SAMoC: Self-Aware Access Monitoring and Controlling Framework for Android

Nanda Kumar Thanigaivelan, Ethiopia Nigussie, Seppo Virtanen, and Jouni Isoaho

Department of Future Technologies, University of Turku, Turku, Finland
{nakuth,ethnig, seppo.virtanen, jisoaho}@utu.fi

Abstract. We present the concept and specification of a self-aware access monitoring and controlling framework for Android. The lack of users' security awareness, Android operating system's security limitations and the widespread use of Android for personal and organizational activities emphasizes the importance of improving its security. The proposed SAMoC framework strengthens the security of Android through learning based operation refinements. The introduction of a monitoring agent on each resource and a self-aware layer in the Android software stack allows to detect unexpected events and perform fine-grained access control over the resources. The self-aware layer can automatically refine the policies or provide suggestions depending on its configuration. The SAMoC framework enforces specific restrictions on an individual user basis and allows the users to play a role in the system operation optimization due to its transparent nature.

Keywords: Self-aware security · Mobile security · Access control.

1 Introduction

Mobile devices are becoming the dominant devices for communication and online service consumption among individual users and organizations. The exponential growth in the number and diversity of the available applications is the main reason for widespread popularity of mobile devices. The increase in popularity of mobile devices along with users' limited security awareness, and operating system vulnerabilities make these devices easy target for security attacks. An increasing number of organizations is allowing employees to access the enterprise's resources through mobile devices and even encouraging to use their own devices (Bring Your own Devices (BYoD)). In addition to enterprise apps, a number of other apps from various channels may also be installed in the same device, exposing the enterprise to numerous security threats and attack vectors.

This work focuses on access control for Android based mobile devices to ensure their security. Android is an open-source operating system developed based on the Linux kernel [1]. The Android operating system deploys mechanisms such as application sandboxing and the Android permission model. The sandboxing provides application isolation and containment using Linux access control and process protection mechanisms. The permission model restricts an applications capabilities by regulating sensitive API calls that access protected resources. Permissions to access certain resources are granted during first access request. Users tend to approve all requested permissions since the users may not understand fully the consequences. Though runtime permission revocation is introduced in Android 6.0. it still fails to allow fine grained permission control. Third-party libraries (for example, advertisement and analytics) cannot be prevented from abusing the granted permissions of their host apps because of the unavailability of separation mechanisms. In other words, Android lacks runtime configurable access control for preventing an application from accessing any open interfaces of another application and abusing its granted permissions.

The future mobile devices will be part of the internet-of-things (IoT) and they can operate as sensing, actuating, and intermediate gateway devices in IoT. Intercommunication among mobile devices is highly likely in future IoT applications. These activities will enhance the roles and usage of mobile devices which makes them an attractive target for attacks since the impact of attacks can be higher. Therefore, it is necessary to come up with a system which is able to reconfigure and adapt its operation to ever evolving threat. To counter such circumstances, the system has to be self-aware.

In this work, we present the concept and specification of a self-aware access monitoring and controlling framework for Android. The main working principle of the framework is to monitor, to learn and then to refine existing policies or defining new ones at runtime. A self-aware layer is introduced in the Android software stack with the sole intention of learning from the monitored information and implementing appropriate controlling measures. The contributions of this work are as follows:

- Introduction and adaptation of self-awareness in Android Mandatory Access Control (MAC): to detect and/or prevent unexpected events including the current and future intentional or unintentional malicious activities and resource abuses.
- Self-optimization of operations: through continuous monitoring and learning, the framework defines new access policies or refines existing policies which in turn optimizes the operation and resource usage.
- Multiuser environment support: to allow enforcing different restriction policies and customizing them accordingly for each user account in a single device.

The rest of the paper is structured as follows. Android and its security limitations are presented in Section 2. The key features of the proposed self-aware access monitoring and controlling (SAMoC) framework are discussed in Section 3. The concept of self-awareness in general and its importance in Android security context are explained in Section 4. The detailed specification of the SAMoC framework is presented in Section 5. Finally, discussion and conclusion are presented in Section 6 and Section 7, respectively.

