

SYSTEM AND METHOD FOR DETECTING POLICY-VIOLATING APPLICATIONS

Field of the Invention

This invention relates to a system and method for detecting policy violating applications. In particular, the invention utilizes features extracted from a selection of policy violating applications to derive a machine learning algorithm that is then employed to detect new policy-violating applications.

Summary of the Prior Art

As modern communication devices become more widely adopted by the masses, mobile devices such as smartphones, tablets, netbooks, and etc. are beginning to replace the function performed by desktops and notebooks in a user's daily life. Hence increasingly, a greater number of software applications have been developed and released for these mobile platforms. As a result, a mobile device user now has the freedom to choose and install different software applications onto their device, thereby allowing the user to customize their mobile device as they wish. However, while there are many reliable and secure software applications available on the market, the ability to interact, install, and operate third party software inevitably leaves a mobile device vulnerable to attacks from malicious sources. These attacks may be in the form of malware, backdoor applications, adware and other such similar types of harmful software applications.

In order to address the threat of malicious applications, various security-related applications have been developed. These applications which are typically installed on a mobile device, allow the device's user to assess the threat level of an application before the user makes the decision whether the application is to be installed onto their device or not. However, these security-related applications are often device-specific, operating system-specific, or application-specific. Hence, there is a need for a universal security-related application for efficiently scanning and defending mobile devices from the ever evolving malicious applications.

One problem faced by security-related application developers is that on a daily basis, many different types of data and applications are developed and released for such devices. While service providers are able to moderate the download rates of these applications and application stores are able to moderate the upload/download rates of these third party applications, there is no way to effectively monitor the behavior of these applications after they have been installed on a user's mobile device. As a result, it is difficult to identify new, previously unknown malicious applications solely by their behavior.

Consequently, existing online application stores become infected with various types of bad applications (“apps”) such as malware apps, privacy-violating apps, and repackaged apps. The online application stores have been trying to remove these apps on a daily basis with the help of anti-malware technologies, and inputs from researchers and users. In particular, those skilled in the art have tried to understand the behavior of malware apps’ by painstakingly collecting real-life malware app samples. Based on their work, they have derived various prevention or detection techniques for malware apps, privacy-violating apps and repackaged apps. However, many apps behave in a manner that is undesirable, and yet less serious than existing malware apps. For example, some apps automatically redirect users to share the user’s experience about the apps on social media while some apps simply generate spam that are then posted unknowingly on the user’s social media accounts or sent to the user’s contacts. Some apps simply lack proper functions and qualities. Most of these apps violate the application store’s policies but little has been discovered about the overall properties of these apps.

For the above reasons, those skilled in the art are constantly striving to come up with a system and method for detecting applications that violate an application store’s policies whereby the detection method and system may be updated as the application store’s policies are updated.

Summary of the Invention

We provide systems and methods to improve the efficacy of detecting applications that violate the policies of an application store as set out by the embodiments in accordance with the invention.

A first advantage of embodiments of systems and methods in accordance with embodiments of the invention is that the proposed system and method may detect applications with policy violating behaviours such as applications that violate intellectual property rights, applications that cause a user to inadvertently generate spam, and/or applications that automatically redirects a user to a questionable server.

A second advantage of embodiments of systems and methods in accordance with embodiments of the invention is that as the application store’s policies are updated, the system and method for detecting policy violating applications may be easily updated as well. This is because the embodiments of the invention allow for a continuous learning loop to be applied so that the detection means may be constantly updated with the application store’s latest policies.

A third advantage of embodiments of systems and methods in accordance with embodiments of the invention is that the invention does not require a list of security rules to be maintained and that the invention may be used to detect both malware and applications that exhibit unscrupulous behaviors.

According to a first aspect of the invention, a computer server for detecting policy violating applications comprises a processor and a non-transitory media readable by the processor, the media storing instructions that when executed by the processor, cause the processor to obtain bad applications from an application store; generate a feature table based on identities of the bad applications, a blacklist, a permission list, an Application Programming Interface (API) list and a metadata list; obtain parameters for bad applications using the feature table; generate a training dataset based on the bad applications and their associated parameters, and good applications and their associated parameters; and train a supervised machine learning algorithm using the training dataset, wherein the trained supervised machine learning algorithm is used to detect policy violating applications.

With reference to the first aspect, in a first possible implementation manner of the first aspect, the instructions to obtain bad applications from the application store comprises: instructions for directing the processor to: retrieve applications from the application store; and determine applications that violate policies of the application store, and classify applications that violate policies of the application store as bad applications.

With reference to the first possible implementation manner of the first aspect, in a second possible implementation manner of the first aspect, the instructions to retrieve the applications from the application store comprises instructions for directing the processor to: obtain a listing of applications from an external source; and retrieve applications contained in the listing of applications from the application store.

With reference to the first or second possible implementation manners of the first aspect, in a third possible implementation manner of the first aspect, the instructions to determine the applications that violate the policies of the application store comprises: instructions for directing the processor to determine, after a period of time, if the retrieved applications are contained in the application store, wherein retrieved applications that are not contained in the application store after the period of time are classified as bad applications.

With reference to the first aspect, in a fourth possible implementation manner of the first aspect, the instructions to generate the feature table comprises: instructions for directing the processor to compile the blacklist based on a set of common terms, Intellectual Property

(IP) violating terms, third-party libraries having aggressive advertisement behavior and identities of undesirable computer servers; compile the permission list based on permissions that access secure device data and device access control data; compile the API call list based on a set of predetermined API calls and obfuscated API calls; compile the metadata list based on a set of predetermined metadata, whereby the blacklist, the permission list, the API call list and the metadata list are each assigned a column in the feature table and each bad application is assigned a row in the feature table.

With reference to the first aspect, in a fifth possible implementation manner of the first aspect, the instructions to obtain parameters for bad applications using the feature table comprises instructions for directing the processor to: for each bad application contained in the feature table, set a blacklist parameter using identified extracted features; set a permission parameter using determined extracted features; set an API call parameter using established extracted features; and set a metadata parameter using discerned extracted features.

With reference to the fifth possible implementation manner of the first aspect, in a sixth possible implementation manner of the first aspect, the instructions to set the blacklist parameter using the identified extracted features comprises instructions for directing the processor to: extract features from the bad application; identify terms and libraries from the extracted features that match terms and libraries contained within the blacklist; identify package files from the extracted features that interact with servers contained in the blacklist; and set the blacklist parameter using the identified extracted features.

