A Generic Lightweight and Scalable Access Control Framework for IoT Gateways

Juan D. Parra Rodriguez(✉)

IT-Security Group, University of Passau, Passau, Germany
dp@sec.uni-passau.de

Abstract. Gateways prevail in IoT (Internet of Things) set-ups for connectivity, privacy, and other reasons; however, there has not been a generic and open-source framework offering authentication, identity management, policy administration and policy evaluation as a service for such a scenario. Meanwhile, cloud-based security solutions are available, but they use too much memory and CPU to be deployed in low-cost hardware typically used for IoT gateways such as the Raspberry Pi.

In our work, we identified critical requirements for a generic security framework that could be deployed to low-cost hardware used for IoT gateways. From this point on, we implemented the security framework, and modified a Content Management System (CMS) to rely on the framework for authentication and policy evaluations.

We evaluated our component's runtime performance and computational resource consumption in comparison to a popular attribute-based security framework written in Java. We measured the CPU, memory, and network usage for each security framework, their databases, and the CMS across three different hardware platforms. To ensure our results are not biased towards a particular hardware set-up, we chose hardware with two different processor architectures, different capabilities and vendors. Our results indicate that our framework not only requires less time to complete requests but also makes less intensive use of the processor and the memory, i.e., the most critical capabilities for IoT gateways today.

Keywords: Access control · IoT gateway · Identity management

1 Introduction

Affordable single-board computer hardware equipped with WiFi, Bluetooth, I/O pins, among other features lets developers and makers create applications to

This research has been supported by the EU under the H2020 AGILE (Adaptive Gateways for dIverse muLtiple Environments), grant agreement number H2020-688088. Also, the authors would like to thank Eduard Brehm for the WordPress integration and various updates in the agile-security code-base.

Published by Springer Nature Switzerland AG 2019
O. Blazy and C. Y. Yeun (Eds.): WISTP 2018, LNCS 11469, pp. 207–222, 2019.
https://doi.org/10.1007/978-3-030-20074-9_15

obtain information from sensors easily[1]. Also, the increasing privacy awareness has influenced users to store sensor data locally instead of delivering it directly to IoT clouds and services[2] when possible.

Keeping data locally is undoubtedly one step towards data privacy; however, in scenarios where gateways need to offer multi-tenant support, there is an additional need to enforce security policies on data stored in the gateway and other services running in the same environment. To support this, we have developed a prototypical implementation of a security framework, called agile-security[3]. Agile-security allows applications running inside or outside the gateway to rely on authentication, identity management, policy administration and authorization as a service. In this way, access to sensors, application APIs, or other security-sensitive assets can be managed centrally by the security framework hosted in the gateway. Agile-security supports a generic attribute-based identity and access control model to remain as flexible as possible.

Despite considerable research towards IoT application security, there has not been an open-source solution to handle the security requirements for gateway-based scenarios in a generic, lightweight and scalable manner. On the one hand, researchers have previously argued for a capability-based approach whereby a certificate referencing a subject, i.e., user, and its access rights is presented to the service provider, i.e., a device [9,15]. However, this requires Certification Authorities (CA)s to sign certificates and assumes that users interact directly with devices. Closer to our research, there have been efforts towards enforcing security policies for brokers or gateways using MQTT, HTTP and CoAP brokers [5,6,12]; however, these integrations with specific protocols fail to provide a generic framework to build security solutions.

From a different perspective, cloud systems rely on centralized components offering authentication, attribute-based authorization and policy management as a service. There are commercially available implementations from Oracle [2], Microsoft [1], and IBM [14], as well as an open-source implementation called WSO2 Balana [3]. However, cloud-based security components require computational resources beyond the capabilities of affordable single-board computers commonly used to host IoT gateways. Instead, our solution can be deployed in smaller single-board computers, and uses less resources than WSO2 and scales[4] to medium cloud-based set-ups with some configuration adjustments.

Our **contributions** can be summarized as follows: (1) we outline requirements and challenges faced while developing of a generic, lightweight, and scalable attribute-based security framework. (2) we explain how we addressed such challenges during the implementation of our security framework. (3) we perform a quantitative comparison between our framework and the WSO2 server in a realistic scenario. To this end, we use automated UI-testing to visit a modified

[1] Single-board computers such as the Raspberry Pi, the Beaglebone board or the UPBoards are computers (ARM- or Intel-based) available from 30 to 60 USD.

[2] There are several IoT specific clouds, sich as Xively, Amazon IoT or Thingspeak.

[3] Available at: https://github.com/agile-iot/agile-security.

[4] Scalability means the agile-security can be configured differently depending on the hardware available, e.g., to use more resources and provide responses faster.

CMS using one of the two security frameworks to assess the runtime performance as well as the usage of computational resources.

