Internet Research Task Force
Internet-Draft
Intended status: Informational
Expires: March 12, 2021

H. Baba
The University of Tokyo
Y. Ishida
Japan Network Enabler Corporation
T. Amatsu
Tokyo Electric Power Company, Inc.
K. Kunitake
BroadBand Tower, Inc.
K. Maeda
Individual Contributor

September 8, 2020

Problems in and among industries for the prompt realization of IoT and safety considerations
draft-baba-iot-problems-09

Abstract

This document contains opinions gathered from enterprises engaging in the IoT business as stated in the preceding version hereof, and also examines the possibilities of new social problems in the IoT era. Recognition of the importance of information security has grown in step with the rising use of the Internet. Closer examination reveals that the IoT era may see a new direct physical threat to users. For instance, the situation at a smart house may lead it to judge that the owner has only temporarily stepped out, causing it to unlock the front door, which in turn makes it easier for thieves to enter. These kinds of scenarios may occur without identity fraud, hacking, and other means of compromising information security. Therefore, for the purpose of this document, this issue shall be referred to as "IoT Safety" to distinguish it from Information Security.

We believe that it is necessary to deepen our understanding of these new IoT-related threats through discussion and ensure there are measures to address these threats in the future. At the same time, we must also coordinate these measures with the solutions to the problems described in the previous version of this document.

Status of This Memo

Table of Contents

1. Introduction

Many activities are progressing in various fields, such as the proposal of standards for creating an IoT world. There are also many reports that analyze and predict the benefits that IoT can bring to the economy and society. These developments remind us of the end of the 20th century, when the effect and impact of the Internet was actively debated.

The authors tried using the following approach to clarify the issues for the prompt realization of IoT. First, the players were conveniently divided into two groups: ICT industry players and Things industry players. Next, we met major players in the ICT industry and Things industry and asked about the challenges they faced and the challenges the other side faced in creating IoT.

The ICT industry players mentioned here include communication carriers, ICT equipment vendors, the Internet service providers, application vendors, and software houses. The Things industry players include home and housing equipment manufacturers, infrastructure providers such as railways companies and power companies, and manufacturers of home appliances such as air conditioners and refrigerators, which are also the ICT users.

This paper is principally a summary of the meetings results, and a presentation of the micro case studies about the challenges for realizing IoT services. It is not an overview of the IoT world or a macro-proposal intended to promote the benefits of IoT.

In addition, the revised version includes an examination of the possibilities of new direct physical threats in the IoT era that have not yet been seen. These threats should affect the safety of our bodies, lives, and "things," which includes property. For this reason, this issue shall be referred to as "IoT Safety" to distinguish it from Information Security for the purpose of this document.

For the past few years, we got new findings through COMMA House, the experimental smart house owned by The University of Tokyo. Therefore, we will add new topics to the next version.

2. Technical Challenges

2.1. Safety, Security and Privacy

2.1.1. Challenges in protecting lives and property from IoT-related threats (IoT Safety)

The introduction of IoT may generate threats to "Safety" through the actual operation of mechanical devices, in addition to the Information Security problems discussed in Section 2.1.2 below. For example, the spread of applications for visualizing electric power consumption allows for mischief in device operation without the use of identity fraud or hacking. In addition, there is the potential for problems to arise in the normal operation of individual devices that are not caused by abnormal current or voltage, another troubling aspect of the introduction of IoT. These issues cannot be resolved with ordinary information security measures for Network Layer 4 or lower. In another case, a command to an IoT device is proper by itself, but it may conflict with the other commands or its environmental status. Therefore, the authors consider it necessary to have a system for interpreting the details of operations of many appliances and preventing operations according to the necessity in Layer 7 (what the authors tentatively call "Sekisyo".)

These threats are categorized into three types: threat to physical safety; the threat of the failure or destruction of equipment and property; and the threat of impeding the proper performance of equipment. The following section introduces examples of the different threats.

2.1.1.1. Safety of body/life

Information on things such as the use of faucets and housing equipment, the locking of the front doors and windows, and the state of electric power consumption based on the smart meter is used by smart houses to regulate homes. This information is used to determine whether anyone is at home, and the electronic lock of the front door and windows is unlocked and a notice of absence is issued to a thief.

2.1.1.2. Safety of equipment

Air conditioners and other equipment that normally are not expected to be frequently started or stopped each a day can be caused to break down by repeatedly turning them on and off as many as hundreds of times a day.

2.1.1.3. Proper performance of equipment

Water heaters containing a hot well can be caused to operate erratically. This is done by frequently transmitting signals from the mischief application instead of operation panel to tell the water heater that only 10% of the normal amount of hot water is needed, leaving the water heater perpetually low on water.

2.1.2. Information Security

We have confirmed two viewpoints regarding the information security of services using IoT equipment and devices. The first is tangible information security involving the critical infrastructure. The second concerns the information security of individuals and homes.

In regards to information security involving the critical infrastructure, the basic policy in the past was to stay physically disconnected from an external network, such as the Internet, to ensure information security. However, because of the advance in the systems from proprietary communication protocols to open IP protocols to detect symptoms of problems and to remotely maintain a large number of facilities spread over a wide area, connecting to an external network will become unavoidable to achieve various goals. In addition, it is clear that isolated networks are also subject to the same kind of risks, even though it is not directly connected to the outside. There is no major difference in the information security risks because isolated networks are already the target of international cyber terrorism, with internal crimes and targeted attacks occurring more frequently. Based on these reasons, the ICT security of the social infrastructure requires an extremely high level of information security.

Looking at the information security of micro units, such as individuals and homes, the improved convenience provided by the introduction of IoT will lead to greater risks. For example, there is a product available for connecting the entrance door to the network. In ICT security technology, increasing the key length of the encryption makes it much harder to break. But even if the latest information security technology is used when it is installed, the information security technology will become obsolete and even pose a risk about halfway through the twenty- to thirty-year lifetime of the entrance door. As has been explained in other items, the ICT sense of time is completely different from that of Things.

2.1.3. Privacy in acquiring data

The problem of privacy in handling acquired data is a huge challenge for companies promoting IoT. In addition, the ownership of this data poses yet another challenge.

For example, railway companies have installed many cameras for station security and for marketing beverage vending machines. This creates problems for personal identification and privacy. At the present time, the companies are processing the images in real time and do not store the images to avoid the problems.

Another huge challenge is the ownership of data. Up until now, there has been a divided debate on whether data belonged to the company or to the users. Likewise, the relationship inside a small user group is also extremely diverse and complicated. One specific example is of a company that had obtained permission from the head of the household to use the data when it carried out an HEMS trial. Later on, the spouse of the head of the household disagreed and as a result permission to use the data was withdrawn.

2.2. Challenges posed by data acquisition, data distribution, data management and data quantity

2.2.1. Traffic patterns

The manner in which data is acquired from and distributed to IoT equipment/devices differs immensely from the traffic patterns of the present Internet. The present form of the Internet focuses on distributing information, and its systems focus on effectively delivering contents to the users. On the other hand, routinely or temporarily sending or receiving data through a huge number of various sensors and devices presents a very different kind of Internet traffic. However, questions such as how much traffic will come from what kind of Things, and how will they superimpose each other have not been sufficiently studied. There is no concrete explanation about the backbone design and operation of traffic, and there have been many cases in which the unclear specifications for IoT traffic made the design difficult on the communication company side. There are many challenges related to the set up and management of IoT equipment. We have heard from the construction companies that the configuration of IoT equipment with a large number of sensors involves a lot of hard work.

2.2.2. Acquired mass data

It is necessary to develop a management method to reuse acquired data safely and effectively. Even now, there are occasional instances of the theft and leakage of social data (such as IDs) that can be used to identify individuals. In the IoT era, there will be mass data that can lead to Things, and the Things in turn will lead to individuals. There are IoT industry players who do not invest as much in ICT systems as government agencies and large companies do, and thus a management system to safely and effectively reuse the acquired data needs to be developed. The laws and regulations related to ID management differ vastly by country and region. These issues related to society and individuals are largely affected by differences in common sense, and therefore need to be localized.

2.2.3. Explosive increase and diversity of data

In the future IoT era, there are concerns about the explosive increase in data quantity and the diversity of data sent from sensors and IoT equipment. On the other hand, M2M communication does not require mass data like images, and an extraordinary increase in traffic will be unlikely despite the increase in the number of sensors.

If data is sent from all Things, there will be an infinite number of different kinds of data. In addition, with the present form of Internet traffic, data is received by people, and most of it consists of video or image downloads. The download traffic is several times greater than that of the upload traffic. If there is a tremendous increase in the use of IoT, such as M2M communication, the difference between upload and download traffic will probably not be that much. It might be necessary to fundamentally review the network and in particular the last mile characteristics. The importance of this issue is not yet widely recognized.

2.3. Mapping of the physical world and the virtual world

2.3.1. Physically handling acquired data

The acquired data simply represents certain kinds of digital value, and it is important to uncover the meaning of this data. As described previously, configuration of IoT equipment, such as the large number of installed sensors, requires a lot of hard work. An even greater amount of effort will be needed to determine the meaning of the data and connect it to the physical world.

In energy management experiments, data is mapped manually. This is a time consuming process, and one that is prone to human error. Cases that rely on the use of human hands require the configuration of automated setting systems to reduce labor, costs, and human errors to introduce IoT

2.3.2. Data calibration

Another important thing is calibration. This involves properly linking the data sent from Things to the Things concerned, and correctly indicating the operating conditions.

It may be necessary to have a tool to treat this problem concerning continuation of operation and the one pertaining to introduction of IoT described previously as a package.

2.4. Product lifetime, generation management, and the cost of equipment updates

2.4.1. Product lifetime

The life of most ICT equipment is about 5 years or less, while the life of IoT equipment and devices is at least 10 years. There is a clear gap between these two types of equipment.

In the example of the entrance door connected to the network mentioned earlier, the door is often used for about twenty to thirty years after installed. If is connected to a network, the communication technology and communication service will most likely have undergone numerous generation changes in that twenty- to thirty-year time span. This presents a large gap between the ICT industry and the Things industry.

A solution to this problem that was reached during the meeting with the housing equipment manufacturers is that with the automatic control of multiple shutters in a building, the portion between the controller and the multiple shutters, the so-called mature technology, can be placed under the control of the shutter manufacturers, while the controller connected to the network will deal with the generation changes of the communication service.

2.4.2. Introducing IoT equipment into commodity equipment

It costs a lot to make the many different types of commodity equipment popular around the world usable as IoT equipment and devices. There are two ways to change commodity equipment into IoT equipment. One way is to convert it to IoT compatible equipment. The other way involves adding devices to commodity equipment. There are costs in both cases, and it will take a long time to introduce IoT unless different incentives are offered to help to overcome the burden of cost.

2.5. Too many related standards and the speed of standardization

2.5.1. Too many related standards

There are many standards related to IoT equipment and devices. There are multiple standards, technologies and services for communication technology, such as Bluetooth, Wi-Fi, NFC, and LTE, and it is difficult to choose which to apply.

The Things industry players do not always have the communication technology professionals needed for IoT. In the meeting, we learned that many companies were uncertain and hesitant about fields outside their own area of expertise. On the other hand, technological competition will improve quality as well as the level of completion, and thus will be beneficial for users.

In the future, a consulting business for clarifying ICT technology for the Things industry players may emerge. If there is a system that can interconnect multiple standards, it will accelerate the Things industry to enter IoT

2.5.2. Speed of standardization

The concept of product life in ICT industry is completely different from that of the Things industry, and as a result the concept of standardization also varies greatly. Before standardization occurs in the ICT industry, many different proposals are made, from which the best is selected. The final decision often changes, and products have to be updated in order to follow the changes in standards. But in the Things industry, the standards have to remain unchanged for as long as possible because of the long product lifetimes. Therefore, it takes a long time to determine when a particular standard has become mature. When the Things industry goes to implement a standard from the ICT industry, it feels that the standard is incredibly fluid and seemingly undecided. Furthermore, the standardization process of the two industries is very different, and making it difficult to work on the other side when trying to determine a standard.

2.6. Interoperability, fault isolation, and total quality assurance

2.6.1. Interoperability

The verification of interoperability poses a major challenge because of the configuration used by multi-vendors. In addition to interoperability between equipment, the ability to ensure backward compatibility is also important for bringing about the IoT world.

If these capabilities cannot be provided, it will be very difficult to create an IoT world in which past products can function.

2.6.2. Fault isolation

The method for fault isolation that may occur presents another challenge.

Many PC users have experienced various kinds of problems. When their PC experiences a problem, they have to isolate the faults by themselves, with no one available to lend a helping hand.

In the IoT world, these issues become more difficult and complicated. For example, a smart home is equipped with air conditioners, kitchen supplies, and doors connected to the Internet. A problem that occurs in the smart home poses a much more serious problem to end users than an e-mail failure or problem with a PC.

If users are left to isolate the fault on their own, they may not know which manufacturer they contact for repairs if they are unable to isolate the fault on their own, or the manufacturer may refuse to perform repairs because they fall outside the scope of their responsibility. As can be seen, the issue is an important challenge that will determine whether the B2C specific IoT world can be established.

2.6.3. Quality assurance

The quality assurance of individual pieces of IoT equipment does not guarantee the total quality of IoT. Since IoT involves connecting multiple Things and communication, it is natural to assume that the total service quality will depend on the quality of the IoT equipment and devices, which can sometimes become bottleneck. However, users are not aware of this.

As was mentioned previously in Section 2.6 issues that are not directly related to the quality of an individual component can be important factors in determining the quality of the service. In this way, the quality of IoT is not decided by each individual Thing, but needs to be considered as a service spread across the network.

2.7. Product design policy

2.7.1. Changes in design policy

The design policy has to be changed from placing emphasis on the high functionality of a single product to stressing the singular function of individual products as well as how they work in coordination with other products. For many years, the Things industry has focused on producing high functionality products with added value. But in the IoT era, the implicit assumption is to confine Things to their basic function and enhance the level of coordination between Things, rather than focusing on the added value. Simplified Things must be able to be controlled with an external application that can also be used by the Things of cross manufacturers.

Given this situation, the Things industry faces the challenge of adopting a completely different policy. During the meeting with the manufacturing industries, we could sense their difficulty in understanding and recognizing the need to change the policy.

2.8. Various technology restrictions within actual usage

2.8.1. Using radio waves

There are many cases that have provided us with insight about issues related to the use of radio waves in IoT (such as the wave traveling range and whether or not it travels further than stated in assumptions available). The suppliers or providers who configure IoT are not always wave communication technology experts. People who are unfamiliar with radio waves seem to think that waves travel from antenna to antenna in a straight line, and that they can be blocked by obstacles. As a result, they often ask questions about how many meters radio waves can travel or whether radio waves can actually travel. Few people understand the fact that the emitted radio waves are reflected from various locations and are superimposed at the reception point where they are received, or that depending on how waves are reflected a change in the reception signal intensity, called fading, may occur. The lack of engineers who can advise on specialized matters such as these poses a major obstacle.

2.8.2. Batteries

The power capacity and lifetime of batteries represent another set of challenges similar in nature to the issue of radio waves traveling distance. There are questions such as the difference between the real and catalog specifications, as well as factors that affect the battery power capacity. The IoT providers, who are also users of IoT, have to solve these issues, while these are difficult problems even for experts.

2.8.3. Wiring

The incredible amount of wiring and its complexity (power lines and communication lines) pose major challenges. The complexity of wiring- such as the large number of sensors and equipment, the power lines that drive them, and the communication lines that connect them to the network for acquiring information-is to the point that people doing IoT installation work will start wishing for a wire harness. In addition, the installation of cables and electric work are often done by different engineers. This make the issue even more complicated.

2.8.4. Being open

A single company alone cannot make all the commodities for IoT. The IoT world needs to be open, and this can only be achieved with the cooperation of many different industries. Up until now, companies in the Things industry have developed products in a closed loop process, seeking to capture users with their company’s own products. For this reason, they lack an open design concept of interoperability. Today, an entirely new design concept is needed to design products that can interconnect with the products of other companies.

3. Non-technical Challenges

3.1. Changing the product paradigm

3.1.1. Ecosystems

While the goal of setting up IoT is to generate new value, it may actually lead to the destruction of the ecosystems in which industries operate. In the IoT era, the traditional vertically integrated way of producing Things in manufacturing industries will consume too much time and cost. This approach also makes it difficult to incorporate the ideas of other cultures. The need for paradigm shift is easy to understand, but difficult to implement. Promoting this shift will pose a management challenge that requires a considerable amount of skill and effort to overcome.

3.1.2. Coordination and significant changes in strategy

It will become necessary to run businesses jointly with new partners, as well as cooperate and work in coordination with other industries and competitors. This issue-even when it is fully understood-will be very difficult to address and put into practice.

We have seen instances in which only a limited amount of information was given when parties exchanged opinions. There have also been instances in which communication was difficult because of differences in terminology and culture.

3.1.3. Competition with existing industries

The issue of competition with existing industries often arises when attempts are made to change or reform a business model change or reform. This issue can also be viewed as the reorganization of industries, rather than competition between existing industries. However, this realignment of industries is difficult to move forward in the absence of supervisors.

3.2. Benefits

3.2.1. Rising costs and monetization

Introducing IoT within products will cause costs to go up, and yet the benefits it provides are unclear. There is no specific killer application available, and the number of users will not rise immediately. Therefore, finding a way to make the business profitable will be very difficult. This issue is especially difficult for businesses and products that rely on cost reductions to deliver low prices that make them competitive.

3.3. Information security and privacy of social systems

3.3.1. Classification of ownership, location, and the usage of data

There are many questions regarding the wide variety of data gathered from IoT equipment, including questions related to ownership, storage location, and the authorization to grant a license to use data. These need to be addressed so that the system and equipment can be accepted by society.

For example, if a company installs a door in a house that gathers data on the opening and closing of the door, questions about the data will arise. Does it belong to the users or the company? Can another company use this data?

3.4. Disclosure of data

3.4.1. Side effects and malicious use potentially caused by the disclosure of data

For example, it has been shown that the electricity smart meter can lead to burglary because it shows when electricity is used and not used, providing an indication of the time when no one is home. This particular example demonstrates the importance of ensuring information security and privacy.

3.5. Preparing social support

3.5.1. Regulations

Systems of laws and regulations are important for ensuring the safety of the conventional products, but they can also be a barrier for innovation.

IoT can be affected by laws and regulations at home and abroad, and can also be influenced by regulations that extend across multiple countries. Regulatory authorities need to monitor IoT carefully and adjust the regulations and laws they oversee in a way that does not negatively impact the global competition environment.

3.5.2. Corporate social responsibility

In addition to pursuing profit, companies that promote IoT also need to improve the benefits offered to users and society

3.5.3. Customization for individual customers

There is an ongoing shift in demand away from general products to customized products for individual customers. This could also be viewed as a shift away from manufacturing businesses to service businesses. IoT will play an important role in this shift.

Instead of manufacturing Things through mass production, it will be easier to customize a product by moving some of the functions to an application. Likewise, the manufacturing business also needs to move forward with the previously mentioned paradigm shift in order to achieve customization

3.5.4. IoT literacy of the users

Because Things are connected to the network, apps will need to be created. Some of these will serve as the interface with which people interact with IoT.

In the IoT era of the future, users will need to possess a certain amount of knowledge about IoT apps

3.5.5. Individual vs. family

The issue of whether the data of Things in the house belongs to the family or the individual will largely affect data analysis and the handling of privacy.

As was mentioned in Section 2.1.2, the spouse could later object to the head of the household granting authorization to use data.

4. Security Considerations

Meetings with the players in various IoT fields provided insight into information security issues. These issues are described in the following sections.

- Section 2.1.2 Physical damper of devices

- Section 2.1.2 Product lifetime and encryption strength

For details, please see the corresponding text.

5. Privacy Considerations

Similarly, issues regarding privacy are described in the following sections.

- Section 2.1.2, Section 3.3.1 Ownership of the data
- Section 3.4.1 Data disclosure and malicious use
- Section 3.5.5 Individual vs. family

For details, please see the corresponding text.

6. IANA Considerations

This document has no actions for IANA.

7. Acknowledgments

We would like to thank the foundation the promotion of industrial science and its RC-88 member companies for their cooperation.

And we also appreciate Ministry of Internal Affairs and Communications.

Authors' Addresses

Hiroyuki Baba
The University of Tokyo
Institute of Industrial Science
4-6-1 Komaba
Meguro-ku, Tokyo 153-8505
Japan

Email: hbaba@iis.u-tokyo.ac.jp

Yoshiki Ishida
Japan Network Enabler Corporation
7F S-GATE Akasaka-Sanno.
1-8-1 Akasaka
Minato-ku, Tokyo 107-0052
Japan

Email: ishida@jpne.co.jp

Takayuki Amatsu
Tokyo Electric Power Company, Inc.
1-1-3 Uchisaiwai-cho
Chiyoda-ku, Tokyo 100-8560
Japan

Email: amatsu.t@tepco.co.jp

Koichi Kunitake
BroadBand Tower, Inc.
Hibiya Parkfront.
2-1-6, Uchisaiwai-cho
Chiyoda-ku, Tokyo 100-0011
Japan

Email: kokunitake@bbtower.co.jp

Kaoru Maeda
Individual Contributor
Japan

Email: kaorumaeda.ml@gmail.com

Thing-to-Thing Research Group
Internet-Draft
Intended status: Informational
Expires: April 21, 2016

M. Burgess
Independent Researcher
H. Wildfeuer
Cisco Systems

October 19, 2015

Federated Multi-Tenant Service Architecture for an Internet of Things
draft-burgess-promise-iot-arch-00

Abstract

This draft describes architectural recommendations for an Internet of Things scenario, based on tried and tested principles from infrastructure science. We describe a functional service architecture that may be applied in the manner of a platform, from the smallest scale to the largest scale, using vendor agnostic principles. The current draft is rooted in the principles of Promise Theory[Bergstra1] and voluntary cooperation.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The scenario we call the Internet of Things (IoT) is an inflection point in the development of information local and global infrastructure. The facilitation of a platform for the next generation of global commerce presents a challenge of both technological and human dimensions. This is a challenge that spans every layer of the software and networking stacks, but can be described in general terms without the need to specific implementations. That is our goal in this draft. Only a few new ideas are needed to synthesize this infrastructure, however several old technology practices must be deprecated for scaling and security considerations.

A platform for society must be vendor agnostic at its root, and must leave ample space for vendor specific creativity on top. What distinguishes IoT from past scenarios is the prolific contact surface it will expose to the physical world, embedding devices pervasively in our close environments, and touching every part of human life. At the time of writing, IoT has barely begun to emerge in domestic and industrial settings; however, choices we make now could help or hinder the development of an adequate platform over the coming decades. The proposed architecture not only scales up to large numbers, it also scales down to small devices of low capability; from the largest installations to the smallest, and from the tiniest amounts of data, to vast data-stores collected by scientific computing at the limits of possibility.

2. Requirements and Promises Language

The term "PROMISE", "PROMISES" in this document are to be interpreted as described in Promise Theory [Bergstra1]

When used, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. Definitions and concepts

IP endpoint A hardware or software agent that is IP addressable, via a TCP/IP capable interface.

Static endpoint A hardware or software agent with an IP address (prefix and subnet) that is fixed over the timescale of application service interactions.

Mobile endpoint A hardware or software agent whose IP address location can change on the timescale of application service interactions.

Application server/service Any agent that promises to respond to requests, from external parties, and perform services of any kind, on a timescale that we may call the application service timescale.

Multi-tenant application service A collection of agents housed as tenants within a single host device, each offering different services, with potentially different timescales.

Client application An agent that consumes data from an application service, requested either by imposed query or by promised schedule.

Standalone Thing (FFD) A full function device (FFD)[OneM2M], with an IP address, that can present its own service gateway or interface to the IP network.

Peripheral Thing (RFD) A reduced function device (RFD)[OneM2M], with no IP address, that attaches to a host gateway device as a peripheral, over an arbitrary network (USB, PCIe, CANbus, Profibus, ModBus, wireless sensor network, etc). Devices are addressable, only through the gateway service. This includes portmapped devices.

Embedded network Any network (IP or non-IP) that is non-IP routed, i.e. contained within a host endpoint as part of a black box, e.g. isolated NAT, device bus, serial channels.

Transducer An agent that consumes a service from another agent, and provides a new service based on the consumed service, e.g. a router, encrypter, compressor, etc.

Trust A unilateral policy assessment of one agent by another, concerning its reliability in honouring promises. Trust is not necessarily a transitive property.

Partial connectivity A device is said to have partial connectivity if it is unavailable for intervals of time, e.g. due to loss of connectivity, mobility, or power napping.

4. Device interconnection

All devices are assumed to live in a partially connected environment. They MUST be fault tolerant to loss of communications, both with other agents in the course of providing application services, and with trusted sources of information. A minimum interdependency design may be recommended to facilitate this.

