An Enhanced Android Security Model Against Collusion Attacks

Yunpeng Zhang

Department of Computer Science

Boise State University

Boise, ID, U.S.A.

yunpengzhang@boisestate.edu

Dianxiang Xu

Department of Computer Science

Boise State University

Boise, ID, U.S.A.

dianxiangxu@boisestate.edu

*Abstract*—With the increasing popularity of Android platform, various access control models have been proposed to enhance Android’s security. For elevation of privilege attacks such as collusion attacks, confusion agent, and shared UID, the existing access control models are inadequate. Based on the analysis of the XManDroid model for defense against collusion attacks and its mandatory access control model, this paper proposes an improved security model. It adopts the communication virtual colored map, and records the historical communication by dynamical colors, then raised the fine-grained access control model based on historical access. The model proposes the separation process of the applications’ original colors and the dynamic colors, thus avoiding the color contamination and solving the shortcomings of the Mandatory access model.

Keywords-Android; collusion attacks; communications history; application staining

# Introduction

With the widespread applications of Android, Android’s security issues could no longer be ignored. For security issues of Android system, a number of models have appeared. At present, the corresponding security research work starts from the hardware layer, the virtual machine monitor layer, the operating system kernel layer, application framework layer and the application layer [3]. The main methods are used include authorization mechanism, run-time dynamic permissions testing, static analysis and others.

However, these models analyze the applications statically. For dynamic attacks caused by collusion attack, confusion agents and UID sharing, these models are not good solutions. For the current attack mode, XManDroid model raises a good solution. The model builds a communication connection graph, which achieves a model that can prevent confusion agents and collusion attacks at the same time by setting covert channels as virtual nodes and combining with a series of access control policy. But this communication connection graph is undirected; it will lead to considerable errors in the detection, and could not detect the covert channel in the kernel layer. At the same time, this model is limited to the combination of risky permissions existed between two applications. For attacks between multiple applications, it is powerless. For this model, Reference [4] proposed a mandatory access control model. The model follows the ideology of communication connection graph, but it builds a directed graph and expands the node properties with nodes colors including nodes’ original colors and dynamic colors. The dynamic colors would be dyed as the communication process in order to achieve the purpose of real-time communication history records. The model indeed achieves the purpose, but it also introduces another problem. In this model, the nodes’ color will be continuous dyeing as the communication process, when the call ends, vertex’s colors do not get timely recovery, when the next call comes, it may be treated as dangerous operation even if it is a safe call between applications.

This article first analyzes the XManDroid model and discusses about the operational mechanism of mandatory access control. Then with the problems of the mandatory access control model, the paper raises an improved access control model, and finally makes verification toward the model.

# Android Security Model

Android security design includes following two principles, the first, applications running under the Android OS do not have permissions to perform harmful actions to other applications, the operating system or users. These operations include reading the users’ private data (such as Contacts or Email), files from other applications and so on. The second, the processes of Android applications run in a secure "sandbox". It could not interfere with other applications unless it is explicitly allowed by declaring permission requests. These permission requests can be processed in different ways. Toward some particular permission requests, it should be automatically allowed or prohibited, based on the certificates and prompts to the users.

## Two Important Security Elements Android Data Security Mechanis

1) UID: each application installed in the Android phone will be allocated with a unified Linux user ID, and the system will create a sandbox for each application in order to prevent the effects from other applications (or other programs affect it). The user ID would be assigned to the application when they are installed in the mobile phone, and remain permanently in the device. Also any file created by the application would be given the user ID, and it could not be accessed by other applications under normal conditions.

2) Permission: it is a label that allows the users’ or applications’ operations including open data file, send messages and calls Android components. Android permissions are not only to set to security identity, but also to achieve some special operating procedures (such as apply for the system service).