2 Android and its Security Limitations

Android is a Linux-based open source mobile operating system, developed by Open Handset Alliance led by Google. The Linux kernel is customized for Android to provide features such as Binder, asynchronous shared memory, process memory allocator and power management. The kernel layer acts as a hardware abstraction layer and provides various services such as networking, file system and device drivers. The middleware layer consists of Android framework, native and runtime libraries including Android Runtime (ART). It is responsible for handling application life cycle management and other system services along with the access restrictions on application resource accessibility. The application layer hosts the core/system applications and other applications written by Android developers. In addition to Java, developers can use C/C++ through the Java Native Interface (JNI). The Android application architecture is unique in such a way that it is designed to provide application compatibility, portability and security.

2.1 Android Security Model

Since Android is Linux-based operating system, it inherits user resource isolation and process isolation from Linux. Android extends the user resource isolation feature by extending to application level. Under this modification, each application is assigned with unique ID (UID) upon installation and executes it as a dedicated process using the same UID. Also, dedicated data directory for its resources is given and restricts to access other files unless explicitly permitted. The system resources such as daemons and system applications are owned either by system or root user and executed under pre-defined UID. Thus, it achieves isolation in terms of both process level as well as file level.

Due to the isolation, application cannot access any resources other than their own resources. In order to provide access to the resources such as internet connectivity, data and hardware features, Android introduced access rights in the form of permissions. Application required to access those features need to obtain appropriate permission and the developer must declare the required permission in the AndroidManifest. Android offers 130 permissions and also allow developers to define their own permissions to enforce access restrictions on their application's critical application resources [11]. Android framework enforces access verification during runtime to ensure applications has appropriate access rights. There are few permissions which are not monitored by the Android framework since they are mapped to the low level operating system control.

2.2 Limitations

Isolation of applications and their resource accessibility is contained through application sandboxing. Android also caters permission to access other resources in order to capitalize their features for enhancing user experiences. Due to its popularity and continuous increase in user base, it has attracted the attention of research community to identify the vulnerabilities and possible solutions [5, 6, 4, 7, 12–14]. Also, it has gained the attention of adversaries, who aim to exploit the vulnerabilities for their own benefits. To enhance the security through the enforcement of MAC at kernel layer, SE Linux is introduced into the Android platform since version 4.3 (as SEAndroid). The implementation of SE Linux is still evolving in Android from permissive mode in version 4.3 to full enforcement mode in version 5.0. SEAndroid provides huge advantage in protecting the device, for example, it has the capability to prevent application installation based on its signatures [10]. However, it is not possible to define the certificates for all available applications in the policy. Extending SE Linux to cover the Android's middleware is challenging mainly because of implementations complexity as well as requiring an invasive and costly set of changes [10]. Due to these constraints the middleware is loosely protected by Android's default security model. SE Linux also requires specialized skills for implementing and enforcing policies.

From version 6.0 onwards, several enhancements are introduced and one example is runtime permission revocation. It allows users to change (grant or revoke) the status of the applications' permissions as needed. This is one important step in improving security but it fails to provide fine granularity for controlling the resources. In this work, we are presenting a framework which provide fine granular access control, and more controllable system features.

3 Key Features of SAMoC Framework

A self-aware system is capable of knowing its environment, the on-going activities and implementing appropriate corrective measures at run-time in order to achieve and maintain the required performance. By introducing a self-aware concept in mobile MAC, we are aiming to achieve the following main goals:

- Self-awareness: to reduce user intervention in security policy enforcement and maintenance by creating self-aware components. These components can access logs, learn, able to predict unexpected behaviors/events, and refine/define new policies accordingly.
- Highly refined policies: to attain higher granularity in policy refinement, access restriction enforcement based on the resource features rather than the enforcement on application level or resource level will be introduced. For example, in the proposed framework, instead of blocking entire internet access resource, it is possible to allow certain protocol on particular port(s) and deny the rest of the protocol for an application.
- BYOD and parental solutions: The system will provide multi-user support and it is tightly coupled with device users. This allows enforcement of different sets of control policies and customization of each policy set for individual user account in a device.
- Improved system resilience: The integration of five subsystems into the framework (application installation and monitor, device resource controller, communication controller, self-aware subsystem, and SAMoC user interface) enhances the system's resiliency by allowing to run atleast one subsystem

continuously. The user can decided to enable/disable the certain subsystem(s)/policies and keep the rest active all the time.
- Easy to use: since there is no predefined structure/syntax for writing most of the policies, the personal users can easily adopt to the system.
- Transparency: Allowing device users to access the system logs and the enforced policies enables the users to understand behind-the-scene operations either fully or partially depending on their knowledge.