For a nascent Internet of Things, the focus is naturally drawn to the specialized leaf devices, where data may be produced or consumed. However, these are only half the picture. ‘Thing’ devices, by design, also communicate with online services deployed ‘higher up’, or ‘Northbound’ in the system, to offload analysis and decision-making. Their physical capabilites thus place them into two broad categories:

Standalone devices These are assumed to connect by an IP addressable underlay network. Connectivity is assumed end-to-end, without reference to tunnels or software defined overlays. Routing is assumed to be provided end-to-end, and fully decoupled from the registration of devices. Segregation and firewalling of certain network regions may be included in network design, but will not be considered here.

Peripherals These include bare sensors and actuators, which do not possess sufficient onboard resources or software interfaces, may attach to hosting standalone devices that act as a gateway and IP endpoint on their behalf.

Transducers These pass-through devices transformers, converters, encapsulation services, etc

```
+------------------+
| FFD / Standalone |--> IP Endpoint
+------------------+

+------------------+
| RFD / Peripheral |--+
+------------------+  |      +------------------+
                      +------| FFD / Standalone |--> IP Endpoint
+------------------+  |      +------------------+
| RFD / Peripheral |--+
+------------------+
```

Devices may be standalone (FFD), with service interfaces, or hosted peripherals (RFD), where data are exposed through service interfaces from other buses, e.g. USB, CANbus, MODbus, Profibus, etc.

Figure 1

Standalone devices are full stack devices that provide data oriented services to data clients

Stand-alone devices and transducers can vary considerably in their processing, memory and connectivity resources and constraints. This architecture assumes a minimum resource level at the stand-alone device, but the device must support ‘full stack’ implementations. In practice, this implies that they contain an embedded OS (e.g. Linux), and are capable of running an agent providing secure service and connectivity interfaces.

5. Federation of agency

Centralization of intent or control is not practical in environments with the density of devices and overlapping concerns exhibited by a pervasive Internet of Things.

5.1. Ownership

Device ownership is an important issue in a multi-tenant consumer environment. While some devices will be centrally managed by providers, many devices in an Internet of Things will be personally owned, and would not be managed completely by centralized services. Devices may thus be managed by:

Their owners This applies in particular to personal consumer electronics, phones, cars, domestic appliances, etc, where users need to retain trusted ownership of their personal belongings.

A service provider This applies to managed services, factory machinery, fleet vehicles, set-top boxes placed in the home, power controllers, etc, where users do not need to interact with the devices on a management level, but there is an advantage to placing a device as a local presence in a smart environment.

5.2. Tenancy and separation of concerns

Federation of intent, aka multi-tenancy or diversity, all point to the need for Special Interest Groups (SIG) or work groups. Workspaces are places that are set aside for a particular purpose, that act as umbrellas for special interest groups. For this, we introduce the notion of workspaces.

Federation can be along a number of lines:

o Geographic partitioning (location)

o Separation of timescales (fast and slow)

o By special interest group (functional)

See sections below for further information.

5.3. Proximity of services to Things

Although devices will be separate from the agencies processing their sensory data, and feeding their guidance systems (policies and renderers), it is impractical to transport data over long distances between leaf devices and ‘cloud’ services. The logical outcome is therefore a decentralization of the cloud itself to insert converged resources close to the data sources themselves. To scale such a distribution, the data services will naturally associate with private workspaces, which bound the scope of data generated by Things.

6. Workspaces

Workspaces may be thought of as a modernization of the domain concept. Domains are typically linked to directory services (DNS, Active Directory, LDAP etc). The demands of multi-tenant environments, where shared resources and separate business-processes mix and compete, make these older services less than optimal, though not inherently flawed.

Workspaces are related to the more familiar notion of namespaces in information technology; however, namespaces refer only to a priority in name-referencing of objects, without underlying resource segmentation. Workspaces MUST support multi-tenant separation of concerns within a hosted space. Today, workspace facilities are commonly offered by user logins on computing devices, and quasi-workspace-like facilities are offered by virtual private networks, and VLANs, etc, in networking.

For a collaborative Internet of Things, where interests span many issues from manufacturer interests, to personal ownership, functional responsibility, and security, the technologies for inter-group collaboration must be modernized to support logical, authenticated segmentation, shared directory information, as well as private naming, across converged resources: compute, network, and storage.

1. Workspaces may or may not be private, but they must be self-contained and separable, in the manner of namespaces.

2. Workspaces may or may not be associated with multiple tenants; but they are associated with multiple issues.

3. They represent a context for human activity, or separation of concerns, e.g. some human activities might be modelled as workspaces include: the home, a children’s playground, a squash court, an office, a shop, a factory floor, building, district, city, emergency channel frequency, hot and cold water pipes, dining room, drinks cabinet, etc.

Ubiquitous computing (the Internet of Things) is all about how networked devices support a wider variety of workspaces. As the density of device resources (compute, storage, sensors, actuators) in a workplace or home environment increases, isolation of regions, and mapping of resources to responsible or interested parties become more difficult problems, both to implement and to understand.

A detailed description of workspaces will be given separately [WORKSPC].

7. Generic Promise-Oriented Architecture

A promise-oriented architecture is described implicitly in [DSOM2005] and [Bergstra1]. It lays out a generic ‘bottom up’ management concept, in which devices each have the responsibility for their own state and roles. It resembles Service Oriented Architecture (SOA) superficially, without reference to specific technologies, implementations or protocols, and relates to the modern notion of microservices [MicroS]

By formulating architecture from the bottom up, one can easily account for multi-contextual concerns, from developer concerns about realtime software updates (Continuous Delivery and DevOps etc), to operational service scaling, governance, and security, in a way that top-down schemes cannot easily achieve.

7.1. Control

A promise-oriented architecture communicates (e.g. intent and data) by authenticated publish-subscribe (aka "pull") methods, for security and predictability. Thing devices MUST not accept control commands imposed upon them by "push" methods, as this exposes a security risk and may lead to inconclusive results if there are uncoordinated pushes. In the vernacular usage of "control plane" and "data plane", control is asserted through agreed service level policies, and data are exchanged within services to carry out functions.

Every standalone device operates autonomously, with direct policy input from its owner, without being managed from an external collective. Similarly, any standalone device can give up that autonomy to a trusted manager, offering policy updates as a service.

7.2. Services

All devices provide services in varying degrees of sophistication. Peripheral devices serve data or actuators to host devices, and standalone devices expose functions to one another as software services. Each server plays a role to be composed into the wider system.

Services may be used both for basic infrastructure support, and for driving user applications. No limitations need be stated about applications. Each fully functional, standalone device is free to host any application services. The result is superficially similar to the Service Oriented Architecture [SOA], but without reference to a specific technology or methodology. In modern parlance, the model is an example of microservices [MicroS].

Data services are also best implemented as with pull methods, for resource-light scalability and security, but extremely limited application devices might initially struggle to support this mode.

7.3. Promises

The basic atom of bottom-up policy is a promise. Each promise consists of three things:

A ‘promiser’ i.e. a resource that will affect a change by keeping its promise to the system, e.g. a file, a process, a transaction, a measurement, device settings, etc.

A description body i.e. the desired-outcome that is achieved when the promise is kept. This SHOULD be implemented in a convergent, idempotent manner [CFENGINE], [CONVERGE].

A context in which the promise applies, based on time, location, type and group membership of the devices referred to in the model.

7.4. Agents and their promises

In a promise architecture, every device is contextually evaluated and integrated from the bottom up, according to the promises is keeps, e.g. the services it provides, its behaviours and properties, etc. Thus every device is modelled by its individual degree of agency to act as a proxy for human intent (policy).

Standalone devices are assumed to be equipped with policy-keeping software agents. Peripheral devices, such as sensors or actuators, are assumed to be integral parts of the standalone devices, and hence maintainable by the their software agents.

No system must push changes or data to such agents ad hoc, without a documented promise to accept; thereafter, ‘fault tolerance’ demands that we reject the word ‘must’ from most descriptions, and replace it with ‘promise of best effort’, as to reply on perfect behaviour leads to brittle systems with unrealistic expectations. For human safety in a rapidly expanding sphere of human involvement, the only ‘must’ is for each agent to be stable and self-correcting, subject to the guidance of policy.

7.5. Standard promises

The following characteristics describe the cooperation between agents:

1. Standalone devices promise to bootstrap to some trusted bootservice, i.e. register to one or more workspaces.

2. Standalone devices promise to refuse direct commands imposed from network peers (as mentioned above).

3. Policy consists of a collection of promises that apply in labelled contexts, each of which describes a unique desired end-state.

4. Promises are kept in a convergent manner, so that all promise-keeping actions lead to the desired end-state, no matter what the initial state of the device.

5. Agents that live on every device have drivers/renderers and make all changes without remote communication.

7.6. Contextual policy-based adaptation

Each policy agent promises to maintain a context evaluator that computes a set of classifying ‘tags’ or ‘labels’ that characterize the state of the agent. This is updated every time the agent verifies policy, as its state may change as a result of repairs. These may be used as conditionals for distributed policy-based decision-making.

Contextual labels characterize the device, its environment, and its location and time. The labels can then be used in policy to make certain promises apply only in specific contexts.

When promises, within a policy, are tagged by issue or context, agents can select those that apply to its condition, within a larger trust relationship implied by policy sourcing. This simplifies logic and promotes stability, as evidenced by experience with software agents [CFENGINE].

7.7. Workspace maintenance

The following characteristics describe compatible policy update processes:

1. Devices subscribe to policy from a trusted source, download changes to the policy model when they can, and cache it locally so that it is always available.

2. Local agents implement cached policy, without any dependence on remote communication, and in a fault tolerant fashion. The failure to keep one promise should have minimal impact on the ability to keep others.

3. By verifying promises continuously, the agent that runs on each standalone device will know (or be able to calculate) its operational context, and can decide which promises are needed from the policy model, and whether or not to keep the promises. This scales O(1), i.e. without bottleneck.

4. Each promise that documents and intended outcome of the system is verified and measured in the process, providing immediate and statistical feedback to policy designers about the success of the policy in describing a stable desired outcome.

7.8. Change of policy (system intent)

Policy change can be initiated from within a workspace, subject to a defined quality assurance, or fit-for-purpose review. Thus change of infrastructure may be instigated from the bottom-up also, as a self-service request.

1. Human operators (owners or managers) decide on a policy model for all devices in an organization or policy group. This may be informed by the feedback about the success rate of previously kept promises.

2. The changes are edited into a model, which consists of a collection of promises that should be kept by all resources on all devices.

3. Changes are checked and tested before publishing.

4. Once changes are approved, they are published by a policy service for download at the convenience of the standalone device.

7.9. Separation of concerns versus timescales

infrastructure stability is supported by a separation of systems into agencies that act in alignment with specific, separable timescales. Separation of fast and slow timescales avoids tight coupling and associated complex behaviours and should be considered a priority for maintaining safe, stable systems for human dependence.

Systems scale along two broad lines, which a promise-oriented architecture helps to resolve:

Dynamical scaling Workload timescales concern the quantitative activity of the system: how fast requests are handled, how quickly service is delivered, and promises are kept.

Semantic (functional) scaling Semantics are normally the concern of software engineers and system designers. This facilitates functional understanding. It is a form of human interface or knowledge management. It is sometimes at odds with the needs of dynamical scaling.

Changes to semantics should generally be slow compared to the workload related dynamical activity, in order to maintain functional stability. Cooperative design of workspaces may observe this principle to foster functional stability and workload efficiency.

7.10. Device roles per workspace or region

A number of functional roles are required to maintain a service lifecycle in a distributed environment. Making these roles self-managed within each workspace is how one scales the diversity of human intent and concerns. Roles are defined by the kinds of promises kept by devices:

Bootstrap server To provide trusted need-to-know data and local contacts so that clients can begin working within a policy domain.

Bootstrap client To accept essential directory information on trust in order to join a local policy domain.

Policy server To deliver current policy from an authorized source, appropriate for each client (tenancy terms) from its global perspective

Policy client To subscribe to the policy, selectively, depending on context from its local perspective.

Data server data server (aka ‘‘Thing’’) To offer a catalogue of data streams to different tenants This includes sensors, actuators.

Data Client To subscribe to the policy, selectively, depending on context from its local perspective.

Identity server Manufacturer User Description service is promised by all Things providing a URI that points to a description of the device, its serial number characteristics, service details etc.

Identity client Identity clients promise to make use of data schemas and encodings involved in the interpretation of data pertaining to the device.

```
                         "Control data"                            "Application
data"
 +--------------------------------------------------------------+
 |+------------------+ +------------------+ +----------------- +| +-------------
----+
 || Bootstrap server | | Policy server    | | Directory server || |  Data client
(s) |
 |+------------------+ +------------------+ +----------------- +| +-------------
----+
 +--------|---------------------|----------------------|--------+          |
          |                     |                      |                   |
          +----------------+    |                      |                   |
                           |    |                      |                   |
     +------------------+  |    |                      |                   |
     | FFD / Standalone |  |    |                      |                   |
     |  Bootstrap client|--+    |                      |                   |
     |  Policy client   |-------+                      |                   |
     |  Directory server|------------------------------+                   |
     |  Data client     |--------------------------------------------------+
     +------------------+

        "Thing(s)"
```

The roles in each collaborative workspace. Devices at the bottom of the figure typically coordinate through workspace services hosted in the "cloud" or any nearby compute resource. Efficiency suggests avoiding long data paths, instead moving computational resources closer to data collection points.

Figure 2

Bootstrapping new devices into a workspace represents the beginning of a device lifecycle. Devices must begin with the location of a known bootstrap server. Devices must also promise to advertise their nature and capabilities, called ‘identification’. This may include Manufacturer Usage Description (MUD) identifiers [MUD].

7.11. Connectivity and Network Policy

So far, much as been said on how the application devices provide services via promises, and how system intent can be described and orchestrated via policy. There is also a connectivity (transport) fabric for these devices that operates on a set of promises that underly the described service framework, i.e. the network. Each network endpoint can be seen as providing its own set of promises that are used by other network elements to deliver routing and switching capabilities [PromiseNet].

Intent driven networking is becoming more relevant as Software Defined Networking (SDN) deployments proliferate. In the described IoT architecture, service policies that describe the IoT system intent can be used as an input to derive partial network policies (e.g. Group Based Policy or some other model-based approach), with modulation by other data discovered from bootstrapping, etc. The figure below illustrates the relationship between the service and network layer policies for IoT.

```
                   +--------------------+
                   | IoT Service Policy |
                   +--------------------+
                             |
     +---------------------+ | +--------------------+
     | Topology / Location | | |   Orchestration    |
     |                     +-+-+
     |   Bootstrap data    | | | Organization policy|
     +---------------------+ | +--------------------+
                             |
                            \|/
                             v
                 +--------------------+
                 |   IoT SDN policy   |
                 +--------------------+
```

Service policy could be partially rendered as an SDN baseline for simplifying dependency management.

Figure 3

8. Characteristics

The architecture, described in this draft, enables densely clustered IT resources to form arbitrary self-service communities that span local or wide area networks. This is decouples a logical patchwork of segments on top of a plain end-to-end IP network. By basing on principles of fault-tolerance, including publish-subscribe dissemination semantics, this may be scaled, without bottleneck, by only the well-known methods currently employed by the World Wide Web.

IPv6 and successors will play a key role in recapturing network simplicity from the many workarounds that have been stacked on top of IPv4 and its limitations. However, currently missing are adequate directory services to support a transparent workgroup concept. The present Internet architecture is still geared principally towards a crudely shared single-tenant, top-down management model, with authority at the top. Top down methods require the leaf domains to be exposed to attack from high up in the network. However, shrink-wrapping workspace boundaries closer around their private resources, their management could be simplified, speeded up, and become less exposed.

9. Summary and Outlook

The issues discussed and laid out in this draft address key issues of scalability, fault tolerance, separation of concerns, and federation of intent within networked information systems. The platform is a synthesis of well-known techniques, and is deliberately aligned with the needs of agile commercial spaces, as well as large industrial distributions, and small domestic needs. We purposely leave open vendor specific concerns, which can easily fit into the described architecture, on top of this common set of principles.

10. Acknowledgments

We are grateful for helpful conversations with K. Burns, M. Dvorkin, D. Maluf, and E. Lear.

11. Security Considerations

With a pervasive contact surface onto both the Internet and the real world, security is obvious a major concern. Experience with pervasive frameworks like [CFENGINE], as well as theoretical studies of pull-based architectures, suggest that the promise-oriented pull-only architecture can reduce the exposure to denial of service attacks and data-based overflow attacks, by rejecting all external data sent without invitation. Moreover, the tie-in between service and network policy reduces the likelihood of errors in policy across the layers.

Workspaces can play a role too here, as a shrink-wrapping of service scope around minimal set of endpoints, thus reducing the logical contact surface for data communications, and publishing information purely on a need-to-know basis. We take is for granted that workspace data are encrypted with workspace authorized credentials.

12. Normative References

[Bergstra1]
Bergstra, J. and M. Burgess, "Promise Theory: Principles and Applications", 2013.

[CFENGINE]
Burgess, M., "A site configuration engine, Computing Systems", 1995.

[CONVERGE]
Burgess, M., "Configurable immunity model of evolving configuration management, Science of Computer Programming", 2004.

[DSOM2005]
Burgess, M., "An Approach to Understanding Policy Based on Autonomy and Voluntary Cooperation, Lecture Notes in Computer Science", 2005.

[MicroS] Richardson, C., "Pattern: Microservices Architecture", 2014.

[MUD] Lear, E., "Manufacturer Usage Description", 2015.

[OneM2M] OneM2M, , "Standards for M2M and the Internet of Things", 2015.

[PromiseNet]
Borrill, P., Burgess, M., Craw, T., and M. Dvorkin, "A Promise Theory of Networking", 2014.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[SOA] Open Group, , "SOA Reference Architecture Technical Standard : Basic Concepts", 2016.

[WORKSPC] Burgess, M., Dvorkin, M., and K. Burns, "Self-Service Workspaces for Federated IT Infrastructure", 2016.

Authors’ Addresses

Mark Burgess
Independent Researcher
Oslo
Norway

Herb Wildfeuer
Cisco Systems
San Jose
USA

Email: hwildfeu@cisco.com

CoRE
Internet-Draft
Intended status: Informational
Expires: March 15, 2014

O. Garcia-Morchon
S. Kumar
Philips Research
S. Keoh
University of Glasgow
R. Hummen
RWTH Aachen
R. Struik
Struik Consultancy

September 11, 2013

Security Considerations in the IP-based Internet of Things
draft-garcia-core-security-06

Abstract

A direct interpretation of the Internet of Things concept refers to the usage of standard Internet protocols to allow for human-to-thing or thing-to-thing communication. Although the security needs are well-recognized, it is still not fully clear how existing IP-based security protocols can be applied to this new setting. This Internet-Draft first provides an overview of security architecture, its deployment model and general security needs in the context of the lifecycle of a thing. Then, it presents challenges and requirements for the successful roll-out of new applications and usage of standard IP-based security protocols when applied to get a functional Internet of Things.

Status of this Memo

Copyright Notice

Table of Contents

1. Conventions and Terminology Used in this Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in "Key words for use in RFCs to Indicate Requirement Levels" [RFC2119].

2. Introduction

The Internet of Things (IoT) denotes the interconnection of highly heterogeneous networked entities and networks following a number of communication patterns such as: human-to-human (H2H), human-to-thing (H2T), thing-to-thing (T2T), or thing-to-things (T2Ts). The term IoT was first coined by the Auto-ID center [AUTO-ID] in 1999. Since then, the development of the underlying concepts has ever increased its pace. Nowadays, the IoT presents a strong focus of research with various initiatives working on the (re)design, application, and usage of standard Internet technology in the IoT.

The introduction of IPv6 and web services as fundamental building blocks for IoT applications [RFC6568] promises to bring a number of basic advantages including: (i) a homogeneous protocol ecosystem that allows simple integration with Internet hosts; (ii) simplified development of very different appliances; (iii) an unified interface for applications, removing the need for application-level proxies. Such features greatly simplify the deployment of the envisioned scenarios ranging from building automation to production environments to personal area networks, in which very different things such as a temperature sensor, a luminaire, or an RFID tag might interact with each other, with a human carrying a smart phone, or with backend services.

This Internet Draft presents an overview of the security aspects of the envisioned all-IP architecture as well as of the lifecycle of an IoT device, a thing, within this architecture. In particular, we review the most pressing aspects and functionalities that are required for a secure all-IP solution.

With this, this Internet-Draft pursues several goals. First, we aim at presenting a comprehensive view of the interactions and relationships between an IoT application and security. Second, we aim at describing challenges for a secure IoT in the specific context of the lifecycle of a resource-constrained device. The final goal of this draft is to discuss the next steps towards a secure IoT.

The rest of the Internet-Draft is organized as follows. Section 3 depicts the lifecycle of a thing and gives general definitions for the main security aspects within the IoT domain. In Section 4, we review existing protocols and work done in the area of security for wireless sensor networks. Section 5 identifies general challenges and needs for an IoT security protocol design and discusses existing protocols and protocol proposals against the identified requirements. Section 6 proposes a number of illustrative security suites describing how different applications involve distinct security needs. Section 7 includes final remarks and conclusions.

3. The Thing Lifecycle and Architectural Considerations

We consider the installation of a Building Automation and Control (BAC) system to illustrate the lifecycle of a thing in a BAC scenario. A BAC system consists of a network of interconnected nodes that perform various functions in the domains of HVAC (Heating, Ventilating, and Air Conditioning), lighting, safety etc. The nodes vary in functionality and a majority of them represent resource constrained devices such as sensors and luminaries. Some devices may also be battery operated or battery-less nodes, demanding for a focus on low energy consumption and on sleeping devices.

In our example, the life of a thing starts when it is manufactured. Due to the different application areas (i.e., HVAC, lighting, safety) nodes are tailored to a specific task. It is therefore unlikely that one single manufacturer will create all nodes in a building. Hence, interoperability as well as trust bootstrapping between nodes of different vendors is important. The thing is later installed and commissioned within a network by an installer during the bootstrapping phase. Specifically, the device identity and the secret keys used during normal operation are provided to the device during this phase. Different subcontractors may install different IoT devices for different purposes. Furthermore, the installation and bootstrapping procedures may not be a defined event but may stretch over an extended period of time. After being bootstrapped, the device and the system of things are in operational mode and run the functions of the BAC system. During this operational phase, the device is under the control of the system owner. For devices with lifetimes spanning several years, occasional maintenance cycles may be required. During each maintenance phase, the software on the device can be upgraded or applications running on the device can be reconfigured. The maintenance tasks can thereby be performed either locally or from a backend system. Depending on the operational changes of the device, it may be required to re-bootstrap at the end of a maintenance cycle. The device continues to loop through the operational phase and the eventual maintenance phase until the device is decommissioned at the end of its lifecycle. However, the end-of-life of a device does not necessarily mean that it is defective but rather denotes a need to replace and upgrade the network to next-generation devices in order to provide additional functionality. Therefore the device can be removed and re-commissioned to be used in a different network under a different owner by starting the lifecycle over again. Figure 1 shows the generic lifecycle of a thing. This generic lifecycle is also applicable for IoT scenarios other than BAC systems.

At present, BAC systems use legacy building control standards such as BACNet [BACNET] or DALI [DALI] with independent networks for each subsystem (HVAC, lighting, etc.). However, this separation of functionality adds further complexity and costs to the configuration and maintenance of the different networks within the same building. As a result, more recent building control networks employ IP-based standards allowing seamless control over the various nodes with a single management system. While allowing for easier integration, this shift towards IP-based standards results in new requirements regarding the implementation of IP security protocols on constrained devices and the bootstrapping of security keys for devices across multiple manufacturers.

```
            _Manufactured           _SW update          _Decommissioned
   /                       /                   /
   |   _Installed          |   _ Application   |   _Removed &
   |  /                    |  / reconfigured   |  /  replaced
   |  |   _Commissioned    |  |                |  |
   |  |  /                 |  |                |  |   _Reownership &
   |  |  |    _Application |  |   _Application |  |  / recommissioned
   |  |  |   /   running   |  |  / running     |  |  |
   |  |  |   |             |  |  |             |  |  |             \\
   +##+##+###+#############+##+##+#############+##+##+##############>>>
       \/  \______________/ \/  \_____________/ \___/         time //
       /           /         \          \          \
   Bootstrapping  /      Maintenance &   \     Maintenance &
                 /      re-bootstrapping  \   re-bootstrapping
           Operational                Operational
```

The lifecycle of a thing in the Internet of Things.

Figure 1

3.1. Threat Analysis

This section explores the security threats and vulnerabilities of a network of things in the IoTs. Security threats have been analyzed in related IP protocols including HTTPS [RFC2818], 6LoWPAN [RFC4919], ANCP [RFC5713], DNS security threats [RFC3833], SIP [RFC3261], IPv6

ND [RFC3756], and PANA [RFC4016]. Nonetheless, the challenge is about their impacts on scenarios of the IoTs. In this section, we specifically discuss the threats that could compromise an individual thing, or network as a whole, with regard to different phases in the thing’s lifecycle. Note that these set of threats might go beyond the scope of Internet protocols but we gather them here for the sake of completeness.

