IMPROVING OF EXISTING PERMISSION SYSTEM IN ANDROID OS

Volodymyr Kazymyr, Igor Karpachev

Abstract*: Smartphones for last five-ten years had become ubiquitous. Despite the fact, that these small devices are very widespread around the world - security are still being understood. As a result, the security is extremely vital. Basically security area is very underdeveloped and still vulnerable. On the one hand this article is an investigation of existing pros and cons of android security system. On the other hand it is an example of how to improve current mobile security state.*

Keywords*: security model, OS Android, functional security.*

ACM Classification Keywords*: D.4.6 Access controls*

1. Introduction

Smartphones has added additional requirements to the mobile computing. All set of application has support variety of the areas on mobile markets. Hardware, software, and different business access – such applications are available on markets (e.g. Apple Store, Google Play, Blackberry App World, Amazon App Store etc.). Moreover most of these applications are surprisingly inexpensive.

Smartphones are shifted now from simple standalone devices to a specific collaborate models (client-server architecture). In this case applications exposed a lot of private data to the external world. Applications usually seek appropriate providers of a service type at run-time, instead of binding itself to the specific implementation during development. This approach is very similar to plugin approach. Such approach has created very extensible culture of “use and extend” which lead to significant increasing of innovative applications on modern market. The very best example from existing platforms is Android operating system. The security of the android is very similar to other, and called “system-centric”. Based on that conclusion, application statically identifies all permission/interfaces required during development process. These rules will govern application’s data at installation time.

The main problem that application/developer has very limited ability to identify to whom or how these permissions will be exercised after words. Developer should assert which level of protection application desires. End user has no idea which set of permission application requires, so they do not have sufficient context to do so.

Purpose: aim of the current article is to find a better way to assign a new permission model to applications in order to improve security.

2. Methodology, Limiting factors and evaluation

Currently PayPal is a most widespread service around the world. Let's take as an example PayPal service built on OS Android. This service is kind of a bridge between the credit card of current user and purchasing of different items/features/goods from such applications as email client, Google Play market, browser, music players, etc. Considering this, PayPal is an application, which shares permissions with a lot of other services. Therefore, user ends up with a situation when it's quite difficult to decide which app should be granted with PayPal billing service. Android doesn't provide any API for clarifying this situation or enforcing a security policy based upon this, unfortunately [Miller, 2012]. Android developers put a lot of efforts in creating huge set of applications and features, which will help end user to protect device, but there is no API, which will help to protect application itself. Basically there are three main features, which are not available in an android application security framework:

1) Permission assignment policy - Applications have limited ability to control to whom permissions for accessing their interfaces are granted, e.g., white or black list applications.

2) Interface exposure policy - Android provides only rudimentary facilities for applications to control how their interfaces are used by other applications.

3) Interface use policy - Applications have limited means of selecting, at run-time, which application's interfaces they use.

This paper introduces the Secure Application Approach (SAA) that extends the existing Android security architecture with policies that address these key application requirements. Figure 1 below depicts basic Secure Approach with payment system on Android OS.

Applications provide installation time policies that regulate the allocation of permissions that protect their interfaces. At runtime, communication between or access of applications is subject to security policies declared by both the caller and callee applications. Saint policies are much more superior than the static permission checks currently available in Android by limiting access based on runtime state, e.g., phone or network configuration, location, time, etc. The Saint framework will be defined and the complexities of augmenting Android with extended policy enforcement features are will be discussed, and mechanisms for detecting incompatibilities and dependencies between applications will be developed. The discussion begins with an encouraging example.

