Institute of Architecture of Application Systems

Internet of Things Security Patterns

Lukas Reinfurt[1,2], Uwe Breitenbücher[1], Michael Falkenthal[1], Paul Fremantle[3]
Frank Leymann[1],

[1]Institute of Architecture of Application Systems,
University of Stuttgart, Germany
{firstname.lastname}@iaas.uni-stuttgart.de

[2]Daimler AG, Stuttgart, Germany
lukas.reinfurt@daimler.com

[3]School of Computing,
University of Portsmouth, UK
paul.fremantle@port.ac.uk

Reinfurt, L., Breitenbücher, U., Falkenthal, M., Fremantle, P., and Leymann, F. 2017. Internet of Things Security Patterns. HILLSIDE Proc. of Conf. on Pattern Lang. of Prog. 24 (October 2017), 28 pages.

BibTeX

```
@inproceedings {Reinfurt.2017,
  author = {Reinfurt, Lukas and Breitenb\"{u}cher, Uwe and
Falkenthal, Michael and Fremantle, Paul and Leymann, Frank},
  title = {Internet of Things Security Patterns},
  booktitle = {Proceedings of the 24th Conference on Pattern
Languages of Programs},
  year = {2017},
  numpages = {28},
  url = {http://dl.acm.org/citation.cfm?id=3290281.3290305},
  publisher = {The Hillside Group},
}
```


Universität Stuttgart
Germany

Internet of Things Security Patterns

Lukas Reinfurt, Daimler AG
Uwe Breitenbücher, University of Stuttgart
Michael Falkenthal, University of Stuttgart
Paul Fremantle, University of Portsmouth
Frank Leymann, University of Stuttgart

The Internet of Things (IoT) is growing, with new technologies, standards, devices, platforms, and applications being constantly developed. This has lead to a confusing solution landscape, which makes understanding the various options and choosing a path between them difficult. In order to help with this problem, we have collected IoT Patterns, which are textual descriptions of common problems and their abstract solutions based on repeatedly found real life examples. With this work, we add some security related IoT Patterns to complement the already existing catalog of security patterns that can be applied to IoT systems. The Trusted Communication Partner and Outbound-Only Connection patterns decrease the attack surface of devices. The Permission Control and Personal Zone Hub patterns give device owners control over what happens with their devices and data. The Whitelist and Blacklist patterns control access to and prevent abuse of resources.

Categories and Subject Descriptors: [**Computer systems organization**] Embedded and cyber-physical systems; [**Software and its engineering**] Design Patterns; [**Security and privacy**] Systems security, Network security

Additional Key Words and Phrases: Internet of Things, Design Patterns, Security, Devices, Data

ACM Reference Format:

Reinfurt, L., Breitenbücher, U., Falkenthal, M., Fremantle, P., and Leymann, F. 2017. Internet of Things Security Patterns. HILLSIDE Proc. of Conf. on Pattern Lang. of Prog. 24 (October 2017), 28 pages.

1. INTRODUCTION

In recent years, the Internet of Things (IoT) has become more and more a reality. It started as a vision of all kinds of things being equipped with sensing, actuating, processing, storage, and communication capabilities. Such a ubiquitous layer of cyber-physical devices would then be able to i) measure itself and its environment, ii) combine, process, and analyze these measurements to gain new insights, and iii) use these insights to optimize itself and the environment [Voas 2016]. Steady progress in the miniaturization of electronics, increased efficiency, and falling prices now make this vision reachable [Gubbi et al. 2013]. As a result, more and more organizations and individuals, from large businesses, to startups, to open-source developers are now trying to be part of this vision. This has lead to a large number of new products, which keeps growing day by day. But since the realization of the IoT is still in early stages, there are also too many standards and technologies, as well as a lack of common understanding in this area [Atzori et al. 2010; Gubbi et al. 2013; Guth et al. 2016; Ishaq et al. 2013]. In addition, development often happens in silos, as each organization builds products for a particular domain, such as home automation or health care [Singh et al. 2016]. Thus, there is a large, confusing IoT market with solutions that fundamentally are often not that different.

We collected frequently re-occurring problems and their proven solutions in an abstract form as IoT Patterns. These

Author's address: Lukas Reinfurt: Epplestraße 225, 70546 Stuttgart, Germany; email: lukas.reinfurt@daimler.com; Uwe Breitenbücher, Michael Falkenthal, and Frank Leymann: Universitätsstraße 38, 70569 Stuttgart, Germany; email: [firstname].[lastname]@iaas.uni-stuttgart.de, Paul Fremantle: Buckingham Building, Lion Terrace, Portsmouth, UK, PO1 3HE; email: paul.fremantle@port.ac.uk

patterns should help IoT architects and developers with understanding and building IoT products. They may also be useful for students and educators as they they provide a high level overview of common parts of IoT systems and their interactions with each other. In our previous work, we presented IoT Patterns[1] for device energy supply and operation modes [Reinfurt et al. 2017c], communication and management [Reinfurt et al. 2016, 2017a], and bootstrapping and registration [Reinfurt et al. 2017b], and applied some of them to smart factory systems [Reinfurt et al. 2017d]. These patterns are mostly not concerned with security, but security is a major problem in the IoT area, which may inhibit adoption and hamper the full realization of the broad IoT vision [Singh et al. 2016]. In IT systems that are virtual in their scope, a security breach may lead to loss of data, system outages, or financial damages, but seldom to a safety issue. In IoT systems, however, where all kinds of machinery may be remotely controlled, a security breach can have real consequences and cause harm to people and property. Besides, there are a number of issues concerning, for example, secure data transportation and management, identification and authentication, trust, or compliance, which arise from or will only increase with the constraints of typical IoT devices, further decentralization into fogs, and the potential scale of future IoT systems [Singh et al. 2016]. Thus, patterns that can be used to increase security in IoT systems are of vital importance. Some already exist, as we will show in the related work in Section 2, but we present in this paper a small selection of additional security patterns for IoT systems.

The remainder of this paper is structured as follows: Section 2 presents work related to IoT security patterns. Section 3 describes the pattern identification process we used for finding the patterns as well as the format used to document the patterns. Section 4 presents our IoT Security Patterns in detail, while Section 5 summarizes and concludes the paper.

2. RELATED WORK

Christopher Alexander introduced the pattern approach in 1977 in his book *A Pattern Language: Towns, Buildings, Construction* [Alexander et al. 1977]. From there the approach spread into other domains, in particular, IT, where many books and other publications are available on various aspects of the field, such as object-oriented software design [Gamma et al. 1995], messaging [Hohpe and Woolf 2004], or cloud computing [Fehling et al. 2014; Strauch et al. 2012a,b]. In addition, there has been more work on the pattern authoring process in general [Fehling et al. 2015a, 2014; Harrison 2006a,b; Meszaros and Doble 1996; Wellhausen and Fießer 2012]. There is also work on making abstract patterns usable for concrete scenarios by treating the examples present in the pattern description as concrete solutions, which can be linked to the patterns and be reused for concrete implementations [Falkenthal et al. 2014a,b]. Abstract patterns can also be linked to other patterns with technology specific explanations [Falkenthal et al. 2016]

In the domain of IoT patterns, we presented four papers with patterns for device energy supplies and operation modes [Reinfurt et al. 2017c], device communication and management [Reinfurt et al. 2016, 2017a], and device bootstrapping and registration [Reinfurt et al. 2017b]. These patterns are mostly not concerned with security, apart from the Remote Lock and Wipe pattern, which allows stolen devices to be remotely locked or wiped to prevent misuse or data theft. Eloranta et al. presented patterns for building distributed control systems [Eloranta et al. 2014a,b]. These patterns are more concerned with safety than with security. Qanbari et al. introduced four patterns for provisioning, deploying, orchestrating, and monitoring edge applications [Qanbari et al. 2016]. These patterns are not concerned with security aspects. Another paper presents a pattern language for IoT applications [Chandra 2016]. It is largely based on blog entries of patterns which are not comparable to our patterns, neither in format nor scope. In the case of the security patterns, they consist of only one sentence per pattern.

In the domain of security patterns there also has been some work which may be relevant to IoT security. Schumacher et al. published security patterns on various topics, including identification and authentication, access control, and cryptographic key management [Schumacher et al. 2005]. Several of these patterns are relevant for and used in IoT solutions, such as Authorization, Role-Based Access Control, Role Rights Definition, Single Access Point, Limited Access, Packet Filter Firewall, Authenticator, Secure Channels, Known Partners, and Certificate Revocation. Schumacher also published the Firewall, Packet Filter, and Proxy patterns [Schumacher 2003].

[1] More information and excerpts of already published IoT Patterns can be found at `http://www.internetofthingspatterns.com`

Fernandez added a more recent publication which adds further patterns that are relevant to IoT security, such as Authenticator, Credential, Policy-Based Access Control, Access Control List, Asymmetrical Encryption, Digital Signature with Hashing, and Secure Distributed Publish/Subscribe [Fernandez 2013]. Villarreal et al. describe a Whitelisting Firewall pattern [Villarreal et al. 2013]. Romanosky lists several security design patterns, of which Authoritative Source of Data and Fail Securely are relevant to IoT [Romanosky 2001]. Another collection of patterns by Kienzle et al. includes general security patterns that may be relevant for IoT [Kienzle et al. 2002]. Among them is the Network Address Blacklist pattern, which can be seen as a variant of our Blacklist pattern. Blakley et al. present several patterns for building available and protected systems [The Open Group 2014]. Among them are the Protected System, Policy, Authenticator, Secure Communication, Security Association, and Secure Proxy patterns, which are applicable to IoT systems. Yoder et al. describe seven architectural patterns for enabling application security [Yoder and Barcalow Yoder and Barcalow], including Single Access Point, Roles, and Limited View, which are used in IoT systems as well. All these existing patterns cover many areas of security in IoT systems. Nevertheless, the IoT presents new features and problems not covered by the more general security patterns for which we present additional patterns in this paper. These new patterns should be seen as complementary to the existing security pattern catalog and often have to be used in combination with existing patterns to provide security.