With reference to the fifth possible implementation manner of the first aspect, in a seventh possible implementation manner of the first aspect, the instructions to set the permission parameter using the determined extracted features comprises instructions for directing the processor to: extract features from the bad application; determine permissions from the extracted features that match permissions contained within the permission list; and set the permission parameter using the determined extracted features.

With reference to the fifth possible implementation manner of the first aspect, in an eighth possible implementation manner of the first aspect, the instructions to set the API call parameter using the established extracted features comprises instructions for directing the processor to: extract features from the bad application; establish API calls from the extracted features that match API calls contained within the API call list; and set the API call parameter using the established API calls.

With reference to the fifth possible implementation manner of the first aspect, in a ninth possible implementation manner of the first aspect, the instructions to set the metadata parameter using the discerned extracted features comprises instructions for directing the processor to: extract features from the bad application; discern metadata from the extracted features that comply with metadata queries contained within the metadata list; and set the metadata parameter using the discerned metadata.

With reference to any one of the fifth to ninth possible implementation manners of the first aspect, in a tenth possible implementation manner of the first aspect, the extracted features from the bad applications comprise brand names, IP terms, network activities, third-party libraries, permissions, API calls or metadata.

With reference to the first aspect, in an eleventh possible implementation manner of the first aspect, the supervised machine learning algorithm comprises algorithms based on regression models, support vector machine models, K-nearest neighbor models, Bayes classifier models, simple decision tree models or decision tree models.

According to a second aspect of the invention, a method for detecting policy violating applications using a computer server comprises: generating a feature table based on identities of the bad applications, a blacklist, a permission list, an Application Programming Interface (API) list and a metadata list; obtaining parameters for bad applications using the feature table; generating a training dataset based on the bad applications and their associated parameters, and good applications and their associated parameters; and training a supervised machine learning algorithm using the training dataset, wherein the trained supervised machine learning algorithm is used to detect policy violating applications.

With reference to the second aspect, in a first possible implementation manner of the second aspect, the obtaining the bad applications from the application store comprises: retrieving applications from the application store; and determining applications that violate policies of the application store, and classifying applications that violate policies of the application store as bad applications.

With reference to the first possible implementation manner of the second aspect, in a second possible implementation manner of the second aspect, the retrieving the applications from the application store comprises: obtaining a listing of applications from an external source; and retrieving applications contained in the listing of applications from the application store.

With reference to the first or second possible implementation manners of the second aspect, in a third possible implementation manner of the second aspect, the determining the applications that violate the policies of the application store comprises: determining, after a period of time, if the retrieved applications are contained in the application store, wherein retrieved applications that are not contained in the application store after the period of time are classified as bad applications.

With reference to the second aspect, in a fourth possible implementation manner of the second aspect, the generating the feature table comprises: the compiling the blacklist based on a set of common terms, Intellectual Property (IP) violating terms, third-party libraries having aggressive advertisement behavior and identities of undesirable computer servers; compiling the permission list based on permissions that access secure device data and device access control data; compiling the API call list based on a set of predetermined API calls and obfuscated API calls; compiling the metadata list based on a set of predetermined metadata, whereby the blacklist, the permission list, the API call list and the metadata list are each assigned a column in the feature table and each bad application is assigned a row in the feature table.

With reference to the second aspect, in a fifth possible implementation manner of the second aspect, the obtaining the parameters for bad applications using the feature table comprises: for each bad application contained in the feature table, setting a blacklist parameter using identified extracted features; setting a permission parameter using determined extracted features; setting an API call parameter using established extracted features; and set a metadata parameter using discerned extracted features.

With reference to the fifth possible implementation manner of the second aspect, in a sixth possible implementation manner of the second aspect, the setting the blacklist parameter using the identified extracted features comprises: extracting features from the bad application; identify terms and libraries from the extracted features that match terms and libraries contained within the blacklist; identifying package files from the extracted features that interact with servers contained in the blacklist; and setting the blacklist parameter using the identified extracted features.

With reference to the fifth possible implementation manner of the second aspect, in a seventh possible implementation manner of the second aspect, the setting the permission parameter using the determined extracted features comprises: extracting features from the bad application; determining permissions from the extracted features that match permissions contained within the permission list; and setting the permission parameter using the determined extracted features.

With reference to the fifth possible implementation manner of the second aspect, in an eighth possible implementation manner of the second aspect, the setting the API call parameter using the established extracted features comprises: extracting features from the bad application; establishing API calls from the extracted features that match API calls contained within the API call list; and setting the API call parameter using the established API calls.

With reference to the fifth possible implementation manner of the second aspect, in a ninth possible implementation manner of the second aspect, the setting the metadata parameter using the discerned extracted features comprises: extracting features from the bad application; discerning metadata from the extracted features that comply with metadata queries contained within the metadata list; and setting the metadata parameter using the discerned metadata.

With reference to any one of the fifth to ninth possible implementation manners of the second aspect, in a tenth possible implementation manner of the second aspect, the extracted features from the bad applications comprise brand names, IP terms, network activities, third-party libraries, permissions, API calls or metadata.

With reference to the second aspect, in an eleventh possible implementation manner of the second aspect, the supervised machine learning algorithm comprises algorithms based on regression models, support vector machine models, K-nearest neighbor models, Bayes classifier models, simple decision tree models or decision tree models.

Brief Description of the Drawings

The above advantages and features in accordance with this invention are described in the following detailed description and are shown in the following drawings:

Figure 1 illustrating an exemplary block diagram of components in an electronic device for implementing embodiments in accordance with embodiments of the invention;

Figure 2 illustrating a timing diagram for detecting policy violating applications in accordance with embodiments of the invention;

Figure 3 illustrating a flow diagram of a process for detecting policy violating applications in accordance with embodiments of the invention; and

Figure 4 illustrating a flow diagram of a process for classifying applications as bad applications in accordance with embodiments of the invention.

Detailed Description

This invention relates to a system and method for detecting policy violating applications. In particular, the invention utilizes features extracted from a selection of policy violating applications or applications classified as bad applications to derive a machine learning algorithm that is then employed to detect previously undiscovered and/or new types of policy-violating applications. For clarity, policy-violating applications refer to applications that are in breach of an application store's policies.

