OCPP Protocol: Security Threats and Challenges

Cristina Alcaraz[1,2] Member, IEEE, Javier Lopez[1] Senior Member, IEEE,
and Stephen Wolthusen[2,3] Senior Member, IEEE

[1]Computer Science Department, University of Málaga, Spain

[2]Information Security Group, Department of Mathematics, Royal Holloway

University of London. Egham TW20 0EX, United Kingdom

[3]Norwegian Information Security Laboratory, Gjøvik University College, Norway

{alcaraz,jlm}@lcc.uma.es

stephen.wolthusen@rhul.ac.uk

August 23, 2017

Abstract

One benefit postulated for the adoption of Electric Vehicles (EVs) is their ability to act as stabilizing entities in smart grids through bi-directional charging, allowing local or global smoothing of peaks and imbalances. This benefit, however, hinges indirectly on the reliability and security of the power flows thus achieved. Therefore this paper studies key security properties of the already-deployed *Open Charge Point Protocol* (OCPP) specifying communication between charging points and energy management systems. It is argued that possible subversion or malicious endpoints in the protocol can also lead to destabilization of power networks. Whilst reviewing these aspects, we focus, from a theoretical and practical standpoint, on attacks that interfere with resource reservation originating with the EV, which may also be initiated by a man in the middle, energy theft or fraud. Such attacks may even be replicated widely, resulting in over- or under-shooting of power network provisioning, or the (total/partial) disintegration of the integrity and stability of power networks.

Keywords: Smart Grid, Charging Infrastructure, Cyber-Physical Systems, Cyber Security, OCPP

1 Introduction

The introduction of electric vehicles (EVs) links two critical infrastructure sectors, namely transportation and electric power networks, which thus far have only been characterized through indirect interdependencies. Apart from helping to reduce emissions overall, contributing to global objectives and reducing local road traffic emissions, the introduction of *bi-directional charging* is also anticipated to play an important role in realizing local and global *control and stabilization of smart grids*. In this scenario, open standards and a shared infrastructure for EV charging are key in finding a satisfactory density of charging stations that also allow EVs to seamlessly operate across service areas or even countries. However, this procedure requires the orchestration of a number of services, including: Metering and payment for energy, communication between the EV battery management system and the *charge point* (CP), communication between the CP and a central management system (CS), and communication between the CS and energy suppliers and the power grid.

These infrastructures, composed of mobile devices, autonomous entities and heterogeneous cyber-physical systems, require the standardization of protocols and the implementation of two primary interfaces, one for electricity and another for control related to status, authorization, metering, and billing. Standards for the individual aspects (e.g., ISO/IEC-15118, ISO/IEC-61850) of this chain of interactions exist, but the integration of these interactions in a common framework has recently become the focus of much interest. Among these interactions, the *Open Charge Point Protocol* (OCPP) offers a way to coordinate communication and ultimately power flows between CPs and a CS [1], keeping direct interactions with EVs and the grid in an *Internet of Energy*. Concretely, the OCPP protocol is mainly concerned with reservations and management of charging processes with restricted security considerations, principally limited to ensuring that charging is performed only when authorized by a billing system. In addition, the specification assumes that the components involved are in themselves trustworthy and cannot be manipulated or compromised, *together with a tacit assumption that OCPP protocol states are always reflected in the power connection*. Beyond simple fraud involving payment systems, we argue that this can permit attacks on the stability of the critical infrastructure and individual components by attempting to alter the charging rate beyond agreed parameters, altering control status or driving energy components outside accepted parameters.

To the best of the authors' knowledge, this is the first study of OCPP security, whereas prior work has concentrated on addressing interoperability aspects (e.g., IEEE 802.15.4 and OCPP over SOAP (simple object access protocol) [2], or IEC 61850 and OCPP [3]) and new charging services [4]. Given this, we believe that this work can become fundamental to show the security problems together with the new research challenges. To do this, the remainder of this paper is structured as follows. We briefly review the currently deployed versions of OCPP and on-going extension efforts in Section 2, and then we identify a set of threats in Section 3 before discussing our findings and challenges.

2 OCPP: Versions and Background

OCPP, currently a de-facto open standard [1], aims to provide a universal open communication standard between CPs (generally located in public places such as petrol stations, supermarkets, and car parks) and vendor CSs [1]. So far, there have been several OCPP versions; 1.2, 1.5, 1.6 (current), 2.0 (proposal) as well as some proprietary extensions. Differences between versions 1.2 and 1.5/1.6 are quite significant as the v1.5 and 1.6 offer new functions and extensions such as a local authorization list (LAL) synchronized with the CS and a cache to streamline the response times with the EV users. Both memory components allow CPs not only to store authorized users with ID tags (containing user's identification and retained in a radio-frequency identification tag or a similar tag), but also to operate and authorize entities in a stand-alone fashion when the communications with the CS are temporarily lost (off-line mode). In these exceptional cases, any transaction must be queued to wait to be authorized by the back-end system.

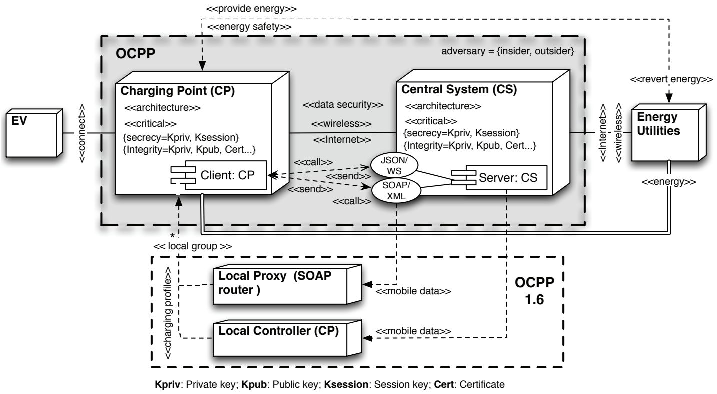

Figure 1: Deployment diagram of the OCPP protocol

OCPP-1.6 also includes some functional improvements over v1.5, mainly related to smart charging to constrain the amount of power delivered during charge transactions. This capacity provides a better local monitoring and power control by automatically restricting charging according to the actual demand over supply and the contextual characteristics of each individual or associated CP (see Figure 1). Namely, OCPP-1.6 allows CPs or intermediary CPs (serving as local proxies or controllers between the CS and a subset of CPs; useful for limited and dynamic scenarios such as underground car parks) to furnish energy, taking into account the current power rate and the availability of the entire power grid. On the other hand, v1.6 operations also adopt diverse communication infrastructures whose connections can be achieved using SOAP/XML (extensible mark up language) or JSON (javascript object notation) over WebSockets (WS).