1 Cloning of things: During the manufacturing process of a thing, an untrusted manufacturer can easily clone the physical characteristics, firmware/software, or security configuration of the thing. Subsequently, such a cloned thing may be sold at a cheaper price in the market, and yet be still able to function normally, as a genuine thing. For example, two cloned devices can still be associated and work with each other. In the worst case scenario, a cloned device can be used to control a genuine device. One should note here, that an untrusted manufacturer may also change functionality of the cloned thing, resulting in degraded functionality with respect to the genuine thing (thereby, inflicting potential reputational risk to the original thing manufacturer). Moreover, it can implement additional functionality with the cloned thing, such as a backdoor.

2 Malicious substitution of things: During the installation of a thing, a genuine thing may be substituted with a similar variant of lower quality without being detected. The main motivation may be cost savings, where the installation of lower-quality things (e.g., non-certified products) may significantly reduce the installation and operational costs. The installers can subsequently resell the genuine things in order to gain further financial benefits. Another motivation may be to inflict reputational damage on a competitor’s offerings.

3 Eavesdropping attack: During the commissioning of a thing into a network, it may be susceptible to eavesdropping, especially if operational keying materials, security parameters, or configuration settings, are exchanged in clear using a wireless medium. After obtaining the keying material, the attacker might be able to recover the secret keys established between the communicating entities (e.g., H2T, T2Ts, or Thing to the backend management system), thereby compromising the authenticity and confidentiality of the communication channel, as well as the authenticity of commands and other traffic exchanged over this communication channel. When the network is in operation, T2T communication may be eavesdropped upon if the communication channel is not sufficiently protected or in the event of session key compromise due to a long period of usage without key renewal or updates.

4 Man-in-the-middle attack: The commissioning phase may also be vulnerable to man-in-the-middle attacks, e.g., when keying material between communicating entities is exchanged in the clear and the security of the key establishment protocol depends on the tacit assumption that no third party is able to eavesdrop on or sit in between the two communicating entities during the execution of this protocol. Additionally, device authentication or device authorization may be nontrivial, or may need support of a human decision process, since things usually do not have a priori knowledge about each other and can, therefore, not always be able to differentiate friends and foes via completely automated mechanisms. Thus, even if the key establishment protocol provides cryptographic device authentication, this knowledge on device identities may still need complementing with a human-assisted authorization step (thereby, presenting a weak link and offering the potential of man-in-the-middle attacks this way).

5 Firmware Replacement attack: When a thing is in operation or maintenance phase, its firmware or software may be updated to allow for new functionality or new features. An attacker may be able to exploit such a firmware upgrade by replacing the thing’s with malicious software, thereby influencing the operational behaviour of the thing. For example, an attacker could add a piece of malicious code to the firmware that will cause it to periodically report the energy usage of the lamp to a data repository for analysis.

6 Extraction of security parameters: A thing deployed in the ambient environment (such as sensors, actuators, etc.) is usually physically unprotected and could easily be captured by an attacker. Such an attacker may then attempt to extract security information such as keys (e.g., device’s key, private-key, group key) from this thing or try and re-program it to serve his needs. If a group key is used and compromised this way, the whole network may be compromised as well. Compromise of a thing’s unique key has less security impact, since only the communication channels of this particular thing in question are compromised. Here, one should caution that compromise of the communication channel may also compromise all data communicated over this channel. In particular, one has to be weary of, e.g., compromise of group keys communicated over this channel (thus, leading to transitive exposure ripple effects).

7 Routing attack: As highlighted in [ID-Daniel], routing information in IoT can be spoofed, altered, or replayed, in order to create routing loops, attract/repel network traffic, extend/ shorten source routes, etc. Other relevant routing attacks

include 1) Sinkhole attack (or blackhole attack), where an attacker declares himself to have a high-quality route/path to the base station, thus allowing him to do anything to all packets passing through it. 2) Selective forwarding, where an attacker may selectively forward packets or simply drop a packet. 3) Wormhole attack, where an attacker may record packets at one location in the network and tunnel them to another location, thereby influencing perceived network behaviour and potentially distorting statistics, thus greatly impacting the functionality of routing. 4) Sybil attack, whereby an attacker presents multiple identities to other things in the network.

8   Privacy threat: The tracking of a thing’s location and usage may pose a privacy risk to its users. An attacker can infer information based on the information gathered about individual things, thus deducing behavioural patterns of the user of interest to him. Such information can subsequently be sold to interested parties for marketing purposes and targeted advertizing.

9   Denial-of-Service attack: Typically, things have tight memory and limited computation, they are thus vulnerable to resource exhaustion attack. Attackers can continuously send requests to be processed by specific things so as to deplete their resources. This is especially dangerous in the IoTs since an attacker might be located in the backend and target resource-constrained devices in an LLN. Additionally, DoS attack can be launched by physically jamming the communication channel, thus breaking down the T2T communication channel. Network availability can also be disrupted by flooding the network with a large number of packets.

The following table summarizes the security threats we identified above and the potential point of vulnerabilities at different layers of the communication stack. We also include related RFCs that include a threat model that might apply to the IoTs.

	Manufacturing	Installation/ Commissioning	Operation
Thing's Model	Device Cloning	Substitution	Privacy threat Extraction of security params
Application Layer		RFC2818 RFC4016	RFC2818, Firmware replacement
Transport Layer		Eavesdropping & Man-in-the-middle attack RFC4919, RFC5713 RFC3833, RFC3756	Eavesdropping Man-in-the-middle
Network Layer			RFC4919,DoS attack Routing attack RFC3833
Physical Layer			DoS attack

The security threat analysis

Figure 2

3.2. Security Aspects

The term security subsumes a wide range of different concepts. In the first place, it refers to the basic provision of security services including confidentiality, authentication, integrity, authorization, non-repudiation, and availability, and some augmented services, such as duplicate detection and detection of stale packets (timeliness). These security services can be implemented by a combination of cryptographic mechanisms, such as block ciphers, hash functions, or signature algorithms, and non-cryptographic mechanisms, which implement authorization and other security policy enforcement aspects. For each of the cryptographic mechanisms, a solid key management infrastructure is fundamental to handling the required cryptographic keys, whereas for security policy enforcement, one needs to properly codify authorizations as a function of device roles and a security policy engine that implements these authorization checks and that can implement changes hereto throughout the system's lifecycle.

In the context of the IoT, however, the security must not only focus on the required security services, but also how these are realized in the overall system and how the security functionalities are executed.

To this end, we use the following terminology to analyze and classify security aspects in the IoT:

1 The security architecture refers to the system elements involved in the management of the security relationships between things and the way these security interactions are handled (e.g., centralized or distributed) during the lifecycle of a thing.

2 The security model of a node describes how the security parameters, processes, and applications are managed in a thing. This includes aspects such as process separation, secure storage of keying materials, etc.

3 Security bootstrapping denotes the process by which a thing securely joins the IoT at a given location and point in time. Bootstrapping includes the authentication and authorization of a device as well as the transfer of security parameters allowing for its trusted operation in a given network.

4 Network security describes the mechanisms applied within a network to ensure trusted operation of the IoT. Specifically, it prevents attackers from endangering or modifying the expected operation of networked things. Network security can include a number of mechanisms ranging from secure routing to data link layer and network layer security.

5 Application security guarantees that only trusted instances of an application running in the IoT can communicate with each other, while illegitimate instances cannot interfere.

```
              ..........................
              :           +-----------+:
              :       *+*>|Application|*****
              :       *|  +-----------+:   *
              :       *|  +-----------+:   *
              :       *|->| Transport |:   *
              :    * _*|  +-----------+:   *
              :    *|  |  +-----------+:   *
              :    *|  |->|  Network  |:   *
              :    *|  |  +-----------+:   *
              :    *|  |  +-----------+:   *    *** Bootstrapping
              :    *|  +->|     L2    |:   *    ~~~ Application Security
              :    *|     +-----------+:   *
              :+--------+              :   *
              :|Security| Configuration:   *
              :|Service |   Entity     :   *
              :+--------+              :   *
              :........................:   *
                                           *
                                           *
  .........................                *  .........................
  :+--------+              :                *  :              +--------+:
  :|Security|   Node B     :                *  :   Node A     |Security|:
  :|Service |              :                *  :              |Service |:
  :+--------+              :                *  :              +--------+:
  :    |     +-----------+:                *  :+-----------+     |*    :
  :    |  +->|Application|:                ****|Application|<*+* |*    :
  :    |  |  +-----------+:                   :+-----------+  |* |*    :
  :    |  |  +-----------+:                   :+-----------+  |* |*    :
  :    |  |->| Transport |~~~~~~~~~~~~~~~~~~~~~| Transport |<-|* |*    :
  :    |__|  +-----------+: ................. :+-----------+  |*_|*    :
  :       |  +-----------+: : +-----------+ : :+-----------+  | *       :
  :       |->|  Network  |: : |  Network  | : :|  Network  |<-|         :
  :       |  +-----------+: : +-----------+ : :+-----------+  |         :
  :       |  +-----------+: : +-----------+ : :+-----------+  |         :
  :       +->|     L2    |: : |     L2    | : :|     L2    |<-+         :
  :          +-----------+: : +-----------+ : :+-----------+            :
  :.......................: :...............: :.......................:
```

Overview of Security Mechanisms.

Figure 3

We now discuss an exemplary security architecture relying on a configuration entity for the management of the system with regard to the introduced security aspects (see Figure 2). Inspired by the security framework for routing over low power and lossy network [ID-Tsao], we show an example of security model and illustrates how different security concepts and the lifecycle phases map to the Internet communication stack. Assume a centralized architecture in which a configuration entity stores and manages the identities of the things associated with the system along with their cryptographic keys. During the bootstrapping phase, each thing executes the bootstrapping protocol with the configuration entity, thus obtaining the required device identities and the keying material. The security service on a thing in turn stores the received keying material for the network layer and application security mechanisms for secure communication. Things can then securely communicate with each other during their operational phase by means of the employed network and application security mechanisms.

4. State of the Art

Nowadays, there exists a multitude of control protocols for the IoT. For BAC systems, the ZigBee standard [ZB], BACNet [BACNET], or DALI [DALI] play key roles. Recent trends, however, focus on an all-IP approach for system control.

In this setting, a number of IETF working groups are designing new protocols for resource constrained networks of smart things. The 6LoWPAN working group [WG-6LoWPAN] concentrates on the definition of methods and protocols for the efficient transmission and adaptation of IPv6 packets over IEEE 802.15.4 networks [RFC4944]. The CoRE working group [WG-CoRE] provides a framework for resource-oriented applications intended to run on constrained IP network (6LoWPAN). One of its main tasks is the definition of a lightweight version of the HTTP protocol, the Constrained Application Protocol (CoAP) [ID-CoAP], that runs over UDP and enables efficient application-level communication for things.

4.1. IP-based Security Solutions

In the context of the IP-based IoT solutions, consideration of TCP/IP security protocols is important as these protocols are designed to fit the IP network ideology and technology. While a wide range of specialized as well as general-purpose key exchange and security solutions exist for the Internet domain, we discuss a number of protocols and procedures that have been recently discussed in the context of the above working groups. The considered protocols are IKEv2/IPsec [RFC4306], TLS/SSL [RFC5246], DTLS [RFC5238], HIP [RFC5201][ID-Moskowitz], PANA [RFC5191], and EAP [RFC3748] in this Internet-Draft. Application layer solutions such as SSH [RFC4251] also exist, however, these are currently not considered. Figure 3 depicts the relationships between the discussed protocols in the context of the security terminology introduced in Section 3.1.

```
             ..........................
             :           +-----------+:
             :       *+*>|Application|*****     *** Bootstrapping
             :       *|  +-----------+:   *     ### Application Security
             :       *|  +-----------+:   *     === Network security
             :       *|->| Transport |:   *
             :    * _*|  +-----------+:   *
             :    *|  |  +-----------+:   *
             :    *|  |->|  Network  |:   *--> -PANA/EAP
             :    *|  |  +-----------+:   *    -HIP
             :    *|  |  +-----------+:   *
             :    *|  +->|     L2    |:   *     ## DTLS
             :    *|     +-----------+:   *     ##
             :+--------+              :   *
             :|Security| Configuration:   *     [] HIP,IKEv2
             :|Service |   Entity     :   *     [] ESP/AH
             :+--------+              :   *
             :........................:   *
                                          *
 .........................                *    .........................
 :+--------+             :                *    :             +--------+:
 :|Security|   Node B    :                *    :   Node A    |Security|:
 :|Service |             :                *    :             |Service |:
 :+--------+             :     Secure     *    :             +--------+:
 :    |     +-----------+:     routing    *    :+-----------+     |*   :
 :    |  +->|Application|:    framework   ******|Application|<*+* |*   :
 :    |  |  +----##-----+:        |            :+----##-----+  |* |*   :
 :    |  |  +----##-----+:        |            :+----##-----+  |* |*   :
 :    |  |->| Transport |#########|#############| Transport |<-|* |*   :
 :    |__|  +----[]-----+:  ......|..........  :+----[]-----+  |*_|*   :
 :       |  +====[]=====+=====+===========+=====+====[]=====+  | *     :
 :       |->|| Network  |:  : |  Network  | :  :|  Network ||<-|       :
 :       |  +|----------+:  : +-----------+ :  :+----------|+  |       :
 :       |  +|----------+:  : +-----------+ :  :+----------|+  |       :
 :       +->||    L2    |:  : |     L2    | :  :|     L2   ||<-+       :
 :          +===========+=====+===========+=====+===========+          :
 :.......................:  :...............:  :.......................:
```

Relationships between IP-based security protocols.

Figure 4

The Internet Key Exchange (IKEv2)/IPsec and the Host Identity protocol (HIP) reside at or above the network layer in the OSI model. Both protocols are able to perform an authenticated key exchange and set up the IPsec transforms for secure payload delivery. Currently, there are also ongoing efforts to create a HIP variant coined Diet HIP [ID-HIP] that takes lossy low-power networks into account at the authentication and key exchange level.

Transport Layer Security (TLS) and its datagram-oriented variant DTLS secure transport-layer connections. TLS provides security for TCP and requires a reliable transport, while DTLS secures and uses datagram-oriented protocols such as UDP. Both protocols are intentionally kept similar and share the same ideology and cipher suites.

The Extensible Authentication Protocol (EAP) is an authentication framework supporting multiple authentication methods. EAP runs directly over the data link layer and, thus, does not require the deployment of IP. It supports duplicate detection and retransmission, but does not allow for packet fragmentation. The Protocol for Carrying Authentication for Network Access (PANA) is a network-layer transport for EAP that enables network access authentication between clients and the network infrastructure. In EAP terms, PANA is a UDP-based EAP lower layer that runs between the EAP peer and the EAP authenticator.

4.2. Wireless Sensor Network Security and Beyond

A variety of key agreement and privacy protection protocols that are tailored to IoT scenarios have been introduced in the literature. For instance, random key pre-distribution schemes [PROC-Chan] or more centralized solutions, such as SPINS [JOURNAL-Perrig], have been proposed for key establishment in wireless sensor networks. The ZigBee standard [ZB] for sensor networks defines a security architecture based on an online trust center that is in charge of handling the security relationships within a ZigBee network. Personal privacy in ubiquitous computing has been studied extensively, e.g., in [THESIS-Langheinrich]. Due to resource constraints and the specialization to meet specific requirements, these solutions often implement a collapsed cross layer optimized communication stack (e.g., without task-specific network layers and layered packet headers). Consequently, they cannot directly be adapted to the requirements of the Internet due to the nature of their design.

Despite important steps done by, e.g., Gupta et al. [PROC-Gupta], to show the feasibility of an end-to-end standard security architecture for the embedded Internet, the Internet and the IoT domain still do not fit together easily. This is mainly due to the fact that IoT security solutions are often tailored to the specific scenario requirements without considering interoperability with Internet protocols. On the other hand, the direct use of existing Internet security protocols in the IoT might lead to inefficient or insecure operation as we show in our discussion below.

5. Challenges for a Secure Internet of Things

In this section, we take a closer look at the various security challenges in the operational and technical features of the IoT and then discuss how existing Internet security protocols cope with these technical and conceptual challenges through the lifecycle of a thing. Table 1 summarizes which requirements need to be met in the lifecycle phases as well as the considered protocols. The structure of this section follows the structure of the table. This discussion should neither be understood as a comprehensive evaluation of all protocols, nor can it cover all possible aspects of IoT security. Yet, it aims at showing concrete limitations of existing Internet security protocols in some areas rather than giving an abstract discussion about general properties of the protocols. In this regard, the discussion handles issues that are most important from the authors' perspectives.

5.1. Constraints and Heterogeneous Communication

Coupling resource constrained networks and the powerful Internet is a challenge because the resulting heterogeneity of both networks complicates protocol design and system operation. In the following we briefly discuss the resource constraints of IoT devices and the consequences for the use of Internet Protocols in the IoT domain.

5.1.1. Tight Resource Constraints

The IoT is a resource-constrained network that relies on lossy and low-bandwidth channels for communication between small nodes, regarding CPU, memory, and energy budget. These characteristics directly impact the threats to and the design of security protocols for the IoT domain. First, the use of small packets, e.g., IEEE 802.15.4 supports 127-byte sized packets at the physical layer, may result in fragmentation of larger packets of security protocols. This may open new attack vectors for state exhaustion DoS attacks, which is especially tragic, e.g., if the fragmentation is caused by large key exchange messages of security protocols. Moreover, packet fragmentation commonly downgrades the overall system performance due to fragment losses and the need for retransmissions. For instance, fate-sharing packet flight as implemented by DTLS might aggravate the resulting performance loss.

	Bootstrapping phase	Operational Phase
Requirements	Incremental deployment	End-to-End security
	Identity and key management	Mobility support
	Privacy-aware identification	Group membership management
	Group creation	
Protocols	IKEv2	IKEv2/MOBIKE
	TLS/DTLS	TLS/DTLS
	HIP/Diet-HIP	HIP/Diet-HIP
	PANA/EAP	

Relationships between IP-based security protocols.

Figure 5

The size and number of messages should be minimized to reduce memory requirements and optimize bandwidth usage. In this context, layered approaches involving a number of protocols might lead to worse performance in resource-constrained devices since they combine the headers of the different protocols. In some settings, protocol negotiation can increase the number of exchanged messages. To improve performance during basic procedures such as, e.g., bootstrapping, it might be a good strategy to perform those procedures at a lower layer.

Small CPUs and scarce memory limit the usage of resource-expensive cryptoprimitives such as public-key cryptography as used in most Internet security standards. This is especially true, if the basic cryptoblocks need to be frequently used or the underlying application demands a low delay.

Independently from the development in the IoT domain, all discussed security protocols show efforts to reduce the cryptographic cost of the required public-key-based key exchanges and signatures with ECC[RFC5246][RFC5903][ID-Moskowitz][ID-HIP]. Moreover, all protocols have been revised in the last years to enable crypto agility, making cryptographic primitives interchangeable. Diet HIP takes the reduction of the cryptographic load one step further by focusing on cryptographic primitives that are to be expected to be enabled in hardware on IEEE 802.15.4 compliant devices. For example, Diet HIP does not require cryptographic hash functions but uses a CMAC [NIST] based mechanism, which can directly use the AES hardware available in standard sensor platforms. However, these improvements are only a first step in reducing the computation and communication overhead of Internet protocols. The question remains if other approaches can be applied to leverage key agreement in these heavily resource-constrained environments.

A further fundamental need refers to the limited energy budget available to IoT nodes. Careful protocol (re)design and usage is required to reduce not only the energy consumption during normal operation, but also under DoS attacks. Since the energy consumption of IoT devices differs from other device classes, judgments on the energy consumption of a particular protocol cannot be made without tailor-made IoT implementations.

5.1.2. Denial-of-Service Resistance

The tight memory and processing constraints of things naturally alleviate resource exhaustion attacks. Especially in unattended T2T communication, such attacks are difficult to notice before the service becomes unavailable (e.g., because of battery or memory exhaustion). As a DoS countermeasure, DTLS, IKEv2, HIP, and Diet HIP implement return routability checks based on a cookie mechanism to delay the establishment of state at the responding host until the address of the initiating host is verified. The effectiveness of these defenses strongly depends on the routing topology of the network. Return routability checks are particularly effective if hosts cannot receive packets addressed to other hosts and if IP addresses present meaningful information as is the case in today’s Internet. However, they are less effective in broadcast media or when attackers can influence the routing and addressing of hosts (e.g., if hosts contribute to the routing infrastructure in ad-hoc networks and meshes).

In addition, HIP implements a puzzle mechanism that can force the initiator of a connection (and potential attacker) to solve cryptographic puzzles with variable difficulties. Puzzle-based defense mechanisms are less dependent on the network topology but perform poorly if CPU resources in the network are heterogeneous (e.g., if a powerful Internet host attacks a thing). Increasing the puzzle difficulty under attack conditions can easily lead to situations, where a powerful attacker can still solve the puzzle while weak IoT clients cannot and are excluded from communicating with the victim. Still, puzzle-based approaches are a viable option for sheltering IoT devices against unintended overload caused by misconfigured or malfunctioning things.

5.1.3. Protocol Translation and End-to-End Security

Even though 6LoWPAN and CoAP progress towards reducing the gap between Internet protocols and the IoT, they do not target protocol specifications that are identical to their Internet pendants due to performance reasons. Hence, more or less subtle differences between IoT protocols and Internet protocols will remain. While these differences can easily be bridged with protocol translators at gateways, they become major obstacles if end-to-end security measures between IoT devices and Internet hosts are used.

Cryptographic payload processing applies message authentication codes or encryption to packets. These protection methods render the protected parts of the packets immutable as rewriting is either not possible because a) the relevant information is encrypted and inaccessible to the gateway or b) rewriting integrity-protected parts of the packet would invalidate the end-to-end integrity protection.

There are essentially four solutions for this problem:

1 Sharing symmetric keys with gateways enables gateways to transform (e.g., de-compress, convert, etc.) packets and re-apply the security measures after transformation. This method abandons end-to-end security and is only applicable to simple scenarios with a rudimentary security model.

2 Reusing the Internet wire format in the IoT makes conversion between IoT and Internet protocols unnecessary. However, it leads to poor performance because IoT specific optimizations (e.g., stateful or stateless compression) are not possible.

3 Selectively protecting vital and immutable packet parts with a MAC or with encryption requires a careful balance between performance and security. Otherwise, this approach will either result in poor performance (protect as much as possible) or poor security (compress and transform as much as possible).

4 Message authentication codes that sustain transformation can be realized by considering the order of transformation and protection (e.g., by creating a signature before compression so that the gateway can decompress the packet without recalculating the signature). This enables IoT specific optimizations but is more complex and may require application-specific transformations before security is applied. Moreover, it cannot be used with encrypted data because the lack of cleartext prevents gateways from transforming packets.

To the best of our knowledge, none of the mentioned security protocols provides a fully customizable solution in this problem space. In fact, they usually offer an end-to-end secured connection. An exception is the usage layered approach as might be PANA and EAP. In such a case, this configuration (i) allows for a number of configurations regarding the location of, e.g., the EAP authenticator and authentication server and (ii) the layered architecture might allow for authentication at different places. The drawback of this approach, however, lies in its high signaling traffic volume compared to other approaches. Hence, future work is required to ensure security, performance and interoperability between IoT and the Internet.

5.2. Bootstrapping of a Security Domain

Creating a security domain from a set of previously unassociated IoT devices is a key operation in the lifecycle of a thing and in the IoT network. In this section, we discuss general forms of network operation, how to communicate a thing's identity and the privacy implications arising from the communication of this identity.

5.2.1. Distributed vs. Centralized Architecture and Operation

Most things might be required to support both centralized and distributed operation patterns. Distributed thing-to-thing communication might happen on demand, for instance, when two things form an ad-hoc security domain to cooperatively fulfill a certain task. Likewise, nodes may communicate with a backend service located in the Internet without a central security manager. The same nodes may also be part of a centralized architecture with a dedicated node being responsible for the security management for group communication between things in the IoT domain. In today's IoT, most common architectures are fully centralized in the sense that all the security relationships within a segment are handled by a central party. In the ZigBee standard, this entity is the trust center. Current proposals for 6LoWPAN/CoRE identify the 6LoWPAN Border Router (6LBR) as such a device.

A centralized architecture allows for central management of devices and keying materials as well as for the backup of cryptographic keys. However, it also imposes some limitations. First, it represents a single point of failure. This is a major drawback, e.g., when key agreement between two devices requires online connectivity to the central node. Second, it limits the possibility to create ad-hoc security domains without dedicated security infrastructure. Third, it codifies a more static world view, where device roles are cast in stone, rather than a more dynamic world view that recognizes that networks and devices, and their roles and ownership, may change over time (e.g., due to device replacement and hand-over of control).