In particular, applications that are classified as bad applications will first be obtained from an application store. Typically, the application store may be an online application store from which registered users may download applications that are to be installed onto the user's mobile devices. A feature list containing features extracted from the bad applications is then compiled. The feature list is then utilized as an input to generate a training sequence to train supervised machine learning algorithms. The trained supervised machine learning algorithms are then employed to detect new and/or previously undiscovered policy violating applications contained in the application store.

In embodiments of the invention, an application store's policies may typically be divided into four main categories: (1) intellectual property and deception; (2) monetization and ads; (3) spam, store listing and promotion; and (4) security and privacy. The most common types of policy-violating applications or apps are apps that use similar titles to branded apps, apps that use photos or screenshots from other reputable brands, apps that provide misleading description, apps that have little or no functions and apps that hold ads simulating the user interfaces.

One skilled in the art will recognize that many functional units in this description have been labelled as modules throughout the specification. The person skilled in the art will also recognize that a module may be implemented as electronic circuits, logic chips or any combination of electrical and/or electronic discrete components within a device. Further, one skilled in the art will also recognize that a module may be implemented in software which may then be executed by a variety of processors. In embodiments of the invention, a module may also comprise computer instructions or executable code that may instruct a computer processor to carry out a sequence of events based on instructions received. The choice of the implementation of the modules is left as a design choice to a person skilled in the art and does not limit the scope of this invention in any way.

One skilled in the art will recognize that many functional units in this description have been labelled as modules throughout the specification. The person skilled in the art will also recognize that a module may be implemented as circuits, logic chips or any sort of discrete component within a computer server or computing device. Further, one skilled in the art will also recognize that a module may be implemented in software which may then be executed by a variety of processors. In embodiments of the invention, a module may also comprise computer instructions or executable code that may instruct a computer processor to carry out a sequence of events based on instructions received. The choice of the implementation of the modules is left as a design choice to a person skilled in the art and does not limit the scope of this invention in any way.

In addition to the above, it should also be noted that the use of algorithms and symbolic representations of operations on data bits within a computer or processor memory are the means used by those skilled in the art to describe their work. In this description, an algorithm is interpreted to be a self-consistent sequence of steps that produces a desired outcome and that the steps are those that require physical manipulations of physical quantities. Typically, these quantities are in the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. Commonly, these signals are referred to as bits, values, elements, symbols, characters, terms, numbers, or the like.

Unless specifically stated otherwise, it is appreciated that throughout the description, terms such as “generating”, “obtaining”, “retrieving”, “calculating” or “determining” or “displaying” or “processing” or “computing” or the like, refer to the action and processes of a computer system, or of a similar electronic computing device, that is capable of manipulating and transforming data represented as physical (electronic) quantities within the computer system's registers and memories into other data similarly represented as physical quantities within the memory of the computer system or registers.

Figure 1 illustrates a block diagram of a computing device or module for implementing embodiments in accordance with embodiments of the invention. One skilled in the art will recognize that the exact configuration of the module or the computer server may be different and the exact configuration of each module or computer server may vary and Figure 1 is provided by way of example only.

In embodiments of the invention, computer server 100 comprises controller 101 and user interface 102. User interface 102 is arranged to enable manual interactions between a user and computer server 100 and for this purpose includes the input/output components required for the user to enter instructions to control computer server 100. A person skilled in

the art will recognize that components of user interface 102 may vary from embodiment to embodiment but will typically include one or more of display 140, keyboard 135 and track-pad 136.

Controller 101 is in data communication with user interface 102 via bus 115 and includes memory 120, processor 105 mounted on a circuit board that processes instructions and data for performing the method of this embodiment, an operating system 106, an input/output (I/O) interface 130 for communicating with user interface 102 and a communications interface, in this embodiment in the form of a network card 150. Network card 150 may, for example, be utilized to send data from computer server 100 via a wired or wireless network to other processing devices or to receive data via the wired or wireless network. Wireless networks that may be utilized by network card 150 include, but are not limited to, Wireless-Fidelity (Wi-Fi), Bluetooth, Near Field Communication (NFC), cellular networks, satellite networks, telecommunication networks, Wide Area Networks (WAN) and etc.

Memory 120 and operating system 106 are in data communication with CPU 105 via bus 110. The memory components include both volatile and non-volatile memory and more than one of each type of memory, including Random Access Memory (RAM) 120, Read Only Memory (ROM) 125 and a mass storage device 145, the last comprising one or more solid-state drives (SSDs). Memory 120 also includes secure storage 146 for securely storing secret keys, or private keys. It should be noted that the contents within secure storage 146 are only accessible by a super-user or administrator of computer server 100 and may not be accessed by any user of computer server 100. One skilled in the art will recognize that the memory components described above comprise non-transitory computer-readable media and shall be taken to comprise all computer-readable media except for a transitory, propagating signal. Typically, the instructions are stored as program code in the memory components but can also be hardwired. Memory 120 may include a kernel and/or programming modules such as a software application that may be stored in either volatile or non-volatile memory.

Herein the term "processor" is used to refer generically to any device or component that can process such instructions and may include: a microprocessor, microcontroller, programmable logic device or other computational device. That is, processor 105 may be provided by any suitable logic circuitry for receiving inputs, processing them in accordance with instructions stored in memory and generating outputs (for example to the memory components or on display 140). In this embodiment, processor 105 may be a single core or

multi-core processor with memory addressable space. In one example, processor 105 may be multi-core, comprising—for example—an 8 core CPU.

Figure 2 illustrates a timing diagram for configuring computer server 250 to detect policy violating applications contained within application store 260 by utilizing information contained within external source 255. It should be noted that computer server 250, external source 255 and application store 260 may all be configured to wirelessly communicate with each other through the Internet or wireless networks such as, but are not limited to, cellular networks, satellite networks, telecommunication networks, Wi-Fi networks or Wide Area Networks (WAN). Communications between these components may also take place via wired means. Further, application store 260 and external source 255 are typically hosted separately on remote servers or cloud computing servers and are accessible by computer server 250 through the wireless and/or wired means described above.

It should be noted that application store 260 may comprise any type of online or remote application store that may be used to amass and distribute software applications. These applications or mobile apps may then be downloaded by registered users and subsequently installed within a user's mobile device. Typically, the applications or apps contained within the application store are provided by various third party developers. However, for security reasons, these applications will all have to comply with the application store's anti-malware policies else they will be removed from the application store. As such, application store 260 usually performs a thorough or detailed malware screening of all the applications uploaded to application store 260 before application store 260 makes the uploaded application available to members of the public. In other words, for each uploaded application, application store 260 will carry out checks to determine whether each uploaded application is in breach of its anti-malware policies. Upon completion of its investigations, if application store 260 determines that the suspicious application is in breach of its anti-malware policies, application store 260 will then classify the suspicious application as a bad application or policy violating application.

