a stop signal	0	0	1	2	0	1	0	0	0	0
SCHED	overview of scheduling APIs	2	2	3	2	2	7	1	2	1	1
GETRLIMIT	get/set resource limits	0	1	1	1	1	0	1	0	0	0
LGETXATTR	retrieve an extended attribute value	0	1	0	1	0	0	0	1	0	0
READLINK	read value of a symbolic link	0	1	1	1	0	0	1	0	0	0
SOCKETPAIR	create a pair of connected sockets	2	1	2	1	0	1	1	1	3	0
STATFS64	get filesystem statistics	2	2	1	1	3	3	2	2	2	0
FDATASYNC	synchronize a file's in-core state with storage device	0	0	0	0	35	6	15	0	190	0
GETPPID	get parent process identification	0	0	1	0	0	0	1	0	0	0
KILL	terminate a process	1	0	0	0	0	0	0	0	0	1
PWRITE	read from or write to a file descriptor at a given offset	0	0	0	0	70	13	30	0	555	0

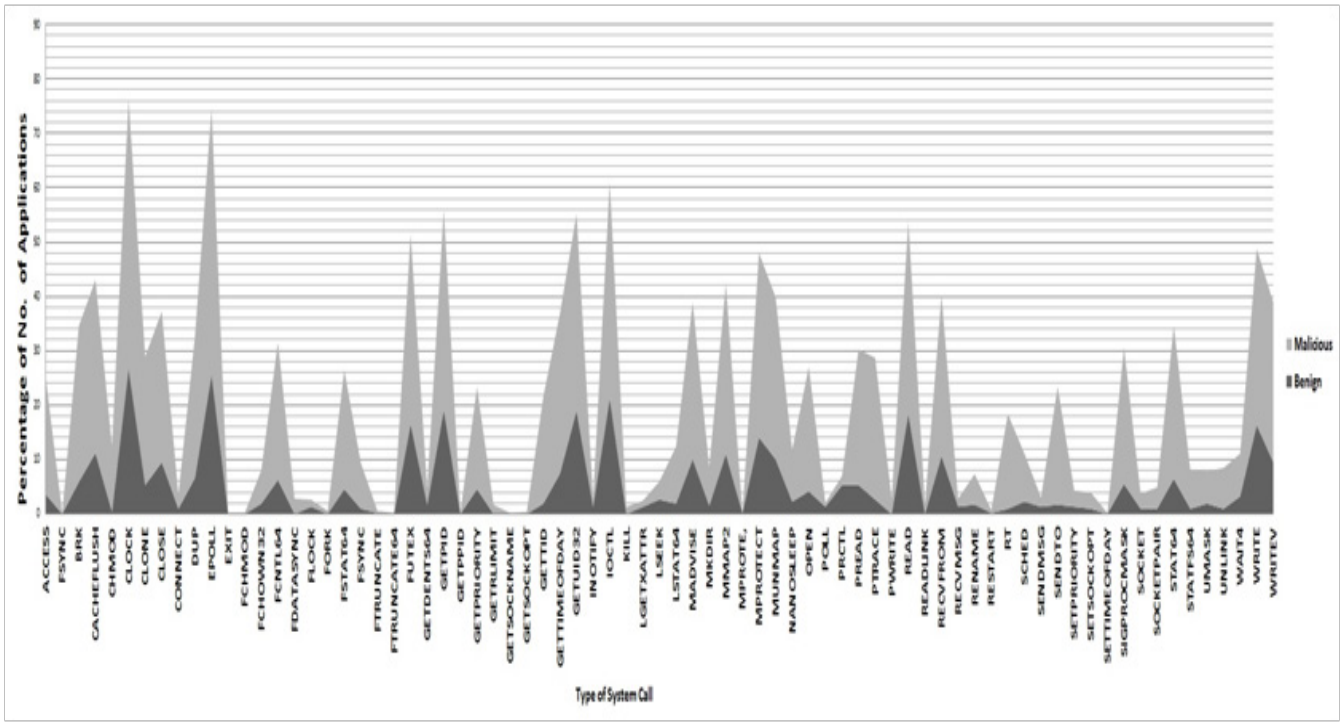

Figure 2. No. of Applications invoked a system call.