The migration from v1.2 to v1.6 has also involved the adaptation and updating of a set of commands[1], which have been grouped into three stages: **Start-up and initial configuration** which includes the boot notification initiated by the CP and upon each re-boot caused by a reset. The configuration process is, however, led by the CS, not necessarily on each re-boot, to synchronize and update the capacities of the CPs, and specify the health monitoring criteria. **Transactions and control** carry out the processes of authorization of an ID tag, the start/stop transactions, the reservation or cancellation of energy, the reset of a CP, the availability and unlocking of an electrical outlet or a CP, the heartbeats to the CS, the meter values, the LAL version update, as well as clearing the cache, the detection of non-supported functions, and the (local or remote) supervision of power filtered by CPs. Lastly, **notification and maintenance** provide a set of instructions related to remote diagnosis, maintenance and configuration

[1] The changes made for the migration are marked with $*$ in this paper.

of (hardware and software) OCPP elements.

From these three stages, we observe that OCPP relies on a rudimentary security provided by lower-level (e.g., TLS (transport layer security) or WS-Security mechanisms) without specifying the commissioning of nodes and the bootstrapping for agreement and session establishment. It only suggests applying TLS/SSL for critical data without exploring other security measures to manage the communications. OCPP-2.0, to the contrary, aims at setting up a liaison with OASIS and IEC so as to improve, extend and standardize some functions of v1.6 such as demand response management, pricing management, and enhanced monitoring and control [5]. As this version is, for the moment, an initial brainstorming approach to be released in the coming years, our research will be principally focused on OPCC-1.6, thus showing the existing vulnerabilities and contributing to a better security for subsequent versions.

3 Threat Scenarios

We describe a set of threat scenarios related to the logical functionality of the OCPP protocol, basing our studies on the IETF framework [6, 7] as well as some recent related work [8, 9]. To formalize the threat scenario, we consider the UMLsec notation (with stereotypes, tagged values and constraints) defined by Jürjens in [10]. UMLsec is an extension of the UML general-purpose modelling language adapted to specify security requirements and visualize the design of security systems. One example of its utility is also depicted in Figure 1, which summarizes all the notations mentioned and represents the exiting relationships between OCPP components and their communication links.

Let A be the adversary targeting CP/CS (hardware and software) assets and their communication links, represented through the stereotypes $\ll$*architecture*$\gg$ (a), and $\ll$*Internet*$\gg$ and/or $\ll$*wireless*$\gg$ (c), respectively. The threats associated with a y c are included in threats$_A(a,c)$ containing: (1) **Disclosure**, which corresponds to illicit *reading* and/or copying of information; (2) **distortion**, any (fake) data *insertion*, spoofing or *modification* action (data, processes or configurations); and (3) **disruption** that comprises the *deleting* or *dropping* of messages, processes or actions. Variants of this last threat can be those related to denial of service (DoS) such as grayhole (selectively forwarding packets to the next hop) or black holes (dropping all messages), reduction of functionality.

The threats$_A(c)$ can be addressed if the communication channels are intercepted, and the *secrecy* $= \{KPriv$ (private key), $K_{session}$ (session key)$\}$, such that $\{KPriv, Kpub$ (public key), $K_{session}\}$ correspond to the security credentials of an OCPP user/object, is known in advance by attackers. The effect of these actions may damage not only the $\ll$*data security*$\gg$ stored in the OCPP domains and their communications but also the $\ll$*energy safety*$\gg$ for its provision. Attackers might take advantage of the interception of a and c to enable other subsequent attacks such as $\{$disrupt, energy theft, overloading$\} \in$ threats$_A(e)$, where e denotes the stereotype $\ll$*energy*$\gg$. By **disruption**, we refer to denial of power resources and services; by **energy theft**, any intentional action $\in$ threats$_A(c)$ that aims to illicitly obtain power, commit fraud or demand mis-forecasting to interfere with dynamic pricing; and by **overload** $-$ to the variation of power levels at particular points in the system, e.g., substations. In these cases, attackers might inject fake OCPP transactions to cause serious local effects and/or reduce network stability.

For the protocol analysis, we also consider the following notations: With A the attacker of type insider (i.e., related to charging infrastructure operators) or outsider (end users and intermediaries), $\_ :: \_$ to represent the concatenation of UMLSec expressions, $\{\_\}_{KPub}$ for encryption, $Dec_{KPriv}(\_)$ decryption, $sign_{KPriv}(\_)$ digital signature, $Ext_{KPub}(\_)$ to extract the signature such that $\{KPub, KPriv\} \in Keys$, and arg the argument obtained in each iteration. Specifically, arg contains information of both the node that analyzes the message and the position of the parameter within the list of arguments defined in each method. Note that for reasons of space, we primarily focus our study on those obligatory parameters of the OCPP specification [1], but we name some optional OCPP parameters to describe the most relevant-security scenarios.

3.1 Stage 1: Start-Up and Configuration

Beyond physical attacks in a and assuming TLS communication, man-in-the-middle (MitM) attacks [11] may arise in OCPP contexts. To characterize this situation, in the interaction covered by Algorithm 1, a CP requests a TLS connection to the CS, transmitting at least its public key ($KPub_{CP}$) and a nonce value (N_i) to avoid replay attacks. If an adversary A with initial knowledge $K_A^0 = \{KPub_A, KPriv_A, KPub_{CA}\}$ is able to intercept the communication and interrupt the request (step 1 - Algorithm 1), it may proceed to carry out the same operation as the CP but with the CS. In other words, the adversary requests connection as a client and sends its public key ($KPub_A$), the original nonce N_i and the ID_CP signed by A to the CS. In this way, the central system can authenticate the source, and if the authentication is valid, the CS can then compute $K_{session}$ (step 2-4 - Algorithm 1). Once this key has been computed, the CS transmits it, encrypted with $KPub_A$, to the supposedly "legitimate client" A together with its certificate ($Cert_{CS}$) (step 5 - Algorithm 1). The attacker then re-sends the $K_{session}$ to the CP with information related to the CS (the ID_CS and its $Cert_{CS}$), so as to convince the CP of the authenticity of the message and its origin (step 7-8 - Algorithm 1). At this point, $K_{session}$ is compromised where secrecy and integrity of the stereotype c are infringed (step 6 - Algorithm 1), and in such a way that threats$_A(c) \subseteq$ {disclosure, distort, disrupt} materialize.