## Weakness in Current Android System

As the application field continues to spread, more and more people pay attention to the security problems on the Android platform. Although the original system uses a mechanism including permissions, sandbox, application signatures to ensure the security of the platform, but there are many weakness in Coarse-grained authorization completed by users and developer-centered access control mechanism, which make system security greatly reduced, especially vulnerable to the privilege escalation attacks of the application layer. As for Weakness existed in current systems, many researchers have proposed many solutions. Toward collusion attack, confusion agent and shared UID, the author proposes an improved access control model based on the XManDroid and mandatory access control model.

# XManDroid Model and Mandatory Access Control Model

## XManDroid model

XManDroid[5] is a secure model which is based on detecting policies and preventing application the layer’s attack by privilege attack. A systematic state graph is defined in the model. In the systematic state graph, vertexes consist of application, provider of system contents and system services; at the same time, a series of mandatory access policies are defined and each policy represents a combination of risky permissions and each vertex represents the state of the application permission (Yes / No).

The XManDroid extends the Android application framework and it consists of three modules: (a) operation monitoring, (b) installing application, (c) installation program of system policies.

It extends the Android application framework by three modules: (a) operation monitoring, (b) installing an application, (c) the system setup program strategy.

On the basis of policy set, an undirected graph about applications is established. While there is a request that has occurred between applications, the model use a series of algorithms to check the existence of matched permission combination with the policy set from start point to end point to determine whether to reject the request.

XManDroid model is an undirected communication graph. Therefore, the model is hard to grasp the accuracy. It will produce error report and can only check out the situation when the combination of risky permission exists in two applications.

## Mandatory access control model

Mandatory access control model inherits the idea of a communication connection graph. It expands the vertexes and adds the vertexes colors which include vertexes’ original colors and dynamic colors. Vertexes’ dynamic colors will be dyed with the changing of the vertex’s communication state to achieve the purpose of recording historical communication. In the event of a call, the call will use monitoring algorithm to search for the invocation path and then transmits the caller’s colors to the callee’s based on the model’s human-created policy set and directed and colored graph. When a matched dangerous combination which is defined in the policy set is found, the decision algorithm will decide how to deal with this situation.

Access control module consists of intermediate software layer, Mac Checker, and inner nuclear layer SELinux. The former is responsible for making access decisions according to the vertex colors and the latter is in charge of sensing the route of the inner nuclear layer’s establishment and control (including creating and deleting of files, internet Sockets). At the same time, SELinux needs MAC Checker to control virtual vertexes, including reporting control information and making feedback about decision report. In the respect of reporting information, its major task is expanding SELinux’s access control port to enable it deliver information to security service of intermediate software layer; in another respect, the major problem is writing a native code library to operate on SELinux, and then it will expand a Java interface by JNI. After MAC Checker receives the secure request from inner nuclear layer and makes decisions according to vertexes’ colors, it will return the decision to SELinux through Java port.

Since the model only considers the vertexes dyeing without considering the color cleaning, after the end of the communication, if the application has been terminated and the data generated by it have been destroyed, but as a result of the model have not eliminated the condition that it can dye to other applications, it will cause color contamination and result in an error report.

# Enhanced Access Control Model

## This section proposes the idea that ​​separate applications’ dynamic colors and applications’ original colors, thus avoiding the color contamination.The model

The proposed model has the following assumptions: (1) when a vertex has a combination of risky permissions, it will apply the corresponding system service by default. (2)This model does not consider the impact of the calling sequence of combination of risky permissions.

Definition 1: System communication virtual connection graph G = (V, E) is a directed graph which is made up of the vertex set V and edge set E. V consists of installed applications A, system services S, files F, and Internet socket I, the attribute of Communication vertex V is marked by a unique identifier U (V). E.g. package name of an application, virtual UID of system component, file path and file name, Internet socket, IP address and port number, etc.. E represents communication links between the vertexes, including ICC calls (M) among applications and system services, file accesses (K) including read and write between applications and files, Internet socket operations (H) between among applications and sockets.

Definition 2: The access permissions defined for all vertex in the graph are PERMS = {p1, p2, ..., pn}, the acquired permissions of each vertex composes a vertex set of permissions Perms, Perms = {pi, ..., pk}, i <= n, k <= n.

