Title of the article

Mahieddine Yaker
and Julien Cartigny
and Gilles Grimaud
IRCICA
University of Lille
address in Lille
Email: yyy@yyy.com

Chrystel Gaber
and Xiao Han
and Vicente Sanchez-Leighton
Orange Labs,
Châtillon, France
Email: firstname.lastname@orange.com

***Abstract*—In recent years, Internet of Things devices(IoT) and Cyber-Physicals Systems(CPS) are ubiquitous and used in many situations (avionic, vehicles, household devices, smartphones...). End-user privacy and security was one of the main concerns of devices designers. Moreover, these systems are becoming more complex and opened to enable industrials to provide different services at the same time on the same device. However, the industrial worries about their data integrity and confidentiality into the devices. Each service provider is in a economic confrontation with others and End-User and service data are a significant resource. In this paper we propose an IoT device architecture based on a small separation kernel and a communication control mechanism to provide a trustworthy environment for each service provider.**

I. Introduction

Internet of things (IoT) devices and cyber-physicals systems (CPS) are becoming ubiquitous in our modern society [1]. IoT is in the interest of many industrials issuer. We can see an increase of devices including multiples hardware and software components in order to provide more complex services [2].

In that kind of environment, privacy, safety and security issues for the end-users are becoming one of the main concern of the industrial. Moreover, in one device, we can identify a certain number of distinct industrial entities which provide software and hardware services for the end-user. IoT devices are becoming opened environments where a lot of On-demand services are deployed by different providers. Due to the economic competition, each industrial service provider can not trust the others services in the device.

By considering the importance of data inner services, we propose in this paper a platform architecture and design based on a separation kernel and a communication mechanism control to guarantee for each service provider a trustworthy environment.

This paper is structured as follow, section II describes a simple security issue in a household IoT. In section III we are going to introduce Pip, a separation kernel which provides a proved memory isolation mechanism. Section IV describes the industrial ecosystem and the responsibility of each actor. In Section V, we can find the ODSI architecture in details and in section VI we can see how to build a real industrial example with Pip and the ODSI platform.

II. Internet of Things security issues

Oravec and co [3] described in their paper the increase of utilization of devices and applications dedicated to Internet of Things, particularly in the case of household appliances and house management. They wished to demonstrated the new issues that should be considered with the expansion of IoT into the house. One of their main concern is the security and privacy integrity of users. They descried scenario about different attacks or other information due to fallible household devices and non-secure softwares.

In that kind of environment, Sivaraman and co [4] described a scenario based on an attack on all house IoT devices via a smartphone application. By using the smartphone and network accesses into the house, they found all available IoT devices. In their paper, they show how it is possible to get control of each device. With this attack, they show that even if there is strong network security mechanisms (like router equipped with firewalls) if any component of the House IoT has an application level issue, the security and privacy is no more guaranteed into the house. This attack works with a smartphone, but we can also imagine a scenario where an another connected device (Webcam, connected TV..) can be an attack vector.

In the past years, security and privacy was one of the main concern of the academic and industrial researchs [1] [5] [2] [6]. They have designed different architecture models in order to provide more trustworthy environments for the IoT industrial and the end-users.

III. Isolation model

A. Memory isolation proto-kernel

PIP [7] is a protokernel: it allows for kernels, ranging from hypervisors to monolithic kernels, to be developed as user mode applications (figure 1). This means that only PIP is executed in kernel mode (i.e., the privileged mode of the hardware)(figure . Indeed, code running in kernel mode has direct access to the whole memory and hardware. It is thus clearly better, from a security point of view, to keep this code as minimal as possible. This stems from the general principle that the trusted computing base (TCB) should be kept minimal. One of the PIP key design is to have a minimal number

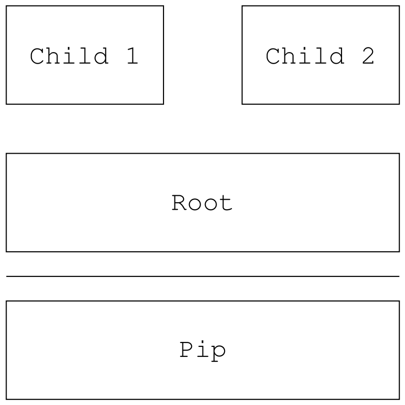

Fig. 1. Pip architecture view

of functionalities while ensuring strong isolation. More precisely, PIP only manages memory isolation and redirection of interruptions to user space code, and has only 10 system calls. Contrary to micro-kernel, it means that components like scheduler, IPC and authorization are not included in the PIP kernel, neither other mechanism available in monolithic kernel like device abstraction layer, file systems, network stack, etc.