3. PATTERN IDENTIFICATION AND FORMAT

Patterns are not invented, they are found. In our previous work, we described our process for finding patterns in more detail [Reinfurt et al. 2016, 2017a], which is based on [Coplien 1996; Fehling et al. 2015b]. We start by reading technical documentation, user manuals, product pages, standards, research papers, and white papers concerned with IoT products or technologies. During reading, we collect and group reoccurring descriptions of solutions to build a catalog of potential pattern ideas. Once we have at least three such solution descriptions per pattern idea (Coplien's Rule of Three [Coplien 1996]), we combine theses solutions into an abstract description and author a pattern around it.

The pattern format we use for that is based on other existing formats and guides, including [Fehling et al. 2015a, 2014; Harrison 2006a,b; Meszaros and Doble 1996; Wellhausen and Fießer 2012]. It is described in more detail in our previous work [Reinfurt et al. 2016, 2017a,b]. In short, it contains the following elements: A **Name** and **Icon** provide textual and visual identifiers for the patterns. The icon can also be used in architectural diagrams, drawings, or GUIs to represent the pattern. A short summary is also added to provide a quick overview of the pattern. If the pattern might be known under different names, these are listed as **Aliases**. The **Context** section details the situation in which the **Problem** occurs. To solve the problem, you often have to weigh different aspects against each other, which are listed as **Forces**. These lead to a **Solution**, which is shortly summarized. The **Solution Details** section goes into more detail on the solution and provides some benefits and drawbacks that should be considered. Sometimes, there are variations of a pattern which are then listed in the **Variants** section. A pattern also often has other **Related Patterns**, with which it could be combined or with which it may be incompatible. In the end, the **Known Uses** of the pattern are listed, which are the real life examples on which the pattern is based.

4. INTERNET OF THINGS SECURITY PATTERNS

In this section, we present the IoT Security Patterns we collected in the scope of this work. Figure 1 provides an overview of the patterns and Table I adds short summaries. The following subsections go into more details on these patterns.

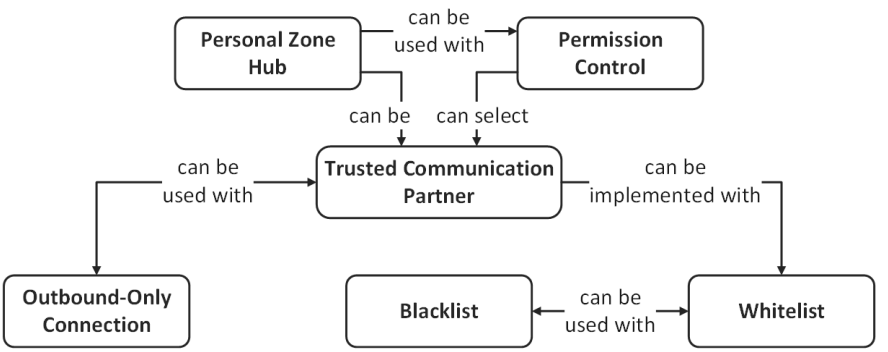
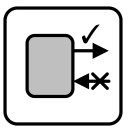
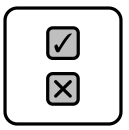


Fig. 1. Overview of and connections between the IoT Security Patterns presented in this paper.

Table I. Short summary of the IoT Security Patterns of this paper.

Icon	Description
	**Trusted Communication Partner** (p. 5) IoT environments may contain all sorts of devices and other components that are not under your direct control. These may pose a security or privacy risk to your devices and data. Limit your devices' autonomy and configure them to communicate only with trusted partners. Block and notify connection attempts from other sources.
	**Outbound-Only Connection** (p. 8) Attackers may send unsolicited connection requests to devices to get them to connect to an infected communication partner. To prevent this, configure devices so that only they initiate connections. Block all incoming connection requests.
	**Permission Control** (p. 11) Device owners are afraid to completely hand over access to their devices and data to others. Allow device owners to choose which functionality and data a backend server or other communication partners are allowed to access when they first connect to the device. Ensure that these choices are respected and updated when something changes.
	**Personal Zone Hub** (p. 14) Managing the permissions, data sharing, and control of devices across multiple gateways and clouds is complex. Create a Personal Zone Hub which handles such settings in a central place under the control of the device owner or a trusted third party. Allow owners to selectively provide access to their devices and data through this hub.
	**Whitelist** (p. 17) Available privileges may be abused, but not all potential abusers are known beforehand. To minimize the risk of abuse, add identifiers of all trusted communication partners to a Whitelist. Block the privileges controlled by the Whitelist for those who are not on it.
	**Blacklist** (p. 21) Available privileges may be abused. To stop this, implement a Blacklist to which identifiers of abusive communication partners can be added. For each partner that requests a privilege first check this list and deny it if its identifier is found on it.

4.1 Trusted Communication Partner

IoT environments may contain all sorts of devices and other components that are not under your direct control. These may pose a security or privacy risk to your devices and data. Limit your devices' autonomy and configure them to communicate only with trusted partners. Block and notify connection attempts from other sources.

Aliases: Well-Known Target

Context: In the IoT, devices may communicate with many other communication partners, such as backend servers, applications, or other devices. Some of these connections may be used regularly while others are used infrequently. Besides, the number of available communication partners may be constantly changing as new apps or services are created and devices change their location.

Problem: In a dynamic environment there may be multiple potential communication partners available for a device. These may not be known or trusted and may pose a security risk as attackers may try to use them to get access to devices and their networks.

Forces:

—**Threats:** Other components may pose a threat by trying to access the device or by launching a denial of service attack.

—**Uninvited Communication:** Uninvited communication should be blocked where possible.

—**Functionality:** The security measures should not impede normal operation and communication.

—**Flexibility:** Some use cases are rather static and do not involve frequently changing communication partners. Others are the opposite and require interactions with frequently changing communication partners, which increases the risk of potential attacks.

Solution: Configure the device with a single communication partner or a list of communication partners that you trust. Only allow incoming or outgoing communication with these Trusted Communication Partners. Block other communication attempts and notify the person responsible for investigating these attempts.

Solution Details: Each device is configured to only communicate with a limited selection of Trusted Communication Partners. These Trusted Communication Partners may have been placed on the device during bootstrapping. They may also be configured and changed later on.

Trusted Communication Partners may be implemented in form of single entries in a configuration file, as shown in the middle of Figure 2. On the device, a component checks each incoming and outgoing connection. If the connection source (for incoming) or target (for outgoing) connections matches the entry in the configuration file, as shown for number 2 and 4 in Figure 2, it is allowed to pass.

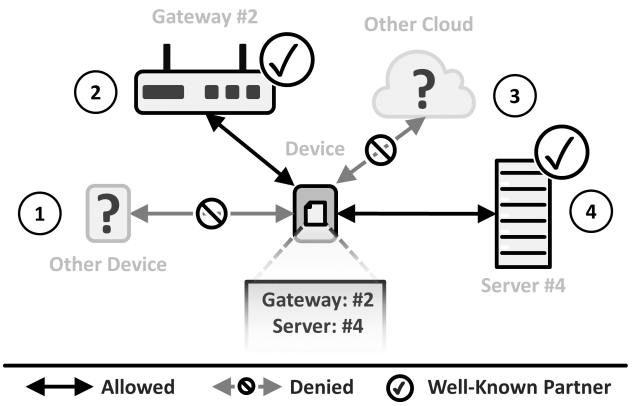


Fig. 2. Solution sketch of the Trusted Communication Partner pattern.

Otherwise, the connection is blocked and logged, for example, number 1 and 3 in Figure 2. The logs may be used for monitoring purposes later on or they may be pushed to another component for live monitoring. They can also be used as a basis for generating a Blacklist.

For some use cases, it may make sense to only use Trusted Communication Partners for certain functionality. For example, it may be required that any communication partner can read the sensor values of your device, while only a selection is allowed to control the actuators. Common examples for Trusted Communication Partners are Device Gateways, Device Shadows, servers for Remote Device Management, or a Personal Zone Hub.

Benefits:

—**Decreased Attack Surface:** Communication is restricted to a small number of Trusted Communication Partners, which decreases the surface for attacks.

—**Explicit Allowance:** Each allowed communication partner has to be specified, making allowed partners explicit.

Drawbacks:

—**Infected Communication Partner:** A Trusted Communication Partner on the list may still be infected and pose a risk. Taking security measurements to limit the likelihood of such infection may help but cannot totally eliminate the possibility.

—**Spoofing:** An attacker may be able to imitate a Trusted Communication Partner by taking over its address. This is known as spoofing. On a local network, a new system can take over the IP address of a Trusted Communication Partner. On the wider Internet, the attacker may take over the DNS entry. To prevent others from posing as a Trusted Communication Partner, use Digital Signature with Hashing so that they can prove that they are legitimate [Fernandez 2013].

—**Limited Flexibility:** Only allowing communication with Trusted Communication Partners limits flexibility when communicating with other partners is required. This also applies when you are offering a product or service where this may block potential customers.

—**Effort:** Each new communication partner has to be specified before it can communicate with the device. This makes Trusted Communication Partner impractical for situations where communication partners change frequently.

Factory Bootstrap[2], On-Site Bootstrap[3], or Remote Bootstrap[4] can be used for the initial setup of Trusted Communication Partners [Reinfurt et al. 2017b]. Remote Device Management[5] can be used to change Trusted Communication Partners later on [Reinfurt et al. 2017a].

Related Patterns:

—**Factory Bootstrap, On-Site Bootstrap, or Remote Bootstrap:** These patterns may be used to configure Trusted Communication Partners at the beginning of a device's lifetime [Reinfurt et al. 2017b].

—**Remote Device Management:** When a device is already deployed, Remote Device Management may be used to change Trusted Communication Partners in the field [Reinfurt et al. 2017a]. A Remote Device Management server is often also a good candidate for a Trusted Communication Partner.

—**Whitelist:** The representation of Trusted Communication Partners can be implemented with a Whitelist which could be in the form of a Whitelisting Firewall [Villarreal et al. 2013].

