2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)

Policy-based Access Control for Robotic Applications

Yi Zong, Yao Guo, Xiangqun Chen
Key Laboratory of High-Confidence Software Technologies (Ministry of Education),
School of Electronics Engineering and Computer Science, Peking University, Beijing, China
{zongyi, yaoguo, cherry}@pku.edu.cn

***Abstract*—With the wide application of modern robots, more concerns have been raised on security and privacy of robotic systems and applications. Although the Robot Operating System (ROS) is commonly used on different robots, there have been few work considering the security aspects of ROS. As ROS does not employ even the basic permission control mechanism, applications can access any resources without limitation, which could result in equipment damage, harm to human, as well as privacy leakage.**

In this paper we propose an access control mechanism for ROS based on an extended policy-based access control (PBAC) model. Specifically, we extend ROS to add an additional node dedicated for access control so that it can provide user identity and permission management services. The proposed mechanism also allows the administrator to revoke a permission dynamically. We implemented the proposed method in ROS and demonstrated its applicability and performance through several case studies.

***Index Terms*—Access control, policy-based access control, permissions, robots, robotic applications.**

I. Introduction

Nowadays, a variety of robots are widely used in many areas including industry, medical and home usage. For example, industrial robots are used in many mission critical scenarios, such as automobile manufacturing pipelines, or robot arms launched into space. Family robots such as NAO and Pepper can be used for doing housework or providing services in restaurants.

While the functionality of robot hardware becomes more powerful, software is also becoming more complicated and flexible. As a result, the functionality of a robot is no longer fixed after it is shipped. For example, many modern robots are equipped with an operating system (such as ROS or Linux), such that it is possible to install new applications (often developed by third-party developers) on the robots.

However, the possibility of including new or third-party applications in a system aggravates the security issues, especially for robots used in critical or sensitive environments. Specifically, it will be dangerous if accesses to certain resources are not well-managed. If all code can access sensors or operate actuators without any protection, it may cause damages to equipment, or even do harm to human users [1]. For example, a malicious application may cause robot arms change their motion trajectory, as a surgery robot may kill the patient.

Traditional computer systems solve similar issues by employing various access control mechanisms. Take the file system as a typical case, where every file has a permission attribute and users are classified into user groups, such that certain files can only be accessed by specific users or groups. In a more recent example, smartphone operating systems such as Android protect nearly all kinds of resources with permissions, such as sensors including GPS location and camera, or accesses to Internet or contact lists. Each permission is used to protect one or more resources, while applications need to declare what permissions they want to use. Users can manage (grant or deny) all the permissions for each application.

However, most robot systems are not equipped with similar access control mechanisms. Take ROS [2], the most popular robot operating system, as an example. The applications written for ROS are represented as nodes, and the communications between nodes are in the forms of *Topics* and *Services*. However, every node has access to every Topic and Service. There is no limitation when accessing all the resources. Although there is a few work on improving the communication security for ROS [3], [4], to the best of our knowledge, there are currently no comprehensive access control mechanisms designed for robot systems.

In this paper, we propose a policy-based access control mechanism for ROS, which enables permission management of robot applications developed with ROS. With the introduction of Android-like permissions, each application in ROS can be controlled as to whether they are allowed to access a specific resource or perform certain actions. The users can control the requests to resources with user-determined policies.

Our model extends the popular policy-based access control (PBAC) method, which offers great flexibility in permission management. PBAC is used to manage the accesses to different types of resources. Each application should have the permission to a resource before accessing it. A permission manager is responsible for making all the access control decisions. It can also change the permission status if needed.

Besides applying and adapting the original version of the PBAC permission model to ROS, we also extend it in our design so that it is able to support dynamic access control such as runtime permission revoking. Our extended model can support this through self-defined policies, which can be modified by the administrator during runtime, when a threat is detected and a revoking action is needed to protect the system.

We implement the proposed access control mechanism by modifying the original ROS code base. As most of the functions in ROS are provided in the form of Topics and Services, we treat each Topic and Service as a function. A permission consists of several functions that are used to perform a certain set of operations, which needs to be protected from certain applications (or nodes in ROS). We add an access management node to control all the access management features. When a node tries to access a permission, its intention will be verified

978-1-7281-1442-2/19/$31.00 ©2019 IEEE
DOI 10.1109/SOSE.2019.00062

by a policy engine with user-defined policies so that it can decide whether the access request should be approved.