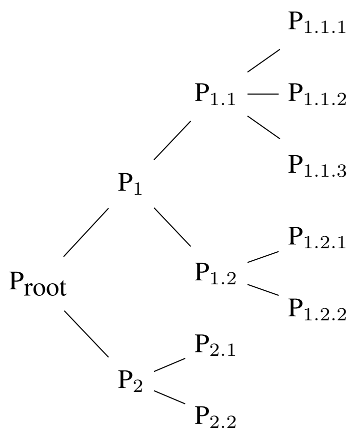

Fig. 2. The view by Pip of the partition tree

PIP proposes a hierarchic memory isolation model (figure 2) to partition the available memory among several memory-limited space called partition. A partition is a set of physical memory pages mapped to virtual address. At start, all the available memory (as defined inside PIP configuration) makes up the ROOT partition. The root partition can include mapped registers from hardware devices, thus providing access to hardware functionalities. Code executed in a parent partition can request (via system calls to PIP) segregation to its memory space: a new child partition is created on which the parent partition can mapped a subset of its own mapped physical memory pages, thus *delegating* part of its memory space to the child partition. This model is recursive: any partition can create a child partition and delegate part of its memory space. While a partition can read and write in the memory of its child partition (we call this property vertical sharing), sibling partitions cannot access each other's memory (we call this property horizontal isolation).

B. Pip memory management

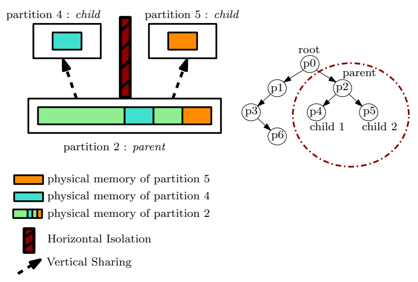

Fig. 3. Isolation model

Pip model ensure formal isolation between partitions. This isolation is guaranteed by two properties :

- **Horizontal isolation** : Two distinct partitions (ex: *child 1* and *child 2* in figure 3) can not share the same memory pages.
- **Vertical sharing** : All pages used by a child are mapped into its parent.

As seen in figure 3, for partition creation, a parent gives the required amount of memory pages from its own pages to its child, and theses pages belong to this new partition.

In order to create partition and manage each child memory, Pip provide a set a function available for each partition.

- **createPartition** : create a new partition.
- **deletePartition** : delete a new partition.
- **addVAddr** : gives a page to a child partition (no more available into the parent).
- **removeVAddr** : reclaims a page from a child partition (no more available into the parent).

C. Interrupt management

To manage interruptions, PIP has two different behaviours depending of the kind of interruption:

- Software interrupts (fault or system calls) are relayed to the parent partition (with the exception of the PIP system calls).
- Hardware interrupts are relayed to the ROOT partition which is in charge to forward it to, for instance, a network card driver.

Furthermore, a parent receives all software interruptions from its children, thus controlling which interactions have its children with the system.

In addition, Pip API provides two function for software interrupt between partitions :

- **dispatch** : send an interrupt signal to a given partition.
- **resume** : restores a previous interrupted partition.

Any partition can send and receive a signal only from its parent, one of its child and from Pip. For example, in figure 3 There is no available signal transmission between *p1* and *p2* or *p3* and *p2*.

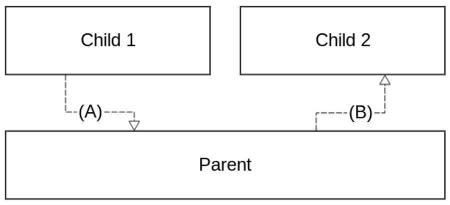

Fig. 4. Simple IPC mechanism

D. IPC mechanism

Pip does not provide IPC between two distinct partitions. However, this mechanism can be implemented in userland, by using software interrupt or Pip API dispatch function. For example (figure 4), if *Child 1* wants to send a message to *Child 2*, it has to send a dispatch to its *parent* (step A). The parent handles this software interrupt and transmit the signal to *Child 2* (step B).

For more complex architecture (like figure 2), this mechanism can be recursively used to send signal between any partitions (ex : P1.1 to P.1.2.2).

IV. INDUSTRIAL ECOSYSTEM

This section describes the industrial ecosystem and the needs for a security architecture based on isolation. First, we identify and define the stakeholders involved in the use and management of an IOT or M2M device. The responsibilities of each actor are then described and finally, the requirements for an isolation-based architecture are listed.

We consider that a domain is an isolated area which belongs to an entity. The notion of domain is precised in section V-A.

We distinguish device ressources and domain ressources. Device ressources correspond to the isolated partitions (domains) and the ressources exposed by software, actuators or sensors drivers installed by the integrator in the owner domain and which are related to the device. Domain ressources correspond to any ressource not related to the device and created within the given domain. An example of device ressource is the firmware or its updater. An example of a domain ressource is a key to create an SSL channel between a remote platform and a given domain.

