Università degli Studi di Padova

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Università di Padova

Dipartimento Di Ingegneria Dell'Informazione (DEI)

Corso di laurea magistrale in Ingegneria Informatica

Architettura e protocolli dell'infrastruttura di ricarica elettrica: vulnerabilità, attacchi e rimedi

Relatore:
Prof. Migliardi Mauro

Laureando:
Signorato Luca
1232102

Anno Accademico 2021-2022

28 febbraio 2022

Abstract

L'infrastruttura di ricarica elettrica, a partire dalle singole colonnine, è considerabile alla stregua di ogni altro sistema Internet of Things, essendo connessa e necessitando di interazioni locali e remote. In quanto tale diventa critico il problema della sicurezza di queste infrastrutture vitali per la mobilità del futuro per garantire confidenzialità, integrità e disponibilità dei servizi offerti tramite questa infrastruttura.
In collaborazione con Gruppo SIGLA S.r.l., società del gruppo RELATECH, si è svolto un lavoro di tesi incentrato sulla cyber-security dell'infrastruttura di ricarica elettrica analizzandone le varie componenti ed i protocolli utilizzati con un focus principale sulla colonnina e le interazioni con il sistema centrale.

Indice

Capitolo 1

Introduzione

La transizione verso la mobilità elettrica è un fenomeno ormai avviato e in rapida crescita. I dati presentati di recente nel SMART MOBILITY REPORT 2021[7] (Politecnico di Milano) ne sono una prova e il trend degli ultimi anni lo dimostra. L'Europa risulta il più grande mercato mondiale, con quasi 1,4 milioni di veicoli elettrici immatricolati nel 2020 (+137% rispetto al 2019). Nel seguente scenario l'Italia si piazza al settimo posto a livello europeo grazie alle 59.875 auto elettrificate immatricolate nel 2020, con una crescita del 251% rispetto all'anno precedente. Il trend continua anche nel 2021, tanto che nei mesi di Luglio, Agosto e Settembre 2021, le immatricolazioni di veicoli elettrici hanno registrato una tendenza positiva rispetto ai corrispettivi mesi del 2020, rispettivamente pari a +217% , +73% e +162%.

In questo scenario diventa sempre maggiore la necessità di spingere sulla contemporanea diffusione delle infrastrutture per supportare la mobilità elettrica, indirizzandosi sulla installazione di colonnine di ricarica, in particolar modo sulle vie ad alto scorrimento e nelle aree urbane.

Si può dire come la crescita delle infrastrutture di ricarica elettrica segua quella dei veicoli elettrici, precisamente, a fine 2020, si stimano in Europa oltre 285.000 punti di ricarica pubblici, in crescita di circa il 35% rispetto all'anno precedente. Sul suolo italiano si attestano oltre 13.300 punti di ricarica pubblici, con una crescita di quasi il 46% rispetto all'anno precedente. Trend che si conferma anche nel 2021 dove si contano circa 21.500 punti di ricarica considerando quelli pubblici e privati ad accesso pubblico.

Alla luce di tali numeri e della loro crescita costante, diventa assolutamente necessario interrogarsi sulla cyber-security delle infrastrutture di ricarica elettrica. Diventa ancor più chiaro se si considera il fatto che le colonnine sono connesse e necessitano

di interazioni locali e remote, potendo quindi essere paragonate al mondo Internet of Things (IoT).
In questo contesto e considerati gli obiettivi di riduzione delle emissioni di gas inquinanti e di quelli per la realizzazione di Smart Cities, l'infrastruttura di ricarica è facilmente prevedibile entri a fare parte delle cosiddette "Infrastrutture Critiche" che sostengono economia e società presenti e future.

1.1 Architettura di riferimento

Prima di addentrarsi nell'analisi delle possibili vulnerabilità riguardanti l'infrastruttura di ricarica elettrica, è importante aver bene in mente come essa sia articolata e quali entità la compongano in linea generale.
A tal proposito, si può far riferimento alla Figura 1.1 in cui viene mostrata un'architettura di riferimento. Come si può notare, è composta da molte entità tra cui: la colonnina (Charging station - Charging plaza in figura) ed il sistema centrale (CSMS, Charging Station Management System). Si può affermare come definiscano la parte principale e più critica dell'infrastruttura e come le comunicazioni possano differire dalla topologia con cui queste entità vengono modellate. Nonostante solitamente la comunicazione sia end-to-end, possono esistere altre configurazione che prevedono un'entità nel mezzo (ad es. proxy).
Oltre a ciò si hanno altri attori i quali possono influenzare le comunicazioni tra colonnina e sistema centrale ed anche il processo di ricarica; a questa categoria appartengono:

- il meccanismo di roaming, il quale mette in collegamento diversi MSP (Mobility Service Provider) che sono fornitori del servizio di ricarica elettrica. Il roaming è considerato un elemento molto importante in quanto migliora l'esperienza utente perché permette all'utente di usufruire del servizio di ricarica su colonnine non appartenenti al fornitore con cui ha un contratto.

- DSO (Distribution System Operators) rappresentano le entità responsabili della distribuzione e gestione dell'energia nel passaggio tra la sorgente ed il consumatore finale. Comunicano con il sistema centrale dando delle previsioni sulla capacità di una certa zona imponendo così dei limiti all'energia/corrente utilizzabile.

- EMS (Energy Management System) indica un sistema di gestione dell'energia, viene applicato in scenari locali. Allo stesso modo dei DSO può imporre dei

limiti derivanti dalla gestione di altre applicazioni locali. Può comunicare direttamente con la colonnina.

In aggiunta, è presente la parte di autenticazione tra l'utente e la colonnina. In particolare può comprendere un'azione fisica da parte dell'utente sulla colonnina (ad es. presentare una carta identificativa oppure un codice PIN) oppure l'autenticazione potrebbe essere gestita anche virtualmente tramite un'applicazione la quale permette all'utente di accedere al proprio profilo e iniziare una ricarica.
Si ha poi il meccanismo di collegamento tra colonnina e veicolo eseguito tramite un cavo, anche qui possono esistere delle comunicazione tra le due parti.
Infine, è presente la componente riguardante la gestione e mantenimento sia lato sistema centrale sia lato colonnina.

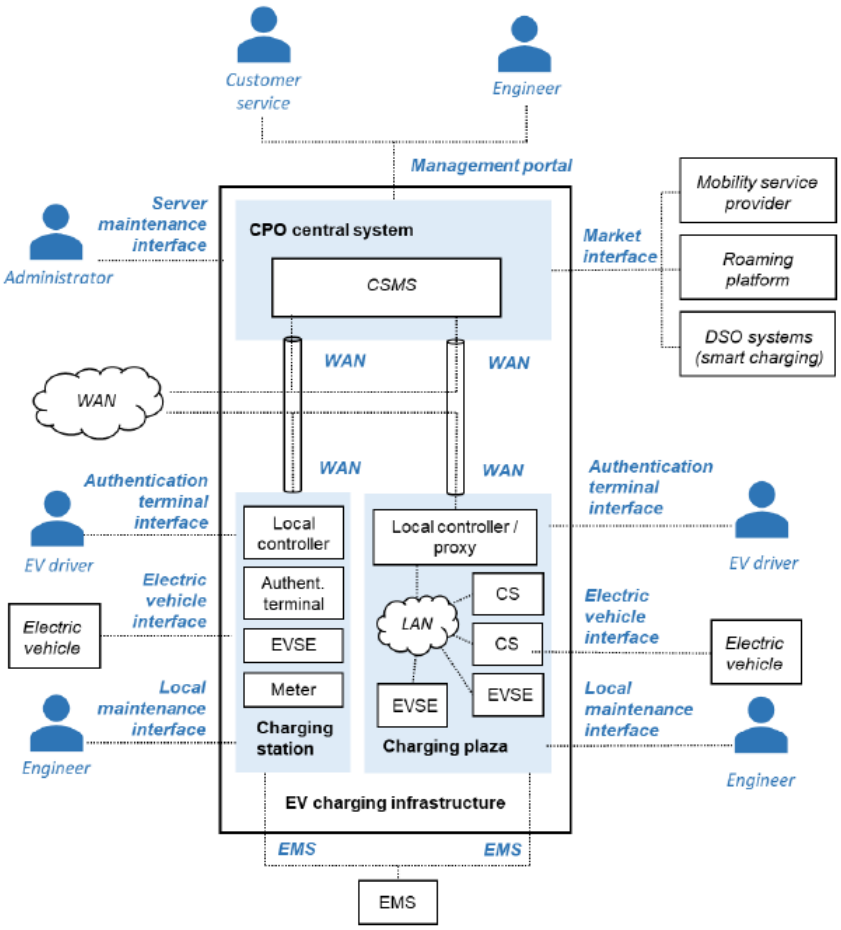

Figura 1.1: Infrastruttura di riferimento

1.2 Cyber-security dell'infrastruttura

Essendo l'infrastruttura di ricarica elettrica connessa e prevedendo interazioni sia locali che remote, essa può essere considerata un sistema IoT. Perciò, diventa essenziale la questione riguardante la sicurezza a partire dalle singole colonnine fino a coprire tutte le interazioni presenti. La sicurezza come requisito fondamentale diventa ancor più chiara se si pensa a come queste infrastrutture cresceranno nel tempo e saranno di vitale importanza per la mobilità del futuro.
Inoltre, si parla di applicazioni direttamente collegate alla rete elettrica, e quindi falle nella sicurezza che potrebbero aprire a scenari in cui i danni causati si propagano alla stessa rete elettrica e di conseguenza a tutte le applicazioni e/o infrastrutture dipendenti da essa.
Per quanto visto nella sezione precedente, sono molte le entità che entrano in gioco nell'infrastruttura di ricarica elettrica, dunque è evidente come un'analisi completa di cyber-security debba tener conto di molte variabili e concentrarsi su svariate tematiche. Si pensi, ad esempio, come anche un'infrastruttura sicura possa venir compromessa da vulnerabilità esterne (ad es. roaming).
Nel lavoro di tesi, il focus principale è stato rivolto alla colonnina e alle interazioni con il sistema centrale con uno studio approfondito del protocollo di riferimento (Open Charge Point Protocol (OCPP)).
In particolare, nel capitolo 2 viene presentato lo stato dell'arte descrivendo, per ogni componente dell'infrastruttura, la sua composizione ed i principali protocolli usati nei specifici scenari. Successivamente viene presentato un elenco delle vulnerabilità note riguardanti l'infrastruttura ed infine vengono analizzati alcuni documenti in cui vengono definiti i requisiti di sicurezza.
Nel capitolo 3 si analizza la versione 1.6 di OCPP definendo attacchi ed esaminando possibili rischi, in seguito vengono presentati i risultati dei test effettuati su un ambiente di simulazione e per finire viene studiato un documento che propone come usare le novità relative agli aspetti di sicurezza introdotte nella versione 2.0 di OCPP in sovrapposizione alla versione 1.6.
Nel capitolo 4 si analizza la versione 2.0 di OCPP sulla falsa di riga di quanto fatto nel capitolo 3. Nel capitolo 5 si ipotizza la possibilità di impersonificare una colonnina con tutti i rischi associati. Nel capitolo 6 vengono proposte alcune tematiche riguardanti le altre entità che entrano in gioco nell'infrastruttura le quali sono emerse durante il lavoro di tesi ma non sono state approfondite. Infine vengono presentate le conclusioni.

1.3 Gruppo SIGLA S.r.l.

Gruppo SIGLA Srl è un'azienda privata di Genova, composta da circa 80 tecnici con elevata specializzazione ed esperienza in ambito ICT, che progetta e sviluppa soluzioni informatiche per aziende ed enti pubblici: presta la propria collaborazione essenzialmente per Grandi Clienti con sedi a Genova, nel resto d'Italia ed all'estero. Nel mese di giugno 2021 la società entra a far parte del Gruppo Relatech, azienda presente sul mercato con soluzioni innovative dedicate alla digital transformation delle imprese e quotata su AIM di Borsa Italiana dal giugno 2019.
Le competenze coprono le diverse fasi di un progetto ICT, dall'analisi alla progettazione, dalla realizzazione alla messa in esercizio, nonchè la consulenza, la formazione e l'assistenza, e sono suddivise in 10 aree applicative differenti: Automazione Industriale, Analisi Dati, CRM, Formazione, IT Service Management, Monitoraggio Ambientale, Servizi IT, Sicurezza Informatica e Sviluppo Software.
L'azienda lavora inoltre con i principali Dipartimenti Universitari e Centri di Ricerca per la progettazione e realizzazione di sistemi ad alto contenuto innovativo. Sono, altresì, sponsor dell'Associazione DataScienceSeed, ed è presente un Laboratorio Congiunto sulla Cybersecurity CYBERTOOTH (cybertooth.grupposigla.it) con CIPI - Centro Interuniversitario di Ricerca sull'Ingegneria delle Piattaforme Informatiche, i cui aderenti sono le Università di Genova, Padova e Sassari.
Questa tesi si colloca nell'alveo del laboratorio CYBERTOOTH e ricopre un interesse fondamentale per Gruppo SIGLA e per le tematiche ricoperte dal laboratorio congiunto tra Gruppo SIGLA e CIPI.

Capitolo 2

Stato dell'arte

In questo capitolo viene proposta una visione d'insieme sullo stato dell'arte del contesto preso in analisi includendo: una descrizione più dettagliata delle varie componenti dell'infrastruttura e un breve sguardo sui principali protocolli utilizzati, le possibili vulnerabilità sia fisiche sia nelle comunicazioni fra le varie entità ed infine un riassunto di due documenti[4][5], stilati da ENCS (European Network for Cyber Security), i quali descrivono i requisiti che un'infrastruttura di ricarica elettrica dovrebbe avere.

2.1 Componenti e protocolli

2.1.1 Colonnina

La colonnina è di solito composta da più EVSE (Electric Vehicle Supply Equipment, contatori) i quali a loro volta possono avere diversi connettori permettendo la ricarica simultanea di più veicoli.
Solitamente le colonnine sono dotate di un lettore RFID[1] e/o un tastierino numerico al fine di permettere all'utente di autenticarsi e conseguentemente autorizzare la ricarica. Nelle versioni più recenti può essere presente un display che fornisce informazioni sulle operazioni in corso.
Internamente hanno un firmware dove viene specificato anche il protocollo con cui avvengono le comunicazioni con il sistema centrale. Il protocollo usato nella maggior parte dei casi è Open Charge Point Protocol (OCPP) versione 1.6 che permette sia

[1]L'identificazione a radiofrequenza è una tecnologia di riconoscimento e validazione e/o memorizzazione automatica di informazioni a distanza, si basa sulla memorizzazione di dati in particolari dispositivi elettronici passivi, capaci di rispondere a chiamate di prossimità da parte di dispositivi attivi, sia fissi che portatili, chiamati reader o lettori.

l'implementazione via SOAP[2], sia via JSON[3] su WebSocket[4]. Si tende a preferire l'ultima anche perché la nuova versione di OCPP (2.0.1) prevede solo JSON.
La connessione alla rete da parte della colonnina può avvenire tramite cavo ethernet oppure tramite SIM permettendo così la connessione anche se la colonnina è in un posto non raggiungibile da un cavo fisico.

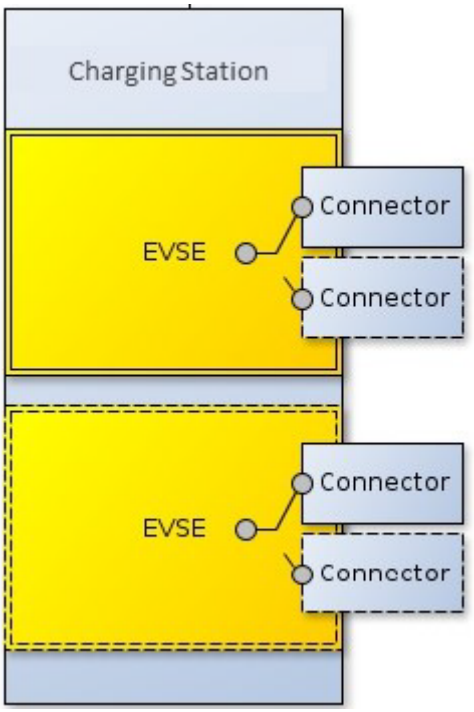

Figura 2.1: Esempio componenti di una colonnina

2.1.2 Colonnina - Sistema centrale

Solitamente i fornitori inseriscono nella colonnina una configurazione iniziale dove è presente l'indirizzo IP del server al quale connettersi (può esserci anche il DNS, dipende sempre dal fornitore).
Inoltre le colonnine possono essere dotate di un indirizzo IP pubblico oppure essere all'interno di una VPN (non raggiungibile dall'esterno).

[2]SOAP è un protocollo per lo scambio di messaggi tra componenti software.
[3]JavaScript Object Notation è un formato adatto all'interscambio di dati fra applicazioni client/server.
[4]WebSocket è una tecnologia web che fornisce canali di comunicazione full-duplex attraverso una singola connessione TCP.

OCPP

Il protocollo standard Open Charge Point Protocol (OCPP) è usato per le comunicazioni tra colonnina e sistema centrale. Ad ora, la versione è la 2.0.1 ma la più usata resta la 1.6, inoltre esiste una certificazione che riguarda quest'ultima versione rilasciata da chi mantiene il protocollo.
Il protocollo definisce procedure con conseguenti scambi di messaggi per: la sicurezza (autenticazione colonnina-sistema centrale), le autorizzazioni, le transazioni, il controllo remoto, visualizzare la disponibilità delle colonnine e riservare alcune di esse. Si occupa anche dell'aggiornamento del firmware e della diagnostica.
Le sessioni di ricarica si dividono in due categorie:

1. Fisica: l'utente effettua un'azione fisica sulla colonnina (carta RFID o PIN). Si possono avere due procedimenti: prima si collega il cavo al veicolo e poi ci si autentica (*Authorize* in OCPP) oppure il contrario, ciò dipende da come è stato implementato il firmware da parte del fornitore. Nel momento in cui ci si autentica la colonnina effettua una chiamata al server il quale controlla se l'utente è abilitato alla ricarica e risponde dando il via alla sessione (*StartTransaction*).

2. Remota: il gestore mette a disposizione un'applicazione nel quale l'utente esegue il login, a questo punto il sistema centrale fa partire una transazione remota (*RemoteStartTransaction*) sulla colonnina di interesse. Poi, se la colonnina accetta la transazione, si ha una sessione di ricarica.

Alcuni esempi di procedure previste dal protocollo sono:

- *BootNotification*, avviene nel momento in cui la colonnina viene accesa oppure dopo un lasso di tempo di inattività, nel messaggio la colonnina manda informazioni (versione del firmware, modello della colonnina, ecc.) di vario tipo al sistema centrale che risponde con parametri utili alla sincronizzazione.

- *ChangeAvailability*, permette al sistema centrale di richiedere il cambio di disponibilità di un certo connettore, la colonnina risponde con uno stato che indica se è in grado di portare a termine la richiesta.

- *SendLocalList*, permette l'invio di una lista di ID utili alla colonnina nel momento in cui la connessione con il sistema centrale non è disponibile, potendo autorizzare localmente un veicolo e iniziare una transazione.

- *SetChargingProfile*, permette l'inserimento di profili di ricarica (*ChargingProfile*), i quali gestiscono l'erogazione di energia sulle colonnine.

- *UpdateFirmware*, permette al produttore della colonnina di mettere a disposizione un aggiornamento del firmware. In questo messaggio viene fornito un indirizzo FTP da cui la colonnina può scaricare l'aggiornamento. Durante tutto il processo la colonnina è tenuta ad informare il sistema centrale sui vari passi.

Topologie

La comunicazione colonnina-sistema centrale è di solito diretta ma possono esserci anche degli intermediari (proxy/controller), si differenziano così varie topologie. Di seguito sono elencate quelle differenti ad una connessione diretta.

Multiple colonnine connesse al sistema centrale via Local Proxy In alcune situazioni è preferibile che tutte le comunicazioni di un gruppo di colonnine siano redirette attraverso un singolo nodo di rete (modem, router, ecc.). Quest'ultimo si connette alla rete e funge da proxy tra il sistema centrale e le colonnine, agisce come un router di messaggi. Per le colonnine il local proxy è il sistema centrale, simmetricamente per il sistema centrale il local proxy sono le colonnine.

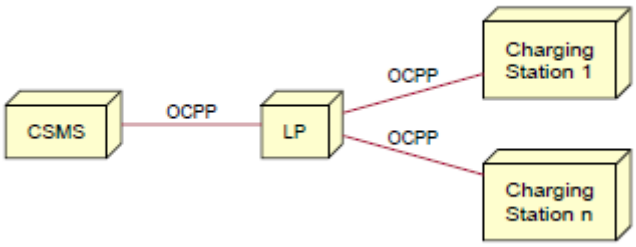

Figura 2.2: Topologia con Local Proxy

Multiple colonnine connesse al sistema centrale via Local Controller Simile al local proxy ma con funzionalità in più, non agisce solo da router ma può mandare dei messaggi alle colonnine indipendentemente dal sistema centrale.

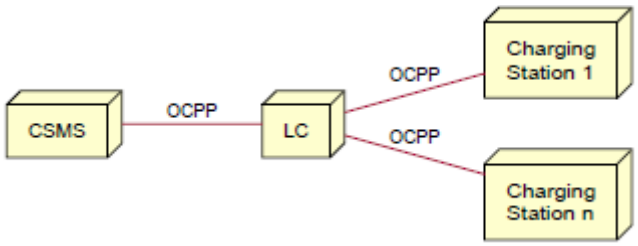

Figura 2.3: Topologia con Local Controller

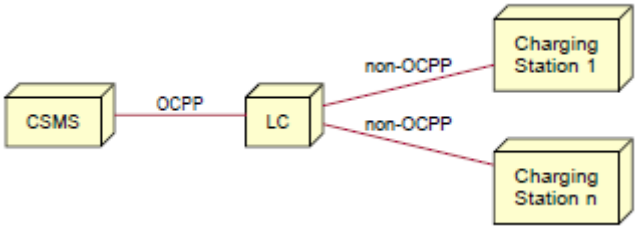

Figura 2.4: Topologia con Local Controller (variante)

Multiple colonnine non-OCPP connesse al sistema centrale via Local Controller Variante della topologia sopra dove la comunicazione tra LC e colonnine non avviene tramite OCPP.

Proxy lato server Nel caso in cui il fornitore non abbia un unico server ma più di uno per far fronte ad un elevato numero di richieste, è possibile avere una topologia in cui la colonnina comunica direttamente con un proxy il quale funge da Load Balancer indirizzando la richiesta ad un determinato server.

2.1.3 Colonnina - Veicolo elettrico

Per il collegamento tra colonnina e veicolo elettrico esiste ISO 15118 il quale è uno standard che definisce un protocollo tra le due parti. Implementa il cosiddetto Plug&Charge ovvero l'unica azione richiesta al guidatore è connettere il veicolo alla colonnina tramite un cavo, il che è considerato un fattore molto importante per contribuire all'adozione dei veicoli elettrici in quanto migliora l'esperienza utente. L'autenticazione è eseguita in automatico tra il veicolo elettrico e la colonnina. Questo standard abilita anche la ricarica bi-direzionale ovvero anche i veicoli elettrici possono ritornare energia alla rete. È importante notare come l'ultima versione di OCPP supporti l'utilizzo di ISO 15118.
Il protocollo sopra descritto è ancora poco utilizzato, solitamente l'autorizzazione viene delegata all'utente.

