UNIVERSITÀ DEGLI STUDI DI PADOVA
FACOLTÀ DI INGEGNERIA

Dipartimento di Ingegneria dell'Informazione

TESI DI LAUREA TRIENNALE
INGEGNERIA MECCANICA E MECCATRONICA
Curriculum Meccatronica

Analisi della Security di OPC UA per applicazioni nell'Industria 4.0

RELATORE: Prof. Stefano Vitturi
Co-RELATORE: Prof. Alberto Morato

LAUREANDO: Giuseppe Migala

Padova, 20 marzo 2022 Anno Accademico 2021/2022

Abstract - OPC UA è uno dei primi protocolli di comunicazione standardizzato pensato appositamente per soddisfare le crescenti esigenze di sicurezza delle implementazioni industriali. Tuttavia, una corretta e completa configurazione, richiede un'ampia conoscenza di tutte le opzioni disponibili. Pertanto, una valutazione delle configurazioni di sicurezza delle implementazioni di OPC UA è necessaria al fine di garantire un funzionamento sicuro, preservando riservatezza e integrità dei dati nei processi industriali.

In questo documento sono stati evidenziati gli strumenti e i metodi principali a capire come effettuare un'analisi della security delle implementazioni OPC UA riportando i risultati ottenuti.

In particolare, si presenta in prima istanza le caratteristiche e la struttura del protocollo OPC UA e successivamente si valutano le impostazioni di sicurezza che il protocollo mette a disposizione, evidenziando le vulnerabilità e i rischi di una "cattiva" configurazione dei dispositivi. A tal fine, si utilizzano delle estensioni che si integrano con il popolare Framework Metasploit, che contengono dei metodi per scoprire i server OPC UA, testarne l'autenticazione e ottenere le loro configurazioni (di sicurezza) in modo da identificare potenziali vettori di attacco.

Sommario

Capitolo 1

1. Introduzione

Il progresso tecnologico e l'avvento dell'Industria 4.0 ha spinto il mondo industriale verso l'automazione dei processi, introducendo nelle fabbriche un numero sempre crescente di macchine, computer, robot e sistemi di controllo. Questo ha portato ad un aumento dei dispositivi interconnessi negli impianti e dei dati raccolti che hanno bisogno di essere memorizzati ed elaborati.

Spesso, negli ambienti industriali, i dispositivi sono tutti eterogenei tra loro perché di produttori diversi; quindi risulta difficile mettere in comunicazione questi sistemi, soprattutto se vi è la necessità di farli comunicare attraverso internet che utilizza protocolli totalmente differenti.

È necessario quindi un protocollo platform-indipendent e altamente scalabile, che sia in grado di fornire un modello standard per lo scambio delle informazioni, dal bus di campo fino al cloud.

La sicurezza dev'essere un altro requisito. La possibilità di poter scambiare dati attraverso internet apre le porte a nuove minacce, di tipo informatico, che necessitano attenzioni da parte delle aziende; soprattutto se impattano in maniera significativa su aspetti reputazionali, economici e legali. È essenziale, quindi, garantire Confidenzialità, Autenticità e Integrità dei dati scambiati.

Infine, è indispensabile che questo protocollo sia facilmente implementabile e che richieda il minor effort possibile per estenderlo a nuove tecnologie e soluzioni.

Un ottimo candidato per soddisfare questi requisiti è OPC UA (Open Platform Communication, Unified Architecture), un protocollo relativamente nuovo, open source, che consente una comunicazione standardizzata e sicura a tutti i livelli.

Nella presente tesi, si espongono le caratteristiche principali di OPC UA e si valutano alcune delle features di sicurezza che il protocollo mette a disposizione, analizzando le informazioni che un server, con una configurazione standard, espone lato client.