Definition 3: Given color set COLORS = {c1, c2, ..., cn} and permissions PERMS , Let ptc(p) denote the color of permission p. The original colors of one application is STATIC = {c1, c2, ...,cm} is a color set of the permission labels in the application’s Manifest file, c1, c2, ..., cn represent the colors of permission labels, COLORS ﬤ STATIC. The set of all STATIC in the system is STATICS, STATICS = {STATIC\_1, …..., STATIC\_m}, m is the number of vertexes of graph. DYNAMIC= {c1, c2, ...,ci} is a collection of newly created dynamic colors when the request is allowed, which includes the colors of the called vertex and includes the DYNAMIC or DYNAMIC of the caller COLORS ﬤ DYNAMIC. When the call is not completed, dynamic colors would be passed on in the color transfer. Vertexes’ DYNAMICs would be cleaned ​​after the final decisions. Pre\_color = {c1, c2, ... , ci} is a color collection created for the process of the detection algorithm, this collection is a copy of the DYNAMIC to make decision by matching with BlackLists, COLORS ﬤ Pre\_color. After decisions, Pre\_color would be emptied automatically. When the communication request is allowed, the color transfer will happen, which is called dyeing. The process of dyeing will create a new file of dynamic colors to record the historical communications of vertexes.

Definition 4: Threat t is a collection of permissions applied by direct or indirect ways, t= {p1, p2, ......, pn}. When the application has these permissions, it will have the ability to destroy the confidentiality and availability of the system. A collection of all the threat is Threats, Threats = {t1, t2, ......, tm}.

According to definition 1 and definition 4, BlackList is definite, BlackList = {ptc (P1), ptc(P2), ...... , ptc(Pn)}, Rank, Number, that is to say, the Blacklist is a collection of certain threats permissions’ colors. For example, a vertex contains the color of the two permissions, FIND\_LOCATION (obtaining information of user’s location) and INTERNET (allow to connect internet), then the position information may be sent to the internet by this vertex, causing the user’s private information leak. Because different collections will produce different threats to user’s security, so this paper definitely ranks to represent level of a threat, it would determine different treatments. When a vertex has a high danger, the system would reject directly; as for other vertexes, the system would prompt the user to decide. The Number is a count for the colors in a threat. Because there are many threats, BlackLists is defined, BlackLists = {BlackList1, BlackList2, ... ,BlackListn}.

Definition 5: RECORDER is a collection of threats that has been found, RECORDER = {BlackList1, BlackList2, ... ,BlackListn}, BlackList1, BlackList2 ..., BlackListn is threats found by OS/Apps supplier. The collection is recorded separately, when requests occur at the vertex call, it would be checked in RECORD firstly. If the same request has occurred before, the system would just make the same judgment as the previous decision. If not, then track step by step and conduct dyeing to determine whether to allow the request.

## Model method

**Method 1: Setting up and maintaining original colors**

When the number of vertexes changes, this method will be trigger.

Input: Permission color file, the existing vertexes

Output: File of vertex’s original colors

Process steps:

1) When the access control model is installed at the first time or there is a new vertex added/deleted /modified, this model will rescan all the vertexes’ permissions.

2) After the model is installed at the first time, it will scan all the existing vertexes’ permissions. In detail, the specific scanning method is detecting file Manifest.xml [4] which is contained by each vertex. The model will make a record of each vertex’s permission set when it is scanned, the permission set of each vertex will be stored in a permissions collection named Perms. In the model, each vertex will have its own Perms accordingly. At last, all the Perms will be stored in a permission file named PERM. In the situation where there is a new vertex is added to the model, the control model will scan this new vertex in the same way. When the model is checking the new vertex’s permission, the newly detected permission will be recorded in a collection named Perms\_new. After the new vertex is checked, Perms\_new will be added to collection PERM; while an existing vertex is deleted, its corresponding Perms will be removed from PERM; when a vertex’s permission is altered, and the model will rescan this vertex and create a new Perms to replace its previous Perms.