This paper is structured as follows: We introduce basic terminology used across the paper in Sect. 2. Afterward, we describe the requirements and provide an overview of the security framework in Sect. 3. Section 4 shows an example of how agile-security can implement role-based access control policies. After concluding the conceptual description of the framework, we evaluate it in comparison to WSO2 in Sect. 5. Finally, we present related work and our conclusions in Sects. 6 and 7, respectively.

2 Attribute-Based Access Control Definitions

For clarity, we describe our work using well-established terminology presented in the attribute-based concepts provided by NIST from Hu et al., which states:

*"A logical **object** -sometimes referred to as a resource- is an entity to be protected from unauthorized use.*

*The term **subject** is used to denote a human or non-person-entity requesting access to an object.*

*Privileges represent the authorized behavior of a subject; they are defined by an **authority** and embodied in policy or rules.*

***Digital Policy (DP):** Access control rules that compile directly into machine executable codes or signals. Subject/object attributes, operations, and environment conditions are the fundamental elements of Digital Policies, the building blocks of Digital Policies rules, which are enforced by an access control mechanism.*

***Meta Policies (MP):** A policy about policies, or policy for managing policies, such as the assignment of priorities and resolution of conflicts between Digital Policies or other Meta Policies."* [10]

3 Overview of the Security Framework

We start by listing the requirements addressed by our security framework. Then, we explain conceptually how generic Digital and Meta Policies can be achieved, followed by a description of the policy evaluation process, the identity model and the support for authentication mechanisms. We have used Node JS, a server-side JavaScript runtime, for the development of the agile-security framework.

3.1 Requirements

The security framework must:

R1. Allow users to define entities, i.e., objects and subjects, and security policies with the highest flexibility possible.

R2. Be usable from different kinds of applications (web, mobile, cron-jobs, command line programs, and other applications) and regardless of their location and operating system (running on the gateway or in a server).

R3. Perform efficiently in affordable single-board computer hardware, as well as more expensive servers (cloud).
R4. Be modular and extensible, so developers can add or disable functionality easily to fulfill their particular needs.
R5. Be easy to integrate for developers through libraries, standard interfaces, or rapid prototyping tools used in IoT environments.

3.2 Generic Digital Policies and Meta Policies

To address R1, developers must be able to define their own security policies with the highest flexibility possible. Thus, we need to provide a generic entity model that allows developers to define subjects and objects freely. Agile-security tackles this by letting developers specify *entities to represent subjects and objects in the same way*. Also, there are two key considerations. First, the framework must allow developers to define Digital Policies on the attributes and actions corresponding to entities. Second, the model needs to provide means to specify who is the *authority* for each attribute, i.e., a Meta Policy, generically.

A policy evaluation mechanism is the main building block for a framework for managing identities and access control rules. To represent how we leverage the policy evaluation mechanism across agile-security, Fig. 1 illustrates the relationship between the policy evaluation mechanism, Digital Policies, Meta Policies and the representation of entities in the identity model. First of all, Fig. 1 shows the mechanisms to evaluate and manage Digital and Meta Policies in grey. They are shown in the same color because they use the same policy evaluation mechanism described in Sect. 3.3. Digital Policies enforce access to attributes and actions that can be performed by, or on, entities. Furthermore, the picture illustrates different levels of customization that may be required by specific applications.

The first level, on the left-hand side of Fig. 1, shows a mechanism with enough flexibility to evaluate Digital Policies on attributes and actions of entities. However, in the first level, such a model would not allow users to define who can update Digital Policies. As a result, policies can only be applied to every kind of entity in the same way without giving users the possibility to update policies. This kind of mechanism is commonly referred to as Mandatory Access Control (MAC) because users cannot choose to override security mechanisms applied system-wide.

On the second level, the security model can be extended with the capability to evaluate Meta Policies, i.e., policies enforcing access to Digital Policies. Having fixed Meta Policies creates the opportunity for users to update the Digital Policies. In other words, models implemented with 2 levels or more allow for Discretionary Access Control (DAC), as users can update Digital Policies according to Meta Policies. Similarly, three levels allow to update the Meta Policies.

The result of our work implements a security framework that can evaluate policy hierarchies of level n. Notwithstanding, we do not foresee the need of using any level higher than 3, as this would increase the complexity of the system significantly and make it prone to human errors.