—**Outbound-Only Connection:** Trusted Communication Partners may be combined with Outbound-Only Connection to only allow communication which is outgoing and targets a known communication partner.

—**Permission Control:** A user may be asked with Permission Control to select or agree to Trusted Communication Partners for communication.

—**Device Gateway[6] or Device Shadow[7]:** For many devices, Device Gateways or Device Shadows are Trusted Communication Partners [Reinfurt et al. 2016, 2017a].

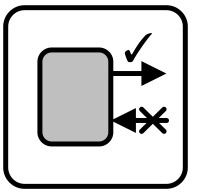
—**Known Partners:** This pattern describes a similar problem, where an organization has to ensure that the entities they interact with are who they claim they are. The solution is to use certificates to proof identities [Schumacher et al. 2005].

—**Firewall:** A Firewall is another way to manage and check communication with Trusted Communication Partners [Schumacher 2003]. It has more sophisticated means for inspecting and blocking communication, but also requires more resources, which may not be available on constrained devices.

Known Uses: The Microsoft Azure IoT Hub documentation mentions that devices should only communicate with Trusted services [Microsoft 2015]. Optigo Networks explains that HVAC devices should only communicate with authorized controllers. Other communication attempts should be blocked and notified [Optigo Networks 2016]. The device libraries, SDKs, or documentation of some IoT platforms define specific communication targets which a developer has to use to connect a device to the platform. IBM's Watson IoT Platform provides a specific MQTT topic format, which has to be used to connect devices to the platform [IBM 2016]. In a similar fashion, Amazon offers SDKs for its AWS IoT platform, which connect devices to the platform's Device Gateway or Device Shadow through MQTT [Amazon Web Services 2017]. In OAuthing, devices use a Trusted Communication Partner to connect to a device identity provider over MQTT [Fremantle and Aziz 2016].

[2]Factory Bootstrap places information required to create an initial connection on a device during the manufacturing process [Reinfurt et al. 2017b].
[3]On-Site Bootstrap allows placing the information required for initial communication on devices while they are in the field by using removable storage media [Reinfurt et al. 2017b].
[4]Remote Bootstrap enables remote distribution of information required for connecting a device to communication partners [Reinfurt et al. 2017b].
[5]Remote Device Management provides functionality to remotely install and update software, enable or disable functionality, and otherwise manage devices [Reinfurt et al. 2017a].
[6]A Device Gateway allows devices which do not support a specific network to participate in this network by acting as a translator [Reinfurt et al. 2016, 2017a].
[7]A Device Shadow provides a virtual representation of a physical device, which is synchronized with the device whenever it is online. Communication partners can communicate with the device through its shadow, even if it is offline [Reinfurt et al. 2016, 2017a].

4.2 Outbound-Only Connection



Attackers may send unsolicited connection requests to devices to get them to connect to an infected communication partner. To prevent this, configure devices so that only they initiate connections. Block all incoming connection requests.

Context: Devices and other components often have to communicate bidirectionally: Devices send data to other devices or services for monitoring, processing, analysis, or storage purposes. Devices also receive commands which control actuators attached to them or trigger functionality built into them. This communication poses a security risk as attackers might try to gain access to devices to manipulate them.

Problem: Devices are a target for attackers who try to gain access to their network. They may send unsolicited communication requests to the devices to get them to connect to an infected communication partner or to misuse them.

Forces:

—**Security:** Unsolicited request must be ignored.

—**Functionality:** The original functionality of the devices has to stay intact. They have to be able to send their data and they also have to be able to receive commands from other communication partners.

—**Constraints:** Many IoT devices are constrained in their capabilities. This limits the viability of more sophisticated solutions for controlling communication, such as Firewalls or Packet Filters [Schumacher 2003].

Solution: Program devices so that only they initiate connections. Deny all incoming communication requests that are not responses to a connection already created by the device.

Solution Details: When using Outbound-Only Connection, devices initiate all connections to other communication partners as shown in Figure 3 A. This requires that the devices know all the possible communication partners. These partners could be stored in some kind of list, configuration file, or database, which is stored on the device during bootstrapping and may be changed later on.

The device may use certain events, such as sensor reading above a threshold, a schedule, or other parameters to work out when it needs to communicate with others. Then it initiates the connection and sends the messages it wants to send to the communication partners, as shown at number 1 in Figure 3 B. It may also retrieve messages from these partners. This may be done in different ways. It may retrieve pending messages each time it creates a connection to send a message itself. It may also poll for pending messages by periodically creating a connection, even if it does not have any messages to send.

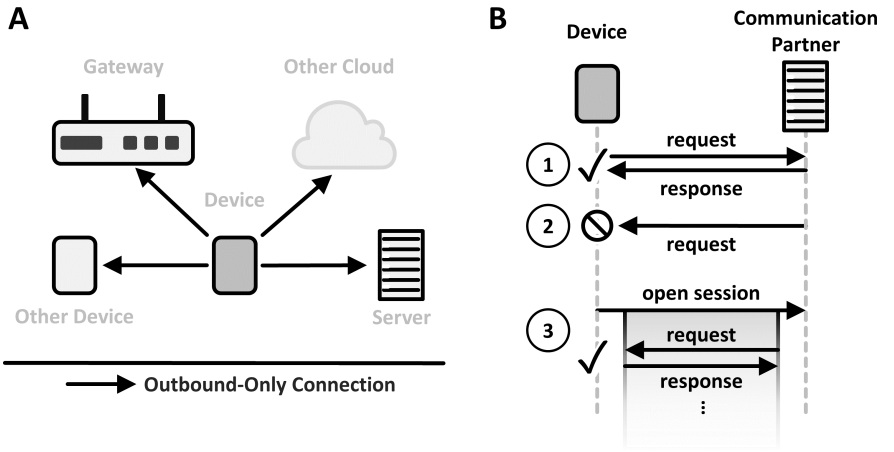


Fig. 3. Solution sketch of the OUTBOUND-ONLY CONNECTION pattern.

Another way is creating a long lasting session, which allows its communication partner to send messages as he wants as long as the connection remains active, as shown at number 3 in Figure 3 B. As such, OUTBOUND-ONLY CONNECTION does not restrict bidirectional communication once a connection is established but it does restrict the initial creation of a connection from the outside.

As all incoming requests that are not an answer to a previously send request are categorically denied, as shown at number 2 in Figure 3 B, there are fewer vectors of attack. Additionally, no ports have to be opened in firewalls which further decreases the attack surface. But this also prevents communication partners to contact the device if it has not already established a connection. They have to wait until the device decides to create a connection which may be too late for some situations.

Benefits:

—**Security:** The device ignores all unsolicited incoming traffic. This filters potentially malicious communication.

—**No Open Ports:** When all communication is initiated from the device outwards, no ports have to be opened. This simplifies deployment and removes security holes.

—**Firewalls:** Devices that only communicate outbound can successfully connect through firewalls and Network Address Translation (NAT).

—**Low Energy:** The device decides when to establish a connection. This allows the device to optimize communication for low energy usage, which is useful for PERIOD ENERGY-LIMITED DEVICES, LIFETIME ENERGY-LIMITED DEVICES, and ENERGY-HARVESTING DEVICES [Reinfurt et al. 2017c].

Drawbacks:

—**Reachability:** The device may not be connected at the point in time when its communication partner wants to communicate with it. The partner has to wait until the device decides to connect. A notification mechanism like the DEVICE WAKEUP TRIGGER[8] may be used to tell the device to reconnect.

—**Checking:** Unless the device has created a session with them, there is no way for other components to check the device's status for reliability purposes. The device could send heartbeat messages at regular intervals to inform others that everything is working normally.

[8]A DEVICE WAKEUP TRIGGER uses a secondary low-power communication channel to wake up sleeping devices if needed [Reinfurt et al. 2016, 2017a].

—**Security:** Using Outbound-Only Connection does not protect against a hacked communication partner or a middleman attack.

Related Patterns:

—**Factory Bootstrap, On-Site Bootstrap, or Remote Bootstrap:** These patterns may be used to specify allowed communication partners on the devices [Reinfurt et al. 2017b].

—**Remote Device Management:** After bootstrapping, communication partners can be changed with Remote Device Management [Reinfurt et al. 2017a].

—**Device Wakeup Trigger:** To get a sleeping device to connect when needed, a Device Wakeup Trigger can be used [Reinfurt et al. 2016, 2017a].

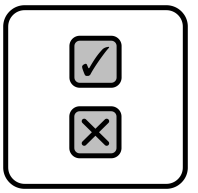
—**Period Energy-Limited Devices[9], Lifetime Energy-Limited Devices[10], or Energy Harvesting Devices[11]:** These devices may use Outbound-Only Connection to manage their communication to get the most out of their limited energy supply [Reinfurt et al. 2017c].

—**Trusted Communication Partner:** Outbound-Only Connection can be combined with Trusted Communication Partner for increased security. In this case, devices only communicate with preconfigured Trusted Communication Partners and they always initiate the communication.

Known Uses: Microsoft's Azure IoT Hub documentation mentions that devices do not accept unsolicited communication requests. Rather, they initiate all connections themselves. To receive commands they regularly establish a connection to the backend [Microsoft 2015]. Devices using the ThingWorx AlwaysOn protocol will initiate all connections. Thus, opening ports in firewalls to allow them to communicate can be avoided. The established connection is persistent and allows the platform to send messages to devices [ThingWorx 2016]. The OMA Device Management Protocol sessions are always initiated by the clients. But servers may trigger clients to initiate a session by sending a notification [Open Mobile Alliance 2015]. Devices using the MQTT protocol initiate all connections to an MQTT broker [OASIS 2014].

[9] A Period Energy Limited Device uses replaceable or rechargeable batteries as power source [Reinfurt et al. 2017c].
[10] A Lifetime Energy-Limited Device uses a build in battery as power source which cannot be renewed once depleted [Reinfurt et al. 2017c].
[11] An Energy Harvesting Device uses environmental energy sources, such as solar or wind, to power itself [Reinfurt et al. 2017c].