3) PERMS->COLORS is a one to one mapping from PERMS to COLORS defined in Definition 3. Therefore, after establishing all the vertexes’ vertex permission collection Permission, the model will conduct a matching scan among Permission to transmit all the permissions to their corresponding colors. In the matching process, the model will create a color collection named STATIC for each vertex correspondingly, STATIC = {ci，…，ck}, ci, …,ck represent each permission’s color. After each vertex has its respective collection STATIC, model will create a color collection file named STATICS, STATICS = {STATIC\_1， ……，STATIC\_m}.

4) At last, file STATICS will be saved and output.

**Method 2: Detection**Input: dynamic color collection files (DYNAMIC), a blacklist collection file (BlackList), temporary color set file Pre\_color)

Output: (Yes, Safe) / (No, output dangerous level)

Process steps:

1) Initialization. Read from file DYNAMIC and BlackList, after reading, a copy of BlackList named BlackList\_copy should be created, BlackList\_copy = {black1，……，blacki ，…，blackn} ，blacki={bc1，bc2，……，bcn，}. At the same time, this paper defined some property value for blacki, such as Rank, Number and Count. Among the values, Rank represents risk assessment level of file blacki and it will determine the model how to operate on the correspond call request; number represents the number of bc included in file blacki and it is designed to judge whether blacki is existing in some special call; Value Count is a temporary variable and its initial value is 0. It is used to record the number of times which a color included in DYNAMIC appeared in a dangerous operation collection. At the same time, model needs to assign all the color included in DYNAMIC to the temporary color collection Pre\_color to make Pre\_color = {ci ，……， ck}.（i<k，i<n,k<n）

2) Risky permission matching. Scan Pre\_color and detect all the colors included in Pre\_color orderly. While scanning, fetch the pointing color (such as ci) and use ci to have an equal match with all the color included in BlackList\_copy. If there is a same ci in blacki, the Value Count of blacki should plus one. Repeat above works until all the Pre\_color’s colors were scanned.

3) Risky permissions scan. Re-test all the collection file named blacki (i is an integer) contained in BlackList\_copy and check its value Number and Count. While scanning the collection orderly, if there is a collection whose value Number equal to value Count, it proves that this black is involved in this request call. In this condition, we need to create a involved danger permission collection Ref\_Black，Ref\_Black={blacki，…，blackj}(blacki，…，blackj is involved black). Then, the model will add these detected dangerous combinations to Ref\_Black. Repeat this action until all the risky permissions combinations are checked.

4) Classification of risky permissions. As different risky permissions will have different influences on system security, the model defined value Rank in Blacklist to identify each threat’s danger level. After all the involved danger permission combination are found, model will check the Ref\_Black’s danger permission combination orderly and model will create a variable R to evaluate its danger level, set the initial value of R is empty. Here, empty means it is in the lowest level. While checking Ref\_Black, model will compare the value Rank of the pointing combination in Ref\_Black with R. If Rank is greater than R, replace R with Rank; otherwise, continue to check the next danger permission combination. Repeat this action until all the combinations of Ref\_Black are scanned.

5) After the whole scanning process is finished, judging whether there is combination of risky permission appears. If it is true, the model will return “YES” and allow this request call; otherwise, it will return “NO” and deal with this request call according to value Rank.

