Received: 31 October, 2022, Accepted: 29 December, 2022, Online: 28 January, 2023
DOI: https://dx.doi.org/10.55708/js0201002

CAPEF: Context-Aware Policy Enforcement Framework for Android Applications

Saad Inshi [1], **Mahdi Elarbi** [1], **Rasel Chowdhury** [1,*], **Hakima Ould-Slimane** [2], **Chamseddine Talhi** [1]

[1] Department of Software Engineering and Information Technology, École de technologie supérieure, Montréal, Canada
[2] Département de Mathématiques et d'Informatique, Université du Québec à Trois-Rivières, Trois-Rivières, Canada

*Corresponding author: rasel.chowdhury.1@ens.etsmtl.ca

ABSTRACT: The notion of Context-Awareness of mobile applications is drawing more attention, where many applications need to adapt to physical environments of users and devices, such as location, time, connectivity, resources, etc. While these adaptive features can facilitate better communication and help users to access their information anywhere at any time, this however bring risks caused by the potential loss, misuse, or leak of users' confidential information. Therefore, a flexible policy-based access control system is needed to monitor critical functions executed by Android applications, especially, those requiring access to user's sensitive and crucial information. This paper introduces CAPEF, which is a policy specification framework that enforces context-aware inter-app security policies to mitigate privacy leakage across different Android applications. It also, provides an instrumentation framework to effectively enforce different behaviors based on automated context-aware policies to each Android application individually without modifying the underlying platform. Accordingly, the modified applications will be forced to communicate with our centralized policy engine to avoid any malware collusion that occur without the users' awareness. Experiments conducted on CAPEF shows an effective performance on the size of the enforced application after the instrumentation. The average size added was 705 bytes, which is about 0.063% of the size of the original applications, which is significantly small compared to other existing enforcement approaches. Also, we have denoted that the size and the execution time of the policy increases whenever the policies become more complex.

KEYWORDS Security, Android applications, Application instrumentation, Context-aware policies, Policy enforcement, Privacy

1. Introduction

Context awarenes service is a key driver for the modern mobile operating systems which are commonly prompting users by showing authorization dialog boxes asking for allowing or denying access to some functionalities. These services opened a big interest in defining, managing, and enforcing context-aware policies especially for those scenarios that put users under the risk of leaking or misusing their credential information. Yet, thousands of malicious applications are developed on the Android store and affecting millions of Android users worldwide. To safeguarded Android users, Google is frequently announcing the cracking down of such malicious applications. For instance, Google has removed over 700,000 malicious applications from the Play Store in 2017 only [1]. Based on Goggle statistics, this is 70% more than what Google removed in 2016. Very recently in 2022, Google has removed 16 bad apps that missuses mobile data and draining batteries. Surprisingly, these apps have been downloaded by more than 20 million users around the world [2].

Android system protects sensitive APIs by granting them permissions to amplify application privileges on the device, including access to stored data and services, such as network, memory, and so on. All permissions required to access the protected APIs in each application's manifest file (AndroidManifest.xml) [3] are necessarily set by the Android app developers. System permissions are divided into two categories, normal and dangerous. Normal permissions do not pose a direct threat to the privacy of the user, although dangerous permissions may allow the application to access the user's confidential data. Existing application authorization system in Android allows you to control only the permissions that are classified as dangerous, whereas our developed policy approach, offers the ability to control all monitored permissions as any application may cause a risk or conflict within a specific context without user awareness. Also, our model will mitigate malware collusion in which two or more malicious apps combine to accomplish their goals. For example, a user can choose to allow a camera app to access the camera, but not to the contact information without his consent or awareness. Another example, where normal applications can be granted permissions to collecting user's contacts, photos, videos, locations, or banking information then sending it over the internet to a remote server and taking into consideration pre-defined context aware access control policies. Therefore, a flexible policy-based access control system is needed to monitor APIs functions in Android applications, especially those requiring access to the user's sensitive and crucial information. The current permission system of Android still has some limitations, where users must grant most permissions requested by an application to install it, without being able to automatically manage most of these permissions based on the user's context afterwards.

This drawback has motivated the researchers to propose context-aware policies and/or define policy languages to enforce the current Android permission system either by modifying the Android platform such as in [4]–[8] or by instrumenting the Android Applications [9]–[18] and more recently in [19]–[21] (more details and comparisons can be seen in the background and literature review section). While, existing works have demonstrated significant effectiveness in protecting users against threats, these approaches are still impeded by several drawbacks.

1.1. Challenges

Defining and monitoring context-aware inter-app policies of sensitive APIs on Android applications presents several challenges. Especially, when we are trying to defend applications collaborating to create malicious contexts:

i Context-aware inter-app policies are difficult to predict as they frequently get changed and need to be updated accordingly for accuracy and correctness.

ii Beside the difficulty of representing the security policies in a logical language which can contain user contexts and semantics, a key challenge is how to design and develop effective and efficient algorithms to monitor private information leakage on semantics levels.

iii There is a need for a policy language that can provide certain agreements that empower users with the ability to prioritize specific mobile resource and specify the amount and kind of information that can be shared within particular contexts. For instance, a user should be able to share a personal data with a specific service provider based on his location or at a specific time of the day to ensure his privacy. In this case, the user must agree on a trade-off between data privacy and the needed service. As a result, a policy should be defined to ensure privacy, while certain context-based information can be shared.

iv Android Sandboxing is introduced in the recent Android version 13.0. Sandboxing protects apps data and permission from getting access from other apps. This new feature will have an impact on our inter-apps policy model, but our main goal still effective which is to allow the user to define his policies to work automatically depending on the context update, to the running apps etc. Therefore our, framework can fully protect the Android OS versions lower than 13.0 in which the installed apps can communicate between each others.

v Due to the resourced constrained mobile devices, we have to decide, in early stages the instrumentation and monitoring location, whether to be on device, external PC or App market.

1.2. Contributions

This article is contributing solutions for the above-mentioned challenges and limitations by introducing:

i A formal context-aware policy specification framework for Android applications that effectively describe users defined consents.

ii A design and implementation of an instrumentation framework to mitigate privacy leakage across different Android applications.

iii Providing a centralized applications controller. This will allow users to manage all API calls performed by the applications installed on the device and to mitigate malware activities.

iv Effectively enforce different behaviors based on automated context-aware policies for each Android application individually without any modification to be entailed in the underlying platform.

v Experiments conducted on our CAPEF in terms of performance by analyzing the size of the enforced application after the instrumentation, also, the execution time of the policy decision, and the policy size which affects the complexity of the applied rules and conditions.

