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*—Abstract**
TBD

I. Introduction

TBD

mds
March 05, 2018

II. Internet of Things security issues

Oravec and co [1] 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 [2] 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.

III. Isolation model

TBD by Lille

A. Memory isolation proto-kernel

PIP [3] 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 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.

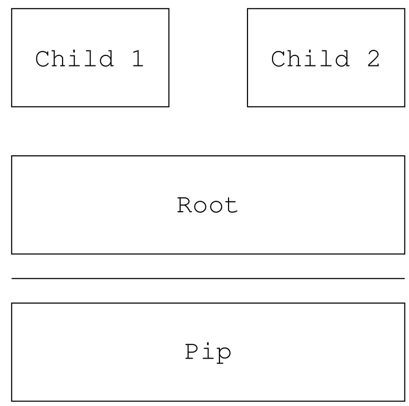

Fig. 1. Pip architecture view

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

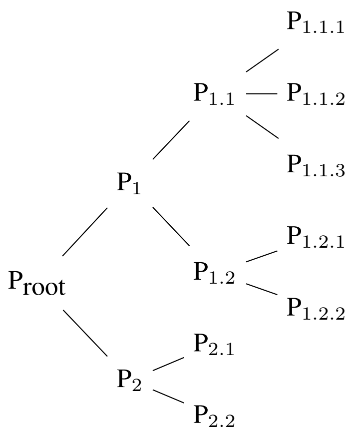

Fig. 2. The view by Pip of the partition tree

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

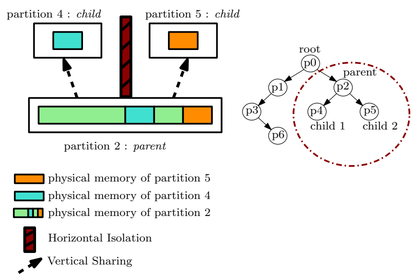

Fig. 3. Isolation model

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*.

D. IPC mechanism

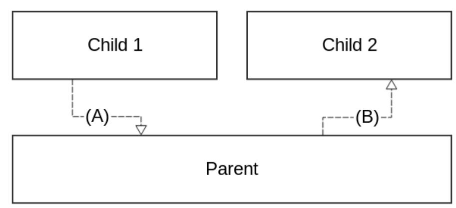

Fig. 4. Simple IPC mechanism

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 aims at describing 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.

In this section, we consider that a domain is an isolated area which belongs to an entity. The notion of domain is further 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 installed in the manufacturer or owner domains or the sensors & actuators drivers. Domain ressources correspond to the ressources that are exposed by a particular domain and created by applications or services in this domain.

A. Actors

We identify 6 stakeholders, namely the manufacturer, maintainer, owner, administrator, service provider and user, which interact with the IOT ot M2M device.

1) Manufacturer: It issues the device and the isolation solution. It also provides each device an identifier that will subsequently allow it to be identified as well as intial secrets that will allow the owner to access the device. It may also provide the drivers and software components for the sensors or actuators on the device.

2) Maintainer: It maintains the device, monitors the hardware components status and applies the patches deliveres by the manufacturer. This role can either be assumed or delegated to a third party by the manufacturer.

3) 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.

4) Administrator: It enforces the policy defined by 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 or delegated to a third party by the owner.

5) 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.

6) 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 manufacturer provides the mechanisms and temporary credentials to personalize the device. At delivery, it provides the keys of the owner domain. The manufacturer is responsible for providing drivers which are compatible with virtualization and usage by multiple entities. For example, to control the position of an armed robot, it is better to literally express the coordinates of the destination. The command "go to (x=10;y=40;z=20)" leads to less confusion than "go to current position + (x=10; y=20; z=30). The manufacturer provides a platform in a safe state to the owner and does not keep any access to the manufacturer domain. In particular, the manufacturer does not manage access control to the drivers.

The Owner finalizes the personalization of the device after delivery by the manufacturer. In particular, the owner should modify its 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 maintainer keeps the firmware, driver or any software in his perimeter up to date. This responsibility is key in the future as the regulators start to take actions on this point. For instance, the European Commission's overall security strategy [4] requires vendors to the 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 [5].

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.

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

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, gain information about each other or interfere with each other's execution.

2) Owner non-interference: Although the owner or the administrator have access to a privileged domain, they must not have any control or visibility on the actions performed by other entities if these actions do not involve device ressources which are under the responsibility of the owner.