4 Adaptation of Self-Awareness in Security Context

A number of researches have been conducted and many solutions have been proposed to thwart the threats in the Android but most of them are concentrated on either providing resource access restriction or prevention of identified threats and few works target both cases [7, 10]. The existing research on mobile security fail to recognize the assimilation of mobile devices in IoT and the resulting change in the mobile usage landscape. Mobile device are part of IoT and play an important role in IoT as a sensing, actuating, and/or intermediate gateway. The interoperability of mobile in IoT is facilitated due to the introduction of 6lowPAN based wireless sensor networks. There is also on-going effort to connect IP networks through bluetooth low energy [15, 16]. One potential example of this integration is the possibility of gathering information through crowd sensing by using the sensors available in the mobile devices along with other sensor networks [17, 18]. Intercommunication between the mobile devices is highly likely in future IoT applications. In these circumstances, it is difficult to know the security threats beforehand. In order to protect the device resources, we need to develop a system which is capable of self-reconfiguration and self-adaptation. To realize this objective, we have to make the system self-aware.

Any system which has the capability to learn and adapt itself through monitoring its own operations and environment is termed as self-aware system. Implementation of self-aware systems is not an easy task but they can offer numerous benefits upon proper employment.

4.1 Self-awareness in Security Context

The advantages of self-awareness especially detection and prevention of malicious activities through monitoring and learning are critical to the security of the systems since threats or threat sources are dynamic and evolving. To capitalize on these benefits, we have adapted the self-awareness into the security context. The customized self-aware agent is shown the Figure 1.

The self-aware agent comprises of five components and the tasks of each component are described as follows.

Self-Configure: the purpose of self-configuration is to update the settings of the self-aware agent components by considering the previous and current circumstances to ensure the required objectives are fulfilled with minimal overheads. The self-configure component is also responsible for guiding the rest of self-aware

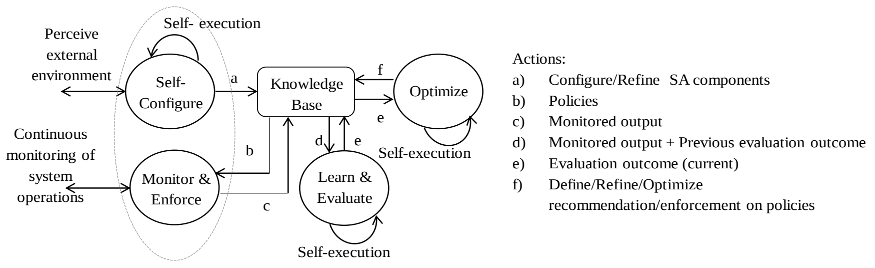
Self- execution
Perceive external environment
Self-Configure
a
Knowledge Base
f
Optimize
e
Continuous monitoring of system operations
b
d
e
Self-execution
Monitor & Enforce
Learn & Evaluate
c
Self-execution
Actions:
a) Configure/Refine SA components
b) Policies
c) Monitored output
d) Monitored output + Previous evaluation outcome
e) Evaluation outcome (current)
f) Define/Refine/Optimize recommendation/enforcement on policies

Fig. 1. Customized self-aware agent for security context

agent components execution for efficient power and device usability. It will oversee the internal and external factors such as peek device usage, active and idle duration, and then define/refine execution period for itself, *Learn and Evaluate* and *Optimize* components.

Monitor and Enforce: it engages in continuous observation and enforcement of the policies on the systems activities. It ensures that the applications behavior and system resources accessibility conform with the provided policies. It is also responsible for transferring the observation to the *Knowledge Base (KB)*. The information observed by this component will be used as a basis for the *Learn and Evaluate* component.

Learn and Evaluate: this component has two operating phases. In the learning phase, information gathered by *Monitor and Enforce* component are analyzed in order to derive the common system behavior patterns. On successive execution of this phase, it will generate a new pattern if there is a change in the systems application behavior, otherwise the same common pattern will be generated. In the evaluate phase, the identified pattern is compared against the common patterns. Upon on detection of deviation, it will construct appropriate changes and submit the recommended changes to the *KB*.