2.1.4 Roaming

In Figura 1.1 si vede come il sistema centrale instauri dei canali di comunicazione anche con altre entità diverse dalle colonnine. In particolare, un aspetto sempre più importante è il roaming, dati i molti provider che offrono il sevizio di ricarica. Infatti un utente può avere la necessità di utilizzare colonnine non appartenenti al fornitore con cui ha un contratto (card, PIN o account su app). A tale proposito, l'utente dovrebbe siglare un nuovo contratto e chiaramente ciò non lo facilità.
Per ovviare, al problema sopra descritto, viene utilizzato il roaming che permette la comunicazione tra più gestori di infrastrutture, in questo scenario emergono due ruoli principali:

- **CPO (Charge Point Operator)**: identifica il ruolo di gestore delle colonnine che permette agli utenti di ricaricare i propri veicoli.
- **eMSP (e-Mobility Service Provider)**, identifica il ruolo di gestore di utenti e l'obiettivo è quello di farli ricaricare su colonnine non appartenenti a lui.

I due ruoli non sono esclusivi, esistono diverse topologie in cui possono esserci piattaforme che operano esclusivamente come CPO o eMSP oppure piattaforme che possono assumere tutti e due i ruoli.
Il protocollo di riferimento è OCPI (Open Charge Point Interface) il quale prevede una connessione diretta fra i server dei gestori con scambio di specifici messaggi.
In alternativa esistono Hubject e Gireve, i quali agiscono come hub/accentratori tra i gestori definendo un proprio protocollo e gestendo le fatturazioni.
Ultimamente l'approccio più usato risulta essere OCPI in quanto permette una connessione diretta senza dover passare per terze parti che gestiscono la comunicazione, anche se nelle ultime versioni del protocollo è prevista la connessione di piattaforme attraverso hub. In Figura 2.5 si vede una topologia generale in cui sono presenti diverse piattaforme (piattaforme solo CPO/eMSP oppure piattaforme che assumono tutti e due i ruoli) dove le comunicazioni avvengono sia direttamente che tramite hub.

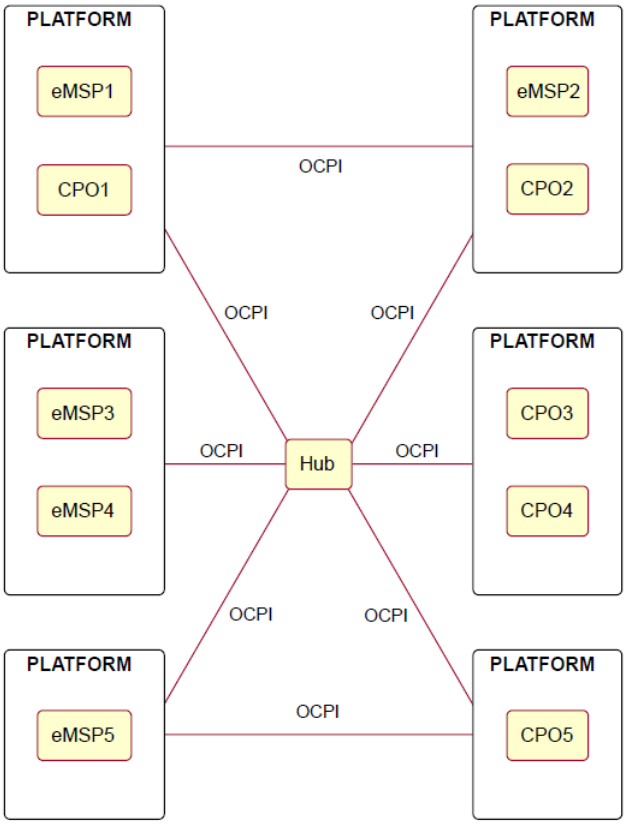

Figura 2.5: Topologia generale OCPI

OCPI

Attualmente la versione di riferimento è la 2.2.1, il protocollo supporta: l'autorizzazione, lo scambio di informazioni tra colonnine, lo scambio di record riguardanti le sessioni di ricarica, comandi remoti alle colonnine e scambio di informazioni riguardanti lo smart-charging tra le parti.
Le principali funzionalità che il protocollo offre sono:

- Un buon sistema di roaming (per un uso bilaterale oppure tramite un hub).
- Informazioni in tempo reale su posizione, disponibilità e prezzo.
- Una procedura uniforme per lo scambio di dati, prima, durante e dopo la transazione.

Il protocollo è basato su HTTP e si appoggia al formato JSON, le comunicazioni sono protette a livello di trasporto usando TLS e autenticazione token-based. Non viene richiesto il certificato lato client ma solo lato server.
Nella specifica sono presenti diversi moduli dove vengono descritti i tipi di messaggio ed il procedimento con cui vengono scambiati tra le parti, alcuni esempi sono:

- **Credentials module**: viene usato per lo scambio di token credenziali che devono essere usati dalle parti per l'autorizzazione di richieste. Viene descritto il processo di registrazione in cui le due parti si scambiano i rispettivi token, i quali devono essere presenti in ogni richiesta. In Figura 2.6 si vede un esempio di oggetto credentials composto dal token, da un url e dal ruolo della parte.


```
{
    "token": "ebf3b399-779f-4497-9b9d-ac6ad3cc44d2",
    "url": "https://example.com/ocpi/versions/",
    "roles": [{
        "role": "CPO",
        "party_id": "EXA",
        "country_code": "NL",
        "business_details": {
            "name": "Example Operator"
        }
    }]
}
```


Figura 2.6: Esempio di oggetto Credentials

- **Charge Detail Record**: è la descrizione di una sessione di ricarica appena conclusa e l'unico oggetto su cui fare riferimento per la fatturazione. Viene mandato dal CPO al eMSP, è composto di molti elementi come **cdr_location** (posizione della ricarica), **charging_periods** (lista di periodi che compongono una sessione di ricarica), **total_cost** (somma di tutti i costi della transazione nella valuta specificata) e molti altri.

- **Commands module**: abilita l'invio di comandi remoti ad una specifica colonnina, i comandi supportati sono: RESERVE_NOW, CANCEL_RESERVATION, START_SESSION, STOP_SESSION, UNLOCK_CONNECTOR.

2.1.5 DSO e EMS

Distribution System Operator (DSO) e Energy Management System (EMS) sono le due entità che implementano lo smart charging, lo fanno imponendo dei limiti sulla potenza/corrente che una od una serie di colonnine può utilizzare, oppure notificando al CS una previsione sulla disponibilità della rete elettrica che permette al sistema centrale di adattare le sessioni tramite dei profili di ricarica.
DSO fanno riferimento alla rete elettrica e comunicano direttamente con il sistema

centrale, EMS invece opera in uno scenario locale e la comunicazione potrebbe avvenire direttamente con la colonnina. Per capirne un po' meglio il funzionamento e la differenza tra le due parti vengono riportati due esempi in cui DSO e EMS operano:

- DSO effettua delle misurazioni sulla stazione elettrica e calcola la capacità disponibile di una certa zona e poi lo notifica al sistema centrale.

- Si consideri uno scenario in cui alcune colonnine sono posizionate all'interno di un edificio il quale ha dei pannelli solari. EMS può calcolare una capacità disponibile che ottimizza l'uso dell'energia locale rinnovabile.

Esiste un protocollo che gestisce questi tipi di comunicazione: Open Smart Charging Protocol (OSCP) il quale fu originariamente designato al fine di dare una previsione (24 ore) della capacità locale disponibile agli operatori delle colonnine, consentendo loro di adattare i profili di ricarica dei veicoli elettrici entro i limiti della capacità disponibile. Con la nuova versione 2.0 si cerca di applicare i messaggi in termini più generici per non limitare le possibilità del protocollo ai soli veicoli elettrici.

2.2 Vulnerabilità infrastruttura

2.2.1 Vulnerabilità interne

Analisi delle vulnerabilità e penetration testing hanno portato a scoprire diverse falle sia nel software che nell'hardware delle colonnine come: processori in esecuzione sotto un kernel Linux-based con password e algoritmi di hashing deboli, controllo degli accessi debole con processi non necessari che riescono a garantire l'escalation di privilegi ed un processo di aggiornamento del firmware non firmato.
Inoltre il protocollo OCPP permette alle colonnine di autenticare loro stesse i veicoli elettrici quando perdono la connessione con il sistema centrale, questo porta ad avere un processo di autenticazione e un database locale. Quindi, un attaccante che riesce a prendere il controllo della colonnina, acquisisce informazioni sensibili riguardanti gli utenti che si sono caricati precedentemente. Un caso di studi viene portato dalle colonnine prodotte da Schneider Electric, le quali avevano vulnerabilità causate da credenziali hard-coded, remote code injections e SQL injections[13]. Tali vulnerabilità possono portare un attaccante ad ottenere il controllo non autorizzato e scatenare attacchi sulle colonnine quali: manomissione dei dati, divulgazione di informazioni sensibili e DoS.

2.2.2 Vulnerabilità sul posto

Le colonnine di solito hanno delle porte USB montante all'esterno, ciò facilità le intrusioni fisiche. Infatti, sono punti di accesso convenienti rendendo possibile: copiare la configurazione corrente di una colonnina, modificare o cancellare dati memorizzati all'interno, accedere alle credenziali di autenticazione e identificare i veicoli che sono stati caricati precedentemente. Inoltre, un attaccante può modificare i dati copiati e reinserirli spacciandoli come un update del sistema.
Attraverso i lettori di carte RFID oppure di QR code, è possibile montare un attacco di phishing e riuscire ad ottenere le credenziali di login, contenute nella RFID card. A questo punto si possono duplicare ed ottenere ricariche a spese della vittima, solitamente i pagamenti avvengono mensilmente quindi ricariche non autorizzate possono essere ignote alla vittima per settimane.

2.2.3 Vulnerabilità Smartphone

Gli smartphone e le applicazioni web-based sono strumenti indispensabili per caricare i veicoli nelle colonnine commerciali, infatti vengono usati per: localizzarle, autenticare un veicolo, controllare da remoto la sessione di caricamento e infine per pagare la ricarica. Applicazioni malevole o bug all'interno di esse possono essere usati in qualità di punti di accesso all'infrastruttura di ricarica elettrica.

2.2.4 Vulnerabilità EMS

La comunicazione tra colonnine ed il sistema che gestisce la rete elettrica è affidata generalmente ad un protocollo proprietario. Il sistema inoltre gestisce le colonnine insieme ad altre applicazioni smart come ad esempio: TV, impianti di condizionamento, telecamere di sicurezza. È risaputo che questi tipi di dispositivi presentino molti problemi in ambito di sicurezza e possano quindi essere usati allo scopo di avere un punto di accesso a EMS e di conseguenza anche all'infrastruttura di ricarica elettrica.

2.2.5 Vulnerabilità aggiornamento del software

Sempre più frequentemente gli aggiornamenti vengono eseguite in modalità wireless anziché in modo fisico, questa nuova modalità è però vulnerabile ad attacchi MitM dove un attaccante può ottenere, negare o alterare gli aggiornamenti.

2.2.6 Vulnerabilità sistema centrale-colonnina

Le comunicazioni tra colonnina ed il sistema centrale non sono standardizzate globalmente, il protocollo usato nella maggior parte dei casi è OCPP. Attualmente non esistono ancora studi sulla sicurezza della versione 2.0.1, in ogni caso molte sono le vulnerabilità trovate nella versione precedente (1.6). In un paper scritto nel 2017[2] venivano esplorati possibili attacchi al protocollo OCPP, tra cui:

- attacco Man in the Middle al fine di riuscire ad intercettare i messaggi attraverso il canale di comunicazione mandando dei pacchetti non legittimi, causando:
  - rivelazione di informazioni come la versione del firmware di una colonnina e altre informazioni.
  - Distorsione delle informazioni come processi e configurazioni. Si possono ottenere furti di elettricità oppure un sovraccaricamento della rete in punti specifici che porta instabilità alla rete stessa.
  - Eliminazione o modifica di specifici comandi che porta ad un DoS.
- Redirezione del traffico della rete che causa un fallimento nella comunicazione o la perdita di alcune informazioni.

2.3 ENCS - Requisiti di sicurezza

La rapida crescita dei veicoli elettrici porta conseguentemente anche la crescita del numero di colonnine le quali gestiscono carichi elettrici sempre più grandi; è noto come queste infrastrutture siano vulnerabili a cyber-attacchi che hanno impatto anche sulla rete elettrica come, per esempio, agire sull'attivazione/disattivazione delle colonnine. Inoltre, se la ricarica intelligente viene usata, un attaccante può forzare la colonnina a usare più potenza del dovuto e danneggiare trasformatori e linee elettriche.
Alla luce di ciò, European Network for Cyber Security (ENCS) propone un modello di infrastruttura in grado di mitigare attacchi a larga scala allo scopo di compiere frodi o sabotare la rete elettrica stessa. Il riassunto seguente è il risultato di due documenti simili tra loro che riguardano un'architettura sicura per l'infrastruttura ed i requisiti per delle colonnine sicure.
Le misure si dividono in diverse sezioni, nei paragrafi successivi per ogni categoria vengono riassunti i requisiti definiti.

2.3.1 Access control

I requisiti qui incontrati riguardano il modo in cui i diritti di accesso sono gestiti e quanto forte debba essere l'autenticazione per i diversi gruppi di utenti.

- Il sistema centrale non deve permettere ad ingegneri e altre entità di agire sull'attivazione/disattivazione di molte colonnine allo stesso tempo.
- Le colonnine gestiscono i diritti di accesso in modo da permettere agli operatori di implementare il principio dei minimi privilegi, così facendo il sistema centrale, i guidatori di veicoli elettrici e gli stessi veicoli elettrici hanno accesso solo ai dati di cui hanno bisogno. Comportamento speculare tra CS e terze parti del mercato (DSO, piattaforme di roaming ecc.). Inoltre se la colonnina supporta la manutenzione locale, deve supportare il controllo degli accessi degli ingegneri con account locali e rinforzare i privilegi di accesso.
- Per il sistema centrale, altre colonnine e sistema di gestione elettrico viene usata l'autenticazione machine-to-machine. Lo stesso meccanismo viene usato anche nell'autenticazione tra CS e terze parti.
- Per gli ingegneri si usa un'autenticazione via password.
- Per i guidatori è necessario autenticarsi tramite l'interfaccia di autenticazione usando il meccanismo scelto dal fornitore del servizio. È importante scegliere meccanismi che siano resistenti ad attacchi già noti come il cloning.
- Se il veicolo elettrico lo supporta, la colonnina deve usare la mutua autenticazione con password o chiavi.

2.3.2 Criptography

La criptografia viene usata per molte funzioni come, per esempio, l'autenticazione machine-to-machine vista sopra.

- Tutte le componenti dell'infrastruttura devono usare chiavi e algoritmi forti resistenti ad attacchi alla stessa criptografia, i requisiti sono scritti in un apposito documento[3].
- Devono inoltre supportare il cambio di tutte le password e chiavi da remoto in modo sicuro, se la colonnina usa i certificati deve essere in grado di usare quelli emessi dalla PKI dell'operatore.

2.3.3 Physical and environmental security

La colonnina è protetta dalla manomissione dalla sua cover ed inoltre dovrebbe:

- rilevare manomissioni fisiche attraverso:
  - una cover che la protegga in maniera adeguata rendendo difficile ad un attaccante raggiungere specifici strumenti senza lasciare tracce visibili,
  - creare un evento di log se qualsiasi parte della cover viene aperta;
- permettere interventi fisici di manutenzione solo dopo che la cover viene aperta.

2.3.4 Operations security

L'infrastruttura deve supportare tutti i processi e le procedure che la rendono sicura in tutto il suo tempo di utilizzo.

- È necessario impostare un processo di backup automatico per i server del sistema centrale.
- La colonnina deve avere abbastanza memoria e potenza computazionale al fine di garantire che tutti gli update di sicurezza riescano ad essere eseguiti, inoltre il terminale di autenticazione dovrebbe essere facilmente sostituibile nel caso in cui nuovi metodi di autenticazione per i guidatori vengano introdotti in futuro.
- Si deve provvedere ad un metodo sicuro per il ripristino alle impostazioni di fabbrica della colonnina in caso di problemi.
- Per supportare il rilevamento ed una riposta ad incidenti di sicurezza, è necessario che la colonnina mantenga un log degli eventi riguardanti la sicurezza (ad es. aggiornamenti del firmware, certificati non validi, cambiamento dei parametri di sicurezza). Devono essere in grado di mettere a disposizione del sistema centrale i log sopra descritti ed inoltre devono essere protetti per evitare manomissioni degli stessi.
- Le colonnine devono permettere update remoti riguardanti le funzioni di sicurezza (ad es. update degli algoritmi di cifratura) ed inoltre devono verificare l'integrità e la sorgente di questi aggiornamenti usando la firma digitale.
- Per supportare una gestione efficace delle vulnerabilità la colonnina:

  - deve disabilitare funzioni che non servono: account di utenti inattivi, servizi di rete e porte esterne accessibili non utilizzate;
  - deve usare applicazioni, librerie e protocolli non affetti da vulnerabilità note;
  - deve applicare la validazione dell'input a tutti i dati che riceve;
  - se è dotata di MPU (Memory Protection Unit) o MMU (Memory Management Unit), deve marcare le regioni dove è presente il codice come read-only e le sezioni dove sono presenti i dati come non eseguibili;
  - se è dotata di MMU, usare ASLR (Address Space Layout Randomization).

2.3.5 Communication security

L'infrastruttura deve proteggere crittograficamente l'integrità e la riservatezza della comunicazione su reti non fidate (bloccando la comunicazione fra colonnine, restringendo l'accesso wireless alle colonnine).

- Misure crittografiche devono essere adottate per proteggere l'integrità e la confidenzialità delle comunicazioni allo scopo di verificare la sorgente dei messaggi e evitare attacchi replays. Può essere fatto attraverso l'applicazione del TLS o di una VPN o ancora attraverso misure applicative.
- Se gli ingegneri utilizzano un approccio wireless per gestire le colonnine è necessario che queste reti siano protette da password forti.
- Le colonnine devono essere resilienti contro attacchi DoS, è importante che non si verifichino crash o si riavviino diventando così inutilizzabili.

2.3.6 System acquisition, development and maintenance

Gli sviluppatori dovrebbero integrare la sicurezza durante tutto il processo di sviluppo:

- definendo norme di programmazione per assicurare la consegna di colonnine sicure;
- testando ogni release del firmware per assicurarsi che rispetti i requisiti;
- permettendo l'esecuzione di test anche a terze parti;

- fornendo linee guida su come configurare e operare in sicurezza sulle colonnine;
- producendo delle patch al fine di correggere le vulnerabilità trovate durante il periodo di utilizzo della colonnina.

2.3.7 Supplier relationships

Gli sviluppatori devono avere un Information Security Management System (ISMS) al fine di proteggere qualsiasi informazione (codice sorgente, progetti di sicurezza dettagliati, chiavi e credenziali specifiche del cliente) che può compromettere l'operatore dell'infrastruttura. Viene proposto ISO 27001.

2.3.8 Information security aspects of business continuity management

L'infrastruttura deve essere progettata per minimizzare l'impatto di un guasto nella sicurezza. Durante deve:

- non divulgare informazioni riservate, come chiavi o credenziali;
- proteggere l'integrità dei dati sensibili;
- non permettere che i controlli di accesso siano aggirati;
- ripristinare la disponibilità il più presto possibile.

2.3.9 OCPP

Infine viene suggerito l'uso del protocollo OCPP perché soddisfa molti dei requisiti elencati, questi ultimi vengono suddivisi in tre categorie in base alla copertura fornita: totalmente, parzialmente e non coperti dal protocollo.
Nella suddetta suddivisione, la versione del protocollo a cui si fa riferimento è la 2.0, quindi è possibile che la classificazione possa risultare leggermente differente se si prende la versione 1.6 (la quale è la più usata al momento).

Totalmente coperti

La funzionalità di registrare tutti gli eventi di sicurezza, la capacità di eseguire update da remoto e verificarne la firma sono funzionalità totalmente coperte dall'implementazione dell'OCPP.

Parzialmente coperti

- Il principio dei minimi privilegi per il sistema centrale.
- Autenticazione machine-to-machine con il sistema centrale grazie all'applicazioni del TLS.
- OCPP definisce algoritmi e lunghezza di chiavi per tutti i meccanismi criptografici in uso.
- Update da remoto di chiavi, password e certificati.
- La confidenzialità e l'integrità delle comunicazioni di rete sono protette dall'uso del TLS.

Non coperti

Ovviamente alcuni casi non sono coperti come tutta la parte di sicurezza fisica, l'autenticazione per i guidatori, la disabilitazione delle porte e dei processi non usati, la parte riguardante i test. Altri esempi sono: il reset dei dati alle impostazioni di fabbrica, la resilienza contro attacchi DoS, la protezione dei log.

Capitolo 3

Analisi OCPP 1.6

In questo capitolo si cerca di analizzare le possibili vulnerabilità ed i conseguenti attacchi alla versione 1.6 del protocollo, la seguente analisi si basa su un paper [2] scritto nel 2017 dove per ogni categoria vengono prese in esame le specifiche procedure e vengono descritti possibili attacchi manipolando i messaggi stessi.
Successivamente viene fatto un riassunto dei rischi principali ed una mappatura degli attacchi sulle entità coinvolte elencandone anche gli effetti; in seguito viene presentata una breve relazione riguardante i test eseguiti all'interno di un ambiente di simulazione al fine di testare l'applicabilità degli attacchi descritti.
Infine viene riportata una breve descrizione del documento che permette di usare alcuni miglioramenti di sicurezza introdotti nella versione 2.0 sopra 1.6.
Dal corrente capitolo in avanti si utilizza CP (Charge Point) come sinonimo di colonnina e CS (Central System) come sinonimo di sistema centrale.

3.1 Start-Up e Configurazione

Vengono analizzate le procedure eseguite all'avvio e per cambiare la configurazione della colonnina. Inoltre viene descritto un attacco al TLS, questo perché nella specifica viene suggerito l'uso del protocollo al fine di proteggere le comunicazioni e perciò, per poi essere in grado di manipolare i messaggi, bisogna avere un modo per rompere il TLS.

3.1.1 Vulnerabilità TLS

Nel paper viene descritto un attacco al TLS, il quale permette di impossessarsi della chiave di sessione usata per crittografare le comunicazioni. Così facendo si riesce

ad instaurare un MitM tra colonnina e sistema centrale, permettendo in questo modo la manipolazione dei messaggi e quindi la realizzazione di tutti gli attacchi che verranno descritti in seguito.
Tuttavia non viene specificato come il TLS sia applicato ovvero quali metodi vengano usati (metodo di Key Exchange, tipo di certificato ecc.). Come si vedrà nella sezione 3.6.2 se il TLS viene applicato con determinate procedure l'attacco descritto non può essere portato a termine.
Quello che viene delineato è un attacco nel handshake del protocollo, in particolare vengono operati i seguenti passi:

1. A intercetta il messaggio iniziale del CP diretto al CS contenente la sua chiave pubblica ($KPub_{CP}$) ed un nonce (N_i). Successivamente, A inoltra il messaggio sostituendo la chiave pubblica con la sua $KPub_A$.

2. Il sistema centrale riceve il messaggio da A, estrae i vari parametri e genera la chiave di sessione ($K_{session}$). Poi, CS risponde inserendo la chiave di sessione concatenata con il nonce in un messaggio firmato dalla sua chiave privata $KPriv_{CS}$, infine cifra il messaggio con la chiave pubblica di A ($KPub_A$) pensando sia quella del CP. Inoltre fornisce un certificato.

3. A questo punto A è in grado di recuperare $K_{session}$, successivamente inoltra il messaggio cifrandolo con $KPub_{CP}$.