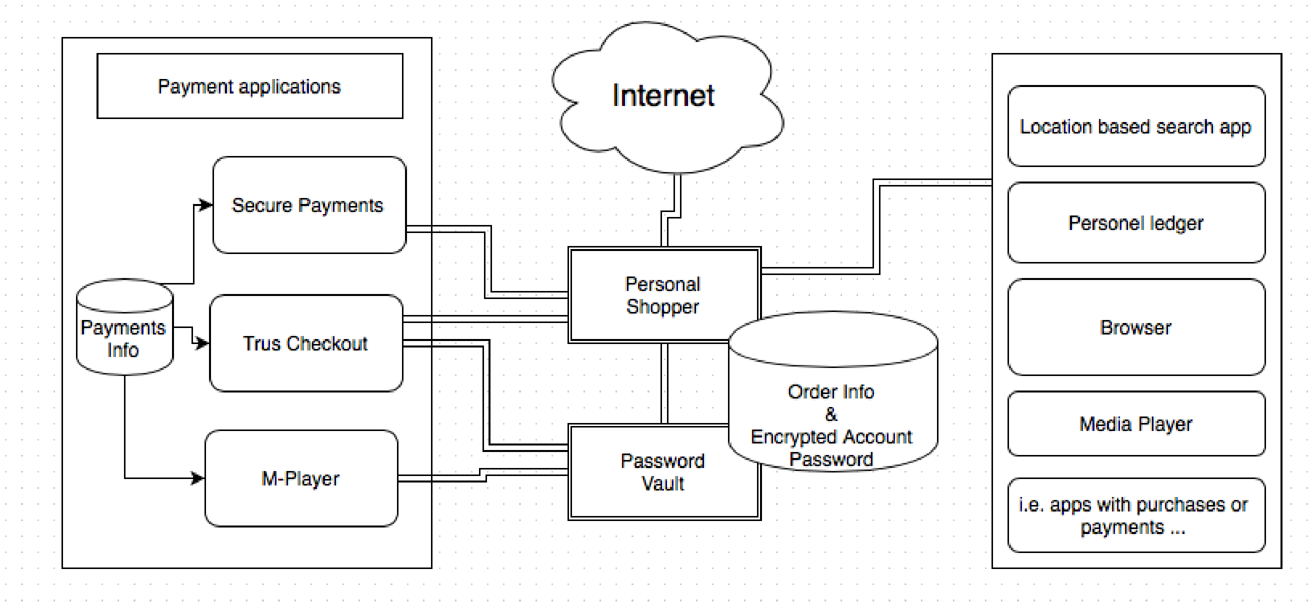

Figure 1. Discretion of the user and interacts with vendors and payment applications to purchase them.

3. Smartphone applications security

Figure 1 presents the made-up smartphone shopping application “Personal Shopper”. “Personal Shopper” tracks the items a user wants to buy and interacts with other payment applications to purchase them. User select the desired goods through the user interface of the smartphone (theoretically by clicking on units on a browser, media player, etc.), creating a seller independent “shopping cart”. Users later purchase items in one of two ways. The user can direct the application to “find” an item by clicking on it. In this case the application will search for all known online vendors or various shopping search sites (e.g., Google Product Search) to find the desired item. Where various vendors provide the same item, the user then selects their vendor choice through a provided menu. The second way for finding a desired product is by geography—a user walking through, for instance, a mall, the usage of location based search application can alert them on the availability of the product in the nearest physical store. In this case, the user will be directed to the brick-a-brick vendor to obtain the item.

Unrelatedly of how the item is found, “Personal Shopper’s” second objective is to aid the purchase process itself. In this particular case, it works with provided example checkout applications as SecurePayer and TrustCheckout. Personal- Shopper gets the access to checkout applications and acts as a mediator between the buyer and the merchants to both aspects as improving the efficiency of shopping and customer privacy protection. Usually procedure is quite simple – user provides credentials in order to authenticate to service. After their completion, all the transactions are recorded in a personal ledger application.

Consider a few (of many) security requirements this application suggests:

1) Only trusted payment services should be ever used by the PersonalShopper. Figure 1 shows, it may trust only SecurePayer and TrustCheckout, but it does not trust any other unknown payment providers (e.g., the M-Payer provider).

2) PersonalShopper may only want to restrict the use of the service to only trusted networks under safe conditions. For instance, it may wish to restrict searches while the phone is roaming or highly unprotected areas (e.g., airports) or while battery is low.

3) The use of certain version of the service software may be required by the PersonalShopper. Such as, the pass- word vault application v. 1.1 may contain a bug that disclosures password information. Thus, the application would require the password vault be v. 1.2 or higher.

4) PersonalShopper may wish to confirm transaction information is not leaked by the phone's ledger application. Thus, the application wishes to only use ledgers that don't have access to the Internet.

5) Security requirements may be placed on PersonalShopper by the applications and services it uses. For example, to save location privacy, the location based search application may only offer PersonalShopper location information only where PersonalShopper has the permissions to access location information itself, e.g., the phone's GPS service.