Optimize: this component examines the recommendations provided by the *Learn and Evaluate* component and change the policies accordingly or provide them as suggestions to the user depending on the chosen configuration. It is also responsible for restoring the policies to the previous states if the applied changes fails to fullfill the requirement.

Knowledge Base: is used as a storage space for all activities. It contains the policies, system activities as logs, derived patterns, and policy recommendations, along with the setting configuration of the self-aware agent. As can be seen from Figure 1, all the inputs and outputs for the other four components of self-aware agent are from and to the *KB*.

5 SAMoC Framework Specification

The primary objective of the proposed framework is to achieve higher degree of security by inhibiting malicious or unintended activities through highly refined policy enforcement, self-learning, and self-reconfiguration. The system which is developed by incorporating the SAMoC framework will reduce the requirement for human involvement in making more appropriate decisions which in turn improve the security of the mobile device as well as system usability. The framework engages in continuous monitoring of the resources and communication channels to ensure that all apps are working appropriately according to the defined policies. It also performs periodic assessment of monitored information as this is necessary for policies and processes refinement. Since the framework will be ingrained into the Android platform, it will start functioning upon device boot. The framework operates along with Android's default security implementation.

5.1 Assumptions

As in other system developments, we have defined certain assumptions to ensure proper functioning of the system. The users and administrators who are configuring the polices are fully trusted and they required to posses knowledge on system functionality principles. We assume that no application is granted root permission and the device will not be rooted in any circumstances as this may result in applications to abuse system resources intentionally or unintentionally. The Android operating system and the components that we are introducing are fully trusted.

5.2 SAMoC Mobile

The mobile part consists of the following components: handlers, self-aware components and synchronization components. The architecture of the SAMoC mobile is shown in Figure 2.

Handlers: The handlers consist of two major components: policy handlers and event-log handlers. They play a crucial role in the framework since all the communication towards logs and policies are directed by them. Policy handlers are responsible for providing appropriate policy to the monitoring and enforcing component to act upon the resources. Event-log handlers perform log maintenance by allowing other SAMoC mobile components to read or write the logs as and when they needed.

Self-Aware Component: self-aware component is the core of the framework. It comprises of self-configuration, monitoring and enforcing components, evaluation and optimization components. The self-configuration component is responsible for guiding the rest of self-aware components by updating their execution settings. Monitoring and enforcing enhances the normal Android execution flow by placing appropriate hooks in the Android components which are responsible for handling applications and their device services accessing capabilities. The evaluation component is responsible for making policy recommendations by accessing the logs, identifying the correlation and establishing relationship among the correlated information. The optimization component will refine the existing policies, define new policies or recommend changes to the user based on the recommendations provided by the evaluation component.

Synchronization Components: Synchronization component is responsible for handling request access to the policies by the SAMoC UI. In addition, it will also handle compression, decompression and purging old logs from the repository.

Repository is a storage component which contains the policies and logs generated by the framework components. It will also acts as knowledge base for self-aware components by storing and retrieving the settings required for their self-execution, behavioural patterns and policy recommendations.