2. Background and Literature review

Android applications are distributed as APK files (Android Package). Each package consists of the application's manifest file, resources and application bytecode encoded for the Dalvik Virtual Machine (DVM) as a single classes.dex file. The APK file has to be signed for verifying its authenticity. Android signed package (Dex files) runs separately in its own DVM. Also, Android system is an open source platform where applications are published in different markets without being monitored or analyzed to guarantee their behavior. For that reason, Android platform protection mechanisms such as Application Sandboxing, Permission Model and Application Signing are developed for privacy and security purposes. Accordingly, at the time of installing Android applications, each application will get a unique user identifier (UID) [22]. Also, no application will be able to access other application's files. Besides, every application run into separate VMs. Accordingly, no vulnerable application will affect other applications.

For Android access control policies, context awareness have become an essential accessory in most mobile platforms and applications. This necessity has motivated many researchers to provide policy enforcement mechanisms to define, manage and enforce different context aware policies. In this context, traditional access control models which generally refer to the process of determining what actions are allowed by a given subject upon objects and resources should be reinforced to fulfill the modern context-aware applications.

The most popular access control models are Discretionary Access Control (DAC), Mandatory Access Control (MAC), Role Based Access Control (RBAC) and Attribute Based Access Control (ABAC) [23] . For instance, RBAC is a model that uses "roles" to determine access control, also permissions are associated with these roles, and users are made members of appropriate roles. In ABAC, requests are granted or denied based on subject and resource attributes, environment conditions, and a set of policies specified in terms of those attributes and conditions. When it comes to using ABAC models, one of the well-known standard system implementations is XACML [24]. The XACML standard defines a declarative access control policy language implemented in XML and provides a processing model on how to evaluate access requests.

Thus, to adopt an effective context-aware access control model on Android platform and its application, there are series of work studying and proposing security mechanisms for privacy and security requirements. In this context, many reviewed efforts in [4]–[8] have been developed to extend the Android security framework in order to improve the standard permission control provided by the operating system. For example, SecureDroid [4] addressed the issue of controlling security policies while applications are executing in the Android environment. During the installation of an application, Android allows the user to grant permission for an application to use certain features of the system. Therefore, SecureDroid introduced an extension of Android's security framework in order to improve the standard permission control provided by the operating system. To achieve this goal, they introduced a new control mechanism adding granularity and flexibility. Moreover, their policy framework is based on customizing the XACML standard to work on the Android system. Also, they have provided the ability to add or edit a policy through a dedicated system service. This will allow users to specify which permissions to grant and which others to deny for each of the defined contexts.

However, modified Android platform has a number of major drawbacks such as the need of building different versions of firmware and platform codes, where applications will be limited by the security policies supported by the modified Android platform. Therefore, many researchers in [9]–[18] and more recently in [19]–[21] have provided solutions that are based on instrumenting Android Applications in ordered to enforce some security policies. These solutions require no modification to the Android platform and can be easily deployed. For instance, Aurasium [9] is a concurrent approach that rewrites Android application to sandbox important native API methods and monitors the behavior of the application to detect any security violations. Also, Capper [17] is a prototype for context-aware policy enforcement to mitigate privacy leakage in Android applications. This mechanism will enforce privacy policy based on user preferences. By using this system, when a user tries to install any Android application the bytecode rewriting engine called BRIFT will rewrite the program of this application by selectively inserts instrumentation code along taint propagation slices for monitoring and preventing any information leakage. Another interesting research called Weave Droid [18] has provided a framework for weaving AspectJ aspects into an Android application. The framework takes two inputs at the beginning: APK and a set of aspects that will be weaved into the APK. The weaving process will be performed on the Android device. Also, very recently in [21] They have developed a lightweight monitoring system to detect malware activities with the log file and they evaluated the proposed model according to Policy-based permissions.

Accordingly, some of the reviewed frameworks have provided enforcement mechanisms to mitigate Malware activities by enforcing context-related policies, however they didn't afford a policy specification language that runs on Android system as an application or a service without modifying the Android platform. Thus, this article is introducing a more featured policy specification language that allow regular users and any company to easily interpret and enforce their complex context-aware inter-apps policies on their Android mobile applications.

2.1. Summary of the literature reviews

The table 1 shows the summary and differences between our research and the other works.

3. System Overview

This section gives an overview of our approach that automatically enforces user specific context-aware policies for android application and monitors all API calls that occur due to the interaction between enforced applications. Our developed system works in the application level of the Android framework, and its main components are illustrated in Fig. 1.

i From left, the first components represented the instrumentation of the targeted Android application (byte code or source code) by injecting monitoring code before each selected API method's call to intercept it at run time.

ii After the instrumentation, the applications will be forced to communicate with our controller that monitors the targeted context-aware inter-app calls.

iii Then, users will use CAPEF interface to create context-based rules and conditions in the form of security and privacy policies. More precisely, the policy represents a rule or set of rules based on a set of conditions and save it in the policies Database.

To motivate and illustrate our approach, we present the following scenario for vulnerability pattern that consider context-awareness policies. In this scenario, a user is using a public WiFi network in a coffee shop and he is trying to consult his credential banking information through his banking application. As the public network is not secure