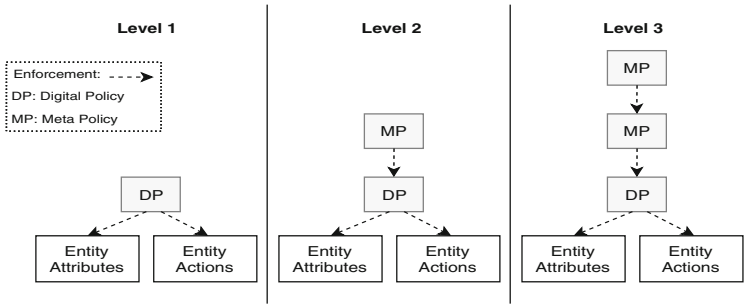


Fig. 1. Policy interactions to realize the security model

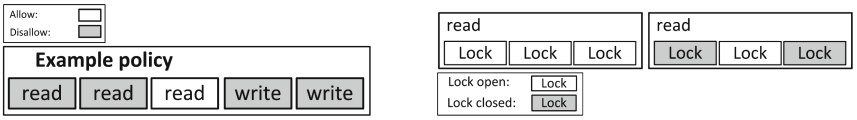
3.3 Policy Evaluation Framework

We used the UPFROnt policy evaluation framework[5]. UPFROnt defines policies as a collection of blocks specifying restrictions on reading or writing operations. Users can execute read, or write, operations as long as there is at least one read, or write, block allowing this. Figure 2a shows an example where a read operation would be allowed, while write operations would be denied. Besides, within each block, there can be zero or more locks where each one evaluates to a boolean value. All locks must evaluate to true to allow a block of the policy, i.e., white read block in Fig. 2a. Thus, the evaluation of a block is calculated by joining the boolean value returned by every lock with an and operator. Figure 2b shows two locks: the read block on the left is allowed, while the read block on the right is not allowed. More to the point, in the case of Fig. 2b, the read action would still be allowed overall, as the requirement is to have at least one block allowing the operation. The composition of locks and blocks in this way creates a boolean formula in Disjunctive Normal Form (DNF).

The approach followed by UPFROnt is an extension to the parametrized locks proposed by Broberg and Sands [4]. During policy evaluations, locks receive attributes for both entities, i.e., subject and object, and additional parameters specified in the policy. For example, an `attributeEquals` lock would receive two arguments: the attribute name, and the expected value. In this way, this lock can be used to assert that a user has an admin role by specifying a policy with a block, which contains the `attributeEquals` lock with the arguments "role" and"admin". Similarly, the `isOwner` lock verifying if a subject owns an object does not take any arguments, but compares the "id" attribute of the subject with the"owner" attribute of the object.

A flexibility aspect of the UPFROnt component is that developers can plug in their code to evaluate locks. These locks are executed within the policy framework and can have a state; moreover, the lock implementation can use any API offered by the Node JS framework which contributes to the generic approach

[5] UPFROnt has been developed by Daniel Schreckling and is available at https://github.com/SEDARI/UPFROnt.



(a) Policy blocks (b) Locks within a block

Fig. 2. Policy components

(R1) and modularity (R4). Initially, UPFROnt includes the `attributeEquals` and the `isOwner` lock. We have developed new locks to support logging of actions and management of groups.

As shown by Fig. 1, the evaluation mechanism described in this section applies to Digital Policies governing write and read access to attributes and actions that can be performed on entities. Moreover, the policy evaluation based on locks is also used to evaluate Meta Policies enforcing access to Digital Policies.

3.4 Identity Model

Based on the policy evaluation already presented, the identity management component within agile-security lets developers define the entity format, i.e., which fields are required for each type of entity through a JSON schema specification loaded from a configuration file. For example, a developer can specify that he requires a kind of entity called "sensor" with the possible attributes "location", and "dataType". Also, developers can specify which attributes are allowed, their type, and whether they are mandatory. If developers choose so, they can even restrict possible values an attribute can take.

The identity management has privileged attribute names that cannot be used by developers; these are "id", "owner", "entity_type" and "auth_type". We do this to ensure that the identity management system assigns an "id", an "owner", and an "entity type" for every entity during its creation. The owner's identifier is set to match the identifier of the entity's creator except for users. Users own themselves as they are the root of the ownership hierarchy. Further, as agile-security supports several authentication mechanisms, the identity management ensures that users always have the "auth_type" to determine which authentication mechanism must be used.

Identity management handles entities and performs access control on attributes. To achieve this, identity management validates whether the user sending a request can perform the action, e.g., update attribute. If this check is successful, the relevant read or write Digital Policies for each attribute are evaluated. Subsequently, if a user can read an entity but the Digital Policies disallow access to a particular attribute, then the identity management framework removes the attribute from the response. This allows for a simple declassification mechanism and grants access to attributes selectively based on the user's Digital Policies.

Like with the definition of entities, the configuration file for agile-security includes default Digital Policies enforcing access to each attribute. This allows

developers to set-up a security model based on their needs. The definition of Digital Policies on attributes specify who is the *authority*, according to the NIST definitions, to update or read the attribute. In addition, the configuration file also specifies the level of Meta Policies supported according to Fig. 1 and default MetaPolicies applied to each Digital Policy. As a result, developers can decide whether they need Meta Policies, how many levels, and how they should rule the access to Digital Policies. Last but not least, we solve the bootstrapping problem by including information in the configuration file to create entities to be created in the first boot, e.g., first administrator of the system.