Decentralized architectures, on the other hand, allow creating ad-hoc security domains that might not require a single online management entity and are operative in a much more stand-alone manner. The ad-hoc security domains can be added to a centralized architecture at a later point in time, allowing for central or remote management.

5.2.2. Bootstrapping a thing’s identity and keying materials

Bootstrapping refers to the process by which a device is associated to another one, to a network, or to a system. The way it is performed depends upon the architecture: centralized or distributed. It is important to realize that bootstrapping may involve different types of information, ranging from network parameters and information on device capabilities and their presumed functionality, to management information related to, e.g., resource scheduling and trust initialization/management. Furthermore, bootstrapping may occur in stages during the lifecycle of a device and may include provisioning steps already conducted during device manufacturing (e.g., imprinting a unique identifier or a root certificate into a device during chip testing), further steps during module manufacturing (e.g., setting of application-based configurations, such as temperature read-out frequencies and push-thresholds), during personalization (e.g., fine-tuned settings depending on installation context), during hand-over (e.g., transfer of ownership from supplier to user), and, e.g., in preparation of operation in a specific network. In what follows, we focus on bootstrapping of security-related information, since bootstrapping of all other information can be conducted as ordinary secured communications, once a secure and authentic channel between devices has been put in place.

In a distributed approach, a Diffie-Hellman type of handshake can allow two peers to agree on a common secret. In general, IKEv2, HIP, TLS, DTLS, can perform key exchanges and the setup of security associations without online connections to a trust center. If we do not consider the resource limitations of things, certificates and certificate chains can be employed to securely communicate capabilities in such a decentralized scenario. HIP and Diet HIP do not directly use certificates for identifying a host, however certificate handling capabilities exist for HIP and the same protocol logic could be used for Diet HIP. It is noteworthy, that Diet HIP does not require a host to implement cryptographic hashes. Hence, some lightweight implementations of Diet HIP might not be able to verify certificates unless a hash function is implemented by the host.

In a centralized architecture, preconfigured keys or certificates held by a thing can be used for the distribution of operational keys in a given security domain. A current proposal [ID-OFlynn] refers to the use of PANA for the transport of EAP messages between the PANA client (the joining thing) and the PANA Authentication Agent (PAA), the 6LBR. EAP is thereby used to authenticate the identity of the joining thing. After the successful authentication, the PANA PAA provides the joining thing with fresh network and security parameters.

IKEv2, HIP, TLS, and DTLS could be applied as well for the transfer of configuration parameters in a centralized scenario. While HIP's cryptographic secret identifies the thing, the other protocols do not represent primary identifiers but are used instead to bind other identifiers such as the operation keys to the public-key identities.

In addition to the protocols, operational aspects during bootstrapping are of key importance as well. Many other standard Internet protocols assume that the identity of a host is either available by using secondary services like certificate authorities or secure name resolution (e.g., DNSsec) or can be provided over a side channel (entering passwords via screen and keyboard). While these assumptions may hold in traditional networks, intermittent connectivity, localized communication, and lack of input methods complicate the situation for the IoT.

The order in which the things within a security domain are bootstrapped plays an important role as well. In [RFC6345], the PANA relay element is introduced, relaying PANA messages between a PaC (joining thing) and PAA of a segment [ID-OFlynn]. This approach forces commissioning based on distance to PAA, i.e., things can only be bootstrapped hop-by-hop starting from those closer to the PAA, all things that are 1-hop away are bootstrapped first, followed by those that are 2-hop away, and so on. Such an approach might impose important limitations on actual use cases in which, e.g., an installer without technical background has to roll-out the system.

5.2.3. Privacy-aware Identification

During the last years, the introduction of RFID tags has raised privacy concerns because anyone might access and track tags. As the IoT involves not only passive devices, but also includes active and sensing devices, the IoT might irrupt even deeper in people's privacy spheres. Thus, IoT protocols should be designed to avoid these privacy threats during bootstrapping and operation where deemed necessary. In H2T and T2T interactions, privacy-aware identifiers might be used to prevent unauthorized user tracking. Similarly, authentication can be used to prove membership of a group without revealing unnecessary individual information.

TLS and DTLS provide the option of only authenticating the responding host. This way, the initiating host can stay anonymous. If authentication for the initiating host is required as well, either public-key certificates or authentication via the established encrypted payload channel can be employed. Such a setup allows to only reveal the responder’s identity to possible eavesdroppers.

HIP and IKEv2 use public-key identities to authenticate the initiator of a connection. These identities could easily be traced if no additional protection were in place. IKEv2 transmits this information in an encrypted packet. Likewise, HIP provides the option to keep the identity of the initiator secret from eavesdroppers by encrypting it with the symmetric key generated during the handshake. However, Diet HIP cannot provide a similar feature because the identity of the initiator simultaneously serves as static Diffie-Hellman key. Note that all discussed solutions could use anonymous public-key identities that change for each communication. However, such identity cycling may require a considerable computational effort for generating new asymmetric key pairs. In addition to the built-in privacy features of the here discussed protocols, a large body of anonymity research for key exchange protocols exists. However, the comparison of these protocols and protocol extensions is out of scope for this work.

5.3. Operation

After the bootstrapping phase, the system enters the operational phase. During the operational phase, things can relate to the state information created during the bootstrapping phase in order to exchange information securely and in an authenticated fashion. In this section, we discuss aspects of communication patterns and network dynamics during this phase.

5.3.1. End-to-End Security

Providing end-to-end security is of great importance to address and secure individual T2T or H2T communication within one IoT domain. Moreover, end-to-end security associations are an important measure to bridge the gap between the IoT and the Internet. IKEv2 and HIP, TLS and DTLS provide end-to-end security services including peer entity authentication, end-to-end encryption and integrity protection above the network layer and the transport layer respectively. Once bootstrapped, these functions can be carried out without online connections to third parties, making the protocols applicable for decentralized use in the IoT. However, protocol translation by intermediary nodes may invalidate end-to-end protection measures (see Section 5.1).

5.3.2. Group Membership and Security

In addition to end-to-end security, group key negotiation is an important security service for the T2Ts and Ts2T communication patterns in the IoT as efficient local broadcast and multicast relies on symmetric group keys.

All discussed protocols only cover unicast communication and therefore do not focus on group-key establishment. However, the Diffie-Hellman keys that are used in IKEv2 and HIP could be used for group Diffie-Hellman key-negotiations. Conceptually, solutions that provide secure group communication at the network layer (IPsec/IKEv2, HIP/Diet HIP) may have an advantage regarding the cryptographic overhead compared to application-focused security solutions (TLS/DTLS). This is due to the fact that application-focused solutions require cryptographic operations per group application, whereas network layer approaches may allow to share secure group associations between multiple applications (e.g., for neighbor discovery and routing or service discovery). Hence, implementing shared features lower in the communication stack can avoid redundant security measures.

A number of group key solutions have been developed in the context of the IETF working group MSEC in the context of the MIKEY architecture [WG-MSEC][RFC4738]. These are specifically tailored for multicast and group broadcast applications in the Internet and should also be considered as candidate solutions for group key agreement in the IoT. The MIKEY architecture describes a coordinator entity that disseminates symmetric keys over pair-wise end-to-end secured channels. However, such a centralized approach may not be applicable in a distributed environment, where the choice of one or several coordinators and the management of the group key is not trivial.

5.3.3. Mobility and IP Network Dynamics

It is expected that many things (e.g., wearable sensors, and user devices) will be mobile in the sense that they are attached to different networks during the lifetime of a security association. Built-in mobility signaling can greatly reduce the overhead of the cryptographic protocols because unnecessary and costly re-establishments of the session (possibly including handshake and key agreement) can be avoided. IKEv2 supports host mobility with the MOBIKE [RFC4555][RFC4621] extension. MOBIKE refrains from applying heavyweight cryptographic extensions for mobility. However, MOBIKE mandates the use of IPsec tunnel mode which requires to transmit an additional IP header in each packet. This additional overhead could be alleviated by using header compression methods or the Bound End-to-End Tunnel (BEET) mode [ID-Nikander], a hybrid of tunnel and transport mode with smaller packet headers.

HIP offers a simple yet effective mobility management by allowing hosts to signal changes to their associations [RFC5206]. However, slight adjustments might be necessary to reduce the cryptographic costs, for example, by making the public-key signatures in the mobility messages optional. Diet HIP does not define mobility yet but it is sufficiently similar to HIP to employ the same mechanisms. TLS and DTLS do not have standards for mobility support, however, work on DTLS mobility exists in the form of an Internet draft [ID-Williams]. The specific need for IP-layer mobility mainly depends on the scenario in which nodes operate. In many cases, mobility support by means of a mobile gateway may suffice to enable mobile IoT networks, such as body sensor networks. However, if individual things change their point of network attachment while communicating, mobility support may gain importance.

6. Security Suites for the IP-based Internet of Things

Different applications have different security requirements and needs and, depending on various factors, such as device capability, availability of network infrastructure, security services needed, usage, etc., the required security protection may vary from "no security" to "full-blown security". For example, applications may have different needs regarding authentication and confidentiality. While some application might not require any authentication at all, others might require strong end-to-end authentication. In terms of secure bootstrapping of keys, some applications might assume the existence and online availability of a central key-distribution-center (KDC) within the 6LoWPAN network to distribute and manage keys; while other applications cannot rely on such a central party or their availability.

Thus, it is essential to define security profiles to better tailor security solutions for different applications with the same characteristics and requirements. This provides a means of grouping applications into profiles and then defines the minimal required security primitives to enable and support the security needs of the profile. The security elements in a security profile can be classified according to Section 3.1, namely:

1   Security architecture,

2   Security model,

3   Security bootstrapping,

4   Network security, and

5   Application security.

In order to (i) guide the design process by identifying open gaps; (ii) allow for later interoperability; and (iii) prevent possible security misconfigurations, this section defines a number of generic security profiles with different security needs. Each security profile is identified by:

1   a short description,

2   an exemplary application that might use/require such a security policy,

3   the security requirements for each of the above security aspects according to our classification in Section 3.1.

These security profiles can serve to guide the standardization process, since these explicitly describe the basic functionalities and protocols required to get different use cases up and running. It can allow for later interoperability since different manufacturers can describe the implemented security profile in their products. Finally, the security profiles can avoid possible security misconfigurations, since each security profile can be bound to a different application area so that it can be clearly defined which security protocols and approaches can be applied where and under which circumstances.

Note that each of these security profiles aim at summarizing the required security requirements for different applications and at providing a set of initial security features. In other words, these profiles reflect the need for different security configurations, depending on the threat and trust models of the underlying applications. In this sense, this section does not provide an overview of existing protocols as done in previous sections of the Internet Draft, but it rather explicitly describes what should be in place to ensure secure system operation. Observe also that this list of security profiles is not exhaustive and that it should be considered just as an example not related to existing legal regulations for any existing application. These security profiles are summarized in the table below:

	Application	Description
SecProf_0	No security needs	6LoWPAN/CoAP is used without security
SecProf_1	Home usage	Enables operation between home things without interaction with central device
SecProf_2	Managed Home usage	Enables operation between home things. Interaction with a central and local device is possible
SecProf_3	Industrial usage	Enables operation between things. Relies on central (local or backend) device for security
SecProf_4	Advanced Industrial usage	Enables ad-hoc operation between things and relies on central device or on a collection of control devices

Security profiles and application areas.

Figure 6

The classification in the table considers different potential applications and situations in which their security needs change due to different operational features (network size, existence of a central device, connectivity to the Internet, importance of the exchanged information, etc) or threat model (what are the assets that an attacker looks for). As already pointed out, this set of scenarios is exemplary and they should be further discussed based on a broader consensus.

SecProf_0 is meant for any application that does not require security. Examples include applications during system development, system testing, or some very basic applications in which security is not required at all.

The second security suite (SecProf_1) is catered for environments in which 6LoWPAN/CoAP can be used to enable communication between things in an ad-hoc manner and the security requirements are minimal. An example, is a home application in which two devices should exchange information and no further connection with other devices (local or with a backend) is required. In this scenario, value of the exchanged information is low and that it usually happen in a confined room, thus, it is possible to have a short period of time during which initial secrets can be exchanged in the clear. Due to this fact, there is no requirement to enable devices from different manufacturers to interoperate in a secure way (keys are just exchanged). The expected network size of applications using this profile is expected to be small such that the provision of network security, e.g., secure routing, is of low importance.

The next security suite (SecProf_2) represents an evolution of SecProf_1 in which, e.g., home devices, can be managed locally. A first possibility for the securing domain management refers to the creation of a centrally managed security domain without any connectivity to the Internet. The central device used for management can serve as, e.g., a key distribution center including policies for key update, storage, etc. The presence of a central device can help in the management of larger networks. Network security becomes more relevant in this scenario since the 6LoWPAN/CoAP network can be prone to Denial of Service attacks (e.g., flooding if L2 is not protected) or routing attacks.

SecProf_3 considers that a central device is always required for network management. Example applications of this profile include building control and automation, sensor networks for industrial use, environmental monitoring, etc. As before, the network manager can be located in the 6LoWPAN/CoAP network and handle key management. In this case, the first association of devices to the network is required to be done in a secure way. In other words, the threat model requires measurements to protect against any vulterable period of time. This step can involve the secure transmission of keying materials used for network security at different layers. The information exchanged in the network is considered to be valuable and it should be protected in the sense of pairwise links. Commands should be secured and broadcast should be secured with entity authentication [ID-CoAPMulticast]. Network should be protected from attacks. A further extension to this use case is to allow for remote management. A "backend manager" is in charge of managing SW or information exchanged or collected within the 6LoWPAN/CoAP network. This requires connection of devices to the Internet over a 6LBR involving a number of new threats that were not present before. A list of potential attacks include: resource-exhaustion attacks from the Internet; amplification attacks; trust issues related a HTTP-CoAP proxy [ID-proHTTPCoAP], etc. This use case requires protecting the communication from a device in the backend to a device in the 6LoWPAN/CoAP network, end-to-end. This use case also requires measures to provide the 6LBR with the capability of dropping fake requests coming from the Internet. This becomes especially challenging when the 6LBR is not trusted and access to the exchanged information is limited; and even more in the case of a HTTP-CoAP proxy since protocol translation is required. This use case should take care of protecting information accessed from the backend due to privacy issues (e.g., information such as type of devices, location, usage, type and amount of exchanged information, or mobility patterns can be gathered at the backend threatening the privacy sphere of users) so that only required information is disclosed.

The last security suite (SecProf_4) essentially represents interoperability of all the security profiles defined previously. It considers applications with some additional requirements regarding operation such as: (i) ad-hoc establishment of security relationships between things (potentially from different manufacturers) in non-secure environments or (ii) dynamic roaming of things between different 6LoWPAN/CoAP security domains. Such operational requirements pose additional security requirements, e.g., in addition to secure bootstrapping of a device within a 6LoWPAN/CoAP security domain and the secure transfer of network operational key, there is a need to enable inter-domains secure communication to facilitate data sharing.

The above description illustrates how different applications of 6LoWPAN/CoAP networks involve different security needs. In the following sections, we summarize the expected security features or capabilities for each the security profile with regards to "Security Architecture", "Security Model", "Security Bootstrapping", "Network Security", and "Application Security".

6.1. Security Architecture

The choice of security architecture has many implications regarding key management, access control, or security scope. A distributed (or ad-hoc) architecture means that security relationships between things are setup on the fly between a number of objects and kept in a decentralized fashion. A locally centralized security architecture means that a central device, e.g., the 6LBR, handles the keys for all the devices in the security domain. Alternatively, a central security architecture could also refer to the fact that smart objects are managed from the backend. The security architecture for the different security profiles is classified as follows.

	Description
SecProf_0	-
SecProf_1	Distributed
SecProf_2	Distributed able to move centralized (local)
SecProf_3	Centralized (local &/or backend)
SecProf_4	Distributed & centralized (local &/or backend)

Security architectures in different security profiles.

Figure 7

In "SecProf_1", management mechanisms for the distributed assignment and management of keying materials is required. Since this is a very simple use case, access control to the security domain can be enabled by means of a common secret known to all devices. In the next security suite (SecProf_2), a central device can assume key management responsibilities and handle the access to the network. The last two security suites (SecProf_3 and SecProf_4) further allow for the management of devices or some keying materials from the backend.

6.2. Security Model

While some applications might involve very resource-constrained things such as, e.g., a humidity, pollution sensor, other applications might target more powerful devices aimed at more exposed applications. Security parameters such as keying materials, certificates, etc must be protected in the thing, for example by means of tamper-resistant hardware. Keys may be shared across a thing’s networking stack to provide authenticity and confidentiality in each networking layer. This would minimize the number of key establishment/agreement handshake and incurs less overhead for constrained thing. While more advance applications may require key separation at different networking layers, and possibly process separation and sandboxing to isolate one application from another. In this sense, this section reflects the fact that different applications require different sets of security mechanisms.

	Description
SecProf_0	-
SecProf_1	No tamper resistant Sharing keys between layers
SecProf_2	No tamper resistant Sharing keys between layers
SecProf_3	Tamper resistant Key and process separation
SecProf_4	(no) Tamper resistant Sharing keys between layers/Key and process separation Sandbox

Thing security models in different security profiles.

Figure 8

6.3. Security Bootstrapping and Management

Bootstrapping refers to the process by which a thing initiates its life within a security domain and includes the initialization of secure and/or authentic parameters bound to the thing and at least one other device in the network. Here, different mechanisms may be used to achieve confidentiality and/or authenticity of these parameters, depending on deployment scenario assumptions and the communication channel(s) used for passing these parameters. The simplest mechanism for initial set-up of secure and authentic parameters is via communication in the clear using a physical interface (USB, wire, chip contact, etc.). Here, one commonly assumes this communication channel is secure, since eavesdropping and/or manipulation of this interface would generally require access to the physical medium and, thereby, to one or both of the devices themselves. This mechanism was used with the so-called original "resurrecting duckling" model, as introduced in [PROC-Stajano]. This technique may also be used securely in wireless, rather than wired, set-ups, if the prospect of eavesdropping and/or manipulating this channel are dim (a so-called "location-limited" channel [PROC-Smetters-04, PROC-Smetters-02]). Examples hereof include the communication of secret keys in the clear using near field communication (NFC) - where the physical channel is purported to have very limited range (roughly 10cm), thereby thwarting eavesdropping by far-away adversarial devices, and in-the-clear communication during a small time window (triggered by, e.g., a button-push) - where eavesdropping is presumed absent during this small time window. With the use of public-key based techniques, assumptions on the communication channel can be relaxed even further, since then the cryptographic technique itself provides for confidentiality of the channel set-up and the location-limited channel - or use of certificates - rules out man-in-the-middle attacks, thereby providing authenticity [PROC-Smetters-02]. The same result can be obtained using password-based public-key protocols [SPEKE], where authenticity depends on the (weak) password not being guessed during execution of the protocol. It should be noted that while most of these techniques realize a secure and authentic channel for passing parameters, these generally do not provide for explicit authorization. As an example, with use of certificate-based public-key based techniques, one may obtain hard evidence on whom one shares secret and/or authentic parameters with, but this does not answer the question as to whether one wishes to share this information at all with this specifically identified device (the latter usually involves a human-decision element). Thus, the bootstrapping mechanisms above should generally be complemented by mechanisms that regulate (security policies for) authorization. Furthermore, the type of bootstrapping is very related to the required type of security architecture. Distributed bootstrapping means that a pair of devices can setup a security relationship on the fly, without interaction with a central device elsewhere within the system. In many cases, it is handy to have a distributed bootstrapping protocol based on existing security protocols (e.g., DTLS in CoAP) required for other purposes: this reduces the amount of required software. A centralized boostrapping protocol is one in which a central device manages the security relationships within a network. This can happen locally, e.g., handled by the 6LBR, or remotely, e.g., from a server connected via the Internet. The security bootstrapping for the different security profiles is as follows.

	Description
SecProf_0	-
SecProf_1	* Distributed, (e.g., Resurrecting duckling) * First key distribution happens in the clear
SecProf_2	* Distributed, (e.g., Resurrecting duckling) * Centralized (local), 6LBR acts as KDC * First key distribution occurs in the clear, if the KDC is available, the KDC can manage network access
SecProf_3	* 6LBR acts as KDC. It handles node joining, provides them with keying material from L2 to application layers * Bootstrapping occurs in a secure way - either in secure environment or the security mechanisms ensure that eavesdropping is not possible. * KDC and backend can implement secure methods for network access
SecProf_4	* As in SecProf_3.

Security boostrapping methods in different security profiles

Figure 9

6.4. Network Security

Network security refers to the mechanisms used to ensure the secure transport of 6LoWPAN frames. This involves a multitude of issues ranging from secure discovery, frame authentication, routing security, detection of replay, secure group communication, etc. Network security is important to thwart potential attacks such as denial-of-service (e.g., through message flooding) or routing attacks.

The Internet Draft [ID-Tsao] presents a very good overview of attacks and security needs classified according to the confidentiality, integrity, and availability needs. A potential limitation is that there exist no differentiation in security between different use cases and the framework is limited to L3. The security suites gathered in the present ID aim at solving this by allowing for a more flexible selection of security needs at L2 and L3.

	Description
SecProf_0	-
SecProf_1	* Network key creating a home security domain at L2 ensuring authentication and freshness of exchanged data * Secure multicast does not ensure origin authentication * No need for secure routing at L3
SecProf_2	* Network key creating a home security domain at L2 ensuring authentication and freshness of exchanged data * Secure multicast does not ensure origin authentication * No need for secure routing at L3
SecProf_3	* Network key creating an industry security domain at L2 ensuring authentication and freshness of exchanged data * Secure routing needed (integrity & availability) at L3 within 6LoWPAN/CoAP * Secure multicast requires origin authentication
SecProf_4	* Network key creating an industry security domain at L2 ensuring authentication and freshness of exchanged data * Inter-domain authentication/secure handoff * Secure routing needed at L3 * Secure multicast requires origin authentication * 6LBR (HTTP-CoAP proxy) requires verification of forwarded messages and messages leaving or entering the 6LoWPAN/CoAP network.

Network security needs in different security profiles

Figure 10

6.5. Application Security

In the context of 6LoWPAN/CoAP networks, application security refers firstly to the configuration of DTLS used to protect the exchanged information. It further refers to the measures required in potential translation points (e.g., a (HTTP-CoAP) proxy) where information can be collected and the privacy sphere of users in a given security domain is endangered. Application security for the different security profiles is as follows.

	Description
SecProf_0	-
SecProf_1	-
SecProf_2	* DTLS is used for end-to-end application security between management device and things and between things * DTLS ciphersuites configurable to provide confidentiality and/or authentication and/or freshness * Key transport and policies for generation of session keys are required
SecProf_3	* Requirements as in SecProf_2 and * DTLS is used for end-to-end application security between management device and things and between things * Communication between KDC and each thing secured by pairwise keys * Group keys for communication in a group distributed by KDC * Privacy protection should be provided in translation points
SecProf_4	* Requirements as in SecProf_3 and * TLS or DTLS can be used to send commands from the backend to the 6LBR or things in a 6LoWPAN/CoAP network * End-to-end secure connectivity from backend required * Secure broadcast in a network from backend required

Application security methods in different security profiles

Figure 11

The first two security profiles do not include any security at the application layer. The reason is that, in the first case, security is not provided and, in the second case, it seems reasonable to provide basic security at L2. In the third security profile (SecProf_2), DTLS becomes the way of protecting messages at application layer between things and with the KDC running on a 6LBR. A key option refers to the capability of easily configuring DTLS to provide a subset of security services (e.g., some applications do not require confidentiality) to reduce the impact of security in the system operation of resource-constrained things. In addition to basic key management mechanisms running within the KDC, communication protocols for key transport or key update are required. These protocols could be based on DTLS. The next security suite (SecProf_3) requires pairwise keys for communication between things within the security domain. Furthermore, it can involve the usage of group keys for group communication. If secure multicast is implemented, it should provide origin authentication. Finally, privacy protection should be taken into account to limit access to valuable information -- such as identifiers, type of collected data, traffic patterns -- in potential translation points (proxies) or in the backend. The last security suite (SecProf_4) further extends the previous set of requirements considering security mechanisms to deal with translations between TLS and DTLS or for the provision of secure multicast within a 6LoWPAN/CoAP network from the backend.

7. Next Steps towards a Flexible and Secure Internet of Things

This Internet Draft included an overview of both operational and security requirements of things in the Internet of Things, discussed a general threat model and security issues, and introduced a number of potential security suites fitting different types of IoT deployments.

We conclude this document by giving our assessment of the current status of CoAP security with respect to addressing the IP security challenges we identified, so as to facilitate discussion of next steps towards workable security design concepts suitable for IP-based IoT in the broader community. Hereby, we focus on the employed security protocols and the type of security architecture.