This means that most implementations of application store 260 are not configured to automatically check if an uploaded application is in breach of its privacy-violating policies and/or behavioral-violating policies. Instead, application store 260 relies on reports or complaints from users or researchers about such privacy/behavioral policy violating applications before application store 260 will investigate such an application which may be in breach of the application store's privacy and/or behavioral policies. Upon completion of its investigations, if application store 260 determines that the subject application is in breach of its privacy and/or behavioral policies, application store 260 will then classify the suspicious

application as a bad application or policy violating application. However, as application store 260 does not publish a list of applications that are classified as bad applications, computer server 250 is instead configured to obtain applications that are classified as bad applications through other indirect means.

At step 202, computer server 250 will request for a list of suspicious applications from external source 255. External source 255 may comprise, but is not limited to, online websites, online forums, online databases, publications, journals, and etc. Typically the information contained within external source 255 is provided by third parties such as users of software applications downloaded from application store 260 or researchers. Hence, a person skilled in the art will recognize that the information contained within external source 255 may not be utilized in its raw form.

In an embodiment of the invention, external source 255 may comprise a public forum accessible by third party users. Users of the public forum will update the forum with the latest applications which they opine violate the policies of application store 260. These policies may include, but are not limited to the application store's malware, behavioral and/or privacy policies. In addition to publicizing the suspicious applications in the public forum, the third parties will also simultaneously report such suspicious applications to application store 260 so that application store 260 may carry out their own investigations. For the avoidance of doubt, applications which violate the behavioral policies of application store 260 typically relate to applications that are imposters or fakes whereby these applications pretend that they are from a reputable application developer. As for applications which violate the privacy policies of application store 260, such applications typically causes unsolicited advertisements to be displayed on the user's device and/or steal the user's private information stored within the user's device.

After external source 255 has processed the request from computer server 250, at step 204, external source 255 will then transmit the identities of the suspicious applications such as, but not limited to, the names, links or download sources of all the suspicious applications to computer server 250. In embodiments of the invention, the identities of the suspicious applications may be transmitted as a listing. Computer server 250 will then store the identities or the listing of the suspicious applications in an internal database. After that, computer server 250 will then proceed to attempt to download all the applications contained in the listing from application store 260.

In particular, at step 206, computer server 250 will attempt to download a first application contained in the listing of suspicious applications from application store 260. If the requested application is contained within application store 260, application store 260 will

transmit the requested application to computer server 250 at step 208. Normally, at any one time, only a single application may be requested and downloaded by computer server 250 from application server 260. Hence, steps 206 to 208 will repeat continuously, selecting the next application in the listing sequentially, until computer server 250 has attempted to download all the applications contained in the list of suspicious applications from application store 260. Applications downloaded from application store 260 are then stored in retrieved list 209 within computer server 250 while applications contained within the list that were not successfully downloaded from application store 260 are then segregated into a list labelled as “unavailable” for future use.

One skilled in the art will appreciate that although steps 206 to 208 were carried out singly, i.e. by downloading one application at a time, steps 206-208 may also be carried out in bulk if application store 260 and computer server 250 are both able to handle multiple download requests and able to initiate multiple application downloads simultaneously.

At step 210, computer server 250 will wait for a period of time to lapse before proceeding to step 212. In embodiments of the invention, the period of time may comprise any time period between one to six months, preferably three months. After the time period has lapsed, computer server 250 will then submit a request, at step 212, to application store 260 to retrieve all the applications contained within the previously populated retrieved list 209. If application store 260 determines that a requested application is contained within its database, application store 260 will transfer the requested application to computer server 250 at step 214. Similarly, the step of requesting for the applications and the step of transferring the requested application, if available, from application store 260 to computer server 250 may be carried out singly or in bulk as previously described.

Computer server 250 then proceeds to classify applications received from application store 260 via step 214 as “approved applications” while applications that were not found in application store 260 or applications that were unable to be downloaded from application store 260 at step 214 are then classified as “bad applications”. This classification of the applications in retrieved list 209 takes places at step 216. If computer server 250 is unable to retrieve an application contained in retrieved list 209 from application store 260 at steps 212-214, this implies that application store 260 had removed the subject application from its databases because the subject application was in breach of application store’s policies. Computer server 250 then gathers all the identified bad applications and stores them to be processed at the next step.

At step 218, computer server 250 then proceeds to generate a feature table based on features extracted from the identified bad applications. In particular, computer server 250

will extract empirical and behavioral type features from the identified bad applications and these features include, but are not limited to, features such as brand names, third-party libraries, network activities, metadata, permissions, and application programming interface (API) calls. The extracted empirical and behavioral type features are then used to generate parameters for columns in the feature table while the identity of each bad application is used to populate the rows of the feature table. At this stage, the rows of the feature table are all populated with the identities of the bad applications. As for the columns (i.e. parameters) of the feature table, each of these columns will be populated based on the content of a pre-compiled blacklist, a pre-compiled permission list, a pre-compiled API call list and extracted metadata.

In embodiments of the invention, the extraction of the features from the bad applications may be implemented using the Python software, and the feature detection algorithms may be executed using Java. In a particular embodiment of the invention, the Androguard library may be used to reverse engineer application codes of bad applications so that information about third-party libraries, network activities, permissions and API calls may be extracted.

In particular, to obtain third-party libraries, the function "*dx.get tainted packages().get packages()*" in Androguard is used to first obtain package lists from the bad applications. The package names of the applications are then excluded from the package lists revealing the third-party libraries contained in these applications.

In order to obtain the network activities of the bad applications, the *String* values of package files, e.g. Android application package (APK) files, are obtained from the bad applications using the function "*d.get strings()*". Values starting with "*http://*" or <https://> are then filtered out from the results to provide the network activities of the bad applications.

As for the retrieval of API calls from third-party libraries, the function "*dx.tainted packages.search methods()*" is first applied to the previously obtained third-party libraries. The resulting API calls are then analysed to determine whether these API calls originated from the intrinsic source codes of the application or third-party libraries' source codes.

In particular, computer server 250 will begin by selecting a bad application. Computer server 250 will then extract all the empirical and behavioral type features from the selected bad application. The extracted features are then tagged to the bad application. Once this is done, computer server 250 will select the next application and will repeat this step for all the bad applications until all the bad applications have been processed in such a manner.