As $K_{session}$ forms part of the knowledge of the adversary in the state $K_A^n = K_A^0 \cup \{KPub_{CP}, KPub_{CS}, ID\_CS, ID\_CP, Cert_{CS}, K_{session}\}$, any control request or transaction may entail an increase of knowledge for learning from new vulnerabilities. For example, in the boot phase, the CP has to send a boot notification to the CS through the command *BootNotification.req(CPModel, CPVendor)*. Here, an attacker A with the secret key in hand may eavesdrop and extract the serial number of the CP (CPModel) and the vendor (CPVendor). This could allow them to obtain more information about the meter model, firmware version or the single in-line memory (SIM) (i.e., threats$_A(a) \subseteq$ $disclosure$) as specified in steps 1-2 of Algorithm 2 such that $K_A^{n+1} = K_A^n \cup$ {CPModel, CPVendor}. Moreover, in response to the boot notification (steps 3-5 of Algorithm 2 through *BootNotification.conf(currentTime, interval, status)* where *interval* defines the heartbeat periods) for the configuration of the CP, a MitM might: (i) Vary the clock of the CP to de-synchronize or delay layered communication; (ii) change the connection status with the CS to be rejected; or (iii) send short heartbeat intervals to produce denial of service

(DoS) (steps 6-8 - Algorithm 2); resulting in threats$_A(a, c)$.

Algorithm 1 Vulnerability of TLS in OCPP (MitM)

```
1: CP → A: Initializes the communication with the "CS".
   $\textbf{\textit{CommTLS.init}}(N_i, KPub_{CP}, sign_{KPriv_{CP}}(ID\_CP\text{::}KPub_{CP}))$
2: A: Computes the parameters and updates its knowledge state $K_A^i$.
   $N' := arg_{A,1}$; $K'_{CP} := arg_{A,2}$; $ID'_{CP} := Ext_{K'_{CP}}(arg_{A,3})$; $K_A^1 := K_A^0 \cup \{ID'_{CP}, K'_{CP}\}$
3: A → CS: Initializes the communication with the CS.
   $\textbf{\textit{CommTLS.init}}(N_i, KPub_A, sign_{KPriv_A}(ID\_CP\text{::}KPub_A))$
4: CS: Computes the parameters and generates the session key.
   $N' := arg_{CS,1}$; $K'_A := arg_{CS,2}$; $Ext'_{K_A}(arg_{CS,3}) = ID\_CP$
5: CS → A: Responds to the "CP" for the connection in TLS.
   $\textbf{\textit{CommTLS.resp}}(\{sing_{KPriv_{CS}}(K_{session}\text{::}N')\}_{K'_A}, sign_{KPriv_{CA}}(ID\_CS\text{::}KPub_{CS}))$
6: A: Computes the parameters and updates its knowledge state.
   $K'_{CS} := Ext_{KPub_{CA}}(arg_{A,2})$; $ID'_{CS} := Ext_{KPub_{CA}}(arg_{A,2})$
   $K'_{session} := Dec_{KPub_A}(Ext_{K'_{CS}}(arg_{A,1}))$; $K_A^2 := K_A^1 \cup \{ID'_{CS}, K'_{CS}, K'_{session}\}$
7: A → CP: Responds to the CP for the connection in TLS.
   $\textbf{\textit{CommTLS.resp}}(\{sing_{KPriv_{CS}}(K_{session}\text{::}N')\}_{K'_{CP}}, sign_{KPriv_{CA}}(ID\_CS\text{::}KPub_{CS}))$
8: CP: Computes the parameters, authenticates the source, validates the freshness of the message and obtains the session key.
   $K'_{CS} := Ext_{KPub_{CA}}(arg_{CP,2})$; $ID'_{CS} := Ext_{KPub_{CA}}(arg_{CP,2})$
   IF $((ID'_{CS} = ID\_CS)$ AND $(Dec_{KPub_{CP}}(Ext_{K'_{CS}}(arg_{CP,1}))) = N_i))$ THEN {
   $K'_{session} := Dec_{KPub_A}(Ext_{K'_{CS}}(arg_{CP,1}))$}
```

MitM may also intercept the channel to lead **on-path** attacks of the kind threats$_A(c)$ $\subseteq$ {distort, disrupt} such as: exhaustion of the channel, delays, replays by not being guaranteed the freshness of the OCPP messages, dropping or impersonation (e.g., sinkholes to attract traffic towards a malicious node, or wormholes to entail a sinkhole with several nodes in conjunction [6, 7]). For example, when CPs wish to connect to a CS through a WS channel, they have to first negotiate the type of WS handshake through an insecure HTTP request. If the MitM has not been mitigated and has set up connectivity with a CP, the end-point of such requests may be spoofed to direct traffic to itself (sinkhole), or to one/several hosts in order to create a distributed threat scenario (wormhole). Note that it is also possible that the CP may be a malicious node itself, which might request multiple WS upgrades with non-supported information (e.g., with a non-existent OCPP version) so as to leave operational services unavailable.

Intermediary nodes may also perturb the configuration of a CP by injecting fake configuration parameters or varying the format or the size of its command, *ChangeConfiguration.req(key, value)* such that *key* corresponds to the configuration setting (location, time, samples, etc.). These configurations might, for example, force the CP to produce short retries for resetting, modifying or injecting the sampled data included within the instruction *MeterValues.req(connectorId, metervalue)*, or modifying the transmission time of the meter values. As for **impersonation** attacks, these intermediaries might send an LAL with false ID tags with unlimited expiration to perform transactions with status equal to accepted through *SendLocalList.req(listVersion, updateType, [LAL)*. This command transfers, through the variable *updateType*, a full or differential of the LAL version with those ID tags that must be authorized by the CP. So a full transference of a periodic and large LAL might reduce functionalities in a and cause delays in c. Moreover, the previous OCPP releases added the option of applying a hash function to transfer the list in a se-

Algorithm 2 Vulnerability in the OCPP boot notification

```
1: CP → A: Establishes synchronization with the supposed legitimate "CS".
   $\textbf{\textit{BootNotification.req}}(\{CPModel::CPVendor\}_{K_{session}})$
2: A: Eavesdrop on communication using $K_{session}$ and updates its knowledge $K_A^i$.
   $CPModel' := Dec_{K_{session}}(arg_{A,1}); CPVendor' := Dec_{K_{session}}(arg_{A,1})$
   $K_A^{n+1} := K_A^n \cup \{CPModel', CPVendor'\}$
3: A → CS: Establishes synchronization with the CS.
   $\textbf{\textit{BootNotification.req}}(\{CPModel::CPVendor\}_{K_{session}})$
4: CS: Computes the parameters and prepares the configuration of the CP.
   $CPModel' := Dec_{K_{session}}(arg_{CS,1}); CPVendor' := Dec_{K_{session}}(arg_{CS,1})$
5: CS → A: Provides the supposed legitimate "CP" with synchronization inf.
   $status$: =‘Accepted’
   $\textbf{\textit{BootNotification.conf}}(\{currentTime::status::interval\}_{K_{session}})$
6: A: Computes the parameters and produces one of the following attacks:
   $(currentTime' := clock_A)$ OR $(status' :=$ ‘Rejected’$)$ OR $(interval' := interval_A)$
7: A → CS: Provides the real CP with a fake synchronization information.
   $\textbf{\textit{BootNotification.conf}}(\{currentTime'::status'::interval'\}_{K_{session}})$
8: CP: Computes the parameters and configures its system according to the "CS".
   $currentlTime := Dec_{K_{session}}(arg_{CP,1}); status := Dec_{K_{session}}(arg_{CP,1})$
   $interval := Dec_{K_{session}}(arg_{CP,1})$
```

cure fashion. However, this new version omits this functionality instead of considering it as a mandatory one. The ease of cloning of the static ID-based RFID cards is also an impersonation problem in OCPP contexts by applying potential electronic equipment located close to the CPs and eavesdropping IDs, where attackers might also prove arbitrary IDs so as to find the correct ID for the access [12]. They may also simplify the effort by approximating the seeking range, making use of a foreknown ID from a legitimate RFID card [12]. If in addition, the access is associated with multiple users through an OCPP *parent idTag* (to manage ID tokens as a ‘group’,), the adversarial influence will be extreme.