With current status, we refer to the feasibility of realizing secure deployments with existing CoAP protocols and the practicality of creating comprehensive security architectures based on those protocols:

1   DTLS has been defined as the basic building block for protecting CoAP. At the time it was first proposed, no DTLS implementation for small, constrained devices was available. In the mean-time, TinyDTLS [TinyDTLS] has been developed offering the first open-source implementation of the protocol for small devices. However, more experience with the protocol is required. In particular, a performance evaluation and comparison should be made with a well-defined set of standard node platforms/networks. The results will help understand the limitations and the benefits of DTLS as well as to give recommended usage scenarios for this security protocol.

2   (D)TLS was designed for traditional computer networks and, thus, some of its features may not be optimal for resource-constrained networks. This includes:

    a   Basic DTLS features that are, in our view, not ideal for resource-constrained devices. For instance, the loss of a message in-flight requires the retransmission of all messages in-flight. On the other hand, if all messages in-flight are transmitted together in a single UDP packet, more resources are required for handling of large buffers. As pointed out in [ID-Hartke] , the number of flights in the DTLS handshake should be reduced, so that a faster setup of a secure channel can be realized. This would definitely improve the performance of DTLS significantly.

    b   Fragmentation of messages due to smaller MTUs in resource-constrained networks is problematic. This implies that the node must have a large buffer to store all the fragments and subsequently perform re-ordering and reassembly in order to construct the entire DTLS message. The fragmentation of the handshake messages can, e.g., allow for a very simple method to carry out a denial of service attack.

    c   The completion of the DTLS handshake is based on the successful verification of the Finished message by both client and server. As the Finished message is computed based on the hash of all handshake messages in the correct order, the node must allocate a large buffer to queue all handshake messages.

    d   DTLS is thought to offer end-to-end security; however, end-to-end security also has to be considered from the point of view of LLN protection, so that end-to-end exchanges can still be verified and the LLN protected from, e.g., DoS attacks.

3   Raw public-key in DTLS has been defined as mandatory. However, memory-optimized public-key libraries still require several KB of flash and several hundreds of B of RAM. Although Moore’s law still applies and an increase of platform resources is expected, many IoT scenarios are cost-driven, and in many use cases, the same work could be done with symmetric-keys. Thus, a key question is whether the choice for raw public-key is the best one. In addition, using raw public keys rather than certified public keys hard codes identities to public keys, thereby inhibiting public key updates and potentially complicating initial configuration.

4   Performance of DTLS from a system perspective should be evaluated involving not just the cryptographic constructs and protocols, but should also include implementation benchmarks for security policies, since these may impact overall system performance and network traffic (an example of this would be policies on the frequency of key updates, which would necessitate securely propagating these to all devices in the network).

5   Protection of lower protocol layers is a must in networks of any size to guarantee resistance against routing attacks such as flooding or wormhole attacks. The wireless medium that is used by things to communicate is broadcast in nature and allows anybody on the right frequency to overhear and even inject packets at will. Hence, IP-only security solutions may not suffice in many IoT scenarios. At the time of writing the document, comprehensive methods are either not in place or have not been evaluated yet. This limits the deployment of large-scale systems and makes the secure deployment of large scale networks rather infeasible.

6   The term "bootstrapping" has been discussed in many occasions. Although everyone agrees on its importance, finding a good solution applicable to most use cases is rather challenging. While usage of existing methods for network access might partially address bootstrapping in the short-term and facilitate integration with legacy back-end systems, we feel that, in the medium-term, this may lead to too large of an overhead and imposes unnecessary constraints on flexible deployment models. The bootstrapping protocol should be reusable and light-weight to fit with small devices. Such a standard bootstrapping protocol must allow for commissioning of devices from different manufacturers in both centralized and ad-hoc scenarios and facilitate transitions of control amongst devices during the device’s and system’s lifecycle. Examples of the latter include scenarios that involve hand-over of control, e.g., from a configuration device to an operational management console and involving replacement of such a control device. A key challenge for secure bootstrapping of a device in a centralized architecture is that it is currently not feasible to commission a device when the adjacent devices have not been commissioned yet. In view of the authors, a light-weight approach is still required that allows for the bootstrapping of symmetric-keys and of identities in a certified public-key setting.

7   Secure resource discovery has not been discussed so far. However, this issue is currently gaining relevance. The IoT, comprising sensors and actuators, will provide access to many resources to sense and modify the environment. The usage of DNS presents

well-known security issues, while the application of secure DNS may not be feasible on small devices. In general, security issues and solutions related to resource discovery are still unclear.

8   A security architecture involves, beyond the basic protocols, many different aspects such as key management and the management of evolving security responsibilities of entities during the lifecycle of a thing. This document discussed a number of security suites and argued that different types of security architectures are required. A flexible IoT security architecture should incorporate the properties of a fully centralized architecture as well as allow devices to be paired together initially without the need for a trusted third party to create ad-hoc security domains comprising a number of nodes. These ad-hoc security domains could then be added later to the Internet via a single, central node or via a collection of nodes (thus, facilitating implementation of a centralized or distributed architecture, respectively). The architecture should also facilitate scenarios, where an operational network may be partitioned or merged, and where hand-over of control functionality of a single device or even of a complete subnetwork may occur over time (if only to facilitate smooth device repair/ replacement without the need for a hard "system reboot" or to realize ownership transfer). This would allow the IoT to transparently and effortlessly move from an ad-hoc security domain to a centrally-managed single security domain or a heterogeneous collection of security domains, and vice-versa. However, currently, these features still lack validation in real-life, large-scale deployments.

9   Currently, security solutions are layered, in the sense that each layer takes care of its own security needs. This approach fits well with traditional computer networks, but it has some limitations when resource-constrained devices are involved and these devices communicate with more powerful devices in the back-end. We argue that protocols should be more interconnected across layers to ensure efficiency as resource limitations make it challenging to secure (and manage) all layers individually. In this regard, securing only the application layer leaves the network open to attacks, while security focused only at the network or link layer might introduce possible inter-application security threats. Hence, the limited resources of things may require sharing of keying material and common security mechanisms between layers. It is required that the data format of the keying material is standardized to facilitate cross-layer interaction. Additionally, cross-layer concepts should be considered for an IoT-driven re-design of Internet security

protocols.

8. Security Considerations

This document reflects upon the requirements and challenges of the security architectural framework for Internet of Things.

9. IANA Considerations

This document contains no request to IANA.

10. Acknowledgements

We gratefully acknowledge feedback and fruitful discussion with Tobias Heer and Robert Moskowitz.

11. References

11.1. Informative References

[RFC6568]Kim, E., Kaspar, D., and JP. Vasseur, "Design and Application Spaces for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6568, April 2012.

[RFC2818]Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC6345]Duffy, P., Chakrabarti, S., Cragie, R., Ohba, Y., Ed., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA) Relay Element", RFC 6345, August 2011.

[ID-CoAP]Z. Shelby, K. Hartke, C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18, June 2013.

[ID-CoAPMulticast]Rahman, A. and E. Dijk, "Group Communication for CoAP",draft-ietf-core-groupcomm-12 (work in progress), July 2013.

[ID-Daniel]Park, S., Kim, K., Haddad, W., Chakrabarti, S., and J. Laganier, "IPv6 over Low Power WPAN Security Analysis",Internet Draft draft-daniel-6lowpan-security-analysis-05, Mar 2011.

[ID-HIP]Moskowitz, R., "HIP Diet EXchange (DEX)", draft-moskowitz-hip-rg-dex-06 (work in progress), May 2012.

[ID-Hartke]Hartke, K. and O. Bergmann, "Datagram Transport Layer Security in Constrained Environments", draft-hartke-core-codtls-02 (work in progress), July 2012.

[ID-Moskowitz]Moskowitz, R., Heer, T., Jokela, P., and Henderson, T., "Host Identity Protocol Version 2", draft-ietf-hip-rfc5201-bis-13 (work in progress), Sep 2013.

[ID-Nikander]Nikander, P. and J. Melen, "A Bound End-to-End Tunnel(BEET) mode for ESP", draft-nikander-esp-beet-mode-09, Aug 2008.

[ID-OFlynn]O'Flynn, C., Sarikaya, B., Ohba, Y., Cao, Z., and R. Cragie, "Security Bootstrapping of Resource-Constrained Devices", draft-oflynn-core-bootstrapping-03 (work in progress), Nov 2010.

[ID-Tsao]Tsao, T., Alexander, R., Dohler, M., Daza, V., and A. Lozano, "A Security Framework for Routing over Low Power and Lossy Networks", draft-ietf-roll-security-framework-07, Jan 2012.

[ID-Williams]Williams, M. and J. Barrett, "Mobile DTLS", draft-barrett-mobile-dtls-00, Mar 2009.

[ID-proHTTPCoAP]Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Best practices for HTTP-CoAP mapping implementation", draft-castellani-core-http-mapping-07(work in progress), Feb 2013.

[RFC3261]Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3748]Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, Ed., "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.

[RFC3756]Nikander, P., Ed., Kempf, J., and E. Nordmark, "IPv6 Neighbor Discovery (ND) Trust Models and Threats", RFC 3756, May 2004.

[RFC3833]Atkins, D. and R. Austein, "Threat Analysis of the Domain Name System (DNS)", RFC 3833, August 2004.

[RFC4016]Parthasarathy, M., "Protocol for Carrying Authentication and Network Access (PANA) Threat Analysis and Security Requirements", RFC 4016, March 2005.

[RFC5246]Dierks, T. and E. Rescorla, "The Transport Layer Security

(TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC4251]Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Protocol Architecture", RFC 4251, January 2006.

[RFC4306]Kaufman, C., Ed., "Internet Key Exchange (IKEv2) Protocol", RFC 4306, December 2005.

[RFC4555]Eronen, P., "IKEv2 Mobility and Multihoming Protocol (MOBIKE)", RFC 4555, June 2006.

[RFC4621]Kivinen, T. and H. Tschofenig, "Design of the IKEv2 Mobility and Multihoming (MOBIKE) Protocol", RFC 4621, August 2006.

[RFC4738]Ignjatic, D., Dondeti, L., Audet, F., and P. Lin, "MIKEY-RSA-R: An Additional Mode of Key Distribution in Multimedia Internet KEYing (MIKEY)", RFC 4738, November 2006.

[RFC4919]Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, August 2007.

[RFC4944]Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.

[RFC5191]Forsberg, D., Ohba, Y., Ed., Patil, B., Tschofenig, H., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA)", RFC 5191, May 2008.

[RFC5201]Moskowitz, R., Nikander, P., Jokela, P., Ed., and T. Henderson, "Host Identity Protocol", RFC 5201, April 2008.

[RFC5206]Nikander, P., Henderson, T., Ed., Vogt, C., and J. Arkko, "End-Host Mobility and Multihoming with the Host Identity Protocol", RFC 5206, April 2008.

[RFC5238]Phelan, T., "Datagram Transport Layer Security (DTLS) over the Datagram Congestion Control Protocol (DCCP)", RFC 5238, May 2008.

[RFC5246]Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5713]Moustafa, H., Tschofenig, H., and S. De Cnodder, "Security Threats and Security Requirements for the Access Node Control Protocol (ANCP)", RFC 5713, January 2010.

[RFC5903]Fu, D. and J. Solinas, "Elliptic Curve Groups modulo a Prime

(ECP Groups) for IKE and IKEv2", RFC 5903, June 2010.

[AUTO-ID]"AUTO-ID LABS", Web http://www.autoidlabs.org/, Sept 2010.

[BACNET]"BACnet", Web http://www.bacnet.org/, Feb 2011.

[DALI]"DALI", Web http://www.dalibydesign.us/dali.html, Feb 2011.

[JOURNAL-Perrig]Perrig, A., Szewczyk, R., Wen, V., Culler, D., and J. Tygar, "SPINS: Security protocols for Sensor Networks",Journal Wireless Networks, Sept 2002.

[NIST]Dworkin, M., "NIST Specification Publication 800-38B", 2005.

[PROC-Chan]Chan, H., Perrig, A., and D. Song, "Random Key Predistribution Schemes for Sensor Networks", Proceedings IEEE Symposium on Security and Privacy, 2003.

[PROC-Gupta]Gupta, V., Wurm, M., Zhu, Y., Millard, M., Fung, S., Gura, N., Eberle, H., and S. Shantz, "Sizzle: A Standards-based End-to-End Security Architecture for the Embedded Internet", Proceedings Pervasive Computing and Communications (PerCom), 2005.

[PROC-Smetters-02]Balfanz, D., Smetters, D., Steward, P., and H. Chi Wong,"Talking To Strangers: Authentication in Ad-Hoc Wireless Networks", Paper NDSS, 2002.

[PROC-Smetters-04]Balfanz, D., Durfee, G., Grinter, R., Smetters, D., and P. Steward, "Network-in-a-Box: How to Set Up a Secure Wireless Network in Under a Minute", Paper USENIX, 2004.

[PROC-Stajano-99]Stajano, F. and R. Anderson, "Resurrecting Duckling - Security Issues for Adhoc Wireless Networks", 7th International Workshop Proceedings, Nov 1999.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[THESIS-Langheinrich]Langheinrich, M., "Personal Privacy in Ubiquitous Computing", PhD Thesis ETH Zurich, 2005.

[TinyDTLS "TinyDTLS", Web http://tinydtls.sourceforge.net/, Feb 2012.

[WG-6LoWPAN]"IETF 6LoWPAN Working Group", Web http://tools.ietf.org/wg/6lowpan/, Feb 2011.

[WG-CoRE]"IETF Constrained RESTful Environment (CoRE) Working Group", Web https://datatracker.ietf.org/wg/core/charter/, Feb 2011.

[WG-MSEC]"MSEC Working Group", Web http://datatracker.ietf.org/wg/msec/.

[ZB]"ZigBee Alliance", Web http://www.zigbee.org/, Feb 2011.

Authors' Addresses

Oscar Garcia-Morchon
Philips Research
High Tech Campus
Eindhoven,   5656 AA
The Netherlands

Email: oscar.garcia@philips.com

Sandeep S. Kumar
Philips Research
High Tech Campus
Eindhoven,   5656 AA
The Netherlands

Email: sandeep.kumar@philips.com

Sye Loong Keoh
University of Glasgow Singapore
Republic PolyTechnic, 9 Woodlands Ave 9
Singapore 838964
SG

Email: SyeLoong.Keoh@glasgow.ac.uk

Rene Hummen
RWTH Aachen University
Templergraben 55
Aachen,   52056
Germany

Email: rene.hummen@cs.rwth-aachen.de

Rene Struik
Struik Security Consultancy
Toronto,
Canada

Email: rstruik.ext@gmail.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: November 6, 2015

D. He
Huawei
B. Sarikaya
Huawei USA

May 5, 2015

Security Bootstrapping of IEEE 802.15.4 based Internet of Things
draft-he-iot-security-bootstrapping-01

Abstract

Network level security bootstrapping and joining device level security bootstrapping mechanisms are described in this document. They are proposed for security bootstrapping of the Internet of Things networks, which implement IETF protocols (e.g. 6LoWPAN, 6lo, RPL, AODV, DSR) over IEEE 802.15.4. The network level security bootstrapping is useful at the very beginning of a newly deployed IoT network. It automatically and hierarchically adds all the devices to security domain and helps establish security communication. The joining device level security bootstrapping provides comprehensive mechanism for different IoT devices joining an existing IoT network.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

An IoT network is composed of various numbers of connected things with communication ability and different functionalities (sensing unit, control logic). They cooperate together to accomplish specific tasks required by users. Things in an IoT network might be supplied by different vendors, and are normally resource-constrained devices that with limited power supply, communication capability, CPU performance and memory volume.

[IEEE802.15.4]is a standard which specifies the physical layer and media access control for low-rate wireless personal area networks (LR-WPANs). It is widely used in wireless sensor networks nowadays, 6LoWPAN WG (concluded) developed RFC 4944[RFC4944] to describe how to transmit IPv6 packets over 802.15.4, and support mesh routing in LR-WPANs. 6lo WG defines generic IPv6 packet header compression method [RFC7400] for LR-WPANs. 6tisch tries to build adaptation protocol for 802.15.4e protocol. Roll develops routing protocol RPL[RFC6550] for IPv6 based low power and lossy networks. IEEE 802.15.4 is foreseen as the most used lower layer protocol for low rate IoT networks with resource constrained devices.

Creating security domains from previously unassociated IoT devices is a key operation in the IoT network and in the lifecycle of a thing. Because IEEE 802.15.4 maximum payload size is 128 Bytes, a standard security bootstrapping protocol should be light-weight with low complexity. The protocol must allow for commissioning of devices from different manufacturers and facilitate transitions of control amongst devices during the device’s and system’s lifecycle.

Traditional security bootstrapping approaches include device authentication and key generation/distribution, which tend to impose configuration burdens upon users. For example, users need to follow a series of instruction steps for WPA2-PSK (WiFi Protected Access 2, Pre-shared key) configuration, even though the pre-shared key mode is the simplest option for using WPA. Establishing security among IoT devices becomes more complicated since they don’t always provide user interface to input necessary security information. Furthermore, the scale of the IoT network can be large, human intervention in large scale security bootstrapping is expensive and low efficient.

[I-D.pritikin-anima-bootstrapping-keyinfra] proposes a zero-touch bootstrapping key infrastructure to allow joining device securely and automatically bootstraps itself based on 802.1AR certificate. It can’t be directly used in 802.15.4 devices due to the high security complexity and heavy communication overhead. Its architecture is not built by considering different possible 802.15.4 network topologies and the underlying routing protocols developed by IETF.

[I-D.struik-6tisch-security-considerations]defines high level requirements and proposes two types of security mechanisms: single-stage and two-stage. Even though the two types of security AA mechanisms offer flexible solutions. The underlying security architecture can neither be used directly by 802.15.4 IoT networks. IEEE 802.15.4 also defines two-step mechanism for nodes joining network with layer 2 authentication. Without considering use of IPv6 infrastructure, the solution is not comprehensive.

Another key challenge for security bootstrapping of a device the above mentioned mechanisms is that they are not feasible to commission a device when the adjacent devices have not been commissioned yet. As a result, this document describes and standardizes two types of automatic bootstrapping methods for 802.15.4 based IoT networks: network level security bootstrapping and joining device level security bootstrapping.

2. new section

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This specification requires readers to be familiar with all the terms and concepts that are discussed in "Neighbor Discovery for IP version 6 (IPv6)" [RFC4861], "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals" [RFC4919].This specification makes extensive use of the same terminology defined in [RFC4944].

3. IEEE 802.15.4 based IoT topologies

A general architectural overview of the IEEE 802.15.4 based IoT is provided in Figure 1. All the devices communicate to backbone server through 6LBR. FFDs communicate with each other directly or indirectly via hopping or 6LBR. RFDs directly connect to FFDs, and the number of RFDs that attach to a FFD may vary.

```
                     /////-------------------\\\\\
                    |            Server           |
                     \\\\\-------------------/////
                                   |
   +-------------------------------------------------------------------+
   |                              6LBR                                 |
   +--------------------------------+----------------------------------+
             |             +--------+-----------+           |
             |       +-**->|        FFD_2       |<--**-+    |
             |       |     +--------------------+      |    |
   +-----------------+--+                          +---+--------------+
   |        FFD_1       | <---------*****--------> |        FFD_N     |
   +--------------------+                          +------------------+
           |           |                                 |
   +--------------+  +--------------+                +--------------+
   |     RFD_11   |  |     RFD_1M   |                |     FFD_N1   |
   +--------------+  +--------------+                +--------------+
```

Figure 1

4. Network level security bootstrapping

At the very beginning of the networking once nodes are deployed, network level security bootstrapping assist automatically creates security domain and hierarchically adds devices to network. The mechanism is realized by three phases:

Phase 1: Security bootstrapping for the first hop FFDs via 6LBR

Phase 2: Security bootstrapping for further FFDs via configured FFDs

Phase 3: Security bootstrapping for RFDs via configured FFDs

4.1. Security bootstrapping for the first hop FFDs via 6LBR

When devices are power-on, 6LBR broadcasts beacon frames to neighboring nodes. The FFDs that receive the beacon frames are the first-hop FFDs. As shown in Figure 2, upon receiving the beacon frame, a first-hop FFD associates with 6LBR at link layer according to IEEE 802.15.4. The FFD then presents credential to 6LBR, which are forwarded to trust center to be validated. EAP can be used to realize the authentication procedure. If the validation is successful, the IP address and network key are generated and delivered to the FFD. Further configurations such as cluster head selection, routing protocol, etc., can be realized afterwards. Otherwise if the validation fails, the 6LBR refuses adding the FFD to its domain.

```
    First-hop FFD                     6LBR                     TC
     |                                 |                        |
     |           Beaconing             |                        |
     |<--------------------------------|                        |
     |                                 |                        |
     |           IEEE 802.15.4         |                        |
     |     MAC unsecure association    |                        |
     |<------------------------------->|                        |
     |                                 |                        |
     |                                 |                        |
     |        Authentication           |    Auth.material check |
     |<------------------------------->|<---------------------->|
     |   Network key and IP address    |      IP address        |
     |                                 |                        |
     |                                 |                        |
     |     Further Configuration       |                        |
     |<------------------------------->|                        |
     |                                 |                        |
     |                                 |                        |
```

Figure 2

4.2. Security bootstrapping for further FFDs via configured FFDs

The configured FFDs broadcast beacon frames to neighboring nodes. The unconfigured FFD that receives the beacon frame associates with the configured FFD at link layer. A FFD may receive multiple beacon frames from more than one configured FFDs, it can select the first one to associate or the one with strongest received power strength. The selection policy is out of the scope of the current document. The unconfigured FFD then presents credential to the associated configured FFD, which are forwarded to 6LBR and TC to be validated. If EAP is used, PANA can be used to relay the authentication message from configured FFDs to 6LBR. If the validation is successful, the IP address and network key are generated and delivered to the FFD. Further configurations such as routing protocol can be realized afterwards. Otherwise if the validation fails, the 6LBR refuses adding the FFD to its domain.

```
  Unconfigured FFD             Configured FFD      6LBR               TC
  |                                 |              |                  |
  |           Beaconing             |              |                  |
  |<--------------------------------|              |                  |
  |                                 |              |                  |
  |           IEEE 802.15.4         |              |                  |
  |     MAC unsecure association    |              |                  |
  |<------------------------------->|              |                  |
  |                                 |              |                  |
  |                                 |              |                  |
  |         Authentication          |     Relay    |    Auth.check    |
  |<------------------------------->|<------------>|<---------------->|
  |   Network key and IP address    |              |   IP address     |
  |                                 |              |                  |
  |                                 |              |                  |
  |     Further Configuration       |              |                  |
  |<-------------------------------- ------------->|                  |
  |                                 |              |                  |
  |                                 |              |                  |
```

Figure 3

4.3. Security bootstrapping for RFDs via configured FFDs

The configured FFDs broadcast beacon frames to neighboring nodes. The unconfigured RFD that receives the beacon frame associates with the configured FFD at link layer. A RFD may receive multiple beacon frames from more than one configured FFDs. It can select one device to associate, e.g. the first one that replies or the one with strongest received power strength. The unconfigured RFD then presents credential to the associated configured FFD, which are forwarded to 6LBR and TC to be validated. If the validation is successful, the IP address and network key are generated and delivered to the RFD. Otherwise if the validation fails, the FFD refuses adding the RFD to its domain.

```
  RFD                           Configured FFD     6LBR               TC
   |                                 |              |                  |
   |           Beaconing             |              |                  |
   |<--------------------------------|              |                  |
   |                                 |              |                  |
   |           IEEE 802.15.4         |              |                  |
   |     MAC unsecure association    |              |                  |
   |<------------------------------->|              |                  |
   |                                 |              |                  |
   |                                 |              |                  |
   |        Authentication           |     Relay    |     Auth.check   |
   |<------------------------------->|<------------>|<---------------->|
   |   Network key and IP address    |              |   IP address     |
   |                                 |              |                  |
   |                                 |              |                  |
   |     Further Configuration       |              |                  |
   |<-------------------------------- ------------->|                  |
   |                                 |              |                  |
   |                                 |              |                  |
```

 Figure 4

5. Joining Device Security Bootstrapping

New devices may be added to an existing IoT due to various reasons. As a result the security bootstrapping can be devided into the bootstrapping of joining RFD and bootstrapping of joining FFD.

