Security Concepts in a Distributed Middleware for Smart Grid Applications

Stephan Cejka · Albin Frischenschlager · Mario Faschang · Mark Stefan

Abstract – Novel software applications are developed and used in order to take full advantage of Smart Grid and Smart City infrastructures. In our concrete Smart City field trial, a distributed middleware is used to connect such interacting Smart Grid applications. This work presents a threat analysis for this middleware-based communication containing six potential attack patterns. As countermeasure against the potential attacks, we present the security concept for the interacting Smart Grid applications consisting of the middleware's encryption layer and trusted applications.

1. Introduction

The evolution from passively operated to intelligent secondary substations (iSSN) allows for novel functions (e.g., voltage and (re-)active power control, distributed generation optimization, market interaction) by having increased computational power and newly attained communication. These functions are realized by distributed software components – so called smart grid applications – within the substations. Such applications are interlinked and interacting through a common middleware. In previous work, we presented a flexible and modular software ecosystem for iSSNs including such a middleware, which acts as communication infrastructure that allows for the operation of distributed smart grid applications (cf., [1-4]). This work focuses on the security considerations of such a middleware to protect both local and remote interaction of the smart grid applications.

2. Gridlink

We introduced the Gridlink – based on vert.x – as a middleware solution for the iSSN, using a distributed event bus based on an asynchronous communication model. Gridlink-based systems are built of several modules, each of them communicating with each other by exchanging messages. Modules are able to join or leave at any time without influencing other modules' execution. As there is no single point of failure, a fail of any module neither prevents other modules to be further executed nor to communicate with remaining modules. A cutback of the overall function of the app in such cases is obvious, but can be limited by using redundant modules and reasonable timeouts to react on failed transmissions.
Messages are either sent to a designated module role address or published to a topic address reaching all registered modules, by default being marshaled into a JSON representation for transmission. Dedicated proxies are executed before the message is transmitted, allowing to execute additional message processing steps, including its modification, e.g., for encryption. Proxies and the demarshal process on the recipient's side works likewise before the message is handed over to the module's service handler and processed. Furthermore, the Gridlink contains a service registry, where all currently attached running modules are listed to all other modules. Further details, including an in-depth description of Gridlink proxies are available in [2, 3].

3. Threat Analysis of Gridlink Message Exchange

We identified several potential security issues regarding the message exchange procedure in Gridlink. The intended scenario for this document is pictured in Fig. 1. There are one source module as sender of messages and one sink module that should receive them. In the given example, two messages *(msgX, msgY)* are sent by a module with *address A* to a module that holds *role B*.

Fig. 1: Intended Scenario

Modules register themselves to roles/topics to receive messages sent/published to the respective address. It is possible to get registered to a role, while another module is already registered to it. The intended behavior of vert.x in that case is to deliver a message to the respective modules in round-robin fashion.

3.1 Role Claim Attack (a)

A malicious module (*B'*) also registers for *role B*. The result is shown in the figure; half of the messages are not received at the intended receiver. Data can be read by the malicious module that should not be visible to this module.

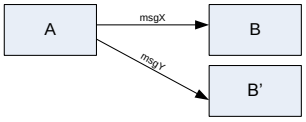

Fig. 2: Role Claim Attack

Stephan Cejka · Albin Frischenschlager
Siemens AG, Corporate Technology
Siemensstraße 90, 1210 Vienna, Austria
[stephan.cejka, albin.frischenschlager]@siemens.com

Mario Faschang · Mark Stefan
AIT Austrian Institute of Technology,
Donau-City-Straße 1, 1220 Vienna, Austria
[mario.faschang, mark.stefan]@ait.ac.at

3.2 Topic Claim Attack (b)

A malicious module (*B'*) also registers for *topic B.* While in this case, in contrast to attack *(a)*, all messages are received at the intended sender, the malicious module also receives all messages. Data can be read by the malicious module that should not be available for this module.

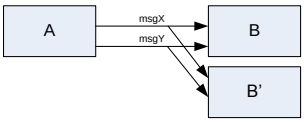

Fig. 3: Topic Claim Attack

3.3 Spoofing Attack (c)

The malicious module (*A'*) is located on the sender side. Messages that are sent from the malicious module look like they were from the intended module. Module B will not be able to tell messages from *module A* and the malicious module apart.

