Edge Device Security for Critical Cyber-Physical Systems

Stephan Cejka
Siemens AG Austria
Corporate Technology
Vienna, Austria
stephan.cejka@siemens.com

Felix Knorr
Siemens AG Austria
Corporate Technology
Vienna, Austria
felix.knorr@siemens.com

Florian Kintzler
Siemens AG Austria
Corporate Technology
Vienna, Austria
florian.kintzler@siemens.com

ABSTRACT

Building upon our previous research regarding privacy issues in the Smart Grid and Smart Building domain we explore these topics in the area of software updates in critical infrastructures. Since vital parts of the control in these cyber-physical systems are realized on edge devices or are dynamically moved from the backend to the edge devices, secure communication and computing in these components are of vital importance to ensure overall dependability of these systems especially when (software) changes are applied to the system. Therefore in this paper, we compare and analyze security features and approaches of three IoT frameworks that provide means to implement distributed control of critical infrastructure.

1 INTRODUCTION

There is a trend of moving computation tasks from the cloud to the edge, which has advantages such as reduced end-to-end latency, continuous service without permanent connection to the cloud, optimized usage of network bandwidth, and reduction of costs [15]. Cloud providers, such as Amazon Web Services (AWS) and Microsoft Azure, have reacted by introducing their own tool set for integrating and managing the edge, which may consist of hundreds of heterogeneous systems. The ERA-Net funded project LarGo! aims are developing and testing processes for the large scale rollout of software in the cyber-physical system (CPS) Smart Grid – in the power domain as well as in the user domain. The project investigates problems that arise from the interlocking of two networks – the power grid and the communication network. A first set of requirements for the rollout process was derived and the chosen approach to verify the resilience of the developed processes under research was described in [13].

Especially when modifying parts of the cyber-physical system – physically and on the software control layer – it is important to guarantee technical data protection, by using authentication, authorization, and encryption; nevertheless, other threats also arise. Security threats regarding confidentiality, integrity, availability, authentication, non-repudiation, and access control were surveyed in [10] showing that in Internet of Things (IoT) use cases, security issues need to be properly handled on all of the three levels – the device level, the network level, and the cloud level. The device level introduces further complications, as there are limitations to the means that are possible to be implemented on these devices due to their constrained battery capacity and computing power [17]. Furthermore, the physical access has to be taken into account, as for example, sensors attached to IoT devices in the CPS could be used to extract information or to trigger malicious activities in order to compromise the device [16].

A concept for ensuring privacy, also incorporated in the European Union's General Data Protection Regulation (GDPR), is 'privacy by design' that takes data privacy into account from the very beginning and throughout the whole product life cycle [11]. It imposes the implementation of data privacy and information security attributes before the products' launch and usage by embedding privacy directly into the design of the product. As in the case of security, privacy also needs to be taken care of on device level, during communication, while processing, and when storing data [14]. In previous work, we surveyed approaches to ensure privacy in the CPSs Smart Grid and Smart Building domain from both the (Austrian) legal as well as from the technical perspective [3, 7]. We argue that the enumerated solutions derived from the Smart Metering context – data minimization, data aggregation, data anonymization, data obfuscation, minimal and local data storage, as well as data sovereignty and control by the customer – can be mapped to other types of personal data arising in cyber-physical systems as well as in IoT use cases.

Ammar et al. compared eight main IoT frameworks, namely AWS IoT, ARM Bed, Azure IoT Suite, Brillo/Weave/AndroidThings, Calvin, HomeKit, Kura, and SmartThings, regarding their security features [1]. While their implementation into the frameworks differ, all of the evaluated frameworks support authentication, authorization and access control, as well as secure communication on a certain level. It is state-of-the-art to encrypt all communication to the outside of an IoT device, irrespective of whether it is to the cloud or local communication between the devices. As IoT devices could be stolen or moved, suitable physical protection is necessary. Therefore, the device itself, as well as every communication, needs to be protected against unauthorized access, interception and modification, for example, by using mechanical locks, authentication, and encryption [3, 7]. Furthermore, it is necessary to keep encryption algorithms exchangeable as the lifetime of an IoT device may exceed the security provided by today's encryption algorithms [12].