Table 1: Comparison of related work

Approach/ System	Methodology	Required modification	Policy Language	Context-aware inter-app Privacy Leakage Prevention
TaintDroid [25]	Dynamic Analysis	Android Platform	✗	✗
Appink [26]	Watermarking	Application	✗	✗
Apex [5]	Policy Enforcement	Application	✗	✗
TISSA [6]	Resources Access Control	Android Platform	✗	✗
AppFince [7]	Dynamic Analysis & Resources Access Control	Android Platform	✗	✗
Aurasium [9]	Rewriting Java Bytecode	Application	✗	✗
I-arm-droid [11]	Rewriting Dalvik bytecode	Application	✗	✗
AFrame [14]	Isolating Advertisements	Application	✗	✗
Capper [17]	Rewriting Java Bytecode	Application	✗	✗
SecureDroid [4]	Policy Enforcement	Android Platform	✓	✗
Weave Droid [18]	Isolating Advertisements	Application	✗	✗
[21]	Policy-based permissions	Application	✗	✗
CAPEF	Policy Language	Application	✓	✓

and there are other people who use the same network, so there is a risk of sending requests to attackers and thefts of private data. Android system checks only if the user has been previously granted the permission of accessing WiFi network and doesn't provide any context-awareness policies to mitigate such dangers scenarios. CAPEF can provide more effective access control not only on the permissions declared in advance by the user but also at run time based on the context of the user, device, and resources. Thus, as a solution to the above-mentioned scenario, a user can use CAPEF to define a policy that prevent the use of banking applications while connecting to a public WiFi network. When an enforced bank application attempt to get public WiFi access, our application controller will first check if the permission is declared in the application manifest file. Then, will check if the user has defined any policies related to this permission in the policies database. Subsequently, the controller access decision will be based on the predefined policies for that specific API. As a result, the controller will notify the user for not being allowed to connect to public WiFi for banking activities.

4. CAPEF

Context-Aware policies are not static and might be changing over time to fulfill users' needs. Therefore, these policies could be used to control the behavior of the applications during run-time, which in our case, means monitoring and controlling all sensitive activities across different applications according to user's context. To achieve this goal, we provided a native Java-based CAPEF that allow regular users and enterprises to interpret and enforce their complex context-Aware policies. Contexts will represent various parameters including time, location, identity, activity, application, device status, resources etc. Moreover, these policies can be exported in multiple formats such as XML and JSON as they are widespread use today for data interchange and structured stores.

To develop the CAPEF language that allows the user to define contextual policies and transform them into security controls, we must rely on a flexible design that varies with the complexity of the policies, rational, able to execute all the conditions and easy to add new contexts.

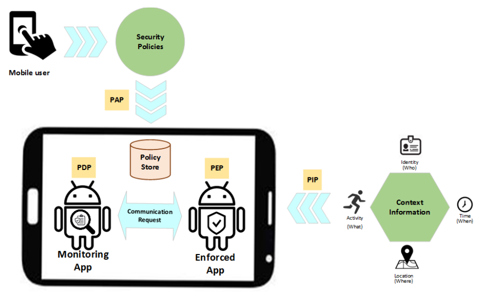


Figure 2: CAPEF Architecture

4.1. CAPEF Architecture

The main components of CAPEF are shown in Fig.2. which represents the following:

i **Security policies**: which is an interface for the user to define context aware policies and save them at the policy store. This component will act as a policy administration point (PAP) which is the source of the policies.

ii **Enforced Application**: Which plays the role of policy enforcement point (PEP) that receives the access request and move it to the Monitoring application for making access decision based on the predefined context aware policies.

iii **Context Information**: Provides context information in a form of attribute values about the targeted applications, resources, activates, actions and so on. This component will play the role of policy information point (PIP) in our system.

iv **Monitoring Application**: This component plays the role of policy decision point (PDP). It takes the access request from the PEP then interacts with PAP and PIP that capture the required context information to identify the appropriate policy. Then evaluates the

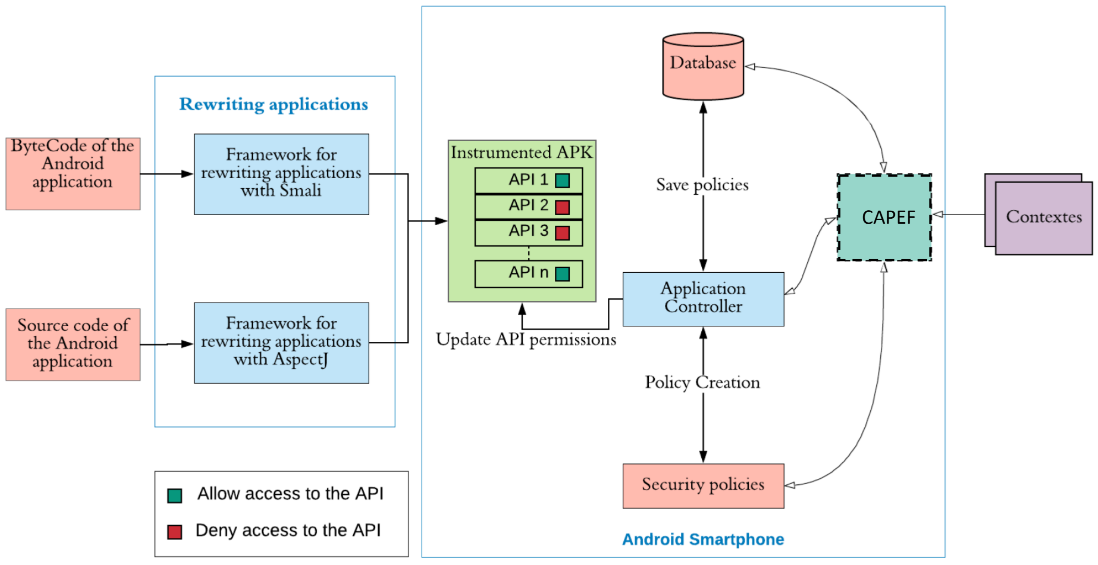


Figure 1: System Overview

request according to the applicable policy and returns the decision to the PEP.

4.2. Formal Definition

Hereafter, we formally define our ABAC policy model, which is composed of three main entities:

1. A, P, and C : sets of application, permissions (resources) and contexts, respectively;
2. AA , PA ,and CA are the pre-defined sets of attributes for applications, permissions, and contexts, respectively.

I An application *app* is a represented by a tuple as follows:

app <name, visibility, class, APIs>, where:

i visibility ∈ {background, foreground}

ii class ∈ { banking, communication, recording, games, media, location}

iii APIs: a set of APIs that can be invoked during execution

II A permissions *per* embeds the access to the resource, it is a represented by a tuple as follows:

per < name, resource, securityLevel >,

i resource ∈ { personal data, calendar, camera, wifi, account, calls, sms,Audio, GPS},

ii securityLevel ∈ { Normal, Dangerous },

III A context *c* is a represented by a tuple as follows:

c < time, location, fgApp, BgApps availableCPU, availableMEM, availableNRG >,

i fgApp indicates which application is running in foreground;

ii BgApps is the set of the applications running in background.