Capitolo 2

2. Protocollo OPC UA

2.1. Requisiti

Il protocollo OPC UA (Open Platform Communications Unified Architecture) nasce con l'intento di creare e definire interfacce Client-Server per l'accesso ai dati di processo, permettendo così una semplice e completa integrazione dei vari dispositivi con funzioni di automazione prodotti da differenti produttori e offrendo dei meccanismi per l'accesso remoto. Questo protocollo è applicabile a componenti in tutti i domini industriali, come sensori e attuatori, sistemi di controllo, Manufacturing Execution Systems (MES) e Enterprise Resource Planning Systems (ERP), inclusi Industrial Internet of Things (IIoT), Machine To Machine (M2M) e Industry 4.0.

Questi sistemi hanno lo scopo di scambiare informazioni e/o controllare i processi produttivi e OPC UA definisce un modello di infrastruttura comune e orientato agli oggetti per facilitare questo tipo di operazioni.

Le caratteristiche, relative alla comunicazione e al modello dati, di OPC UA sono riportate in Tabella 1.

Comunicazione tra sistemi distribuiti	Modello Dati
Robustezza e fault-tolerance	Comune per tutti gli OPC data
Indipendenza dalla piattaforma	Orientato agli oggetti
Scalabilità	Tipi estensibili
Alte performance	Metadati
Internet e Firewall	Dati complessi e metodi
Sicurezza e controllo degli accessi	Scalabilità da modelli semplici a complessi
Interoperabilità	Modello di base astratto
Ridondanza dei server	Base per altri modelli dati standard

Tabella 1 - Requisiti OPC UA

OPC è oggi utilizzato come interfaccia di sistema, pertanto, l'affidabilità per la comunicazione tra sistemi distribuiti è molto importante. Poiché la comunicazione di rete non è affidabile per definizione, robustezza e tolleranza ai guasti sono requisiti importanti, inclusa la ridondanza per un'elevata disponibilità.

L'indipendenza dalla piattaforma e la scalabilità sono necessarie per poter integrare le interfacce OPC direttamente nei sistemi in esecuzione su molte piattaforme diverse. Un altro requisito importante è l'elevata performance negli ambienti intranet, come anche la possibilità di comunicazione Internet tramite firewall, il che rende la sicurezza e il controllo degli accessi di fondamentale importanza. L'interoperabilità tra sistemi di diversi fornitori è il requisito più importante.

2.2. Struttura

Per garantire i vari requisiti che si prefissa, lo standard OPC UA è strutturato a strati (Figura 1). I due pilastri fondamentali sono il livello *Transport* e il *Meta Model*.

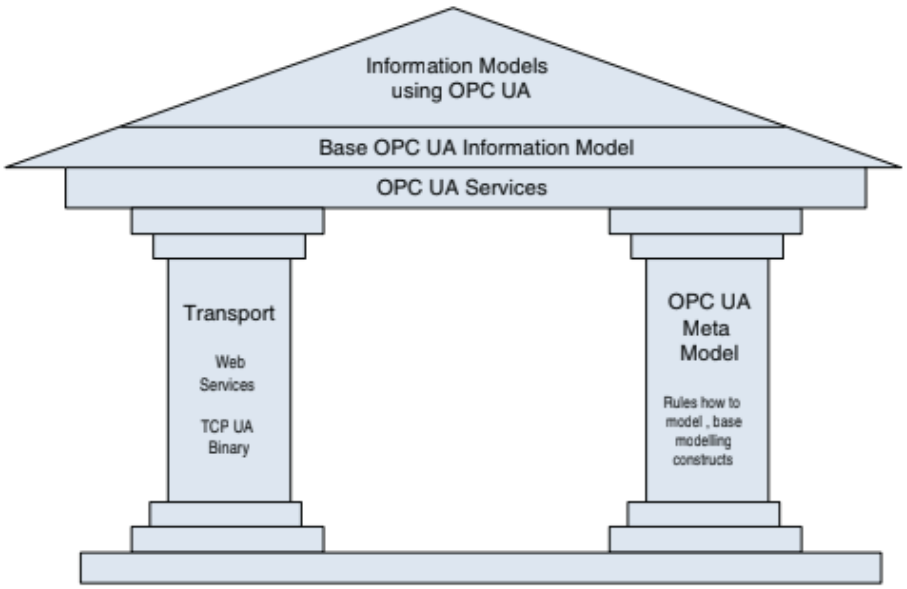

Figura 1 - The OPC UA Foundation

Il livello *Transport* definisce due diversi meccanismi per la trasmissione, ottimizzati per i diversi casi d'uso:

1. Binary TCP, ottimizzato per comunicazioni intranet ad alte prestazioni (il dato viene codificato con una serie di numeri esadecimali ed incapsulato all'interno di un frame TCP);
2. Web Services, per una comunicazione Internet Firewall-Friendly (il dato viene incapsulato all'interno di altri protocolli, ad esempio HTTP, HTTPS, XML, JSON, ecc, comunemente usati per la navigazione Web).

Entrambi utilizzano lo stesso modello di sicurezza message-based utilizzato nei Web Services.

L'OPC UA Meta Model, invece, definisce le regole necessarie a rappresentare le informazioni all'interno di un dispositivo che implementa OPC UA. In questo modo, qualunque sia la tipologia di dispositivo e/o fornitore, si ha un metodo standardizzato per scambiare dati e informazioni.

Secondo il Meta Model, le informazioni devono essere rappresentate da un oggetto composto da nodi organizzati in una struttura gerarchica ben definita.

Nella definizione strutturata di OPC UA i servizi OPC (OPC UA services) costituiscono interfacce tra server e client, definiti distintamente come "fornitori" di modelli informativi e "consumatori" di tali modelli. La comunicazione tra queste due entità avviene tramite i meccanismi di trasporto per lo scambio dei dati. Un client può accedere alla più piccola porzione di dati di un sistema complesso senza che questo sia a conoscenza di come è strutturato l'intero sistema informativo.

La suddivisione del modello informativo è strutturata in 4 sezioni (Figura 2).

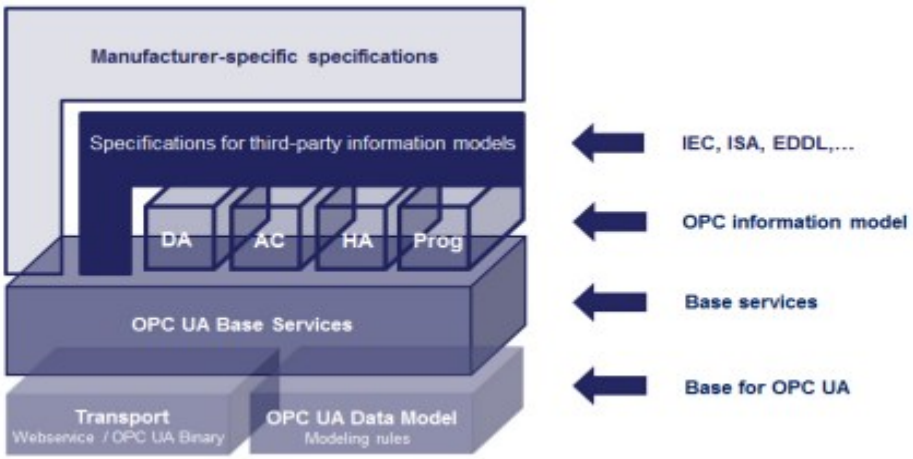

Figura 2 - OPC UA Layered Architecture

1. **DA:** *(Data Access)* definisce estensioni specifiche dei dati come la modellazione di dati analogici o digitali e come esporre la qualità del servizio.
2. **AC:** *(Alarm e Conditions)* specifica un modello avanzato per la gestione degli allarmi di processo e il monitoraggio delle condizioni.
3. **HA:** *(Historical Access)* definisce i meccanismi per accedere ai dati e agli eventi storici.
4. **PROG:** *(Programs)* specifica un meccanismo per avviare, manipolare e monitorare l'esecuzione dei programmi.

Un'applicazione OPC UA è un sistema che ha la funzione di esporre dati e al contempo contiene sia le sue funzionalità specifiche che il loro mapping attraverso l'OPC UA Stack (Figura 3) che implementa diversi meccanismi di trasporto ed è diviso in tre strati come segue:

• *MESSAGE SERIALIZATION*: definisce i metodi per serializzare i dati scambiati in modalità binaria.