Computer server 250 then proceeds to process the extracted features collectively. In particular, features extracted from the bad applications that relate to brand names, third-party libraries and network activities are analyzed by computer server 250 using a pre-compiled blacklist. In embodiments of the invention, the blacklist may be compiled as follows.

Popular brand names, third-party advertisement libraries that include aggressive advertisement behaviors and/or identities of undesirable computer servers are first identified from data obtained from various sources such as external source 255 or from an empirical analysis of applications developed by reputable sources.

Computer server 250 then adds the identified popular or commonly used brand names or Trademarks or Intellectual Property (IP) violating terms to a blacklist. These popular brand names may include, but are not limited to, brands or keywords such as '2', 'II', 'Demo', 'Free', 'Pro', '3D', 'HD', 'flash', 'light', 'bejeweled', 'blitz', 'wAsk', 'racing', 'live', 'wallpaper', 'construction', 'city', 'studio', 'candy', 'bluetooth', 'free', 'game', 'sniper', 'crime', 'craft', 'mine', 'pikachu', 'font', 'farm', 'app', 'video', 'download', 'tube', 'pou', 'gangster', 'bird', 'flappy', 'subway', 'surf', 'dash', 'grand', 'theft', 'sonic', 'rio', 'ninja', 'demo', 'pro', '3D', '2', 'II', 'hay', 'day', 'flv', 'adobe', 'install', 'despicable', 'font', 'galaxy', 'monotype', 'volume', 'boost', 'mp3', and 'music'. A person skilled in the art will appreciate that this blacklist may be periodically updated as the popularity of certain keywords changes or as new keywords are added.

Computer server 250 then adds the identified third-party advertisement libraries that include aggressive advertisement behaviours to the blacklist created above. These third-party advertisement libraries may include, but are not limited to, libraries such as 'startapp', 'inmobi', 'umeng', 'ironsource', 'actionbarsherlock', 'millennialmedia', 'ad sdk', 'revmob', 'chartboost', 'fmod', 'furry', 'mobclix', 'appflood', 'tapjoy', 'jirbo', and 'squareup'. Similarly, one skilled in the art will appreciate that this blacklist may be periodically updated as new third-party advertisement libraries are identified.

Computer server 250 then adds the identities of undesirable computer servers to the blacklist described above. One skilled in the art will recognize that undesirable computer servers relate to computer servers that are of a suspicious nature and are not related to reputable sources or entities. The identities of these undesirable computer servers may include, but are not limited to, servers identified by identities such as 'admob', 'gstatic', 'startappexchange', 'ad-market', 'searchresults', 'inmobi', 'umeng', 'googleapis', 'akamaihd', 'appsdt', 'spear-mint-browser', 'mobilecore', 'avazutracking', 'cloudfront', 'youtube', 'rightyoo',

'iron', 'scmpacdn', 'airpush', and 'yting'. One skilled in the art will appreciate that this blacklist may be periodically updated as new undesirable servers are identified.

Once the blacklist has been compiled, the contents of the pre-compiled blacklist are then assigned to a new column in the feature table. Computer server 250 then proceeds to set a blacklist parameter for each bad application listed in the feature table using this assigned column. This is done by computer server 250 first selecting an application together with its tagged extracted features whereby the application is selected from a row in the feature table. If the selected application has tagged features that are contained within the pre-compiled blacklist, the selected application's blacklist parameter will be set. In embodiments of the invention, an application's blacklist parameter may be set by setting the value in the corresponding column in the feature table to "1". Alternatively, if the application's blacklist parameter is not set, the value in the corresponding column in the feature table may be set to "0". In another embodiment of the invention, the application's blacklist parameter may be set by citing the name of the tagged feature found in the pre-compiled blacklist. For example, if App A is tagged with the feature 'Pikachu', this feature may be cited in the corresponding column for the selected application as "Pikachu = 1". If the application has more than one tagged feature found in the pre-compiled blacklist, all the matched features may be cited in the corresponding column, e.g. 'Pikachu = 1, sonic = 1, mp3 = 1'. One skilled in the art will recognize that the matched features may be cited in other formats without departing from this invention as long as the matched features are cited in the corresponding column.

In addition to the above, computer server 250 also utilizes network activities tagged to the selected bad application to identify package files within the selected application that will instruct the application to connect to undesirable computer servers contained in the pre-compiled blacklist. If the selected application has package files that are contained within the pre-compiled blacklist, the selected application's blacklist parameter will be set either by setting the value in the corresponding column in the feature table to "1" or the name of the package file will be cited in the corresponding column for the selected application, e.g. "PackageFile_1 = Admob = 1".

Computer server 250 will then repeat all the processes described above until all the bad applications listed in the rows of the feature table have been selected and processed accordingly. Once this is done, this means that the blacklist parameter for all the bad applications in the feature table would have been set.

Computer server 250 then proceeds to process extracted features relating to the applications' metadata. In particular, computer server 250 will analyze each application's extracted metadata using a pre-compiled metadata query list. In embodiments of the invention, the pre-compiled metadata query list may include parameter queries such as, but are not limited to, the number of downloads, the app's file size, the number of ratings, the average star rating, the number of users rating one star, the number of users rating two stars, the number of users rating three stars, the number of users rating four stars and/or the number of users rating five stars. The contents of the pre-compiled metadata query list are then assigned to a new column in the feature table.

Computer server 250 then proceeds to set a metadata parameter for each bad application listed in the feature table using this assigned column. This is done by computer server 250 first selecting an application together with its tagged extracted metadata. Computer server 250 then utilizes the extracted metadata tagged to the selected application to set the metadata parameter for the selected application. For example, if the pre-compiled metadata query list comprises parameter queries such as "Number of Downloads" and "App File Size", computer server 250 will obtain the corresponding answers to these parameter queries from the extracted metadata tagged to the selected application. In this example, this means that the corresponding column for the selected application may be set as "Number of Downloads = 1000" and "App File Size = 100kB" where the numbers 1000 and 100kB are obtained from the selected application's tagged metadata.

Computer server 250 will then repeat all the processes described above until all the bad applications listed in the rows of the feature table have been selected and processed accordingly. Once this is done, this means that the metadata parameter for all the rows of applications in the feature table would have been set.