In this paper, we make the following main contributions:

1) We propose a permission control model for robot applications in the context of ROS, based on the policy-based access control (PBAC) model.
2) We implement the proposed permission control mechanism by modifying the ROS code base. It allows the system administrator to control the permission dynamically and revoke a specific permission when necessary.
3) We demonstrated the applicability and efficiency of our permission control mechanism with experiments and several case studies.

II. Background and Related Work

A. Applications in ROS

ROS [2] is a middleware platform (or operating system) for programming robot applications. It was originally designed for PR2 robots built by Willow Garage Inc. Since PR2 exploits a distributed architecture that contains many nodes running separate operating systems communicating with each other by the Ethernet, ROS is designed to support the architecture in which different nodes connected through network.

A *Node* is the basic unit in ROS applications, which runs as a separate process in a robotic system. There are many kinds of nodes and among them is a *master node* serving as the root of all nodes. The master node stores information of all nodes and provides index services for other nodes.

Topic and *Service* are the default communication mechanisms provided by the ROS framework. Users can transmit structured messages in these ways. A node can publish/-subscribe to topics to share common messages and request services provided by other nodes in different ROS applications.

Since ROS provides an interface to write ROS nodes, there are many third parties who write common nodes for ROS applications. There is a community emerged and many popular open source packages can be found on the ROS website, which are available to everyone for free.

However, security mechanisms are generally missing in ROS. In the vanilla version of ROS, any node can publish/subscribe to any interested topics without proper auditing, as well as request any service in the ROS system. This can potentially cause serious harm to equipment or even humans as a robotic system may involve various critical operations.

B. Security Enhancement for ROS

There are some previous work on improving the security of ROS. For example, Dieber *et al.* provide encryption and a basic access control mechanism [3] for ROS applications. It provides node identification and communication encryption. All nodes should be aware which nodes they can communicate with in order to prevent unintended access. This prevents malicious users from connecting to the ROS application.

SROS [4] is an experimental project that deals with security in ROS. It uses the X.509 security chain to provide encryption and trust in the system. It also provides automated tools to deal with keys or other concepts needed to enhance the security of ROS applications. However, the lack of a comprehensive permission control model can make access management difficult, especially for scenarios that requires fine-grained control.

C. Access Control Mechanisms

Many existing OSs (such as Android) are equipped with some kind of access control models. For example, Mandatory Access Control (MAC) [5] has been adopted by many systems. It provides basic functions to manage permissions. It is widely used since the model is simple and easy to implement.

Role-based access control (RBAC) is another popular mechanism widely used in web applications. It assigns each user a role (eg. User, Administrator) so that different roles have different access permissions. Policy-based Access Control(PBAC) or Attribute-based Access Control [6] is another popular access control model which processes each access request with a policy engine. It decides whether a request is permitted by the attributes provided by users, resources or contexts [7]. This results in a flexible framework for managing permissions. It is also easily extensible since the policy enforcement endpoint can load user-defined policies.

III. An Access Control Model for ROS

A. Overview

Robotic systems are different from traditional computers for several aspects. For example, in modern robot applications based on ROS, there are many nodes in the system communicating with each other through the network. So designing access control for ROS is facing several challenges:

- Robot systems contain many functions that involve interacting with humans or other robots and equipment. The access control model should be versatile so that it can deal with complex running scenarios.
- Threats can outbreak anytime while the system is running. So we should support dynamic access control, where the policies can be modified at run time. The model should support the revoking of a given permission, which is important when facing malicious actions that has previously been granted certain permissions.
- Since the permission control mechanism is designed to extend the ability of ROS, it should be compatible to the existing ROS code and provide an easy way to support existing applications.

In order to meet these requirements, the key idea of our design is to introduce a customized policy-based access control model to the original ROS system. By introducing a PBAC permission control mechanism, it is possible to meet the requirements of robotic applications, as well as adapting to the ROS framework.