• *MESSAGE SECURITY*: specifica i criteri di protezione dei messaggi secondo le regole dettate dalla sicurezza dei Web Services.

• *MESSAGE TRANSPORT*: definisce il protocollo di rete utilizzato, che può essere UA TCP o HTTPs/HTTP per i Web Service.

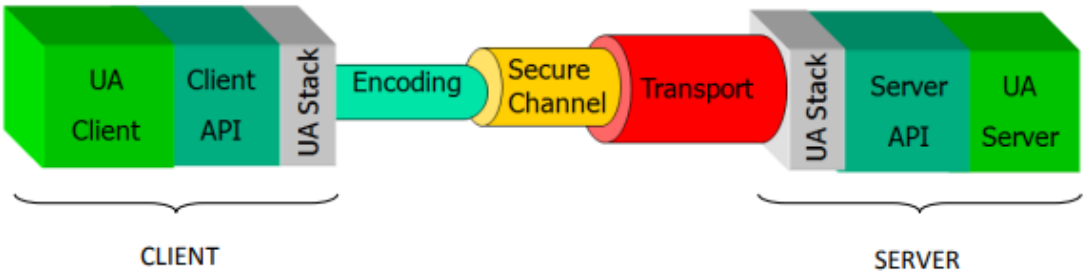

Figura 3 - The OPC UA Stack

2.3. Address Space Model

I concetti base dell'Information Modelling in OPC UA sono i *Nodes* (nodi) e le *References* (collegamenti); questi due elementi vengono raggruppati nell'*AddressSpace* che definisce il set di informazioni esposte lato Server OPC UA e accessibile da qualunque Client OPC UA.

2.3.1. Object Model

L'obiettivo principale di OPC UA AddressSpace è fornire un modo standard per i server di rappresentare gli oggetti ai client. OPC UA definisce gli oggetti in termini di variabili e metodi e consente inoltre di esprimere relazioni con altri oggetti.

Gli elementi di questo modello sono rappresentati nell'AddressSpace come Nodes e ogni Node è assegnato a una NodeClass e ogni NodeClass rappresenta un elemento diverso dell'Object Model (Figura 4).

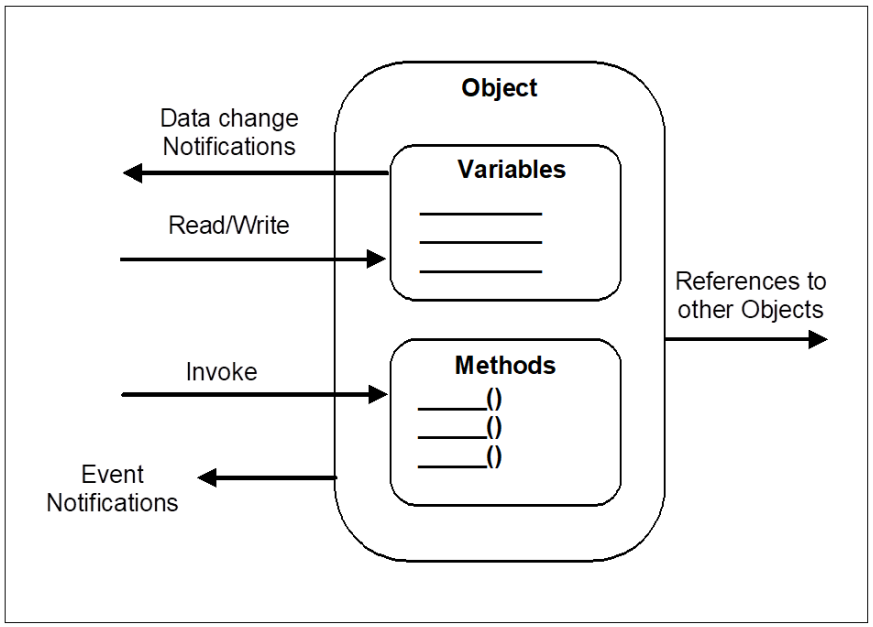

Figura 4 - OPC UA Object Model

2.3.2. Node Model

Gli oggetti e i loro componenti sono rappresentati nell'AddressSpace come un insieme di Nodes descritti da Attributes e interconnessi da References. Ogni Node è un'istanza di una NodeClasses.