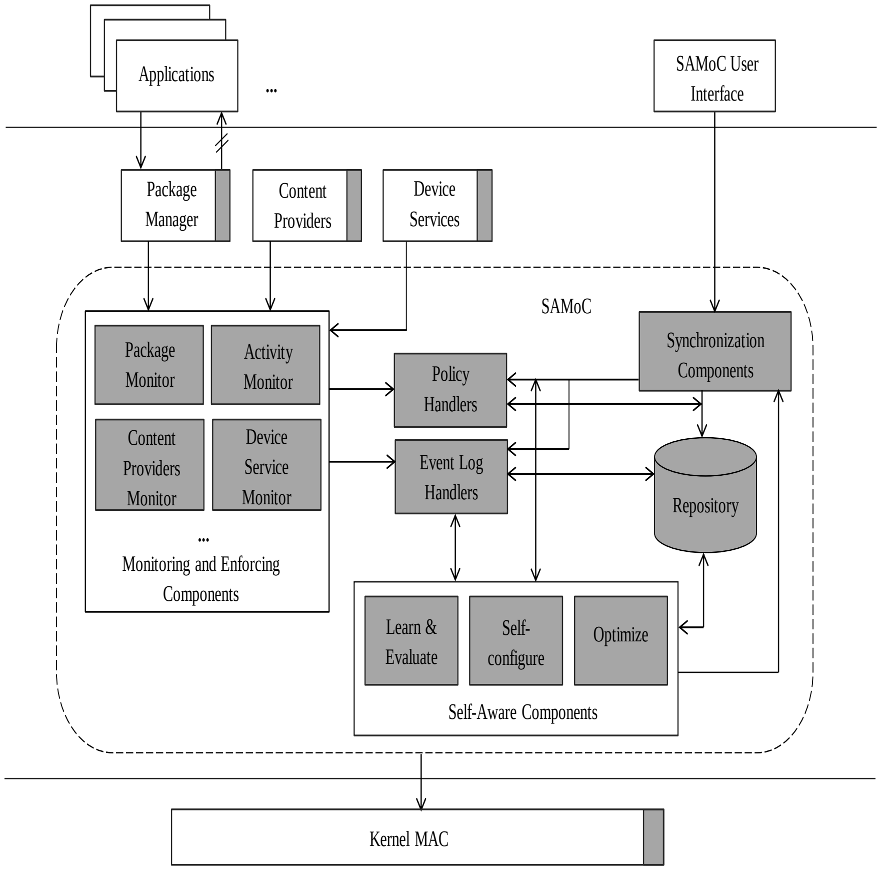

Fig. 2. Mobile software stack with SAMoC framework. The fully shaded blocks are the proposed new modules of the framework and the partially shaded blocks consist of framework module extensions in Android.

5.3 SAMoC Mobile Subsystems

One of the goals of the framework is to guarantee the existence of least degree of security at any time in order to maintain system resiliency. In order to achieve this goal, SAMoC mobile is classified into five subsystems: application installation and monitor, device resource controller, communication controller, self-aware subsystem and SAMoC user interface. The subsystems relies on different components and in certain cases, they can share the same components for the fulfilment their operations.

Application installation and monitor subsystem is responsible for handling the applications related activities including installation restriction and intercommunication between the applications.

Device resource controller functions are responsible to oversee the resource accessibility by the applications and enforce appropriate policies to restrict the access.

Communication controller will regulate the applications communication with external networks. It is in charge for allowing or refusing the outside connections based on the protocols and networks.

Self-aware subsystem observes the external environment and configure its own operations accordingly. It will also access logs, learn, create and test the established patterns, construct and apply/revoke suggestions to improve the framework operations.

SAMoC user interface is the interface application for SAMoC mobile which provides appropriate user interfaces for different subsystems. The user is able to control the subsystems or access logs through the interface.

5.4 SAMoC Mobile Software Architecture

In order to reduce development complexity as well as future enhancement and maintenance, we have adopted layered architecture for the implementation. The layered architecture allows to have complete control over the SAMoC mobile components individually and it is easier to add additional components in future. The architecture also helps in restricting the visibility of the implementations. The SAMoC mobile software architecture is shown in Figure 3.

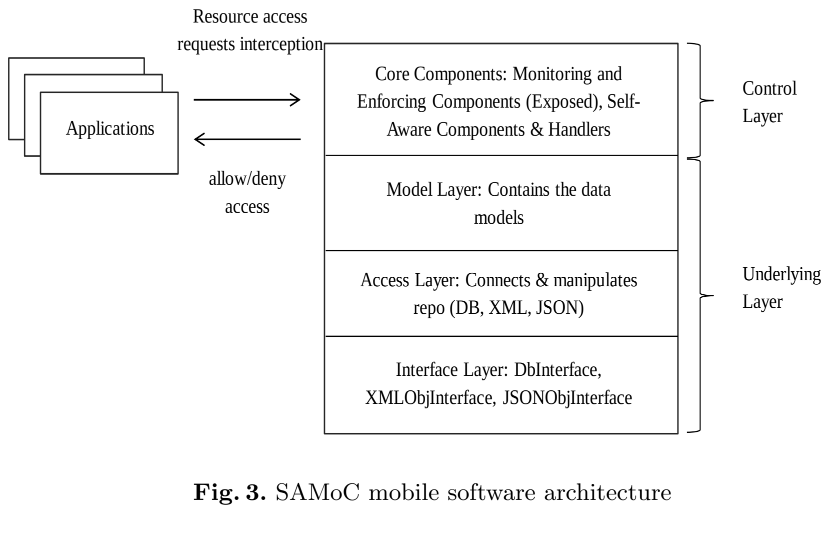
Resource access requests interception
Applications
allow/deny access
Core Components: Monitoring and Enforcing Components (Exposed), Self-Aware Components & Handlers
Model Layer: Contains the data models
Access Layer: Connects & manipulates repo (DB, XML, JSON)
Interface Layer: DbInterface, XMLObjInterface, JSONObjInterface
Control Layer
Underlying Layer