3.5 Wide Support for Authentication Mechanisms

Applications can rely on agile-security as an OAuth2 Identity Provider (IdP). On the one hand, this tackles ease of integration requirement (R5) because many libraries are implementing OAuth2 clients in many programming languages and operating systems. On the other hand, by implementing every token grant specified by OAuth2 in agile-security, we ensure that all sorts of applications running inside or outside the gateway can rely on agile-security as an IdP (R2). Particularly, as we implemented all the authorization flow grants from OAuth2, we ensure that not only Web applications are supported. Also, command line or even cron-jobs can rely on agile-security. In addition to offering standard OAuth2 interfaces, we developed a JavaScript library encapsulating the authentication, identity management, policy administration, and policy decision for ease of integration (R5). We also provide extensions for Node-RED, a visual development environment used for IoT applications, connecting the policy framework too.

For ease of integration into existing applications that already rely on other IdPs, e.g., Google, agile-security addresses the extensibility requirement R4 by letting developers define passport source files (a Node Js authentication framework) to add new authentication mechanisms besides local users handled by agile-security. To exemplify this, agile-security already contains strategies to rely on authentication from Google, Dropbox, PAM (Linux Pluggable Authentication Modules) and WebID.

4 Digital Policies Example (Role-Based Access Control)

This section illustrates an instance where the security model is used to represent a simple role-based access control model using identity definitions and policies. Even though passwords are not needed when developers rely on external authentication mechanisms, e.g., Google, we show an example of role-based access control where users do have a password attribute. First, the entity schema needs to specify the "role" and"password" attributes.

The policies represented by Fig. 3 rely on the `attributeEquals` and `isOwner` locks from Sect. 3.3 to define the role-based access control model[6]. The left-hand

[6] In addition to these policies, a policy allowing everyone to execute an action may also be needed. To this end, a block without any locks lets the user access everything.

side shows a policy allowing only users who own an entity to perform an action the attribute, Digital Policy or Meta Policy. Similarly, the figure on the right-hand shows a policy where only administrators are allowed.

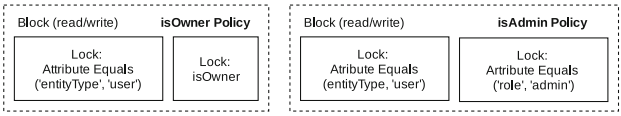


Fig. 3. Policy samples

In addition to the evaluation process presented in Sect. 3.3, UPFROnt has a hierarchical way of evaluating policies for simplicity and efficiency. In particular, entities are objects containing attributes that can be strings, numbers, but also objects. As a result, entities can have nested attributes. To avoid forcing users to set a policy for each nested property, UPFROnt uses the concept of a top-level policy. The top-level policy from any point in the object hierarchy applies to child nodes of a given attribute.

Figure 4 shows a possible agile-security configuration implementing a simple role-based access control model using the policies introduced in Fig. 3 and the top-level concept. To ensure that only administrators can create users, the top level policy for the user entity is set to allow everyone to read, but ensure that only owners or administrators can write to attributes within the entity (unless elements below in the hierarchy override them). As the creation of a user implies writing the attributes of the newly created user to set them, agile-security prevents non-admin users from creating users in this setting.

For clarity, we show light grey policies in front of the attributes when they have been inherited by a top-level policy above them, i.e., id and owner. In addition to this, we show an entity model where only administrators can set the attribute role. This ensures that users cannot upgrade their privileges on their own because there is no writeOwner policy for the role attribute. Conversely, the password attribute can be set by the owner and administrators; however, administrators cannot read the password. The previous example shows how to achieve interesting properties to handle the password and role attribute to balance the authority for a role attribute (set role), and the user's privacy (read password).

In more complex scenarios where Meta Policies are involved, agile-security links them using the tree structure presented in Fig. 1. In this way, the security framework traverses the tree, starting from the entity or action, to validate whether a particular Digital Policy or Meta Policy can be changed. If there is no parent for a Digital or Meta Policy, this means it cannot be updated.

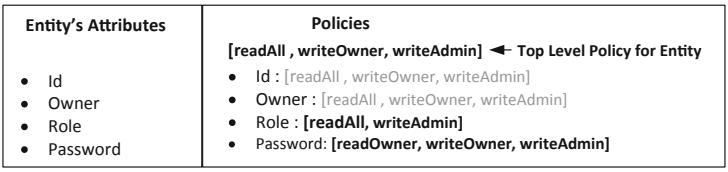


Fig. 4. Role-based access control model example

5 Evaluation

Every system faces a trade-off between using computational resources and providing a good response time. For example, loading all data in memory instead of placing it in a hard drive decreases the response time but may starve other processes of memory. To validate how well our security framework can execute in single-board computers, as well as bigger setups (R3), we perform an extensive quantitative evaluation. This is critical to obtain all aspects related to trade-offs between resource consumption and efficiency.