A. Actors

We identify 7 stakeholders, namely the manufacturer, integrator, maintainer, owner, administrator, service provider and user, which interact with the IOT ot M2M device. One device may be associated to 1 integrator, 1 owner, 1 administrator and several manufacturers, maintainers and service providers. One actor may hold several roles. For example, one actor can be at the same time a manufacturer, an integrator and a maintainer.

1) Manufacturer: Manufacturers provide at least one component of the device, either a physical component with its driver, an operating system or an isolation solution. Manufacturers also provide updates & patches for the component and the driver to the integrator and/or maintainers.

2) Integrator: It issues the device and the isolation solution. It also provides each device an identifier that will subsequently allow it to be identified. It provides either intial secrets or a mechanism to initiate them automatically so that the owner is able to access the device. It may also provide the drivers and software components for the sensors or actuators on the device.

3) Maintainer: It maintains the device on behalf of the owner. It monitors the hardware components status and deploys the patches delivered by the manufacturer or integrator. This role can be assumed by the integrator or a third party.

4) Owner: The device belongs to the Owner who defines an access policy to authorize which other entities are authorized to use or manage the device or domains within the device. The Owner can possess several devices. It receives initial credentials from the manufacturer that allows it to access each device in his fleet.

5) Administrator: It enforces the policy on behalf of the owner for his fleet of devices. If any modification, such as adding a new entity or modifying its granted permissions, is required, it requests a decision from the owner. This role can either be assumed by the owner or delegated to a third party.

6) Service Provider: It delivers a user-friendly service on one or multiple devices using the ressources authorized by the Owner. It installs or activates a service within the device. Muliple Service Providers can co-exist on the same device.

7) User: Users consume the services provided by the entities mentioned above. We can distinguish platform users and technical users. Service users subscribe to the services provided by the Service Providers. Technical users are typically members of the Owner, Manufacturer, Maintainer entities and they will perform authorized administration actions such as creating a new domain, granting rights to a new user, loading a service on the device. In the rest of this article, we focus on platform users.

B. Responsibilities model

The responsibilities described in this section are summarized in table I.

The integrator gathers hardware, drivers and OS from various manufacturers and creates a device with this collection of components. It provides the mechanisms or temporary credentials that allow the owner to personalize the device. The integrator provides a platform in a safe state to the owner and does not keep any access to the owner domain. In particular, the integrator does not manage access control to the drivers.

The owner finalizes the personalization of the device after delivery. In particular, the owner should modify its

TABLE I
RESPONSIBILITIES MATRIX

Responsibilities	Manufacturer	Integrator	Maintainer	Owner	Administrator	Service Provider
Provide device components (hardware & drviers, OS, isolation solution)	X					
Gather components into a device		X				
Use or provide virtualization & multi-tenant compatible components		X				
Provide mechanisms & temporary credentials or credentials creation mechanisms for initial device personalization before delivery to the owner		X				
Provide temporary credentials or credentials creation mechanisms for initial domain personalization to entities after delivery to the owner		X		X	X	
Update regularly the credentials to access its domain			X	X	X	X
Provide components or devices to owner without remote backdoor access	X	X				
Create / Modify / Delete a domain before delivery to owner		X				
Create / Modify / Delete a domain after delivery to owner				X	X	
Configure the access policy of the device ressources				X	X	
Enforce the device ressource access policy				X	X	
Configure its domain ressource access policy			X	X	X	X
Enforce its domain ressource access policy			X	X	X	X
Provide updates & patches of components	X	X				
Follow up, deploy updates of firmware or drivers			X			
Keep the service applications up to date						X

temporary credentials. Owner authorizes the usage of the device resources. If some resources are very sensitive, he provides the access to them through the use of a token which he delivers and verifies. The owner can choose that some resources are accessible freely without the use of a token and thus reducing the security. The owner must not have any control or visibility on the actions performed by other entities unless it touches sensitive functions which the owner has decided to control through token verification.

If the owner delegates his tasks to an administrator, then the administrator can configure which resources need to be accessed with a token and the administrator is responsible for verifying the token and should not be able to control or visualize the actions performed by other actors unless they concern his perimeter of action.

The maintainers keep the firmware, driver or any software in their perimeter up to date. Their role is to keep track of the latest updates and changes provided by manufacturers and deploy them on the field. Both responsibilities (correcting software and deploying updates) are key in the future as the regulators start to take actions on this point. For instance, the European Commission's overall security strategy [8] requires vendors to commit to update their software in the event of newly disclosed vulnerabilities, as part of ”duty of care” principle. Such initiative also exists in the United States with the Internet Of Things (IoT) Cybersecurity Improvement Act of 2017 [9].