**Method 3: Dyeing** Input: Dynamic color of callee vertex (DYNAMIC), Original color of called vertex (STATICS).

Output: Dyed dynamic color collection

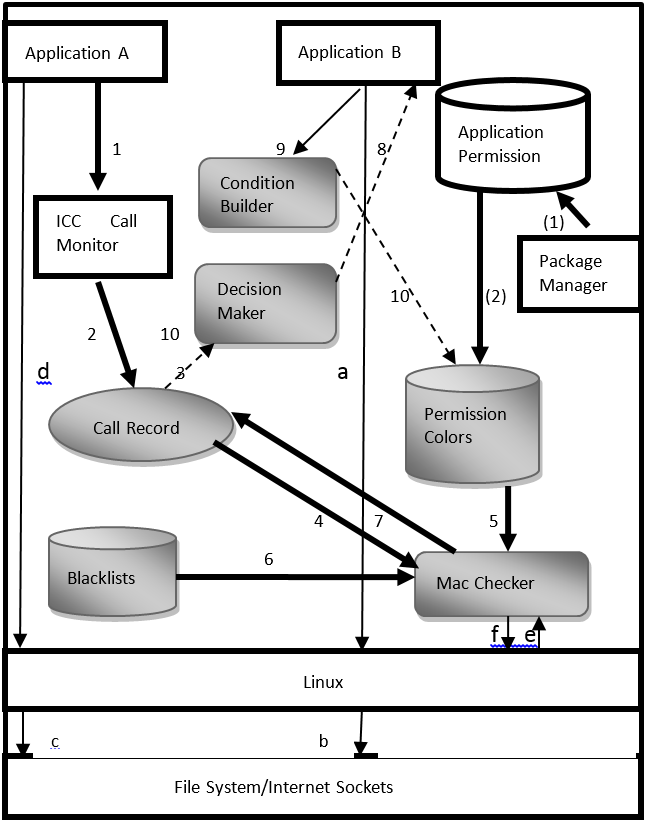
Announcements: Before a series of request is finished, the colors being transmitted in dyeing algorithm are always dynamic colors.

Process steps:

1) While the model is being installed at Android at the first time, it need to execute algorithm 1 and find out all the vertexes’ permission and its corresponding color, then, create an original color collection file STATICS and save it. In the later use of this model, when the number of vertexes changes (such as add or delete application, creating files and create sockets), step 1 will be executed again, which means dyeing algorithm will be initialized again. At the same time, the model needs to load all-side blacklist color files BlackList when it is initialized. Detection algorithm needs BlackList to realize matching algorithm.

2) When there is a request call be given in the system, the model will search whether the call was been given in the past from detected danger combination collection M. If there is a same call record in M, model will invoke the past dealing method. Otherwise, it means it is a new call in the system and the model will execute step 3.

3) Now, model will use detection algorithm to detect the required permissions in the call. If there isn’t risky permission combination found in the detected result, the model will allow the request call and dye the callee’s dynamic color collection file DYNAMIC. The process of dyeing is making a union set of caller’s dynamic color collection file DYNAMIC of caller’s dynamic color collection file DYNAMIC1 and callee’s dynamic color collection file DYNAMIC. It can be expressed in this way: DYNAMIC=DYNAMIC∪DYNAMIC1. If the model finds there is a risky permission combination through detection algorithm, it will not continue to execute dyeing algorithm. It needs only add this call path to detected danger combination collection M so that it can make a quick judgement to this call request when the call occurs later on.



Unmodified components  New components

Figure 1 Model operating mechanism

## Model Structure and Operational Mechanism

As is shown in Figure 1, the Enhanced access control model run as follows:

1. Vertex update. When the model is first installed or when a new vertex is added, the model will perform (1), (2) step, scan the vertexes’ permissions and find the corresponding color collection by Method 1, store the result into STATICS. When a vertex is deleted, all vertexes of the model will be completely re-initialized.