B. Policy-based Access Control

The typical architecture of a policy-based access control model [8] consists of several endpoints that works together. The access manager includes components such as Policy

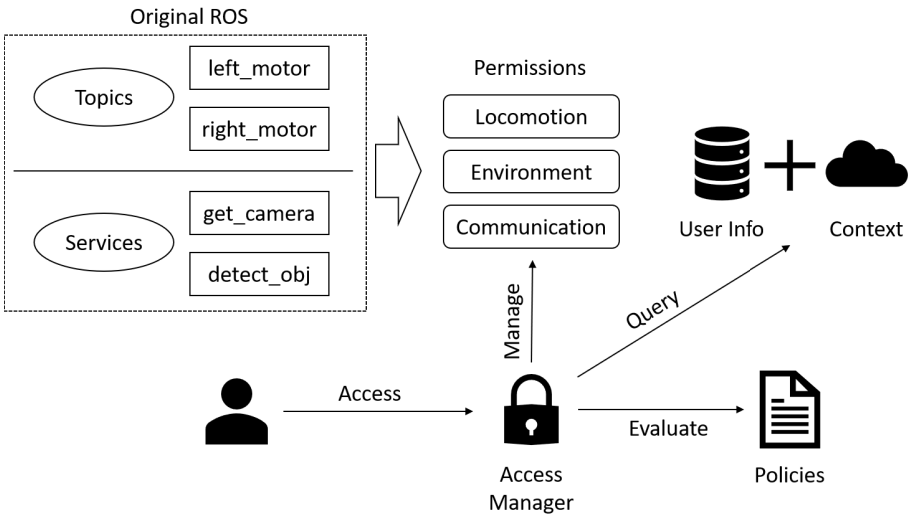

Fig. 1: Overall architecture of policy-based access control for ROS.

Enforcement Point (PEP), Policy Decision Point (PDP) and Policy Information Point (PIP).

When a robot application wants to access the locomotion permission, it should send a request to the PEP. PEP serves as a gatekeeper to all permissions. It forwards the request to the PDP, which decides if the permission is granted. PDP is loaded with policies defined by the administrator and evaluates the request. If additional information such as context data is necessary, it should consult the PIP to retrieve related information.

C. Attributes

In our PBAC model, an attribute can be any user-defined information for anyone in the system, which can be accessed by the permission control manager. Generally, there are four kinds of common attributes:

- **Subject.** It describes the user's information. For example, this user application is developed by some trusted organization and the attribute can be the signature of the code signed by that organization.
- **Resource.** This contains the information of the resources being requested. For example, if a user wants to request the maintenance information of a running system, the attributes should contain this information and deliver it to the PEP.
- **Action.** This kind of attributes describe the types of user's intention to certain kind of resources, such as getting a resource or deleting some obsolete information.
- **Context.** This contains auxiliary information related to the context such as the running environment. A typical example is the timing information. If a system wants to grant accesses only during work time or work days, this attributes will be useful.

D. Policies

Policies are statements that describe if a user access should be granted according to its information attached to the request. For example, administrators can use a domain specific language (DSL) to describe the policies for certain systems or use a commonly used scripting language to represent policies, since the result of a certain policy evaluation is usually *grant, deny or uncertain*.

In real-world scenarios, policies should have different priorities so that important policies should be evaluated first. For example, if a threat is detected and it is found that the malicious access is from a certain node, the administrator can insert a high priority policy to the permission control unit. After the policy takes effect, the threat could be eliminated.

IV. The PBAC Architecture for ROS

A. Overview

ROS supports many programming languages. The most widely adopted languages for writing ROS nodes are Python and C++. ROS itself also provides libraries and tools to build different nodes, creating topics and services. Our work extends the ROS framework by adding a permission control node. The overall architecture of our design is shown in Fig. 1.

B. PBAC Model Design

1) Permission Categories: Our method permission control method is based on the widely used ROS framework and ROS itself provides interfaces for nodes to interact with each other using the predefined concept of *Topics* and *Services*. Our permission control model treats these two methods as basic units.

As described above, a function of a robot system is provided by topics and services. Programming a robot application requires a combination of multiple functions. Since a certain set of functions are often used together to implement a kind of capability, the functions can be divided into categories.

For example, a humanoid can contain multiple types of actuators for different types of joints. Controlling the pose of the robot requires multiple functions such as controlling ankles and waists. If we want to control the access of the pose of a robot, we may categorize these functions into one permission category.

Some functions are commonly used by robot applications, such as locomotion. Many scenarios require common functions. For example, both moving forward and grabbing objects require environment information. So functions like detecting objects should belong to multiple categories.