Computer server 250 then proceeds to process extracted features relating to the applications' permissions. Features extracted from the bad applications that relate to permissions are then analyzed by computer server 250 using a pre-compiled permission query list. In embodiments of the invention, the permission query list may be compiled based on data obtained from various sources such as external source 255 or from an empirical analysis of well-known bad applications. Permissions compiled in the permission query list may include permissions commonly employed by bad applications or are relevant to malware behaviors such as, but are not limited to, permissions requesting access to secure device data, device access control data, a user's credentials, a user's contact list, permissions requesting access to hardware, such as camera, audio or video, and/or permissions such as INSTALL SHORTCUT, WRITE HISTORY, BOOKMARKS, BILLING,

SEND SMS and etc. One skilled in the art will appreciate that this permission query list may be periodically updated as new types of permissions are identified.

Once the permission query list has been compiled, the contents of the pre-compiled query permission list are then assigned to a new column in the feature table.

Computer server 250 then proceeds to set a permission parameter for each bad application listed in the feature table using this assigned column. This is done by computer server 250 first selecting an application together with its tagged extracted permissions. Computer server 250 then utilizes the extracted permissions tagged to the selected application to set the permission parameter for the selected application. For example, if the pre-compiled permission query list comprises permissions such as "INSTALL SHORTCUT" and "BILLING", computer server 250 will obtain the corresponding answers to these permission queries from the selected application's extracted permissions. In this example, this means that if computer server 250 determines from the extracted features that the "INSTALL SHORTCUT" permission is present but the "BILLING" permission is missing, computer server 250 will set the corresponding column as "INSTALL SHORTCUT= 1" and "BILLING = 0", where one skilled in the art will recognize that "1" implies that the permission is present while "0" implies that the permission is not available.

Computer server 250 will then repeat all the processes described above until all the bad applications listed in the rows of the feature table have been selected and processed accordingly. Once this is done, this means that the permission parameter for all the bad applications in the feature table would have been set.

Computer server 250 then proceeds to process extracted features relating to the applications' API calls. API calls extracted from the bad applications are then analyzed by computer server 250 using a pre-compiled API call query list. In embodiments of the invention, the API call query list may be compiled based on data obtained from various sources such as external source 255, or from an empirical analysis of well-known bad applications that exhibit adware or malware behaviors. Computer server 250 then compiles the API call query list using API calls usually employed by bad applications. API calls commonly employed by bad applications or are relevant to malware or adware behaviors include, but are not limited to, obfuscated API calls, API calls for sending advertisements as notifications, changing browser settings, Java reflection, loading dynamic code, importing cryptographic packages, e.g. *javax.crypto*, overwriting the *onBackPressed()* function, executing suspicious strings, downloading suspicious package files and etc. One skilled in the art will appreciate that this API call query list may be periodically updated as new types of API calls employed by adware or malware applications are identified.

Once the API call query list has been compiled, the contents of the pre-compiled API call list are then assigned to a new column in the feature table.

Computer server 250 then proceeds to set an API call parameter for each bad application listed in the feature table using this assigned column. This is done by computer server 250 first selecting an application together with its tagged extracted API calls. Computer server 250 then utilizes the selected application's extracted API calls to set the API call parameter for the selected application. For example, if the pre-compiled API call query list comprises an obfuscated API call, computer server 250 will determine from the selected application's extracted features if the application contains an obfuscated API call. If computer server 250 determines that such an API call is present, computer server 250 will set the corresponding column as "Obfuscated API call = 1". Similarly, one skilled in the art will recognize that "1" implies that the API call is present while "0" implies that the API call is not available.

Computer server 250 will then repeat all the processes described above until all the bad applications listed in the rows of the feature table have been selected and processed accordingly. Once this is done, this means that the API call parameter for all the bad applications in the feature table would have been set.

An exemplary feature table, i.e. Table 1, which may be generated by computer server 250 at step 218, is set out below. One skilled in the art will recognize that the parameters of the various applications may vary and is not limited to the parameters illustrated to the table alone. In the table below, apps A, B, C and D are applications that were previously identified as bad application hence; the identities of these bad applications were used to populate the rows of the feature table.

Further, as illustrated in the table below, the headers of the columns comprise the pre-compiled blacklist, the metadata query list, the permission query list and the API call query list. As computer server 250 processes each app in the table below, computer server 250 will determine, from the extracted features tagged to each app, the parameters or attributes that are to be assigned or set for each app. For example, apps A, B, C and D in the feature table below will be set with the following parameters:

- App A ('free = 1', 'startapp = 1', 'Package_1 = admod'; 'No. of downloads = 10', 'Average star rating = 2'; 'INSTALL SHORTCUT = 1'; 'Obfuscated API call = 1')
- App B ("HD = 1', 'music = 1', 'adobe = 1'; "No. of ratings = 2", 'WRITE HISTORY = 1'; 'Changing browser setting = 1')

- App C ('Package_2 = inmobi'; 'No. of users rating 4 stars = 5'; 'Permission = 0'; 'Load dynamic code = 1')
- App D ('umeng = 1', 'startapp = 1', 'tapjoy = 1'; 'No. of ratings = 1', 'BOOKMARKS = 1', 'INSTALL SHORTCUT = 1'; API call = 0)

App Identity	Blacklist	Metadata	Permissions	API Call
App A	'free = 1', 'startapp = 1', 'Package_1 = admod'	'No. of downloads = 10', 'Average star rating = 2'	'INSTALL SHORTCUT = 1'	'Obfuscated API call = 1'
App B	'HD = 1', 'music = 1', 'adobe = 1'	'No. of ratings = 2'	'WRITE HISTORY = 1'	'Changing browser setting = 1'
App C	'Package_2 = inmobi'	'No. of users rating 4 stars = 5'	0	'Load dynamic code = 1'
App D	'umeng = 1', 'startapp = 1', 'tapjoy = 1'	'No. of ratings = 1'	'BOOKMARKS = 1', 'INSTALL SHORTCUT = 1'	0

Table 1

Once computer server 250 has completed setting all the parameters, i.e. the blacklist parameter, the metadata parameter, the permission parameter, and the API call parameter, for all the applications contained in the feature table, computer server 250 then proceeds to step 220.

At this step, computer server 250 will then retrieve good applications or good apps together with their respective parameters. In embodiments of the invention, the good apps

and their parameters may be obtained from various sources such as external source 255, or from an empirical analysis of well-known good applications that do not exhibit adware or malware behaviors. An example of a good application and its parameters is:

App_Good ('Blacklist = 0'; 'Downloads = 500,000', 'Permission = 0'; 'API Call = 0')

In general, a good application would not have any features that are contained within the blacklist parameter and it would not have permissions or API calls contained within the permission and API call lists in the feature table.