None of these policies are currently supported by today's Android security system. While some of these may be partially emulated using combinations of complex application code, permission structures, and code signing they are simply outside the scope of Android's security policy. As a result (and core to our widespread experience building systems in Android), applications must cobble together custom security features on top of the fundamental structures currently provided by the Android system. Where possible at all, this process is ad hoc, error prone, repetitive, and inexact [Anderson, 1992].

What is needed for Android is to provide applications a more semantically rich policy infrastructure. The following investigation begins by outlining the Android system and security mechanisms. Section IV examines a range of policies that are potentially needed for fulfillment of the applications' security requirements as well as highlighting those that cannot be satisfied by the current Android. Further, goals, design, and implementation of the Saint system is introduced

«Android»

Android is an operation system, which has been developed by OHA – Open Handset Alliance in 2005. Android became extremely popular among developers and end-users for it's open source nature and common language for development Java (or C++ - JNI).

The Content Provider API implements an SQL-like interface; however, the application developer is left with the backend implementation. The API involves support to read and write data streams, e.g., if Content Provider shares files. Unlike the other component types, Content Providers are not addressed via Intents, but rather a content Uniform Resource Identifier (URI). That is the collaboration of application components for which we are more concerned. Figure 2 shows the common IPC between component types.

The bases of the Android's application-level security framework are permission labels, which are enforced in the middleware reference monitor [Enck, 2009]. Basically, a permission label is a unique text string that can be described by both the OS and third party developers. Android defines various base permission labels. From an OS centric point, permission labels are statically assigned for the applications, indicating the sensitive interfaces and resources available at run time; the permission set cannot grow after installation.

Application developers identify a list of permission labels the application requires in its package manifest; however, requested permissions are not always granted.

Permission label descriptions are distributed across the framework and package manifest files. Each definition specifies "protection level." The protection level can be "normal," "dangerous," "signature," or "signature or system." After application installation, the protection level of requested permissions is checked. Permission with the "normal" protection level is always granted [Cheswick, 2003]. Permission with the "dangerous" protection level is always granted if the application is installed. However, the user must confirm all requested dangerous permissions together. Finally, the signature protection level influences permission granting without user input. Each application package is signed by a developers' key (as is the framework package containing OS defined permission labels). A signature-protected permission is only granted if the requesting permission labels is signed by the same developer key that is signed the package defining the permission label. Many OS defined permissions use the signature protection level to ensure that the access is granted only to the applications distributed by the OS vendor. Finally, the "signature or system" protection level operates the same as the signature level, moreover the permission is granted to applications that are signed by the system image key.

Additional use for permission label policy model is to protect applications from each other. Most permission label security policy could be found in an application's package manifest. As mentioned before, the package manifest specifies the permission labels that correspond to the application's functional requirements. The package manifest also specifies a permission label to protect each component of the application (e.g., Activity, Service, etc). Inter Process Communication may initiate communication in another or the same component of if target has specific permission.