Any entity which owns a partition (administrator, service provider, and maintainer) has to authenticate the machine (or user) who is sending the commands to the partition. This entity can choose to not perform and access control verification, thus reducing the security of the system. Any entity which owns a partition (administrator, service provider, and maintainer) is responsible for modifying the temporary credentials of its associated domains.

C. Architecture requirements

1) Domain isolation: Two services which run in two isolated domains should not be able to access the memory of one another or gain information about each other.

2) Owner non-interference: Each entity which owns a domain has the possibility to ensure end-to-end encryption of the commands and data received and sent by the domain. It bears the responsibility for authorizing access to its domain ressources. Although the owner or the administrator have access to a privileged domain, they agree to not use it to observe the data or control the actions performed by other entities. The only exception concerns device ressources which other entities request to the owner domain.

3) Ressource access control: A device ressource access policy under the control of the owner or the administrator is mandatory. For each domain, a domain ressource access policy under the control of the domain owner (maintainer, service provider) is optional.

4) Minimal on-device processing: The processing required to authorize an action in the device must be minimal.

V. Proposed Security model and architecture

This section describes the architecture proposed in the ODSI project. First, we describe the overall architecture. Then, we define the notion of domain in this architecture as well as its components. Finally, we describe the communication and management models.

A. *Definition of a domain*

In this architecture, the device is shared between several parties with different roles (described in section IV-A). As one party can execute various pieces of code in different spaces, the following vocabulary is used:

- A partition is a piece of code managed by one party with its own memory space
- A domain is the union of all partitions managed and developped by one party.

Figure 5 illustrates an example of memory layout with 3 domains. The owner domain is the most privileged and the parent of domains 2 (maintainer) and 3 (service provider). In each domain, subpartitions allow to further isolate components. For example, in the owner domain (Dom0), the drivers (sensor driver, actuator driver, network driver) and the managers which allow to virtualise them are isolated from each others. They are also isolated from the scheduler, the configuration manager and the components Admin Manager, Key Vault, Token & Security Validator and Internal Communicator which enforce the security management model described in this article.

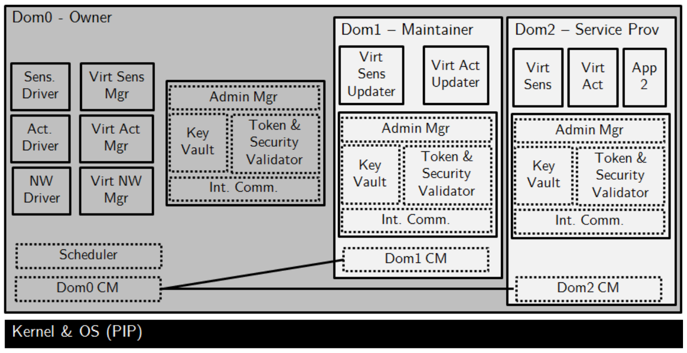

Fig. 5. Proposed architecture - Memory layer view

B. *Components of a domain*

This section describes the role of the components which enforce the proposed security model, namely the Configuration Manager, the Administration Manager, the Token & Security Validator and the Key Vault. The architecture proposed is also based on the use of virtual sensors and actuators.

Figure 6 represents the software layers which correspond to the example used in figure 5. The PIP kernel exposes isolation services to an OS on which domains and partitions are created. The OS layer contains a scheduler which organizes the time and operations of the domains and partitions and each domain configuration manager. The application layer contains the drivers and all the modules of the architecture defined here. The OS and application layers both belong to user land.

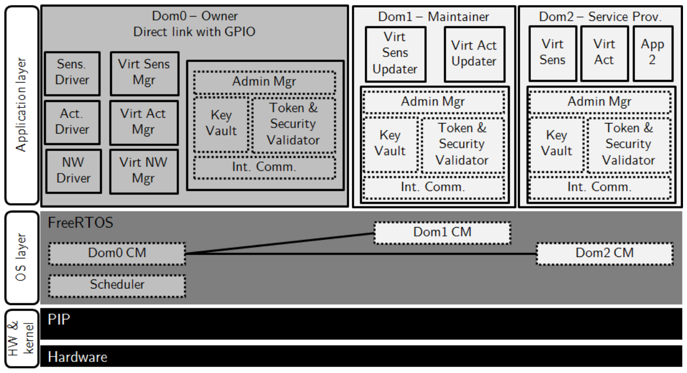

Fig. 6. Proposed architecture - Software layer view

1) Configuration Manager: The Configuration Manager (CM) contains a description of the domain's sub partitions, the associated code and the authorized communication channels between two partitions. It enforces the communications mechanism based on shared memory and described in section V-D.