5.1. Bootstrapping of joining RFD via configured FFD

A joining RFD broadcasts beacon frames to neighboring nodes. The configured FFDs that receive the beacon frames, decide whether allowing the RFD associating at link layer. A RFD may receive multiple replies from more than one configured FFDs. It can select one device to associate, e.g. the first one that replies or the one with strongest received power strength. The joining RFD then presents credential to the associated configured FFD, which is forwarded to 6LBR and TC to be validated. If the validation is successful, the IP address and network key are generated and delivered to the RFD. Otherwise if the validation fails, the FFDrefuses adding the RFD to its domain.

```
   Joining RFD                    Configured FFD     6LBR             TC
    |                                 |              |                 |
    |           Beaconing             |              |                 |
    |-------------------------------->|              |                 |
    |                                 |              |                 |
    |           IEEE 802.15.4         |              |                 |
    |     MAC unsecure association    |              |                 |
    |<------------------------------->|              |                 |
    |                                 |              |                 |
    |                                 |              |                 |
    |        Authentication           |     Relay    |   Auth.check    |
    |<------------------------------->|<------------>|<--------------->|
    |   Network key and IP address    |              |   IP address    |
    |                                 |              |                 |
    |                                 |              |                 |
    |     Further Configuration       |              |                 |
    |<-------------------------------- ------------->|                 |
    |                                 |              |                 |
    |                                 |              |                 |
```

  Figure 5

5.2. Bootstrapping of joining FFD via configured FFD/6LBR

A joining FFD broadcasts beacon frames to neighboring nodes. The configured FFDs that receive the beacon frames, decide whether allowing the FFD associating at link layer. A FFD may receive multiple replies from more than one configured FFDs or directly from the 6LBR. It can select one device to associate, e.g. the first one that replies or the one with strongest received power strength. The joining FFD then presents credential to the associated configured FFD/6LBR, which is forwarded to TC to be validated. If the validation is successful, the IP address and network key are generated and delivered to the FFD. Further configurations such as routing protocol can be realized afterwards. Otherwise if the validation fails, the 6LBR refuses adding the FFD to its domain.

```
                              +---------------------------+
   Joining FFD                | Configured FFD      6LBR  |           TC
   |                          +------+--------------+-----+            |
   |           Beaconing             |              |                  |
   |-------------------------------->|              |                  |
   |                                 |              |                  |
   |           IEEE 802.15.4         |              |                  |
   |     MAC unsecure association    |              |                  |
   |<------------------------------->|              |                  |
   |                                 |              |                  |
   |                                 |              |                  |
   |         Authentication          |     Relay    |     Auth.check   |
   |<------------------------------->|<------------>|<---------------->|
   |   Network key and IP address    |              |   IP address     |
   |                                 |              |                  |
   |                                 |              |                  |
   |     Further Configuration       |              |                  |
   |<-------------------------------- ------------->|                  |
   |                                 |              |                  |
   |                                 |              |                  |
```

  Figure 6

6. Security Considerations

   TBD

7. Acknowledgement

   TBD

8. References

8.1. Normative References

[IEEE802.15.4]
IEEE Standard, , "IEEE Std. 802.15.4-2011", October 2011, <http://standards.ieee.org/findstds/standard/802.15.4-2011.html>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, September 2007.

[RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, August 2007.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.

[RFC6550] Winter, T., Thubert, P., Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, JP., and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks", RFC 6550, March 2012.

[RFC7400] Bormann, C., "6LoWPAN-GHC: Generic Header Compression for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 7400, November 2014.

8.2. Informative References

[I-D.pritikin-anima-bootstrapping-keyinfra]
Pritikin, M., Behringer , M., and S. Bjarnason , "Bootstrapping Key Infrastructures", November 2014.

[I-D.struik-6tisch-security-considerations]
Struik , R., "6TiSCH Security Architectural Considerations", January 2015.

Authors’ Addresses

Danping He
Huawei

Email: ana.hedanping@huawei.com

Behcet Sarikaya
Huawei USA
5340 Legacy Dr. Building 3
Plano, TX 75024

Email: sarikaya@ieee.org

Network Working Group
Internet-Draft
Intended status: Informational
Expires: April 21, 2016

A. Keranen
Ericsson
M. Kovatsch
ETH Zurich
K. Hartke
Universitaet Bremen TZI

October 19, 2015

RESTful Design for Internet of Things Systems
draft-keranen-t2trg-rest-iot-00

Abstract

This document gives guidance for designing Internet of Things (IoT) systems that follow the principles of the Representational State Transfer (REST) architectural style.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The Representational State Transfer (REST) architectural style [REST] is a set of guidelines and best practices for building distributed hypermedia systems.

When REST principles are applied to a design of a system, the result is often called RESTful and in particular an API following these principles is called a RESTful API.

Different protocols can be used with RESTful systems, but at the time of writing the most common protocols are HTTP [RFC7231] and CoAP [RFC7252].

RESTful design facilitates many desirable features for a system, such as good scaling properties. RESTful APIs are also often simple and lightweight and hence easy to use also with various IoT applications. The goal of this document is to give basic guidance for designing RESTful systems and APIs for IoT applications and give pointers for more information.

2. Terminology

This section explains some of the common terminology that is used in the context of RESTful design for IoT systems.

Application State: The set of pending requests, history of requests, bookmarks (URIs stored for later retrieval), and application-specific state that the client keeps between requests.

Cache: A local store of response messages and the subsystem that controls storage, retrieval, and deletion of messages in it.

Client: A node that sends requests to servers and receives responses.

Content Negotiation: The practice of determining the "best" representation for a client when examining the current state of a resource.

Form: A hypermedia control that enables a client to change the state of a resource.

Forward Proxy: An intermediary that is selected by a client, usually via local configuration rules, and that can be tasked to make requests on behalf of the client. This may be useful, for example, when the client lacks the capability to make the request itself, or to service the response from a cache in order to reduce response time and network bandwidth or energy consumption.

Gateway: See "Reverse Proxy".

Hypermedia Control: A component embedded in a representation that describes a request. By performing the request, the client can change resource state and/or move the application state forward.

Idempotent Method: A method where multiple identical requests with that method lead to the same visible resource state as a single such request. For example, the PUT method replaces the state of a resource with a new state; replacing the state multiple times with the same new state still results in the same result.

Link: A hypermedia control that enables a client to navigate between resources and thereby change the application state.

Media Type: A sequence of characters such as "text/html" or "application/json" that is used to label representations so that it is known how the representation should be interpreted, and how it is encoded.

Method: A procedure associated with a resource. Common methods include GET, PUT, POST, and DELETE (see Section 3.5 for details).

Origin Server: A server that is the definitive source for representations of its resources and the ultimate recipient of any request that intends to modify its resources. In contrast, intermediaries (such as proxies caching a representation) can assume the role of a server, but are not the source for representations as these are acquired from the origin server.

Proactive Content Negotiation: A content negotiation mechanism where the server selects a representation based on the client’s content negotiation preferences. For example, in an IoT application, the preferences of a client could be media types "application/senml+json" and "text/plain".

Reactive Content Negotiation: A content negotiation mechanism where the client selects a representation from a list of available representations. The list may, for example, be included by a server in an initial response. If the user agent is not satisfied by the initial response representation, it can request one or more of the alternative representations, selected based on metadata included in the list.

Representation Format: A set of rules for encoding information in a sequence of bytes. In the Web, the most prevalent representation format is HTML. Other common formats include plain text (in UTF-8 or any other encoding), JSON or XML. With IoT systems, often compact formats such as JSON, CBOR, and EXI are used.

Representation: A sequence of bytes, plus representation metadata, that captures the current or intended state of a resource and that can be transferred between clients and servers (possibly via one or more intermediaries).

Representational State Transfer (REST): An architectural style for Internet-scale distributed hypermedia systems.

Resource State: A mapping of a resource to a set of values that may change over time.

Resource: An item of interest identified by a URI. Anything that can be named can be a resource. A resource often encapsulates a piece of state in a system. Typical resources in an IoT system can be, e.g., a sensor, the current value of a sensor, the location of a device, or the current state of an actuator.

Reverse Proxy: An intermediary that appears as a server towards the client but satisfies the requests by forwarding them to the actual server (possibly via one or more other intermediaries). A reverse proxy is often used to encapsulate legacy services, to improve server performance through caching, and to enable load balancing across multiple machines.

Safe Method: A method that does not result in any state change on the origin server when applied to a resource. For example, the GET method only returns a representation of the resource state but does not change the resource.

Server: A node that listens for requests, applies the requested actions to resources, and sends responses back to the clients.

Uniform Resource Identifier (URI): A global identifier for resources. See Section 3.4 for more details.

3. Basics

3.1. Architecture

The components of a REST system are assigned one of two roles: client or server. User agents are always in the client role and have the initiative to issue requests. Intermediaries (such as forward proxies and reverse proxies) implement both roles, but only forward requests to other intermediaries or origin servers. They can also translate requests to different protocols, for instance, CoAP-HTTP cross-proxies.

Note that the terms "client" and "server" refer only to the roles that the nodes assume for a particular message exchange. The same node might act as a client in some communications and a server in others.

```
 ________                     _________
|        |                   |         |
| User  (C)-------------------(S) Origin |
| Agent  |                   |  Server |
|________|                   |_________|
(Browser)                    (Web Server)
```

Figure 1: Client-Server Communication

```
    ________       __________                        _________
   |        |     |          |                      |         |
   | User  (C)---(S) Inter- (C)--------------------(S) Origin |
   | Agent  |     |  mediary |                      |  Server |
   |________|     |__________|                      |_________|
   (Browser)     (Forward Proxy)                    (Web Server)
```

Figure 2: Communication with Forward Proxy

Reverse proxies are usually imposed by the origin server. In addition to the features of a forward proxy, they can also provide an interface for non-RESTful services such as legacy systems or alternative technologies such as Bluetooth ATT/GATT. This property is enforced by the layered system constraint of REST, which says that a client cannot see beyond the server it is connected to.

```
    ________                        __________       _________
   |        |                      |          |     |         |
   | User  (C)--------------------(S) Inter- (x)---(x) Origin |
   | Agent  |                      |  mediary |     |  Server |
   |________|                      |__________|     |_________|
   (Browser)                     (Reverse Proxy)   (Legacy System)
```

Figure 3: Communication with Reverse Proxy

Nodes in IoT systems often implement both roles. Unlike intermediaries, however, they can take the initiative as a client (e.g., to register with a directory, such as CoAP Resource Directory) and act as origin server at the same time (e.g., to serve sensor values).

```
    ________                                         _________
   |        |                                       |         |
   | Thing (C)-------------------------------------(S) Origin |
   |       (S)                                      |  Server |
   |________| \                                     |_________|
    (Sensor)   \   ________                        (Resource Directory)
                \ |        |
                 (C) Thing |
                  |________|
                 (Controller)
```

Figure 4: Constrained RESTful environments

3.2. System design

When designing a REST system, the state of the distributed application must be assigned to the different components. Here, it is important to distinguish between "session state" and "resource state".

Session state encompasses the control flow and the interactions between the components (see Section 2). Following the statelessness constraint, the session state must be kept only on clients. On the one hand, this makes requests a bit more verbose since every request must contain all the information necessary to process it. On the other hand, this makes servers efficient, since they do not have to keep any state about their clients. Requests can easily be distributed over multiple worker threads or server instances. For the IoT systems, it lowers the memory requirements for server implementations, which is particularly important for constrained servers and servers serving large amount of clients.

Resource state includes the more persistent data of an application (i.e., independent of the application control flow). This can be static data such as device descriptions, persistent data such as system configuration, but also dynamic data such as the current value of a sensor on a thing.

3.3. Resource modeling

Important part of RESTful API design is to model the system as a set of resources whose state can be retrieved and/or modified and where resources can be potentially also created and/or deleted.

Resource representations have a media type that tells how the representation should be interpreted. Typical media types for IoT systems include "text/plain" for simple UTF-8 text, "application/octet-stream" for arbitrary binary data, "application/json" for JSON [RFC7159], "application/senml+json" [I-D.jennings-core-senml] for Sensor Markup Language (SenML) formatted data, "application/cbor" for CBOR [RFC7049], "application/exi" for EXI [W3C.REC-exi-20110310]. Full list of registered internet media types is available at the IANA registry [IANA-media-types] and media types registered for use with CoAP are listed at CoAP Content-Formats IANA registry [IANA-CoAP-media].

3.4. Uniform Resource Identifiers (URIs)

Uniform Resource Identifiers (URIs) are used to interact with a resource, to reference a resource from another resource, to advertise or bookmark a resource, or to index a resource by search engines.

```
     foo://example.com:8042/over/there?name=ferret#nose
     \_/   \______________/\_________/ \_________/ \__/
      |           |            |            |        |
   scheme     authority       path        query   fragment
```

A URI is a sequence of characters that matches the syntax defined in [RFC3986]. It consists of a hierarchical sequence of five components: scheme, authority, path, query, and fragment (from most significant to least significant). A scheme creates a namespace for resources and defines how the following components identify a resource within that namespace. The authority identifies an entity that governs part of the namespace, such as the server "www.example.org" in the "http" scheme. A host name (e.g., a fully qualified domain name) or an IP address, potentially followed by a transport layer port number, are usually used in the authority component for the "http" and "coap" schemes. The path and query contain data to identify a resource within the scope of the URI's scheme and naming authority. The path is hierarchical; the query is non-hierarchical. The fragment allows to refer to some portion of the resource, such as a section in an HTML document.

For RESTful IoT applications, typical schemes include "https", "coaps", "http", and "coap". These refer to HTTP and CoAP, with and without Transport Layer Security (TLS) [RFC5246]. (CoAP uses Datagram TLS (DTLS) [RFC6347], the variant of TLS for UDP.) These four schemes also provide means for locating the resource; using the HTTP protocol for "http" and "https", and with the CoAP protocol for "coap" and "coaps". If the scheme is different for two URIs (e.g., "coap" vs. "coaps"), it is important to note that even if the rest of the URI is identical, these are two different resources, in two distinct namespaces.

The query parameters can be used to parametrize the resource. For example, a GET request may use query parameters to request the server to send only certain kind data of the resource (i.e., filtering the response). Query parameters in PUT and POST requests do not have such established semantics and are not commonly used.

3.5. HTTP/CoAP Methods

Section 4.3 of [RFC7231] defines the set of methods in HTTP; Section 5.8 of [RFC7252] defines the set of methods in CoAP. The following lists the most relevant methods and gives a short explanation of their semantics.

3.5.1. GET

The GET method requests a current representation for the target resource. Only the origin server needs to know how each of its resource identifiers corresponds to an implementation and how each implementation manages to select and send a current representation of the target resource in a response to GET.

A payload within a GET request message has no defined semantics.

A response to a successful GET request is cacheable; a cache may use it to satisfy future, equivalent GET requests. The GET method is safe and idempotent.

3.5.2. POST

The POST method requests that the target resource process the representation enclosed in the request according to the resource’s own specific semantics.

If one or more resources has been created on the origin server as a result of successfully processing a POST request, the origin server sends a 201 (Created) response containing a Location header field that provides an identifier for the resource created and a representation that describes the status of the request while referring to the new resource(s).

The POST method is not safe nor idempotent.

3.5.3. PUT

The PUT method requests that the state of the target resource be created or replaced with the state defined by the representation enclosed in the request message payload. A successful PUT of a given representation would suggest that a subsequent GET on that same target resource will result in an equivalent representation being sent.

The fundamental difference between the POST and PUT methods is highlighted by the different intent for the enclosed representation. The target resource in a POST request is intended to handle the enclosed representation according to the resource’s own semantics, whereas the enclosed representation in a PUT request is defined as replacing the state of the target resource. Hence, the intent of PUT is idempotent and visible to intermediaries, even though the exact effect is only known by the origin server.

The PUT method is not safe, but is idempotent.

3.5.4. DELETE

The DELETE method requests that the origin server remove the association between the target resource and its current functionality.

If the target resource has one or more current representations, they might or might not be destroyed by the origin server, and the associated storage might or might not be reclaimed, depending entirely on the nature of the resource and its implementation by the origin server.

The DELETE method is not safe, but is idempotent.

3.6. HTTP/CoAP Status/Response Codes

Section 6 of [RFC7231] defines a set of Status Codes in HTTP that are used by application to indicate whether a request was understood and satisfied, and how to interpret the answer. Similarly, Section 5.9 of [RFC7252] defines the set of Response Codes in CoAP.

The status codes consist of three digits (e.g., "404" or "4.04") where the first digit expresses the class of the code. Implementations do not need to understand all status codes, but the class of the code must be understood. Codes starting with 1 are informational; the request was received and being processed. Codes starting with 2 indicate successful request. Codes starting with 3 indicate redirection; further action is needed to complete the request. Codes stating with 4 and 5 indicate errors. The codes starting with 4 mean client error (e.g., bad syntax in request) whereas codes starting with 5 mean server error; there was no apparent problem with the request but server was not able to fulfill the request.

Responses may be stored in a cache to satisfy future, equivalent requests. HTTP and CoAP use two different patterns to decide what responses are cacheable. In HTTP, the cacheability of a response depends on the request method (e.g., responses returned in reply to a GET request are cacheable). In CoAP, the cacheability of a response depends on the response code (e.g., responses with code 2.04 are cacheable). This difference also leads to slightly different semantics for the codes starting with 2; for example, CoAP does not have a 2.00 response code.

4. Security Considerations

This document does not define new functionality and therefore does not introduce new security concerns. However, security consideration from related specifications apply to RESTful IoT design. These include:

- o HTTP security: Section 9 of [RFC7230], Section 9 of [RFC7231], etc.

- o CoAP security: Section 11 of [RFC7252]

- o URI security: Section 7 of [RFC3986]

5. Acknowledgement

The authors would like to thank Mert Ocak for the review comments.

6. References

6.1. Normative References

[REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine , 2000.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <http://www.rfc-editor.org/info/rfc7049>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

[W3C.REC-exi-20110310]
Schneider, J. and T. Kamiya, "Efficient XML Interchange (EXI) Format 1.0", World Wide Web Consortium Recommendation REC-exi-20110310, March 2011, <http://www.w3.org/TR/2011/REC-exi-20110310>.

6.2. Informative References

[I-D.jennings-core-senml]
Jennings, C., Shelby, Z., Arkko, J., and A. Keranen, "Media Types for Sensor Markup Language (SENML)", draft-jennings-core-senml-01 (work in progress), July 2015.

[IANA-CoAP-media]
"CoAP Content-Formats", n.d., <http://www.iana.org/assignments/core-parameters/core-parameters.xhtml#content-formats>.

[IANA-media-types]
"Media Types", n.d., <http://www.iana.org/assignments/media-types/media-types.xhtml>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <http://www.rfc-editor.org/info/rfc7252>.

Appendix A. Future Work

o More details on the definition of application state. Is server involved and to what extent.

o Discuss design patterns, such as "Observing state (asynchronous updates) of a resource", "Executing a Function", "Events as State", "Conversion", "Collections", "robust communication in network with high packet loss", "unreliable (best effort) communication", "3-way commit", etc.

o Discuss directories, such as CoAP Resource Directory

- More information on how to design resources; choosing what is modeled as a resource, etc.

Authors' Addresses

Ari Keranen
Ericsson
Jorvas 02420
Finland

Email: ari.keranen@ericsson.com

Matthias Kovatsch
ETH Zurich
Universitaetstrasse 6
Zurich CH-8092
Switzerland

Email: kovatsch@inf.ethz.ch

Klaus Hartke
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Email: hartke@tzi.org

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 21, 2016

J. Mattsson
J. Fornehed
G. Selander
F. Palombini
Ericsson

October 19, 2015

Controlling Actuators with CoAP
draft-mattsson-core-coap-actuators-00

Abstract

Being able to trust information from sensors and to securely control actuators is essential in a world of connected and networking things interacting with the physical world. In this memo we show that just using COAP with a security protocol like DTLS or OSCOAP is not enough. We describe several serious attacks any on-path attacker can do, and discuss tougher requirements and mechanisms to mitigate the attacks. While this document is focused on actuators, one of the attacks applies equally well to sensors using DTLS.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Being able to trust information from sensors and to securely control actuators is essential in a world of connected and networking things interacting with the physical world. One protocol used to interact with sensors and actuators is the Constrained Application Protocol (CoAP). Any Internet-of-Things (IoT) deployment valuing security and privacy would use a security protocol such as DTLS [RFC6347] or OSCOAP [I-D.selander-ace-object-security] to protect CoAP, but we show that this is not enough. We describe several serious attacks any on-path attacker (i.e. not only "trusted" intermediaries) can do, and discusses tougher requirements and mechanisms to mitigate the attacks. The request delay attack (valid for both DTLS and OSCOAP and described in Section 2.2) lets an attacker control an actuator at a much later time than the client anticipated. The response delay and mismatch attack (valid for DTLS and described in Section 2.3) lets an attacker respond to a client with a response meant for an older request. In Section 3, a new CoAP Option, the Repeat Option, mitigating the delay attack in specified.

2. Attacks

Internet-of-Things (IoT) deployments valuing security and privacy, MUST use a security protocol such as DTLS or OSCOAP to protect CoAP. This is especially true for deployments of actuators where attacks often (but not always) have serious consequences. The attacks described in this section are made under the assumption that CoAP is already protected with a security protocol such as DTLS or OSCOAP, as an attacker otherwise can easily forge false requests and responses.

2.1. The Block Attack

An on-path attacker can block the delivery of any number of requests or responses. The attack can also be performed by an attacker jamming the lower layer radio protocol. This is true even if a security protocol like DTLS or OSCOAP is used. Encryption makes selective blocking of messages harder, but not impossible or even infeasible. With DTLS, proxies have access to the complete CoAP message, and with OSCOAP, the CoAP header and several CoAP options are not encrypted. In both security protocols, the IP-addresses, ports, and CoAP message lengths are available to all on-path attackers, which may be enough to determine the server, resource, and command. The block attack is illustrated in Figure 1 and 2.

```
                 Client   Foe   Server
                    |      |      |
                    +----->X      |      Code: 0.03 (PUT)
                    | PUT  |      |     Token: 0x47
                    |      |      |  Uri-Path: lock
                    |      |      |   Payload: 1 (Lock)
                    |      |      |
```

Figure 1: Blocking a Request

Where ’X’ means the attacker is blocking delivery of the message.

```
               Client   Foe   Server
                  |      |      |
                  +------------>|      Code: 0.03 (PUT)
                  |      | PUT  |     Token: 0x47
                  |      |      |  Uri-Path: lock
                  |      |      |   Payload: 1 (Lock)
                  |      |      |
                  |      X<-----+      Code: 2.04 (Changed)
                  |      | 2.04 |     Token: 0x47
                  |      |      |
```

Figure 2: Blocking a Response

While blocking requests to, or responses from, a sensor is just a denial of service attack, blocking a request to, or a response from, an actuator results in the client losing information about the server’s status. If the actuator e.g. is a lock (door, car, etc.), the attack results in the client not knowing (except by using out-of-band information) whether the lock is unlocked or locked, just like the observer in the famous Schroedinger's cat thought experiment. Due to the nature of the attack, the client cannot distinguish the attack from connectivity problems, offline servers, or unexpected behavior from middle boxes such as NATs and firewalls.

Remedy: In actuator deployments where confirmation is important, the application MUST notify the user upon reception of the response, or warn the user when a response is not received. The application SHOULD also indicate to the user that the status of the actuator is now uncertain.

2.2. The Request Delay Attack

An on-path attacker may not only block packets, but can also delay the delivery of any packet (request or response) by a chosen amount of time. This is true even if DTLS or OSCOAP is used, as long as the delayed packet is delivered inside the replay window. The replay window has a default length of 64 in DTLS and is application dependent in OSCOAP. The attacker can control the replay window by blocking some or all other packets. By first delaying a request, and then later, after delivery, blocking the response to the request, the client is not made aware of the delayed delivery except by the missing response. The server has in general, no way of knowing that the request was delayed and will therefore happily process the request.

If some wireless low-level protocol is used, the attack can also be performed by the attacker simultaneously recording what the client transmits while at the same time jamming the server. The request delay attack is illustrated in Figure 3.

```
                Client   Foe   Server
                   |      |      |
                   +----->@      |      Code: 0.03 (PUT)
                   | PUT  |      |     Token: 0x9c
                   |      |      |  Uri-Path: lock
                   |      |      |   Payload: 0 (Unlock)
                   |      |      |
                     ....   ....
                   |      |      |
                   |      @----->|      Code: 0.03 (PUT)
                   |      | PUT  |     Token: 0x9c
                   |      |      |  Uri-Path: lock
                   |      |      |   Payload: 0 (Unlock)
                   |      |      |
                   |      X<-----+      Code: 2.04 (Changed)
                   |      | 2.04 |     Token: 0x9c
                   |      |      |
```

Figure 3: Delaying a Request

Where '@' means the attacker is storing and later forwarding the message (@ may alternatively be seen as a wormhole connecting two points in spacetime).

While an attacker delaying a request to a sensor is often not a security problem, an attacker delaying a request to an actuator performing an action is often a serious problem. A request to an actuator (for example a request to unlock a lock) is often only meant to be valid for a short time frame, and if the request does not reach the actuator during this short timeframe, the request should not be fulfilled. In the unlock example, if the client does not get any response and does not physically see the lock opening, the user is likely to walk away, calling the locksmith (or the IT-support).