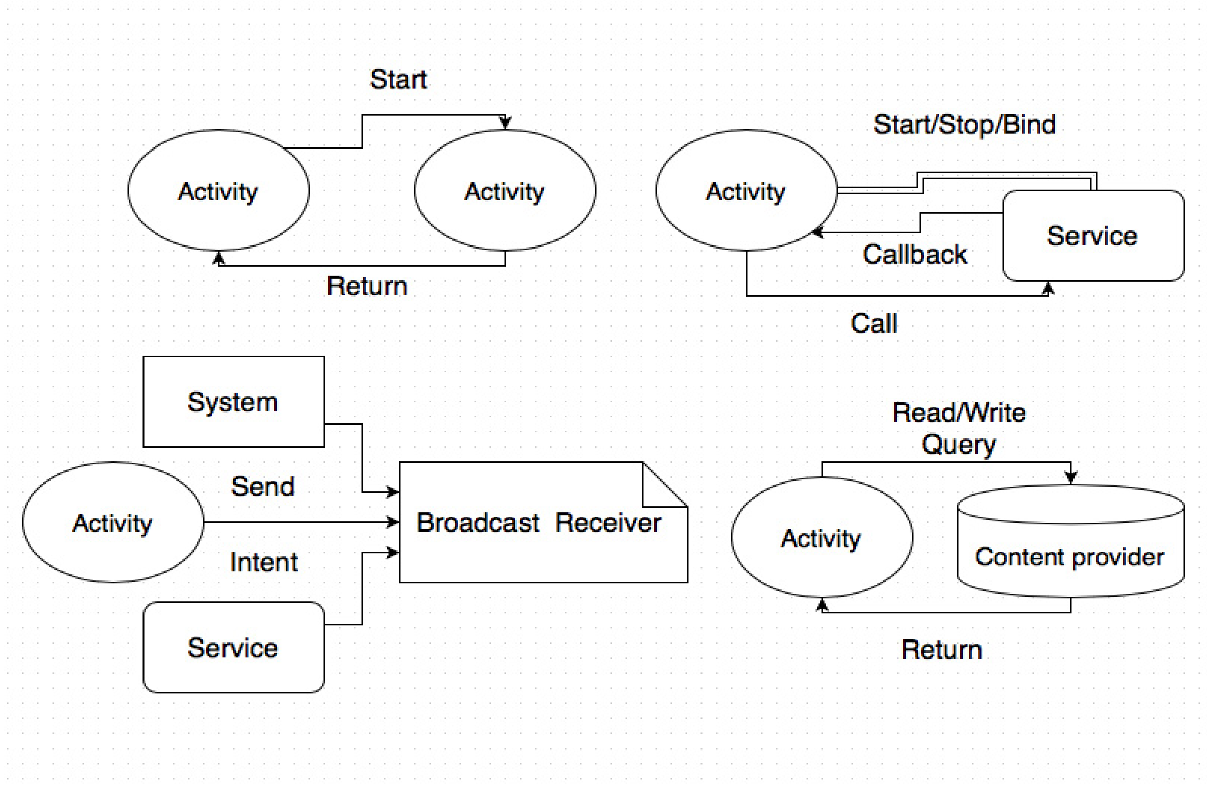

Figure 2. Typical Android application component IPC

While Android is based on Linux, the middleware presented to application developers hides usual OS concepts. The platform focuses on applications, and much of the core phone functionality is implemented as applications in the same manner used by third party developers [McDaniel, 2012].

Android applications are primarily written in Java and compiled into a custom byte-code (DEX). Each application executes in a separate Dalvik virtual machine interpreter instance running as a unique user identity.

From the perspective of the underlying Linux system, applications are ostensibly isolated. This design minimizes the effects of a compromise, e.g., an exploited buffer overflow is restricted to the application and its data [Cheswick, 2003].

All inter-application communication passes through middleware's binder IPC mechanism (our discussion assumes all IPC is binder IPC). Binder provides base functionality for application execution. Applications are comprised of components. Components primarily interact using the Intent messages. While Intent messages can explicitly address a component in an application by name, Intent versatility is more apparent for Intent messages addressed with implicit action strings, for which the middleware automatically resolves how to handle the event, potentially prompting the user. Recipient components assert their desire to receive Intent messages by defining Intent filters specifying one or more action strings.

There are four types of components used to construct applications; each type has a specific purpose. Activity components interface with the user via the touchscreen and keypad. Typically, each displayed screen within an application is a different Activity. Only one Activity is active at a time, and processing is suspended for all other activities, regardless of the application. Service components provide background processing for use when an application's Activities leave focus. Services can also export Remote Procedure Call (RPC) interfaces including support for callbacks [Bishop, 2003]. Broadcast Receiver components provide a generalized mechanism for asynchronous event notifications. Traditionally, Broadcast Receivers receive Intents implicitly addressed with action strings. Standard event action strings include "boot completed" and "SMS received." Finally, Content Provider components are the preferred method of sharing data between applications.