4. Così facendo la colonnina accetta il messaggio (eseguendo controlli sul nonce e sul certificato) e recupera $K_{session}$.

Se tutto va a buon fine, A è in possesso di $K_{session}$ senza che ne CP ne CS se ne accorgano ed quindi in grado di vedere, intercettare e modificare le comunicazioni tra colonnina e sistema centrale.
D'ora in avanti si assume che l'attaccante sia in grado di visualizzare e modificare i messaggi scambiati nel canale di comunicazione CP $\leftrightarrow$ CS.

3.1.2 Boot notification

La procedura *BootNotification* (Figura 3.1) viene eseguita dopo l'accensione della colonnina.

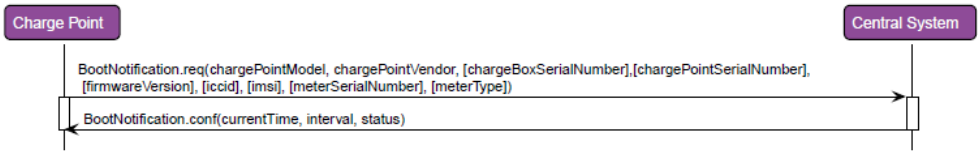

Figura 3.1: BootNotification

Il CP manda una richiesta al CS con informazioni riguardanti la sua configurazione. Un attaccante può bloccare la risposta del sistema centrale e modificarne i tre parametri presenti causando i seguenti scenari.

- **currentTime**: si può variare causando la desincronizzazione o riuscendo a ritardare le comunicazioni.
- **status**: si può modificare in rejected causando la chiusura del canale di comunicazione tra il CP ed il CS fino a che l'intervallo di tempo contenuto in **interval** non finisce.
- **intervals**: definisce l'intervallo di tempo tra un heartbeat (sezione 3.2.4) e l'altro, impostando un valore piccolo si possono verificare dei malfunzionamenti fino ad arrivare alla completa inutilizzabilità della colonnina.

3.1.3 Change Configuration

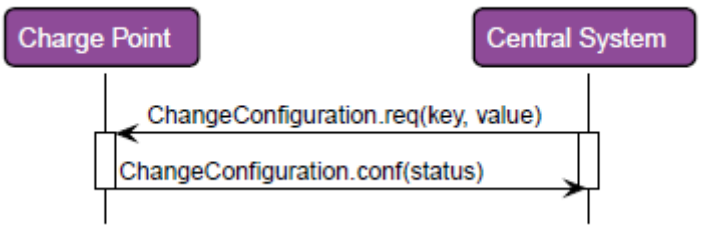

Figura 3.2: ChangeConfiguration

Attraverso il metodo raffigurato in Figura 3.2, è possibile modificare i parametri di configurazione nel CP attraverso la coppia key (corrisponde all'impostazione di configurazione) value (il valore che deve assumere).

Un esempio è la procedura *MeterValues* la quale fornisce informazioni al CS riguardo il contatore del CP. Attraverso la procedura vista sopra si possono modificare gli intervalli (key: **MeterValueSampleInterval**) nei quali queste informazioni vengono inviate al CS ed anche intervenire su quali tipi di dati debbano essere acquisiti e inviati (key: **MeterValuesSampledData**).

3.2 Transazioni e Controllo

3.2.1 Charging Profiles

I **ChargingProfiles** influenzano il processo di ricarica stabilendo, ad esempio, la quantità di potenza o corrente che viene fornita ad un CP in un dato intervallo. In Figura 3.3 è rappresentata la classe con tutti i suoi attributi, la parte più interessante è la **ChargingSchedules** la quale, tra le varie cose, definisce il periodo di validità ed il limite di potenza/corrente utilizzabile dal CP.

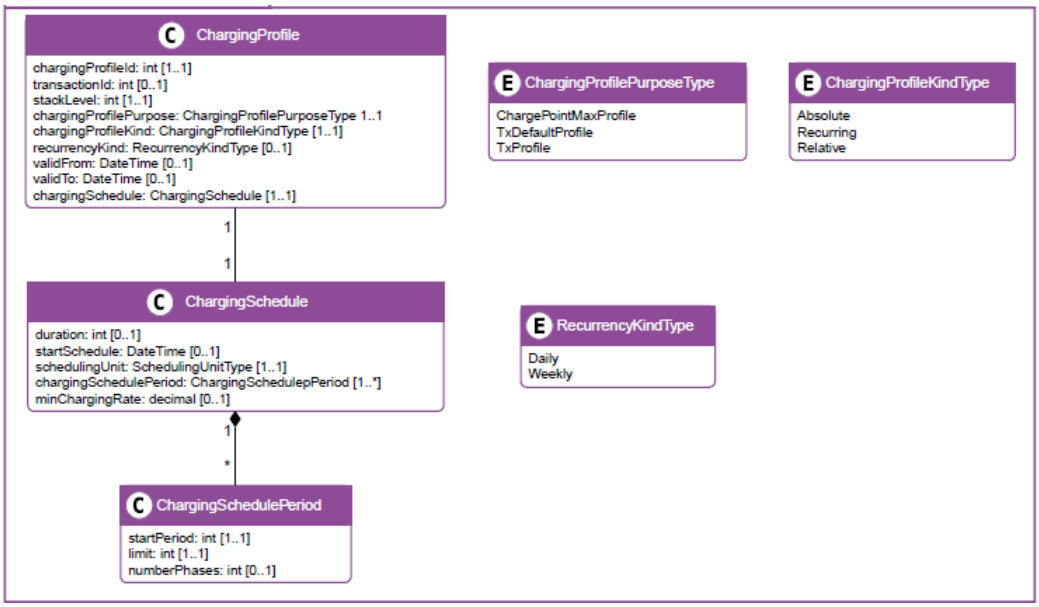

Figura 3.3: ChargingProfiles

In accordo con la specifica del protocollo i **ChargingProfiles** possono essere inseriti dal CS verso un CP:

- all'inizio di una transazione (*StartTransaction*);
- nella richiesta di *RemoteStartTransaction*;
- durante una transazione per cambiare il profilo attivo;

- fuori dal contesto delle transazioni usando la specifica funzione *SetChargingProfiles*.

Un attaccante potrebbe essere in grado quindi di iniettare falsi **ChargingProfiles**, per esempio, grazie a *RemoteStartTransaction* (Figura 3.4) che permette di definirli nella richiesta.

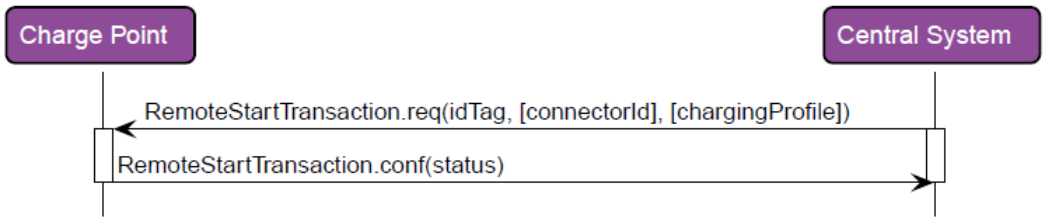

Figura 3.4: Esempio inserimento ChargingProfiles

In sostanza, agendo su tali profili è possibile variare il rate di ricarica portandolo ai massimi livelli per tutto il tempo o perturbare la schedules al fine di provocare instabilità nella rete elettrica.
Se sono collegati ad un specifico connettore (**connectorId**) possono essere aggiornati contro/per un specifico utente (ad es. cambiando il rate massimo al minimo).
Inoltre molti CP sono progettati per offrire il processo di carica bidirezionale, si può intervenire sui Charging Profiles forzando le colonnine ad immettere energia nei veicoli elettrici durante i periodi di picco quando invece le batterie dovrebbero ritornare energia alla rete. Per fare ciò è necessario che il consumo di potenza nei picchi sia maggiore od uguale a quello nei periodi non di picco. Anche questo causa instabilità nella rete elettrica.

3.2.2 Inoltro transazione: CP1 a CP2

Se un attaccante è in grado di intervenire su due canali di comunicazione quali CP1 $\leftrightarrow$ CS e CP2 $\leftrightarrow$ CS, allora si apre la possibilità di riuscire ad inoltrare una transazione autorizzata in CP1 a CP2 dove è presente l'attaccante che compie un furto di elettricità.
Per compiere ciò sono necessari i seguenti passi:

1. *A* intercetta la richiesta *Authorize* di CP1 diretta al CS. Così facendo *A* entra in possesso di un identificativo abilitato alla ricarica. D'ora in avanti le comunicazione nel canale CP1 $\leftrightarrow$ CS vengono bloccate e ignorate.

2. Successivamente, *A* invia una richiesta di *RemoteStartTransaction* a CP2 inserendo l'ID recuperato nel punto precedente.

3. A questo punto, l'attaccante non ha più bisogno di intervenire sulla comunicazione in quanto CP2 richiede l'inizio di una transazione (*StartTransaction*) con l'ID ricevuto nel punto precedente. Siccome l'identificativo è valido, il CS risponde positivamente e quindi si ha una sessione di ricarica a spese dell'utente associato all'identificativo sottratto nel punto 1.

Il flusso descritto nel punto 3 potrebbe variare in base al valore della chiave **AuthorizeRemoteTxRequests**, infatti se è settata a true impone al CP di dover autorizzare l'ID tramite *Authorize* prima di poter far partire una transazione. Comunque il risultato finale non cambia, si ottiene un furto di elettricità a danno dell'utente associato all'identificativo.

3.2.3 Reservation

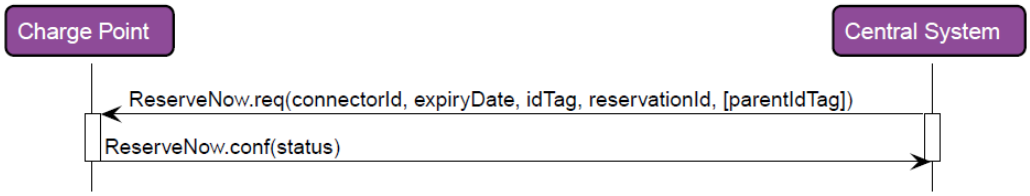

Figura 3.5: ReserveNow

Il CS può riservare un specifico connettore per essere usato da un specifico **idTag** (Figura 3.5). Quindi, il CP deve rifiutare tutte le ricariche dirette al connettore riservato a meno che **idTag** non coincida con quello presente nella richiesta.
Se la chiave di configurazione **ReserveConnectorZeroSupported** è settata a true, nella richiesta **connectorId** può essere 0, ciò vuol dire che il CP non riserva un specifico connettore ma durante tutto il periodo di validità della richiesta (**expiryDate**), deve tenere un connettore libero.
È possibile generare un Denial of Energy Service (DoES) generando queste richieste con un **expireDate** settato con un tempo molto lungo ed un **idTag** inesistente, ciò è possibile anche per il fatto che i CP non sono autorizzati a cancellare la prenotazione finché non si raggiunge **expireDate**.
È importante far notare come il CP sia tenuto ad informare il CS nel momento in cui un connettore cambia il proprio stato tramite la procedura *StatusNotification*,

quindi per un attaccante è necessario bloccare i messaggi di tale procedura allo scopo di non permettere al CS di reagire ad un cambio di stato sospetto.

3.2.4 Heartbeat

Per fa sapere al CS di essere ancora attivo, il CP deve mandare un heartbeat (Figura 3.6) dopo un intervallo configurabile, il CS risponde con il tempo corrente per permettere al CP di sincronizzarsi. Due minacce possono presentarsi:

- manomettere il tempo corrente (**currentTime**) in risposta per causare una desincronizzazione,
- oppure modificare la chiave **HeartbeatInterval** allo scopo di inserire un intervallo molto piccolo che causa malfunzionamenti o addirittura DoS alla colonnina.

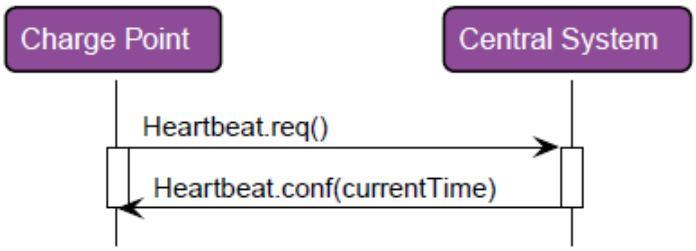

Figura 3.6: Heartbeat

3.2.5 Local Authorization

Nell'evento in cui la comunicazione con il CS non è possibile, il CP è designato per operare in autonomia, può quindi autorizzare localmente gli ID usando una Local Authorization List (LAL) oppure una Authorization Cache.
La chiave di configurazione **LocalAuthorizeOffline** determina se il CP è autorizzato ad operare offline, inoltre la chiave **LocalPreAuthorize** permette al CP di avviare una transazione senza aspettare l'autorizzazione dal CS.

Un attaccante può utilizzare la funzione *SendLocalList* (Figura 3.7) mandando una lista vuota che rimpiazza quella presente o nel caso di cache pulirla attraverso *ClearCache*. A questo punto si può forzare il CP ad uno stato offline agendo sul canale di comunicazione, così facendo si rende di fatto il CP inutilizzabile.
Un altro attacco può essere il seguente: un attaccante invia una LAL con falsi

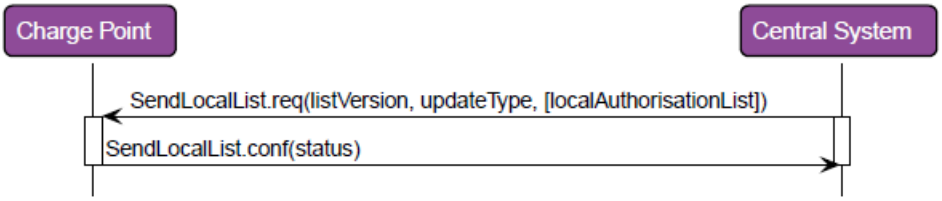

Figura 3.7: SendLocalList

ID, successivamente li presenta al CP che gli autorizza ottenendo così un furto di elettricità.

3.2.6 Change Availability

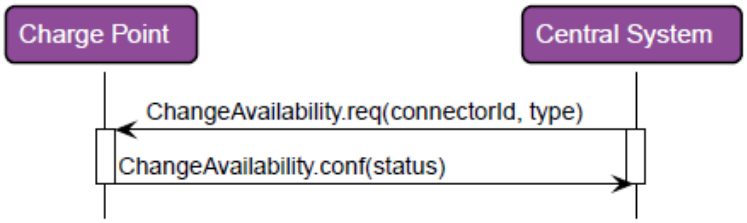

Figura 3.8: ChangeAvailability

Il CS può richiedere ad un CP di cambiare la sua disponibilità per un connettore (**connectorId** > 0) oppure per l'intero CP (**connectorId** $= 0$), ciò può essere usato da un attaccante al fine di ridurre le funzionalità del sistema, chiedendo sistematicamente ai CP di cambiare la propria disponibilità.
Anche qui vale la considerazione fatta in sezione 3.2.3 sulla necessità dell'attaccante di bloccare i messaggi che informano il CS di un cambio di stato del connettore.

3.2.7 Unlock Connector

Il CS può richiedere ad un CP di sbloccare un connettore al fine di aiutare gli utenti a risolvere problemi riguardanti il cavo e/o il connettore.
Un attaccante può attivare dei connettori con malfunzionamenti al fine di impattare sia sulla ricarica che sulla stabilità della rete.

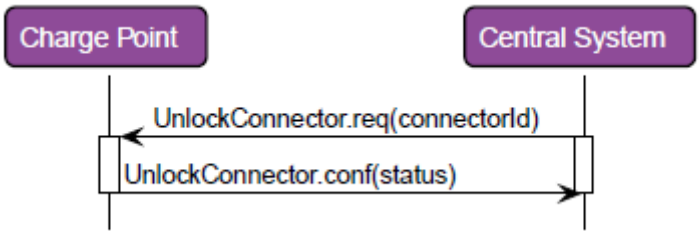

Figura 3.9: UnlockConnector

3.2.8 Reset

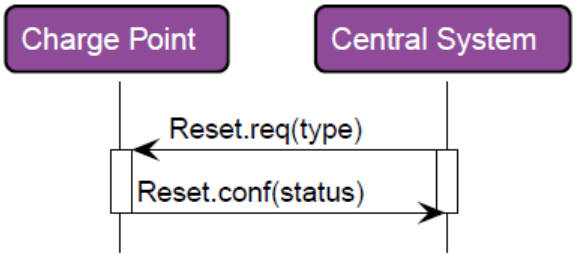

Figura 3.10: Reset

Il CS può richiedere al CP di effettuare un reset (Figura 3.10), due tipi disponibili: hard e soft. Nel caso di soft reset il CP deve interrompere le transazioni in corso "gentilmente" attraverso *StopTransaction*, nel caso di hard reset non è tenuto a farlo.
Ciò può essere sfruttato da un attaccante per minare la disponibilità e l'integrità delle risorse.

3.2.9 Data Transfer

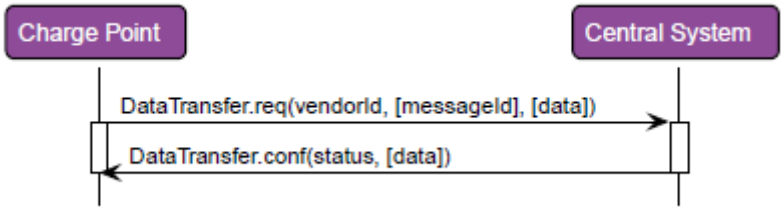

Figura 3.11: DataTransfer

Il CP può scambiare informazioni le quali non sono coperte dalle funzioni del protocollo, nella richiesta è presente un **vendorId** che definisce un identificativo

dell'implementazione, in risposta il CS risponde con uno stato (accepted, rejected, unknown). La stessa funzione esiste a parti invertite.
In questo scenario gli attaccanti possono sovraccaricare un CS/CP richiedendo funzioni non supportate con lunghezza non regolare, infatti la dimensione ed il formato del campo **data** non sono definiti.

3.3 Notifiche e Mantenimento

3.3.1 Update Firmware

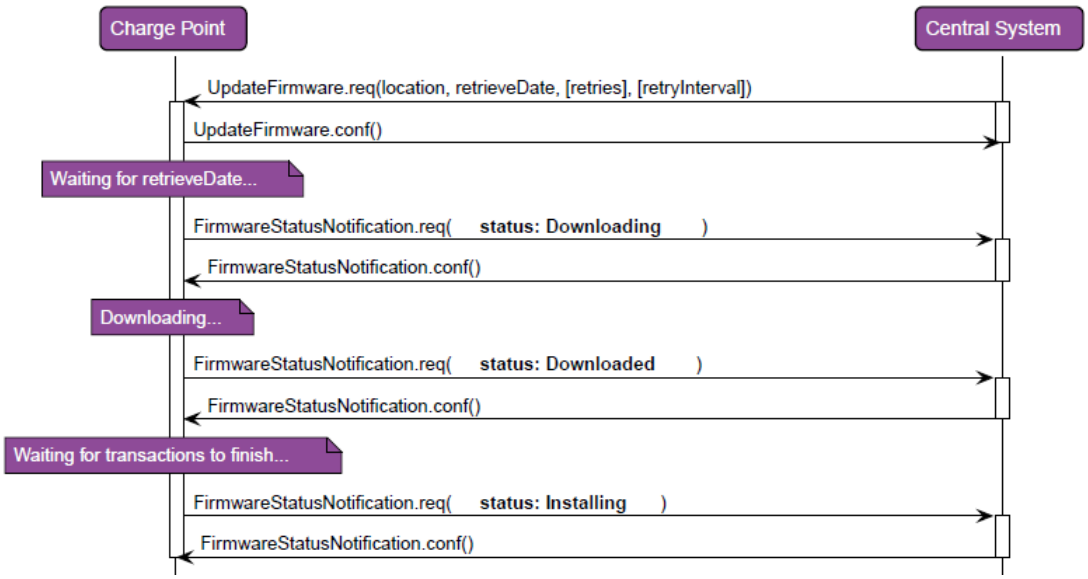

Figura 3.12: Processo di aggiornamento del firmware

Il CS può mettere a disposizione del CP un aggiornamento del firmware, fornendo **location** (URL) e una data e un orario (**retrieveDate**) dopo il quale il CP è autorizzato a scaricare l'aggiornamento.
Durante il download e l'installazione, il CP è tenuto ad informare il CS sullo stato tramite *FirmwareStatusNotification*. È importante notare come il CP non controlli la sorgente e nemmeno l'integrità dell'aggiornamento.
Teoricamente, un attaccante potrebbe fornire un URL malevolo dove, per esempio, è presente un malware, riuscendo così ad infettare la colonnina.
Inoltre in *UpdateFirmware* sono presenti due parametri opzionali: **retries** e **retryInterval** che definiscono il numero di tentativi nel provare ad eseguire il download e l'intervallo che intercorre tra ognuno di essi. Perciò agendo su questi parame-

tri (incrementando i tentativi e diminuendo l'intervallo) è possibile provocare dei malfunzionamenti fino ad arrivare a un DoS.

3.3.2 Diagnostic

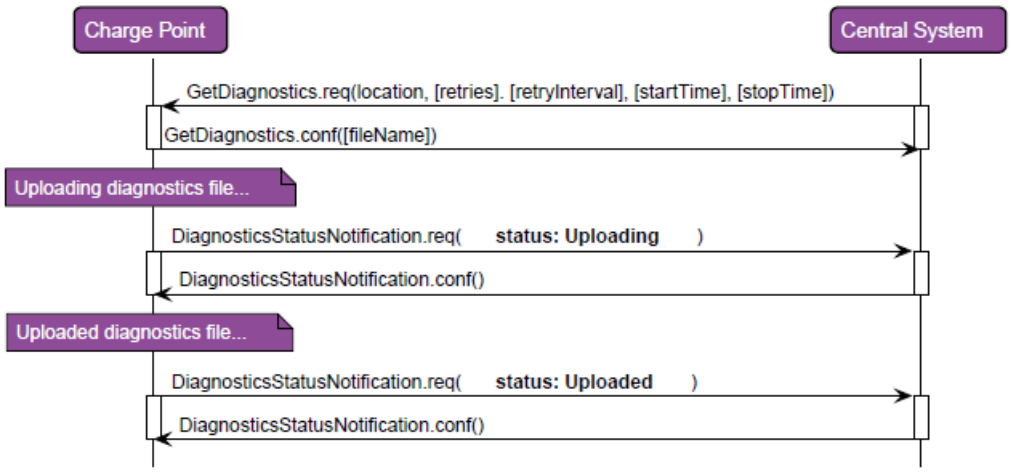

Figura 3.13: Processo di recupero file di log

Il CS può richiedere al CP informazioni riguardanti la diagnostica fornendo una **location** dove la colonnina dovrà caricare i dati.
Anche in questo caso gli attaccanti possono fornire falsi URL oppure agire sui parametri **retries** e **retryInterval** come visto nella sezione 3.3.1.

3.4 Riassunto rischi principali

Viene presentato un riassunto sui potenziali rischi principali a cui l'infrastruttura di ricarica elettrica è esposta se gli attacchi sopra descritti hanno successo.

3.4.1 Esposizione dati sensibili

- Informazioni riguardanti la configurazione del CP in *BootNotification*. (sezione 3.1.2)
- Identificativi di utenti in *Authorize*, *Start/StopTransaction*, *ReserveNow* e *SendLocalList*
- Campi **meterStart** in *StartTransaction* e **meterStop** in *StopTransaction*, se tutti e due sono visibili è possibile calcolare la potenza immessa nel veicolo e quindi poter costruire un profilo personale dell'utente.