3 Attr(*app*),Attr(*per*), and Attr(*c*) are attribute assignment relations for application app, permission per and context c, we have respectively

I Attr(*app*) ⊆ name × visiblity × class × APIs;

II Attr(*per*) ⊆ name × resource × securityLevel;

III Attr(*c*) ⊆ time × location × fgApp × BGaps × availableCPU × availableMEM × availableNRG.

For the value assignment of each attribute, we use the following notation: **entity.attribute= value** ,

For example, for an application **app**, a permission **per** and a context **c**, we have the following assignments:

app.visiblity = ’background’, per.securityLevel = ’Dangerous’, c.location = ’Montreal’.

4 The ABAC policy rule that decides whether or not an application **app** is allowed is allowed to get the permission **per** under a particular context **c**, is denoted as a predicate **PR** over the attributes of **app**, **per** and **c** as follows:

Rule : Allow(app, per, c) ← PR(Attr(app), Attr(per), Attr(c))

Given all the attribute assignments of *app*, *per*, and *c*, if the predicate’s evaluation is true, then the application **app** is allowed to get the permission **per** under the context **c**; otherwise, the permission is denied. Using the formal definition, we can have different types of policies for the *app*, for example:

1. A rule that dictates that " When a banking applications is being used, so the TakeScreenshot actions should be prevented from running" can be written as:

$\text{Allow}_{screenShot}(\text{app, per, c}) \leftarrow (\text{ TakeScreenshot} \in \text{app.APIs}) \wedge (\text{per.resource==screen}) \wedge (\text{c.fgApp.class != banking})$

2. A rule that dictates: "RecordVoice and RecordCall applications should be prevented from running when the user is dialing Skype from 9:00 to 10:00" can be written as:

   i $Allow_{recordVoice}$(app, per, c) ← (RecordVoice ∈ app.APIs) ∧ (per.resource==voice) ∧ ((c.fgApp != 'skype') ∨ (c.time <9:00 am ∨ c.time > 10:00am))

   ii $Allow_{recordCall}$(app, per, c) ←(RecordCall ∈ app.APIs) ∧ (per.resource==call) ∧ ((c.fgApp != 'skype') ∨ (c.time < 9:00am ∨ c.time > 10:00am))

   Or simply by combining the two rules as following:

   $Allow_{record}$(app, per, c) ←(RecordVoice , RecordCall ∩ app.APIs!= ϕ) ∧ (per==voice ∨ per==call) ∧ ((c.fgApp != 'skype') ∨ (c.time < 9:00am ∨ c.time > 10:00am))

4.3. CAPEF Policy Specification

CAPEF language is based on the definition of a policy that consists of a user ID, a policy name, a policy execution state, a control rule, and a list of applications to control. Each of these applications is characterized by a name, package name, execution status and a list of permissions. The permissions consist of a name and a execution state. The security rule consists of a list of objects that can be Parentheses, Conditions, Logical Operators, Conditional Operators, CPU, Time, Resource Used, Location and Battery.

As shown in Fig.3, user-defined security policies and its rules can contain multiple conditions, different contexts, multiple logical operators, and parentheses to specify priority between conditions. Also, these policies can be executed simultaneously in different inter-app activities across different applications. In this case, policy decision becomes more complex. Therefore, to facilitate and accelerate the execution of any compound policy, our algorithm will receive the current contexts and the control rule as parameters then returns the policy decision of the controller.

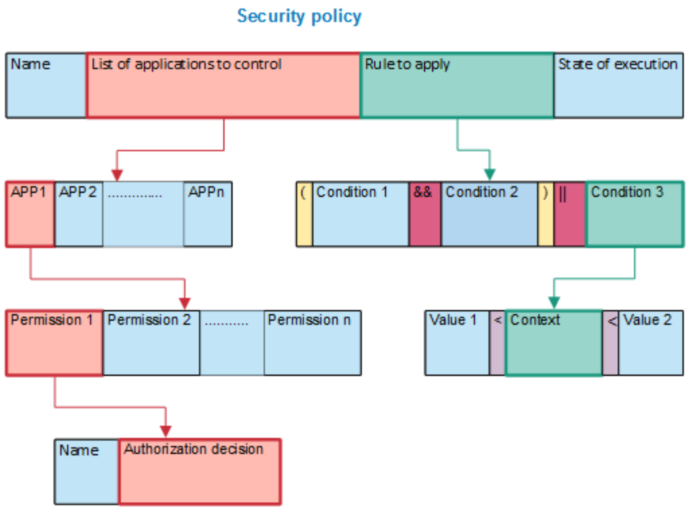


Figure 3: CAPEF Policy Execution Structure

In addition, the security rule might have sub-nodes of other rules, and themselves are sub-rules of the main rule. Therefore, we have adopted a recursive technique to reduce the complexity of the composed security rules. In this case, the algorithm will call itself, and recursion stops condition must be checked, otherwise the program will be stuck in an infinite loop.

To show the usefulness of our solution, we have chosen two examples of dangerous scenarios and we will translate them into security policies.

I Critical scenario 1: If the user uses his banking application to check his data and deposit a check in his account, while a TakeScreenshot application is installed on his smartphone.This application that takes screenshots automatic present a risk on banking data that is personal life data.

   i Solution: You must prevent the TakeScreenshot application and any screenshot function from running when the user is using their banking application.

   ii Security Policy: If [BankApp] is running, then stop the [TakeScreenShot] application.

II Critical Scenario 2: If the user is in a private work meeting every Monday from 9:00 am to 10:00 am by Skype while many other apps are able to record his speech and share it in public as RecordVoice and RecordCall applications.

   i Solution: RecordVoice and RecordCall applications should be prevented from running when the user is dialing Skype from 9:00 to 10:00.