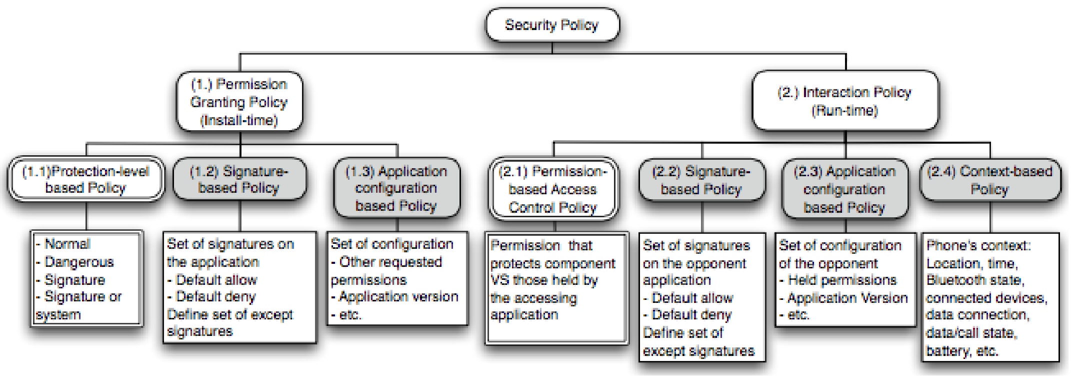
Security Policy
(1.) Permission Granting Policy (Install-time)
(2.) Interaction Policy (Run-time)
(1.1)Protection-level based Policy
(1.2) Signature-based Policy
(1.3) Application configuration based Policy
(2.1) Permission-based Access Control Policy
(2.2) Signature-based Policy
(2.3) Application configuration based Policy
(2.4) Context-based Policy
- Normal - Dangerous - Signature - Signature or system
Set of signatures on the application - Default allow - Default deny Define set of except signatures
Set of configuration - Other requested permissions - Application version - etc.
Permission that protects component VS those held by the accessing application
Set of signatures on the opponent application - Default allow - Default deny Define set of except signatures
Set of configuration of the opponent - Held permissions - Application Version - etc.
Phone's context: Location, time, Bluetooth state, connected devices, data connection, data/call state, battery, etc.

Figure 3. Policy tree illustrates the example policies required by applications. The double-stroke boxes indicate support by the existing platform.

Using this policy and permission protection levels, application developers can specify how other applications access its components. The permission label-based security policy stems from the nature of mobile phone development. Manually man- aging access control policies of hundreds (thousands) of potentially unknown applications is infeasible in many regards. Hence, Android simplifies access control policy specification by having developers define permission labels to access their interfaces. The developer does not need to know about all existing (and future) applications. Instead, the permission label allows the developer to indirectly influence security decisions. However, herein lies the limitations of Android's security framework.

«Application policies»

It was explored a myriad of applications as a means of understanding the appropriate set of policy expressibility. Initial policy taxonomy is presented in Figure 3.

The permission-granting policy regulates permission assignment. In addition to controlling permission granting using Android's protection level-based policy (1.), an application A may require signature-based policy (1.2) to control how the permissions it declares are granted based on the signature of the requesting application B (A and B may be signed by different developer keys). Instead, the policy grants (or denies) the permission by default with an exception list that denies (grants) the applications signed by the listed keys. An application may also require configuration-based policy (1.3) to control permission assignment based on the configuration parameters of the requesting application, e.g., the set of requested permissions and application version.

The interaction policy (2.) regulates runtime interaction between an application and its opponent. An application A's opponent is an application B that accesses A's resources or is the target of an action by A, depending on the access control rule (i.e., B is A's opponent for rules defined by A, and A is B's opponent for rules defined by B). Android's existing permission-based access control policy (2.1) provides straightforward static policy protection, as described in Section III. However, this policy is coarse-grained and insufficient in many circumstances. Applications may require signature-based policy (2.2) to restrict the set of the opponent applications based on their signatures. Similar to above, the default-allow and default-deny modes are needed. With configuration- based policy (2.3), the applications can define the desirable configurations of the opponent applications; for example, the minimum version and a set of permissions that the opponent is allowed (or disallowed). Lastly, the applications may wish to regulate the interactions based on the transient state of the phone. The phone context- based policy (2.4) governs runtime interactions based on context such as location, time, Bluetooth connection and connected devices, call state, data state, data connection network, and battery level. Note that initially, policy types 2.2 and 2.3 may appear identical to 1.2 and 1.3; however, the former types also place requirements on the target application, which cannot be expressed with 1.2 and 1.3. However, 1.2 and 1.3 are desirable, because when applicable, they have insignificant runtime overhead. We now present two example application policies related to our motivating example, Personal Shopper, which interacts with checkout applications, password vaults, location-based search applications, and personal ledgers [Enck, 2009].

Install-time Policy Example: In our Personal Shopper example, the location-based search application (com.abc.lbs) wants to protect against an unauthorized leak of location information from its “Query By Location” service. Permission granting policy can be applied when the Personal Shopper requests the permission com.abc.perm.getloc used to protect “Query By Location”. It needs application configuration-based policy to specify that for the permission com.abc.perm.getloc to be granted, the requester must also have the “ACCESS LOCATION” permission.