2. Monitor calls. As is shown in Figure 1, when a call occurs between two vertexes (in figure 1, Application A is called by Application B). Detect if there is any request by the ICC Monitor first, then the model will send the request to the Call Record to determine whether there was a same call request in history. If so, the historical decision would be sent to the Decision Maker in step 3, the Decision Maker determines whether to allow the call by that decision; on the contrary, the step 4 would be executed (dotted line represents conditional judgment). In the Mac Checker, method 2 will be executed to determine whether allow the request, algorithm 2 includes the step 5, 6 in the graph 2. After the check is completed, the model will save the result in the Call Record. After that, step 3 will be executed to determine whether to allow the request. If allowed, the permission requested by B will be sent to Application B; if not allowed, then the call ends. (Step 8)

3. Logically judgment. After one call is executed, the model will check whether the call has been over. In some cases, a collusion attack may be completed simultaneously by multiple vertexes. If the call is over, the model will clean all DYNAMICs of involved vertexes. If not completed, the model will monitor the next call. (Step 9, 10)

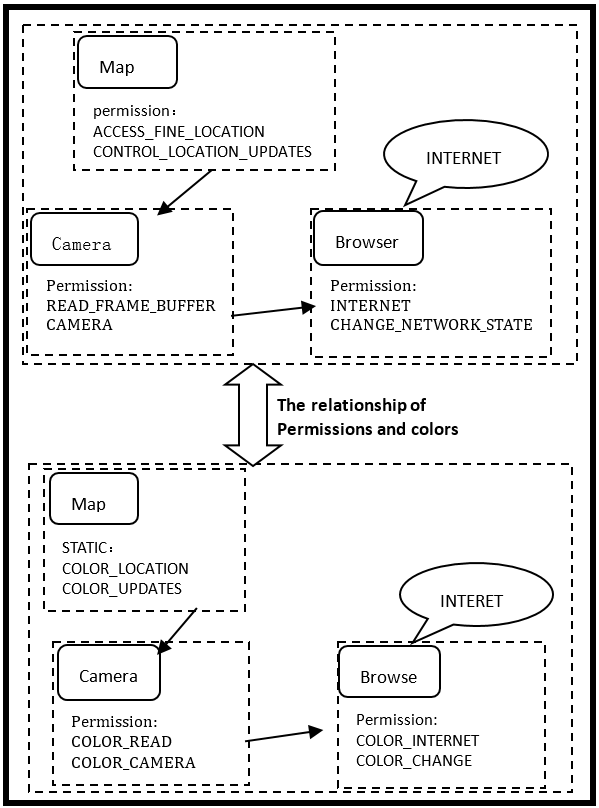


Figure 2 Example of attack

## Model verification

1) Attack examples

This paper gathers an example of multiple-permission collusion attacks from existing applications. There are three existing applications in this example and they are location application (Map), camera application (Camera) and browser application (Browser). To explain the attack process clearly, only necessary permissions are given and others are ignored. The picture below shows its permissions distribution.

In this example, risky permission combination T is involved, T={ACCESS\_FIND\_LOCATION，INTERNET}. As is shown in the figure 2, application Map possesses permission named ACCESS\_FIND\_LOCATION which has the ability to obtain the user’s accurate location; application Browser possesses permission INTERNET which has the ability to connect with internet; application Camera does not possess permission which involves in T. However, there is a path created by camera to allow Camera to obtain the user’s accurate location, at the same time, it allows application Browser call itself, so, Browser is able obtain the user’s location through Camera and send it to internet. In this condition, the user’s location is leaked and the attack is successful.

Primarily, assuming that the attack example is based on improved mandatory access control model and the path of this call is:

Map(A)->Camera(B)->Browser(C)