3.2 Stage 2: Transactions and Control

All transactions must be authorized by the CP. However, attackers can trivially take advantage of wireless communications by **jamming**. In this way, they might oblige the CP to locally manage authorizations and queuing transactions that must later be authorized by the CS. Indeed, OCPP forces the CS to always “accept” any queued request regardless of whether the process has been authenticated or not.

Distortion in *c* and **over-/under-shooting** of *e* are also possible by manipulating the changing profiles defined by the CS through *SetChargingProfile.req*(connectorId, chargingProfiles)*, where *chargingProfiles* establish the amount of power (watts) or current (amperes) that can be delivered per time interval to a CP or a group of CPs together with their charging schedules (time period, rate). Attackers might, for example, vary the charging rate or perturb its schedule (e.g., to keep the charging limits to maximum values at all times, restart a schedule periodically or define profiles with beneficial time periods) so as to hit the stability of the grid. This same threat also occurs when malicious CSs or MitMs serving as CSs are able to inject spoofed charging profiles through the *RemoteStartTransaction.req*(idTag,[connectorId],[chargingProfile])*, which is initiated by the CS and activated when the CP accepts and transfers the PDU (payload data unit) *StartTransaction.req*(connectorId, idTag, meterStart, timestamp)* (steps 4-8 of Algorithm 3). If, in addition, these profiles are related to particular connectors (EV supply equipment (EVSEs) ID or $\mathrm{ID}_{\mathrm{EVSE}}$) and open transactions, MitMs might also dynamically upgrade them (using *SetChargingProfile.req**) with a planned charging limit for or against specific users (e.g., reduce the maximum rate to minimum values). Note that another way to do this would be to first preconfigure the desired profile in a CP, intentionally according to the attacker's convenience (e.g., increase the charging rate), to later push the CP to enter in off-line mode and take advantage of the illicit authorization.

Algorithm 3 Vulnerability in the start/stop transaction (1/2)

```
1: $\mathbf{CP}_1 \rightarrow \mathbf{A}$ and $\mathbf{CP}_2 \rightarrow \mathbf{A}$: $A$ "only" interferes the channel $CS \longleftrightarrow \mathrm{CP}_1$ through Algorithm 1, captures the $ID_{CP_1}$ in order to update its initial knowledge state $K_A^0 = \{KPub_A, KPriv_A, KPub_{CA}, KPub_{CP_2}, ID\_CP_2, K_{session_{CP_2}}\}$. At this point $A$ further takes advantage of the TLS attack to impersonate the identity of $ID_{CP_2}$ as $ID_{CP_1'}$ so as to regulate the entire charging process as a legitimate node. At the end of this process, the attacker obtains $K_{session_{CP_1}}$ associated with $ID_{CP_1'}$, i.e.:
   $K_A^n = K_A^0 \cup \{KPub_{CP_1}, ID_{CP_1}, K_{session_{CP_1}}, KPub_{CS}, ID\_CS, Cert_{CS}\}$
2: $\mathbf{CP}_2 \rightarrow \mathbf{A} \rightarrow \mathbf{CS}$: Boots the system with the $ID_{CP_1'}$ through the MitM, which in turn relays the request to the CS.
   BootNotification.req$(\{CP_2Model::CP_2Vendor\}_{K_{session_{CP_{2-1}}}})$
3: $\mathbf{CS} \rightarrow \mathbf{A} \rightarrow \mathbf{CP}_2$: Accepts the boot of $\mathrm{CP}_2$ and starts the transactions.
   $status$: ='Accepted'
   BootNotification.conf$(\{currentTime::status::interval\}_{K_{session_{CP_{1-2}}}})$
4: $\mathbf{CP}_1 \rightarrow \mathbf{A}$: Requests the authorization of an user with $idTag_{CPS_1}$ to its supposedly "CS".
   authorize.req$(\{idTag_{CPS_1}\}_{K_{session_{CP_1}}})$
5: $\mathbf{A}$: Updates its $K_A^n$, and from now on ignores the communications with $\mathrm{CP}_1$.
   $idTag_{CPS_1} := Dec_{K_{session_{CP_1}}}(arg_{A,1})$; $K_A^{n+1} = K_A^n \cup \{idTag_{CPS_1}\}$
6: $\mathbf{CS} \rightarrow \mathbf{A}$: Requests the starting of a remote transaction to the supposed $ID_{CP_1'}$ under an specific charging profile (optional action).
   $chargingProfile := idChProfile::typeProfile::chargingShedule$; $chargingSchedule :=$ 08:00-13:00::$X$kWh
   RemoteStartTransaction.req*$(\{idTag_{CPS_1}::ID_{\mathrm{EVSE}}::charingProfile\}_{K_{session_{CP_1}}})$
7: $\mathbf{A} \rightarrow \mathbf{CP}_2$: Captures the information and modifies the charging profile.
   $\mathrm{ID}_{\mathrm{EVSE}} := Dec_{K_{session_{CP_1}}}(arg_{A,1})$; $chargingProfile' := idChProfile::typeProfile::chargingShedule'$
   $chargingSchedule' :=$ 08:00-18:00::$Y$kWh such $X\mathrm{kWh} \neq Y\mathrm{kWh}$
   RemoteStartTransaction.req*$(\{idTag_{CPS_1}::ID_{\mathrm{EVSE}}::chargingProfile'\}_{K_{session_{CP_2}}})$
8: $\mathbf{CP}_2 \rightarrow \mathbf{A} \rightarrow \mathbf{CS}$: $\mathrm{CP}_2$ accepts the transaction; and $A$ relays the confirmation.
   $status$: ='Accepted'
   RemoteStartTransaction.conf*$(\{status\}_{K_{session_{CP_{2-1}}}})$
9: $\mathbf{CP}_2 \rightarrow \mathbf{A} \rightarrow \mathbf{CS}$: $\mathrm{CP}_2$ initiates the transaction, providing $e$ to the EV connected in the $\mathrm{CP}_2$, and $A$ relays the request.
   StartTransaction.req*$(\{ID_{\mathrm{EVSE}}::idTag_{CPS_1}::meterStart::timestamp\}_{K_{session_{CP_{2-1}}}})$
10: $\mathbf{CS} \rightarrow \mathbf{A} \rightarrow \mathbf{CP}_2$: Accepts the transaction and $A$ relays the confirmation.
   $status$: ='Accepted'; $expireDate$: ='01/12/2017'
   StartTransaction.conf*$(\{status::transactionId::expireDate\}_{K_{session_{CP_{2-1}}}})$
11: $\mathbf{CP}_2 \rightarrow \mathbf{A} \rightarrow \mathbf{CS}$: Disrupts the transaction for the energy purchase.
   StopTransaction.req*$(\{meterStop::transactionId::reason::timestamp\}_{K_{session_{CP_{2-1}}}})$
12: $\mathbf{CS} \rightarrow \mathbf{A} \rightarrow \mathbf{CP}_2$: Stops the transaction and charges the purchase to $idTag_{CPS_1}$.
   StopTransaction.conf*()
```