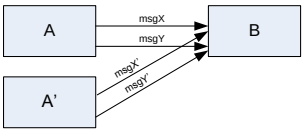

Fig. 4: Spoofing Attack

3.4 Denial of Service Attack (d)

A module may be forced to a denial of service if a malicious module sends messages that are complicated to handle or the number of messages exceeds the number of messages able to handle.

3.5 Event Bus Blocking Attack (e)

vert.x – and thus Gridlink – uses an event bus thread shared between all modules running on one node. If a malicious module gets stuck or behaves unexpectedly, all other modules on the node are affected or even blocked.

3.6 Invisible Module Attack / Registry Attack (f)

A Gridlink module registers itself in the registry on startup. If any module is a vert.x "verticle", but not a Gridlink module, communication over vert.x happens as intended but the module remains invisible to other Gridlink modules as no register in the registry takes place.

4. Introduction of Gridlink Security Measures

The use of **Gridlink proxies** for encrypting messages before sending and decrypting them again at the receiver denies that malicious module can read the communication. These proxies can either use symmetric or asymmetric cryptography. By use of such proxies only features that are already included in the Gridlink are utilized; cryptography proxies are thus termed the Gridlink Security Layer establishing end-to-end encryption (Fig. 5).

Mentioned attacks – however – can only partly be solved by it:

- Malicious modules of attack scenarios *(a)* and *(b)* will still receive messages. However, they will not be able to read these encrypted messages.
- An open problem is that the second message will still not be received at *module B* in attack *(a)*. The only possibility is to resent the message, introducing other issues regarding the detection of messages that were not received on the sender's side.
- Messages that are sent by the malicious module in attack *(c)* cannot be understood by the receiver due to their incorrect or not existing encryption.

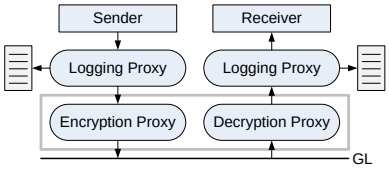

Fig. 5: Example for encryption of payloads by use of proxies (Gridlink Security Layer) [2]

The Denial of Service Attack *(d)* cannot fully be sorted out by encrypting the messages as the recipient has effort for decryption. Malicious messages that would require high effort at the recipients' side can however be sorted out.

The introduction of **Trusted Gridlink Applications** allows for module certification. An execution of uncertified and thus untrusted modules beside trusted ones on the same physical node may be prohibited. This partly solves the Event Bus Blocking Attack *(e)*, as the influence of untrusted to trusted modules' execution remains limited.

The introduced security measures are state-of-the-art and will solve most of the presented typical and severe attack scenarios. Currently work is ongoing in order to sort out system-specific attacks like the Module Registry Attack *(f)*.

Acknowledgements

The presented work is conducted in the "iNIS" project, funded and supported by the Austrian Ministry for Transport, Innovation and Technology (BMVIT) and the Austrian Research Agency (FFG).

References

1. Faschang M, Stefan M, Kupzog F, Einfalt A, Cejka S (2016) "iSSN Application Frame" – a flexible and performant framework hosting smart grid applications. CIRED Workshop 2016, doi:10.1049/cp.2016.0694
2. Faschang M, Cejka S, Stefan M, Frischenschlager A, Einfalt A, Diwold K, Pröstl Andren F, Strasser T, Kupzog F (2016) Provisioning, deployment and operation of smart grid applications on substation level. Computer Science – Research and Development (CSRD), doi: 10.1007/s00450-016-0311-x
3. Cejka S, Hanzlik A, Plank A (2016) A framework for communication and provisioning in an intelligent secondary substation. 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA), doi: 10.1109/ETFA.2016.7733591
4. Einfalt A, Cejka S, Diwold K, Frischenschlager A, Faschang M, Stefan M, Kupzog F (2017) Interaction of Smart Grid Applications supporting Plug&Automate for intelligent secondary substations. 24th International Conference on Electricity Distribution (CIRED), to appear