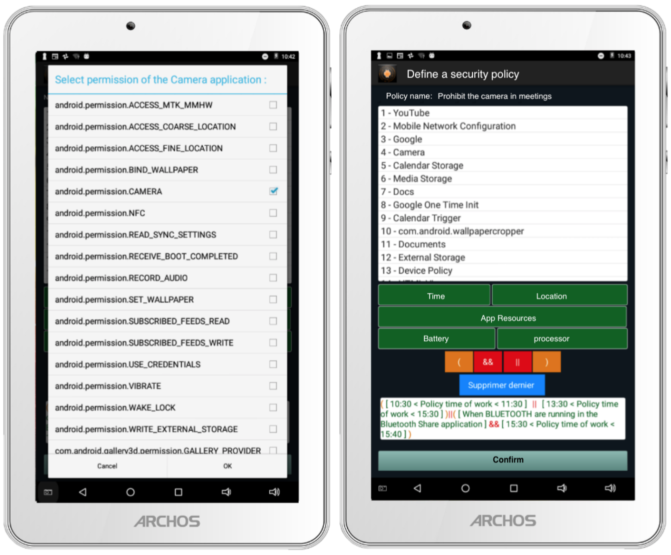
   ii Security Policy: If ((CALL_PHONE in [Skype]) && (9: 00 <= Current_Time <= 10: 00)), then stop or prevent (if not yet executing) the application [RecordVoice && RecordCall].

To apply and evaluate the defined context aware policies, an application controller has been developed to allow users to define policies depending on different types of contexts and conditions.

5. Centralized Application Controller

The developed controller provides a user interface to translate the dangerous scenarios into security policies using the CAPEF. Based on the defined policies, the controller will make the adequate access control decision to allow or block applications from using certain permissions. Moreover, provide centralized control of installed applications which capture mandatory decisions that are automatically dependent on the current context.

Fig. 4, shows an example of how to define a security policy with our controller. The control scenario is to block the execution of the Camera resource in the Camera application if the user is in a meeting from 10:30 to 11:30 or from 13:30 to 15:30 otherwise it is in a meeting from 15:30 to 15:40 and that Bluetooth is enabled in the BluetoothShare Application.


Select permission of the Camera application :
android.permission.ACCESS_MTK_MMHW ☐
android.permission.ACCESS_COARSE_LOCATION ☐
android.permission.ACCESS_FINE_LOCATION ☐
android.permission.BIND_WALLPAPER ☐
android.permission.CAMERA ✓
android.permission.NFC ☐
android.permission.READ_SYNC_SETTINGS ☐
android.permission.RECEIVE_BOOT_COMPLETED ☐
android.permission.RECORD_AUDIO ☐
android.permission.SET_WALLPAPER ☐
android.permission.SUBSCRIBED_FEEDS_READ ☐
android.permission.SUBSCRIBED_FEEDS_WRITE ☐
android.permission.USE_CREDENTIALS ☐
android.permission.VIBRATE ☐
android.permission.WAKE_LOCK ☐
android.permission.WRITE_EXTERNAL_STORAGE ☐
com.android.gallery3d.permission.GALLERY_PROVIDER ☐

Define a security policy
Policy name: Prohibit the camera in meetings
1 - YouTube
2 - Mobile Network Configuration
3 - Google
4 - Camera
5 - Calendar Storage
6 - Media Storage
7 - Docs
8 - Google One Time Init
9 - Calendar Trigger
10 - com.android.wallpapercropper
11 - Documents
12 - External Storage
13 - Device Policy

([10:30 < Policy time of work < 11:30] || [13:30 < Policy time of work < 15:30])||([When BLUETOOTH are running in the Bluetooth Share application] && [15:30 < Policy time of work < 15:40])

Figure 4: Screenshots of a policy definition within our controller

5.1. *Managing Security Policies*

The Application controller interface has been developed in a way that the user will be able to define any security policies in a few simple clicks. We chose our solution to be ergonomic, personalized, and user-centric design to have a convenient and easy-to-use service. It has also been taken into consideration that our user interface must reduce the search effort and limit data entry. In addition, all policies created by the user have been saved in a database. With the database, the user will be able to import, create, view, and modify security policies.

5.2. *Export/Import Security Policies*

Our developed solution allows the user to extract his defined policies and share them with other users of the controller or send them to nay server or cloud database. Therefore, our CAPEF flows same related policy language's architecture and structure. As discussed in the literature XACML is one of the good examples to extract our policies to its format. While Android system does not compile XACML language and all reviewed languages, our policies will be translated into java language to execute them, then will be extracted on different languages such as XML, JSON etc. Therefore, in order for our language to be compatible with other languages, we kept the same generic policy structure, objects and attributes applied by other languages as as shown in Table 2. Similar translation procedure will be applied when importing policies from other languages.

Among the values that can be assigned to attributes such as the names of applications, permissions, etc., we have defined symbols that allow us to simplify the rules, for example:

i ANY: it means no, for example a rental context, we do not need permission in this condition.

ii ALL: it means that we want to control all the permissions or all the applications it depends on the attributes used.

iii APPS: it means that we want to force the shutdown of an application.

iv API: that means we're going to apply the control on an API permission.

```
<?xml version="1.0" encoding="UTF-8"?>
<policy combine="deny-overrides" id="1" AUTHOR-KEY-CN="Mahdi" AUTHOR-KEY-FINGERPRINT="Mahdi">
   <target>
      <subject>
         <subject-match attr="id_ScreenShot" match="com.apps.TakeScreenShot" />
         <subject-match attr="id_BNC" match="com.apps.BNCbanque" />
      </subject>
   </target>
   <rule effect="deny">
      <condition>
         <ressources>
            <ressource>
               <ressource-match attr="APPS" subject-match="id_ScreenShot" match="ALL" />
            </ressource>
         </ressources>
         <contexts>
            <context>
               <context-match attr="UsedRessources" subject-match="id_BNC" match="android.permission.CAMERA" />
            </context>
         </contexts>
      </condition>
   </rule>
</policy>
```

Figure 5: An example of a security policy presented in the form of XML

```
{
    "@combine": "deny-overrides",
    "@id": "1",
    "@AUTHOR-KEY-CN": "Mahdi",
    "@AUTHOR-KEY-FINGERPRINT": "Mahdi",
    "target": {
        "subject": [
            {
                "@attr": "id_ScreenShot",
                "@match": "com.apps.TakeScreenShot"
            },
            {
                "@attr": "id_BNC",
                "@match": "com.apps.BNCbanque"
            }
        ]
    },
    "rule": {
        "@effect": "deny",
        "condition": {
            "ressources": {
                "ressource": {
                    "ressource-match": {
                        "@attr": "APPS",
                        "@subject-match": "id_ScreenShot",
                        "@match": "ALL"
                    }
                }
            },
            "contexts": {
                "context": {
                    "context-match": {
                        "@attr": "UsedRessources",
                        "@subject-match": "id_BNC",
                        "@match": "android.permission.CAMERA"
                    }
                }
            }
        }
    }
}
```

Figure 6: An example of a security policy presented in the form of JSON

The following scenario is established to extract CAPEF policies to communicate with other policy languages such as XML and JSON:

i Scenario: Prohibit launching the TakeScreenShot application that allows you to take automatic screenshots when the user opens the camera in his BankApp application to send a check.