4.3 Permission Control



Device owners are afraid to completely hand over access to their devices and data to others. Allow device owners to choose which functionality and data a backend server or other communication partners are allowed to access when they first connect to the device. Ensure that these choices are respected and updated when something changes.

Aliases: Explicit Choice

Context: In the IoT there are often multiple stakeholders involved, such as device and platform manufacturers, owners, and users. Building and using IoT solutions often requires communication between the components of these stakeholders, for example between devices and a backend server, as data and functionality are shared.

Problem: Device owners are afraid to completely hand over access to their devices and data to third parties without any control. Often, it is unclear what data a device shares with communication partners or what others can access and control.

Forces:

—**Choice:** Device owners should have a choice as to what rights they grant a particular communication partner. This may include scenarios where they do only grant limited rights with which communication partners have to work. The available choices depend on the use case and its semantics.

—**Granularity:** Different levels of granularity may be required depending on the use case.

—**Enforcement:** The mechanism has to protect users' rights. If a user chooses not to allow something it has to make sure that this choice is enforced.

—**Simplicity:** Such a mechanism has to be easy to use for end users. It will not be adopted or it will be wrongly used if it is too complicated.

—**Updates:** Capabilities of devices and the communication partners which access them change. In such a case, the user may change his mind about previous choices or has to make new choices. The backend server and other systems have to make sure to work with the current set of choices at all times.

Solution: When first connecting a device to a backend server, require an explicit choice from the user regarding which functionality and data the backend and other communication partners are allowed to use. Build your backend server so that it adheres to these choices. Require the user to confirm these choices if something on the device or the backend server changes.

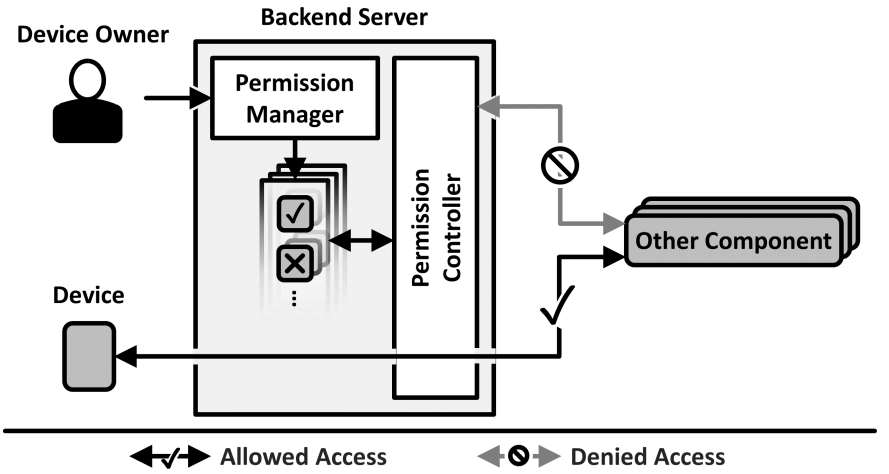


Fig. 4. Solution sketch of the PERMISSION CONTROL pattern.

Solution Details: Implementing PERMISSION CONTROL provides device owners with the means to control what external communication partners are allowed to do with a device or the data it produces. Before a device is added to the backend server or before another communication partner is allowed to access the device, the device owner is presented with a Permission Manager interface where he has to explicitly allow or deny the server or other communication partners some rights, as shown in Figure 4. This part of PERMISSION CONTROL thus implements the existing AUTHORIZATION pattern [Schumacher et al. 2005]. Make these choices granular enough to allow the device owner control, but not so granular that it is overwhelming.

These granted authorizations are persistently stored for later reference. From then on, they have to be enforced each time that another component wants to access the functionality of a particular device. This is the job of the Permission Controller, also shown in Figure 4, which implements the already existing REFERENCE MONITOR pattern [Schumacher et al. 2005].

Devices or components can change, for example, when a device gets new functionality through a firmware update or when a component is extended in functionality and now needs additional permissions. Thus, every time such a change occurs, you should require the user to confirm or alter his previously granted permissions. Additionally, allow the user to get an overview over his granted permissions and change them at any time.

Benefits:

—**Explicit Choice:** Users have to specifically give allowance to a device or service so that it may be used. Without this allowance, some or all features will be blocked. This makes it hard for communication partners to do something which the user does not want.

—**Transparency:** Since all choice are made explicit, there is more transparency about who has access to what.

—**Legal:** Communication partners can demonstrate that they are abiding by the user's expressed consent and therefore cannot be accused of breaking privacy laws or other regulations.

Drawbacks:

—**Transparency:** It may not be obvious to the device owner for what exactly a component requires access to some device functionality. It may also not be obvious which functionality is strictly required for the component to work. If possible, give the user an explanation why access to a particular function is requested and mark required functionality as such.

—**Agreement Blindness:** Users are accustomed to blindly agreeing to terms of services or similar documents without reading the content. A similar effect could happen when a lot of new communication partners are added all the time. Users may blindly agree to the choices requested by new communication partners.

—**Trust:** The user has to trust the backend server that it will respect his choices. This may not be the case for third party providers which are intransparent about their practices. One option is to store and enforce the permission settings on the devices themselves. For devices where this is not implemented or not possible, the user could employ a Personal Zone Hub under his own control instead.

—**Scalability:** Accepting a choice dialog for each new device may be a lot of work when devices are installed. If these devices are similar in functionality, they may be grouped and a choice could be presented once for the whole group instead of every single device. Allowing the user to provide default settings that are applied to every new connection can also help. Additionally, allowing the user to bulk edit permissions makes handling large amount of devices and components easier.

—**Restricted Functionality:** Communication partners may have restricted or no functionality at all if some or all rights are not allowed. Instead of not just working if one right is not given, communication partners should be designed to offer some functionality with limited rights.

Related Patterns:

—**Authorization:** The device owner creates Authorization through the Permission Manager component while setting up the device [Schumacher et al. 2005].

—**Reference Monitor:** The Permission Controller component implements the Reference Monitor pattern [Schumacher et al. 2005].

—**Automatic Client-Driven Registration[12], Automatic Server-Driven Registration[13], or Manual User-Driven Registration[14]:** Permission Control may be used during these registration processes.

—**Limited Access** deals with the other side of Permission Control. When access to functionality is limited by Permission Control, showing the potentially available functionality to users without access to it may be confusing or even a security problem. Thus, only show users what they can actually use by modifying the GUI based on their access rights [Schumacher et al. 2005].

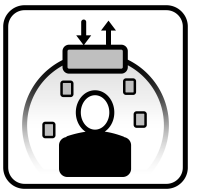
Known Uses: On both Android and iOS, app developers have to use interfaces provided by the operating systems to access smartphone features. Once a user installs such an app, a screen is shown where all permissions requested by the app are listed. Here, the user has a choice to grant or deny these permissions to the app. The user may also change these permissions at a later point in time. When an app is updated and requests new permissions, the screen is shown again [Apple 2017; Google 2017]. Samsung's SmartThings Platform has a similar approach as users have to explicitly select the devices which the platform should be able to control. Devices which are not selected cannot be controlled by the platform [SmartThings 2015]. In OAuthing, users must explicitly authorize all systems that need to get data from sensors and send commands to actuators [Fremantle and Aziz 2016].

[12]Automatic Client-Driven Registration allows devices to register themselves by calling a registration API and providing the required information [Reinfurt et al. 2017b].

[13]Automatic Server-Driven Registration uses out-of-band communication mechanisms to inform a backend server of a new device. The server then contacts the device to initiate the registration process [Reinfurt et al. 2017b].

[14]Manual User-Driven Registration requires users to register new devices by providing the required information via an API or GUI [Reinfurt et al. 2017b].

4.4 Personal Zone Hub



Managing the permissions, data sharing, and control of devices across multiple gateways and clouds is complex. Create a Personal Zone Hub which handles such settings in a central place under the control of the device owner or a trusted third party. Allow owners to selectively provide access to their devices and data through this hub.

Context: Devices, services, applications, and the data they produce usually have an owner, such as a person or an organization. Together, these components create the personal zone of their owner. This personal zone may contain all kinds of devices, applications, and data that may allow others to identify, track, or gain insights about the owner. Thus, the owner is usually interested in controlling and restricting access to his personal zone.

Problem: Users have an increasing number of IoT devices. Managing the permissions, data sharing and control of these devices across multiple gateways and cloud systems is complex.

Forces:

—**Ownership:** You own all the components and data in your personal zone and thus should be in charge when it comes to granting access to them.

—**Manageability:** Handling access to the devices in your personal zone on a per device basis may be cumbersome. Each device may have a different mechanism for handling access.

—**Trust:** The user may not trust third parties with the management of his devices, apps, software, and data.

—**Decentralization:** Having only one party being responsible for matters of authorization may be a disadvantage. It would concentrate these capabilities in one place and allow this party to take advantage of its position. It may also lead to vendor lock-in.

Solution: Create a Personal Zone Hub which unifies the management and control of all devices, services, apps, and data of one user into a single, trusted system. Make the hub permanently addressable. Allow the user to selectively share access to some or all the data and functionality encompassed by the zone.

Solution Details: A Personal Zone Hub acts as the entry point and coordinating agent of a personal zone. A personal zone encompasses the pool of digital resources belonging to one person. Conceptually it creates a logical boundary around a person and all of their devices, apps, services, and data, as shown in Figure 5. Technically, it combines multiple types of devices and networks, connected or disconnected, as well as cloud providers through a virtual network.