Run-time Policy Example: To ensure that the checkout application used for payment is trusted, their signatures must be checked. The Personal Shopper needs signature-based policy to specify that when the source “Personal Shopper” (com.ok.shopper) starts an Activity with action “ACTION PAY”, the policy ensures resolved applications are signed by keys in a given set.

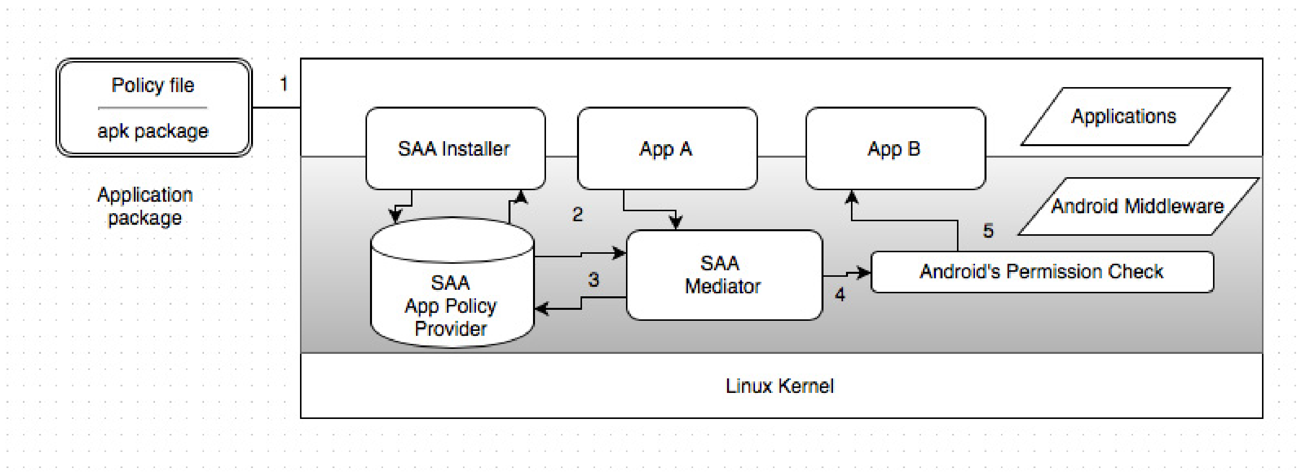

Figure 4. Process (a-c) with additional permission granting policies and mediates component IPC (1-5) to enforce interaction policies specified by both the caller and callee applications.

4. Conclusions

This article addressed existing limitations in android operation system. Particularly in real-time permission assignment and inter process communication policies. The main idea was to provide clear way of expose android permissions to application's functionality. Driven by an analysis of the PayPal service this article gives an example of an initial taxonomy of relevant security content. Current approach has a lot of drawbacks as well and not going to replace existing permission assign policy in android operation system. This is just an example of how these technologies can be improved.

Bibliography

[Miller, 2012] Miller C. Mark D. Independent Security Evaluators, Exploiting Android Devices 2012.

[Anderson, 1992] Anderson J. P. Computer security technology planning study, volume II. 1992, 82-94.

[Enck, 2009] Enck W. Understanding Android Security, 2009, 50-58.

[Cheswick, 2003] Cheswick W., Firewalls and Internet Security: Repelling the Wily Hacker, 2003, 82-94.

[McDaniel, 2012] McDaniel, Prakash A. Methods and Limitations of Security Policy Reconciliation, 2012, 77-87.

[Bishop, 2003] Bishop M, *Computer Security:A standad science*, 2003.

[Enck, 2009] Enck W., Ongtang M., McDaniel P. On Lightweight Mobile Phone Application Certification, 2009.

Authors' Information

Volodymyr Kazymyr – Dr. Sc., Prof. Chernihiv, National University of Technology, 95, Shevchenko street, Chernihiv-27, Ukraine, 14027; vvkaymyr@gmail.com

Major Fields of Scientific Research: computer science, information technologies, complicated computer systems.

Ihor Karpachev – Ph.D. Student, Chernihiv National University of Technology, 95, Shevchenko street, Chernihiv-27, Ukraine, 14027; benchakalaka@gmail.com

Major Fields of Scientific Research: security problems in existing mobile operating system.