TABLE I
RESPONSIBILITIES MATRIX

Requirements	Manufacturer	Maintainer	Owner	Administrator	Service Provider
Provide mechanisms & temporary credentials for initial device personalization before delivery to the owner	X				
Provide drivers compatible with virtualization & multi-tenant usage	X				
Provide temporary credentials for initial domain personalization to entities after delivery to the owner			X	X	
Update regularly the credentials to access the domain under its responsibility	X	X	X	X	X
Provide the device to owner without a remote backdoor access	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		
Enforce the device ressource access policy			X	X	
Configure the domain ressource access policy	X	X	X	X	X
Enforce the domain ressource access policy	X	X	X	X	X
Keep the firmware & drivers up to date	X	X			
Keep the service applications up to date					X

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.

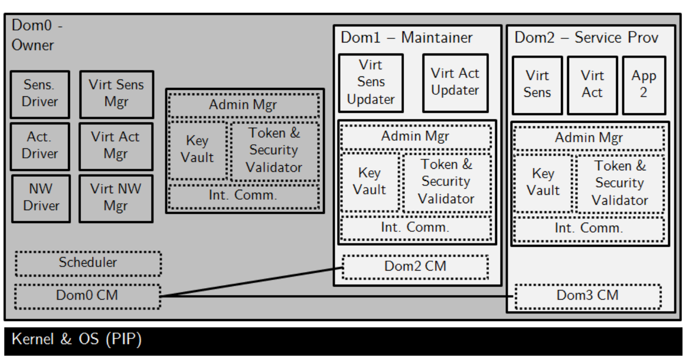

Fig. 5. Proposed architecture - Memory layer view

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 code in different space, the following vocabulary is used:

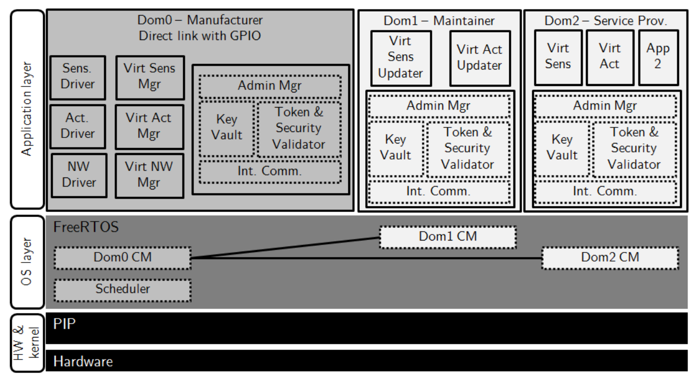

Fig. 6. Proposed architecture - Software layer view

- 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 code by one party.

Do we design ODSI platform & ODSI Mesovisor specifically in this article ?

Domains are implemented in ODSI Mesovisor via the hierarchical memory model. As the memory space of partition can contain the (disjoint) memory space of several sub-partitions (which can themselves contains others sub-sub-partitions), the parent partition controller by one party is the party domain and every enclosed partitions are controlled by the same party.

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.

1) Configuration Manager: The Configuration Manager is part of the

2) Internal Communicator: The Internal Communicator 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: The drivers are provided by the manufacturer of the device and are stored in a separate domain. Each domain contains a virtual sensor or actuator which exposes the functions of the actual sensor or actuator and acts as an interface with it. 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.

4) 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).

5) 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.

6) 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

Image ?

The model proposed is based on a master/agent model. Each domain contains an agent, the administration manager which receives commands from a master, generally an application in a server.

1) 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) [6] and CWT (CBOR Web Token) [7].

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.

In order 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 [8] or a Keyed-hash Message Authentication Codes (HMACS) compliant NIST recommendations and using a NIST-approved hash algorithm such as SHA-256 [9], [10], 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.

2) Domain ressources management:

3) Device ressources management:

D. Communication model

TBD. Section to be described by Lille

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 [11] cannot be used as such because it does not allow adding the token in its payload.

1) Internal communication:

2) External communication:

VI. Pip and ODSI architecture

The aim of ODSI architecture is to provide a safe environment for each service provider. In other words, the aim of ODSI and Pip is how to preserve the trust between each service provider and between the platform and the service provider.

A. Trust between service providers

Each service provider has to trust the platform and the platform has to guarantee to each

TODO: - Confiance entre service, chaque service doit se voir garantir que son tat ne peut łtre altr par un autre service, potentiellement concurent
- Chaque service doit faire confiance la platforme, tant donne qu'elle est dans sa TCB.