2) Policy-based Access Control: PBAC provides strong flexibility so as to allow administrators to control all the accesses easily. For each access of certain resources, a boolean value is calculated based on certain criteria using user attributes. So the permission can be versatile to adapt to most permission control scenarios. Furthermore, attributes can contain role information, so part of role based access control (RBAC) can also be implemented using boolean logic.

We gave each user a certain set of attributes that can be customized by the system manager. When an access to a certain resource occurs, the system will check if the user is allowed to access the resource. If the access is permitted, the resource owner is notified that the user's request is permitted. Also, the system provides an interface to allow the resource to check if the access is still valid. This notification step is meant to support dynamic revoking of certain permissions.

3) Identity Tokens & Access Tokens: Given a permission model, we should use a method to grant a permission of accessing a certain function. We use identity tokens and access tokens to deal with this problem. When a node running user application (we denote this as N_{app}) tries to access a permission P_{perm}, it sends its user credential to the access manager N_{mgr}, which itself is a ROS node. The access manager compares the credential to the information stored in its database and returns a randomly generated id token, $Token_{N_{app}} =$ random() if the credential is correct. The identity token has an expiration time t_{exp}. The $(N_{app}, Token_{N_{app}}, t_{exp})$ pair is stored in the memory for further use and returned to N_{app}. The temporary identity token is used because we want to keep user credentials in secret when making requests to other nodes.

Next, N_{app} will use its identity token to request an access token from the N_{mgr}, it sends $req = (Token_{N_{app}}, P_{perm})$ to N_{mgr} in which P_{perm} is a list of permissions N_{app} wants to access. After getting the permission request, the request is evaluated by the policy engine and returns $(result, Token_{req})$ to N_{app} in which $result$ is a boolean value, with related information indicating whether the request is granted. If it is granted, $Token_{req}$ is an access token for the request, otherwise the value should be $NULL$. After getting the access token, N_{app} can request services provided by N_{serv} by adding the access token to each of the request as a parameter. N_{serv} can contact N_{mgr} to verify whether this token is valid for relevant permissions.

This design allows the revoking of both identity token and access token. When a threat is detected, after the cause is found by the system automatically or found by administrators, N_{mgr} is notified which permission or user is compromised. It can use a *Topic* to inform others about this incident and the related tokens are revoked. After this, further requests involving compromised tokens will be automatically rejected. This process is described in Fig. 2.

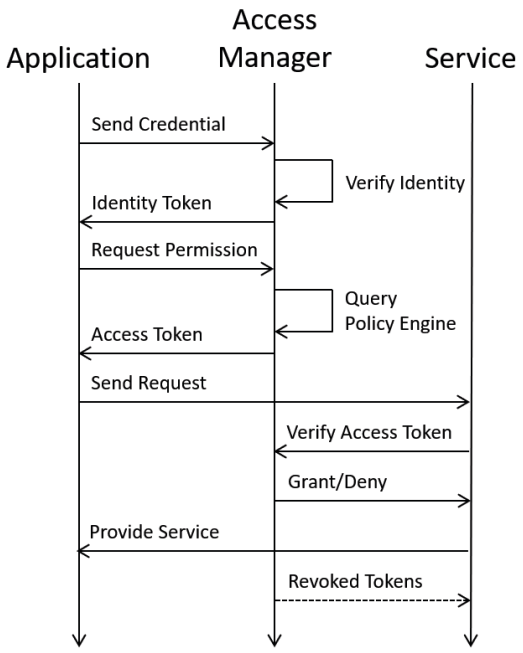

Fig. 2: The access control process.

C. Access Manager

As mentioned above, we use the *access manager* to control all the access permissions in a robot system. It is designed as a normal ROS node so that our access control mechanism can be used in every ROS compatible system without modification. Since we are using policy-based access control, we store all the user information into a database, which is accessible by the permission manager.

A permission implements a set of services and topics using the libraries provides by the ROS framework. These services and topics are used to manage all the permissions in the robot system. Here we describe some key functions of the access manager:

- **Service:RequestIdentityToken.** This service takes the user's identity information as input. The main job is to authenticate the user's identity. If it is valid, it generates an identity token to represent the user's identity. Since all the permission-related information is stored in the access manager, it can act like a central user information manager, so that a resource owner does not need the credentials of the user to authenticate the user's identity.
- **Service:RequestAccessToken.** This service is to grant the user's access request according to policies. The input is the user's identity token and the permissions the user want to access. If the access request is granted, this

service will return an access token to the user. The user can access resources of the resource owner.