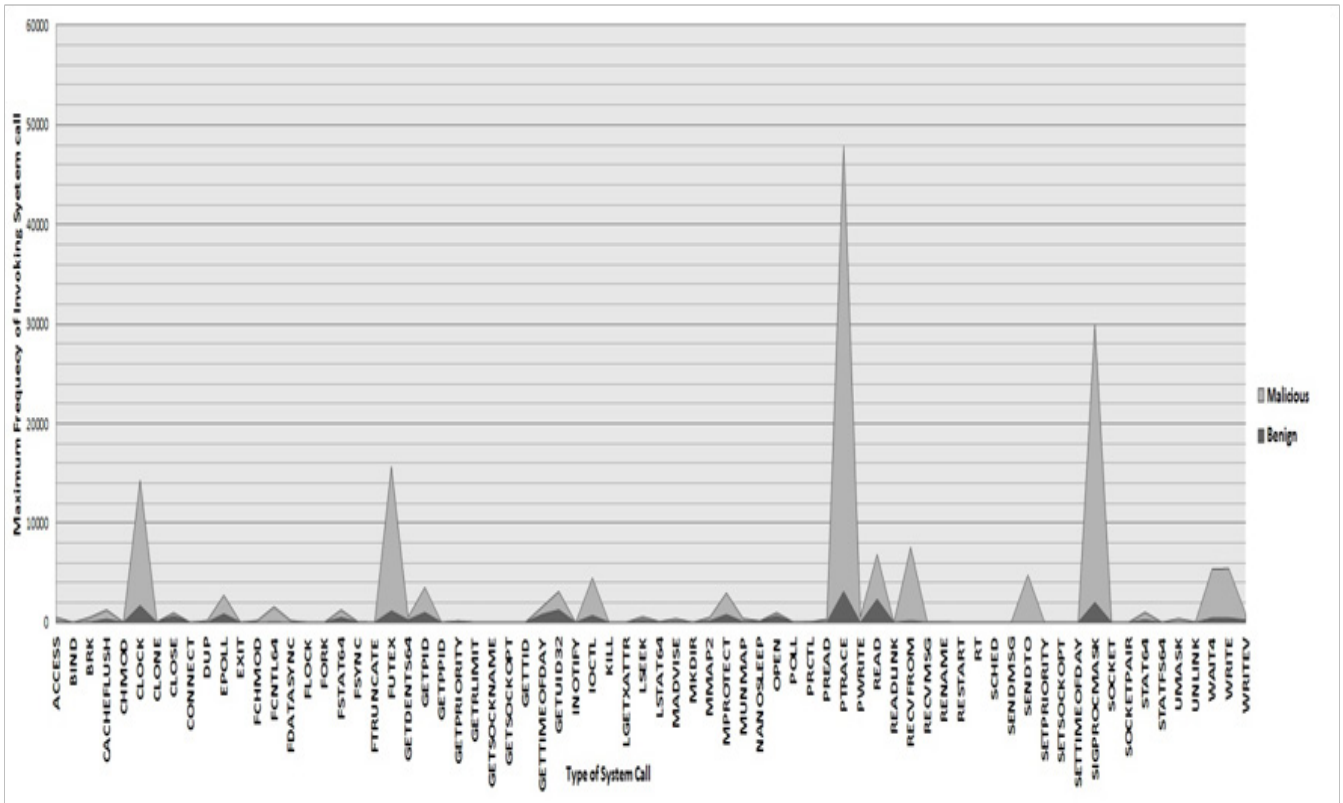

Figure 3. Maximum frequency of invoking a system call by malicious vs benign application.

mask(), futex(), recevfrom(), sendto(), write(), read(), clock(), wait4(), ioctl(), mprotect(), getpid(), getuid(). We can combine the System Call Trace and other features like Method call and API call for efficient and effective Intrusion-Detection System for Android Mobile Device.