Computer server 250 then proceeds to compile a training dataset using the good applications and their parameters, and the bad applications and their parameters – as obtained from the previously generated feature table. The training dataset is then used as training data for training supervised machine learning algorithms so that the trained supervised machine learning algorithms may then be used to determine whether an application is to be classified as a good application or bad application. The trained supervised machine learning algorithm may then be deployed to mobile computing devices to detect policy violating applications.

In embodiments of the invention, the training data may be used to train supervised machine learning algorithms such as, but not limited to, Naïve Bayes classifiers, Logistic classifiers, Sequential Minimal Optimization (SMO) classifiers, Lazy-Ibk classifiers, Random Committee classifiers, Decision Table classifiers, Decision Part classifiers, J48 classifiers, Logistic Model Tree (LMT) classifiers, and Random Forest Tree classifiers. One skilled in the art will recognize that Naive Bayes is a family of simple probabilistic classifiers based on the Bayes' theorem, the logistic classifier applies the regression model, SMO applies Support Vector Machines (SVM), and the Lazy-Ibk classifier applies K-nearest neighbours algorithm. As for the Random Committee classifier, this classifier uses a group of base classifiers, and the result is the average of the predictions generated by the individual base classifiers. Unlike the other classifiers, the Decision Table classifier uses a simple decision table while the J48 Tree classifier, LMT classifier and Random Forest Tree classifier are algorithms based on decision trees. One skilled in the art will recognize that other types of classifiers may be utilized without departing from this invention.

In accordance with an embodiment of the invention, a computer implemented method for detecting policy violating applications, comprises the following four steps:

Step 1, obtain bad applications from an application store;

Step 2, generate a feature table based on identities of the bad applications, a blacklist, a permission list, an Application Programming Interface (API) list and a metadata list;

Step 3, obtain parameters for bad applications using the feature table;

Step 4, generate a training dataset based on the bad applications and their associated parameters, and good applications and their associated parameters; and

Step 5, train a supervised machine learning algorithm using the training dataset, wherein the trained supervised machine learning algorithm is used to detect policy violating applications.

In order to provide such a system or method, a process is needed for detecting policy violating applications. The following description and Figures 3 and 4 describe embodiments of processes that provide processes in accordance with this invention.

Figure 3 illustrates process 300 that is performed by a module or computer processor installed within a computer server or computing device to detect policy violating applications in accordance with embodiments of this invention.

Process 300 begins at step 305 whereby process 300 retrieves applications that violate an application store's policies from the application store. Such applications that breach the application store's policies are classified as bad applications by process 300. Process 300 then proceeds to extract features from these retrieved bad applications. At step 310, process 300 will then utilize the extracted features to populate parameters of bad applications contained in a feature table. It should be recalled that this feature table was generated based on identities of the bad applications, a pre-compiled blacklist, a pre-compiled permission list, a pre-compiled Application Programming Interface (API) list and a pre-compiled metadata list.

The feature table is then used by process 300 to generate a training dataset for training supervised machine learning (SML) algorithms at step 315. At step 320, process 300 then trains supervised machine learning algorithms using the training dataset generated at step 315. The trained supervised machine learning algorithms may then be deployed by process 300 to detect policy violating applications. Process 300 then ends.

Figure 4 illustrates process 400 that is performed by a module or computer processor installed within a computer server for classifying applications as bad applications in accordance with embodiments of the invention.

Process 400 begins at step 405 whereby process 400 obtains a list of applications from an external source. In embodiments of the invention, the list of applications contains identities or descriptions of applications which third party users opine are suspicious or bad applications. At step 410, process 400 will then attempt to retrieve all the applications contained within the list of applications retrieved at step 405 from an application store. Applications that are successfully retrieved at this step are then segregated into a retrieved application list. Process 400 then waits for a period of time to lapse at step 415. Once the period of time has lapsed, process 400 then proceeds to step 420.

At step 420, process 400 will select an application that was retrieved from the application store at step 410. Process 400 then determines whether the retrieved application may still be downloaded from the application store at step 425. If the application may still be downloaded, process 400 proceeds to step 430. It should be noted that it is not required for process 400 to actually download the subject application from the application store at step 425. Process 400 may initiate the download process and upon successful initiation of the download process may then terminate the download process as it has been confirmed that the subject application is still available in the application store.

Process 400 then proceeds to select another application from the retrieved application list at step 435. Steps 425 to 430 are repeated until all the applications in the retrieved application list have been processed by process 400 at step 425. Process 400 then ends.

Returning to step 425, if at this step process 400 determines that a subject application may not be downloaded from the application store or if process 400 determines that the application is no longer available on the application store, process 400 proceeds to step 440. At step 440, process 400 will classify the subject application as a bad application. Process 400 then proceeds to step 430 to evaluate another application. As described above, process 400 then repeats steps 425-430 until all applications in the retrieved list has been processed by steps 425 and 430. Process 400 then ends.

The above is a description of embodiments of a system and process in accordance with the present invention as set forth in the following claims. It is envisioned that others may and will design alternatives that fall within the scope of the following claims.

CLAIMS:

1. A computer server for detecting policy violating applications comprising:
 - a processor; and
 - a non-transitory media readable by the processor, the media storing instructions that when executed by the processor, cause the processor to:
 - obtain bad applications from an application store;
 - generate a feature table based on identities of the bad applications, a blacklist, a permission list, an Application Programming Interface (API) list and a metadata list;
 - obtain parameters for bad applications using the feature table;
 - generate a training dataset based on the bad applications and their associated parameters, and good applications and their associated parameters; and
 - train a supervised machine learning algorithm using the training dataset, wherein the trained supervised machine learning algorithm is used to detect policy violating applications.
2. The computer server according to claim 1 wherein the instructions to obtain bad applications from the application store comprises:
 - instructions for directing the processor to:
 - retrieve applications from the application store;
 - determine applications that violate policies of the application store, and
 - classify applications that violate policies of the application store as bad applications.
3. The computer server according to claim 2 wherein the instructions to retrieve the applications from the application store comprises:
 - instructions for directing the processor to:
 - obtain a listing of applications from an external source; and
 - retrieve applications contained in the listing of applications from the application store.
4. The computer server according to any one of claims 2 or 3 wherein the instructions to determine the applications that violate the policies of the application store comprises:
 - instructions for directing the processor to:

determine, after a period of time, if the retrieved applications are contained in the application store, wherein retrieved applications that are not contained in the application store after the period of time are classified as bad applications.