In addition, many CPs have already been designed to offer bidirectional interfaces for power charging/discharging, where batteries should charge during off-peak periods and discharge during peak times. If attackers are able to affect several controllers of diverse CPs and reverse the power flow into the distribution line during off-peak periods at the same time, this can result in unanticipated power flows, potentially overloading distribution networks. This effect may occur when multiple hacked CPs inject energy into EVs during peak periods, and probably with fake charging profiles (e.g., steps 4-8 of Algorithm 3). Indeed, MitMs might interfere with different profiles to intentionally attract a high demand at peak periods when the batteries are being discharged into the grid. To do this, the adversary needs to first modify the current charging profiles such that the intensity of Wh has to be greater at peak hours (e.g, X_1kWh) or equal to the power consumption in off-peak periods (X_2kWh), such that $X_1\text{kWh} \geq X_2\text{kWh}$. This change can be done through the instructions *SetChargingProfile.req** or *StartTransaction.req**. Algorithm 3 also shows the ability of an attacker to forward the transactions from a CP_1 to another malicious CP_2, whose power purchases may be charged to the user authorized in CP_1. This threat, which additionally entails an **energy theft/fraud** attack, requires, on the one hand, that the CP_2 must impersonate the identity of CP_1, and CP_1 must think that the connection has properly been established with the CS. And, on the other hand, the threat assumes that A has been able to compromise the channel $CS \longleftrightarrow \text{CP}_1$ from the deployment and fool the CS using the identity of ID_{CP_2}, thanks in part to Algorithm 1.

Any change in meter values reported in the *meterStart* of *StartTransaction.req** and in the *meterStop* of *StopTransaction.req*(meterStop, timestamp, transactionId, stopReason)* may also cause an over-provisioning of e and energy theft/fraud. For example, if an EV needs 10kWh and starts the transaction with a meter value X for a connector, the CP must stop the transaction after $X+10$kWh. A malicious CP serving as a MitM might request the same transaction but starting with $X+10$kWh, thereby obtaining unmetered energy at the end of the transaction. Likewise, third parties emulating legitimate CSs (and by means of the *StartTransaction.req**) may also abuse the energy levels by previously synchronizing a new LAL with false ID tags through the command *SendLocalList* (cf. Section 3.1). Once access has been granted, malicious EVs linked to these false ID tags can obtain unmetered energy. Moreover CPs may be intentionally tampered with, more likely by insiders with access to **replace or configure components**, to consume the energy demanded by forwarding a small fraction of reserved power to another EVSE while the EV receives only a fraction of the energy purchased [8].

Through *ReserveNow.req(connectorId, expiryDate, idTag, reservationId)* a CS can demand an ID_{EVSE} before starting a transaction, but such a demand has to be validated by the CP with an accepted/rejected confirmation. If the CP receives a reservation request with $\text{ID}_{\text{EVSE}} > 0$, then it has to refuse any incoming ID tags that want to connect with the reserved EVSE, except when the incoming ID tag coincides with the ID tag of the reservation. This can even invalidate the use of connectors for a long time if the *expiryDate* parameter is not limited properly. If $\text{ID} = 0$, then the CP does not reserve a specific connector of the EVSE, but it has to make sure that at all times during the validity of the reservation, a connector of the EVSE remains available for the reserved ID tag. In these cases, energy theft may also arise by reserving an EVSE ID associated with a false ID tag, which has first had to be incorporated into the LAL, as stated earlier. The result of the threat may also cause a **denial of energy service** (DoES, aimed at a service disrupt $\subseteq \text{threats}_A(e)$), which may become more critical when there is a selective downgrading or an existing reserve activation with a long/unlimited expiration. This is because CPs are not authorized to terminate the process until the time specified in the expiry field is reached. Another way to cause DoES and disrupt the reservation procedure would be to tamper with a CP so that it emits fake signals attributing a faulty or unavailable status to an EVSE demanded. Moreover, a MitM acting as a supposedly legitimate "CS" might also halt any reserve process by taking advantage of the command *CancelReservation.req*(reservationId), and a CP that has been tampered with might likewise send back rejected signals so as to disobey any cancellation order and obtain extra provisioning.

For control, the CS should periodically receive the responsiveness degree of each CP within the infrastructure. To do this, the CP must first request the current date for synchronization with the CS through the instruction *Heartbeat.req()*. After this process, the CS can configure the inactivity interval (except during OCPP information exchanges), specifying in this case both the function key and its value associated with the heartbeat interval as described in Section 3.1 for *ChangeConfiguration.req**. Any **desynchronization** with the CS or tampering with configuration values could lead to malfunctions or unavailability of charging resources (DoES). Similarly, metering samples are also transferred to the CS periodically using *MeterValues.req* (also mentioned in Section 3.1). Through this instruction, the CS receives information from a specific EVSE ($\text{ID}_{\text{EVSE}} > 0$) or from the main power meter ($\text{ID}_{\text{EVSE}} = 0$). There exists the option of emitting further information about: (i) The type of measurement; (ii) the type of context, defining, for example, when samples take place, location (inlet or outlet), unit (Wh by default), format (raw by default or signed data); and (iii) the time-stamp for measured values. Nonetheless, this optional feature should be mandatory to specify when, where and how they were measured.

Apart from refreshing the LAL through a new list version in each CP (cf. Section 3.1), the CS can also order the LAL and the cache to be wiped. As mentioned, both components act as local authorization elements when there is no connectivity with the CS, so malicious entities might take advantage of the commands *SendLocalList.req* and *ClearCache.req()* to delay or disrupt the authorization processes, or even to leave the CPs inoperative by forcing them to move to off-line mode. For the attack to be successful, the attacker first needs to clear the LAL by remotely sending a new empty local authorization list and the cache, and then disrupt the communication with flooding, collisions in channels, jamming or any other similar DoS attacks, such as dropping. Another way to perform a **functionality reduction** would be through the *ChangeAvailability.req(connectorId, availabilityType)*, which allows the CS to block a specific connector ($\text{ID}_{\text{EVSE}} > 0$) or the availability of a CP as a whole ($\text{ID}_{\text{EVSE}} = 0$). This weakness may even encourage stronger attackers to selectively exploit system vulnerabilities, and in this way cause a resource downgrading. Namely, attackers might, for example, launch multiple and coordinated attacks (such as those described in this paper) to degrade the performance of the system step by step by reducing the effectiveness of the resources and energy provisioning.