5.1 Scenario

To obtain a realistic scenario, we modified the most popular CMS[7] currently, i.e., WordPress, to outsource authentication and authorization to an external security framework. We created two branches of WordPress version 4.9.5 overriding security functions validating whether users are allowed to see a particular page or open the administrative dashboard. One branch of WordPress uses the Balana WSO2is server version 5.3.0, and the other one uses agile-security.

A factor motivating us to use WordPress as an example is that it evaluates more than 70 capabilities (mapped to each security framework) while actions are taken by a user; more to the point, each capability is evaluated separately. This sub-optimal setting is not desirable for a production environment because it triggers a separate network request with headers or XML content from WordPress to the security framework. However, this sub-optimal environment provides us a worst-case scenario where an application makes intensive use of the APIs from the security frameworks under evaluation. What is more, if we can establish that our security framework works for this environment, the runtime performance and resource use can only improve after optimizations are applied.

Concerning the software set-up, we always had a modified WordPress relying on one of the two security frameworks (WSO2 or agile-security). However, agile-security can be executed in two ways: either using an external database (MongoDB) or using a database running in the same process (LevelDB). This helps agile-security to remain flexible to the requirements of a given application and execute in less or more resource-constrained environments. Thus, we

[7] As of October 2018, WordPress has 59.9% of the CMS market share: https://websitesetup.org/popular-cms/.

evaluate the performance of WSO2 connected to an external MySQL database, agile-security connected to an external MongoDB database, or agile-security using LevelDB (same process as the security framework).

To isolate resource consumption per component, we have deployed WordPress, the security frameworks and their databases (when they are separate processes) in separate containers. In turn, this allows us to use docker APIs to monitor the amount of memory, network, and CPU used by each component. Also, to ascertain properties for the security frameworks used without relying on a particular hardware implementation, and to obtain a big picture on the performance of the security frameworks, we executed the evaluation on three different hardware devices shown in Table 1. All our experiments use 64-bit processors and use the two most prominent processor architectures.

Table 1. Hardware configurations

Property	Raspberry Pi 3(B)	Upboard(UP-CHT01)	Lenovo T470S
Memory	1 GB	2 GB	16 GB
CPU	Quad Core 1.2 GHz	Quad Core 64 1.92 GHz	Quad Core 2.70 GHz
Storage	SD card size	16 GB	500 GB (Solid state)
Architecture	ARMv7	x86_64	x86_64

Also, we created an automated web test to interact with the WordPress interface to log in a user, open the dashboard, log out and visit the public site, using Cypress (a Web UI automation framework). Figure 5 shows the use of the docker containers, the security frameworks, and the UI testing framework. We recorded resource consumption continuously, while 100 interactions were performed automatically by the UI testing framework. Each interaction from the UI framework had two actions. First, an admin user logged in and then WordPress would forward him to the administrative dashboard page. Afterward, the user would log out and therefore load the public site. We recorded the time to load the dashboard and the public page, i.e., login and log out.

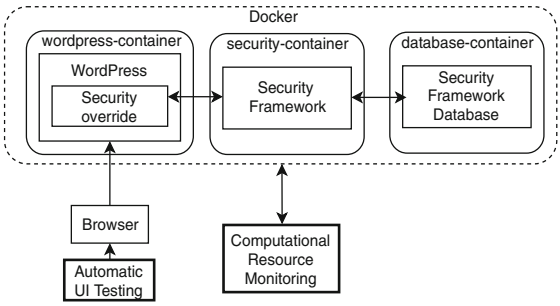
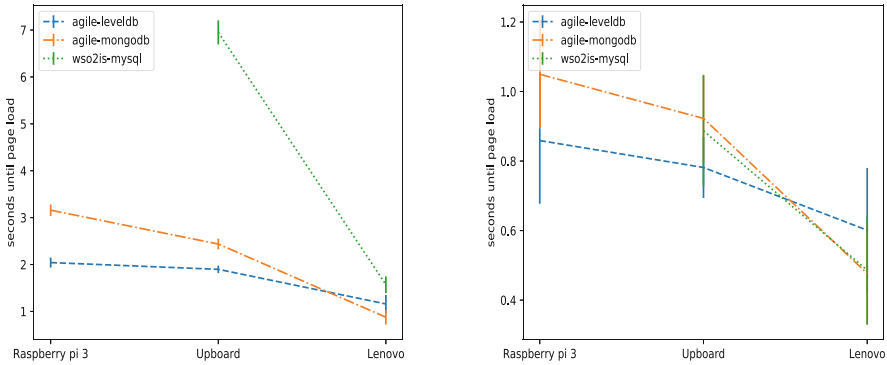


Fig. 5. Evaluation set-up

In the following measurements, there are no results for WSO2 on the Raspberry Pi 3 because WSO2 requires the hardware to have at least 2 GB of RAM to execute. On the contrary, we show that agile-security can be executed in the Raspberry Pi 3, and therefore imposes much less restrictive requirements on the hardware level than WSO2.