5. The computer server according to any one of claims 1-4 wherein the instructions to generate the feature table comprises:
 - instructions for directing the processor to:
 - compile the blacklist based on a set of common terms, Intellectual Property (IP) violating terms, third-party libraries having aggressive advertisement behaviour and identities of undesirable computer servers;
 - compile the permission list based on permissions that access secure device data and device access control data;
 - compile the API call list based on a set of predetermined API calls and obfuscated API calls;
 - compile the metadata list based on a set of predetermined metadata, whereby the blacklist, the permission list, the API call list and the metadata list are each assigned a column in the feature table and each bad application is assigned a row in the feature table.
6. The computer server according to any one of claims 1-5 wherein the instructions to obtain parameters for bad applications using the feature table comprises:
 - instructions for directing the processor to:
 - for each bad application contained in the feature table,
 - set a blacklist parameter using identified extracted features;
 - set a permission parameter using determined extracted features;
 - set an API call parameter using established extracted features; and
 - set a metadata parameter using discerned extracted features.
7. The computer server according to claim 6 wherein the instructions to set the blacklist parameter using the identified extracted features comprises:
 - instructions for directing the processor to:
 - extract features from the bad application;
 - identify terms and libraries from the extracted features that match terms and libraries contained within the blacklist;
 - identify package files from the extracted features that interact with servers contained in the blacklist; and
 - set the blacklist parameter using the identified extracted features.

8. The computer server according to claim 6 or 7 wherein the instructions to set the permission parameter using the determined extracted features comprises:
instructions for directing the processor to:
 - extract features from the bad application;
 - determine permissions from the extracted features that match permissions contained within the permission list; and
 - set the permission parameter using the determined extracted features.
9. The computer server according to any one of claim 6-8 wherein the instructions to set the API call parameter using the established extracted features comprises:
instructions for directing the processor to:
 - extract features from the bad application;
 - establish API calls from the extracted features that match API calls contained within the API call list; and
 - set the API call parameter using the established API calls.
10. The computer server according to any one of claims 6-9 wherein the instructions to set the metadata parameter using the discerned extracted features comprises:
instructions for directing the processor to:
 - extract features from the bad application;
 - discern metadata from the extracted features that comply with metadata queries contained within the metadata list; and
 - set the metadata parameter using the discerned metadata.
11. The computer server according to any one of claims 6 to 10 wherein the extracted features from the bad applications comprise brand names, IP terms, network activities, third-party libraries, permissions, API calls or metadata.
12. The computer server according to anyone of claims 1-11 wherein the supervised machine learning algorithm comprises algorithms based on regression models, support vector machine models, K-nearest neighbour models, base classifier models, simple decision tree models or decision tree models.

13. A method for detecting policy violating applications using a computer server comprising:
 - obtaining bad applications from an application store;
 - generating a feature table based on identities of the bad applications, a blacklist, a permission list, an Application Programming Interface (API) list and a metadata list;
 - obtaining parameters for bad applications using the feature table;
 - generating a training dataset based on the bad applications and their associated parameters, and good applications and their associated parameters; and
 - training a supervised machine learning algorithm using the training dataset, wherein the trained supervised machine learning algorithm is used to detect policy violating applications.
14. The method according to claim 13 wherein the obtaining bad applications from the application store comprises:
 - retrieving applications from the application store;
 - determining applications that violate policies of the application store, and
 - classifying applications that violate policies of the application store as bad applications.
15. The method according to claim 14 wherein the retrieving the applications from the application store comprises:
 - obtaining a listing of applications from an external source; and
 - retrieving applications contained in the listing of applications from the application store.
16. The method according to any one of claims 14 or 15 wherein the determining the applications that violate the policies of the application store comprises:
 - determining, after a period of time, if the retrieved applications are contained in the application store, wherein retrieved applications that are not contained in the application store after the period of time are classified as bad applications.
17. The method according to any one of claims 13-16 wherein the generating the feature table comprises:
 - compiling the blacklist based on a set of common terms, Intellectual Property (IP) violating terms, third-party libraries having aggressive advertisement behaviour and identities of undesirable computer servers;

compiling the permission list based on permissions that access secure device data and device access control data;

compiling the API call list based on a set of predetermined API calls and obfuscated API calls;

compiling the metadata list based on a set of predetermined metadata, whereby the blacklist, the permission list, the API call list and the metadata list are each assigned a column in the feature table and each bad application is assigned a row in the feature table.

18. The method according to any one of claims 13-17 wherein the obtaining the parameters for bad applications using the feature table comprises:
 - for each bad application contained in the feature table,
 - setting a blacklist parameter using identified extracted features;
 - setting a permission parameter using determined extracted features;
 - setting an API call parameter using established extracted features; and
 - setting a metadata parameter using discerned extracted features.
19. The method according to claim 18 wherein the setting the blacklist parameter using the identified extracted features comprises:
 - extracting features from the bad application;
 - identifying terms and libraries from the extracted features that match terms and libraries contained within the blacklist;
 - identifying package files from the extracted features that interact with servers contained in the blacklist; and
 - setting the blacklist parameter using the identified extracted features.
20. The method according to claim 18 or 19 wherein the setting the permission parameter using the determined extracted features comprises:
 - extracting features from the bad application;
 - determining permissions from the extracted features that match permissions contained within the permission list; and
 - setting the permission parameter using the determined extracted features.
21. The method according to any one of claims 18-19 wherein the setting the API call parameter using the established extracted features comprises:
 - extracting features from the bad application;

establishing API calls from the extracted features that match API calls contained within the API call list; and

setting the API call parameter using the established API calls.

22. The method according to any one of claims 18-21 wherein the setting the metadata parameter using the discerned extracted features comprises:
 - extracting features from the bad application;
 - discerning metadata from the extracted features that comply with metadata queries contained within the metadata list; and
 - setting the metadata parameter using the discerned metadata.
23. The method according to any one of claims 18 to 22 wherein the extracted features from the bad applications comprise brand names, IP terms, network activities, third-party libraries, permissions, API calls or metadata.
24. The method according to any one of claim 13-23 wherein the supervised machine learning algorithm comprises algorithms based on regression models, support vector machine models, K-nearest neighbour models, base classifier models, simple decision tree models or decision tree models.