Security is especially important, once the CPS is modified via a software rollout. Therefore, this paper focuses on security features of three state-of-the-art IoT frameworks which, when used within critical cyber-physical systems like Smart Grids, can influence the resiliency of the overall system during a software update. When software is rolled out, there are estimations about the devices' behavior (e.g., communication frequency, communication partners, control values) before, during and after the update. There are two options for systems to react on violations of the expected behavior (anomalies):

- *a priori* by interventing message transmission by filtering out unexpected and thus not-permitted messages, or
- *a posteriori* by monitoring all messages or by receiving notifications on the unexpected behavior and reacting afterward.

We will show, how those options have been integrated into an application framework we introduced in earlier publications (Section 2). This workshop paper cannot provide a detailed survey on a high number of IoT frameworks, such as provided by *Ammar et al.* [1]. Thus, we selected the IoT solutions of the two currently leading cloud providers ([2]) Amazon Web Services and Microsoft Azure to describe how they handle those aspects (Section 3 and 4). In addition, we will briefly describe how updates (including security updates) are handled in these frameworks.

2 ISSN APPLICATION FRAMEWORK

In previous work we introduced an application framework for modular Java applications in the Smart Grid domain [5, 6, 9], running in the intelligent secondary substation node (iSSN) and communicating over a shared message bus. Until now about 20 of those applications have been developed throughout various project settings in which this framework is in use both within simulations as well as in field tests [4]. We introduced proxies that can be attached to a vendor-created module by an operator-controlled configuration. They can be attached to both the sender and the recipient module (Figure 1). Using these proxies, the normal message transmission procedure of the module is interrupted to run operator-defined code for executing additional message processing steps, including validity checks and modifications of the received or sent messages. Furthermore, the number of proxies used in the context of a module is not limited, thus a sequence of proxies can be configured. This can be done independently for incoming and for outgoing messages.

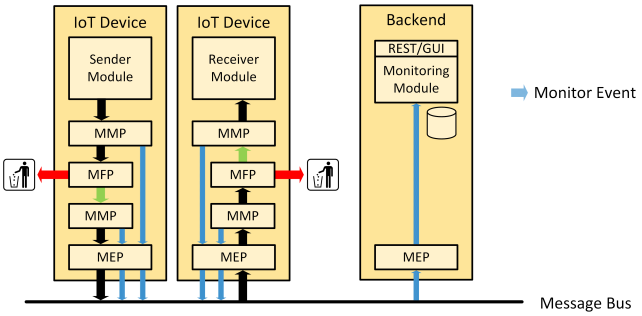

Figure 1: iSSN Security and Monitoring Proxies

2.1 Message Filter Proxies

Message Filter Proxies (MFP) [5] are used to restrict communication between modules. In a motivating example, a module is responsible to control the on-load-tap-changer (OLTC) transformer in the secondary substation. Applications can access current information on the tap position and can issue requests regarding tap position changes. However, the operation of the transformer is critical due to active interference with grid operation. Thus, it must be ensured that only authorized modules are able to issue those tap change requests and that the commands of these modules do not contradict each other. Therefore, modules are equipped with proxies that monitor and control incoming and outgoing messages, such that messages which are identified as prohibited can be filtered out. The simplest variant, introduced in [5], just checks whether the type of the message is among the allowed types and communication with the other party is permitted:

- On the sender side: check message type and receiver; if permitted send message, drop it otherwise.
- On the receiver side: check message type and sender; if permitted hand message over to module, drop it otherwise.

In summary, message filter proxies allow to automatically monitor and control the communication of a third-party module without requiring an in-depth inspection by the operator. Thus, the operator is not required to fully trust third-party modules. In the given example case, a simple message filter proxy was defined at the recipient side of the module to drop all incoming tap change request messages that were not issued by an allowed sender module. These message filter proxies can be further extended by including other information into their decision.