5.2 Runtime

We calculated the mean of 100 WordPress interactions, and their standard deviations to assess the response time of WordPress while relying on WSO2 or agile-security to evaluate the policies associated with the capabilities required to render the dashboard or log out the user. Figures 6a and b show the results measured in seconds to load each page. The amount of time measured during the page load is higher than the actual time due to the execution of the automation framework to interact with the UI. Still, this quantitative measure helps to compare the performance of each setting. To represent the results intuitively, we sort the hardware, in ascending order, from less to more powerful.



(a) Login and loading of dashboard

(b) Logout and loading of public stie

Fig. 6. Mean and standard deviation of loading time for 100 visits

From both figures, it can be observed that agile-security has a setting, either with levelDB or MongoDB, that offers better speed than WSO2. Also, in the case of agile-security it is better to use LevelDB, i.e., running the database code within the same process as the security framework, than executing a separate database, i.e., MongoDB, for small single-board computers, e.g., the Raspberry Pi. Notwithstanding, as more hardware is available for the external database, it is more efficient to separate the security business logic from the database to take advantage of the computational resources and achieve a better response. For both actions, login, and log out, the point where one should divide the database from agile-security lies somewhere between the resources available in the Upboard and

the Lenovo laptop. The standard deviations represented for both figures show that measurements have stable values across the 100 experiments.

Figure 6a also shows that agile-security provides the same loading time on a Raspberry Pi 3 than the time achieved in the Lenovo laptop for WSO2. Besides, agile-security provides better loading time for the dashboard (log in) than WSO2 for all our experiments. We must also clarify that, although the deviations seem bigger in Fig. 6b, this is a visual effect due to the change of scale, i.e., all logout actions lie below 1.2 s.

5.3 Resource Consumption

It is clear that agile-security provides better runtime performance than WSO2. However, this section assesses the computational resource consumption to validate whether the performance improvement is sustainable in terms of hardware.

We obtained the number of bytes used in memory, the number of bytes sent through the network, and the number of processor ticks used by each component. Even though counting ticks does not have an intuitive meaning, it provides a way to compare the use of computational power in each scenario. We opted for this approach instead of showing percentage of use of CPU, as the latter is inherently biased by the underlying hardware, i.e., 10% of use in the Lenovo laptop is not comparable to 10% of use the Raspberry.

Docker provides one event every second with statistics on resource consumption. Thus, we calculate the average consumption value per second for every resource and then plot it in Table 2. Moreover, we use the same ordering as Sect. 5.2 to improve readability. On top, we represent values in the table as a heat-map showing higher numbers with a darker background. The values are only compared vertically; that is to say, there are separate scales for memory, network and CPU ticks. Also, the agile-leveldb set-up does not contain a database measurement because LevelDB is executed in the same memory space as the agile-security framework and therefore does not require a separate process.

By considering the results of the runtime evaluation, agile-security should be used without the database in both single-board computers because agile-security requires more processing power, network and memory than LevelDB. On the contrary, it is sensible to use agile-security with MongoDB for the Lenovo laptop, where there is a runtime improvement in comparison to LevelDB.

Following this reasoning, the table shows that agile-security and its database make intensive use of the network; especially, if this compared to the network consumption between WSO2 and MySQL. This is clearly due to the transport protocols used, i.e. MySQL is binary and MongoDB uses HTTP. Luckily, intensive use of network between agile-security and its database is not an issue when users deploy the security framework and the database in the same device. Even though communications between agile-security and its database are classified as networking in our experiment, this traffic is routed through the loopback interface, without requiring actual network transmissions.