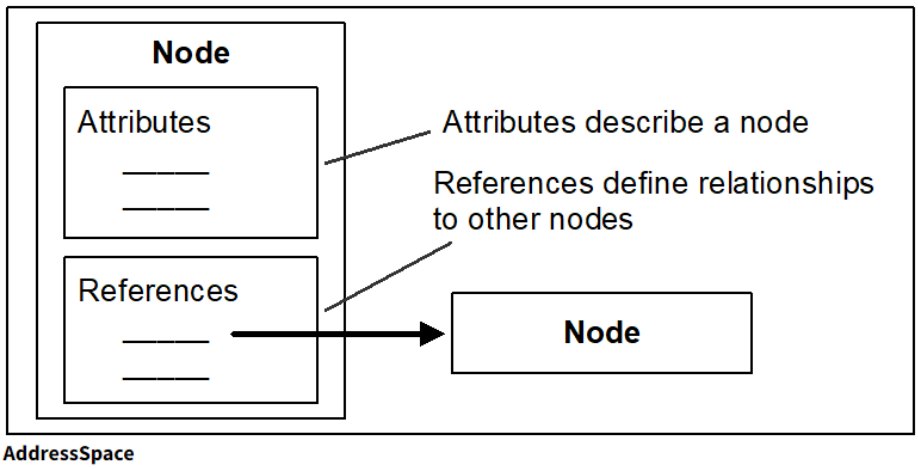

Figura 5 - OPC UA Node Model

Gli Attributes sono dati che servono a descrivere un Node e i client possono accedervi utilizzando i servizi di Read, Write, Query e Subscription/MonitoredItem. Ogni Attribute consiste in un ID attributo, un nome, una descrizione, un tipo di dati e un indicatore obbligatorio/opzionale. Quest'ultimo indica se l'Attribute deve essere istanziato o meno.

Una Reference serve a descrivere una relazione tra due nodi e viene identificata tramite un nodo sorgente, un nodo target, una ReferenceType e una direzione. Il collegamento tra Reference e Node è univoco. Si deduce quindi che una Reference può puntare a un solo Node e un Node può essere puntato da una sola Reference.

Una Reference contiene:

1. NodeId del nodo a cui punta.
2. OPC UA Server del nodo puntato.
3. Il ReferenceType, che rappresenta la semantica della Reference.
4. La direzione della Reference in questione.

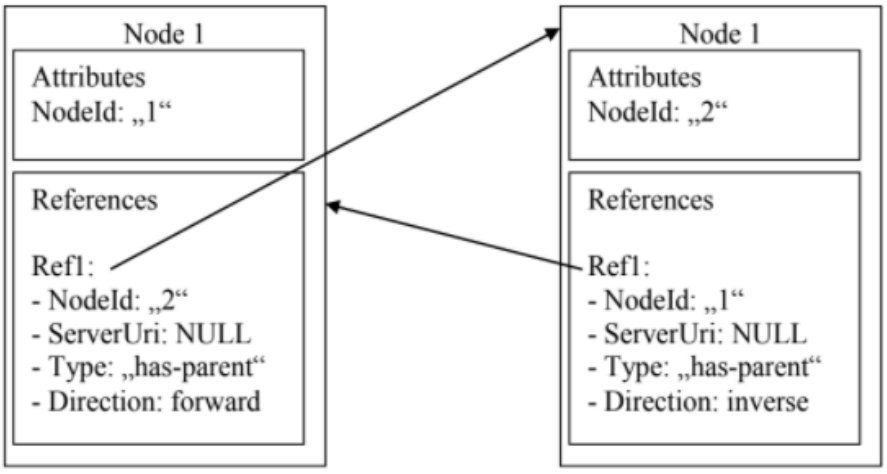

Figura 6 - Reference Description

Il nodo che contiene la Reference è detto SourceNode mentre il nodo a cui punta è detto TargetNode. Quest'ultimo può appartenere allo stesso AddressSpace o ad un altro AddressSpace di un altro OPC UA Server.