3.4.2 Malfunzionamenti, DoS

- Impostando un intervallo di heartbeat piccolo. (sezioni 3.1.2 - 3.2.4)
- Riservando connettori/colonnine a falsi ID con tempi molto lunghi. (sezione 3.2.3)
- Cambiando la disponibilità dei connettori/colonnine. (sezione 3.2.6)
- Cancellando la cache o LAL e forzando il CP ad uno stato offline rendendolo così inoperativo. (sezione 3.2.5)
- Utilizzando la funzione *Reset* durante una transazione. (sezione 3.2.8)
- Incrementando **retries** e diminuendo **retryInterval** in *UpdateFirmware* e *GetDiagnostics*. (sezioni 3.3.1 - 3.3.2)
- Servendosi di *ChangeConfiguration* al fine di modificare il valore delle chiavi di configurazione. (sezione 3.1.3)
- Intervenendo nel rate di ricarica contenuto nel **ChargingProfiles** contro un specifico utente. (sezione 3.2.1)
- Utilizzando *DataTransfer* al fine di sovraccaricare il CP/CS. (sezione 3.2.9)

3.4.3 Frode/furto di elettricità

- Tramite la capacità di inoltrare una transazione da un CP ad un altro. (sezione 3.2.2)
- Servendosi degli identificativi esposti.
- Usando *SendLocalList* per immettere falsi ID nei CP.(sezione 3.2.5)
- Usando *ReserveNow* al fine di riservare un connettore/colonnina ad un falso ID. (sezione 3.2.3)

3.4.4 Instabilità rete elettrica

- Iniettando **ChargingProfiles** dannosi. (sezione 3.2.1)
- Agendo sui connettori con malfunzionamenti tramite **UnlockConnector**. (sezione 3.2.7)

3.5 Mappatura attacchi

Per ogni sezione contenente l'analisi attacchi usando una specifica procedura OCPP, viene effettuata una mappatura in cui si elencano i possibili effetti e le entità danneggiate.

- **3.1.1**: l'attacco al TLS descritto porta a ricavare la chiave di sessione con cui un attaccante può decifrare tutte le comunicazioni tra CP e CS. Questi ultimi sono anche le due entità coinvolte e che vengono danneggiate direttamente, naturalmente ciò si verifica se il TLS viene applicato.
- **3.1.2**: si è già visto come sveli all'attaccante informazioni riguardanti la configurazione del CP (ad es. modello della colonnina, versione del firmware). Inoltre modificando i parametri in risposta si possono causare desincronizzazioni e dei rallentamenti che provocano dei malfunzionamenti alla colonnina fino ad arrivare ad un DoS. L'entità coinvolta e danneggiata è il CP.
- **3.1.3**: permette di modificare il valore delle chiavi presenti nella configurazione del CP, nella specifica vengono elencate molte chiavi le quali possono essere optional o required e solo lettura (R) oppure lettura e scrittura (RW); è possibile quindi modificare solo le chiavi RW. Siccome le chiavi opzionali

potrebbero esserci come no (dipende dall'implementazione e quindi dal produttore), vengono riportati alcuni esempi di chiavi required e RW e possibili conseguenze derivanti da attacchi che coinvolgono la modifica delle stesse.

  - **HeartbeatInterval**: descrive il tempo di inattività dopo il quale il CP è obbligato a mandare un *Heartbeat* (sezione 3.2.4). Se questa chiave viene settata ad un valore molto piccolo, si potrebbero causare dei malfunzionamenti/rallentamenti alla colonnina fino ad arrivare ad una inutilizzabilità totale della stessa (DoS).
  - **LocalAuthorizeOffline**: se settata a true permette, nel caso in cui la connessione con il CS non sia disponibile, di autorizzare localmente gli utenti tramite una lista o una cache. Un attaccante quindi, può settare a false la chiave e agire sul canale di comunicazione tra CP e CS rendendolo non utilizzabile. Se ciò ha successo, la colonnina risulta praticamente inadoperabile.
  - **MeterValuesSampledData**: indica i valori misurati nel contatore da inserire nella funzione *MeterValues*, modificandone il valore è possibile influenzare la comunicazione inserendo dei valori non utili o fasulli al CS.

- **3.2.1**: Iniettando dei Charging Profiles è possibile provocare un'instabilità nella rete elettrica ed influenzare la sessione di ricarica. In questo scenario le entità coinvolte sono due: l'utente e la rete elettrica.

- **3.2.2**: permette di inoltrare una richiesta di transazione di un utente autorizzato (A) proveniente da una colonnina ad un'altra dove è presente un utente malevolo (B). Perciò, B può ricaricare a spese di A compiendo un furto ai danni dell'utente autorizzato.

- **3.2.3 - 3.2.6**: attraverso le due procedure descritte, è possibile rendere inutilizzabile una colonnina.

- **3.2.5**: può essere usata al fine di ottenere un furto di elettricità, stavolta a carico del provider che fornisce il servizio. Infatti attraverso *SendLocalList* è possibile inserire dei falsi ID nella colonnina, dove poi un utente malevolo in possesso di quell'identificativo può ottenere una ricarica.

- **3.2.9**: è possibile sovraccaricare il CP inviando continuamente richieste con un campo **data** con lunghezza non regolare. Può essere usato anche a parti invertite danneggiando il CS.

- **3.3.1**: la procedura non sicura di aggiornamento del firmware apre alla possibilità di infettare la colonnina con un malware.
- **3.3.2**: la possibilità di redirigere i file di log danneggia il CS in quanto potrebbe non essere informato su eventi critici riguardanti la colonnina.
- Nelle procedure degli ultimi due punti, sono presenti dei parametri opzionali (**retries** e **retryInterval**) i quali se modificati opportunamente possono provocare dei rallentamenti alla colonnina fino ad arrivare ad una completa inutilizzabilità della stessa.

Di seguito viene riportata una tabella che riassume i contenuti delle due sezioni precedenti ovvero i possibili rischi e le principali entità coinvolte in ogni tipo di attacco analizzato.

#	Scenario	Tipo attacco	Entità danneggiate	Effetti
1	3.1.1	MitM	CP - CS	Comunicazioni in chiaro
2	3.1.2	Intercettazione messaggio	CP - vendor	Esposizione dati sensibili
3	3.1.2	Modifica parametri in risposta	CP	Malfunzionamenti - DoS
4	3.1.3 - 3.2.4	Modifica valore chiave	CP	Malfunzionamenti - DoS
5	3.1.3 - 3.2.5	Modifica valore chiave	CP	DoS
6	3.1.3 - *MeterValues*	Modifica valore chiave	CS	Informazioni errate/inutili
7	3.2.1	Modifica parametri	Rete elettrica - Utente	Instabilità - Malfunzionamenti ricarica
8	3.2.2	Inoltro transazione	Utente autorizzato	Furto di elettricità
9	3.2.3 - 3.2.6	Modifica o creazione del messaggio	CP	DoS
10	3.2.5	Inserimento falsi ID	Provider	Furto di elettricità
11	3.3.1	Iniezione URL malevolo	CP	Infezione malware
12	3.3.2	Iniezione URL malevolo	CS	Redirezione file di log
13	3.3.1 - 3.3.2	Modifica parametri	CP	Malfunzionamenti - DoS

Tabella 3.1: Attacchi ed effetti

3.6 Test

Una volta analizzati i possibili attacchi, si è cercato di replicarli attraverso un ambiente di simulazione fornito da Gruppo SIGLA.

3.6.1 Descrizione ambiente di simulazione

L'ambiente di simulazione fornito da Gruppo SIGLA emula lo scambio di messaggi tra una colonnina ed il sistema centrale, è implementata la versione 1.6 di OCPP, in particolare messaggi in formato JSON scambiati attraverso il protocollo WebSocket. L'ambiente è composto da: un simulatore di colonnina ed un server.

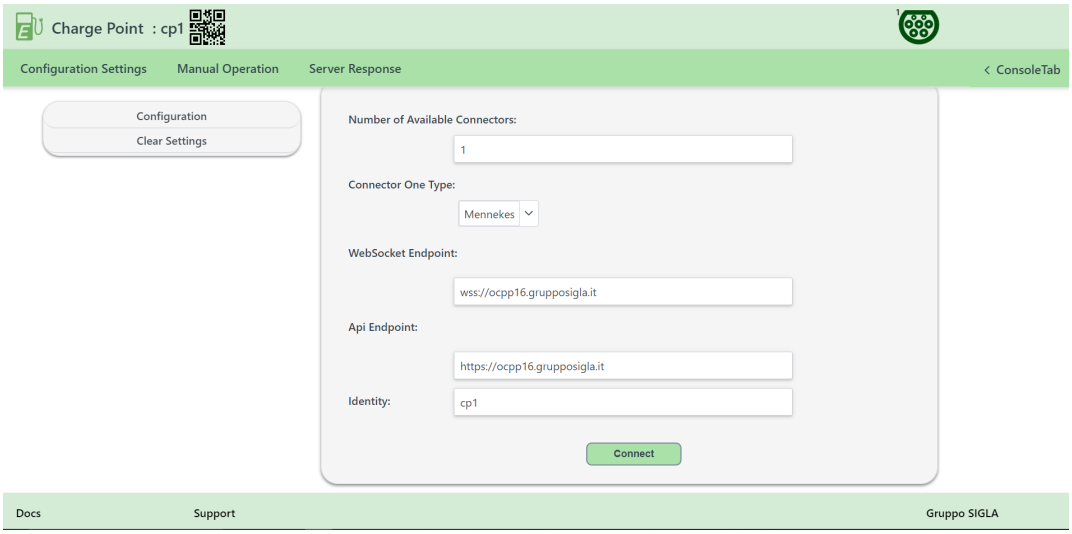

Figura 3.14: Simulatore di colonnine

Il simulatore è raggiungibile ad un specifico indirizzo e si presenta come mostrato in Figura 3.14. Inserendo l'endpoint WebSocket (l'indirizzo del server) e l'identità della colonnina è possibile inizializzare una connessione WebSocket con il server, successivamente è possibile creare ed inviare i messaggi OCPP (Figura 3.15) che si riferiscono alle procedure iniziate dalla colonnina. In risposta si riceve un messaggio di errore da parte del server (Figura 3.16), ciò accade in quanto l'ambiente simula esclusivamente lo scambio di messaggi e non è implementata la logica sottostante di elaborazione del messaggio e conseguente risposta.

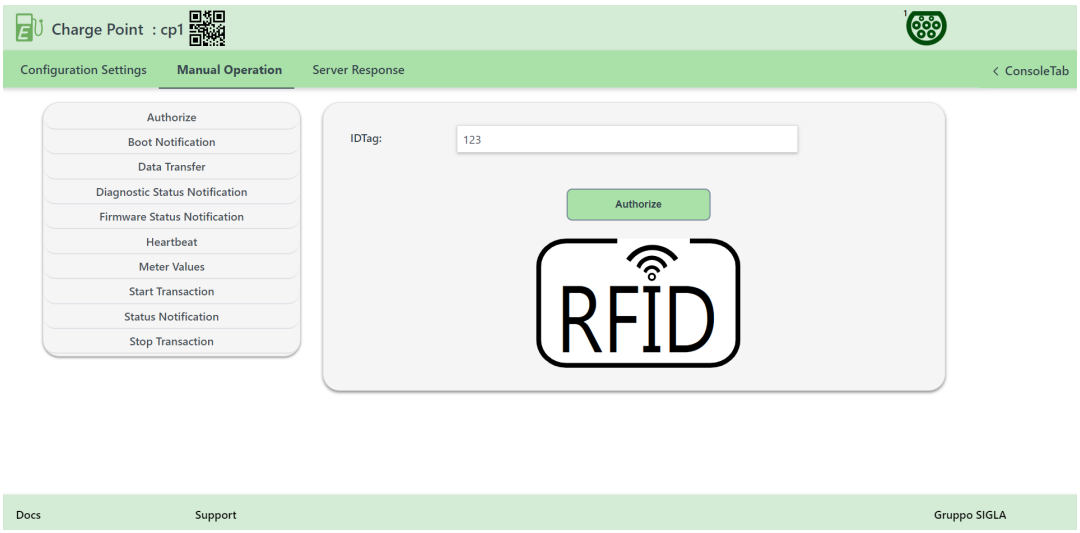

Figura 3.15: Creazione messaggio Authorize

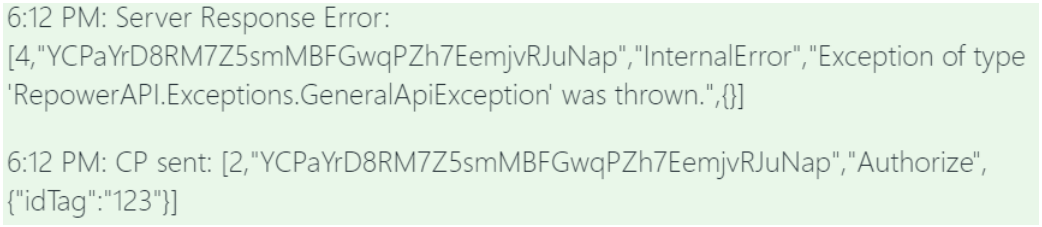

Figura 3.16: Invio messaggio e risposta dal server

Sono inoltre presenti le API per le procedure iniziate dal sistema centrale, sono messe a disposizione attraverso Swagger[1]. In Figura 3.17 viene mostrato il flusso di scambio messaggi: viene effettuata una chiamata HTTP alla specifica API, la quale produce un messaggio in formato JSON che viene mandato attraverso la connessione WebSocket precedentemente instaurata. Successivamente il simulatore di colonnina invia una risposta (qui il messaggio non è di errore ma nella maggior parte dei casi ne viene creato uno in cui si accetta la richiesta, anche qui nessuna logica è implementata) che viene ricevuta dal server.
In aggiunta è stata fornita una macchina situata all'interno della stessa sottorete del server con installata una versione di Kali Linux[2]. Ciò allo scopo di essere in grado di sniffare il traffico ed eseguire un MitM.

[1]Swagger è un linguaggio di descrizione dell'interfaccia per descrivere le API RESTful espresse utilizzando JSON.

[2]Kali Linux è una distribuzione GNU/Linux basata su Debian, pensata per l'informatica forense e la sicurezza informatica.

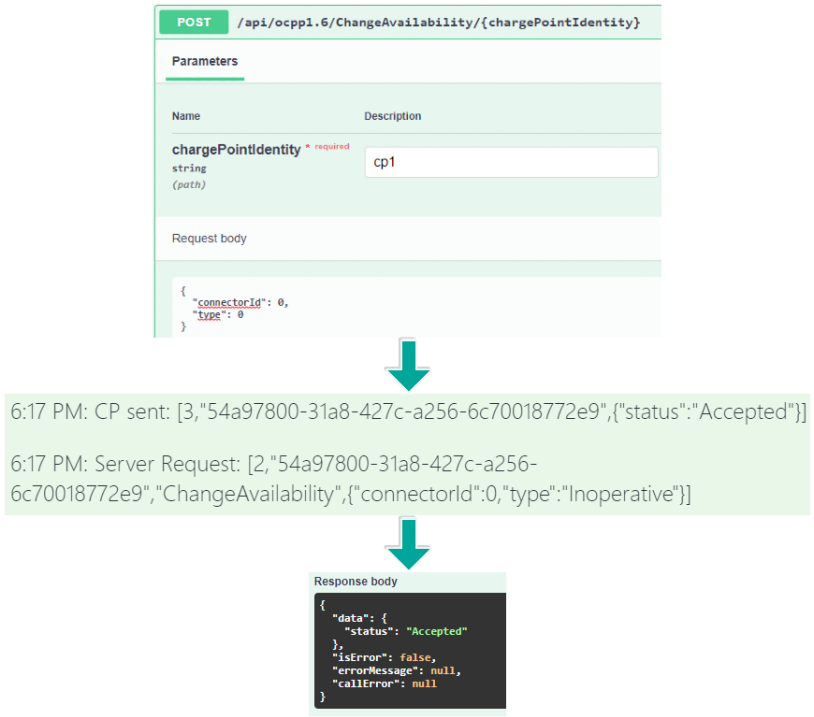

Figura 3.17: Flusso API sistema centrale → colonnina

3.6.2 Test attacco MitM

Si è cercato, innanzitutto, di portare a termine un MitM tra il server (sistema centrale) ed il client (simulatore di colonnina) al fine di riuscire a vedere le comunicazioni tra le due parti e successivamente manipolarle. L'attacco è stato eseguito dalla macchina Kali che è all'interno della stessa sottorete del server.
Sono stati utilizzati tre strumenti:

- **Ettercap**[3]: usato per eseguire un ARP poisoning tra il server ed il gateway allo scopo di far passare tutto il traffico per la macchina Kali. In sostanza, si avvelena la tabella ARP di tutte e due le parti con il risultato che il server pensa che la macchina Kali sia il gateway e simmetricamente per il gateway la macchina Kali è il server.

- **iptables**[4]: usato per definire delle regole le quali redirigono il traffico in entrata di specifiche porte.

- **mitmproxy**[5]: usato al fine di visualizzare il traffico tra server e gateway e

[3]Ettercap è un software il quale comprende una suite completa per attacchi man in the middle. Consente lo sniffing di connessioni dal vivo, il filtraggio dei contenuti al volo e molti altri metodi.
[4]iptables è un tool usato per impostare, gestire e ispezionare le tabelle delle regole di filtraggio dei pacchetti IPv4 nel kernel Linux.
[5]mitmproxy è un HTTPS proxy interattivo open source.

quindi anche del client.

Setup

Innanzitutto è necessario inserire le due regole che permettono di redirigere tutto il traffico in entrata dalle porte 80 e 443 sulla porta 8080.

```
$ iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 -j
 REDIRECT --to-port 8080
$ iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 443 -j
 REDIRECT --to-port 8080
```

A questo punto si esegue un attacco ARP poisoning tra il server ed il gateway, ovvero avvelenare la tabella ARP sia del server che del gateway con lo scopo di far passare tutte le comunicazione per la macchina Kali.
Ettercap dispone di una utile e semplice interfaccia grafica con cui si possono lanciare diversi tipi di attacco, è comunque possibile lanciare l'attacco tramite command line:

```
$ ettercap -T -i eth0 -M arp:remote //192.168.140.1//
 //192.168.140.41//
```

L'ultimo passaggio è quello di avviare un proxy in ascolto sulla porta 8080.

```
$ mitmproxy --mode transparent --showhost
```

È necessario che il proxy sia settato in modalità trasparente.

Risultati

Grazie alla configurazione sopra descritta si dovrebbe essere in grado di vedere le comunicazioni tra il sistema centrale e la colonnina, questo non avviene in quanto essendo il traffico protetto da TLS l'handshake con il proxy fallisce (per il fatto che il client non riconosce il certificato di mitmproxy) come si può vedere in Figura 3.18 dove viene riportato il log di mitmproxy. L'indirizzo IP (192.168.68.10) si riferisce

```
info: 192.168.68.10:62124: clientconnect
warn: 192.168.68.10:62124: Client Handshake failed. The client may not
trust the proxy's certificate for evchargersimulator.grupposigla.it.
info: 192.168.68.10:62124: clientdisconnect
```

Figura 3.18: Log mitmproxy

alla macchina dove è attivo il simulatore di colonnine e da cui partono le richieste. Per avere una prova che l'attacco è impostato correttamente al di fuori del TLS, è possibile forzare il browser a procedere comunque anche se non riconosce il certificato. Così facendo il traffico appare su mitmproxy (Figura 3.19).

```
GET https://evchargersimulator.grupposigla.it/ HTTP/2.0
    ← 200 text/html 9.88k 128ms
GET https://evchargersimulator.grupposigla.it/runtime.1ea08ae093d97e0931bf.js HTTP/2.0
    ← 200 application/javascript 1.05k 592ms
GET https://evchargersimulator.grupposigla.it/polyfills.c54a0d23b00ea1f58d71.js HTTP/2.0
    ← 200 application/javascript 36.21k 501ms
GET https://evchargersimulator.grupposigla.it/main.26e92b5ffd35fddee958.js HTTP/2.0
    ← 200 application/javascript 689.24k 611ms
GET https://evchargersimulator.grupposigla.it/styles.099f5df5a0e6d4508f79.css HTTP/2.0
    ← 200 text/css 441.47k 1.04s
GET https://evchargersimulator.grupposigla.it/assets/svg/E-ChargerIconDisc.svg HTTP/2.0
    ← 304 [no content] 119ms
GET https://evchargersimulator.grupposigla.it/primeicons.3a0d4a58da62cf7c55bb.ttf HTTP/2.0
    ← 304 [no content] 136ms
GET https://evchargersimulator.grupposigla.it/favicon.ico HTTP/2.0
    ← 200 image/x-icon 948b 262ms
GET https://evchargersimulator.grupposigla.it/primeicons.e61f3495a7ecd3d571a4.woff
    HTTP/2.0
    ← 200 font/x-woff 56.11k 155ms
```

Figura 3.19: Traffico proveniente dal client

Tuttavia, se si prova a creare una connessione WebSocket con il sistema centrale si ha il seguente errore dovuto allo stesso motivo spiegato sopra.

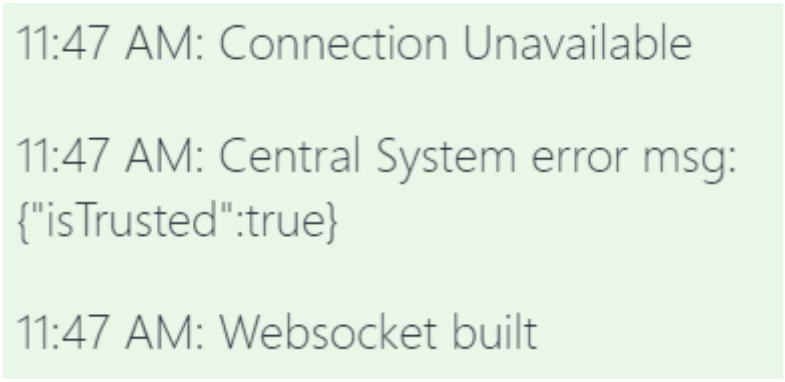

Figura 3.20: Errore creazione WebSocket

Conclusioni

In conclusione l'attacco riesce a metà, per completarlo servirebbe intercettare l'handshake iniziale del TLS e riuscire a ricavare la chiave di sessione utilizzata per criptare le comunicazioni. A tale scopo è stato analizzato l'handshake tra il server ed il client, tramite Wireshark[6], nel momento in cui viene creata la connessione WebSocket, i pacchetti in esame sono quelli mostrati in Figura 3.21.

[6]Wireshark è un software per analisi di protocollo o packet sniffer utilizzato per la soluzione di problemi di rete, per l'analisi e lo sviluppo di protocolli o di software di comunicazione.

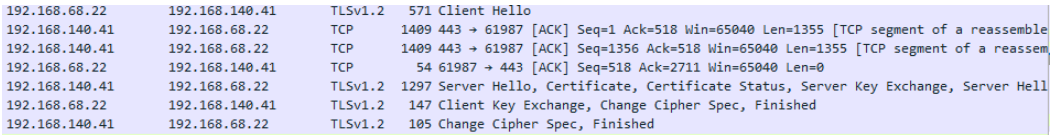

Figura 3.21: Pacchetti handshake TLS

In particolare, si noti il pacchetto Server Hello dove il server indica il cipher scelto e quindi più nel dettaglio il metodo di scambio chiavi da utilizzare. In questo caso viene scelto **ECDHE_RSA** ossia (Elliptic Curve) Diffie-Hellman Ephimeral ovvero viene utilizzato il protocollo Diffie-Hellman[7] con la caratteristica che la coppia di chiavi generate dal server non sono permanenti. Inoltre e soprattutto i parametri DH (PubKey), necessari al client per calcolare le chiavi, vengono mandati con il pacchetto Server Key Exchange e vengono firmati utilizzando il certificato del server come si può vedere in Figura 3.22.