Fig. 3. SAMoC mobile software architecture

The entire architecture is grouped into two layers: control layer and underlying layer. The control layer contains all the implementation of the SAMoC mobile components (handlers, self-aware component and synchronization). The visibility of the handlers and self-aware component except the monitoring and enforcing tasks are hidden. The monitoring and enforcing components are visible and hooked into the Androids system service implementations while synchronization component provides interface to SAMoC application (SAMoC user interface). Using SAMoC user interface, users are able to perform administration tasks, such as writing policies and reviewing logs. It will be installed by default as system application in Android operating system.

The underlying layer comprises of three sublayers (Model, Access and Interface Layers) which contains the basic classes required to manipulate the repository. The model layer represents the data models of the database tables and XML structures. The access layer will manipulate the repository objects whose references will be provided by the interface layer. The Control layer components use the models to store/retrieve the information by calling the Access layer methods which in turn get the appropriate reference on the information objects from the Interface layer.

5.5 SAMoC Mobile Policy

SAMoC mobile policies are highly refined which restrict access to resources using its very own features. The efficiency of the framework depends on the policies available in the repository which will be defined by the users. The defined policies play a crucial role in determining the system efficiency during initial phase of its operations. There are certain control policies which will be delivered as settings to control the self-aware agent behaviors such as automatic enforcement of learned changes or provide them as recommendation to the user.

There are no pre-defined profiles/profile names in the SAMoC framework but pre-defined policies are available. SAMoC framework has default policies which will be enforced upon device boot for example denial of non-market application installation and denial of adb install command. Policies are created and enforced on individual account basis.

SAMoC framework does not force users to learn/develop special skills before using the system since most of the policies do not have any syntax or pre-defined structure. One of the goals of SAMoC is to make the system easy to use even for the user who has limited knowledge. To realize this, most of the policies will configured through the SAMoC user interface application. It will provide appropriate user interface in the form of settings for policy writing.

6 Challenges

At the current development stage, self-awareness including self-configuration of the self-aware agent is adopted. The self-configuration allows to configure and adapt the learn and evaluate processes, and to optimize the components of the self-aware agent according to the external environment and device usage. The learn, evaluate and optimize processes can not run continuously due to the following reasons: 1) for optimal learning, they require a sufficient number of log entries, 2) initiating the learning process results in resource wastage if there is no considerable number of new log entries recorded in-between the successive learning sessions, and 3) these processes should not consume the resources while the device is busy with other activities. These three processes run on certain time frames and the self-configuration subsystem is responsible for deciding the ideal time frame for the execution. Care has to be taken in determining the successive time frames. If the interval between successive time frames is too large, the device may fail to detect unexpected events or new type of attacks because the self-aware agent has not learned the new patterns. If it is too small, it leads to unnecessary resource consumption. As discussed, the self-configuration component is in a key role in configuring the initial settings of self-aware agent components which will have impact on identification of new threat patterns and optimization of the controlling process. To devise appropriate logic for implementation of these tasks requires careful analysis.

In the framework, two options for optimization are given. The first option is the self-aware agent taking the optimization action by itself without requiring the user's approval. The other one is the self-aware agent providing suggestions for optimization to the user. The decision to accept or deny the recommendations will be made by the user. In this case, the enterprise or individual users, who may hesitate to adopt the SAMoC framework can choose the second option. These options grant flexibility in controlling the optimization component only. The rest of the self-aware agent components will operate on their own but their processes can be reviewed through monitored logs. It is not wise to control the entire self-aware agent operation since the threats evolve dynamically. Therefore, the system which handles the threats needs to be continuously improved, especially on its own. In addition, to detect the unforeseen events or unknown threats, the system requires agility, and to be agile, it has to operate on its own.