Malicious entities might also act as legitimate CSs to request, through the *UnlockConnector.req(connectorId)*, the activation of those EVSE ($\text{ID}_{\text{EVSE}} > 0$) with malfunctions to impact either on the charging of EVs, or on the local grid performance and its stability. Another way of interfering with the availability and integrity of resources, would be to reset CPs in a hard (full reboot and immediate) or soft (delay reboot until no more transactions are active) manner, both contained within the instruction *Reset.req(resetType)*. Finally, the command *DataTransfer.req(vendorId)* allows CPs and the CS to send information of non-supported functions to the CS or CPs, respectively. These entities have to validate the request according to the functions provided by the vendor, returning the corresponding status (accepted, rejected, or unknown). In this scenario, attackers might, for example, overload a CP/CS by requesting non-supported functions or transactions with irregular lengths. In addition, the specification recommends the use of a unique vendor ID associated with a domain name system (DNS) namespace, but it can be easily forged.

3.3 Stage 3: Notification and Maintenance

Accountability and safety-related audit measures are considered by OCPP. However, this information must contain *trustworthy* information including location (*where*), identity (*who*), problem (*what* and *why*), activities (*how*) and time (*when*). As shown above, most commands use a common identifier, generally $\mathrm{ID}_{\mathrm{EVSE}} > 0$ for connectors and $\mathrm{ID}_{\mathrm{EVSE}} = 0$ for the CP or its meter. It tends to be a generic value, without identifying a particular CP from among a set. So, it is necessary to offer fine-grained information based on locations and unique identifiers, as well as the MAC address of each device involved. For example, the current release of OCPP only notifies through *StatusNotification.req(connectorId, errorCode, status, [timestamp])* the type of faulty component in an $\mathrm{ID}_{\mathrm{EVSE}}$, the type of error and its status, discarding in this case, the time-stamp as a mandatory parameter.

Maintenance and diagnostic tasks are also vulnerable to MitMs, sinkholes or wormholes. CPs may be provided with false URLs to intentionally retrieve firmware versions from malicious hosts, or upload their diagnostic files to such locations. For this, attackers have to force the CP to accept the location field contained within the *UpdateFirmware.req*(location, retrieveDate, [retries], [retryInterval])* and *GetDiagnostics.req(location, [retries]), [retryInterval], [startTime], [stopTime])*. Both commands allow optionally specifying the number of retries to download a firmware or upload a diagnostic file, as well as the interval in seconds for each attempt. At this point, the attacker could intentionally increase the number of retries to short time intervals to reach a DoS, or perform a DNS attack. Moreover, according to OCPP, if these two fields are not present, the CP is responsible for deciding how many frequent retries it will undertake, and how long to wait between them. If the controller is compromised, the effect would evidently be the same. So it is prudent to mandatorily fix these parameters using reasonable configurations from the CS, where the communication channels need to be secured. This also includes the mandatory use of transport layer security for FTP and HTTP together with identity management and authorization [9].

However, OCPP-1.6 positively addresses the firmware update procedure by adding a new optional field (*startTime*) to the *UpdateFirmware.req**, which helps the CP decide when to start with the installation. Again, a short or long *startTime* value may once more impact on the updating of a CP and its availability. Apart from this, v1.6 also offers a new error field to the *FirmwareStatusNotification.req*(status)* so as to notify of the real status of the firmware (downloaded, failed or idle). But despite this improvement, the protocol does not consider (i) the case where the installation is in progress and requires be halting, and (ii) there is no suitable control of parameters and states. So any successful abort could put the restoration process at maximum risk.

3.4 Beyond OCPP: Common Threats and Vulnerabilities

In this section, we explore a number of common and cross-cutting threats at the different levels of the protocol $\in$ $\text{threats}_A(a,c)$, and beyond the OCPP protocol design. For instance, CPs can be highly prone to **Byzantine** faults due to the architectonic features of the system and its interdependencies; or to **advanced persistent threats** (APTs) which include **stealth attacks**. Within this last category, we underline (i) **side- and covert-channel** exploitation by observing the traffic load in the physical level (e.g., timing attacks to extract details about computations and parameters) or hiding data flows within the regular messages; and (ii) **passive traffic analysis** to learn from the network topology and/or deduce users' activity patterns, lifestyle and routes of end users. A subcategory of this last threat is neighborhood monitoring, in which intruders can penetrate the channels of those CPs located around residential areas to discover, e.g., unoccupied dwellings. As a complementary action, attackers may also **deliberately expose** critical data to other unauthorized entities or competitors.

The use of web technologies (e.g., SOAP/XML, JSON-WS, HTTP, FTP) generally adds vulnerabilities; e.g., the lack of parameter validation may entail threats related to **false injection** on official websites belonging to users or the control. Other typical threats in c, and particularly in SOAP/XML, are as follows [13]: (Distributed) DoS where attackers might, for example, lead buffer overflow attacks or exhaust the memory of parsers in CSs/CPs; SOAPAction **spoofing** with **obfuscation actions** within the body of the HTTP request to bypass authentication mechanisms and firewalls; WS-addressing spoofing with different request and response endpoints; or XML injection, modification or delays by sending oversized payloads (e.g., *SendLocalList.req* where *updateType* retains the full list) or by taking advantage of XML namespaces to overload the parsing process (coercive parsing). On the other hand and as stated in Section 3.1, WS connection-based networks establish identities during the handshake to avoid implementing authentication mechanisms at the network level. However, this feature may not be very effective when JSON objects are distorted as is the case of SOAP/XML [14], where their communication channels may be threatened by typical attacks such as DoS or on-path. Moreover, all synchronization and DoS problems arising from TCP/IP protocol usage also remain. Attackers could, for example, forge TCP RST (reset) segments to disrupt any TCP connection between two peers, or cause DoS through Jellyfish (multiple delays in the TCP transmissions) or SYN flooding (multiple TCP connections without completing) to exhaust resources. Lastly, the absence of robustness mechanisms in the different (remote, local and wireless) communication infrastructures, the lack of non-repudation measures together with strong authentication and authorization mechanisms and the inherent weaknesses inherited from current technologies with limited resources (e.g., smart meters in CPs) $-$ not only in the OCPP domains [12] but also in the majority of Smart Grid applications [9] $-$, do not currently help prevent $\text{threats}_A(a,c)$.