On the other hand, WSO2 uses more networking between the security framework and WordPress. Unlike in the case of agile-security and its database, WSO2

Table 2. Memory, network and CPU consumption per component

hardware	setup	container	memory (MB)	network (KB/s)	cpu (Mtics/s)
Raspberry-pi3	agile-mongo	framework	75.73	217.44	344.51
		wordpress	63.66	50.24	62.20
		database	73.41	207.44	19.29
	agile-leveldb	framework	41.46	8.86	270.31
		wordpress	64.32	118.85	73.85
Upboard	agile-mongo	framework	90.04	185.23	266.61
		wordpress	77.33	39.47	58.59
		database	65.82	210.54	48.44
	agile-leveldb	framework	94.20	16.31	254.05
		wordpress	112.69	104.79	58.66
	wso2is-mysql	framework	1648.28	90.98	570.23
		wordpress	117.05	76.78	123.14
		database	126.43	45.97	42.40
Lenovo-t430	agile-mongo	framework	89.74	228.63	92.15
		wordpress	95.30	52.36	19.33
		database	33.15	220.53	15.57
	agile-leveldb	framework	82.57	8.45	67.79
		wordpress	109.68	64.17	22.14
	wso2is-mysql	framework	1239.83	157.00	196.97
		wordpress	141.08	160.36	59.94
		database	80.77	112.81	14.69

(or agile-security) and WordPress are more likely to be deployed in separate hardware as WordPress is a relying party decoupled from the security framework. Also, WSO2 makes intensive use of memory and processor in both scenarios where it can be executed, i.e., Upboard and Lenovo. These are resources that are critical to ensuring that additional applications can be deployed in an IoT gateway.

Even though we already established that WSO2 has lesser runtime performance than agile-security for all the hardware we tested, it is particularly problematic to use WSO2 for deployments where the IoT gateway has limited memory capacity. In particular, Table 2 shows that WSO2 consumes 1.6 GB out of 2 GB available to the whole system in the Upboard. Also, the overhead of the eXtensible Access Control Markup Language (XACML) policies can be observed by an increase in the amount of memory and processor required by WordPress in comparison to the settings where agile-security was used in the same hardware.

5.4 Limitations

We performed precise measurements regarding resource consumption and interactions involving the system modified to use the security frameworks, i.e., WordPress. However, we were able to run experiments only to the point equivalent to a medium-sized server, i.e., 16 GB of RAM. So, even though we conclude that our framework is better for IoT gateways and medium-sized set-ups, we do not assert that agile-security replaces the niche where WSO2 is currently used, i.e., bigger cloud set-ups. In this sense, Java technologies are deployable in enterprise servers to form clusters, which lie beyond the capabilities of agile-security.

6 Related Work

Fysarakis et al. described an instance where a centralized ABAC system was used to enforce policies in a Smart Home environment based on the Sun Java implementation of XACML. Although their approach focuses on attribute-based access control, like ours, their contribution is on an architectural level [8]. Also, The approach from Fysarakis et al. requires capabilities beyond affordable single-board computers, i.e., at least 4 GB of RAM to execute the policy decision point.

Colombo et al. [6] and Niesse et al. [12] propose the integration of policies for the IoT directly within MQTT brokers. Although this approach shares the gateway-centric perspective, it goes into details regarding each protocol. Instead, we provide a security component usable from internal and external applications. Hao et al. proposed JACPoL: a simple access control policy language in JSON [11]. Their approach is to provide an attribute-based language more lightweight than XACML. Their evaluation was done using 16 GB memory and a 2.6 GHz and considered only the time required to reply to policy requests. Although we also use a JSON-based policy language, we based our approach on parametrized locks [4] which allows developers to integrate code for the policy evaluation. Also, our evaluation is more concerned with a realistic scenario and provides a better overview of computational resource consumption. There is an analysis of gateway-centric scenarios sharing data with third parties [13] by Parra et al., but their focus considers only architectural aspects and technologies useful to provide access control towards some parties involved. Also, there have been extensions to provide an OAuth2-based architecture for the IoT [5] and to extend the WSO2 server to use flows authenticate devices [7].

7 Conclusion

XACML-based access control is used for enterprise large-scale applications, where there is trained personnel to configure XACML policies, and policy decision points. However, the knowledge required to specify policies and the nuances related to its configuration are baffling to most developers dealing with smaller set-ups. On top, the resource consumption of XACML servers, e.g., WSO2, is prohibitively high for an IoT set-up. Thus, we close the gap where developers need authentication and attribute-based policies deployed in a single-board computer.

To save resources while offering flexibility for the policy definition and evaluation, agile-security loads an entity specification along with default policies applied to new entities of each type such as users, OAuth2 clients, devices, or any other entity defined by developers. Also, agile-security can allow the update of Digital Policies used to enforce access to attributes and other read or write actions on entities. This is achieved through a generic, hierarchical, structure of policies that yield Meta Policies. For clarity, we show a simple scenario where agile-security is used to implement a role-based access control model. In this model, the role attribute is protected from unauthorized writes, yet keeping the user's password secret even from administrators.

The policy framework lowers the entry barrier for developers to use a security framework, in comparison to XACML servers. In particular, agile-security allows the definition of policies based on atomic and simple building blocks, i.e., locks, computed for the policy evaluation. At the same time, security experts and developers can plug-in custom logic in locks achieving extensibility. From the authentication perspective, agile-security can be used as an IdP from a vast set of applications ranging from batch jobs to mobile or web applications because it supports all authorization codes specified in the OAuth2 protocol.