During a software update in a Smart Grid parts of the control components could be deactivated. In this case, it might be necessary to temporarily inhibit certain control flows that might lead to undesired states. By using MFPs the control flow from an application can be deactivated in parts, without the need to disable the complete application.

2.2 Message Monitoring Proxies

A less intrusive approach is the monitoring of a system's behavior via Message Monitoring Proxies (MMP). In [9] we introduced a simple logging proxy to log all passing messages; however, in use cases with a high number of modules (cf. [4]), this approach quickly becomes unmanageable. Therefore, we are currently working on a graphical monitoring tool that makes use of MMPs to audit all messages in real time or post-mortem by utilizing an optional NoSQL database. In cooperation with the MFP, it is suitable to configure a pair of MMPs with the MFP in-between (Figure 1). Thus, not-permitted message transfers (i.e., messages dropped by the MFP) could be identified and visualized. The operator can react on detected violations afterward; for example, by attaching a properly configured MFP.

A software update process needs to verify that the update led to the desired system behavior. Therefore the system must be able to detect anomalies, i.e., deviations from the expected behavior. Since this also involves the message flow between different system components, MMPs can be used to retrieve the needed information for their detection.

2.3 Message Encryption Proxies

Message Encryption Proxies (MEP) [5] are used for encrypted transmission of messages on the message bus[1]. In that way, they deny malicious third parties to read communication between modules. As all messages, including the events generated by the MMP, shall be properly encrypted, the MEP needs to be the last proxy defined

[1] For simplification, MEPs include the decryption proxy on the receiver side. Monitoring events originating at the receiver side, in fact, cannot use the same MEP as incoming messages.

on the sender side, and the first one on the receiver side. Furthermore, messages could obviously not be evaluated at any other proxy below the MEP. Necessarily, such an MEP also needs to be attached to the backend's monitoring module.

2.4 Management and update mechanisms

The operator manages and observes a high number if iSSNs remotely by use of a dashboard. This includes provisioning of software on these devices [5, 6, 9]. Thus, the system includes abilities for issuing installation, update, or uninstallation tasks for applications on the devices, as well as for monitoring the results of periodical health checks. Furthermore, the configuration of a module can be modified during the module's runtime, including to attach and detach proxies. Within the LarGo! project, we currently investigate the large scale and controlled deployment of applications to a high number of devices – for example, by utilizing a schedule or based on conditions in the CPS.

3 AWS IOT

Amazon Web Services (AWS) provides a suite of tools for the IoT[2]. In this paper, we only evaluate a selection of their tools, as relevant for security and resilience of edge devices in critical CPSs.

3.1 AWS IoT Device Defender

The AWS IoT Device Defender[3] audits device configurations and detects anomalous behavior of devices by utilizing configured security policies. Devices are continuously checked for deviations of those policies to check whether they work within defined borders. Once a behavior profile violation is detected, an alert is sent either to the AWS IoT Console, to Amazon Cloudwatch or via a Simple Notification Service (SNS) topic, which in turn can notify the user by push messages to a mobile devices' app, by AWS Lambda, Amazon SQS, Email or SMS. Observable behavior violations[4] can be distinguished into cloud-side metrics[5] and device-side metrics[6]. The operator then needs to react properly on these alerts. Thus, – similar to the monitoring proxies in the iSSN application framework – the AWS IoT Device Defender does not prevent the attached programs to communicate. However, the derived information can be used in the context of software updates in critical infrastructures to detect anomalies.

3.2 AWS IoT Greengrass

AWS IoT Greengrass[7] extends AWS cloud capabilities to local devices, allowing to collect and analyze data locally, and thus process it closer to the source of information. It allows to locally run AWS Lambda functions and to communicate with other devices securely, while not necessarily being always connected to the cloud or to the Internet at all[8]. Locally deployed Lambda functions are either on-demand functions, triggered by local events, messages from the cloud, or other sources, or long-lived functions that run indefinitely. They can also access local resources and are thus able to interact with peripherals.