Table 2: Generic policy description

Element	Description
Policy-set	Presents a table that groups the list of policies.
Policy	Presents the policy object that contains the" Target and" Rule Sub-objects, as well as the attributes:" Combine "which presents the role of the policy, the" AUTHOR-KEY-CN attribute the author identifier of the policy and the attribute AUTHOR-KEY-FINGERPRINT" presents the fingerprint key of the author of the policy.
Target	This is the object that contains the definition of the target applications to control.
Rule	It is the object that defines the security rule, the attribute" effect "presents the decision of the control to give or withdraw the authorization.
Condition	Contains the permissions to remove and the contexts.
Resource-match	Contains different attributes:" attr "which can be an application to block or an API permission," subject-match "contains the application to control and" match "contains the permission to remove.

ii Policy: if (CAMERA in [BNCApp]), then stop the application [TakeScreenShot].

Fig.5. shows the extraction of the above security policy scenario to XML and Fig.6. shows the extraction of the same scenario to JSON security policy.

6. Experimental results

This section presented the evaluation of our CAPEF and the application controller in terms of performance by analyzing: (1) the size of the enforced application after the instrumentation, 2) the execution time of the policy decision (3) The policy size due to the complexity of the applied rules and conditions.

6.1. Enforced Application Size

To measure the effect of the instrumentation method on the original size of the applications, we have instrumented a set of 109 applications using our rewriting framework. Table 3 shows a sample of eighteen applications, the original size and the new size after the instrumented. Indeed, this percentage represents the size of the code added during the control of APIs calls for each application individually

For all instrumented 109 applications, the average size added was 705 bytes, which is about 0.063% of the size of the original applications. Also, as shown in Fig.7, it is very clear that the size added is very small and will have a very small impact on the size of the original applications.

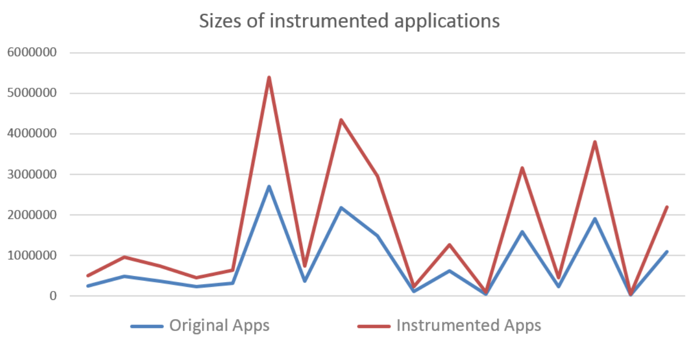


Figure 7: Average added size for 109 instrumented applications

6.2. Execution time of the policy decision

To calculate and evaluate the execution time of our defined policies, we have calculated the decision execution time for several context aware policies with different rules and conditions based on some selected scenarios. The following is a set of scenarios that has been chosen among many others used during our tests. These scenarios will be ranked in ascending order according to their level of complexity.

1. **Scenario 1**: Prohibit launching the TakeScreenShot application that allows you to take automatic screenshots when the user opens the camera in his BankApp application to send a Check.

   Policy: $Deny_{TakeScreenShot}$(app, per, c) ← (TakeScreenshot ∈ app.APIs) ∧ (per.resource==screen) ∧ (c.fgApp.class = banking)

2. **Scenario 2**: Prohibits the RecordAudioMedia application from recording when the user is making a phone call through the PhoneCall application.

   Policy: $Deny_{RecordAudioMedia}$ (app, per, c) ← (RecordAudioMedia ∈ app.APIs) ∧ (per.resource == microphone) ∧ (c.fgApp.class = (callPhone ∨ receivePhoneCall))

3. **Scenario 3**: Prohibit the FakeGPS application from changing the user's location when using one or more of these BankApp, Uber, and Google-Map applications.

   Policy: $Deny_{FakeGPS}$ (app, per, c) ← (FakeGPS ∈ apps.APIs) ∧ (per.resouce == (accessCoarseLocation ∧ accessFineLocation) ∧ (c.fgApp.Class = (BankApp ∧ UBER ∧ GoogleMap))

4. **Scenario 4**: Prohibition of the BankApp application to access the Internet or use the camera when the user is at TimHortons knowing that its longitude = 45.491318 and its latitude = -73.727987.

   Policy: $Deny_{internet \wedge camera}$ (app, per, c) ← ((internet ∧ camera) ∈ apps.APIs) ∧ (per.resource == (GPS = [45.491318, -73.727987]) ∧ (c.fgApp.Class = BankApp)

5. **Scenario 5**: When the user is at the meeting at ETS from 8am to 9am. Prohibit Facebook, Instagram and Gmail applications from accessing the Internet, the

Table 3: The size of applications before and after instrumentation

Application	Original (Bytes)	Instrumented (Bytes)	Size added (Bytes)	Percentage
Contact Identicons	246904	247965	603	0.24%
GPS tracker	22420823	22421668	845	0.0037%
Show web view	483839	484460	621	0.12%
Contact Search	368610	369278	668	0.18%
Contacts Widget	227386	228034	648	0.28%
Beta Updater for WhatsApp	321673	322448	775	0.24%
Contact loader	2701541	2702019	478	0.01%
Photo Manager	366100	366668	568	0.15%
Wi-Fi setup	2177239	2177747	508	0.02%
Time tracker	1477841	1478711	870	0.06%
Calender Trigger	119863	120732	869	0.72%
Calender Color	629361	630232	871	0.13%
Calender Import Export	45823	46772	949	2.07%
CamTimer	1580753	1581393	640	0.04%
OpenCamera	226585	227420	835	0.36%
Microphone	1905254	1905910	656	0.03%
SMS backup	26071	26984	913	3.50%

camera and the location. Prohibit Message application from receiving SMS and MMS. Also, Prohibit the recording feature of RecordAudio application.