Aside from showing the way to achieve flexibility, extensibility, and ease of deployment, we evaluate the resource consumption and response time of agile-security in comparison to WSO2, a popular open-source Java XACML solution. We establish that our approach saves resources and provides a lightweight framework. Also, our solution scales as more hardware is available after changing the configuration settings and using an external database.

After executing experiments with 100 visits to a modified WordPress instance, we conclude that our framework offers better runtime performance than WSO2 in all scenarios. More to the point, computational resource consumption is also lower as our solution uses more networking in the loopback interface than the WSO2 server, but saves memory and CPU: the most limited resources in an IoT gateway. Still, we clarify that our analysis in the scope of the paper does not claim that agile-security outperforms WSO2 in all set-ups. We believe there is a clear need for services like WSO2; however, such services should not be used for IoT gateways due to their high resource consumption.

References

1. Microsoft Claim-based Identity Model (2018). https://docs.microsoft.com/en-us/dotnet/framework/security/claims-based-identity-model. Accessed 03 Oct 2018
2. Oracle Identity Mgmt. Fine Grained Authorization: Technical Insights for using Oracle Entitlements Server (2018). http://www.oracle.com/technetwork/middleware/oes/oes-product-white-paper-405854.pdf. Accessed 03 Oct 2018
3. WSO2 Balana Implementation (2018). https://github.com/wso2/balana. Accessed 03 Oct 2018
4. Broberg, N., Sands, D.: Paralocks: role-based information flow control and beyond. In: Proceedings of the 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2010, pp. 431–444. ACM, New York (2010). https://doi.org/10.1145/1706299.1706349
5. Cirani, S., Picone, M., Gonizzi, P., Veltri, L., Ferrari, G.: IoT-OAS: an OAuth-based authorization service architecture for secure services in IoT scenarios. IEEE Sens. J. **15**(2), 1224–1234 (2015). https://doi.org/10.1109/JSEN.2014.2361406
6. Colombo, P., Ferrari, E.: Access control enforcement within MQTT-based internet of things ecosystems. In: Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies, SACMAT 2018, pp. 223–234. ACM, New York (2018). https://doi.org/10.1145/3205977.3205986
7. Fremantle, P., Aziz, B., Kopeck, J., Scott, P.: Federated identity and access management for the internet of things. In: 2014 International Workshop on Secure Internet of Things, pp. 10–17, September 2014. https://doi.org/10.1109/SIoT.2014.8

8. Fysarakis, K., Konstantourakis, C., Rantos, K., Manifavas, C., Papaefstathiou, I.: WSACd - a usable access control framework for smart home devices. In: Akram, R.N., Jajodia, S. (eds.) WISTP 2015. LNCS, vol. 9311, pp. 120–133. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24018-3_8
9. Gusmeroli, S., Piccione, S., Rotondi, D.: IoT access control issues: a capability based approach. In: 2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, pp. 787–792, July 2012. https://doi.org/10.1109/IMIS.2012.38
10. Hu, V.C., et al.: Guide to Attribute Based Access Control (ABAC) Definition and Considerations (2014). https://doi.org/10.6028/NIST.SP.800-162
11. Jiang, H., Bouabdallah, A.: JACPoL: a simple but expressive JSON-based access control policy language. In: Hancke, G.P., Damiani, E. (eds.) WISTP 2017. LNCS, vol. 10741, pp. 56–72. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93524-9_4
12. Neisse, R., Steri, G., Baldini, G.: Enforcement of security policy rules for the Internet of Things. In: 2014 IEEE 10th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), pp. 165–172, October 2014. https://doi.org/10.1109/WiMOB.2014.6962166
13. Rodriguez, J.D.P., Schreckling, D., Posegga, J.: Addressing data-centric security requirements for IoT-based systems. In: 2016 International Workshop on Secure Internet of Things (SIoT), pp. 1–10, September 2016. https://doi.org/10.1109/SIoT.2016.007
14. Schefenacker, S.: Portal Access Control Attribute Based Security for WCM Content (2018). https://www.ibm.com/developerworks/community/groups/service/html/communityview?communityUuid=8f2bc166-3bdc-4a9d-bad4-3620dbb3e46c#fullpageWidgetId=Wc5d73787a343_444e_a578_049379d72276&file=d898a782-82e5-43a1-86f1-4d983b342256. Accessed 03 Oct 2018
15. Tandon, L., Fong, P.W.L., Safavi-Naini, R.: HCAP: a history-based capability system for IoT devices. In: Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies, SACMAT 2018, pp. 247–258. ACM, New York (2018). https://doi.org/10.1145/3205977.3205978