6 References

1. Kang H, Cho J, Kim H. Application study on android application prototyping method using App inventor. Indian Journal of Science and Technology. 2015 Aug; 8(18):1–5. DOI: 10.17485/ijst/2015/v8i18/75919.
2. Jerlin MA, Jayakumar C. A dynamic malware analysis for windows platform - a survey. Indian Journal of Science and Technology. 2015 Oct; 8(27):1–5. DOI: 10.17485/ijst/2015/v8i27/81172.
3. Schmidt AD, Schmidt HG, Clausen J, Yuksel KA, Kiraz O, Camtepe A, Albayrak S. Enhancing security of Linux-based android devices. Proceedings of 15th International Linux Kongress; 2008. p. 1–16.
4. Kolbitsch C, Comparetti PM, Kruegel C, Kirda E, Zhou X, Wang XF. Effective and efficient malware detection at the end host. Proceedings of the 18th conference on USENIX security Symposium; 2009. p. 351–98.
5. Wang X, Jhi V, Zhu S, Liu P. Detecting software theft via system call based birthmarks. Proceedings of the Computer Security Applications Conference; 2009. p. 149–58.
6. Lanzi A, Balzarotti D, Kruegel C, Christodorescu M, Kirda E. Accessminer: using system-centric models for malware protection. Proceedings of the 17th ACM conference on Computer and Communications Security; 2010. p. 399–12.
7. Isohara T, Takemori K, Kubota A. Kernel-based behavior analysis for android malware detection. Proceedings of Seventh International Conference on Computational Intelligence and Security(CIS), Hainan; 2011. p. 1011–15.
8. Reina A, Fattori A, Cavallaro L. A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors. Proceedings of Euro Sec'13; 2013. p. 1–6.
9. Tchakounte F, Dayang P. System calls analysis of malware on android. International Journal of Science and Technology. 2013; 2(9):1–6.
10. Canfora G, Mercaldo F, Visaggio CA. A classifier of malicious android applications. Proceedings of the 2nd International Workshop on Security of Mobile Applications, in conjunction with the International Conference on Availability, Reliability and Security, (ARES), Regensburg; 2013. p. 607–14.
11. Canfora G, Medvet E, Mercaldo F, Visaggio CA. Detection of malicious web pages using system calls sequences. Proceedings of the 4th International Workshop on Security and Cognitive Informatics for Homeland Defense (SeCIHD 2014); 2014. p. 226–38.
12. Jeong Y, Lee H, Cho S, Han S, Park M. A kernel-based monitoring approach for analyzing malicious behavior on android. Proceedings of the 29th Annual ACM Symposium on Applied Computing; 2014. p. 1737–38.
13. Ahmad DM, Javed P. Security comparison of android and IOS and implementation of User Approved Security (UAS) for Android. Indian Journal of Science and Technology. 2016 Apr; 9(14):1–7. DOI: 10.17485/ijst/2016/v9i14/87071.
14. Jeyaseelan WRS, Hariharan S. Malware detection and elimination using Bayesian technique and nymble algorithm. Indian Journal of Science and Technology. 2015 Dec; 8(34):1–7. DOI: 10.17485/ijst/2015/v8i34/85244.
15. Sathish V, Khader PSA. Deployment of proposed botnet monitoring platform using online malware analysis for distributed environment. Indian Journal of Science and Technology. 2014 Jan; 7(8):1087–1093. DOI: 10.17485/ijst/2014/v7i8/48583.
16. Zhou Y, Jiang X. Dissecting android malware: characterization and evolution. Proceeding of IEEE Symposium on Security and Privacy, San Francisco: CA; 2012. p. 95–109.
17. Arp D, Spreitzenbarth M, Malte H, Gascon H, Rieck K. Drebin: effective and explainable detection of android malware in your pocket. Symposium of Network Distribution System and Security; 2014. p. 23–6.
18. Kang H, Jang J-W, Mohaisen A, Kim HK. Detecting and classifying android malware using static analysis along with creator information. International Journal of Distributed Sensor Networks-Special issue on Advanced Big Data Management and Analytics for Ubiquitous Sensors; 2015. p. 7.
19. Grace M, Zhou Y, Zhang Q, Zou S, Jiang X. Risk ranker: scalable and accurate zero-day android malware detection categories and subject descriptors. Proceeding of 10th International Conference Mobile System Application Services; 2011. p. 281–94.