2) Internal Communicator: The Internal Communicator (Int. Comm.) is the unique entry point of the domain. It verifies incoming messages whether they come from the network or another domain, similarly to a firewall.

3) Virtual sensors or actuators managers: These managers provides synthetic sensor and actuator support to the virtual sensors and actuators over the internal communication module.

4) Virtual sensors or actuators: Only the owner domain can manage the machine-level functions such as the drivers or the physical memory. Each domain contains a virtual sensor or actuator (noted Virt Sens or Virt Act) which are a synthetic sensor or actuator instance. It consumes the hardware ressources provided by the virtual sensor or actuator manager, it requires and forwards a token to the owner domain if needed. This allows the entity which owns the domain to see his domain as an actual device without knowledge of the isolation. The instruction and response are transmitted to and from the real driver using the internal communication mechanism. The virtual sensor or actuator updater is specific and allows the update of the related sensor or actuator driver in the owner domain.

5) Administration Manager: This module exposes the domain's ressources to an external server managed by the entity which owns the domain. It routes the command to read, write, execute the ressource to the expected manager or virtual sensor, actuator. It sends each command received to the Token & Security Validator (T&S Validator).

6) Token & Security Validator: The Token & Security Validator (T&S Validator) validates each device management command against the token provided in the command and optionnally an internal security policy. For example, the security policy defines which ressources require a token verification to be accessed or whether a specific command can be processed accordaccording to the device battery status.

7) Key Vault: This module stores the keys that are used by the domain. In particular, the keys used by the Token & Security Valdiator are stored here. It also provides the functions to add a new key, modify or delete an existing key.

C. Management model

As depicted in figure 7, the model proposed is based on master/agent model. Each domain receives commands from a server which authenticates the origin of the command and authorizes it. The administration manager in the domain is an agent which expects requests from its master, the administration application in the server. The Token & Security Validator validates the token generated according to the access policy on the server side. For simplicity's sake, only the components & modules related to the owner domain are detailed in figure 7.

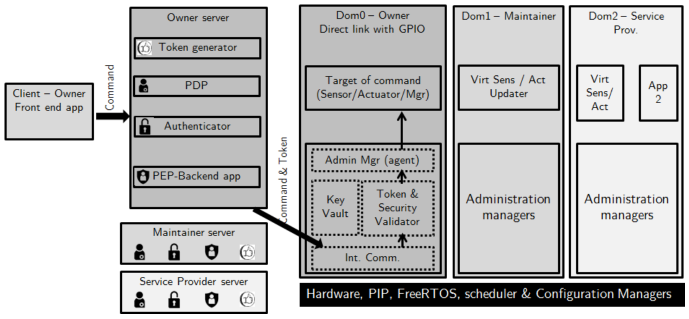

Fig. 7. Proposed architecture - Management model

Each entity is responsible for managing and authorizing the access of its own domain ressources (e.g. domain token keys, application installation) to its users. The owner also manages and authorizes the access to device ressources (e.g. sensor drivers & virtual managers) to its users and to other entities (e.g. maintainer or service provider).

1) Domain ressources management: An example is the modification of the URL from which to retrieve the package which contains a driver update. This parameter is in the perimeter of the maintainer and does not require any modification on the drivers or parameters controled by the owner. Therefore, it is the maintainer's responsiblity to ensure that this modification can be performed only by authenticated & authorized users.

As illustrated in figure 8, first, the client which requests the modification is authenticated. Then, it choses an operation proposed by the administration manager and the access policy evaluates whether this particular user is authorized to perform the selected operation. If yes, the token A is generated and the maintainer sends the command and the token are sent to the maintainer's domain where the command is processed.

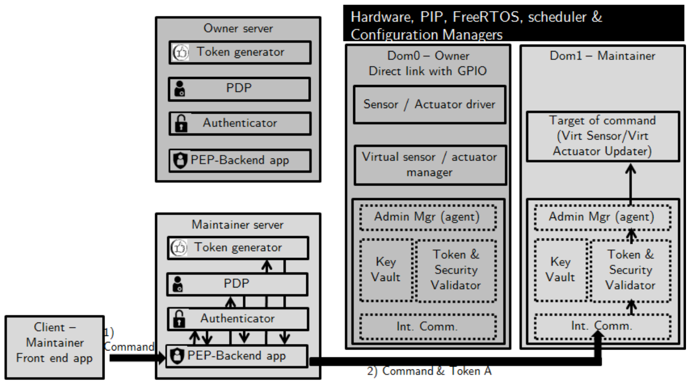
Owner server
Token generator
PDP
Authenticator
PEP-Backend app
Hardware, PIP, FreeRTOS, scheduler & Configuration Managers
Dom0 – Owner Direct link with GPIO
Sensor / Actuator driver
Virtual sensor / actuator manager
Admin Mgr (agent)
Key Vault
Token & Security Validator
Int. Comm.
Dom1 – Maintainer
Target of command (Virt Sensor/Virt Actuator Updater)
Maintainer server
Client – Maintainer Front end app
1) Command
2) Command & Token A