7 Summary

The concept and specification of a self-aware access monitoring and controlling framework (SAMoC) for Android were presented. The SAMoC framework introduces the self-aware agent into the Android operating system to harden the security of the devices. The agent performs information gathering, learning from gathered information, evaluation of the available policies in comparison with the learning outcome and optimization or suggestion of the necessary improvements. The integration of a self-aware agent enables the SAMoC Framework to detect unexpected events, enforce highly refined restrictions over the resource accessibility and change the statuses of the enforcement (revoke and grant rights) during runtime. Provisions are provided to control the framework's subsystem operations, such as disabling an individual subsystem and implementing/suggesting the policies' refinements. The framework also allows to apply different sets of control policies in a single device in a multi-user environment and has provisions to engage users in the security operations and enhancements. All these features of the framework make it a prominent choice for Android security.

References

1. The Android source code, https://source.android.com/source/index.html.
2. Zhou, Y. and Jiang, X.: Dissecting android malware: Characterization and evolution. In: 2012 IEEE Symposium on Security and Privacy, pp. 95–109 (2012)
3. Sood, A. K. and Enbody, R. J.: Malvertising–exploiting web advertising. Computer Fraud & Security, 11–16. Elsevier (2011)
4. Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., Le Traon, Y., Octeau, D., and McDaniel, P.: Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. ACM SIGPLAN Notices, vol. 49 (6), pp. 259–269, ACM (2014)
5. Heuser, S., Nadkarni, A., Enck, W., and Sadeghi, A.-R.: Asm: A programmable interface for extending android security. In: proceedings of 23rd USENIX Security Symposium (SEC 2014), Usenix (2014)
6. Wang, X., Sun, K., Wang, Y., and Jing, J.: DeepDroid: Dynamically Enforcing Enterprise Policy on Android Devices. In: proceedings of 22nd Annual Network and Distributed System Security Symposium (NDSS 2015), The Internet Society (2015)
7. Bugiel, S., Davi, L., Dmitrienko, A., Fischer, T., and Sadeghi, A.-R.: Xmandroid: A new android evolution to mitigate privilege escalation attacks. Technische Universität Darmstadt, Technical Report TR-2011-04 (2011)
8. Bugiel, S., Davi, L., Dmitrienko, A., Fischer, T., Sadeghi, A.-R., and Shastry, B.: Towards Taming Privilege-Escalation Attacks on Android. n: proceedings of 19th Annual Network and Distributed System Security Symposium (NDSS 2012), The Internet Society (2012)
9. Bugiel, S., Heuser, S., and Sadeghi, A.-R.: Flexible and Fine-grained Mandatory Access Control on Android for Diverse Security and Privacy Policies. Usenix security, pp. 131–146, Usenix (2013)
10. Smalley, S., and Craig, R.: Security Enhanced (SE) Android: Bringing Flexible MAC to Android. n: proceedings of 20nd Annual Network and Distributed System Security Symposium (NDSS 2013), vol. 310, pp. 20–38, NDSS (2013)
11. Android Developers-Mainfest.permission, http://developer.android.com/reference/android/Manifest.permission.htm
12. Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B-G., Cox, L. P., Jung, J., McDaniel, P., and Sheth, A. N.: TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. ACM Transactions on Computer Systems (TOCS), vol. 32 (2), ACM (2014)
13. Fawaz, K., and Shin, K. G.: Location privacy protection for smartphone users. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 239–250, ACM (2014)
14. Wei, F., Roy, S., and Ou, X.: Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 1329–1341, ACM (2014)
15. Isomaki, M., Nieminen, J., Gomez, C., Shelby, Z., Savolainen, T., and Patil, B.: IPv6 over BLUETOOTH (R) Low Energy, 2015

16. Wang, H., Xi, M., Liu, J., and Chen, C.: Transmitting IPv6 packets over Bluetooth low energy based on BlueZ. In: proceedings of 15th International Conference on Advanced Communication Technology (ICACT), pp. 72–77, IEEE (2013)
17. Skorin-Kapov, L., Pripuzic, K., Marjanovic, M., Antonic, A., and Zarko, I. P.: Energy efficient and quality-driven continuous sensor management for mobile IoT applications. In: proceedings of 2014 International conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), pp. 397–406, IEEE (2014)
18. Angelopoulos, C. M., Evangelatos, O., Nikoletseas, S., Raptis, T. P., Rolim, J. D. P., and Veroutis, K.: A user-enabled testbed architecture with mobile crowdsensing support for smart, green buildings. In: the proceedings of 2015 IEEE International conference on communications (ICC), pp. 573–578. IEEE (2015)