The Personal Zone Hub controls access to the personal zone through a logical Firewall [Schumacher 2003], where the owner may allow different parties to access some or all components, functionality, or data in the personal zone. Thus, the Personal Zone Hub may be seen as a digital representation of the owner. As each owner is unique and

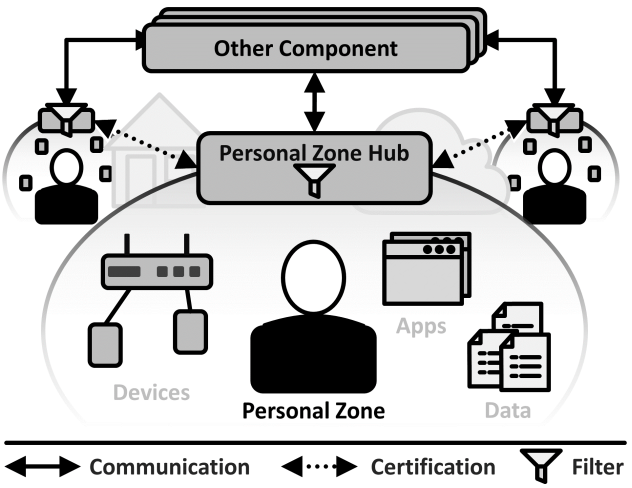


Fig. 5. Solution sketch of the PERSONAL ZONE HUB pattern.

should be able to be uniquely identified, the PERSONAL ZONE HUB needs to be permanently addressable with a unique routable URI, or via a discovery service. This may be achieved by placing the PERSONAL ZONE HUB into a cloud, where it is accessible for other communication partners. But it may also be hosted locally, if cloud providers are not trusted, as long as it is web routable.

The PERSONAL ZONE HUB may act as a certification authority for devices in the personal zone. This requires a model of user authorization, where a user can explicitly add local devices. In some cases, devices can use these certificates to communicate as peers in the case where the PERSONAL ZONE HUB is not reachable.

Benefits:

—**Distributed Certification Authority:** Certification authority is distributed over the PERSONAL ZONE HUBS and not in the control of one single party. This makes it more resilient against failures and prevents vendor lock-in.

—**Centralized Access Control:** Despite the certification authority being distributed, the control over access to all devices in a personal zone is centralized into the PERSONAL ZONE HUB. This simplifies the management of all components in a users personal zone, as users no longer have to go to different interfaces to manage access to their components.

—**User Control:** A user can run his own PERSONAL ZONE HUB to maximize control.

—**Trust:** Because users have control over the PERSONAL ZONE HUB, there can be greater trust. For example, the source code of the hub may be open, allowing many eyes to validate that it behaves as expected.

Drawbacks:

—**Effort:** Running once own PERSONAL ZONE HUB may be a lot of effort and require technical skills and knowledge. Using a third party provider would make this easier.

—**Trust:** Using PERSONAL ZONE HUB provided by a third party requires trust in this party.

—**Single Point of Failure:** As the PERSONAL ZONE HUB is the single central point of control of the components in its personal zone, it is also a single point of failure when it malfunctions. For critical applications, multiple redundant PERSONAL ZONE HUBS with failover could be used.

Variants:

—**Anonymizing Personal Zone Hub:** Devices may be fingerprinted because they exhibit enough complexity to differ from other devices. For example, it has been shown that communication behavior, accelerometers, batteries, microphones, and other sensors can be uniquely fingerprinted. A Personal Zone Hub may implement additional functionality, such as information hiding strategies, to prevent this. For example, it could only publish averaged data rather than raw data. Additionally, a Personal Zone Hub may also hide the IP addresses and other metadata of the devices in its zone. These approaches can prevent fingerprinting and enhance privacy for the owner of the device and data controlled by an Anonymizing Personal Zone Hub.

Related Patterns:

—**Permission Control:** A Personal Zone Hub can provide its user with Permission Control over who has access to his devices and data.

—**Trusted Communication Partner:** The Personal Zone Hub acts as a Trusted Communication Partner for the devices in its personal zone. Besides, it may only communicate outwards with Trusted Communication Partners that were previously allowed by the owner.

—**Device Gateway:** A Personal Zone Hub may be located on a Device Gateway if all user devices are connected to this gateway.

—**Single Access Point:** This is a pattern similar to the Personal Zone Hub but without the aspect of distributed certification and anonymization. It describes, how a system containing several components can be made accessible to others while protecting it from misuse and damage. A Single Access Point is used to check incoming request and blocks them if they are not valid [Schumacher et al. 2005].

—**Controlled Virtual Address Space:** The Controlled Virtual Address Space pattern describes a related concept in the domain of operating system processes [Fernandez 2013].

—**Façade:** A Personal Zone Hub can provide others a high-level interface to all the digital resources in the user's personal zone, thus implementing the Façade pattern [Gamma et al. 1995].

Known Uses: The Webinos project describes a personal zone hub as a virtual network and logical firewall which protects its user's personal zone. A personal zone encompasses all user devices, service, apps, and data. The hub, which may be cloud based or installed in your home, allows a user to control access to his devices and data without involving a central party [Allott 2014]. Fremantle published a similar concept, in which a personal cloud middleware is used to control the sharing of personal data between devices and the cloud. The user's identity is handled separately from the devices, which allows the user to share access to his devices anonymously [Fremantle 2016]. The OAuthing middleware implements this concept and uses the Docker cloud system to run a Personal Zone Hub for each user in the cloud. Summarization and filtering are envisioned for this system [Fremantle and Aziz 2016]. The dowse project aims to replace or supplement ISP provided routers and modems with the dowse hub. The hub is a transparent proxy under full control of the user and runs applications on behalf of the user, which can interact with and control the local sphere of devices, thus allowing the user more control over his devices and data [Dyne.org Foundation and Waag Society 2014].

4.5 Whitelist

Available privileges may be abused, but not all potential abusers are known beforehand. To minimize the risk of abuse, add identifiers of all trusted communication partners to a WHITELIST. *Block the privileges controlled by the* WHITELIST *for those who are not on it.*

Aliases: Filter

Context: For IoT systems to work, communication partners need privileges to access parts of others. Some examples include devices that need access privileges on the backend server to send it their data, or applications that need privileges to access certain functionality or to be executed on a device or a server. Some privileges may be freely available by default, but others should only be made available to trusted partners. In both cases, abuse is a problem which has to be handled. This can be done by using the AUTHENTICATOR pattern [Schumacher et al. 2005], which relies on cryptographic mechanisms and tokens but requires some processing power. Thus, this may not be applicable everywhere in IoT systems, where very constrained devices may not have enough resources to execute such a mechanism. Besides, tokens used by these mechanisms may be stolen and used on other devices, or devices may be stolen and used from other locations. Denial-of-service attacks might also be a problem. Such illegitimate request should be blocked to not waste too many resources on them.

Problem: Privileges may be abused and attackers may try to overwhelm a system with requests. Using a BLACKLIST to block known abusive communication partners helps, but such a list is never complete. There has to be a mechanism to limit access to known communication partners to minimize the possible range of attacks.

Forces:

—**Prevention:** The mechanism has to prevent future abuse where possible.

—**Identification:** To be able to grant only certain communication partners the controlled privileges, is has to be possible to identify them reliably.

—**Control**: The mechanism has to provide control over what communication partners or actions are deemed trustworthy. It has to allow manual adjustment if these circumstances change.

—**Efficiency:** The mechanism has to be resource efficient. It should not have a large impact on the normal operation of an IoT system. Thus, it should be able to run on constrained devices.

Solution: Implement a WHITELIST to which identifiers of communication partners may be added with an administrative interface. Implement checks for every privilege which the WHITELIST controls. If such a privilege is requested and the requester's identifier is on the WHITELIST, allow it to proceed. If not, deny the request.

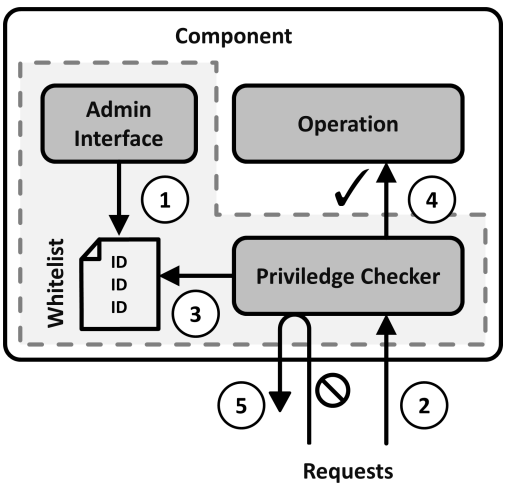


Fig. 6. Solution sketch of the WHITELIST pattern.

Solution Details: A WHITELIST is a list of entries which identify communication partners that are allowed to have a certain privilege. Usually, one WHITELIST controls one privilege. Examples of such a privilege are access to a network, system, or device, or permission to execute an action or read data.

To be able to grant these privileges to a communication partner, someone has to trust this partner and add an identifier for that partner to the WHITELIST, as shown in Step 1 in Figure 6. If a request to access an operation guarded by a WHITELIST is received, as shown in Step 2 in Figure 6, it can now be checked against the list (Step 3). If the requester's ID is on the WHITELIST, the request is passed on (Step 4), otherwise, it is blocked (Step 5).

This requires two things: An administrative interface which allows WHITELIST items to be managed, and an identity of a trusted partner. The interface may be as simple as a text file which contains identifiers separated by new lines or other delimiting characters. But some additional elements may be added for better usability and manageability. A checkmark for each item allows users to quickly enable or disable single entries. The ability to use wildcards to add ranges of identifiers simplifies the management of large numbers of identities.

Choosing the identifier to be used on the WHITELIST is important. It should be an identifier common to all communication partners that are expected to be checked by the WHITELIST. Where this is not possible, multiple WHITELISTS or one WHITELIST which supports multiple identifier types may be used. Ideally, such an identifier is globally unique and cannot be spoofed, but this not easy in reality. The exact type of identifier depends on the use case and can vary from IP Addresses to complex device fingerprints (see the Variants section).

To be effective, the WHITELIST has to apply to all connections, irrespective of their protocol. Besides, it has to work not only on backend servers but also on small devices. Similar to backend servers, these devices are also targets of abusive behavior. Because of their often limited performance, such behavior may quickly render them unusable.

Benefits:

—**Security by Default:** A WHITELIST is secure by default as all unknown requesters are denied the privilege which the WHITELIST controls.

—**Explicit Allowance:** The allowance is made explicit. It is clear by looking at the list which communication partners are trusted.