Policy: Deny$_{all}$ (app, per, c) ← ((Facebook ∧ Instagram ∧ Gmail∧ RecordAudioMedia ∧ SMS ∧ MMS) ∈ apps.APIs) ∧ (per.resouce ==(GPS = [45.491318, -73.727987] ∧ internet ∧ microphone ∧ phoneCall ∧ receiveCall)) ∧ (c.fgApp.Class = meeting) ∧ (c.time >= 8am ∧ c.time <=9am)

The decision execution time has been calculated for each policy individually as following:

i For the Policy 1 and Policy 2, the test results were fixed because the context does not vary when entering random test values. The execution time for the first policy is 116 ms and for the second policy is 234 ms.

ii For the policy 3, the context is related to three different running applications, but it remains fixed. The execution time for the whole policy is 307 ms.

iii For the policy 4, our context is the location, so the results were more or less close, but they vary according to the change in GPS values. In this case the execution time of the whole policy is 314 ms.

iv For the policy 5, two different contexts were used time and location. The average execution time for notifying each application also was calculated. For the Skype application the execution time is 549 ms, for the Messages application is 592 ms, for the Instagram application is 634 ms, for the Gmail application is 758 ms and for Facebook is 814. Also, the average execution time for the whole policy is 818 ms.

Fig.8, shows the different policies execution times according to the complexity for each policy. All calculations and testes where repeated several times to ensure accuracy. As a result, we have noticed that as more the policy becomes complex the execution time becomes bigger.

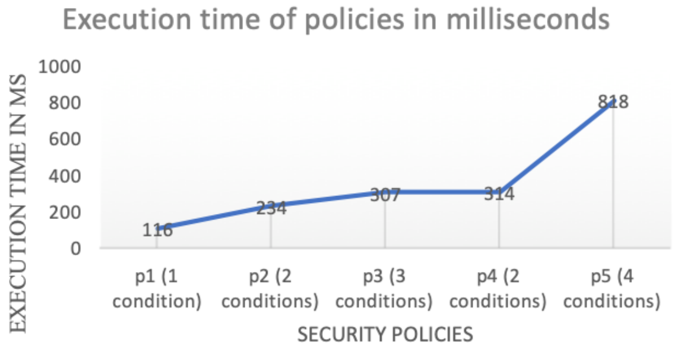


Figure 8: Policies execution times

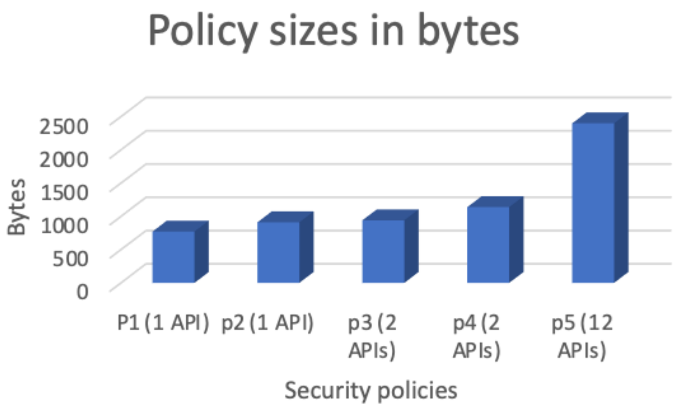


Figure 9: Policies sizes according to their complexities

6.3. Policy Size

The policy size is also changing due to the complexity of the applied rules and conditions. We have calculated the policy sizes on the list of 109 enforced applications. Also, we took the same fife scenarios and their security policies mentioned above in the previous section to make our simulation. Fig. 9 shoes the progression of policy sizes according to their complexities.

7. Conclusion and Future Work

This work addressed problems related to context aware policies for Android applications as its one of the main targets of attackers. We, therefore developed CAPEF, which is a policy specification framework that enforces context-aware inter-app security policies to effectively describe users defined consents. Thorough experiments we have performed a study on the efficiency of CAPEF with respect to the size and execution time of the enforced applications. The evaluation results demonstrated the feasibility of our framework and the effectiveness of our policy specification language in enforcing complex context-aware policies on different Android applications.

In the future, we are planning to improve our model using different ML techniques for varying IoT smart environments. Furthermore, we will implement a security framework that is capable of data security and access control by encrypting all sensitive data and making it available only for the authorized service providers according to the pre-defined context-aware policies.

References

[1] J. Maring, "Android central", Online[Access 12/07/2022] urlhttps://www.androidcentral.com/google-removed-over-700000-malicious-apps-play-store-2017, 2018.

[2] I. Rathore, "Google gets rid of these 16 apps having millions of downloads", Online[Access 15/09/2022] https://dazeinfo.com/2022/10/25/ google-removes-apps-that-have-affected-20-million-android-users-worldwide/, 2022.

[3] "Android developers", Online[Access 02/01/2022]url-https://developer.android.com /guide/topics/manifest/ manifest-intro.

[4] V. Arena, V. Catania, G. La Torre, S. Monteleone, F. Ricciato, "Securedroid: An android security framework extension for context-aware policy enforcement", "Privacy and Security in Mobile Systems (PRISMS), 2013 International Conference on", pp. 1–8, IEEE, 2013, doi:10.1109/PRISMS.2013.6927185.

[5] M. Nauman, S. Khan, X. Zhang, "Apex: extending android permission model and enforcement with user-defined runtime constraints", "Proceedings of the 5th ACM symposium on information, computer and communications security", pp. 328–332, 2010, doi: 10.1145/1755688.1755732.

[6] Y. Zhou, X. Zhang, X. Jiang, V. W. Freeh, "Taming information-stealing smartphone applications (on android)", "International conference on Trust and trustworthy computing", pp. 93–107, Springer, 2011, doi:10.1007/978-3-642-21599-5_7.

[7] P. Hornyack, S. Han, J. Jung, S. Schechter, D. Wetherall, "These aren't the droids you're looking for: Retrofitting android to protect data from imperious applications", "Proceedings of the 18th ACM Conference on Computer and Communications Security", CCS '11, p. 639–652, Association for Computing Machinery, New York, NY, USA, 2011, doi:10.1145/2046707.2046780.