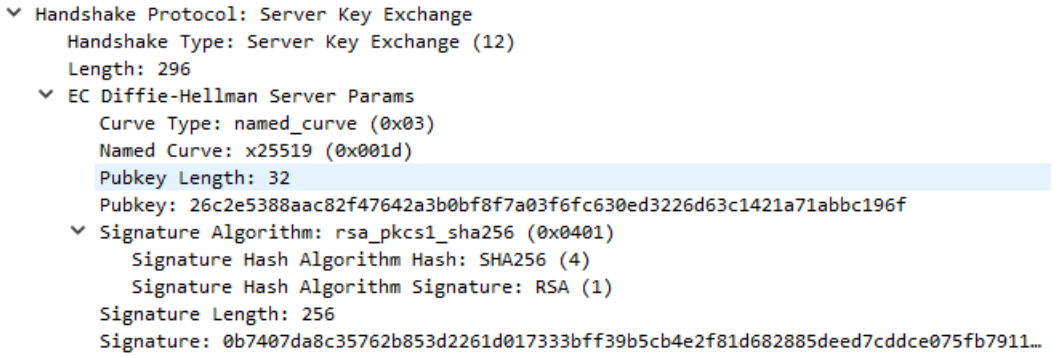

Figura 3.22: Server Key Exchange

Il sopra citato metodo dovrebbe impedire un attacco MitM perché anche se un attaccante blocca il pacchetto e rimpiazza i parametri con i suoi non è chiaramente in grado di generare la signature esatta e quindi l'handshake fallirebbe. Inoltre viene garantita PFS (Perfect Forward Secrecy)[8] proprio perché la coppia di chiavi generate con DH non sono permanenti.
In linea teorica quindi un attacco MitM volto a intercettare la chiave di sessione non è possibile.
Tutto ciò va un po' in contrasto con l'attacco descritto in sezione 3.1.1 nel quale si riusciva a portare a termine l'obiettivo di posizionarsi nel mezzo tra colonnina e

[7]Diffie-Hellman è un protocollo crittografico che consente a due entità di stabilire una chiave condivisa e segreta utilizzando un canale di comunicazione insicuro senza la necessità che le due parti si siano scambiate informazioni o si siano incontrate in precedenza.

[8]PFS, è una proprietà dei protocolli di negoziazione delle chiavi che assicura che se una chiave di cifratura a lungo termine viene compromessa, le chiavi di sessione generate a partire da essa rimangono riservate.

sistema centrale. Come già detto in precedenza però non viene specificato come il TLS venga applicato e quindi il tipo di metodo di Key Exchange; è pur vero che le applicazioni del TLS potrebbero differire da produttore a produttore e quindi potrebbero esserci dei casi in cui attacchi di questo tipo potrebbero essere realizzati.

3.6.3 Test attacco messaggi OCPP

Anche se attraverso l'ambiente di simulazione non si è riusciti ad eseguire un MitM completo ed essere in grado di leggere e manipolarli le comunicazioni, si è comunque cercato di intercettare i messaggi scambiati da sistema centrale e colonnina ed eventualmente modificarli assumendo di aver trovato un modo per rompere il TLS. Per fare ciò si è utilizzato Burp Suite[9] che permette di intercettare sia le chiamate HTTP che i messaggi WebSocket, inoltre per poter visualizzare le comunicazioni in chiaro è stato installato nel browser dove vengono eseguiti i test il certificato di Burp.
Un esempio è il seguente:

- Tramite lo swagger si effettua una richiesta di UpdateFirmware la quale genera un messaggio in formato JSON e lo invia attraverso la connessione WebSocket.

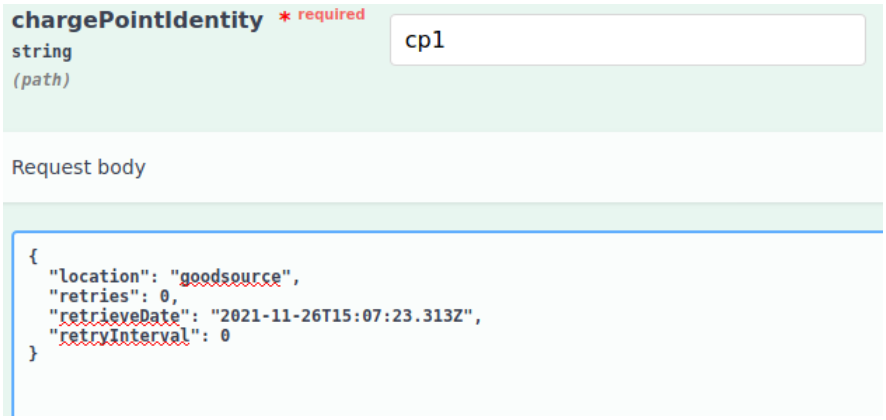

Figura 3.23: Richiesta UpdateFirmware swagger

- Il messaggio viene intercettato da Burp.

```
[2,"45f471cc-d083-4b64-8c51-ff4da3902e55","UpdateFirmware",
{"location":"goodsource","retries":0,"retrieveDate":"2021-1
1-26T15:07:23.313Z","retryInterval":0}]
```

Figura 3.24: Messaggio richiesta UpdateFirmware

[9]Burp Suite è una suite che include un gran numero di strumenti per la sicurezza delle applicazioni web.

- Si possono quindi modificare i parametri, per esempio **location** iniettando un URL malevolo. Successivamente il messaggio viene inoltrato al CP.

```
[2,"45f471cc-d083-4b64-8c51-ff4da3902e55","UpdateFirmware",
{"location":"badsource","retries":0,"retrieveDate":"2021-11
-26T15:07:23.313Z","retryInterval":0}]
```

Figura 3.25: Messaggio modificato richiesta UpdateFirmware

- A questo punto la colonnina riceve il messaggio modificato e lo accetta.

```
10:11 AM: CP sent: [3,"45f471cc-d083-4b64-8c51-
ff4da3902e55",{}]

10:11 AM: Server Request: [2,"45f471cc-d083-4b64-8c51-
ff4da3902e55","UpdateFirmware",
{"location":"badsource","retries":0,"retrieveDate":"2021-11-26T15:07
```

Figura 3.26: Ricezione richiesta UpdateFirmware e risposta

Allo stesso modo sono stati testati gli scenari analizzati in precedenza, per quanto le funzionalità del simulatore siano limitate ovvero non è implementata nessuna logica e quindi è difficile valutarne le conseguenze. Nonostante ciò, è stato possibile valutarne in un primo approccio la fattibilità degli attacchi, infatti in tutti i test l'intercettazione e la conseguente modifica delle comunicazioni non hanno poi portato la colonnina a rifiutare i messaggi ma bensì sono stati accettati come si è visto nell'esempio sopra. Sintomo di come il protocollo non adotti nessuna misura di sicurezza oltre al TLS.
Fortunatamente con la versione 2.0 del protocollo, vengono introdotti nuovi strumenti che possono aiutare a mitigare o rilevare gli attacchi discussi.

3.7 White paper - Miglioramento sicurezza 1.6

È risaputo come la versione 1.6 sia la più utilizzata in pratica, fortunatamente insieme all'uscita di 2.0, è stato redatto un white paper in cui alcune novità in ambito di sicurezza vengono estese anche alla versione 1.6.
I miglioramenti riguardano tre categorie: Setup sicuro per la connessione, il processo di aggiornamento del firmware e una procedura che notifica direttamente il CS quando accadono eventi critici riguardanti la sicurezza insieme ad una rivisitazione di una procedura già presente che permette di recuperare i log generati dalla colonnina.

3.7.1 Setup connessione sicura

PROFILE	CHARGE POINT AUTHENTICATION	CENTRAL SYSTEM AUTHENTICATION	COMMUNICATION SECURITY
1. Unsecured Transport with Basic Authentication	HTTP Basic Authentication	-	-
2. TLS with Basic Authentication	HTTP Basic Authentication	TLS authentication using certificate	Transport Layer Security (TLS)
3. TLS with Client Side Certificates	TLS authentication using certificate	TLS authentication using certificate	Transport Layer Security (TLS)

Figura 3.27: Profili sicurezza

Vengono definiti 3 profili:

1. Non offre nessuna sicurezza, viene usata solo una HTTP Basic Authentication in cui il CP fornisce username e password che servono al CS per autenticarlo. Viene esplicitamente detto come questo profilo debba essere usato in reti fidate (VPN).

2. Il canale di comunicazione viene reso sicuro usando TLS, il CS si autentica usando un certificato TLS server, mentre il CP usa una HTTP Basic Authentication come sopra. Viene fortemente raccomandato l'uso di TLS 1.2.

3. Il canale di comunicazione viene reso sicuro utilizzando TLS, si ha una mutua autenticazione usando certificati sia da parte di CS che di CP.

La password usata in HTTP Basic Authentication può essere aggiornata dal CS usando la procedura riportata in Figura 3.28.

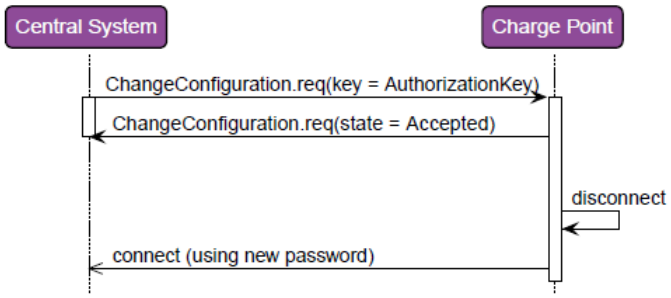

Figura 3.28: Update Password

Il cambiamento viene segnalato nei security log, ma non viene informato direttamente il CS.
Vengono inoltre descritte le procedure per aggiornare i certificati e viene menzionato come i CPO (operatori dei CP) possono agire da CA (Certificate Authority) rendendo così il server CA locale.
La scelta del profilo è controllata dalla chiave **SecurityProfile** che può essere cambiata, nella specifica viene esplicitamente detto che se un CP riceve una richiesta di modifica del valore con uno minore od uguale a quello già presente, deve rifiutare la richiesta. Ciò stronca in partenza un tentativo di un attaccante di diminuire il livello di sicurezza.
In conclusione, tralasciando il profilo 1 che come detto deve essere usato solo in reti fidate e che non offre nessuna sicurezza, i profili 2 e 3 grazie all'uso di certificati dovrebbero garantire un sufficiente livello di protezione impedendo anche attacchi MitM. Questo a patto che vengano utilizzati metodi i quali garantiscono ciò ovvero quale cipher suites viene scelto nel handshake iniziale, nei requisiti vengono indicate delle possibili scelte (vedere sezione 4.1.1).

3.7.2 Eventi di sicurezza - logging

Viene definita una procedura per informare immediatamente il CS quando si verificano eventi che riguardano la sicurezza. I passi sono descritti nella Figura 3.29, nel momento in cui un security event si verifica ed è critico allora il CP notifica il CS. Viene data una lista di eventi di sicurezza distinguendoli tra critici e non (Figura 3.30). Un esempio di evento critico è la modifica dell'orario interno al CP. Un esempio di evento non critico è la riconfigurazione dei parametri di sicurezza (le chiavi) come per esempio **SecurityProfile** oppure **AuthorizationKey** viste in precedenza. Proprio quest'ultimo punto lascia molte perplessità, infatti sono molte le chiavi che influenzano direttamente il comportamento della colonnina (ad es.

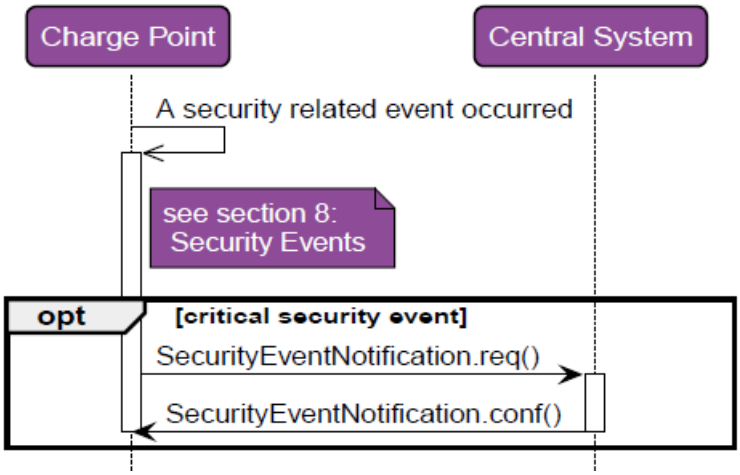

Figura 3.29: Notifica diretta eventi critici

SecurityProfile, **AuthorizationKey**, **HeartbeatInterval**) ed una loro modifica indesiderata potrebbe avere conseguenze molto gravi sul CP stesso.

SECURITY EVENT	DESCRIPTION	CRITICAL
CentralSystemFailedToAuthenticate	The authentication credentials provided by the Central System were rejected by the Charge Point	No
SettingSystemTime	The system time on the Charge Point was changed	Yes
StartupOfTheDevice	The Charge Point has booted	Yes
ResetOrReboot	The Charge Point was rebooted or reset	Yes
SecurityLogWasCleared	The security log was cleared	Yes
ReconfigurationOfSecurityParameters	Security parameters, such as keys or the security profile used, were changed	No
MemoryExhaustion	The Flash or RAM memory of the Charge Point is getting full	Yes

Figura 3.30: Esempio eventi critici

Viene definita un'altra procedura allo scopo di permettere al CP il caricamento di file di log ad un dato URL, molto simile ad una funzione già presente nella versione 1.6 che permetteva di caricare i dati di diagnostica (sezione 3.3.2). In aggiunta qui si caricano log riguardanti la sicurezza (ad es. cambio di password HTTP Basic Authentication).
In conclusione, la nuova funzione *SecurityEventNotification* può essere un utile strumento contro alcuni attacchi proprio perché informa tempestivamente il CS che può

quindi agire di conseguenza. Lascia un po' di perplessità la scelta di alcuni eventi classificati come non critici (riconfigurazione dei parametri di sicurezza su tutti), una spiegazione potrebbe essere il fatto che i suddetti eventi accadono molto frequentemente e quindi la notifica diretta ogni volta produrrebbe dei rallentamenti. In 2.0 vengono introdotti dei nuovi strumenti che mitigano il problema sopra descritto ma che in 1.6 non sono presenti lasciando così la questione irrisolta. Probabilmente nella versione 1.6 sarebbe il caso di rivedere la criticità di alcuni eventi.
Inoltre si ha un'incoerenza su quanto viene detto nei requisiti dei profili di sicurezza e sulla lista degli eventi, ovvero nei requisiti è richiesto che alcuni eventi (InvalidTLSCipherSuite, InvalidCentralSystemCertificate ecc.) siano notificati al CS ma poi nella lista non vengono segnalati come critici.

3.7.3 Update del firmware sicuro

Il processo di aggiornamento del firmware (Figura 3.31) viene reso sicuro grazie all'uso di cerificati e firma digitale.

In *SignedUpdateFirmware.req* è presente il certificato con cui è stato firmato il firmware, il CP verifica la validità del certificato rispetto al Manufacturer root certificate (questo perché solitamente chi gestisce i CP e chi li produce sono entità diverse). Se la verifica va a buon fine, il CP esegue il download del firmware e successivamente ne controlla la firma (presente anch'essa in *SignedUpdateFirmware.req*). Quindi la possibilità per un attaccante di riuscire nell'intento di far scaricare e installare un malware viene azzerata perché anche se si riesce a far accettare un URL malevolo e quindi ad eseguire il download, la verifica della firma fallirebbe in quanto non è possibile produrre una signature coerente. Per fare ciò si dovrebbe cambiare anche il certificato ma allora la verifica fallirebbe nel momento in cui viene controllato rispetto al Manufacturer root certificate.

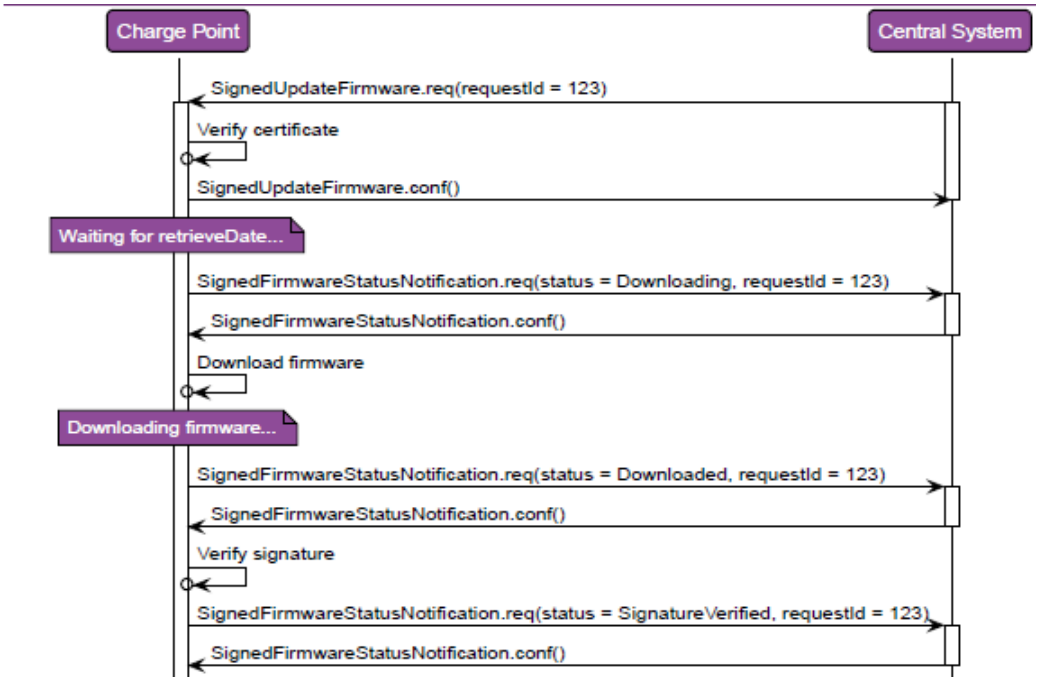

Figura 3.31: Update firmware sicuro

Capitolo 4

Analisi OCPP 2.0

Nel seguente capitolo si è cercato di analizzare la nuova versione del protocollo OCPP: per ogni categoria, così come sono suddivise nella specifica, si è data innanzitutto una breve spiegazione del funzionamento e le maggiori differenze con la versione precedente. Sono state poi inserite alcune considerazioni riguardanti i vari casi d'uso (indicati con la stessa nomenclatura utilizzata in [10]) allo scopo di evidenziare: le possibili nuove vulnerabilità e l'utilità dei nuovi strumenti rispetto ad attacchi già noti da 1.6.
È necessario chiarire come queste considerazioni siano fatte sulla falsa riga dell'analisi a 1.6 ed inoltre non sia possibile verificare la bontà di esse in quanto in letteratura non esistono ancora documenti che trattano la sicurezza della nuova versione.
Infine si è cercato di riassumere quali attacchi descritti in 1.6 sono stati risolti o per lo meno mitigati utilizzando le nuove funzionalità introdotte, inoltre si è provato ad ideare dei nuovi scenari di attacco sfruttando le nuove vulnerabilità.

4.1 Start-up e Configurazione

4.1.1 Security

Si opera allo stesso modo visto nel white paper di miglioramento di 1.6 (sezione 3.7.1) ovvero vengono definiti tre profili.
Viene detto come non siano incluse misure di sicurezza a livello applicativo e che la sicurezza del protocollo è basata sul TLS e la criptografia a chiave pubblica.

Considerazioni security profiles

- Nei requisiti dei profili 2 e 3 viene richiesto che il CP debba supportare almeno i seguenti ciphers:
  (**TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256** AND
  TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384) OR
  (**TLS_RSA_WITH_AES_128_GCM_SHA256** AND
  TLS_RSA_WITH_AES_256_GCM_SHA384). È noto come i primi due non permettano attacchi MitM volti a trovare la chiave di sessione usata per criptare i messaggi e anche supportino PFS (Perfect Forward Secrecy). Mentre per i secondi due viene detto esplicitamente come non supportino PFS, inoltre in letteratura esistono degli attacchi contro il metodo RSA, tanto che viene suggerito di non usarlo più e anzi nella versione 1.3 del TLS non si può adottare. Quindi l'adozione degli ultimi due metodi potrebbe compromettere la sicurezza della comunicazione tra CP e CS.

Considerazioni casi d'uso

- Viene detto come sia importante aggiornare il prima possibile le credenziali di autenticazione della colonnina e che finché non è stato fatto limitare le funzionalità del CP. È una buona pratica per evitare che le colonnine abbiano delle credenziali di default facilmente deducibili aprendo a scenari esplorati nel capitolo 5.

- **A01-A02-A03**: permettono di aggiornare la password oppure il certificato della colonnina. Data la criticità di questi messaggi ed i dati che trasportano, si potrebbe pensare di applicare il meccanismo di *SignedMessage* (sezione 4.3.3).

- **A04**: viene introdotta la procedura *SecurityEventNotification* esattamente come visto nella sezione 3.7.2. Si applicano le stesse considerazioni viste nel capitolo precedente per quanto riguarda la classificazione (critici e non) degli eventi. Al contrario di 1.6, però, in 2.0 si hanno degli strumenti per poter risolvere il problema senza cambiare la criticità di alcuni eventi (ad es. impostando dei monitor (sezione 4.3.2)).

- **A05**: permette al CS di poter modificare la priorità dei **NetworkProfile** (Figura 4.2) adottandone così uno diverso da quello in utilizzo. È esplicitamente detto come un CP non debba accettare una richiesta contenente un profilo con un livello di sicurezza minore di quello in uso.

4.1.2 Provisioning

Questa categoria descrive le funzionalità che aiutano un operatore a ricevere informazioni sulla configurazione di un CP ed eventualmente a modificarla. Molto simile a 1.6, troviamo ancora la procedura *BootNotification*, la possibilità di cambiare il valore delle variabili e di richiedere un reset alla colonnina.

Considerazioni casi d'uso

- **B01**: la procedura *BootNotification* è uguale a 1.6 e quindi anche gli attacchi restano ancora possibili (sezione 3.1.2). L'unica differenza con 1.6 è che successivamente a *BootNotification*, la colonnina è obbligata a riferire lo stato di ciascun connettore tramite *StatusNotificationRequest* potendo così aiutare a scovare un attacco in cui si forzano i connettori allo stato di Unavailable/Reserved/Faulted. Infatti si ricorda come lo stato debba persistere anche dopo un reboot.

- **B07-B08** viene introdotta la procedura mostrata in Figura 4.1.

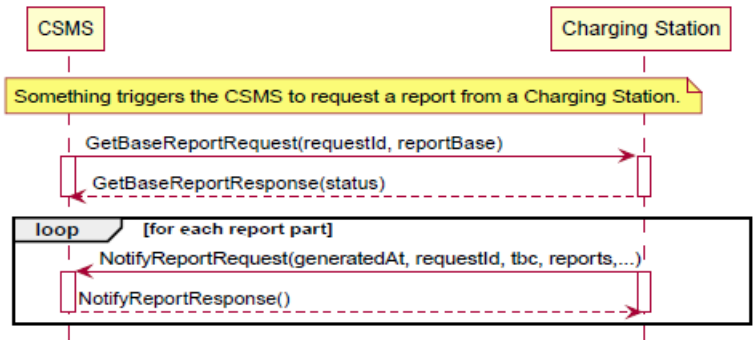

Figura 4.1: GetBaseReport

  Potrebbe essere utile al fine di controllare lo stato di un CP e quindi scovare anche attacchi che mirano a cambiare certi parametri o la disponibilità stessa di una colonnina. Esiste anche una procedura analoga *GetCustomReport* in cui si può specificare quali tipi di informazioni debbano essere presenti nel report.