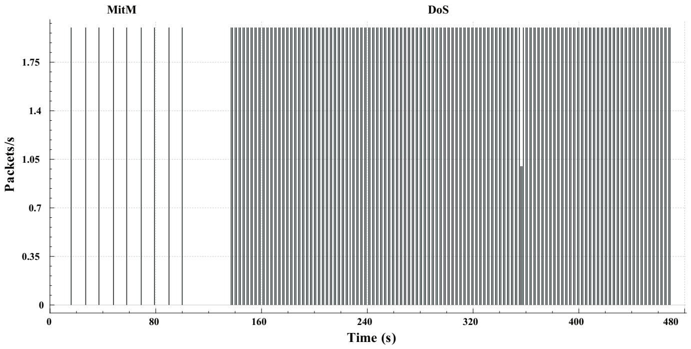

Figure 2: Denial of service defined in Algorithm 2

4 Challenges and Discussion

This section presents a practical analysis carried out through the OCPP simulator implemented by GIR (Giaume Industrie & Recherche) in [15]. This simulator, called *ocppjs* and based on Nodes.js, consists in a framework capable of providing the necessary tools to exchange messages between the CS and the CPs. The last version of ocppjs includes the last OCPP releases, SOAP over HTTP and WS (RFC 6455). Based on this environment, our validations principally focus on WSs and on the implementation of Algorithms 2 and 3 by encompassing the most significant threats described in this paper such as DoS, impersonation, data tampering, spoofing, fraud and energy theft. For the experiments, we have used several virtual machines (VMs) based on Kali Linux 2016, where we have applied ettercap for the poisoning of the network and the MitM, the filters of which specify actions to be executed during communications. For example, we have defined the following filter for Algorithm 2:

```
if (ip.proto == tcp && tcp.scr == <local port> && search(DATA.data, ‘
    heartbeatInterval’)){
   DATA.data + 1 = 0xX;
   replace(’current interval’,‘new interval’);}
```

such that DATA.data + 1 defines the new dimension of the packet associated with the heartbeat interval. Namely, the simulator specifies, by default, the value of the heartbeat interval ($interval'$ in Algorithm 2) to 1000 seconds, while the filter reduces its value to 10 seconds each time. This functional feature evidently introduces a slight DoS such as illustrated in Figure 2. This figure depicts the packet delivery impulses obtained from Wireshark and produced between the CP, A and the CS, clearly showing when the MitM and DoS attacks are launched over time. The first actions include the connection with A together with the manipulation of the packet *BootNotification.conf*; the rest of the impulses correspond to the result of modifying the heartbeat interval to the CS.

Ettercap was also applied in the implementation of Algorithm 3 and in the first phases of the attack, so as to poison the network and impersonate the ID_{CP_1} and the ID_{CS}, respectively. The manipulation of the network data streams among A, the CS and the charging points was, to the contrary, carried out by the tool netsed, which defines:

```
netsed tcp {<localPort>} {<remoteNode>} {<remotePort>}
           {s/<search>/<replace>}
```

such that remote node and port correspond to the IP and port of the VM containing the CS, and s/<search>/<replace> = $s/ID_{CP_2}/ID_{CP_1}$ to create a tunnel between the CP_2 and the CS using the ID_{CP_1}. Figure 3 shows the experimentation results together with the steps of the threat, where: A first takes over the communications between CP_1, CP_2 and the CS (steps 1-5 of Algorithm 2), and produces the MitM to later initiate a fake transaction using a *RemoteStartTransaction.req** from CS (modified by A later) and a *StartTransaction.req** from CP_2. Steps 5-12 of Algorithm 2 are shown in Figure 3 through the thicker impulses, in which A deals with manipulating the traffic between the CS and CP_2, ignoring the communication of CP_1 (square signals in Figure 3). The fraud is successful once the simulation exceeds 320 seconds, and the CS charges CP_1 for the energy purchase.

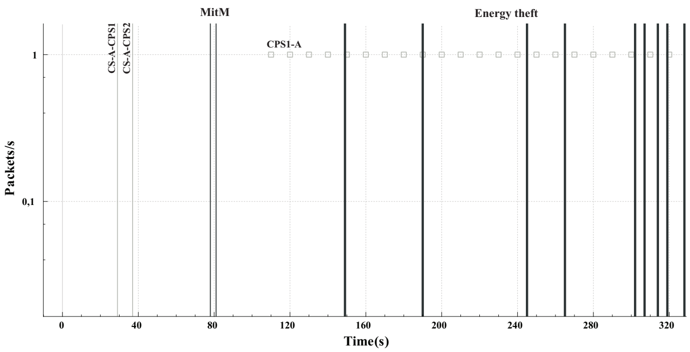

Figure 3: Fraud and energy theft defined in Algorithm 3

Apart from this, Table 1 summarizes the security analyses done in this paper and the influence of the diverse threats$_A(a,c,e)$ on the end services, looking carefully at the availability, integrity, and confidentiality of the communications, as well as the energy provisioning to end users. This table also characterizes, with a '$*$' symbol, the most relevant OCPP threats and their dependent-layered communications, emphasizing in this case the existing dependencies between threats$_A(a,c)$ and threats$_A(e)$. These dependencies highlight the existing security insufficiencies running across the charging architecture, where different stakeholders and local distribution grids may be affected, and whose effect might be widely extended towards other critical infrastructures. That is, from the table and the simulations, we highlight the vulnerability of the protocol to potential and multiple attacks such as DoS, data tampering or stealth attacks, all of them motivated in part by the actions led by MitMs in threats$_A$(a,c). The main aim of the attackers in these highly critical contexts, consists in shaking control and the natural flow of e to directly or indirectly hit the grid, though this also implies putting, in most cases, the security and safety of other main stakeholders, such as the control systems, their remote assets and end-users, at risk. Only in exceptional cases, such as energy theft, are the end-users targeted, through fraud.

Table 1: Goals of the Attack and impact on OCPP contexts

	Threats$_A$(a,c)[a]			**Threats$_A$(e)**			**Impact on**			
	Disrupt	Distort	Disclosure	Disrupt services	Overloading	Energy Theft	Control - CS-CP	Asset - CP	User - EV	Power grid
MitM	◇	◇	◇	∗	∗	∗	✓	✓	✓	✓
Impersonation		◇		∗	∗	∗	✓	✓	✓	✓
over-/undershooting		◇		∗	∗					✓
Abuse of off-line	○			∗	∗	∗	✓	✓	✓	✓
Data tampering		◇		∗	∗	∗	✓	✓	✓	✓
Fraud/energy theft		○			∗	∗			✓	✓
False injection		◇		∗	∗	∗	✓	✓	✓	✓
On-path att.	○	○		∗			✓			
Render useless	◇			∗			✓	✓	✓	✓
Redirect traffic	○	○	○				✓			
Byzantine faults	◇	◇	◇	∗	∗	∗	✓	✓		
APTs	◇	◇	◇	∗	∗	∗	✓	✓	✓	✓
Stealth Att. $\subset$ APTs	◇	◇	◇	∗	∗	∗	✓	✓	✓	✓
Side-channel att.			○	✓			✓			
Covert-channel att.			○				✓			
Passive traffic an.			○				✓	✓	✓	
Deliberate expos.			◇				✓	✓	✓	✓
Web/TCP-IP	○	○	○	∗	∗	∗	✓	✓	✓	✓
DoS	◇			∗			✓	✓		✓
DoES				✓						✓
de-synchronization	◇	◇		∗			✓	✓		✓
Physical att.	◇			✓			✓	✓	✓	✓
Jamming $\subset$ Phys. att.	◇			∗			✓	✓	✓	✓
Replacement		●		∗	∗	∗	✓	✓	✓	✓

[a] ◇: has influence in *a* and *c*, and ● and ○: impact on *a* and on *c*, respectively.