If a non-zero replay window is used (the default in DTLS and unspecified in OSCOAP), the attacker can let the client interact with the actuator before delivering the delayed request to the server (illustrated in Figure 4). In the lock example, the attacker may store the first "unlock" request for later use. The client will likely resend the request with the same token. If DTLS is used, the resent packet will have a different sequence number and the attacker can forward it. If OSCOAP is used, resent packets will have the same sequence number and the attacker must block them all until the client sends a new message with a new sequence number (not shown in Figure 4). After a while when the client has locked the door again, the attacker can deliver the delayed "unlock" message to the door, a very serious attack.

```
                Client   Foe   Server
                   |      |      |
                   +----->@      |      Code: 0.03 (PUT)
                   | PUT  |      |     Token: 0x9c
                   |      |      |  Uri-Path: lock
                   |      |      |   Payload: 0 (Unlock)
                   |      |      |
                   +------------>|      Code: 0.03 (PUT)
                   | PUT  |      |     Token: 0x9c
                   |      |      |  Uri-Path: lock
                   |      |      |   Payload: 0 (Unlock)
                   |      |      |
                   <-------------+      Code: 2.04 (Changed)
                   |      | 2.04 |     Token: 0x9c
                   |      |      |
                     ....   ....
                   |      |      |
                   +------------>|      Code: 0.03 (PUT)
                   | PUT  |      |     Token: 0x7a
                   |      |      |  Uri-Path: lock
                   |      |      |   Payload: 1 (Lock)
                   |      |      |
                   <-------------+      Code: 2.04 (Changed)
                   |      | 2.04 |     Token: 0x7a
                   |      |      |
                   |      @----->|      Code: 0.03 (PUT)
                   |      | PUT  |     Token: 0x9c
                   |      |      |  Uri-Path: lock
                   |      |      |   Payload: 0 (Unlock)
                   |      |      |
                   |      X<-----+      Code: 2.04 (Changed)
                   |      | 2.04 |     Token: 0x9c
                   |      |      |
```

Figure 4: Delaying Request with Reordering

While the second attack (Figure 4) can be mitigated by using a replay window of length zero, the first attack (Figure 3) cannot. A solution must enable the server to verify that the request was received within a certain time frame after it was sent. This can be accomplished with either a challenge-response pattern or by exchanging timestamps. Security solutions based on timestamps require exactly synchronized time, and this is hard to control with complications such as time zones and daylight saving. Even if the clocks are synchronized at one point in time, they may easily get out-of-sync and an attacker may even be able to affect the client or the server time in various ways such as setting up a fake NTP server, broadcasting false time signals to radio controlled clocks, or expose one of them to a strong gravity field. As soon as client falsely believes it is time synchronized with the server, delay attacks are possible. A challenge response mechanism is much more failure proof and easy to analyze. One such mechanism, the CoAP Repeat Option, is specified in Section 3.

Remedy: The CoAP Repeat Option specified in Section 3 SHALL be used for controlling actuators unless another application specific challenge-response or timestamp mechanism is used.

2.3. The Response Delay and Mismatch Attack

The following attack can be performed if CoAP is protected by a security protocol where the response is not bound to the request in any way except by the CoAP token. This would include most general security protocols, such as DTLS and IPsec, but not OSCOAP. The attacker performs the attack by delaying delivery of a response until the client sends a request with the same token. As long as the response is inside the replay window (which the attacker can make sure by blocking later responses), the response will be accepted by the client as a valid response to the later request. CoAP [RFC7252] does not give any guidelines for the use of token with DTLS, except that the tokens currently "in use" SHOULD (not SHALL) be unique.

The attack can be performed by an attacker on the wire, or an attacker simultaneously recording what the server transmits while at the same time jamming the client. The response delay and mismatch attack is illustrated in Figure 5.

```
             Client   Foe   Server
                |      |      |
                +------------>|      Code: 0.03 (PUT)
                | PUT  |      |     Token: 0x77
                |      |      |  Uri-Path: lock
                |      |      |   Payload: 0 (Unlock)
                |      |      |
                |      @<-----+      Code: 2.04 (Changed)
                |      | 2.04 |     Token: 0x77
                |      |      |
                  ....   ....
                |      |      |
                +----->X      |      Code: 0.03 (PUT)
                | PUT  |      |     Token: 0x77
                |      |      |  Uri-Path: lock
                |      |      |   Payload: 0 (Lock)
                |      |      |
                <------@      |      Code: 2.04 (Changed)
                | 2.04 |      |     Token: 0x77
                |      |      |
```

Figure 5: Delaying and Mismatching Response to PUT

If we once again take a lock as an example, the security consequences may be severe as the client receives a response message likely to be interpreted as confirmation of a locked door, while the received response message is in fact confirming an earlier unlock of the door. As the client is likely to leave the (believed to be locked) door unattended, the attacker may enter the home, enterprise, or car protected by the lock.

The same attack may be performed on sensors, also this with serious consequences. As illustrated in Figure 6, an attacker may convince the client that the lock is locked, when it in fact is not. The "Unlock" request may be also be sent by another client authorized to control the lock.

```
             Client   Foe   Server
                |      |      |
                +------------>|      Code: 0.01 (GET)
                | GET  |      |     Token: 0x77
                |      |      |  Uri-Path: lock
                |      |      |
                |      @<-----+      Code: 2.05 (Content)
                |      | 2.05 |     Token: 0x77
                |      |      |   Payload: 1 (Locked)
                |      |      |
                +------------>|      Code: 0.03 (PUT)
                | PUT  |      |     Token: 0x34
                |      |      |  Uri-Path: lock
                |      |      |   Payload: 1 (Unlock)
                |      |      |
                |      X<-----+      Code: 2.04 (Changed)
                |      | 2.04 |     Token: 0x34
                |      |      |
                +----->X      |      Code: 0.01 (GET)
                | GET  |      |     Token: 0x77
                |      |      |  Uri-Path: lock
                |      |      |
                <------@      |      Code: 2.05 (Content)
                | 2.05 |      |     Token: 0x77
                |      |      |   Payload: 1 (Locked)
                |      |      |
```

Figure 6: Delaying and Mismatching Response to GET

As illustrated in Figure 7, an attacker may even mix responses from different resources as long as the two resources share the same DTLS connection on some part of the path towards the client. This can happen if the resources are located behind a common gateway, or are served by the same CoAP proxy. An on-path attacker (not necessarily a DTLS endpoint such as a proxy) may e.g. deceive a client that the living room is on fire by responding with an earlier delayed response from the oven (temperatures in degree Celsius).

```
       Client   Foe   Server
          |      |      |
          +------------>|      Code: 0.01 (GET)
          | GET  |      |     Token: 0x77
          |      |      |  Uri-Path: oven/temperature
          |      |      |
          |      @<-----+      Code: 2.05 (Content)
          |      | 2.05 |     Token: 0x77
          |      |      |   Payload: 225
          |      |      |
            ....   ....
          |      |      |
          +----->X      |      Code: 0.01 (GET)
          | GET  |      |     Token: 0x77
          |      |      |  Uri-Path: livingroom/temperature
          |      |      |
          <------@      |      Code: 2.05 (Content)
          | 2.05 |      |     Token: 0x77
          |      |      |   Payload: 225
          |      |      |
```

Figure 7: Delaying and Mismatching Response from other resource

OSCOAP is not susceptible to these attacks since it provides a secure binding between request and response messages.

Remedy: If CoAP is protected with a security protocol not providing bindings between requests and responses (e.g. DTLS) the client MUST NOT reuse any tokens for a given source/destination which the client has not received responses to. The easiest way to accomplish this is to implement the token as a counter and never reuse any tokens at all, this approach SHOULD be followed.

2.4. The Relay Attack

Yet another type of attack can be performed in deployments where actuator actions are triggered automatically based on proximity and without any user interaction, e.g. a car (the client) constantly polling for the car key (the server) and unlocking both doors and engine as soon as the car key responds. An attacker (or pair of attackers) may simply relay the CoAP messages out-of-band, using for examples some other radio technology. By doing this, the actuator (i.e. the car) believes that the client is close by and performs actions based on that false assumption. The attack is illustrated in Figure 8. In this example the car is using an application specific challenge-response mechanism transferred as CoAP payloads.

```
   Client   Foe         Foe   Server
      |      |           |      |
      +----->| ......... +----->|      Code: 0.02 (POST)
      | POST |           | POST |     Token: 0x3a
      |      |           |      |  Uri-Path: lock
      |      |           |      |   Payload: JwePR2iCe8b0ux (Challenge)
      |      |           |      |
      |<-----+ ......... |<-----+      Code: 2.04 (Changed)
      | 2.04 |           | 2.04 |     Token: 0x3a
      |      |           |      |   Payload: RM8i13G8D5vfXK (Response)
      |      |           |      |
```

Figure 8: Relay Attack (the client is the actuator)

The consequences may be severe, and in the case of a car, lead to the attacker unlocking and driving away with the car, an attack that unfortunately is happening in practice.

Remedy: Getting a response over a short-range radio MUST NOT be taken as proof of proximity and therefore MUST NOT be used to take actions based on such proximity. Any automatically triggered mechanisms relying on proximity MUST use other stronger mechanisms to guarantee proximity. Mechanisms that MAY be used are: measuring the round-trip time and calculate the maximum possible distance based on the speed of light, or using radio with an extremely short range like NFC (centimeters instead of meters). Another option is to including geographical coordinates (from e.g. GPS) in the messages and calculate proximity based on these, but in this case the location measurements MUST be very precise and the system MUST make sure that an attacker cannot influence the location estimation, something that is very hard in practice.

3. The Repeat Option

The Repeat Option is a challenge-response mechanism for CoAP, binding a resent request to an earlier 4.03 forbidden response. The challenge (for the client) is simply to echo the Repeat Option value in a new request. The Repeat Option enables the server to verify the freshness of a request, thus mitigating the Delay Attack described in Section 2.2. An example message flow is illustrated in Figure 9.

```
                Client  Server
                   |       |
                   +------>|        Code: 0.03 (PUT)
                   | PUT   |       Token: 0x41
                   |       |    Uri-Path: lock
                   |       |     Payload: 0 (Unlock)
                   |       |
                   |<------+ t0     Code: 4.03 (Forbidden)
                   | 4.03 |       Token: 0x41
                   |       |      Repeat: 0x6c880d41167ba807
                   |       |
                   +------>| t1     Code: 0.03 (PUT)
                   | PUT   |       Token: 0x42
                   |       |    Uri-Path: lock
                   |       |      Repeat: 0x6c880d41167ba807
                   |       |     Payload: 0 (Unlock)
                   |       |
                   |<------+        Code: 2.04 (Changed)
                   | 2.04 |       Token: 0x42
                   |       |
```

Figure 9: The Repeat Option

The Repeat Option may be used for all Methods and Response Codes. In responses, the value MUST be a (pseudo-)random bit string with a length of at least 64 bits. A new (pseudo-)random bit string MUST be generated for each response. In requests, the Repeat Option MUST echo the value from a previously received response.

The Repeat Option is critical, Safe-to-Forward, not part of the Cache-Key, and not repeatable.

Upon receiving a request without the Repeat Option to a resource with freshness requirements, the server sends a 4.03 Forbidden response with a Repeat Option and stores the option value and the response transmit time t0.

Upon receiving a 4.03 Forbidden response with the Repeat Option, the client SHOULD resend the request, echoing the Repeat Option value.

Upon receiving a request with the Repeat Option, the server verifies that the option value equals the previously sent value; otherwise the request is not processed further. The server calculates the round-trip time RTT = (t1 - t0), where t1 is the request receive time. The server MUST only accept requests with a round-trip time below a certain threshold T, i.e. RTT < T, otherwise the request is not processed further, and an error message MAY be sent. The threshold T is application specific.

An attacker able to control the server’s clock with high precision, could still be able to perform a delay attack by moving the server’s clock back in time, thus making the measured round-trip time smaller than the actual round-trip time. The times t0 and t1 MUST therefore be measured with a steady clock (one that cannot be adjusted).

EDITORS NOTE: The mechanism described above gives the server freshness guarantee independently of what the client does. The disadvantages are that the mechanism always takes two round-trips and that the server has to save the option value and the time t0. Other solutions involving time may be discussed:

o The server may simply send the client the current time in its timescale, i.e. a timestamp (option value = t0). The client may then use this timestamp to estimate the current time in the servers timescale when sending future requests (i.e. not echoing). This approach has the benefit of reducing round-trips and server state, but has the security problems discussed in Section 2.2.

o The server may instead of a pseudorandom value send an encrypted timestamp (option value = E(k, t0)). CTR-mode would from a security point be like sending (value = t0). ECB-mode or CCM-mode would work, but would expand the value length. With CCM, the server might also bind the option value to request (value = AEAD(k, t0, parts of request)). This approach does not reduce the number of round-trips but eliminates server state.

4. IANA Considerations

This document defines the following Option Number, whose value have been assigned to the CoAP Option Numbers Registry defined by [RFC7252].

Number	Name
29	Repeat

5. Security Considerations

The whole document can be seen as security considerations for CoAP.

6. References

6.1. Normative References

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <http://www.rfc-editor.org/info/rfc7252>.

6.2. Informative References

[I-D.selander-ace-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "June 29, 2015", draft-selander-ace-object-security-02 (work in progress), June 2015.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

Authors’ Addresses

John Mattsson
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: john.mattsson@ericsson.com

John Fornehed
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: john.fornehed@ericsson.com

Goran Selander
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: goran.selander@ericsson.com

Francesca Palombini
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

Internet Engineering Task Force
INTERNET-DRAFT
Intended Status: Standard Track
Expires: April 19, 2016

Vasu K
Rahul A J
Yangneng
Huawei

Oct 19, 2015

Service Provisioning for Constrained Devices
draft-vasu-core-ace-service-provisioning-00

Abstract

As more constrained devices are integrating with current Internet, the ubiquitous computing in scenarios like smart home is very important. In smart home, the constrained devices (ex. thermostat) need to be provisioned in such a way that it can inter-operate with any kind of devices like other constrained devices (ex. Air conditioner) or client devices (ex. smart phone). This document provides a method to support service provisioning based on pre-configured admission and resource control policies, where this method explains device's service access in two different use cases: first provisioning the service when a constrained device accessing the service provided by other constrained device, second, accessing the service provided by constrained device from the client device (non constrained device).

Copyright and License Notice

Table of Contents

1 Introduction

The work on Constrained Restful Environment (CoRE) aimed to realize the restful architecture for constrained devices [RFC7228] in constrained networks [RFC4944]. The CORE work group has recently standardized constrained application protocol (CoAP) [RFC7252] for interacting with constrained resources where general HTTP is not memory/energy efficient. The use of web linking for resources description and discovery hosted by constrained web servers is specified by CORE [RFC6690]. Even though, CoAP allows the direct resource access for constrained devices, it is not advisable for direct access of resources in networks where multicast procedures are infeasible due to heavy network load, and the networks where sleepy nodes exist. So, the CoRE working group comes up with a solution called resource directory (RD) [draft-ietf-core-resource-directory] to host the devices service information, and allow other devices to perform lookup procedures through .well-known/core path to resources.

The services advertised by these constrained devices needs to be commissioned and provisioned properly to allow other devices to access it. CoRE RD solution is a directory based solution that depends on CoAP protocol. CORE RD solution uses registration/update/delete/lookup procedures for service registration, service update, deleting service, lookup of services respectively. Service commissioning is a method which verifies a pre registered services with special commissioning tools/agents. These tools can be tablets or special embedded devices which initially stores the devices identifications in secure manner. Once the services are advertised by any device, those services need to be verified using commissioner. CORE RD provides a standard procedure to interact with commissioner, where commissioner acts like a client device to look up and verify the advertised services. Once the commissioner verifies the pre-registered services, commissioner can put some policy rules on services hosted by devices for resource control. These rules defined on (1) how to access the services either with other constrained devices or client devices, and (2) on operational instructions.

Architecture is defined to authenticate and authorize client requests for a resource on a server using logical entities such as client(C), client authorization manager(CAM), server(S), and server authorization manager(SAM)[draft-gerdes-ace-actors]. The main goal of delegated CoAP authentication and authorization framework (DCAF) is the setup of a datagram transport layer security channel between two nodes to securely transmit authorization tickets   [draft-gerdes-core-dcaf-authorize]. The CAM sends an access request message on behalf of client by embedding requested permissions in client authorization information (CAI) field of access request message to

SAM. A ticket grant message is sent from SAM by embedding the permissions given from the server on a specific resource in server authorization information (SAI) field of ticket grant message to the client. These SAI, CAI use authorization information format (AIF) that describes the permissions requested from access request in a ticket request, where the underlying access control model will be that of an access matrix, which gives a set of permissions for each possible combination of a subject and an object [draft-bormann-core-ace-aif]. This simple information model also doesn’t allow conditional access (e.g.,"resource /s/tempC is accessible only if client belongs to group1 and does not belong to group2"). Finally, the model does not provide any dynamic functions such as enabling special access for a set of resources that are specific to a subject. But, the services provided by resources in constrained environment, need to be authorized and controlled conditionally based on some service level agreements or preconfigured policies on resource control.

Considering an example use case scenario such as thermostat device measures the current room temperature, and can service for air conditioner device to set automatic temperatures. In a smart home, user wants to regulate his room temperature automatically using his airconditioner device. Here, this airconditioner device can adjust its temperature to either cool the room or heat the room by accessing the service provided by the thermostat. Suppose this user leaves the home in the morning in hot summer and leaves the office in the evening to reach to home. But, before he reaches his room he wants to make his room cool enough. So he has to switch on the airconditioner from his mobile one hour before he leaves the office. So, before adjusting his aircconditioner to make the room cool enough, he might have to know the current room temperature. Thus he access the service provided by the thermostat to read the room temperature and adjust the airconditioner. However, there is a problem here on how to access these services which are provided by user’s home devices itself, what is the authenticity level to access from outside the home, even within home what is the access control/resource control of these devices because the neighboring device which are not authenticated can also access these service if those devices are within the constrained network range. Finally it is important to admit access of the service by client based on the configuration policies so that the devices can be protected from hazardous conditions, and allows only pre-agreed operations on devices.

The service provisioning presented in this document provides a method to support admission, and resource control policies using commissioning procedure. The method explains the device’s service access in two different use cases: first provisioning the service when a constrained device accessing the service provided by other constrained device, second, accessing the service provided by constrained device from the client device. Even though it is out of scope of the present document, it also considers a secure way of service commissioning as part of security.

2 Motivation

CORE RD solution provides various automated operations such as service registrations, service update, service removal, and service lookups initiated by endpoints and clients. However, managing this centralized directory server by allowing authorized users to perform these tasks, setting some service level agreements on clients to access these services, and providing limited or scope oriented lookups by other endpoints or clients require efficient service provisioning mechanism. The service provisioning method presented in this document deals on how a registered service from devices can be accessed by various clients or other devices. Moreover, it also provides a method for handling this resource/service access control mechanism using web service model for efficient service provisioning from outside the constrained home environment.

3 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

- o "CORE", CORE is a Constrained RESTful Environment providing a framework for resource-oriented application intended to run on constrained networks [RFC7228].

- o "COAP" The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and networks [RFC7252].

- o "RD" The Resource Directory (RD) is a directory based server to host the descriptions of resources and allowing the lookups to be performed for those resources by various client devices.

- o "Commissioner" Commissioning agent is tool/device that verifies the devices operation, integrity check with the network.

- o "Constrained Device" These are embedded computing devices that are expected to be as resource constrained in terms of RAM/ROM size, and to be deployed with the constrained environment such as 6LoWPAN Networks.

o "Client" A client device is like resource constrained client such as other constrained device (ex. Air conditioner) or rich client devices such as Mobile/Laptop/Tablet etc, which access the services hosted by constrained devices (ex. thermostat).

o "Provisioning Server" this server is a process of verifying service requester, providing access controls or admission controls on resources to be accessed and inter-operating with various devices without bothering about kind of network protocols used. It also provides web access model outside the constrained environment.

o "Device Profile" A device profile comprises a set of attributes that are associated with a particular device. These include services, features, names, descriptions etc.

4 System Architecture

The system architecture is better explained with two different scenarios: (1) Constrained device access the service advertised by other constrained device is as shown in Fig 1. Here, one constrained device such as air-conditioner can access the service such as current room temperature advertised by other constrained device (ex. thermostat). This advertised service is to be commissioned by commissioner, and then it should be set with some admission and resource control policies by provisioning server. And, finally the service is allowed to advertise its service access from other constrained devices. Any device that is interested in that advertised service, need to do service lookup from RD Server. Once obtaining the path to the advertised service, the constrained   client device can request a service to the device which hosts the service. Before sending the request, it MUST establish a secure channel between these two nodes [draft-schmitt-ace-twowayauth-for-iot]. Once the incoming request comes from the constrained client device, the device which hosts the service MUST authorize and provision for conditional access of its service from the provisioning server. The notification regarding the registered services to the commissioning agent can be sent from the RD server, which can be implementation specific and left for the user to choose any standard procedures and is out of scope of present document. Detailed operational procedure will be explained in the later sections of this document.

```
       +------+  +-------+   +-----+   +--------+  +----------+
       |Device1  |Device2|   | RD  |   |Provis  |  |Commision
       |(Air     |(Thermo|   |     |   |ioning  |  |ing       |
       |Condi |  |  stat)|   |Serv     |Sever   |  |Agent     |
       |tioner)  |       |   |er   |   |        |  |          |
       +--|---+  +----|--+   +--|--+   +----|---+  +-----|----+
          |           |         |           |            |
          |           |         |           |            |
          |           |Register |           |            |
          |           ----------//          |            |
          |           | Service/|  Verify Preregistered\ |/
          |           |         ------------------------//
          |           |         |    Service|         // |
          |           |         |           |        /   |
          |           |         |           |            |
          |           |         |           |            |
          |           |         |           |            |
          |           |         |           |//Define    |
          |           |         |          /-------------
          |           |         |         / \ Policies   |
          |Search a Service  \  |           |            |
          ---------------------//           |            |
          |           |       //|           |            |
          |           |      /  |           |            |
          |           |         |           |            |
          |           |         |           |            |
          |Request    |         |           |            |
          -----------/          |           |            |
          |Service   /|         |           |            |
          |         / |         |           |            |
          |           |Check for authorization           |
          |           |admission, Resource  |            |
          |           ---------------------//            |
          |           | Control Policies  //|            |
          |           |         |        /  |            |
          |           |         |           |            |
          |           |         |           |            |
          |           | // Service Grant/Deny            |
          |           /---------------------             |
          |          /|\        |           |            |
          |  /        |         |           |            |
         \//Service  |          |           |            |
          /\----------          |           |            |
          |    Grant/Deny       |           |            |
          |           |         |           |            |
```

Fig 1.Constrained device accessing service from constrained device

```
       +--------+  +-------+  +-------+  +---------+ +---------+
       |Client  |  |Device2|  |RD     |  |Provision| |Commissi |
       |(Smart  |  |(Thermo|  |Server |  |ing Server |oning    |
       | Phone) |  | stat) |  |       |  |         | |Agent    |
       |        |  |       |  |       |  |         | |         |
       |        |  |       |  |       |  |         | |         |
       +---|----+  +---|---+  +---|---+  +----|----+ +-----|---+
           |           |          |           |            |
           |           |          |           |            |
           |           |Register  |           |            |
           |           -----------/           |            |
           |           |Service   /           |            |
           |           |         /|           |            |
           |           |          |  /        |            |
           |           |          |//Verify Preregistered  |
           |           |          --------------------------
           |           |          |\    Service            |
           |           |          |           |            |
           |           |          |           |            |
           |           |          |           |  /         |
           |           |          |           |//Define    |
           |           |          |           -------------
           |           |          |           |\ Policies  |
           |           |          |           |            |
           | Request for Service  |        \               |
           ----------------------------------//            |
           |           |          |         //|            |
           |           |          |        /  |            |
           |           |          |           |            |
           |           |  /       |           |            |
           |           |// Request| for Service            |
           |           -----------------------             |
           |           |          |           |            |
           |           |          |           |            |
           |           |          |           |            |
           |           | Service Grant/Deny\  |            |
           |           ----------------------/             |
           |           |          |         //|            |
           |           |          |        /  |            |
           |           |          |           |            |
           |           |          |           |            |
           |  //       |          |           |            |
           |//   Service Grant/Deny           |            |
           \----------------------------------            |
           | \         |          |           |            |
           |           |          |           |            |
```

Fig 2. Client accessing service from Constrained device

2) Client device access the service advertised by constrained device is as shown in Fig 2. For example, the client device such as smart phone can access the service (ex. room temperature) advertised by other constrained device (ex. thermostat). The client can access the service within a home environment or outside the home environment. So, in this scenario, the provisioning server maintains the service as a web service.

This advertised service is to be commissioned by commissioner, then to be set with some admission and resource control policies by provisioning server. And, finally the service is allowed to advertise its access from the client devices. Any client that wishes to access this web service looks for corresponding operations provided from the provisioning server.