Assuming that B connected with A firstly, then B will obtain A’s permission collection through detection algorithm and B’s permission collection will turn into {ACCESS\_FIND\_LOCATION，CONTROL\_LOCATION\_UPDATES，READ\_FRAME\_BUFFER，CAMERA}. At the same time, its corresponding color collection Pre\_color turns into {COLOR\_LOCATION, COLOR\_UPDATES, COLOR\_BUFFER, COLOR\_CAMERA}. Above color collection does not contain any group which is contained in BlackList. So, this round of matching is failed and this request call is allowed. Up to now, B’s color collection turns into changed collection as a result of dyeing algorithm. Because the request call does not reach to the end, application C will continue calling B according to the model’s operating mechanism. C will obtain B’s color collection this time and C’s color collection will turn into {COLOR\_LOCATION，COLOR \_UPDATES，COLOR\_BUFFER，COLOR\_CAMERA，COLOR\_INTERNET，COLOR\_CHANGE}. Using the detection algorithm to detect whether there is risky permission combination in C’s color collection. Then, the risky permission combination T will be found, and T={ACCESS\_FIND\_LOCATION，INTERNET}. Its corresponding BlackList combination is {COLOR\_LOCATION ，COLOR\_INTERNET}. In this condition, detection algorithm will return “NO” and refuse the call. Up to now, there is not any problem; improved mandatory access control model checks collusion attack out this time and refused it. However, it is not difficult to find that the application’s color collection is contaminated. After last request call, if a new call which the path is B->C is occurred, the call will be refused as a result of color contamination. Application C’s permission collection contains application B’s permission as a result of the call: Map（A）->Camera（B）->Browser(C) , and C’s permission collection is {COLOR\_LOCATION，COLOR \_UPDATES，COLOR\_BUFFER，COLOR\_CAMERA，COLOR\_INTERNET，COLOR\_CHANGE}. When the call is occurred, the model will find that all the B’s permissions are included in A’s permission collection and C’s color will not have changed. However, the model will find a BlackList{ACCESS\_FIND\_LOCATION，INTERNET} in C’s permission collection. So, model will refuse the request call this time.

If we go back to the moment when all the request call hadn’t occurred and then make a call from B to C, after all the steps are executed and C’s color collection will turn into {COLOR\_BUFFER，COLOR\_CAMERA，COLOR\_INTERNET，COLOR\_CHANGE}. There is not any item of BlackList is included in this collection. So, this call will be allowed. Towards different results of a same call, we find it results from color contamination.

If the attack is executed in the model by this paper, the call path is: Map (A) -> Camera (B) -> Browser (C); assume that B calls A first, then B’ Pre\_color will obtain A’s STATICS, in another way, B’s Pre\_color is {COLOR\_LOCATION, COLOR\_UPDATES, COLOR\_BUFFER, COLOR\_CAMERA}. No threat would be detected when the detection algorithm runs, so this request would be allowed. Next, B’s DYNAMIC would be dyed (dyeing algorithm), and the cleaning B’s Pre\_color. Because the call does not reach the end, C continues to call B. In the pre-stained algorithm, C’s Pre\_color would obtain B's DYNAMIC. At this time, C's Pre\_color is {COLOR\_LOCATION, COLOR \_UPDATES, COLOR\_BUFFER, COLOR\_CAMERA, COLOR\_INTERNET, COLOR\_CHANGE}, the blacklist T = {ACCESS\_FINE\_LOCATION, INTERNET} would be detected by the detection algorithm, so this request would be refused. Then C’s Pre\_color and B’s are cleaned immediately. At this point, all the applications return back to the initial state. Whenever C calls B, C's pre\_color just obtains B’s STATICS. That is to say, C’s DYNAMIC is {COLOR\_BUFFER, COLOR\_CAMERA, COLOR\_INTERNET, COLOR\_CHANGE}, because the collection does not contain any BlackList, the request would be allowed.

## Model Features

1) The model in this paper makes some improvements toward color contamination of the original model. It handles the applications’ original colors and dynamic colors separately, avoiding the color contamination.

2) The original model’s tracking algorithm depends on the existing graph. In the improved model, the tracking algorithm is based on the calls between vertexes, that is to say, the model operates dyeing algorithm according to the vertexes’ call.

3) The model creates a database of threats to avoid repeated detection, thereby reducing the time costs.

4) The threats are divided into different levels (serious, dangerous, general) in the model. When discovering serious threat, the system would reject automatically; in other cases, the system would prompt the user to decide whether to allow the request.

# Problems in Model

1) Because the relationship between permissions and services in the existing Android is complex, the relationship has not been considered temporarily, which would reduce the accuracy. In the latter work, the model will give further consideration to the relationship between permissions and services to improve the accuracy of the model.