B. Need in separation

TODO: - Pip fourni l'isolation recherche par les service pour garantir leur espace de travail/ espace d'adressage - Le mechanisme de token garantie que chaque service ne peut łtre accdd que via l'API qu'il aura expose et sans plus

VII. Illustration & discussion

TBD. Section to show how the isolation model allows to address the industrial needs

From the scenario described above [2], we can use our proposed ODSI security architecture and Pip to avoid this kind attack on any household devices.

We can take the example presented in figure 7. Each device is connected to a service provider hosted by the ODSI platform, and it has not a direct connexion to other devices. We can illustrate this architecture with figure 8. If the Smartphone wants to start a music on the music provider, it has to use the application available into the ODSI platform.

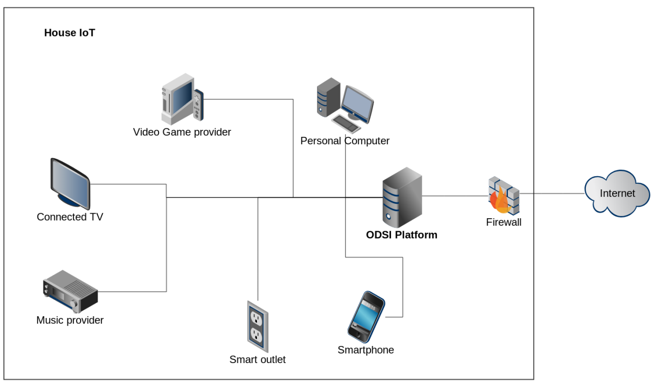

Fig. 7. House IoT installation example

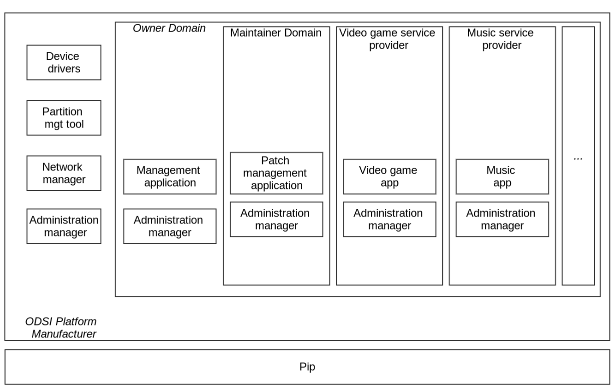

Fig. 8. ODSI Platform

Now, with the same attack scenario as described above, if the music application is not secure and can give access to attacker from outside the house, how the ODSI platform can prevent from any attack propagation ?

As explained, memory isolation provided by Pip has the ability to avoid for music app to access to other domain's memory spaces (Video game app, Owner domain...). The only way for this application to contact other available devices is to use the internal communication mechanism provided by Pip and the ODSI platform. The token mechanism allows only a set of authorized message to a specific service. This list of these commands are provided by the service provider itself. For example, for the music service, it can only authorize three commands : find song/singer, play music, stop music. If any malicious application running into the ODSI platform, it can not send corrupted command to the music service. Moreover, any corrupted service can not access to the hardware due to Pip and it can not use the network devicein order to find more fallible devices.

VIII. Conclusion

TBD. The conclusion goes here. Next steps : implementation and performance evaluation

Acknowledgment

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

References

[1] 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.

[2] 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.

[3] 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.

[4] ENISA. Baseline security recommendations for iot in the context of critical information infrastructures. Technical report, ENISA, 2017.

[5] 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.

[6] M. Jones, J. Bradley, and N. Sakimura. Json web token (jwt). Technical report, IETF, 2015.

[7] M. Jones, E. Wahlstroem, S. Erdtman, and H. Tschofenigs. Cbor web token (cwt). draft version.

[8] Cameron F.Kerry and Patrick D.CGallagher. Digital signature standard. Technical report, National Institute of Standards and Technology, July 2013.

[9] Carlos Gutierrez and James M.Turner. The keyed-hash message authentication code (hmac). Technical report, National Institute of Standards and Technology, July 2008.

[10] Quynh Dang. Recommendation for applications using approved hash algorithms. Technical report, National Institute of Standards and Technology, 2012.

[11] OMA. Lightweight machine to machine technical specification. Technical report, Open Mobile Alliance, 2017.