5 Network Topology

The constrained devices such as Thermostat, Airconditioner may use small memory constrained sensors/actuators for simple services such as cooling/heating the room or just to measure the current room temperature. These memory constrained embedded devices may implement the 6LoWPAN stack such as uIP (provided by Contiki), and provide access for communication to other external queries from client devices such as smart phone which typically implements rich stack TCP/IP. Even though RD server or Provisioning server are shown as separate servers in the LAN as given in Fig 3, these can be hosted on a single server running two different processes. Moreover, the commissioner implements a standard procedure to interact with devices as a separate agent process which is out of scope of the present document and has been left to user’s choice while satisfying the mentioned operations in the current draft. On the other hand, these specific operations can be implemented separately as a third party and to be used at the commissioning agent. The lower level communication technology can be implemented either through Bluetooth (BT) or near field communication (NFC) to verify the devices unique ID (for ex. using MAC). Even though, the implementation procedure for commissioner is out of scope for the present document, it is shown as sample interaction with RD server/provisioning server as part of commissioning procedure in subsequent sections. Even though the present document discusses about 6LoWPAN based sensor network, it can be easily moved to any other technology such as Zigbee/BLE/Wireless HART without any changes in the architecture or design, because the present document abstracted the communication networks with their edge routers. The communication and routing mechanisms or procedure between edge router and sensor devices/client devices are out of scope of the present document.

```
             -------                      --------
           //       \                  //        \
          /                           //            \
         /                           /
        /                           /
        | +--------+    |           |        +--------+|
       |  |RD Server---| |         | +-----+ |Thermostat|
       |  +--------+   | | LAN     | |Edge | +--------+ |
       |               |------|-------     |            |
       |  +----------+ | |    |    | |Router 6LoWPAN    |
        | |Provision --||     |     |+-----+           |
         |Server    |  /     |          +--------+   /
         +----------+ /      |          |Aircondioner
                     /       |       \  +--------+//
           \       //        |         \        //
             -------          |           --------
                              |
                              |
                              |
                              |
                              |
                             -|-----
                          //- |     -\
                        //  +-|----+   \
                       /    |Edge  |
                      /     |Router|
                      |     +------+      |
                     |                     |
                     |    WiFi             |
                     |   +-------+ +-----+ |
                     |   |Smart  | |Commisioning
                      |  |Phone  | | Agent|
                        +-------+ +-----+/
                                        /
                        \             //
                          \-       -//
                             -------
```

Fig 3. Network Topology

6 Operations

6.1 Register Service

The constrained device which hosts the service MUST register its service with the RD server using its unique identifier (for ex. MAC id, UDDI registry etc.) and IP address as shown in Fig 4. The device MUST send a POST request for registering its service.

Before sending a request, it MUST establish a secure channel between these two nodes [draft-schmitt-ace-twowayauth-for-iot]. Once the service has been registered with the RD server, the RD server may notify the registered information of a device (for ex.its unique identifier and device name) to a commissioning agent.

```
         +---------+                                         +---------+
         |         |                                         |         |
         | Device  |                                         |RD Server|
         |         |                                         |         |
         +----+----+                                         +-----+---+
              |                                                    |
              |                                                    |
              |                                               ‘.   |
              | POST /rd?ep=node1&d=example.com&et=temperature-no‘.|
              +--------------------------------------------------,’.
              | gp=thermostat&con=DeviceID(100)                ,’  |
              |                                             ,’     |
              |   ,’                                               |
              | ,’  2.01 Created Location: /rd/7521                |
              ‘.----------------------------------------------------
              | ‘-.                                                |
              |    ‘.                                              |
              |                                                    |
```

Fig. 4 Registering a Service

6.2 Verify pre-registered service

The commissioning agent MUST verify any pre registered service with the RD server as shown in Fig 5. The commissioning agent sends a GET request for domain lookup. Before sending the request, it MUST establish a secure channel between these two nodes [DTLS][TLS]. Once obtaining the specific domain, it MUST look for the group to which the service belongs. Once obtaining the specific domain and group, it MUST send a service look up with the RD server for the registered service. Once obtaining the service information about a specific device, the commissioning agent MUST verify the registered service. This service information is later used to create service registry in the provisioning server as explained in the following section. The example service information (denoted as SRV) looks like as shown in Fig 6.

```
       +----------------+                                  +-----------+
       |Commissioning   |                                  | RD Server|
       |Agent           |                                  |           |
       +------+--------+                                   +--------+-+
              |                                                     |
              |                                                     |
              |     GET /rd-lookup/d                            '.  |
              +---------------------------------------------------:'.
              |                                                 .'  |
              |                                                     |
              | .'2.05 Content </rd>;d=example.com,</rd>;d=example.com
              ::----------------------------------------------------+
              | '-.                                                 |
              |   GET /rd-lookup/gp?ep=node1&d=example.com       '. |
              +----------------------------------------------------/.
              |                                                  .' |
              |                                                     |
              | .'2.05 Content <coap://ip:port>;gp=thermostat;ep=node1
              ::----------------------------------------------------+
              | '-.                                                 |
              |                                                '.   |
              | GET /rd-lookup/res?rt=temp&gp=thermostat&d=example.com
              +---------------------------------------------------:'.
              |                                                 .'  |
              |                                                     |
              | .'2.05 Content <coap://host:port>;rt=temp;gp=thermostat
              ::----------------------------------------------------+
              | '-.                              d=example.com      |
      +----------------------------+                                |
      |Authentication of Service   |                                |
      |Info and DeviceID           |                                |
      +----------------------------+ POST Verified User; DeviceID'. |
              +----------------------------------------------------::
              | .'                                              .-' |
              .'.___________________________________________________|
              |  '.    2.00 OK                                      |
```

Fig. 5 Verify pre registered service

```
                 SRV {
                       Name: Node1
                       Group: Thermostat
                       Domain: myhome.com
                       Type: Temperature node
                       Device ID: 1001
                       Device IP: <host:port>
                     }
                       Fig 6. Example Service Informaion
```

6.3 Define policies on resource control

```
       +----------------+                        +---------------+
       |Commissioning   |                        |Provisioning   |
       | Agent          |                        | Server        |
       +------+---------+                        +--------+------+
              |  POST /thermostat /HTTP/1.1            '. |
              +------------------------------------------/.
              |  HOST thermostat.ps.example.com        .' |
              |  Content-Type: application/text           |
              |  SRV { Name: Node1                        |
              |        Group: Thermostat                  |
              |        Domain: myhome.com                 |
              |        Type: Temperature-node             |
              |        DeviceID: 1001                     |
              |        DeviceIP: <host:port> }            |
              |                                           |
              | .'     HTTP/1.1 200 OK                    |
              ::------------------------------------------'
              | '.     Content-Type: application/text     |
              |        { sID (service ID) }               |
              |                                           |
              |  POST /thermostat /HTTP/1.1            '. |
              +------------------------------------------::
              |  HOST thermostat.ps.example.com        .' |
              |  Content-Type: application/text           |
              |  AC { ServiceID: 1234                     |
              |       Auth: Basic Auth Support            |
              |       Count: 10                           |
              |       Admission Control: R,W,R/W,D }      |
              |                                           |
              | .'    HTTP/1.1 200 OK                     |
              ::------------------------------------------+
              | '.    Content-Type: application/text      |
              |                                           |
              |  POST /thermostat /HTTP/1.1            '. |
              +------------------------------------------::
              |  HOST thermostat.ps.example.com        .' |
              |  Content-Type: application/text           |
              |  RC { If C is from G1 allow {R,W};        |
              |       If C is from G2&!G3 allow {R};      |
              |       If C is from d1&g1 allow {R,W,D};   |
              |        : }                                |
              | .'    HTTP/1.1 200 OK                     |
              ::------------------------------------------+
              | '.    Content-Type: application/text      |
              |                                           |
```

Fig. 7 Defining Policies on Resource and Access Control

Once the hosted service has been verified by commissioning agent (CA), the CA MUST create a service registry with the provisioning server as explained in Fig 7. The provisioning server SHOULD send a service ID as a response back to the commissioning agent after creating the service entry.

This service ID can be later used by the commissioning agent to permanently DELETE the service entry (if required). The commissioning agent MUST create some admission control policies such as read (R), write (W), read/write (R/W), delete (D), number of simultaneous connection on resource etc. on the registered service. Once the admission control policies has been set on a specific device, the resource control policies such as conditional access of a service, quality of service agreements (based on the priority levels set for clients) can be set on that registered service. These conditional access on service can be implemented with simple conditional statements as explained in section 6.3.1 (for ex. "client (c) can access service with only read (R), write (W) permissions if it only belongs to group (g)"). The implementation or information format details of these conditional statements is out of scope of the present document (TBD). The example admission control and resource control policies are as shown in Fig 8, and Fig 9 respectively.

```
         AC {
               Service ID: 12345
               Auth: Basic Auth Support
               Count: 10
               Admission Control: R, W, R/W, D
                 :
                 :
            }
```

Fig 8. Example Admission Control Policies

```
         RC {
               If c is from g1 allow {R,W}
               If C is from g2 & !g3 {R}
               If C is from d1 & g1 allow {R, W, D}
                 :
                 :
            }
```

Fig 9. Example Resource Control Policies

6.3.1 Resource Control

Resource control policies for constrained devices are expressed in terms of conditional expressions as explained in Fig. 9. Consider a scenario where we define the client (C) (who accesses the resource) in terms of groups/levels. For example in a typical home building, we assign each floor as a group. Suppose for a three floor building, the clients such as mobile phone/air conditioner can belong to any of the floor within a building. And we allow various permissions for the clients according to the group it belongs to, as specified in Fig 10.

Client	R	W	U	D
G1	*	-	*	-
G2	*	*	-	-
G3	-	-	-	*

Fig 10. Example Permissions on Methods

Supposed we assigned the priorities for different groups as C belongs to {G1, G2, G3} => {P1, P3, P2}. Moreover, if we would like to assign different QoS classes for clients, depending on the applications they use then it is required to control QoS policies in resource control. QoS is defined in terms of various parameters such as {availability, reliability, serviceability, data accuracy, aggregation delay, coverage, fault tolerance, network lifetime} in wireless sensor networks. It is assumed that based on these parameters, QoS is defined in terms of various classes such as {Q1, Q2, Q3}, then it is required that some of the clients can make some pre-level agreements on QoS requirement for their applications either based on the groups it belongs to or based on the priority of the clients request (Suppose, C belongs to {Q1, Q2, Q3}). Method for defining QoS classes is out of scope of the present document. Once defining the groups, its priorities, QoS classes, and permissions, then the conditional statements which define the resource control policies can be defined as follows:

ST1: If the client belongs to G1 then it is allowed with permissions {R, R/W, U}, priority {P1}, QoS {Q1}, and operations {turn it up, read}; else if the client belongs to G2 then it is allowed with permissions {R, W, R/W}, priority {P3}, QoS {Q2}, and operations {turn it up, read}; else if the client belongs to G3 then it is allowed with permissions {D}, priority {P2}, QoS {Q3}, and operations {turn it down}.

ST2: Allow the client with priority {P1}, QoS {Q1}, operations {turn it up, turn it down, read}, and allow only with permissions {R} in G1; permissions {R, R/W, D} in G2; and permissions {D} in G3.

ST3: Allow the client with priority {P1}, QoS {Q1}, and allow with permissions {R}, operations {read} in G1; allow with permissions {R, R/W, D}, operations {turn it up, turn it down, read} in G2; and allow with permissions {D}, operations {turn it down} in G3.

Above conditional statements are few examples on how to define the conditional statements, the statements can be defined on any manner based on the resource control policies we would like to achieve. The above statements can be better explained in plain semantic notation as shown in Fig 11(a)-13(a), and the corresponding JSON representations for message exchange is explained in Fig 11(b)-13(b). These statements can be even implemented using data modeling language such as YANG or ASN 1.1 which is out of scope of the present document.

```
 C
 {
    G1                                 |"[
    {                                  |"C":{"G1":{"Allow":"R,U",
        Allow {R,U}                    |"Priority":"P1","QoS":"Q1",
        Priority {P1}                  |"Operations":"turnup,read"},
        QoS {Q1}                       |"G2":{"Allow":"R,W",
        Operations {tunr it up, read}|"Priority":"P3","QoS":"Q2",
    }                                  |"Operations":"turn it
    G2                                 |up,read"},"G3":{"Allow":"D",
    {                                  |"Priority":"P2","QoS":"Q3",
        Allow {R,W}                    |"Operations":"turn it down"
        Priority {P3}                  |}}]"
        QoS {Q2}                       |
        Operations {turn it up, read}|
    }                                  |
    G3                                 |
    {                                  |
        Allow {D}                      |
        Priority {P2}                  |
        QoS {Q2}                       |
        Operations {turn it down}      |
    }                                  |
 }                                     |

               (a)                                 (b)
```

Fig 11. ST1: (a) Semantic Notation (b) JSON Representation

```
       C                                  | "[
       {                                  | "Priority":"P1","QoS":"Q1",
          Priority {P1}                   | "Operations":"turn it up,
          QoS {Q1}                        | turn it down, read",
          Operations {turn it up,turn it  | "C":{"G1":{"Allow":"R"},
                      down, read}         |      "G2":{"Allow":"R,W,D"},
          G1                              |      "G3":{"Allow":"D"}}
          {                               | ]"
              Allow {R}                   |
          };                              |
          G2                              |
          {                               |
              Allow {R,W,D}               |
          };                              |
          G3                              |
          {                               |
              Allow {D}                   |
          };                              |
       }                                  |

                    (a)                                 (b)
             Fig 12. ST2: (a) Semantic Notation  (b) JSON Representation

       C                                     | "[
       {                                     | "Priority":"P1","QoS":
           Priority {P1}                     | "Q1","C":{"G1": {"Allow":
           QoS {Q1}                          | "R","Operations":"read"},
           G1                                | "G2":{"Allow":"R,W,D",
           {                                 | "Operations":"turn it up,
               Allow {R}                     | turn it down, read"},
               Operations {read}             | "G3":{"Allow":"D",
           };                                | "Operations":"turn it
           G2                                | down"}}]"
           {                                 |
               Allow {R,W,D}                 |
               Operations {turn it up, turn  |
                           down, read}       |
           };                                |
           G3                                |
           {                                 |
               Allow {D}                     |
               Operations {turn it down}     |
           };                                |
       }                                     |

                    (a)                                  (b)
             Fig 13. ST3: (a) Semantic Notation (b) JSON Representation
```

6.4 Search for services by device

Any client device (as explained for scenario 2) MUST interacts with the provisioning server and looks for deployed services by devices. Moreover, the provisioning server can verify the complete authorization, admission, and resource control of any device’s services. Whereas, if any other constrained devices (ex. air conditioner) searches for services hosted by other constrained device (as explained for scenario 1) MUST interact with the RD server as shown in Fig 10. Here, initially the device queries for all services that are hosted by other devices, then it searches within the domain for specific service, its SRV info, and path to the hosted service. Before sending a request, it MUST establish a secure channel between these two nodes [draft-schmitt-ace-twowayauth-for-iot].

```
       +---------------+                                   +----------+
       | Device        |                                   | RD Server|
       | (aircondit    |                                   |          |
       |   ioner)      |                                   |          |
       +-----+---------+                                   +-------+--+
             |                                                     |
             |  GET /rd-lookup/gp?d=example.com                '.  |
             +---------------------------------------------------'.:
             |                                                 .-' |
             | .'2.05 Content <gp="thermostat">                    |
             ::----------------------------------------------------+
             | '-.                                                 |
             |   GET /rd-lookup/ep?gp=thermostat                '. |
             +----------------------------------------------------::
             |                                                  .' |
             | .'2.05 Content <Node1> <Node2>                      |
             ::----------------------------------------------------+
             | '.                                                  |
             |                                                     |
             | GET /rd-lookup/ep?et=temperature&gp=thermostat   '. |
             +----------------------------------------------------'.
             |                                                  .' |
             |                                                     |
             | .'2.05 Content <coap://ip:port>;ep="Node1"          |
             ::----------------------------------------------------+
             | '-.                                                 |
             |                                                     |
```
Fig. 10 Search for services by device

6.5 Service request and response

In scenario 1 (as shown in Fig 1), service request and response MUST use coap based communication to access the service as shown in Fig 11. Before sending a request, it MUST establish a secure channel between these two nodes [draft-schmitt-ace-twowayauth-for-iot]. Suppose, the constrained client device (for ex. airconditioner) want to access the service hosted by another constrained device (for ex. thermostat), then the client device MUST send a coap based GET request to thermostat. Then, this device (thermostat) SHOULD send a POST request to provision this service request with the provisioning server by sending clients <IP:port>. Based on the clients <IP:port>, the provisioning server MUST find the client (ex. airconditioner) details such as service information, group, domain, and type details.

```
   +------------+              +-------------+          +-----------+
   |Airconditi  |              |Thermostat   |          |Provision  |
   |oner        |              | (IP1)       |          |ining Server
   | (IP2)      |              |             |          | (IP3)     |
   +-----+------+              +------+------+          +--------+--+
         |                            |                          |
         |coap://thermostat.        ‘.|                          |
         +----------------------------::                         |
         |    example.com/temp     .’ |POST /thermostat       ‘. |
         |                            +--------------------------::
         |                            |HOST thermostat.ps.    .’ |
         |                            |             example.com  |
         |                            |Content-Type: application/txt
         |                            |{ SRC: <IP1,port>         |
         |                            |  DST: <IP3,port>         |
         |                            |  Client: <IP2,port> }    |
         |                            |                          |
         |                            |  +-------------------------+
         |                            |  |Check for Admission,      |
         |                            |  |ResourceControl of thermost
         |                            |  |for airconditioner        |
         |                            |  +-------------------------+
         |                            |                          |
         |                            | .’2.00 OK { Permit/Deny }|
         | .’URI-Path: temp CON 200    ::------------------------+
         ::-------------------------+ ‘-.                        |
         | ‘.("thermostat","aaaa::212.|                          |
         |  7402.2.202","temp",27)    |                          |
         |                            |                          |
      Fig. 11 Request/Response within Constrained Environment
```

Once the client is identified, the provisioning server MUST check for authorization, admission and resource control policies of hosted service (ex. thermostat). Once the service request is authorized to access then the URI-Path for hosted service along with the value is sent as a coap response to client device (air conditioner). Here, the request is conditional i.e. based on the resource control policies of a resource (such as thermostat) for a client (airconditioner), the permissions are given to access the resource.

```
+-------------+               +------------+           +---------+
|             |               |Provision   |           |         |
| Client      |               |ining Server|           |Thermostat
|             |               |            |           |         |
+-----+-------+               +-----+------+           +------+--+
      |                             |                         |
      |http://thermostat.        ‘. |                         |
      +----------------------------::                         |
      |  example.com/temp        .’ |                         |
      |            +-----------------------------+            |
      |            |Check for Admission,         |            |
      |            |Resource Control of thermostat            |
      |            |for airconditioner           |            |
      |            +-----------------------------+            |
      |                             |                         |
      |                             | coap://thermostat.   ‘. |
      |                             +------------------------::
      |                             |example.com/temp      .’ |
      |                             |                         |
      |                             |                         |
      |                             | .’URI-Path: temp CON 200|
      |                             ::------------------------+
      |                             | ‘.                      |
      | .’ HTTP/1.1 200 OK          |                         |
      ::----------------------------+                         |
      | ‘. Temperature: 27          |                         |
      |                             |                         |
```
Fig. 12 Request/Response from outside Constrained Environment

Service request and response in scenario 2 (as shown in Fig 2), uses simple http based communication to access the service from the PS. Provisioning Server then sends a coap based GET request to the ultimate device that hosts service. Before sending this request to the actual device for service, PS authorizes the service request. Once, the service request is authorized to access, then the URI-path for hosted service along with the value is sent as HTTP response to client device. PS can implement a reverse proxy case for HTTP-CoAP protocol translation defined in [draft-ietf-core-http-mapping].

```
------------------HTTP begin -------------------------------------
HTTP POST
Request:
POST /thermostat   /HTTP/1.1
HOST thermostat.example.com
Content-Type:  application/x-www-form-urlencoded
Content-Length:  length
licenseID=string & content=string & paramsXML=string

Response:
HTTP/1.1   200 OK
Content-Type:  text/xml; charset=utf-8
Content-Length:  length
<?xml version="1.0" encoding="utf-8"?>
<string xmlns="http://xyz.com/">
string
</string>

------------------HTTP end  -------------------------------------

------------------ REST via HTTP begin --------------------------
REST via HTTP POST
Request:
POST /thermostat   /HTTP/1.1
HOST thermostat.example.com
Content-Type:  application/x-www-form-urlencoded
Content-Length:  length

licenseID=string & content=string & paramsXML=string

Response:
HTTP/1.1   200 OK
Content-Type:  text/xml; charset=utf-8
Content-Length:  length

string

------------------REST via HTTP end -----------------------------

------------------SOAP begin ------------------------------------

SOAP 1.2
Request:
POST /Thermostat   /HTTP/1.1
HOST: www.example.org
```

```
      Content-Type:  application/soap+xml; charset=utf-8
      Content-Length:  length

      <?xml version="1.0"?>
      <soap:envelop>
      Xmlns:soap=http://www.w3.org/2001/12/soap-envelop
      Soap:encodingStyle=http://www.w3.org/2001/12/soapencoding>
      <soap:body xmlns: m="http://www.myhome.org/thermostat">
      <m:GetTemperature>
      <m:thermostat>1</m:thermostat>
      </m:GetTemperature>
      </soap:body>
      </soap:envelop>

      Response:
      HTTP/1.1 200 OK
      Content-Type:  application/soap+xml; charset=utf-8
      Content-Length:  length

      <?xml version="1.0"?>
      <soap:envelop>
      Xmlns:soap=http://www.w3.org/2001/12/soap-envelop
      Soap:encodingStyle=http://www.w3.org/2001/12/soapencoding>
      <soap:body xmlns: m="http://www.example.org/thermostat">
      <m:GetTemperatureResponse>
      <m:temperature>27.8</m:temperature>
      </m:GetTemperatureResponse>
      </soap:body>
      </soap:envelop>

      ------------------SOAP end ---------------------------------
```

7 Security Considerations

Security level for message authentication is out of scope of the present document. However, the following security consideration needs to be considered for the present proposed method. Services that run over UDP are unprotected and vulnerable to unknowingly become part of a DDoS attack as UDP does not require return routability check. Therefore, an attacker can easily spoof the source IP of the target entity and send requests to such a service which would then respond to the target entity. The TLS/DTLS based security solution can be considered for secure message communication.

8 IANA Considerations

TBD

9 References

9.1 Normative References

9.2 Informative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, May 2014.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.

[draft-ietf-core-resource-directory] Shelby, Z., and Bormann, C., "CoRE Resource Directory", draft-ietf-core-resource-directory-02 (work in progress), November 2014.

[draft-gerdes-ace-actors] Gerdes, S., "Actors in the ACE Architecture", draft-gerdes-ace-actors-03 (work in progress), March 2015.

[draft-gerdes-ace-dcaf-authorize] Gerdes, S., Bergmann, O., Bormann, C., "Delegated CoAP Authentication and Authorization Framework (DCAF)", draft-gerdes-ace-dcaf-authorize-02, March 2015.

[draft-bormann-core-ace-aif] Bormann, C., "An Authorization Information Format (AIF) for ACE", draft-bormann-core-ace-aif-oo, January 2014.

[draft-schmitt-ace-twowayauth-for-iot] Schmitt, C., Stiller, B., "Two-way Authentication for IoT", draft-schmitt-ace- twowayauth-for-iot-01, December 2014.

[DTLS] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security", RFC 6347, January 2012.

[TLS] Dierks, T. and C. Allen, "The TLS Protocol Version 1.2", RFC 5246, August 2008.

[draft-ietf-core-http-mapping] Castellani, A., Loreto, S., Rahman, A., Fossati, T., and Dijk, E., "Guidelines for HTTP-CoAP Mapping Implementations", draft-ietf-core-http-mapping-05, (work in progress), Oct 2015.

10 Acknowledgements

Special thanks to Amit Kumar S,Zhengfei, Fubaicheng, Yangjun,Vijayachandran Mariappan, Shashidhar C Shekar, Jayaraghavendran K, Ajay Sankar, Puneet Balmukund Sharma, and Rabi Narayan Sahoo for extensive comments and contributions that improved the text.

Thanks to Hedanping (Ana), Behcet Sarikaya, and Carsten Bormann for helpful comments and discussions that have shaped the document.

Authors’ Addresses

Vasu K
Huawei Technologies
Bangalore
India

EMail: vasu.kantubukta@huawei.com

Rahul A Jadhav
Huawei Technologies
Bangalore
India

EMail: rahul.jadhav@huawei.com

yangneng
Huawei Technologies
Shenzhen
China

EMail: yangneng@huawei.com