—**Trust:** Only trusted communication partners are allowed access, which limits the possible sources of abuse.

—**Flexibility:** Entries can be changed easily to accommodate changing circumstances.

—**Simplicity:** A WHITELIST is both simple to implement and simple to use.

—**Completeness:** Each communication partner that has to have a certain privilege has to be put on the WHITELIST. Thus the list always provides a complete view of who is granted the privilege controlled by the WHITELIST.

Drawbacks:

—**Implicit Blocking:** Communication partners not on the list are implicitly blocked which may lead to confusion. Use a BLACKLIST to make blocking explicit.

—**Identifier Uniqueness:** A WHITELIST only works reliably if the identifier used is unique to the communication partner and cannot be changed or spoofed. Depending on the use case, different identifiers are applicable (see the Variants section). If possible, use an identifier that cannot be easily changed or spoofed.

—**Granularity:** A whitelist usually only provides a binary choice of either allowing or blocking. For more fine-grained control, use the AUTHORIZATION or ROLE-BASED ACCESS CONTROL patterns [Schumacher et al. 2005].

—**Manageability:** Managing a WHITELIST may be difficult for use cases where a lot of communication partners are present or where they change frequently. This requires a lot of manual labor. This may be simplified by using wildcards or by subscribing to a trusted third party service which provides managed lists of trusted communication partners.

—**False Positive:** Depending on the nature of the chosen identifier and the timeliness of the entries on the WHITELIST, it may come to some false positives, where entries exist on the list which should have been removed.

Variants:

—**GREYLIST:** A GREYLIST is a list to which new communication partners are automatically added if they are not yet on a WHITELIST (or BLACKLIST). It may grant these communication partners some limited privileges, for example, a subset of privileges granted by a WHITELIST that are not risky for unknown communication partners to have. It may also only be used as an automatically generated list of potential entries that should be added to a WHITELIST (or BLACKLIST). Adding an entry from the GREYLIST to another list should automatically remove it from the GREYLIST.

—**NETWORK ADDRESS WHITELIST:** A NETWORK ADDRESS WHITELIST is a simple WHITELIST that uses network addresses as identifier. As every communication partner has to have at least some kind of network address to be able to communicate, e.g., an IP or MAC addresses, this is often the easiest way to implement a WHITELIST. However, network addresses can change and can be spoofed, which decreases the reliability of this method.

—**FINGERPRINT WHITELIST:** A FINGERPRINT WHITELIST relies on fingerprints to identify which communication partners are granted the privileges controlled by the FINGERPRINT WHITELIST. A fingerprint is a device-specific signature generated by collecting stable and hard to forge patterns during communication with the device, such as variations in clock skew, communication sequences, and other network behavior. Such a signature is hard to forge and provides a more reliable way to identify devices compared to network addresses, which can be changed or spoofed.

—**APPLICATION WHITELIST:** An APPLICATION WHITELIST controls which applications can be executed on a device or server. For every application that should be whitelisted, a hash of the application code is calculated and stored on the list as identifier. Before an application is started, the hash of the application code is calculated and compared to the stored hashes on the list. If it does not match a hash on list, the application is blocked from execution.

Related Patterns:

—**BLACKLIST:** A BLACKLIST is the opposite of a WHITELIST, as it blocks all entries on the list. It may be used together with a WHITELIST to explicitly block a communication partner who has behaved badly in the past.

—**TRUSTED COMMUNICATION PARTNER:** A WHITELIST can be used to implement TRUSTED COMMUNICATION PARTNER.

—**FAIL SECURELY:** A WHITELIST is an example of a system that is designed to FAIL SECURELY [Romanosky 2001]. If a legitimate communication partner is not on the WHITELIST, its communication attempts are blocked, which is a failure but does not negatively impact the security of the system.

—**Firewall:** A Firewall is another way to control network access to resources [Schumacher 2003]. Compared to a Whitelist, it offers more sophisticated means for identifying, inspecting, and blocking communication, but it does not work on the level of single resources or functions and also requires more resources, which may not be available on constrained devices in the IoT.

—**Whitelisting Firewall:** A Whitelisting Firewall uses a Whitelist to only allow access to approved websites [Villarreal et al. 2013].

—**Factory Bootstrap, On-Site Bootstrap, or Remote Bootstrap:** These patterns may be used to initially set up the entries on the Whitelist [Reinfurt et al. 2017b].

—**Remote Device Management:** After bootstrapping, entries on the Whitelist can be changed with Remote Device Management [Reinfurt et al. 2017a].

Known Uses: Many IoT platforms support the concept of whitelisting. AWS IoT by default does not allow any entity to execute actions. The first have to be whitelisted by generating and applying a policy to them [Amazon Web Services 2015]. Azure IoT Hub supports both whitelisting and blacklisting of individual devices [Microsoft 2015]. Wind River Helix Device Cloud supports whitelisting on devices [Wind River 2015]. Ayla Networks' IoT Platform allows manufacturers to upload serial numbers of modules to effectively whitelist them for being used on the platform [Ayla Networks 2015]. The Dowse concept envisions owners of the Dowse hub whitelisting guest devices to grant them more privileges [Dyne.org Foundation and Waag Society 2014].

Device fingerprinting, which can generate device identifiers for whitelisting, has been researched for a while. It can be done actively or passively without a device knowing about it and on different network layers [Xu et al. 2016]. It can also be done remotely over large distances, through firewalls, and independent of location and network technology [Kohno et al. 2005]. Fingerbank is an online service which stores such fingerprints (which can also be created with their collector tool) and makes them accessible to other services, for example, PacketFence, which can grant network access based on fingerprints [Inverse 2018].

Linux has a subsystem called Integrity Measurement Architecture (IMA), which calculates hashes of applications and other files before they are loaded. It can validate if these hashes are present on a predefined list, thus implementing an Application Whitelist [Gentoo Foundation 2017]. McAfee Embedded Control supports similar functionality on embedded systems [McAfee 2013].

4.6 Blacklist

Available privileges may be abused. To stop this, implement a Blacklist *to which identifiers of abusive communication partners can be added. For each partner that requests a privilege first check this list and deny it if its identifier is found on it.*

Aliases: Filter

Context: For IoT systems to work, communication partners need privileges to access parts of others. Some examples include devices that need access privileges on the backend server to send it their data, or applications that need privileges to access certain functionality or to be executed on a device or a server. Some privileges may have to be explicitly granted, others may be freely available. In the latter case, abuse is a problem which has to be handled. But even when privileges are explicitly granted, abuse is still possible, for example if a device is hijacked by a malicious intruder.

Problem: Privileges, no matter if freely available or explicitly granted, may be abused. There has to be a mechanism to stop existing abuse and to limit potential future abuse.

Forces:

—**Intervention:** Abuse may already be happening. The mechanism has to be able to intervene and stop existing abuse.

—**Prevention:** The mechanism has to prevent future abuse where possible.

—**Identification:** To be able to prevent abuse, it must be possible to identify abusers. But identification may be imprecise or easy to forge.

—**Control:** The mechanism has to provide control over what communication partners or actions are deemed abusive. It has to allow manual adjustment if these circumstances change.

—**Simplicity:** The mechanism should be easy to understand and easy to use.

—**Efficiency:** The mechanism has to be resource efficient. It should not have a large impact on the normal operation of an IoT system. It should be able to run on constrained devices.

Solution: Implement a Blacklist to which identifiers of abusive communication partners can be added with an administrative interface. Implement checks for every privilege that the Blacklist controls. If such a privilege is requested and the identifier of the requesting communication partner is on the Blacklist, block it. If not, allow it to proceed.

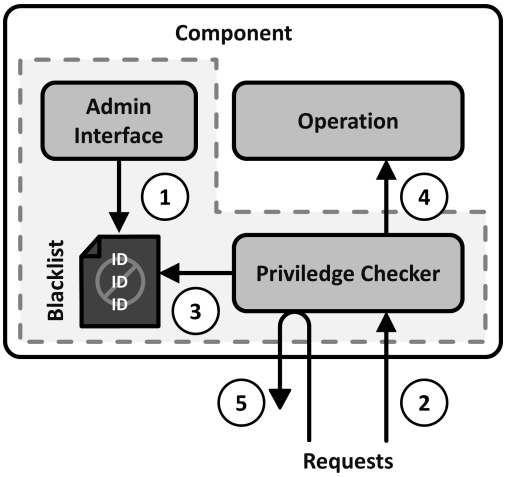


Fig. 7. Solution sketch of the BLACKLIST pattern.

Solution Details: A BLACKLIST is a list of entries, which identify communication partners that are not allowed to have a certain privilege. Usually, one BLACKLIST controls one privilege. Examples of such a privilege are access to a network, system, or device, or permission to execute an action or read data.

To be able to deny these privileges to an entity, someone has to identify this entity as abusive and add an identifier for that entity to the BLACKLIST, as shown in Step 1 in Figure 7. If a request to access an operation guarded by a BLACKLIST is received, as shown in Step 2 in Figure 7, it can now be checked against the list (Step 3). If the requester's ID is not on the BLACKLIST, the request is passed on (Step 4), otherwise, it is blocked (Step 5).

This requires two things: An administrative interface which allows BLACKLIST items to be managed, and an identity of an abusive entity. The interface may be as simple as a text file which contains identifiers separated by new lines or other delimiting characters. Some elements may be added for better usability and manageability. A checkmark for each item allows users to quickly enable and disable single entries. A text field for comments enables user to describe the entry in more detail, which may be useful if the identifiers themselves do not provide much information. The ability to use wildcards to add ranges of identifiers simplifies the management of large numbers of identities.

Choosing the identifier to be used on the BLACKLIST is important. It should be an identifier common to all entities that are expected to be checked by the BLACKLIST. Where this is not possible, multiple BLACKLISTS or one BLACKLIST which supports multiple identifier types may be used. Ideally, such an identifier is globally unique and cannot be spoofed. But this is not easy in reality. The exact type of identifier depends on the use case and can vary from IP Addresses to complex device fingerprints (see the Variants section).