- **Service:VerifyToken.** When a user accesses a function, it provides the request parameters along with the access token. The resource owner can send the access token to the permission manager. If the permission manager have the record, it will return to the resource owner the permission categories which have been approved, such that the resource owner can provide service to the user.
- **Topic:RevokedTokens.** When the robot system is running, some incidents may threaten the normal running of the system. Administrators knows all the permissions used in the system and all the access tokens linked to them. Thus they can dynamically revoke access tokens affected by the threats. The **RevokedTokens** topic is used to broadcast the invalidated access tokens. When the service provider receives them, it will prevent further accesses to the requested services immediately.

D. Policy Engine

1) Policy Representation: Policies are used by the permission manager to decide whether to allow an access request. They are functions written and managed by system administrators. In this work, we use the LUA [9] scripting language to represent them. A policy is represented as a function that takes attributes as input and returns a value indicating the access is granted, rejected or undecided.

The policy engine exposes the user's attributes and permission categories to the LUA scripting environment. In our implementation, administrators can use *UserInfo* and *Permissions* to access user attributes and permission category information in LUA scripts.

All the policy files can be loaded both during start-up or runtime. When a request is received, the policy engine will examine each policy according to a preset priority order to see if a policy would grant or deny the access. If none of the policies have granted the access request, the policy engine will reject the access request by default.

2) User Identity: Identifying a user is fundamental to access control. An identity should be kept secret, both at start-up and during run time. We employ a centralized user identity authentication method. The access control node assigns a user credential to each node. If a node wants to access certain permission controlled by the manager, it should provide its credential first and get an identity token. Then it will use this token to access other nodes as it can be verified by other nodes.

3) Permission Revoking: When there occurs a potential risk which may cause fatal error to the ROS system, it should be dealt with immediately and do not interfere other parts of the system. After the administrator detects the incident, the policy can be modified so that the node who provides services can know that the access token is revoked and stop serving the affected node. At the same time, the system can revoke the user's credential, so that the user can not get its identity token and access other permissions any more.

TABLE I: Permissions defined for Pioneer 3DX.

Service	Permission
left_wheel	Locomotion
right_wheel	Locomotion
Camera	ImageCapture
InfraRed	Communication
Bluetooth	Communication

V. Experiments

A. Implementation

We implemented the proposed access control mechanism based on the ROS framework. We used Python to implement the access control node that provides user identity service and access control service. The policies are defined by the administrator, which can be written in either LUA or Python. All functions are implemented by services and topics in the access control node. Every node can connect to the service providing user identity authentication or permission related operations, but only the system administrator can modify the access policies.

B. Experimental Setup

We use a popular robot system to carry out our experiments: the Pioneer 3-DX[10]. It is a general-purpose mobile platform with many sensors and an on-board computer. It is widely used in many indoor or outdoor scenarios and is supported by ROS in both simulators (Fig. 3) and real-world scenarios along with a number of packages specifically designed for this type of robots. In our experiments, the robot will perform a patrolling task in a room, getting images of different places. This will require the ability of locomotion and sensing in order to take photos and avoid obstacles. We assume that only the users whose role is the *operator* can perform this task. Table I describes a list of functions used in this test case along with the permission categories these functions belong to. We will test three scenarios to demonstrate our access control mechanism:

- A normal user without the *operator* role requests permissions to perform this task and it should be rejected by the access control manager.
- A user designated as an *operator* requests permission to perform this task and it is granted by the access manager.
- Suppose one of the sensors is malfunctioning, so that the robot collides with obstacles. This situation will be reported to the system administrator. At the same time, the system goes into the fail-safe mode and the permissions given to the *operator* group will be revoked.

This involves two policies. One is allowing only the *operators* to perform the patrolling task, and the other is when the system is set to fail-safe mode, only administrators can access these functions. We omit the detailed policies here as they are straightforward. Since all the nodes are connected in the local network with low latency (less than one millisecond), the expiration time of an identity token is set to 1 second. All the requests should be handled within that time period.

Fig. 3: The Pioneer 3DX robot used in our experiment.