Communication between the edge devices as well as to the AWS cloud is done using the MQTT protocol. For secure communication, authentication and authorization, local connections use MQTT over TLS, utilizing a public/private key pair and certificates. For connections to the cloud, web sockets and the AWS identity (access key id, secret access key pair) is used. Furthermore, every device has an attached policy to define what it is allowed to (Listing 1). For example, it can be restricted to which MQTT topic a device can publish to, or which resources it is permitted to access.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect"  : "Allow",
      "Action"  : [ "iot:Publish", "iot:Connect" ],
      "Resource": [ "*" ]
    }
  ]
}
```

Listing 1: Example AWS Policy for a device

For evaluation of the AWS IoT suite, especially for edge devices, we built up a smart home demo use case connecting several Raspberry Pis with attached sensors and actuators (Figure 2). On each received value, a local AWS Lambda function is executed that adds the sensed value to a local time-series database instance [8]. Thus, in this installation, only the configuration of the devices is located in the cloud, while data is stored and analyzed on the edge backend only – therefore privacy can be kept.

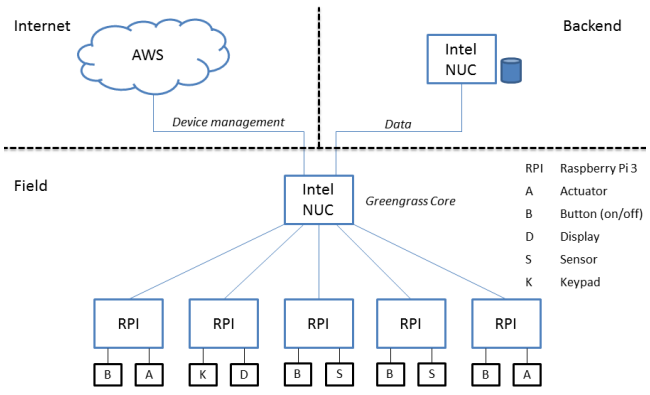

Figure 2: AWS IoT Greengrass Demo Architecture

[2] https://aws.amazon.com/iot
[3] https://aws.amazon.com/iot-device-defender/
[4] Behavior violations can only be combined by a conjunction (logical AND). Disjunctions (logical OR) are possible by defining additional rules.
[5] Supported cloud-side metrics are configurable thresholds for the number of authorization failures, of connection attempts, of disconnects, and of messages received; furthermore, the message size, and the source IP.
[6] Supported device-side metrics are configurable thresholds for the number of bytes in and out, and the number of packets in and out; furthermore the destination IPs, listening TCP/UDP ports or their count, and established TCP connections count.
[7] https://aws.amazon.com/greengrass/
[8] One device, namely the Greengrass Core, necessarily needs to reconnect to the cloud periodically, e.g., to renew certificates.

3.3 Management and update mechanisms

The AWS IoT Device Management[9] tool is used to manage device groups that may contain even hundreds of cores. Since each device has a device shadow in the cloud, the device can be managed even if currently not connected. Even though the AWS IoT Device Defender does not prevent attached programs or devices to communicate, it is thus easy to react on violation notifications by rolling out patches to the devices or by restoring it to a "good" state.

Though mainly suited for the "normal" Lambda functions deployed in the AWS cloud, local Lambda functions can be retrieved from the app-store-like AWS Serverless Application Repository[10]. They can only be executed on the Greengrass core device; however, as the only device in a Greengrass group connected to AWS IoT is the Greengrass core, it is not possible to roll-out any software to other devices in the group, anyways.

4 MICROSOFT AZURE IOT

Microsoft, similarly, provides cloud-based tools for the IoT that enables bi-directional communication between devices and the cloud, named Azure IoT[11]. Furthermore, Azure extends the cloud capabilities by allowing data collected and manipulated on the edge using Azure IoT Edge[12], comparable to AWS Greengrass. It uses locally running Azure Functions, analogous to Greengrass' local Lambda functions; and the IoT Hub Device Provisioning Service, analogous to AWS' Device Management. To secure the traffic between the cloud and the edge device, or between the edge device and its children devices TLS is used; the device identity management unit ensures that only authorized devices are able to connect to the cloud. Besides MQTT, AMQP, and HTTP, other communication protocols are supported via so-called protocol gateways, which securely transport the aggregated data to the cloud. Behind those gateways there are usually constrained devices, hence the communication in-between is not secured due to their limited capacity.