[8] D. Feth, A. Pretschner, "Flexible data-driven security for android", "Software Security and Reliability (SERE), 2012 IEEE Sixth International Conference on", pp. 41–50, IEEE, 2012, doi:10.1109/SERE.2012.14.

[9] R. Xu, H. Saidi, R. Anderson, "Aurasium: Practical policy enforcement for android applications", "21st USENIX Security Symposium (USENIX Security 12)", pp. 539–552, USENIX Association, 2012, 21st USENIX Security Symposium ; Conference date: 08-08-2012 Through 10-08-2012.

[10] J. Jeon, K. K. Micinski, J. A. Vaughan, A. Fogel, N. Reddy, J. S. Foster, T. Millstein, "Dr. android and mr. hide: Fine-grained permissions in android applications", "Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices", SPSM '12, p. 3–14, Association for Computing Machinery, New York, NY, USA, 2012, doi:10.1145/2381934.2381938.

[11] B. Davis, B. Sanders, A. Khodaverdian, H. Chen, "I-arm-droid: A rewriting framework for in-app reference monitors for android applications", *Mobile Security Technologies*, vol. 2012, no. 2, p. 17, 2012.

[12] B. Davis, H. Chen, "Retroskeleton: Retrofitting android apps", "Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services", MobiSys '13, p. 181–192, Association for Computing Machinery, New York, NY, USA, 2013, doi: 10.1145/2462456.2464462.

[13] P. von Styp-Rekowsky, S. Gerling, M. Backes, C. Hammer, "Idea: Callee-site rewriting of sealed system libraries", J. Jürjens, B. Livshits, R. Scandariato, eds., "Engineering Secure Software and Systems", pp. 33–41, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[14] X. Zhang, A. Ahlawat, W. Du, "Aframe: Isolating advertisements from mobile applications in android", "Proceedings of the 29th Annual Computer Security Applications Conference", ACSAC '13, p. 9–18, Association for Computing Machinery, New York, NY, USA, 2013, doi:10.1145/2523649.2523652.

[15] P. Pearce, A. P. Felt, G. Nunez, D. Wagner, "Addroid: Privilege separation for applications and advertisers in android", "Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security", ASIACCS '12, p. 71–72, Association for Computing Machinery, New York, NY, USA, 2012, doi:10.1145/2414456.2414498.

[16] S. Shekhar, M. Dietz, D. S. Wallach, "Adsplit: Separating smartphone advertising from applications", "Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)", pp. 553–567, 2012, doi:10.48550/arXiv.1202.4030.

[17] M. Zhang, H. Yin, "Efficient, context-aware privacy leakage confinement for android applications without firmware modding", "Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security", ASIA CCS '14, p. 259–270, Association for Computing Machinery, New York, NY, USA, 2014, doi: 10.1145/2590296.2590312.

[18] Y. Falcone, S. Currea, "Weave droid: aspect-oriented programming on android devices: fully embedded or in the cloud", "Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering", pp. 350–353, 2012, doi:10.1145/2351676.2351744.

[19] O. Riganelli, D. Micucci, L. Mariani, "Controlling interactions with libraries in android apps through runtime enforcement", *ACM Trans. Auton. Adapt. Syst.*, vol. 14, no. 2, 2019, doi:10.1145/3368087.

[20] M. Alhanahnah, Q. Yan, H. Bagheri, H. Zhou, Y. Tsutano, W. Srisa-An, X. Luo, "Dina: Detecting hidden android inter-app communication in dynamic loaded code", *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 2782–2797, 2020, doi:10.1109/TIFS.2020.2976556.

[21] M. Grace, M. Sughasiny, "Behaviour analysis of inter-app communication using a lightweight monitoring app for malware detection", *Expert Systems with Applications*, vol. 210, p. 118404, 2022, doi:https://doi.org/10.1016/j.eswa.2022.118404.

[22] A. Developers, "Preparing for the android privacy sandbox beta", Online[Access 15/12/2022]urlhttps://android-developers.googleblog.com/2022/11/preparing-for-android-privacy-sandbox-beta.html , 2022.

[23] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, C. E. Youman, "Role-based access control models", *Computer*, vol. 29, no. 2, pp. 38–47, 1996, doi:10.1109/2.485845.

[24] OASIS, "Oasis extensible access control markup language (xacml)", Online[Access 02/05/2017]urlhttp://www.oasis-open.org/committees/xacml, 2011.

[25] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, A. N. Sheth, "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones", *ACM Trans. Comput. Syst.*, vol. 32, no. 2, 2014, doi:10.1145/2619091.

[26] W. Zhou, X. Zhang, X. Jiang, "Appink: Watermarking android apps for repackaging deterrence", ASIA CCS '13, p. 1–12, Association for Computing Machinery, New York, NY, USA, 2013, doi: 10.1145/2484313.2484315.

SAAD INSHI is currently pursuing his PhD in software engineering from École de technologie supérieure, University of Quebec, Montreal, Canada and completed his MASc degree in Information Systems Security from Concordia University, Montreal.

His research interests includes Android and IoT Privacy and security. He is also interested in Context aware privacy and security of devices.

MAHDI ELARBI has completed Masters from École de technologie supérieure, University of Quebec, Montreal, Canada. He is currently working as a senior Software Developer in Montreal.

His research interests includes Android and IoT security.

RASEL CHOWDHURY is pursuing his PhD in software engineering and completed his MSc degree in Information Technology Engineering from École de technologie supérieure, University of Quebec, Montreal, Canada.

His research interests includes Cloud Computing, Cloud Native orchestration, security and privacy of IoT, IoE and IoV.

HAKIMA OULD-SLIMANE is currently a professor at the Département de Mathématiques et d'Informatique, Université du Québec à Trois-Rivières, Trois-Rivières, Canada. She obtained her Ph.D. degree in Computer Science from Laval University, Québec, Canada.

Her research interests include mainly: information security, cryptography, preserving data privacy in smart environments, reliability of collaborative computing and formal methods.

CHAMSEDDINE TALHI is currently a Full Professor with the Department of Software Engineering and IT, École de Technologie Supérieure, University of Quebec, Montreal, Quebec, Canada.

He is leading a research group that investigates efficient security mechanisms for smartphone, IoT, edge and cloud infrastructures. His current research interests include cloud native telco services management and security, DevOps security, and federated learning for mobile cloud and IoT.