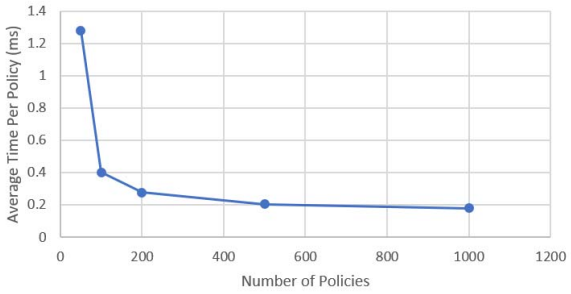

Fig. 4: The average time overhead of access control with respect to the number of policies.

C. Results

1) Access Control: For the scenario mentioned above, the system was able to work properly as expected:

- When a non-operator user tried to perform the task, it was denied access.
- When an operator user tried to perform the same task, it was allowed and executed normally.
- Finally, when the permission was revoked from the operator group, all users except the administrator could not perform the given task.

2) Performance Overhead: Compared to the original ROS framework, the main overhead of our work is mainly due to the negotiation process. Since our work focuses on access control, which does not involve data encryption (although encryption can be combined with our work), it does incur much computational overhead. It requires several additional requests over the network through the ROS framework and some extra computational time when evaluating the policies. For the scenario above, we compare the time cost of the responses with and without our access control mechanism, and the average time overhead is about *3ms*, which includes the time spent in getting the identity token, requesting and verifying the access token.

Note that in our experiments, we made 1000 requests and calculated the average time overhead. After the connection is established, it requires no additional overhead over the normal ROS communication. In order to get rid of the influence of different policies, we assume that every request would be granted immediately.

Because the policy retrieving and evaluation overhead is related to the number of policies, we then test the influence on time overhead with different numbers of policies. Fig. 4 shows the average time cost of each policy when the number of policies increase. We can see that the average time cost per policy is decreasing when the number of policies increases. When the number is large enough, nearly all computation is due to policy evaluation, while the average time is reduced to *0.2ms* per policy.

The experiment result shows that the performance overhead incurred by access control is very small and almost negligible, which is adequate for most real-time tasks.

VI. Conclusion

This paper has proposed a policy-based access control (PBAC) mechanism for ROS, such that we are able to control the resource accesses in robotic application written for ROS. With an extended PBAC model, we can support not only common access control functionalities, but also dynamic access control with run time revoking. We have demonstrated the applicability and efficiency of the proposed method with experiments and case studies. Our future work include expanding the coverage of the access control method to cover more functionality and different types of robots.

Acknowledgement

This work was partly supported by the National Key Research and Development Program (2017YFB1001904) and the National Natural Science Foundation of China (61772042).

References

[1] A. M. Zanchettin, N. M. Ceriani, P. Rocco, H. Ding, and B. Matthias, "Safety in human-robot collaborative manufacturing environments: Metrics and control," *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 2, pp. 882–893, 2016.

[2] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, Japan, 2009.

[3] B. Dieber, S. Kacianka, S. Rass, and P. Schartner, "Application-level security for ros-based applications," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pp. 4477–4482, IEEE, 2016.

[4] R. White, D. Christensen, I. Henrik, D. Quigley, *et al.*, "Sros: Securing ros over the wire, in the graph, and through the kernel," *arXiv preprint arXiv:1611.07060*, 2016.

[5] S. Osborn, "Mandatory access control and role-based access control revisited," in *Proceedings of the second ACM workshop on Role-based access control*, pp. 31–40, ACM, 1997.

[6] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM conference on Computer and communications security*, pp. 89–98, Acm, 2006.

[7] G. Bai, L. Gu, T. Feng, Y. Guo, and X. Chen, "Context-aware usage control for android," in *Security and Privacy in Communication Networks - 6th Iternational ICST Conference, SecureComm 2010, Singapore, September 7-9, 2010. Proceedings*, pp. 326–343, 2010.

[8] X. Jin, R. Krishnan, and R. Sandhu, "A unified attribute-based access control model covering DAC, MAC and RBAC," in *IFIP Annual Conference on Data and Applications Security and Privacy*, pp. 41–55, Springer, 2012.

[9] R. Ierusalimschy, L. H. De Figueiredo, and W. C. Filho, "Luaan extensible extension language," *Software: Practice and Experience*, vol. 26, no. 6, pp. 635–652, 1996.

[10] Y. Mei, Y.-H. Lu, Y. C. Hu, and C. G. Lee, "A case study of mobile robot's energy consumption and conservation techniques," in *Advanced Robotics, 2005. ICAR'05. Proceedings., 12th International Conference on*, pp. 492–497, IEEE, 2005.