To address threats$_A(a,c)$ and OCPP's vulnerability to MitMs, future releases and architectures should be subject to tunneling and secure end-to-end communications using secure protocols (HTTPS, FTPS). These networks must also be equipped with (i) lightweight detection mechanisms based on knowledge of the environment like, for example, restricted intrusion detection systems (IDSs), and (ii) lightweight trust-based systems to make sure that the information received by a node is reliable. Apart from constrained firewalls and IDSs, the network designs should also be based on strong authentication and authorization mechanisms in the whole TCP/IP stack [12, 16, 17] where the actions in the field should be restricted to roles, permissions and contextual attributes as specified in IEC-62351-(7-8), and logged. These logs will be vital to properly address all those aspects related to governance, audit and maintainability, where OCPP accountability should be restricted to fine grain information and non-repudiation measures to explain, in detail, a specific situation and responsible entity.

The prevention of Byzantine faults, APTs and their derivatives over large distributions is, however, not such a trivial task. Constant resource monitoring and design of fault-tolerant services under dynamic proactive measures and lightweight self-healing systems are also fundamental in providing a wide area situation awareness and managing unforeseen faults. Other good practices against stealth threats such as side- and covert channels, code injection or active eavesdropping, should be validation, assessment and updating of systems, application of specialized techniques (e.g., hiding timing variations, blinding, masking), reputation mechanisms, security controls through proper policies and governance, and cryptography as recommended by the IEC-62351-

3 [18]. Likewise, it is of utmost importance to improve the architecture of the controllers and meters to make them tamper-resistant to physical manipulation or attacks.

All these security issues still form part of the security challenges in OCPP contexts, and hence, they must form part of future work; particularly in OCPP-2.0 where the intention is to extend v1.6 to further integrate the role of the end-user in the OCPP architecture. The goal is to allow the user to choose the type of charging profile with its purchase schedule, the amount of power and its pricing scheme with its tariffs. However, this feature will also bring about numerous problems if any new charging policy collides with the ones pre-established by the providers, entailing serious threats in *e* such as energy theft and power overload. Therefore, it is essential to pay special attention to each part of the specification and its architecture to consider some of the security requirements mentioned in this paper.

Acknowledgment

This work has been partially supported by the research projects PERSIST (TIN2013-41739-R) and SADCIP (RTC-2016-4847-8), both financed by the Ministerio de Economía y Competitividad. The authors also thank J. Rodriguez (NICS Lab.) for his valuable comments, ideas and help.

References

[1] OCA. Open charge point protocol 1.6. Technical report, Open Charge Alliance, Duiven, The Netherlands, 2015.

[2] A. Rodriguez-Serrano, A. Torralba, E. Rodriguez-Valencia, and J. Tarifa-Galisteo. A communication system from EV to EV service provider based on ocpp over a wireless network. In *IEEE Conference Industrial Electronics Society*, pages 5434–5438, 2013.

[3] J. Schmutzler, C. A. Andersen, and C. Wietfeld. Evaluation of OCPP and IEC 61850 for smart charging electric vehicles. In *Electric Vehicle Symposium and Exhibition (EVS27)*, pages 1–12, 2013.

[4] M. Faschang, M. Nöhrer, J. Stöckl, and F. Kupzog. Extensible co-simulation framework for electric vehicle charging infrastructure testing. In *SmartGridComm*, pages 182–187, 2014.

[5] OCA. Standarization of OCPP at OASIS and IEC. Technical report, July 2016.

[6] E. Rescorla and B. Korver. RFC 3552: Guidelines for writing RFC text on security considerations. Internet Society Req. for Comm., 2003.

[7] T. Sao, R. Alexander, M. Dohler, V. Daza, A. Lozano, and M. Richardson. RFC 7416: A security threat analysis for routing over low-power and lossy networks. version 11, draft-ietf-roll-security-threats-05, 2013.

[8] R. Falk and S. Fries. Securely connecting electric vehicles to the smart grid. *International Journal on Advances in Internet Technology*, 6(1 and 2):57–67, 2013.

[9] J. Liu, Y. Xiao, S. Li, W. Liang, C. Chen, and C. L. Philip. Cyber security and privacy issues in smart grids. *IEEE Comm. Surveys & Tutorials*, 14(4):981–997, 2012.

[10] Jan Jürjens. Secure systems development with UML. In *Fundamental Approaches to Software Engineering*, volume XIX of *LNCS*, page 300. Springer Berlin Heidelberg, 2005.

[11] Jan Jürjens. UMLsec - presenting the profile. In *Sixth Annual Workshop on Distributed Objects and Components Security*, 2002.

[12] Fabian Van den Broek, Erik Poll, and Bárbara Vieira. *Securing the Information Infrastructure for EV Charging*, pages 61–74. Springer International Publishing, Cham, 2015.

[13] C. Mainka, J. Somorovsky, and J. Schwenk. Penetration testing tool for web services security. In *IEEE Eighth World Congress on Services*, pages 163–170, 2012.

[14] Vanessa Wang, Frank Salim, and Peter Moskovits. *The Definitive Guide to HTML5 WebSocket*. APRESS, New York, NY, USA, 2013.

[15] GIR. ocppjs - An experimental OCPP Simulator. `http://www.gir.fr/ocppjs/`, last access August 2016, 2012-2013.

[16] A. Chan and J. Zhou. A secure, intelligent electric vehicle ecosystem for safe integration with the smart grid. *IEEE Transactions on Intelligent Transportation Systems*, 16(6):3367–3376, Dec 2015.

[17] A. Chan and J. Zhou. Cyber-physical device authentication for the smart grid electric vehicle ecosystem. *IEEE Journal on Selected Areas in Communications*, 32(7):1509–1517, July 2014.

[18] ISO/IEC. ISO/IEC 62351 parts 1–8: Security. TC-57 (Power Systems Manag. and Associated Infor. Exchange), 2011.