Azure also provides tools to detect anomalies and undesired behavior, but in contrast to AWS' Device Defender, at a finer grained level. Thus, it is possible to define rules that not only describe the correct behavior of the IoT Edge modules themselves, but also the correct behavior of devices connected to those modules. For example, if the edge device detects an anomaly via a rule, defined centrally in the cloud and deployed to the device, a notification is sent to the cloud which causes further actions. On the Azure cloud level, it is possible to react on alerts of one or more devices and to create jobs that execute methods taking countermeasures on those devices. In addition, machine learning models can be trained to detect anomalies by using the metrics from the devices. These means can be used to implement anomaly detection to ensure that a software rollout in the system leads to the desired state.

Furthermore, Azure provides means to partition the set of devices the system consists of into Trust Zones[13], to build up smaller segments, for example, for analyses at a fine-grained level. For each of these zones, it is possible to define separate authentication and authorization requirements as well as zone-specific threats in order to isolate the potential damage.

5 CONCLUSION AND OUTLOOK

It is important to secure a CPS like the Smart Grid especially when parts of the CPS are modified during a software update. To ensure security, the system must be able to react on deviations of the devices' expected behavior during the software rollout. Therefore either the communication between devices in the CPS can be filtered or, less intrusive, be monitored for anomaly detection. The approaches of three IoT frameworks to ensure security on the edge in these systems were compared as summarized in Table 1.

Both AWS and Azure provide a full framework including provisioning, secure communication, and possibilities to detect anomaly behavior on the device level, the network level, and the cloud level. While AWS provides simple metrics detection only, Azure also allows to quickly react to this behavior by taking countermeasures and offers more sophisticated methods at different levels. However, both cloud service providers only provide means of security if their solution is applied to all components in the system; thus, it is not possible to manage heterogeneous systems with AWS or Azure.

[9] https://aws.amazon.com/iot-device-management/
[10] https://aws.amazon.com/serverless/serverlessrepo
[11] https://azure.microsoft.com/overview/iot/
[12] https://azure.microsoft.com/services/iot-edge/
[13] https://buildazure.com/2018/12/20/iot-security-architecture-trust-zones-and-boundaries/

	iSSN	**AWS IoT**	**Azure IoT**
Provisoning / Rollout	via web-based provisioning backend	via cloud	via cloud
Rollout Target	all devices	to Greengrass core only	all devices
Encryption	as defined by MEP	asymmetric	asymmetric
Violation Reporting	to monitoring backend, or as defined by MMP	to cloud	to cloud
Extendability	very flexible using proxies	rigid	flexible
Violation Handling	filtering (MFP), reporting (MMP)	reporting	user defined
Function Exec Environment	native	sandbox	sandbox

Table 1: Comparison Summary

ACKNOWLEDGMENTS

The presented work is conducted in the LarGo! project, funded by the joint programming initiative ERA-Net Smart Grids Plus with support from the European Union's Horizon 2020 research and innovation programme. On national level, the work was funded and supported by the Austrian Climate and Energy Fund (KLIEN, ref. 857570).