To be effective, the BLACKLIST has to apply to all connections, irrespective of protocol. Besides, it has to work not only on backend servers, but also on small devices. Similar to backend servers, these devices are also targets of abusive behavior. Because of there often limited performance, such behavior may quickly render them unusable.

It usually makes sense to combine multiple BLACKLISTS. A general BLACKLIST provided and maintained by a platform provider or other third parties about generally known malicious communication partners may be combined with a personal BLACKLIST, which contains additional sources of abuse. This combines the shared knowledge of a general BLACKLIST with the possibility to easily add personal entries.

Benefits:

—**Security:** Adding known malicious communication partners to the BLACKLIST effectively blocks them.

—**Flexibility:** Entries can be changed easily to accommodate changing circumstances.

—**Simplicity:** A Blacklist is both simple to implement and simple to understand for users.

—**Explicit Blocking:** A Blacklist makes blocking explicit. It is clear by looking at the list which communication partners are blocked.

—**Outdated Entries:** Outdated entries produces false positives which might be inconvenient but do not decrease security.

Drawbacks:

—**Implicit Allowance:** Entries not on the list are implicitly allowed which may lead to confusion. Use a Whitelist to make allowance explicit.

—**Identifier Uniqueness:** A Blacklist only works reliably if the identifier used is unique to the communication partner and cannot be changed or spoofed. Depending on the use case, different identifiers are applicable (see the Variants section). If possible, use an identifier that cannot be easily changed or spoofed.

—**Manageability:** Constantly changing threats means that a Blacklist has to be updated regularly to be effective. Some threats currently on a Blacklist may also only be temporarily malicious and should be removed after a while. This requires a lot of manual labor. Subscribing to some kind of service that manages such lists increases the efficiency of list maintenance, as work and knowledge is shared.

—**False Positive:** Because it is hard to find unique identifiers it may come to false positives, i.e., something is on the list which is not (anymore) dangerous.

—**Completeness:** It is impossible to know all potentially malicious communication partners. A Blacklist will never be complete, and, thus, does not provide complete protection.

Variants:

—**Greylist:** A Greylist is a list to which new communication partners are automatically added if they are not yet on a Blacklist (or Whitelist). It may block some privileges for these communication partners, for example, a subset of privileges blocked by a Blacklist that are too risky for unknown communication partners to have. It may also only be used as an automatically generated list of potential entries that should be added to a Blacklist (or Whitelist). Adding an entry from the Greylist to another list should automatically remove it from the Greylist.

—**Network Address Blacklist:** A Network Address Blacklist is a simple Blacklist that uses network addresses as identifier [Kienzle et al. 2002]. As every communication partner has to have at least some kind of network address to be able to communicate, e.g., an IP or MAC addresses, this is often the easiest way to implement a Blacklist. However, network addresses can change and can be spoofed, which decreases the reliability of this method.

—**Fingerprint Blacklist:** A Fingerprint Blacklist relies on fingerprints to identify which communication partners should be blocked from the privileges controlled by the Fingerprint Blacklist. A fingerprint is a device-specific signature generated by collecting stable and hard to forge patterns during communication with the device, such as variations in clock skew, communication sequences, and other network behavior. Such a signature is hard to forge and provides a more reliable way to identify devices compared to network addresses, which can be changed or spoofed.

Related Patterns:

—**Whitelist:** A Whitelist has the inverse functionality of a Blacklist. By default, all privileges are blocked unless the communication partner is listed on the Whitelist. A Whitelist may be used together with a Blacklist to allow flexible control.

—**Factory Bootstrap, On-Site Bootstrap, or Remote Bootstrap:** These patterns may be used to initially set up the entries on the Blacklist [Reinfurt et al. 2017b].

—**Remote Device Management:** After bootstrapping, entries on the Blacklist can be changed with Remote Device Management [Reinfurt et al. 2017a].

Known Uses: GSMA operates a Blacklist service for mobile devices. When a user reports a device as lost or stolen to his mobile operator, the devices International Mobile Equipment Identity (IMEI) number is added to the Blacklist and distributed to other operators. This allows such devices to be blocked internationally. Organizations can subscribe to GSMA's Device Check service to get access to the Blacklist [GSMA 2014]. Microsoft's Azure IoT Hub allows backend services to Blacklist individual devices based on their unique security keys [Microsoft 2015]. Besides, Azure IoT Hub itself has an IP filter feature which allows blacklisting based on IPv4 addresses [Oltean Beatrice 2016]. Device Authority's D-FACTOR Device Authentication Engine supports a blacklisting feature to control device access [Device Authority 2016].

Device fingerprinting, which can generate device identifiers for blacklisting, has been researched for a while. It can be done actively or passively without a device knowing about it and on different network layers [Xu et al. 2016]. It can also be done remotely over large distances, through firewalls, and independent of location and network technology [Kohno et al. 2005]. Fingerbank is an online service which stores such fingerprints (which can also be created with their collector tool) and makes them accessible to other services, for example, PacketFence, which can deny network access based on fingerprints [Inverse 2018].

5. SUMMARY AND CONCLUSION

The IoT is becoming a reality. However, due to a lack of common standards and development in silos, the solution landscape can be confusing. To help IoT architects, developers, and other interested individuals with understanding this space and designing and building their own solutions, we collected in our previous work IoT Patterns for devices [Reinfurt et al. 2017c], device communication and management [Reinfurt et al. 2016, 2017a], and device bootstrapping and registration [Reinfurt et al. 2017b]. But these patterns are not concerned with security, which is an important aspect in the IoT. Several security patterns exist in previous work, which can be applied to IoT systems We identified additional patterns, which we presented in this paper, which supplement the existing patterns. The Trusted Communication Partner and Outbound-Only Connection patterns help to minimize the attack surface of devices. The Permission Control and Personal Zone Hub patterns allow device owners some control over who has access to their devices and data. The Whitelist and Blacklist patterns provide devices, backend servers, and other components with a means to control access and prevent abuse. We have already collected additional IoT Patterns in several categories. In the future, we want to create a pattern language for IoT, which connects all of these patterns into a coherent form that can be used by IoT architects and developers to understand, design, and build IoT systems.

ACKNOWLEDGMENTS

We would like to thank our shepherd, Eduardo Fernandez, for the discussions and comments that helped to improve this paper. This work was partially funded by the BMWi projects SePiA.Pro (01MD16013F).

REFERENCES

Christopher Alexander, Sara Ishikawa, and Murray Silverstein. 1977. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, New York.

Nick Allott. 2014. Introductory Whitepaper: What is webinos. (2014). http://webinos.org/files/2014/06/webinos-foundation-whitepaper.pdf

Amazon Web Services. 2015. Authorization - AWS IoT (Beta). (2015). http://docs.aws.amazon.com/iot/latest/developerguide/authorization.html

Amazon Web Services. 2017. AWS IoT Device SDK - Amazon Web Services. (2017). https://aws.amazon.com/iot/sdk

Apple. 2017. Requesting Permission - Interaction - iOS Human Interface Guidelines. (2017). https://developer.apple.com/ios/human-interface-guidelines/interaction/requesting-permission/

Luigi Atzori, Antonio Iera, and Giacomo Morabito. 2010. The internet of things: A survey. *Computer networks* 54, 15 (2010), 2787–2805.

Ayla Networks. 2015. *The Security of Things*. Technical Report.

Gullena Satish Chandra. 2016. Pattern language for IoT applications. (2016).

James O. Coplien. 1996. *Software Patterns*. SIGS, New York, NY.

Device Authority. 2016. Establishing Trust with the Internet of Things. (2016). http://www.deviceauthority.com/assets/DA-IoT_solution_brief.pdf

Dyne.org Foundation and Waag Society. 2014. *Dowse Whitepaper*. Technical Report. https://files.dyne.org/dowse/dowse_whitepaper.pdf

Veli-Pekka Eloranta, Johannes Koskinen, Marko Leppänen, and Ville Reijonen. 2014a. *Designing distributed control systems: A pattern language approach*. Wiley, Hoboken, NJ.

Veli-Pekka Eloranta, Johannes Koskinen, Marko Leppänen, and Ville Reijonen. 2014b. Patterns for the Companion Website. (2014). http://media.wiley.com/product_ancillary/55/11186941/DOWNLOAD/website_patterns.pdf

Michael Falkenthal, Johanna Barzen, Uwe Breitenbücher, Christoph Fehling, and Frank Leymann. 2014a. Efficient Pattern Application: Validating the Concept of Solution Implementations in Different Domains. *International Journal on Advances in Software* 7, 3&4 (2014), 710–726. http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=ART-2014-13&engl=0

Michael Falkenthal, Johanna Barzen, Uwe Breitenbücher, Christoph Fehling, and Frank Leymann. 2014b. From Pattern Languages to Solution Implementations. In *Proceedings of the Sixth International Conferences on Pervasive Patterns and Applications (PATTERNS 2014)*. IARIA, Wilmington, DE, 12–21. http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2014-37&engl=0

Michael Falkenthal, Johanna Barzen, Uwe Breitenbücher, Christoph Fehling, Frank Leymann, Aristotelis Hadjakos, Frank Hentschel, and Heizo Schulze. 2016. Leveraging Pattern Application via Pattern Refinement. In *Proceedings of the International Conference on Pursuit of Pattern Languages for Societal Change (PURPLSOC)*.

Christoph Fehling, Johanna Barzen, Uwe Breitenbücher, and Frank Leymann. 2015a. A Process for Pattern Identification, Authoring, and Application. In *Proceedings of the 19th European Conference on Pattern Languages of Programs (EuroPLoP)*. ACM, New York, NY. DOI:http://dx.doi.org/10.1145/2721956.2721976

Christoph Fehling, Johanna Barzen, Michael Falkenthal, and Frank Leymann. 2015b. PatternPedia - Collaborative Pattern Identification and Authoring. In *PURPLSOC (In Pursuit of Pattern Languages for Societal Change): The Workshop 2014*. epubli GmbH, Berlin, 252–284.