- **B09-B10**: è possibile aggiungere nuovi **NetworkProfile** i quali sono composti dai parametri mostrati in Figura 4.2.

Field Name	Field Type	Card.	Description
ocppVersion	OCPPVersionEnumType	1..1	Required. Defines the OCPP version used for this communication function.
ocppTransport	OCPPTransportEnumType	1..1	Required. Defines the transport protocol (e.g. SOAP or JSON). Note: SOAP is not supported in OCPP 2.0, but is supported by other versions of OCPP.
ocppCsmsUrl	string[0..512]	1..1	Required. URL of the CSMS(s) that this Charging Station communicates with.
messageTimeout	integer	1..1	Required. Duration in seconds before a message send by the Charging Station via this network connection times-out. The best setting depends on the underlying network and response times of the CSMS. If you are looking for a some guideline: use 30 seconds as a starting point.
securityProfile	integer	1..1	Required. This field specifies the security profile used when connecting to the CSMS with this NetworkConnectionProfile.
ocppInterface	OCPPInterfaceEnumType	1..1	Required. Applicable Network Interface.
vpn	VPNType	0..1	Optional. Settings to be used to set up the VPN connection
apn	APNType	0..1	Optional. Collection of configuration data needed to make a data-connection over a cellular network.

Figura 4.2: NetworkProfile

Inoltre come spiegato in **B10** è possibile modificare la variabile **NetworkConfigurationPriority** che contiene la lista dei profili in ordine di priorità, il CP poi utilizzerà quello con più alta priorità. Questo metodo è utilizzato per migrare ad un nuovo CS. Le due funzionalità qui definite aggiungono un metodo interessante ma allo stesso tempo potrebbe essere utilizzato da un attaccante al fine di far migrare il CP verso un CS malevolo. Probabilmente sarebbe meglio che la migrazione fosse gestita al di fuori di OCPP oppure utilizzando il meccanismo di *SignedMessage* (4.3.3) sui messaggi che compongono la procedura descritta.

4.2 Transazioni e controllo

4.2.1 Authorization

Nella parte di autorizzazione vengono inseriti nuovi tipi di ID (Figura 4.3), nella versione 1.6 gli identificativi erano composti da una stringa e non si faceva differenza sui vari metodi di autorizzazione.

Value	Description
Central	A centrally, in the CSMS (or other server) generated id (for example used for a remotely started transaction that is activated by SMS). No format defined, might be a UUID.
eMAID	Electro-mobility account id as defined in ISO 15118
ISO14443	ISO 14443 UID of RFID card. It is represented as an array of 4 or 7 bytes in hexadecimal representation.
ISO15693	ISO 15693 UID of RFID card. It is represented as an array of 8 bytes in hexadecimal representation.
KeyCode	User use a private key-code to authorize a charging transaction. For example: Pin-code.
Local	A locally generated id (e.g. internal id created by the Charging Station). No format defined, might be a UUID
MacAddress	
NoAuthorization	Transaction is started and no authorization possible. Charging Station only has a start button or mechanical key etc. IdToken field SHALL be left empty.

Figura 4.3: Tipologie di ID

Vengono inoltre inseriti i **GroupId** ovvero un insieme di ID è trattato come un gruppo, ciò dà la possibilità di iniziare una transazione con un ID e di finirla con un altro appartenente allo stesso gruppo.
L'autorizzazione offline viene gestita analogamente a 1.6 con Authorization Cache e/o Local Authorization List (LAL), inoltre viene menzionato il fatto che queste due strutture possono essere usate anche quando la connessione con il CS è lenta per migliorare l'esperienza utente. Viene esplicitamente detto come le due strutture siano differenti e se sono supportate tutte e due da un CP, allora LAL ha priorità sulla cache.

Considerazioni casi d'uso

- **C04**: è possibile autorizzare un utente utilizzando un PIN-code (Figura 4.4), quindi la possibilità di un attacco brute-force è concreta. A tal proposito, nei requisiti viene raccomandato di prendere delle precauzioni, ad esempio aumentando il tempo tra l'inserimento di un PIN sbagliato ed il nuovo tentativo.

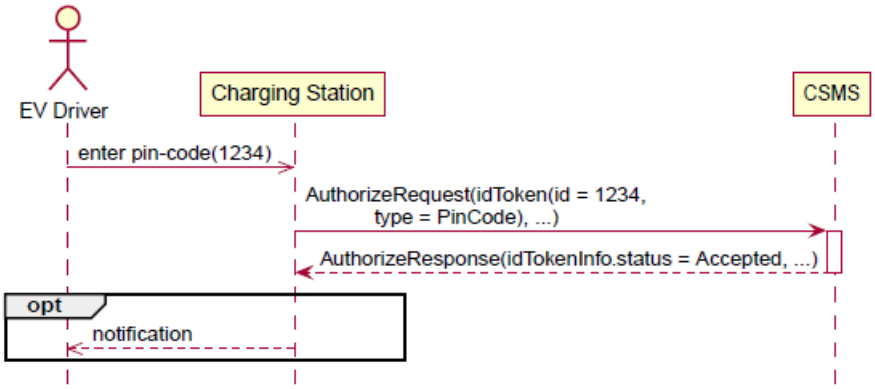

Figura 4.4: Autorizzazione via PIN-code

Inoltre considerando l'attacco visto in 1.6 in cui si iniettavano dei falsi identificativi attraverso LAL per arrivare ad un furto di elettricità, iniettando dei PIN-code l'attacco sembra essere ancora più semplice.

- **C10**: viene raccomandato di salvare gli ID nella cache in modo sicuro, per esempio salvando solo l'hash. Potrebbe essere una soluzione al furto di dati quando la colonnina è attaccata fisicamente.

- **C15**: se la variabile **OfflineTxForUnknownIdEnabled** esiste ed è settata a true la colonnina può accettare un ID che non è presente nella cache e/o nella LAL e nello scenario in cui il CP è offline. La variabile è inoltre ReadWrite, perciò il suo valore può essere cambiato usando la procedura *SetVariable*.
  A questo punto un attaccante potrebbe essere autorizzato ad una ricarica presentando un ID di qualsiasi tipo e intervenendo nella comunicazione CP $\leftrightarrow$ CS al fine di forzare la colonnina in uno stato offline. Se la variabile sopra descritta è implementata e con valore true, si può quindi ottenere un furto di elettricità anche senza dover fare i conti con il TLS o con qualunque altra misura di sicurezza in quanto basta impedire la comunicazione tra CP e CS.

- **C16**: utilizzando un ID che appartiene al gruppo definito in **MasterPassGroupId** è possibile fermare una transazione in atto; può essere utilizzato dalle forze dell'ordine. Ipoteticamente un attaccante in grado di cambiare il valore della variabile per farla coincidere con il suo gruppo avrebbe in mano un master pass in grado di fermare qualsiasi transazione.

4.2.2 LocalAuthorizationList Management

Qui il comportamento è analogo a 1.6.

Considerazioni casi d'uso

- **D01**: la procedura *SendLocalList* è uguale a 1.6 e quindi persistono i possibili attacchi visti utilizzando questa funzione. Inoltre, come si diceva in 4.2.1, è possibile inserire ID di tipo PIN-code i quali potrebbero facilitare l'attacco che mira al furto di elettricità.

4.2.3 Transactions

Nella nuova versione i messaggi riguardanti le transazioni cambiano, esiste un solo messaggio *TransactionEventRequest* che al suo interno ha un attributo **eventType** il quale indica lo stato della transazione (Started,Updated,Ended). Inoltre in 1.6 l'invio del messaggio StartTransactions non era definito precisamente, ora invece le due variabili **TxStartPoint** e **TxStopPoint** definiscono rigorosamente quando una transazione deve iniziare e quando deve finire. Le due variabili contengono una lista di eventi mostrati in Figura 4.5.

Value	Description
ParkingBayOccupancy	An object (probably an EV) is detected in the parking/charging bay.
EVConnected	Both ends of the Charging Cable are connected (if this can be detected, otherwise detection of a cable being plugged in to the socket), or for wireless: initial communication between EVSE and EV is established.
Authorized	Driver or EV is authorized, this can also be some form of anonymous authorization like a start button.
DataSigned	Signed data is received from the energy meter which is required by some legislation. There are countries that require signed metering data before a billable transaction can be started.
PowerPathClosed	All preconditions are met, power can flow. In case of a wired charger, the cable is properly connected, driver is authorized, power relay is closed etc. This does not mean that the EV is read to charge it's battery, it might, for example, be to warm.
EnergyTransfer	Energy is being transferred between EV and EVSE.

Figura 4.5: Possibile eventi per cui una transazione inizia/finisce

La transazione viene avviata quando tutti gli eventi in **TxStartPoint** si verificano, mentre termina nel momento in cui almeno uno degli eventi in **TxStopPoint** non è più valido.
In 2.0 l'ID della transazione è generato dal CP a differenza di 1.6 in cui se si avevano transazione offline, le stesse rimanevano senza ID rendendo difficile poi ricollegarle.

Un'altra funzionalità aggiunta è il sequence number ovvero il CP mantiene un counter per ogni sua EVSE e inserisce questo numero nel messaggio di transazione. Ciò permette al CS sia di riordinare i messaggi relativi ad una transazione sia di verificare che ci siano tutti.

Considerazioni casi d'uso

- **E05**: il CS deve controllare in ogni *TransactionEventRequest* che riceve la validità dell'ID associato all'utente e se per caso quest'ultimo non è più valido, il CP viene notificato e deve procedere a stoppare la transazione. Il modo in cui il CP esegua tale compito è regolamentato da due variabili: **StopTxOnInvalidId** e **MaxEnergyOnInvalidId**. La prima determina se il CP è abilitato a fermare una transazione quando riceve uno stato di autorizzazione diverso da Accepted, mentre la seconda definisce l'ammontare di energia da immettere quando un ID viene de-autorizzato. Quindi se la prima è settata a false e la seconda viene implementata e impostata ad un valore alto, si potrebbe avere una ricarica nonostante l'ID non sia più valido.

- **E12**: una volta che la connessione con il CS viene ripristinata, il CP è tenuto ad inviare tutti i messaggi delle transazioni effettuate nel periodo offline (Figura 4.6). Non è chiaro come ci si debba comportare nel momento in cui si riceve una transazione con un identificativo non abilitato alla ricarica.

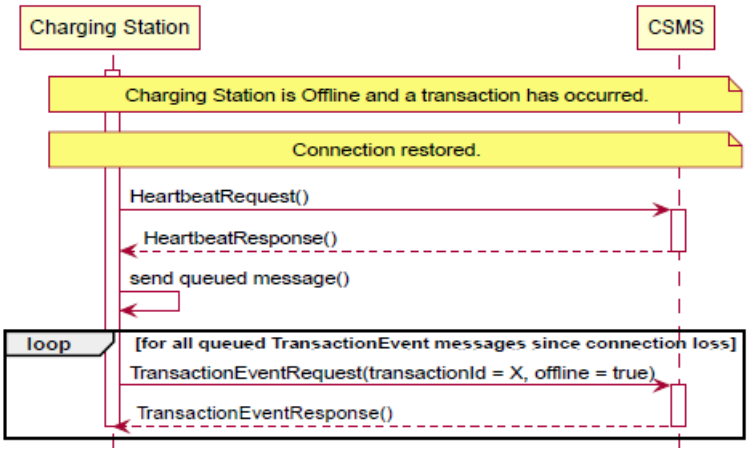

Figura 4.6: Invio transazioni offline

- **E13**: esistono scenari in cui il messaggio non arriva a destinazione o non viene accettato dal CS. Il CP quindi riprova l'invio un numero di volte definito in **MessageAttemptsTransactionEvent** e con un intervallo definito in

MessageAttemptIntervalTransactionEvent. Anche qui non è chiaro come procedere se per ipotesi l'invio di *TransactionEventRequest* con **eventType** uguale a Ended non va a buon fine e si raggiunge il numero di volte definito nella prima variabile. Infatti senza le informazioni contenute in quel messaggio non è possibile, per esempio, eseguire la fatturazione della transazione.

4.2.4 RemoteControl

Alla pari di 1.6, si ha la possibilità di iniziare o finire una transazione da remoto ovvero su richiesta del CS e di sbloccare un connettore nel caso in cui gli utenti abbiano dei problemi. Inoltre viene aggiunta la funzionalità Remote Trigger ossia il CS può richiedere l'invio di determinati messaggi da parte del CP o sapere lo stato dei processi in corso.

Considerazioni casi d'uso

- **F02**: sembra che l'attacco in cui si inoltra una transazione usando proprio una transazione remota sia ancora possibile. Probabilmente la difficoltà aumenta per la mole di messaggi che vengono continuamente scambiati durante una sessione di ricarica.

- **F06**: qui viene descritto come usare la nuova funzionalità del Remote Trigger, i tipi di messaggi che possono essere richiesti sono mostrati in Figura 4.7.

Value	Description
BootNotification	To trigger BootNotification.
LogStatusNotification	To trigger LogStatusNotification.
FirmwareStatusNotification	To trigger FirmwareStatusNotification.
Heartbeat	To trigger Heartbeat.
MeterValues	To trigger MeterValues.
SignChargingStationCertificate	To trigger a SignCertificate with typeOfCertificate: ChargingStationCertificate.
SignV2GCertificate	To trigger a SignCertificate with typeOfCertificate: V2GCertificate
StatusNotification	To trigger StatusNotification.
TransactionEvent	To trigger TransactionEvent.
SignCombinedCertificate	To trigger a SignCertificate with typeOfCertificate: ChargingStationCertificate AND V2GCertificate
PublishFirmwareStatusNotification	To trigger PublishFirmwareStatusNotification.

Figura 4.7: Tipi di messaggio da richiedere con Remote Trigger

Può risultare un utile strumento nel caso in cui alcuni messaggi non arrivino oppure richiedere lo stato di alcuni processi sospetti. Potrebbe anche essere usato in modo malevolo cercando di sovraccaricare il CP con queste richieste.

4.2.5 Availability

In questa categoria vengono descritte le procedure che informano il CS dello stato corrente della colonnina, inoltre vengono fornite delle funzioni allo scopo di modificare la disponibilità del CP.

Considerazioni casi d'uso

- **G01**: parimenti a 1.6, ogni qualvolta un connettore cambia stato il CP è tenuto ad informare il CS tramite *StatusNotification* (Figura 4.8).

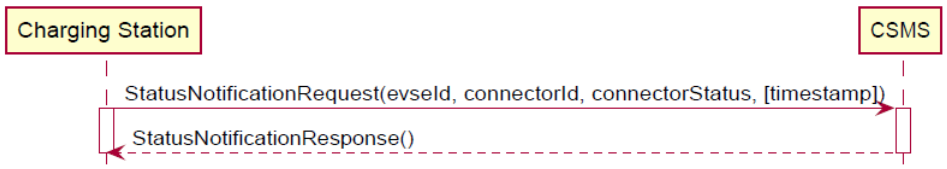

Figura 4.8: StatusNotification

  Può essere un utile strumento al fine di rilevare attacchi che mirano a cambiare la disponibilità dei connettori.

- **G02**: la procedura *Heartbeat* non cambia rispetto a 1.6, quindi un attacco nel quale si punta a minimizzare l'intervallo fra un heartbeat ed un altro è ancora possibile. Viene detto come questa procedura non abbia la funzione di tenere la connessione WebSocket attiva dal momento che esiste un meccanismo il quale si occupa di ciò (ping-pong WebSocket). Comunque viene raccomandato di mandare almeno un heartbeat ogni 24 ore per permettere al CP di sincronizzare l'orario. Date le considerazioni precedenti, si potrebbe pensare ad impostare un limite minimo sotto il quale il valore della variabile **HeartbeatInterval** non deve scendere.

4.2.6 Reservation

Le procedure usate per riservare un connettore ad un utente sono similari a quelle presenti in 1.6.

Considerazioni casi d'uso

- **H01**: non è chiaro perché non utilizzare lo stesso procedimento visto in sezione 4.2.5 ovvero nel momento in cui un connettore/EVSE/CP cambia il proprio stato si informa il CS (si usa solo nel caso in cui sia specificata la EVSE che

si vuole riservare).
Viene però raccomandato di autorizzare l'ID utilizzato in *ReserveNow*, ciò potrebbe essere un rimedio all'attacco che mira a riservare postazioni di ricarica con ID falsi.
Non è specificato come si debba comportare il CP dopo un reboot ovvero se mantenere o meno i connettori/EVSE riservati.

4.2.7 TariffAndCost

Nuove funzionalità aggiunte in 2.0, permettono di dare informazioni sulle tariffe ed i costi quando una colonnina è in grado di mostrarli su un display. Può anche essere usato per mostrare all'utente il costo totale durante una ricarica.

Considerazioni casi d'uso

- **I01**: in risposta all'autorizzazione può venire inserita la tariffa per il specifico utente (Figura 4.9).

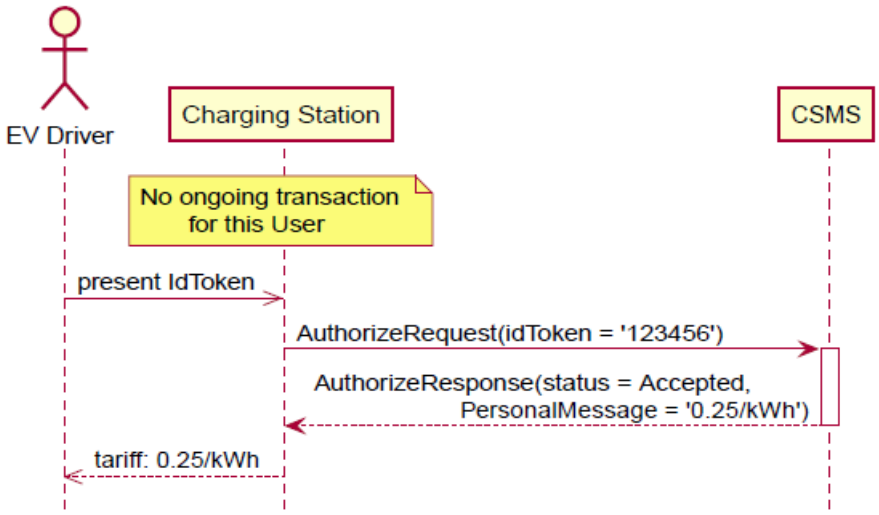

Figura 4.9: Inserimento tariffe in AuthorizeResponse

Se un attaccante intercetta la risposta e modifica la tariffa ad un valore molto alto, si potrebbe aprire uno scenario in cui nessuno ricarica nella colonnina di interesse rendendola inoperativa.

- **I02**: usata dal CS al fine di aggiornare il costo totale durante una ricarica (Figura 4.10). Non viene specificato chi decide l'intervallo Y che passa tra un messaggio ed un altro, viene detto come un valore piccolo crei molti

messaggi. Quindi se si è in grado di agire su Y, si potrebbero provocare dei malfunzionamenti sulla colonnina.

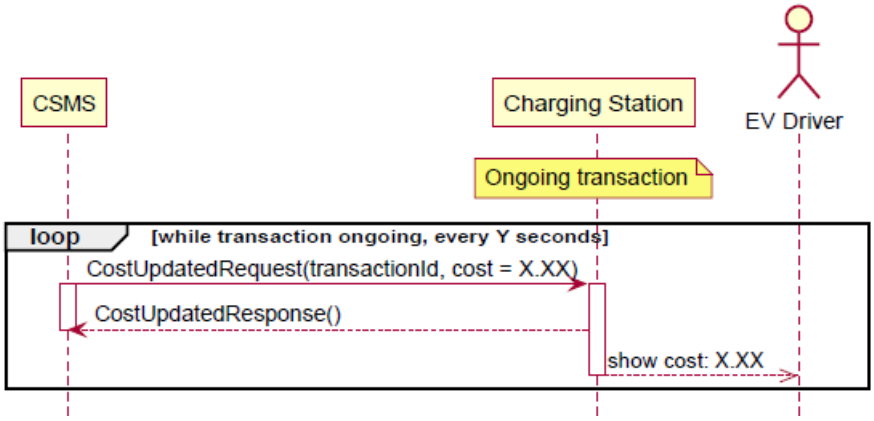

Figura 4.10: Aggiornamento costo totale transazione

- **I04**: simile a **I01** solo che qui la tariffa è generale e non applicata ad un specifico utente, viene controllata dalla variabile **TariffFallbackMessage**. Quindi parimenti a sopra, se si è in grado di modificare questa variabile inserendo un valore di tariffa molto alto, probabilmente nessuno ricaricherà rendendo la colonnina di fatto inoperativa.

4.2.8 MeterValues

Nella corrente categoria vengono descritte le funzionalità che permettono al CP di inviare MeterValues (valori del contatore), vengono descritti due scenari in cui si mandano queste informazioni: durante una transazione per informare l'utente sullo stato della ricarica ma anche per poter determinare il costo della ricarica. Il secondo scenario riguarda un invio programmato, generalmente ogni 15/30 minuti.
Viene inoltre inoltre introdotto la possibilità di firmare i valori, così che non possano essere alterati.

4.2.9 SmartCharging

Qui vengono descritte le funzionalità che permettono agli operatori di influenzare la corrente/potenza trasferita durante una transazione oppure impostare dei limiti alla corrente/potenza che un CP può sottrarre alla rete elettrica.
L'uso dello smart charging può avvenire in diversi scenari e comprendere diverse entità, si possono avere applicazioni dovute ad un bilanciamento interno ai CP oppure può essere direttamente il CS ad applicare dei limiti o ancora ci possono essere delle

entità terze (EMS,DSO) che impongono delle limitazioni.
Proprio nell'ultimo scenario si possono fare delle considerazioni sulla sicurezza: infatti se si suppone che la comunicazione tra queste terze parti ed il CS/CP abbia delle vulnerabilità, le stesse potrebbero essere usate come punto di accesso per tentare di iniettare profili di ricarica dannosi. Ciò potrebbe essere mitigato attraverso un requisito presente nella specifica che impone al CP di notificare il CS quando un un profilo viene cambiato (*NotifyChargingLimitRequest*), questo però viene controllato dalla variabile **LimitChangeSignificance** che definisce la percentuale sotto il quale un cambio di limite nel profilo viene ignorato. Quindi se la variabile ha un valore alto si rischia che la modifica non venga segnalata.
Considerato l'impatto che possono avere dei ChargingProfiles dannosi, si potrebbe pensare di usare la nuova funzionalità *SignedMessage* (sezione 4.3.3) sui messaggi che compongono le varie procedure di attuazione dello smart charging.

Considerazioni casi d'uso

- **K03**: viene data la possibilità ad un local controller (se esiste) di inviare dei profili di ricarica che possono essere predefiniti al suo interno o mandati dal CS. Ciò implica una nuova superficie di attacco: infatti se il local controller viene compromesso, può essere usato allo scopo di iniettare ChargingProfiles dannosi.

- **K07**: se il CP è offline, ogni transazione è regolata dal profilo definito in **TxDefaultProfile**. Ipotizzando quindi che un attaccante sia in grado di iniettare un profilo di ricarica di default dannoso, tutte le transazioni ne sono influenzate e quindi l'impatto è ancora più alto.