Fig. 8. Management of domain ressources

2) Device ressources management: An example is the download, deployment & installation of a package with a driver udpate. In this case, the maintaier authenticates the user which requests this operation and then requests the owner the authorization to perform it.

Figure 9 illustrates the steps necessary for this operation. First, the client which requests the update is authenticated and authorized by the maintainer server. Then, the maintainer server forwards the request to the owner. The owner authenticates the maintainer server which initiates the request and which will deploy the update package, authorizes the operation according to its own access policy and generates token B. The maintainer server now sends the command, token A and token B to the maintainer's domain. Once token A is verified by the maintainer's Token & Security Validator, the virtual sensor updator sends the command and token B to the Owner domain. If the owner validates token B, the update is authorized.

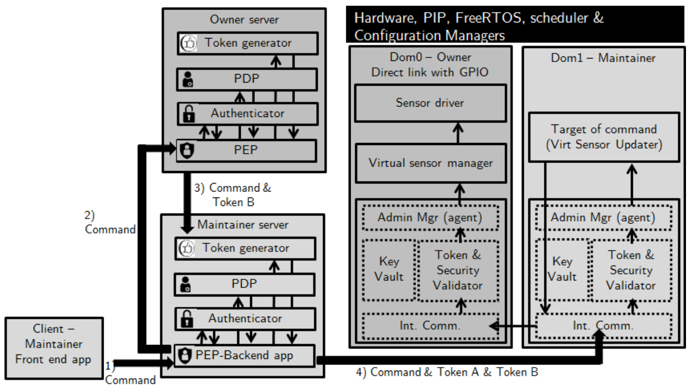
Owner server
Token generator
PDP
Authenticator
PEP
Hardware, PIP, FreeRTOS, scheduler & Configuration Managers
Dom0 – Owner Direct link with GPIO
Sensor driver
Virtual sensor manager
Admin Mgr (agent)
Key Vault
Token & Security Validator
Int. Comm.
Dom1 – Maintainer
Target of command (Virt Sensor Updater)
2) Command
3) Command & Token B
Maintainer server
PEP-Backend app
Client – Maintainer Front end app
1) Command
4) Command & Token A & Token B

Fig. 9. Management of device ressources

3) Token scheme: The token format should contain at minimum a payload and a signature of this payload. Compatible token formats include JWT (JSON Web Token) [10] and CWT (CBOR Web Token) [11].

The token's objectives are to 1) ensure that any command will reach the intended destination, 2) give the rights to request some operations on ressources, 3) prove that the entity initiating the command has been authenticated authorized and protect against replay. To achieve these goals, the token should contain at minimum:

- the ID of the partition to which this token is destined,
- the rights granted,
- a proof ensuring that the token was generated by the token provider,
- a token ID and/or an expiration date.

The proof can either be a digital signature, such as ANS X9.31, ANS X9.62 or PKCS#1 as recommended by NIST [12] or a Keyed-hash Message Authentication Codes (HMACS) compliant NIST recommendations and using a NIST-approved hash algorithm such as SHA-256 [13], [14], depending on the level of security and non-repudiation required. The token ID and expiration date are used to prevent replay. The token ID can be compared to a set of previously used IDs stored in a cache. The size of the cache should be adapted to the memory constraints of the partition and device. The combination of the size of the cache and the expiration date allows achieving best effort replay detection. Therefore, it is strongly recommended to use both elements and to tune them carefully.

D. Communication model

The main requirement for the communication protocol is to allow master/agent communication and the transport of a payload with the command content and an associated token. Protocols such as TCP/IP or CoAP are eligible. LwM2M [15] would require some modification to add a token to the message.

1) Internal communication: Each domain has an internal communication mechanism. It can be implemented by the domain itself, however, it is always based on Pip sending signal mechanism.

2) External communication: For external communication, only one partition is in charge of Network hardware sending and receiving functions. In order to send a external message, each domains has to use the internal communication to discuss with the Network manager, and send it the external message.

VI. Illustration & discussion

A. Pip and ODSI architecture

The ODSI architecture can be build using Pip partition creation mechanism (figure 10). The Owner is the first domain created and it is Pip root partition. For each sensitive component, a dedicated partition is created (Hardware Drivers, Administration manager). The Maintainer domain is also into a partition in order to give the owner the ability to limit the maintainer's rights. Each other services has also to be into partition, because we need to isolate them from any threat or data leak, only the Owner can access to them.