REFERENCES

[1] Mahmoud Ammar, Giovanni Russello, and Bruno Crispo. 2018. Internet of Things: A survey on the security of IoT frameworks. *Journal of Information Security and Applications* 38 (2018), 8–27.

[2] Alexey V. Bataev, Dmitriy G. Rodionov, and Darya A. Andreyeva. 2018. Analysis of world trends in the field of cloud technology. In *2018 International Conference on Information Networking (ICOIN)*. 594–598.

[3] Stephan Cejka. 2017. Vorschläge für Datenschutz und Privatsphäre bei Smart Metern und deren Umsetzung im österreichischen Recht. In *Jusletter IT*. in german.

[4] Stephan Cejka, Konrad Diwold, Albin Frischenschlager, and Philipp Lehninger. 2018. Integrating Smart Building Energy Data into Smart Grid Applications in the Intelligent Secondary Substations. *Journal of Electronic Science and Technology* 16, 4 (2018), 291–303.

[5] Stephan Cejka, Albin Frischenschlager, Mario Faschang, Mark Stefan, and Konrad Diwold. 2018. Operation of Modular Smart Grid Applications Interacting through a Distributed Middleware. *Open Journal of Big Data* 4, 1 (2018), 14–29.

[6] Stephan Cejka, Alexander Hanzlik, and Andreas Plank. 2016. A framework for communication and provisioning in an intelligent secondary substation. In *IEEE International Conference on Emerging Technologies and Factory Automation*.

[7] Stephan Cejka, Felix Knorr, and Florian Kintzler. 2019. Privacy Issues in Smart Buildings by Examples in Smart Metering. In *25th International Conference on Electricity Distribution (CIRED)*. submitted.

[8] Stephan Cejka, Ralf Mosshammer, and Alfred Einfalt. 2015. Java embedded storage for time series and meta data in Smart Grids. In *IEEE International Conference on Smart Grid Communications*. 434–439.

[9] Mario Faschang, Stephan Cejka, Mark Stefan, Albin Frischenschlager, Alfred Einfalt, Konrad Diwold, Filip Pröstl Andrén, Thomas Strasser, and Friederich Kupzog. 2017. Provisioning, deployment, and operation of smart grid applications on substation level. *Computer Science - Research and Development* 32, 1 (2017), 117–130.

[10] Paul Fremantle and Philip Scott. 2017. A survey of secure middleware for the Internet of Things. *PeerJ Computer Science* 3 (May 2017), e114.

[11] Seda Gürses, Carmela Troncoso, and Claudia Diaz. 2011. Engineering privacy by design. *Computers, Privacy & Data Protection* 14, 3 (2011).

[12] Himanshu Khurana, Mark Hadley, Ning Lu, and Deborah A. Frincke. 2010. Smart-Grid Security Issues. *IEEE Security and Privacy* 8, 1 (Jan. 2010), 81–85.

[13] Florian Kintzler, Tobias Gawron-Deutsch, Stephan Cejka, Judith Schulte, Mathias Uslar, Eric Veith, Ewa Piatkowska, Paul Smith, Friederich Kupzog, Henrik Sandberg, Michelle Chong, David Umsonst, and Marco Mittelsdorf. 2018. Large Scale Rollout of Smart Grid Services. In *2018 Global Internet of Things Summit (GIoTS) (GIoTS'18)*.

[14] J. Sathish Kumar and Dhiren R. Patel. 2014. A Survey on Internet of Things: Security and Privacy Issues. *International Journal of Computer Applications* 90, 11 (March 2014), 20–26.

[15] Shadi A. Noghabi, John Kolb, Peter Bodik, and Eduardo Cuervo. 2018. Steel: Simplified Development and Deployment of Edge-Cloud Applications. In *10th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 18)*.

[16] Amit Kumar Sikder, Giuseppe Petracca, Hidayet Aksu, Trent Jaeger, and A. Selcuk Uluagac. 2018. A Survey on Sensor-based Threats to Internet-of-Things (IoT) Devices and Applications. *CoRR* abs/1802.02041 (2018). arXiv:1802.02041

[17] Wade Trappe, Richard Howard, and Robert S. Moore. 2015. Low-Energy Security: Limits and Opportunities in the Internet of Things. *IEEE Security & Privacy* 13 (01 2015), 14–21.