4.2.10 DataTransfer

Viene data la possibilità di estendere comandi esistenti con attributi personalizzati oppure aggiungere direttamente nuovi comandi. OCPP offre due meccanismi a tale scopo.

- Il metodo *DataTransfer* esattamente come in 1.6.
- In tutti gli schemi JSON è presente un elemento opzionale **CustomData** (Figura 4.11).

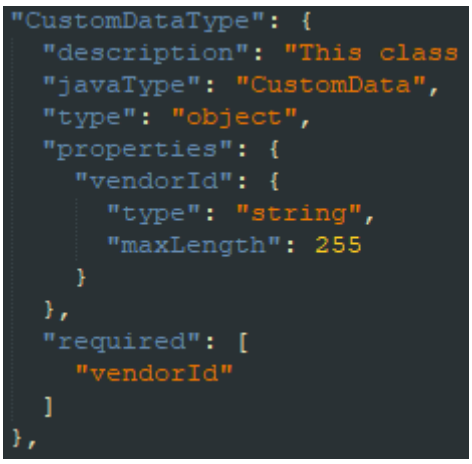

Figura 4.11: Custom Data

È l'unico elemento che permette di aggiungere proprietà aggiuntive e quindi può essere usato per aggiungere attributi di ogni tipo. Chiaramente può essere fonte di nuove vulnerabilità, comunque sia viene detto di non utilizzarlo in implementazioni standard.

Considerazioni casi d'uso

- **P01-P02**: la procedura *DataTrasnfer* risulta analoga 1.6, il campo **data** non ha una lunghezza predefinita. Quindi si può sovraccaricare sia il CP che il CS con messaggi di questo tipo con il campo **data** arbitrariamente lungo.

4.3 Notifiche e Mantenimento

4.3.1 FirmwareManagement

La parte di aggiornamento del firmware viene gestita esattamente come descritto nel white paper di miglioramento di 1.6 (sezione 3.7.3), cambia solo il nome dei messaggi.

Considerazioni casi d'uso

- **L02**: viene descritta la possibilità di eseguire un update del firmware non sicuro come in 1.6, non è chiaro il perché venga data tale possibilità. Forse si pensa a scenari in cui è presente una VPN.

- **L03**: in presenza di un local controller il processo cambia (Figura 4.12). Qui l'unico controllo che viene fatto è la verifica di MD5[1] checksum la quale però dovrebbe garantire soltanto che il file non è stato alterato da terze parti durante il trasporto nel canale di comunicazione, ma se la richiesta proviene da un entità malevola è proprio la stessa ad inserire MD5 e farlo combaciare con il suo file. Non si capisce come mai non si possa applicare lo stesso meccanismo visto in sezione 3.7.3.

4.3.2 Diagnostics

Definisce un insieme di procedure per eseguire la diagnostica del CP. In particolare viene aggiunta la possibilità di impostare un monitor su particolari settaggi o anche su specifiche variabili, ciò può essere un rimedio al fatto che la re-configurazione dei parametri di sicurezza non è un evento critico. Infatti se per esempio viene impostato un monitor sulla variabile **HeartbeatInterval**, il CP notifica il CS quando questa variabile raggiunge i criteri del monitor. Lo stesso procedimento può essere usato per molte altre variabili e quindi è uno strumento molto utile per rilevare attacchi. Di contro può avere un effetto sulle performance della colonnina.

[1] L'MD5 è una funzione hash crittografica realizzata da Ronald Rivest nel 1991 e standardizzata con la RFC 1321.

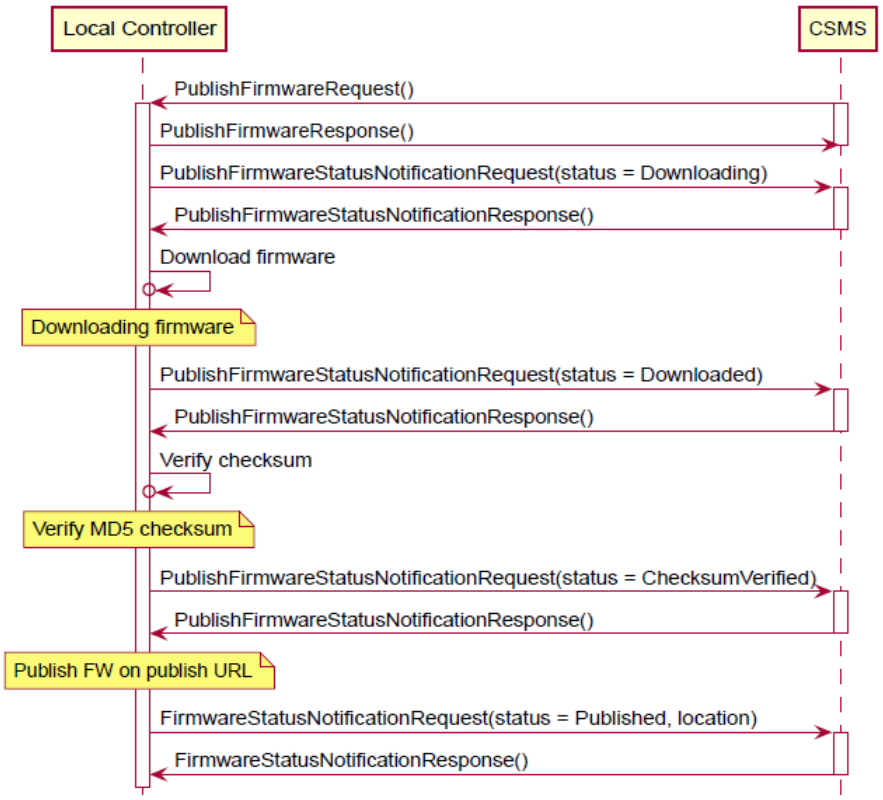

Figura 4.12: Processo di aggiornamento firmware in presenza di local controller

Considerazioni casi d'uso

- **N01**: i log si dividono in diagnostic e security, la procedura di upload è uguale a 1.6, quindi l'attacco che mira a cambiare URL dove i log vengono caricati è ancora possibile.
- **N04-N07-N08**: vengono descritte le procedure per inserire monitor i quali possono essere di 5 tipi (Figura 4.13).

Value	Description
UpperThreshold	Triggers an event notice when the actual value of the Variable rises above *monitorValue*
LowerThreshold	Triggers an event notice when the actual value of the Variable drops below *monitorValue*.
Delta	Triggers an event notice when the actual value has changed more than plus or minus *monitorValue* since the time that this monitor was set or since the last time this event notice was sent, whichever was last. For variables that are not numeric, like boolean, string or enumerations, a monitor of type Delta will trigger an event notice whenever the variable changes, regardless of the value of *monitorValue*.
Periodic	Triggers an event notice every *monitorValue* seconds interval, starting from the time that this monitor was set.
PeriodicClockAligned	Triggers an event notice every *monitorValue* seconds interval, starting from the nearest clock-aligned interval after this monitor was set. For example, a *monitorValue* of 900 will trigger event notices at 0, 15, 30 and 45 minutes after the hour, every hour.

Figura 4.13: Tipi di monitor

La comunicazione con il CS avviene tramite il messaggio *NotifyEvent*.

- **N09**: procedura che permette al CS di ricevere informazioni riguardanti un utente associato ad un ID (Figura 4.14).

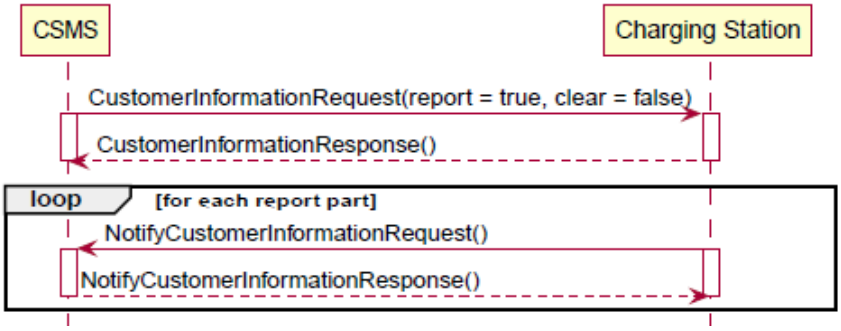

Figura 4.14: Customer Information

Non è chiaro che tipo di informazioni siano, se sono sensibili potrebbero essere a rischio nel caso in cui la comunicazione venisse compromessa.

4.3.3 Considerazioni su parte 4 - JSON over WebSocket

- Per iniziare una connessione websocket la colonnina ha bisogno di un URL a cui connettersi e che è specifico della colonnina (contiene il suo ID). Tuttavia viene raccomandato di non affidarsi al solo URL per identificare una colonnina ma eseguire un doppio controllo con le credenziali di autenticazione (HTTP Basic Authentication).

- Viene ripetuto come Websocket definisca già un Ping Pong frames che viene usato per controllare se l'endpoint è attivo, tuttavia questo meccanismo non copre del tutto le funzionalità di Heartbeat che quindi deve essere implementato.

- Esistono delle topologie in cui la comunicazione tra colonnina e sistema centrale non è end-to-end ma bensì è presente un dispositivo nel mezzo (ad es. proxy, controller) il quale potrebbe essere in grado di leggere e manipolare i dati mandati tra CP e CS. Perciò viene raccomandato di adottare delle misure sufficienti al fine di proteggere l'entità intermedia perché se viene compromessa, allora si aprono numerosi scenari pericolosi compresi gli attacchi fin qui analizzati. In risposta a ciò, viene suggerito l'uso del meccanismo *SignedMessage* su messaggi critici al fine di rilevare se il dispositivo tenta di manipolare i dati.

- Nella sezione 7 viene descritto il meccanismo *SignedMessage* il quale fornisce un metodo che aggiunge al messaggio una digital signature, così facendo si ha la sicurezza che provenga da chi di dovere. Chiaramente non può essere applicato a tutti i messaggi, anche perché se è presente il TLS sarebbe una procedura ridondante oltre che dispendiosa ma può essere utile per alcuni comandi critici e in alcune topologie (local controller).

4.4 Conclusioni

In Tabella 4.1 vengono ripresi gli attacchi analizzati in 1.6 (ci si riferisce all'attacco utilizzando i numeri usati in Tabella 3.1), vengono mappati sulle nuove categorie e vengono proposti possibili rimedi per ognuno di essi utilizzando le nuove funzionalità aggiunte in 2.0.

#	Scenario	Possibili rimedi
1	4.1.1 Attacco al TLS	Dipende dal tipo di cipher suites scelto nel handshake ed in particolare quale metodo di Key Exchange viene utilizzato, infatti, come si è visto, esistono metodi che non permettono un MitM.
2	4.1.2 BootNotification	Un'applicazione corretta del TLS.
3	4.1.2 BootNotification	Un possibile rimedio consiste nell'impostare un monitor sulla variabile **HeartbeatInterval** così facendo una volta che viene modificata il CS viene notificato. Un altro modo potrebbe essere impostando un limite sotto al quale il CP rifiuta la richiesta di modifica.
4	4.1.2 Heartbeat	I rimedi sono gli stessi di cui sopra.
5	4.2.2 Local Authorization	Una possibile soluzione potrebbe essere quella di firmare i messaggi che cancellano LAL e/o cache assicurandosi così che provengano veramente dal CS.
6	4.2.8 MeterValues	Viene aggiunta la funzionalità di firmare questi valori impedendo così che possano essere alterati nel tragitto.

7	4.2.9 Charging Profiles	Visto l'impatto che un profilo di ricarica dannoso può avere, si potrebbe adottare il meccanismo di *SignedMessage*.
8	4.2.4 Inoltro transazione	Il meccanismo risulta molto simile a 1.6 e non sembrano esserci particolari aggiustamenti per non permettere l'inoltro di una transazione ad un secondo CP.
9	4.2.6/4.2.5 Reservation/ Change Availability	Per quanto riguarda la disponibilità il CP è tenuto ad informare il CS quando un connettore cambia il proprio stato, mentre per la prenotazione questo meccanismo viene applicato solo a certe condizioni. Comunque viene raccomandato di autorizzare l'ID usato per la prenotazione.
10	4.2.2 Local Authorization	Anche qui un possibile rimedio consiste sempre nel firmare i messaggi che iniettano liste di autorizzazione locale.
11	4.3.1 UpdateFirmware	La vulnerabilità del processo di aggiornamento del firmware viene totalmente risolta (ad eccezione di alcune topologie: local controller) con la nuova procedura che utilizza certificato e digital signature.
12	4.3.2 Diagnostic	L'unica contromisura plausibile è il meccanismo *SignedMessage*.
13	4.3.1-4.3.2	Si veda 11 e 12.

Tabella 4.1: Possibili rimedi ad attacchi 1.6

In Tabella 4.2 invece vengono riportati dei possibili nuovi attacchi che sfruttano delle vulnerabilità introdotte con alcune nuove procedure di OCPP 2.0.

#	Scenario	Tipo attacco	Entità danneggiate	Effetti
14	4.1.2:**B09-B10**	Modifica o creazione messaggio - Modifica valore variabile	CP - CS	Migrazione ad un CS malevolo
15	4.2.1:**C04**	Brute force PIN-code	Utente	Furto di elettricità
16	4.2.1:**C15**	Modifica valore variabile	Provider	Furto di elettricità
17	4.2.1:**C16**	Modifica valore variabile	CP-Utente	Malfunzionamenti
18	4.2.3:**E05**	Modifica valore variabili	Provider	Furto di elettricità
19	4.2.7:**I01-I04**	Modifica messaggio	CP-Utente	CP inutilizzato
20	4.3.1:**L03**	MitM	CP	Infezione malware

Tabella 4.2: Nuovi attacchi ed effetti

In conclusione, il passo più importante fatto dalla versione 2.0 (e anche dall'estensione a 1.6) è la standardizzazione dell'utilizzo del TLS attraverso i profili di sicurezza. In questo senso la configurazione efficace del TLS può impedire una riuscita di un MitM e quindi di conseguenza la maggior parte degli attacchi non sono attuabili.
Tuttavia, nella nuova versione (ma anche in 1.6) emerge la possibilità di come la comunicazione tra colonnina e sistema centrale non sia più end-to-end e preveda dei dispositivi nel mezzo (ad es. local controller), tuttalpiù vengono discussi casi d'uso specifici per queste topologie e viene sottolineato come sia presente la possibilità che le entità nel mezzo possano leggere e manipolare i dati. Quindi, tutte le componenti presenti nel canale di comunicazione devono essere adeguatamente protette, altrimenti il TLS potrebbe non bastare.
In aggiunta a ciò, nella versione 2.0 vengono introdotti altri strumenti (monitor, messaggi firmati ecc.) i quali possono aiutare l'infrastruttura a rilevare possibili attacchi nel caso in cui il TLS dovesse venire bypassato oppure nel caso in cui un'entità nella comunicazione comunicazione bidirezionale tra CP e CS dovesse venire compromessa.

Capitolo 5

Impersonificazione colonnina

Dopo aver visto, nei capitolo precedenti, gli attacchi al protocollo i quali presupponevano un MitM tra colonnina e sistema centrale, si è pensato a come sfruttare il fatto di essere in grado di impersonificare una colonnina usando la sua identità nell'URL. Nel caso in cui si usi una autenticazione con username e password è necessario esserne in possesso (rubandole oppure ipotizzandole). Se invece viene usata una mutua autenticazione con certificati l'unica possibilità è essere in possesso del certificato della colonnina (poco probabile).

5.1 Attacchi, rischi e possibili contromisure

Riferendosi principalmente alla versione 1.6 di OCPP (più estensione) con un occhio anche a 2.0, si sono prese in esame le operazioni che vengono iniziate dalla colonnina provando ad ideare degli attacchi utilizzando messaggi specifici. Inoltre si sono analizzati i principali rischi e si è tentato di escogitare delle possibili contromisure.

5.1.1 DataTransfer

Si è già visto come la vulnerabilità di questa procedura sia il fatto che il campo **data** non abbia una lunghezza predefinita e quindi si possa incrementare a piacimento. Se è possibile impersonificare delle colonnine, si potrebbe pensare ad un attacco in cui si inviano a ripetizione richieste di *DataTransfer* con un campo **data** arbitrariamente dimensionato. Nella versione 2.0 del protocollo la suddetta funzione è presente senza alcuna modifica, ciò significa che l'attacco sopra descritto è ancora possibile.

Principali rischi

- Un primo rischio è quello di riuscire a generare un DoS o per lo meno una riduzione delle funzionalità del server. Questo dipende da come il sistema centrale processa le informazioni contenute nel messaggio, infatti se il campo **data** ha una dimensione elevata e la logica sottostante prevede un'elaborazione dello stesso, allora è più probabile che il rischio presentato si materializzi.

- Un altro scenario plausibile riguarda la pre-allocazione che un server può fare nel momento in cui la colonnina segnala l'arrivo di un messaggio con una certa dimensione. In questo caso specifico, il campo **data** può avere una dimensione molto elevata e quindi una pre-allocazione delle risorse potrebbe risultare fatale per il sistema centrale in quanto potrebbe saturare le risorse disponibili arrivando a un DoS o una riduzione delle funzionalità.

Contromisura

Una banale contromisura è l'applicazione di un limite al campo **data**, ovviamente essendo la corrente procedura un modo per scambiare delle informazioni non coperte dai messaggi originali del protocollo, è difficile immaginare un limite valido per tutti i casi possibili. Eventualmente se ci fosse la necessità di superare quel limite si potrebbe pensare ad applicare delle misure di sicurezza più elevate come, per esempio, il meccanismo di *SignedMessage* introdotto in 2.0.

5.1.2 MeterValues

La procedura *MeterValues* viene utilizzata dalla colonnina al fine di inviare informazioni riguardanti il proprio contatore. Anche in 2.0 si applica lo stesso procedimento. Un possibile attacco è l'invio di informazioni fasulle al fine di far reagire il CS in un determinato modo. Si può pensare, ad esempio, come tali dati vengano poi usati per implementare una sorta di smart charging che quindi risulterebbe applicato in un modo scorretto.

Principali rischi

- I dati raccolti dal sistema centrale potrebbero essere usati anche da entità terze come DSO e EMS al fine di implementare del smart charging. Ora, se ciò avviene è chiaro come le conseguenze interessino non solo l'infrastruttura di

ricarica elettrica, ma anche altre infrastrutture o applicazioni che sono influenzate da DSO e EMS. Inoltre il rischio è anche quello di impattare direttamente sulla rete elettrica creando instabilità.

- Uno scenario probabilmente più plausibile è quello in cui questi valori vengano usati dal CS per effettuare una sorta di bilanciamento del carico locale, ancora una volta si avrebbe una applicazione sbagliata ed un effetto:
  - sull'esperienza utente ovvero iniettando valori volutamente alti per far si che vengano applicati dei limiti al rate di ricarica rendendola più lenta;
  - sulla rete elettrica ovvero inviando valori bassi al fine di alzare i rate di ricarica i quali potrebbero diventare troppo elevati per la capacità disponibile e portare instabilità nella rete.

Contromisura

Nella versione 2.0 del protocollo viene data la possibilità di firmare i valori ottenuti dal contatore, quindi implementando ciò l'attacco descritto non potrebbe essere realizzato in quanto i messaggi verrebbero rifiutati se non accompagnati da una signature coerente.

5.1.3 Start/Stop Transaction

Le due procedure potrebbero essere usate per creare delle transazioni fittizie inserendo una grande differenza tra **meterStart** e **meterStop**, i quali vengono usati poi per calcolare il consumo di energia per la ricarica e la conseguente fatturazione. È necessario essere a conoscenza di un ID valido e autorizzato alla ricarica ed inoltre se è obbligatorio autorizzare l'ID prima di procedere con la transazione, è fondamentale inviare un messaggio di *Authorize* prima di eseguire l'attacco.
In 2.0 lo scenario è simile, l'unica differenza è l'esistenza di un solo messaggio di transazione (*TransactionEvent*) dove vengono indicati i vari tipi (Started,Updated,Ended).

Principali rischi

Eseguendo l'attacco sopra descritto è quindi possibile creare delle transazioni fittizie a carico di un specifico utente. Inoltre, agendo sui campi **meterStart** e **meterStop** il costo potrebbe risultare cospicuo.

Contromisura

L'unica contromisura plausibile è la protezione degli ID, infatti adottare il meccanismo di *SignedMessage* non sarebbe sostenibile per la grande mole di messaggi relativi alle transazioni che vengono scambiati. Per avere quanto meno una risposta rapida all'attacco sopra descritto, si potrebbe ideare un meccanismo che notifica tempestivamente all'utente di transazioni a suo carico, dandogli la possibilità di segnalare l'accaduto se non riconosce le transazioni.

5.1.4 StatusNotification

Viene usata per informare il CS di un cambiamento di stato oppure di un errore riguardante un connettore. In 2.0 in aggiunta, è presente il messaggio *NotifyEvent* il quale si applica ad uno scenario più generale rispetto ai soli connettori (usato, ad esempio, nella procedura definita dai monitor).
Si può ipotizzare un attacco in cui vengono inviati dei messaggi contenenti uno stato fasullo e quindi forzare il sistema centrale a dover agire di conseguenza.

Principali rischi

- Il segnalamento di un guasto fittizio potrebbe costringere gli operatori a dover attivarsi al fine di risolvere il problema, inoltre se non è risolvibile da remoto potrebbe essere necessario recarsi fisicamente nel luogo.

- Un altro rischio è quello di arrivare ad una inutilizzabilità da remoto della colonnina, infatti questi messaggi possono essere usati per rendere inutilizzabili le componenti della colonnina. Anche se poi la connessione originale viene ripristinata, lo stato persiste non permettendo, per esempio, l'avvio di una transazione remota da parte del CS sulla componente interessata. Inoltre ipotizzando che un provider mostri le disponibilità delle proprie colonnine su un'app, il CP interessato risulterebbe indisponibile e quindi non verrebbe preso in considerazione dall'utente per un'eventuale ricarica.

Contromisura

Anche qui si potrebbe pensare all'applicazione del meccanismo di *SignedMessage*, probabilmente però non è realisticamente applicabile in quanto questi messaggi vengono scambiati spesso tra colonnina e sistema centrale.

5.1.5 Security Events

Nel white paper di miglioramento di 1.6 viene descritta una procedura che notifica al sistema centrale quando un evento critico si verifica. Quindi, è possibile segnalare l'avvenimento di un evento critico fasullo il quale porterebbe il sistema centrale a dover reagire di conseguenza (scenario simile a 5.1.4). Anche in 2.0 la procedura è presente.

Principali rischi

I rischi derivanti dal segnalare eventi di sicurezza fittizi dipendono da come reagisce il CS alle notifiche. Per esempio, si può ipotizzare come per l'evento "rilevata manomissione fisica" si proceda ad inviare delle persone nel luogo fisico.

Contromisura

La contromisura è analoga a quella proposta per 5.1.4, qui però l'applicazione potrebbe essere realistica se si pensa che un evento di sicurezza non capiti così spesso.

5.1.6 Authorize

Al fine di usare la procedura *Authorize* per un attacco, si potrebbe procedere nel seguente modo: si impersonifica la colonnina, si itera l'invio del messaggio *Authorize* con un ID random finché non si riceve in risposta uno status **Accepted** o **ConcurrentTx**, ciò significa che l'ID esiste ed è abilitato alla ricarica.
L'attacco si presenta come una sorta di brute-force sull'ID, se quest'ultimo è composto da molti caratteri diventa chiaramente molto difficile trovarne uno esistente in un tempo ragionevole. In 2.0, però si definiscono alcuni identificatori come il PIN-code che hanno una lunghezza ridotta e quindi l'attacco potrebbe riuscire facilmente.