Christoph Fehling, Frank Leymann, Ralph Retter, Walter Schupeck, and Peter Arbitter. 2014. *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications*. Springer, Wien. DOI:http://dx.doi.org/10.1007/978-3-7091-1568-8

Eduardo B. Fernandez. 2013. *Security Patterns in Practice: Designing Secure Architectures Using Software Patterns*. Wiley.

Paul Fremantle. 2016. Privacy-enhancing Federated Middleware for the Internet of Things. In *Proceedings of the Doctoral Symposium of the 17th International Middleware Conference*. ACM.

Paul Fremantle and Benjamin Aziz. 2016. OAuthing: privacy-enhancing federation for the Internet of Things. (2016).

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, Massachusetts.

Gentoo Foundation. 2017. Integrity Measurement Architecture. (2017). https://wiki.gentoo.org/wiki/Integrity_Measurement_Architecture

Google. 2017. Control your app permissions on Android 6.0 and up - Google Play Help. (2017). https://support.google.com/googleplay/answer/6270602?hl=en

GSMA. 2014. Don't Suspect Device Fraud - Know With Certainty. (2014). http://www.gsma.com/managedservices/wp-content/uploads/2014/04/Device-Blacklist-Brief.pdf

Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. 2013. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems* 29, 7 (2013), 1645–1660. DOI:http://dx.doi.org/10.1016/j.future.2013.01.010

Jasmin Guth, Uwe Breitenbücher, Michael Falkenthal, Frank Leymann, and Lukas Reinfurt. 2016. Comparison of IoT Platform Architectures: A Field Study based on a Reference Architecture. In *Proceedings of the International Conference on Cloudification of the Internet of Things (CIoT)*. IEEE.

Neil B. Harrison. 2006a. Advanced Pattern Writing: Patterns for Experienced Pattern Authors. In *Pattern languages of program design 5*. Software patterns series, Vol. 5. Addison-Wesley, Upper Saddler River, NJ, 433–452. http://www.europlop.net/sites/default/files/files/1_2003_Harrison_AdvancedPatternWriting.pdf

Neil B. Harrison. 2006b. The Language of Shepherding: A Pattern Language for Shepherds and Sheep. In *Pattern languages of program design 5*. Software patterns series, Vol. 5. Addison-Wesley, Upper Saddler River, NJ, 507–530. http://www.europlop.net/sites/default/files/files/3_TheLanguageOfShepherding1.pdf

Gregor Hohpe and Bobby Woolf. 2004. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley, Boston, Massachusetts.

IBM. 2016. MQTT connectivity for devices. (2016). https://console.ng.bluemix.net/docs/services/IoT/devices/mqtt.html

Inverse. 2018. Fingerbank: Device Fingerprints. (2018). https://fingerbank.org/

Isam Ishaq, David Carels, Girum Teklemariam, Jeroen Hoebeke, Floris Abeele, Eli Poorter, Ingrid Moerman, and Piet Demeester. 2013. IETF Standardization in the Field of the Internet of Things (IoT): A Survey. *Journal of Sensor and Actuator Networks* 2, 2 (2013), 235–287. DOI:http://dx.doi.org/10.3390/jsan2020235

Darrell M. Kienzle, Matthew C. Elder, David Tyree, and James Edwards-Hewitt. 2002. Security Patterns Repository Version 1.0. (2002). http://www.scrypt.net/~celer/securitypatterns/repository.pdf

Tadayoshi Kohno, Andre Broido, and Kimberly C. Claffy. 2005. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing* 2, 2 (2005), 93–108.

McAfee. 2013. McAfee Embedded Control. (2013). https://www.mcafee.com/us/resources/data-sheets/ds-embedded-control.pdf

Gerard Meszaros and Jim Doble. 1996. Metapatterns: A Pattern Language for Pattern Writing. In *Third Pattern Languages of Programming Conference*. Addison-Wesley. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.15.8186&rep=rep1&type=pdf

Microsoft. 2015. What is Azure IoT Hub? (2015). https://azure.microsoft.com/en-us/documentation/articles/iot-hub-what-is-iot-hub/

OASIS. 2014. MQTT Version 3.1.1. (2014). http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf

Oltean Beatrice. 2016. IP Filter. (2016). https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-ip-filtering

Open Mobile Alliance. 2015. OMA Device Management Protocol. (2015). http://www.openmobilealliance.org/release/DM/V2_0-20150122-C/OMA-TS-DM_Protocol-V2_0-20150122-C.pdf

Optigo Networks. 2016. Optigo Integrity. (2016). http://www.optigo.net/integrity/

Soheil Qanbari, Samim Pezeshki, Rozita Raisi, Samira Mahdizadeh, Rabee Rahimzadeh, Negar Behinaein, Fada Mahmoudi, Shiva Ayoubzadeh, Parham Fazlali, Keyvan Roshani, Azalia Yaghini, Mozhdeh Amiri, Ashkan Farivarmoheb, Arash Zamani, and Schahram Dustdar. 2016. IoT Design Patterns: Computational Constructs to Design, Build and Engineer Edge Applications. In *Proceedings of the First International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE, 277–282. DOI:http://dx.doi.org/10.1109/IoTDI.2015.18

Lukas Reinfurt, Uwe Breitenbücher, Michael Falkenthal, Frank Leymann, and Andreas Riegg. 2016. Internet of Things Patterns. In *Proceedings of the 21st European Conference on Pattern Languages of Programs (EuroPLoP)*. ACM. http://dl.acm.org/citation.cfm?id=3011789

Lukas Reinfurt, Uwe Breitenbücher, Michael Falkenthal, Frank Leymann, and Andreas Riegg. 2017a. Internet of Things Patterns for Communication and Management. *LNCS Transactions on Pattern Languages of Programming* (2017).

Lukas Reinfurt, Uwe Breitenbücher, Michael Falkenthal, Frank Leymann, and Andreas Riegg. 2017b. Internet of Things Patterns for Device Bootstrapping and Registration. In *Proceedings of the 22nd European Conference on Pattern Languages of Programs (EuroPLoP) (EuroPLoP '17)*. ACM, New York, NY, USA. https://dl.acm.org/citation.cfm?doid=3147704.3147721

Lukas Reinfurt, Uwe Breitenbücher, Michael Falkenthal, Frank Leymann, and Andreas Riegg. 2017c. Internet of Things Patterns for Devices. In *Proceedings of the Ninth International Conferences on Pervasive Patterns and Applications (PATTERNS) 2017*. Xpert Publishing Services, 117–126. https://www.thinkmind.org/index.php?view=article&articleid=patterns_2017_9_10_70019

Lukas Reinfurt, Michael Falkenthal, Uwe Breitenbücher, and Frank Leymann. 2017d. Applying IoT Patterns to Smart Factory Systems. In *Proceedings of the 11th Advanced Summer School on Service Oriented Computing*. IBM Research Division.

Sasha Romanosky. 2001. Security Design Patterns. (2001). http://www.cgisecurity.com/lib/securityDesignPatterns.html

Markus Schumacher. 2003. Firewall Patterns. In *Proceedings of the 8th European Conference on Pattern Languages of Programms (EuroPLoP '2003)*.

Markus Schumacher, Eduardo B. Fernandez, Duane Hybertson, Frank Buschmann, and Peter Sommerlad. 2005. *Security Patterns: Integrating Security and Systems Engineering*. Wiley.

Jatinder Singh, Thomas Pasquier, Jean Bacon, Hajoon Ko, and David Eyers. 2016. Twenty Security Considerations for Cloud-Supported Internet of Things. *IEEE Internet of Things Journal* 3, 3 (2016), 269–284. DOI:http://dx.doi.org/10.1109/JIOT.2015.2460333

SmartThings. 2015. Writing Your First SmartApp. (2015). http://docs.smartthings.com/en/latest/getting-started/first-smartapp.html

Steve Strauch, Vasilios Andrikopoulos, Uwe Breitenbuecher, Oliver Kopp, and Frank Leyrnann. 2012a. Non-functional data layer patterns for Cloud applications. In *Proceedings of the 4th IEEE International Conference on Cloud Computing Technology and Science (CloudCom'12)*. IEEE Computer Society Press.

Steve Strauch, Uwe Breitenbuecher, Oliver Kopp, Frank Leymann, and Tobias Unger. 2012b. Cloud Data Patterns for Confidentiality. *CLOSER* 12 (2012), 387–394.

The Open Group. 2014. *Security Design Patterns*. Technical Report. http://pubs.opengroup.org/onlinepubs/9299969899/toc.pdf

ThingWorx. 2016. Features. (2016). http://support.ptc.com/cs/help/thingworx_hc/thingworx_edge/index.jspx?id=c_twx_emsg_what_is_thingworx_concept&action=show

Isaura N. Bonilla Villarreal, Eduardo B. Fernandez, Maria M. Larrondo-Petrie, and Keiko Hashizume. 2013. A Pattern for Whitelisting Firewalls (WLF). *PLoP* 13 (2013).

Jeffrey Voas. 2016. Networks of 'Things'. *NIST Special Publication* 800 (2016), 183. http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-183.pdf

Tim Wellhausen and Andreas Fießer. 2012. How to write a pattern? A rough guide for first-time pattern authors. In *Proceedings of the 16th European Conference on Pattern Languages of Programs*. ACM, New York, NY.

Wind River. 2015. *Wind River Helix Device Cloud*. Technical Report. http://www.windriver.com/products/product-overviews/wr-device-cloud_overview.pdf

Qiang Xu, Rong Zheng, Walid Saad, and Zhu Han. 2016. Device fingerprinting in wireless networks: Challenges and opportunities. *IEEE Communications Surveys & Tutorials* 18, 1 (2016), 94–104.

Joseph Yoder and Jeffrey Barcalow. Architectural Patterns for Enabling Application Security. https://www.idi.ntnu.no/emner/tdt4237/2007/yoder.pdf

PLoP'17, OCTOBER 22-25, Vancouver, Canada. Copyright 2017 is held by the author(s). HILLSIDE 978-1-941652-06-0