An example of application of the proposed architecture is illustrated in figures 11, 12 and 13. One or several home smart devices are associated to a domain in the gateway, 11. Each domain in the gateway exposes the associated small devices ressources to the outside worlds 12. The devices are managed by only one entity and contains only one domain. Several partitions are created to isolate functions of the device.

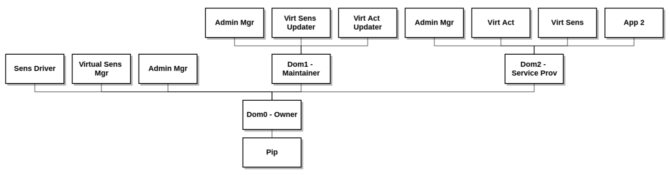

Fig. 10. ODSI platform build with Pip partition tree

As depicted in figure 12, domain 1 is managed by an audiovisual Service Provider which proposes an application to control a connected TV and audio device. Domain 2 is managed by an electronic device management which proposes an application to control a game console, electric outlets and connected lamps. These applications forward the commands and their token to the associated devices. The authentication and token generation can be done by the a service in the domain (e.g. domain 2) or by a service in the cloud (e.g. domain1). The NAT functionality is also isolated from other services in the gateway. Similary to small devices, all applications the three domains perform actions only if they are authenticated and authorized by a token issued by an application in the cloud.

As depicted in figure 13, small devices only have one domain and several isolated functionalities. For example, network drivers which are a potential entry point for attacks are isolated from other drivers and applications in the device.

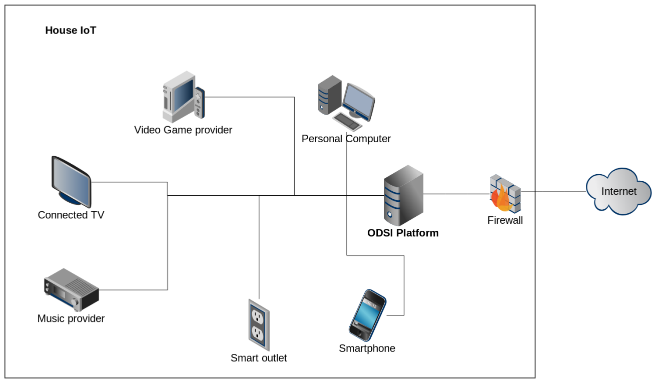

Fig. 11. House IoT installation example

The proposed security architecture avoids the attack described by Sivaraman et.al. [4]. Indeed, the token mechanism is used to authenticate each command sent to the devices or to the NAT configuration service in the router.

Using Pip enhances the protection against this attack by creating isolated partitions in the router and in the devices. Therefore, a successful attack against a domain in the gateway

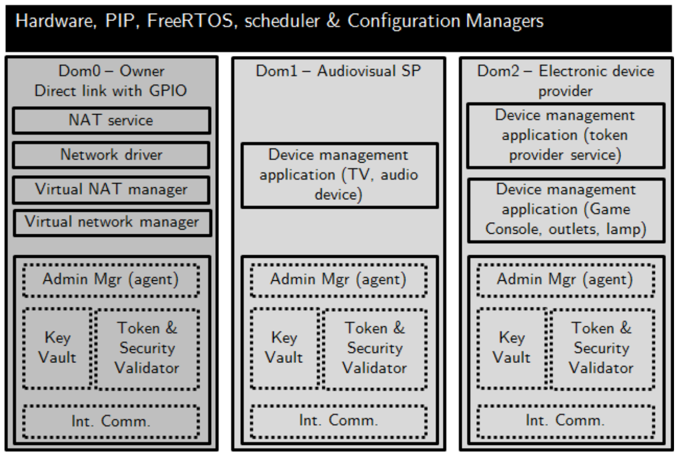
Hardware, PIP, FreeRTOS, scheduler & Configuration Managers
Dom0 – Owner Direct link with GPIO
NAT service
Network driver
Virtual NAT manager
Virtual network manager
Admin Mgr (agent)
Key Vault
Token & Security Validator
Int. Comm.
Dom1 – Audiovisual SP
Device management application (TV, audio device)
Admin Mgr (agent)
Key Vault
Token & Security Validator
Int. Comm.
Dom2 – Electronic device provider
Device management application (token provider service)
Device management application (Game Console, outlets, lamp)
Admin Mgr (agent)
Key Vault
Token & Security Validator
Int. Comm.