Principali rischi

Il fatto di avere un metodo per trovare ID esistenti può essere usato in due scenari:

- al fine di compiere un furto di elettricità a carico dell'utente associato all'ID;
- oppure usare l'ID trovato per portare a termine l'attacco descritto in 5.1.3.

Contromisura

Oltre all'applicazione del meccanismo di *SignedMessage*, qui si potrebbe pensare ad un time-out tra una richiesta di *Authorize* ed un'altra.

5.1.7 Heartbeat

Lo scenario è simile a 5.1.1 ovvero si può iterare l'invio del messaggio di Heartbeat al fine di provocare un DoS o una riduzione delle funzionalità del sistema centrale. In realtà l'attacco può essere portato a termine con qualsiasi messaggio che viene mandato dal CP, il più efficace probabilmente resta *DataTransfer*.

Contromisura

Ovviamente non è possibile applicare il meccanismo di *SignedMessage* a tutti i messaggi. Se si pensa ad un scenario in cui il CS riceve lo stesso messaggio un gran numero di volte dalla stessa colonnina in un lasso breve di tempo, si potrebbe applicare un time-out alla comunicazione o bloccare quel specifico messaggio.

In tabella 5.1 vengono riassunti gli attacchi descritti ed i possibili effetti.

Scenario	Entità danneggiate	Effetti
5.1.1 - 5.1.7	CS	DoS o riduzione delle funzionalità (maggior tempo necessario affinché il CS risponda)
5.1.2	CS - rete elettrica - terze parti (DSO,EMS)	Informazioni fasulle potrebbero portare ad applicazioni di smart charging sbagliate
5.1.3	Utente	Carico transazioni fittizie a spese dell'utente
5.1.4	CS - Utenti	Segnalazione guasti fittizi oppure inutilizzabilità da remoto
5.1.5	CS	Dipende da come reagisce il CS a determinati eventi
5.1.6	Utente	Furto di elettricità e/o carico transazioni fittizie

Tabella 5.1: Attacchi ed effetti

5.2 Test

Si è utilizzato l'ambiente di simulazione (descritto nella sezione 3.6.1) a disposizione per provare ad impersonificare una colonnina conoscendo l'identità della stessa.
Nel simulatore è presente solo l'autenticazione tramite identità nell'URL, se si prova ad avviare una connessione WebSocket con il sistema centrale utilizzando un ID che ha già una connessione attiva non si ricevono errori e viene instaurato il canale di comunicazione. Tuttavia la connessione originale viene chiusa, quindi:

1. non è possibile avere due canali di comunicazione attivi con la stessa identità;

2. non si ricevono messaggi di errore, si ha solo una chiusura della connessione originale.

Perciò sembra possibile impersonificare una colonnina.
Un rimedio parziale potrebbe essere il controllo da parte del sistema centrale sulle connessioni già attive e non permettere l'avvio di comunicazioni già presenti. Ciò, per lo meno limiterebbe temporalmente l'attacco nei momenti in cui la connessione tra colonnina e sistema centrale è attiva.
Dopo aver dimostrato la fattibilità dell'impersonificazione, si è provato a mettere in atto gli attacchi descritti nella sezione precedente. I test hanno riguardato solamente gli scenari 5.1.1 e 5.1.7 in quanto sono gli unici in cui è possibile valutarne la riuscita e le conseguenze sul simulatore; infatti per gli altri attacchi non si è in grado di far ciò perché:

1. per alcuni dovrebbe essere presente una logica sottostante (ad es. 5.1.6 si realizza se il CS risponde adeguatamente avendo memorizzato gli ID validi);

2. per altri dipende dalle scelte prese in sede di implementazione nel momento in cui accade un determinato evento (ad es. 5.1.5);

3. per altri ancora dipende dal tipo di uso di certi dati (ad es. 5.1.2).

In particolare per i punti 2-3 gli effetti potrebbero variare da provider a provider in quanto potrebbero essere operate scelte differenti e anche un uso diverso dei dati a disposizione.

5.2.1 Setup

Al fine di testare gli scenari 5.1.1 e 5.1.7 è stato ideato un breve script in JavaScript al fine di automatizzare il processo di avvio di una connessione WebSocket con il sistema centrale e invio del messaggio, permettendo così la possibilità di impersonificare un gran numero di colonnine.

```
1  var text1 = "wss://ocpp16.grupposigla.it/ocppj/";
2  var text2 = process.argv[2];
3  var url = text1.concat(text2);
4
5  console.log(`${url}`);
6
7  var WebSocket = require('ws');
8  var socket = new WebSocket(url, ["ocpp1.6", "TestSimulatorKey"]);
9
10
11 socket.onopen = function(e) {
12   console.log("[open] Connection established");
13 };
14
15 socket.onmessage = function(event) {
16   console.log(`[Server response]: ${event.data}`);
17 };
18
19 var message = "[2,\"wgUXzZgfUMheNR2juBzmnFhC4xmGtC1bIN3D\",\"
     Heartbeat\",{\"currentTime\":\"2022-01-19T09:56:39.977Z\"}]";
20 sendHeartbeat();
21
22 async function sendHeartbeat (){
23   while(1){
24     await new Promise(r => setTimeout(r, 1000));
25     socket.send(message);
26     console.log("Send Heartbeat");
27   }
28 }
```

Listing 5.1: Connessione e invio messaggio

Le prime tre righe definiscono l'URL completo a cui connettersi composto anche dall'identità della colonnina che si vuole impersonificare (parametro passato da riga di comando), successivamente si procede alla connessione al server (righe 7-13) e viene definita una funzione che mostra i messaggi che arrivano (righe 15-17). A questo punto si inizializza una variabile la quale contiene il messaggio da inviare

(riga 19), infine si definisce una funzione che itera l'invio del messaggio ogni secondo (righe 22-28).
L'esempio 5.1 fa riferimento all'attacco utilizzando lo scenario 5.1.7, per lo scenario 5.1.1 l'unica modifica necessaria è il messaggio (riga 19).
In figura 5.1 viene mostrato un esempio di esecuzione: la connessione viene stabilita, dopo di che si procede all'invio del messaggio e si riceve la risposta dal server.

```
C:\Users\Signo\Desktop\MasterThesis>node imp_col.js cp1
wss://ocpp16.grupposigla.it/ocppj/cp1
[open] Connection established
Send Heartbeat
[Server response]: [4,"wgUXzZgfUMheNR2juBzmnFhC4xmGtC1bIN3D","InternalError","Exception of type
'RepowerAPI.Exceptions.GeneralApiException' was thrown.",{}]
```

Figura 5.1: Esecuzione impersonificazione colonnina

5.2.2 Risultati

Si sono impersonificate 500 colonnine le quali iterano l'invio del messaggio *DataTransfer* oppure *Heartbeat* ogni secondo.
A questo punto si è provato a: inizializzare un'altra connessione WebSocket e provare l'invio di messaggi diretti al sistema centrale al fine di vedere se si notava almeno un maggior tempo di risposta da parte del server.
Sia per *DataTransfer* che per *Heartbeat* non si nota alcun tipo di riduzione delle funzionalità del server, una spiegazione a ciò potrebbe essere la mancanza di un vero processamento del messaggio da parte del sistema centrale; infatti nel simulatore il server risponde semplicemente con un messaggio di errore (dovuto alla mancanza della logica sottostante).
Quindi, i tentativi di provocare un DoS sul server falliscono ma ciò era preventivabile. Infatti, sarebbe peculiare riuscire a provocare un DoS con solo centinaia di messaggi al secondo senza una computazione degli stessi.
Alla luce di ciò, anche per i due scenari in esame vale quanto detto in precedenza per gli altri ovvero le conseguenze a questi tipi di attacco dipendono dalle scelte implementative operate dai provider. Si pensi, per esempio, allo scenario descritto in 5.1.1 in cui il server effettua una pre-allocazione delle risorse per prepararsi all'arrivo di un messaggio di una certa dimensione; se ciò avviene allora effettivamente l'attacco potrebbe riuscire nel suo intento.
In conclusione, anche se non si è riusciti a verificare le conseguenze degli attacchi, la possibilità per un attaccante di riuscire ad impersonificare una colonnina è pericolosa pure solo pensando al fatto che la connessione originale viene chiusa e quindi si avrebbe la presenza di una colonnina fasulla con tutti gli scenari che si aprono.

Si è già parlato di una possibile contromisura (verificare le connessioni già attive), tuttavia una soluzione ancor più efficace è chiaramente l'uso di un certificato lato colonnina (già previsto in 2.0).

Capitolo 6

Lavori futuri e conclusioni

Il focus principale di questa tesi è stato sulla comunicazione tra colonnina e sistema centrale e conseguentemente sul protocollo di riferimento (OCPP). Tuttavia come si è visto nel primo capitolo, sono diversi gli attori che entrano in gioco nell'infrastruttura di ricarica elettrica e di conseguenza più entità sono presenti e sono collegate tra di loro, più aumentano le superfici di attacco.
Perciò, si evidenziano alcune tematiche emerse durante le varie analisi effettuate, ma che non sono state approfondite nel dettaglio nel corso del lavoro di tesi, auspicando che possano essere di aiuto oppure uno spunto per lavori futuri.
Infine vengono presentate le conclusioni del lavoro di tesi.

6.1 Lavori futuri

6.1.1 Roaming

Nella sezione 2.1.4 si è già visto cosa si intenda per roaming, quali siano i vari ruoli ed il protocollo standard usato in questo contesto (OCPI).
Nelle varie analisi è emerso il seguente scenario: si ipotizzi due provider A e B che implementano il roaming tra di loro, ora se A è affetto da una vulnerabilità, la stessa può essere usata come punto di accesso a B. Per poter contestualizzare meglio questo scenario si parte da OCPI ed in particolare da due moduli specifici del protocollo ovvero **Commands** e **ChargingProfiles**, i quali permettono di inoltrare dei comandi ad una specifica colonnina ed addirittura di inviare dei profili di ricarica. In Figura 6.1 viene raffigurato il flusso che segue un comando di OCPI dove il passaggio tra CPO e colonnina è gestito con OCPP.

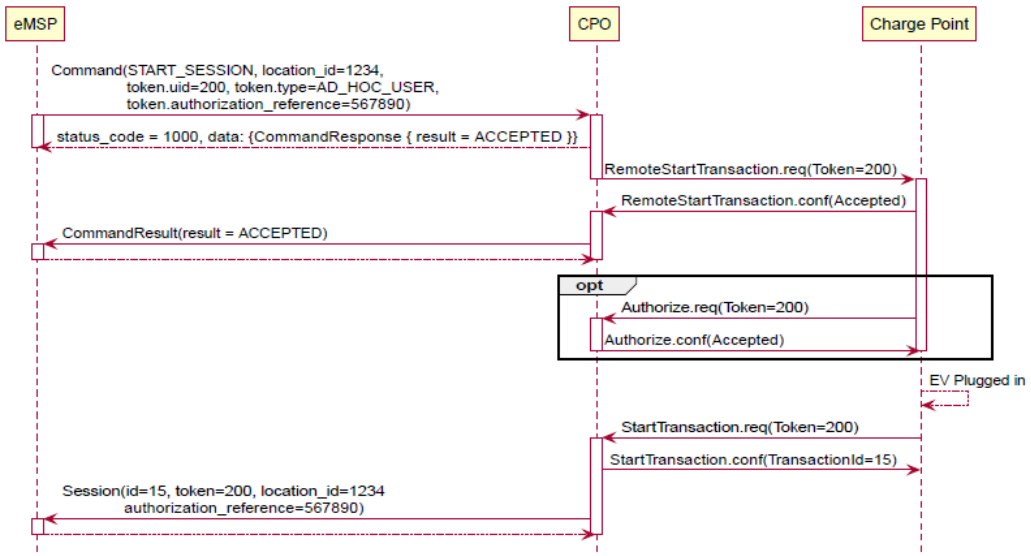

Figura 6.1: Flusso START_SESSION

Ora, eMSP è il provider A affetto da una vulnerabilità la quale permette ad un attaccante di riuscire ad inviare comandi o ChargingProfiles tramite OCPI, mentre CPO è il provider B che riceve i messaggi pensando arrivino da A. Quindi se questo scenario dovesse realizzarsi, è chiaro come il roaming aggiunga una nuova superficie di attacco: infatti un attaccante potrebbe non solo essere in grado di influenzare il comportamento di una colonnina, ma anche di inserire dei ChargingProfiles che come si è già visto sono molto delicati in quanto hanno un impatto diretto sulla rete elettrica.

Inoltre sarebbe interessante anche una analisi sul protocollo OCPI al fine di valutarne possibili exploit (come fatto per OCPP nei capitoli 3 e 4), i quali potrebbero essere usati allo scopo di eseguire un MitM e quindi controllare la comunicazione tra A e B.

In conclusione, si è visto come il roaming aggiunga una funzionalità molto interessante e che potrebbe migliorare di molto l'esperienza utente. Di contro però, introduce innanzitutto un nuovo protocollo che potrebbe avere delle vulnerabilità e soprattutto mette in collegamento provider i quali potrebbero aver operato scelte diverse e con vari livelli di sicurezza in sede implementativa, aprendo così alla possibilità che una vulnerabilità di uno diventi un punto di accesso agli altri.

6.1.2 Smart Charging

Nei capitoli precedenti si è parlato spesso di Smart Charging e si è visto come possa essere implementato in due scenari: il primo interno all'infrastruttura ovvero il sistema centrale che direttamente impone dei parametri sulle varie ricariche. Nel secondo entrano in gioco entità terze come DSO e EMS, si è già visto nella sezione 2.1.5 le differenze tra le due e come agiscono.
Anche qui, similmente al roaming, si possono immaginare due possibili casi di studio:

1. Il primo riguarda un'analisi dei vari protocolli utilizzati nelle comunicazioni tra l'infrastruttura e le terze parti interessate. Questo allo scopo di rilevare potenziali vulnerabilità che permettano ad un attaccante di influenzare la comunicazione e quindi anche lo smart charging, portando a galla una serie di problematiche già analizzate (ad es. instabilità rete elettrica).

2. Il secondo è similare alla problematica sollevata per il roaming ovvero possibili vulnerabilità presenti in DSO e EMS potrebbero essere usate al fine di danneggiare l'infrastruttura di ricarica elettrica. Inoltre, quest'ultima gestirà in futuro carichi sempre più importanti di energia e potrebbe essere usata al fine provocare danni alla stessa rete elettrica, quindi non essere il bersaglio principale ma uno strumento per un attacco.

6.1.3 Hardware colonnina

Durante le varie analisi si è visto come alcuni attacchi mirano a provocare un DoS o per lo meno una riduzione delle funzionalità della colonnina. Sarebbe interessante uno studio su che tipo di hardware è presente nelle colonnine (ovviamente può differire da produttore a produttore) e capire se attacchi tipo quelli visti in sezione 3.1.2 in cui si riduce al minimo l'intervallo tra un heartbeat ed un altro, sono sufficienti a provocare dei malfunzionamenti sulle colonnine.
Se durante la produzione vengono applicati i requisiti stilati da ENCS (riassunti nel capitolo 2), allora le colonnine non dovrebbero aver particolari problemi anche nel caso in cui siano costrette a inviare ripetutamente dei messaggi. Ovviamente, se ciò non avviene e le colonnine vengono dotate di un hardware scadente, è semplice provocare malfunzionamenti e disservizi.

6.2 Conclusioni

Come detto all'inizio, l'attenzione della tesi è rivolta alle interazioni tra colonnina e sistema centrale. In particolare, si è vista l'analisi di due versioni del protocollo di riferimento allo scopo di riuscire ad intervenire e manipolare i messaggi scambiati tra le due entità.
Per la versione 1.6 di OCPP, l'analisi dei singoli messaggi ha portato ad ideare degli attacchi con lo scopo di danneggiare l'infrastruttura e non solo; la realizzazione degli scenari descritti è però legata alla riuscita di un MitM. Si è visto infatti come sia necessario essere in grado di intervenire nel canale di comunicazione bilaterale tra colonnina e sistema centrale: intercettando i messaggi e modificandoli oppure essere in grado di crearne di nuovi. Questo dipende da più fattori, se si da' per assodato che venga usato il TLS per proteggere le comunicazioni OCPP (anche se nella specifica 1.6 viene solo suggerito l'uso del protocollo), si deve rivolgere uno sguardo a come il TLS viene applicato e anche alla topologia dell'infrastruttura stessa. Infatti, esistono due contesti in cui il protocollo di sicurezza potrebbe venire compromesso:

- il primo riguarda direttamente come il TLS viene applicato ovvero quali metodi vengono usati. Come si è visto nei test (sezione 3.6.2), esistono configurazioni del protocollo che se usate impediscono attacchi MitM grazie all'uso di certificati e garantiscono anche PFS. Quindi, si può affermare come la protezione delle comunicazioni dipenda dalla scelta dei metodi da utilizzare.

- Il secondo è rivolto alla topologia con cui si articola l'infrastruttura ed in particolare la comunicazione tra colonnina e sistema centrale. Infatti se, per esempio, si prende la topologia con il local controller (Figura 2.3), si ha un'entità in mezzo la quale necessariamente può vedere i messaggi e manipolarli. Questo aggiunge una nuova superficie di attacco anche se il TLS viene applicato correttamente, tanto è vero che se si riesce a compromettere il local controller tutti i messaggi che passano per di esso potrebbero risultare compromessi.

Alla luce di ciò si può affermare che se il TLS viene applicato correttamente e la comunicazione è end-to-end tra colonnina e sistema centrale, allora si ha una buona protezione della comunicazione e si riduce drasticamente la possibilità di portare a termine gli attacchi analizzati.
Nonostante ciò, nella versione 1.6 sono presenti procedure le quali lasciano molte perplessità come, per esempio, la procedura di aggiornamento del firmware (dove non viene controllato né l'integrità né la sorgente). Difatti, sarebbe opportuno che il protocollo adottasse delle misure di sicurezza al di fuori del TLS.

A tal proposito, nella versione 2.0 di OCPP, vengono introdotti nuovi strumenti i quali sembrano essere un rimedio ad alcuni attacchi ideati in 1.6 (ad es. la nuova procedura di aggiornamento del firmware con certificato e controllo della firma oppure la possibilità di inserire dei monitor sulle variabili). Di contro però, si è visto come nuove funzionalità possano portare a nuove vulnerabilità e quindi ad ideare nuovi attacchi (Tabella 4.2). È corretto far notare come per l'analisi del protocollo 2.0 non si possano avere dei riscontri in quanto in letteratura non sono presenti articoli che trattano la nuova versione, al contrario di 1.6 in cui l'analisi è stata basata su un paper specifico. Nel bilancio totale, i passi in avanti fatti da 2.0 in ambito di sicurezza sono diversi ed il primo fra tutti è sicuramente aver standardizzato l'uso del TLS attraverso la definizione di tre profili di sicurezza.
Tuttavia, la versione più utilizzata resta 1.6 ma fortunatamente esiste un white paper in cui vengono estese alla versione 1.6 alcune novità di sicurezza presentate in 2.0 (sezione 3.7). Grazie a questo anche la versione precedente viene resa più sicura.
Nei capitoli 3 e 4 ci si è concentrati su scenari in cui si prevedeva un MitM, nel capitolo 5, invece, si è provato ad immaginare un diverso contesto. In particolare, si è analizzata la possibilità di impersonificare una colonnina al fine di interagire con il sistema centrale e cercare in qualche modo di danneggiare l'infrastruttura. Si è visto come questa possibilità sia concreta e risulti pericolosa, anche se le conseguenze dipendono dalle scelte implementative esercitate dagli operatori dell'infrastruttura. Un rimedio a ciò è portato direttamente da 2.0 con il terzo profilo di sicurezza in cui si ha una mutua autenticazione con certificati, quindi un'infrastruttura che adotta questo profilo sarebbe immune ai tentativi di impersonificazione di una colonnina.
In conclusione, si può dire come l'infrastruttura di ricarica elettrica presenti non pochi problemi per quanto riguarda la sicurezza, anche alla luce di quanto detto nel corrente capitolo sulle altre entità che operano nel contesto in analisi. Quindi, un investimento sulla produzione security-by-design di questo tipo di infrastrutture è assolutamente necessario, tanto più che come visto nel capitolo 2 esistono dei documenti i quali descrivono i requisiti da adottare. Risulta ancor più chiaro come sia importante focalizzarsi sulle tematiche di sicurezza se si immagina che le infrastrutture di ricarica saranno sempre più presenti negli anni a venire, perciò anche vulnerabilità ora poco rilevanti potrebbero creare veri e propri disastri in un futuro prossimo. Tanto è vero che l'infrastruttura in analisi impatta direttamente sulla rete elettrica e quindi in numerose altre applicazioni e infrastrutture.

Bibliografia

[1] Samrat Acharya et al. «Cybersecurity of Smart Electric Vehicle Charging: A Power Grid Perspective». In: *IEEE Access* 8 (2020), pp. 214434–214453. DOI: 10.1109/ACCESS.2020.3041074.

[2] Cristina Alcaraz, Javier Lopez e Stephen Wolthusen. «OCPP Protocol: Security Threats and Challenges». In: *IEEE Transactions on Smart Grid* 8.5 (2017), pp. 2452–2459. DOI: 10.1109/TSG.2017.2669647.

[3] ECRYPT – CSA. *Algorithms, Key Size and Protocols Report*. Rapp. tecn. 28 Feb. 2018. URL: https://www.ecrypt.eu.org/csa/documents/D5.4-FinalAlgKeySizeProt.pdf.

[4] ElaadNL-ENCS. *Security architecture for electric vehicle charging infrastructure*. 24 Dic. 2019. URL: https://encs.eu/resources/security-requirements/.

[5] ElaadNL-ENCS. *Security requirements for procuring EV charging stations*. 24 Dic. 2019. URL: https://encs.eu/resources/security-requirements/.

[6] EVRoaming Foundation. *Open Charge Point Interface 2.2.1*. Rapp. tecn. 12 Giu. 2020. URL: https://evroaming.org/downloads/.

[7] Politecnico di Milano. *SMART MOBILITY REPORT 2021*. 1 Ott. 2021. URL: https://www.missionline.it/wp-content/uploads/2021/10/Smart-Mobility-Report-2021.pdf.

[8] OCA. *Improved security for OCPP 1.6-J. edition 2 FINAL*. Rapp. tecn. 31 Mar. 2020. URL: https://www.openchargealliance.org/protocols/ocpp-16/.

[9] OCA. *Open Charge Point Protocol 1.6. edition 2 FINAL*. Rapp. tecn. 28 Set. 2017. URL: https://www.openchargealliance.org/downloads/.

[10] OCA. *Open Charge Point Protocol 2.0.1. FINAL*. Rapp. tecn. 31 Mar. 2020. URL: https://www.openchargealliance.org/downloads/.

[11] OCA. *Open Smart Charging Protocol 2.0. FINAL*. Rapp. tecn. 12 Ott. 2020. URL: https://www.openchargealliance.org/downloads/.

[12] *Open protocols.* URL: https://www.greenflux.com/spotlights/open-protocols/.

[13] *Schneider Electric EVLink Parking.* URL: https://www.cisa.gov/uscert/ics/advisories/ICSA-19-031-01.

[14] *Secure use of communications and protocols at charging stations.* URL: https://www.incibe-cert.es/en/blog/secure-use-communications-and-protocols-charging-stations.

[15] *Smart car chargers. Plug-n-play for hackers?* URL: https://www.pentestpartners.com/security-blog/smart-car-chargers-plug-n-play-for-hackers.