2) There will be more and more new combinations of risky permissions in the development process; the model could not add these combinations of risky permissions automatically. This job requires compiling and executing the model manually again, and we must release it again.

3) Due to a larger number of Android permissions, so this model failed to do a comprehensive definition of all combinations of risky permissions.

4) According to (3), this model failed to do a comprehensive definition of all combinations of risky permissions, so the partition of combinations is just an idea; the model will achieve the idea in the latter work.

# Relate work

Most of the current models analyze the applications statically. For example, reference [6] implements a permission request mechanism authorized by the user, refining the system authorization, but could not defend the privilege escalation attacks. Also there is a mechanism [7] which prevents the abuse of permissions by checking whether to apply for combinations of risky permissions when an application is installed. It has a good defense on shared UID, but it could not find the threats happening in the runtime. Reference [8] prevents confusion agent by checking the process’ call stack and reducing the effective permissions of the caller's permissions into the intersection of the caller and the callee. However, because of the diversity of inter-process communications, the model could not detect the collusion attack by using covert channels. Reference [9] proposed a mechanism: CrossDroid, which combined the fine-grained permissions mechanisms and privacy data coloring mechanism. The model deploys the applications’ permissions dynamically in a fine-grained way to block private data leaks from applications and realized the quarantine control of private data by using the colored track to mark private data.

However, these models analyze the applications statically. For dynamic attacks caused by collusion attack, confusion agents and UID sharing, these models are not good solutions.

# Conclusion

This paper raises an access control model based on the existing mechanisms and XManDroid.

The dyeing mechanism in the model tracks the historical communications to get all the permissions which belong to a running application. Then the model will match them with the blacklists to realize dynamical, real-time detection of dangerous actions of applications and remind the user to protect their information security. On this basis, the cleaning mechanism and the historical record mechanism are added to make the model more quickly and effectively. Toward the issues raised above, we will make further research.

##### Acknowledgment

##### REFERENCES

[1] [Sourabh](http://sourcedigit.com/author/sourabh/), “3 Android Data Showing “Why Android is the New King of Technology?””, http://sourcedigit.com/1913-smartphone-os-global-market-share-data-2014/, Feb 16, 2014

[2] Lynn, [J., “Android OS App Development Is Fastest”](http://www.droidreport.com/reporters/jennifer-lynn) <http://www.droidreport.com/android-os-app-development-fastest-8050>, April 3, 2014

[3] H. Peng, “Research and application based on permission’s management of Android”, Nanjing university of Science and Technology, Sept, 2012.

[4] Jiang, S., Wang, J., Yu, H., Zhang, T., Chen, R., “Improved mandatory access control for Android”, Journal of Computer Applications. Vol.33, No.6, June 01, 2013, pp. 1630-1636.

[5] Bugiel, S., Davi, L., Dmitrienko, A., Fischer, T., Sadeghi, A., ”XManDroid: a new android evolution to mitigate privilege escalation attacks” . Center for Advanced Security Research Darmstadt, Technische University Darmstadt. System Security Lab, Apr, 2011.

[6] Nauman, M., Khan, S., Zhang X., “Apex: Extending Android permission model anden forcement with user-defined runtime constraints”, Proceeding of the 5th ACM Symposium on Information, Computer and Communications Security. New York, ACM, 2010, pp. 328-332.

[7] Enck, W., Ongtang, M., McDaniel, P., “On lightweight mobile phone application certification”, ACM Conference on Computer and Communications Security. New York, ACM, Nov 09, 2009, pp. 235-245.

[8] Felt, A. P., Wang, H. J., Moshchuk, A., “Permission re-delegation: Attacks and defenses”, Proceedings of the 20th USENIX Security Symposium. Berkeley, CA, USENIX Association, Aug 08, 2011, pp. 22-37.

[9] Dai, W.，Zheng, T., “Dynamic privacy protection model based on Android permissions”, Application Research of Computers. Vol.29 No.9, Sept 2012, pp. 3478-3482.