Fig. 12. ODSI Gateway

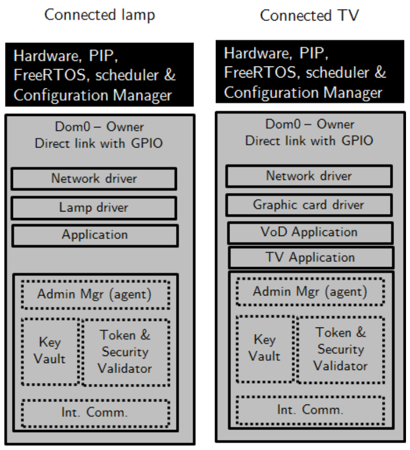
Connected lamp
Connected TV
Hardware, PIP, FreeRTOS, scheduler & Configuration Manager
Dom0 – Owner Direct link with GPIO
Network driver
Lamp driver
Application
Admin Mgr (agent)
Key Vault
Token & Security Validator
Int. Comm.
Hardware, PIP, FreeRTOS, scheduler & Configuration Manager
Dom0 – Owner Direct link with GPIO
Network driver
Graphic card driver
VoD Application
TV Application
Admin Mgr (agent)
Key Vault
Token & Security Validator
Int. Comm.

Fig. 13. Two examples of smart home devices (connected lamp and connected TV)

or a partition in the home smart devices can be contained.

VII. Conclusion

In this paper we have described an architecture based on a separation kernel in order to provide a trustworthy environment for multiple economic competitors into the same IoT device. In order to achieve this objective, we have made a classification of different roles and responsibilities found in each IoT device. On the basis of this taxonomy, we proposed an architecture defined by a set of actors where each one has its capabilities limited, depending on its role into the platform. The capabilities and roles are guaranteed by using the separation kernel Pip. With this mechanism we can guarantee data integrity and confidentiality for each service. The Token mechanism used into the platform guarantee all external and internal communication for the IoT device and avoid any data leak during services communications.

Acknowledgment

This article was done in the scope of the European Celtic-Plus project ODSI.

References

[1] M. Wolf and D. Serpanos. Safety and security in cyber-physical systems and internet-of-things systems. *Proceedings of the IEEE*, 106(1):9–20, Jan 2018.
[2] L. D. Xu, W. He, and S. Li. Internet of things in industries: A survey. *IEEE Transactions on Industrial Informatics*, 10(4):2233–2243, Nov 2014.
[3] Jo Ann Oravec. Kill switches, remote deletion, and intelligent agents: Framing everyday household cybersecurity in the internet of things. *Technology in Society*, 51:189–198, 2017.
[4] Vijay Sivaraman, Dominic Chan, Dylan Earl, and Roksana Boreli. Smart-Phones Attacking Smart-Homes. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks - WiSec '16*, pages 195–200, New York, New York, USA, 2016. ACM Press.
[5] Jorge Bernal Bernabe, Jose Luis Hernandez Ramos, and Antonio F. Skarmeta Gomez. TACIoT: multidimensional trust-aware access control system for the Internet of Things. *Soft Computing*, 20(5):1763–1779, may 2016.
[6] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke. Middleware for internet of things: A survey. *IEEE Internet of Things Journal*, 3(1):70–95, Feb 2016.
[7] Quentin Bergougnoux, Julien Iguchi-Cartigny, and Gilles Grimaud. Pip, un proto-noyau fait pour renforcer la sécurité dans les objets connectés. In *Conférence d'informatique en Parallélisme, Architecture et Système (ComPAS)*, Sophia Antipolis, France, June 2017. Université Sophia Antipolis.
[8] ENISA. Baseline security recommendations for iot in the context of critical information infrastructures. Technical report, ENISA, 2017.
[9] Mark Warner, Gardner, Wyden, and Senate of the United States Daines. Internet of things (iot) cybersecurity improvement act of 2017. https://www.congress.gov/115/bills/s1691/BILLS-115s1691is.pdf, August 2017.
[10] M. Jones, J. Bradley, and N. Sakimura. Json web token (jwt). Technical report, IETF, 2015.
[11] M. Jones, E. Wahlstroem, S. Erdtman, and H. Tschofenigs. Cbor web token (cwt). draft version.
[12] Cameron F.Kerry and Patrick D.CGallagher. Digital signature standard. Technical report, National Institute of Standards and Technology, July 2013.
[13] Carlos Gutierrez and James M.Turner. The keyed-hash message authentication code (hmac). Technical report, National Institute of Standards and Technology, July 2008.
[14] Quynh Dang. Recommendation for applications using approved hash algorithms. Technical report, National Institute of Standards and Technology, 2012.
[15] OMA. Lightweight machine to machine technical specification. Technical report, Open Mobile Alliance, 2017.