OPC UA non richiede che il TargetNode esista, quindi i References possono puntare a un nodo che non esiste.

2.3.3. Variables

La Variables vengono utilizzate per rappresentare valori. Sono definiti due tipi di Variables:

- Properties;
- DataVariables.

Si differenziano per il tipo di dati che rappresentano e se possono contenere altre Variables.

Le *Properties* si utilizzano per descrivere le caratteristiche di un nodo, per esempio l'unità di misura della temperatura misurata. In generale le Properties sono usate quando devono essere descritte caratteristiche non presenti tra gli Attributes del nodo.

Le *DataVariables* sono utilizzate per rappresentare i dati di un Object, come la temperatura misurata da un sensore; esse possono essere complesse, cioè avere sotto-variabili contenenti parte dei dati e proprietà che le descrivono.

2.3.4. TypeDefinitionNodes

I server OPC UA devono fornire definizioni di tipo per oggetti e variabili. Il riferimento HasTypeDefinition deve essere utilizzato per collegare un'istanza con la sua definizione rappresentata da un TypeDefinitionNode. Gli oggetti e le variabili ereditano gli attributi specificati dal loro TypeDefinitionNode.

L’esempio seguente descrive l’uso di HasTypeDefinition (Figura 7). Un parametro di SetPoint “SP” è rappresentato come DataVariable nell’AddressSpace. Per fornire una definizione di SetPoint comune che può essere utilizzata da altri Objects, viene utilizzato un VariableType speciale. Ciascun SetPoint DataVariable che utilizza questa definizione comune avrà un HasTypeDefinition che identificherà il SetPoint VariableType comune.

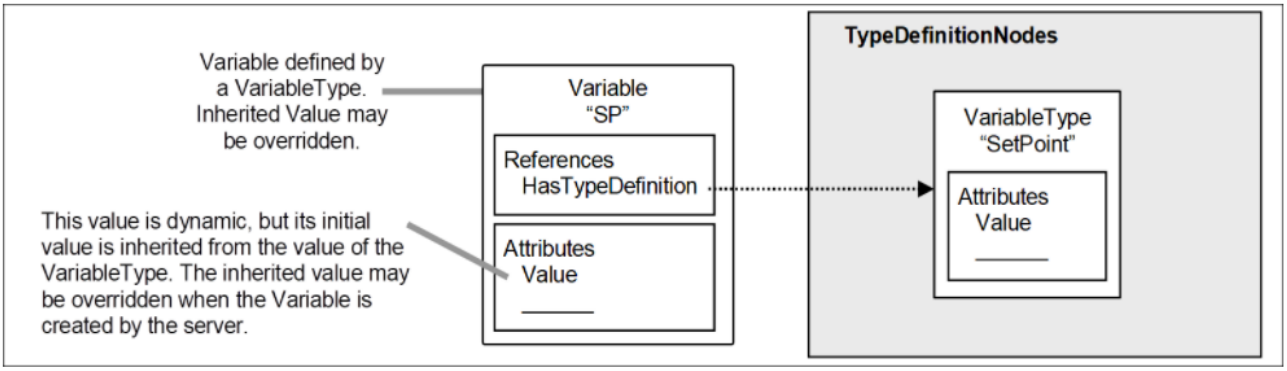

Figura 7

2.3.5. Event Model

Gli Events rappresentano specifici eventi transitori, come ad esempio modifiche alla configurazione del sistema o errori di sistema. Essi non sono direttamente visibili nell’Address Space, ma sono accessibili solo tramite Object e View.

L’attributo EventNotifier identifica se il nodo consente la sottoscrizione agli eventi e in tal caso i client si iscrivono per ricevere notifiche.

Ogni evento è di un EventType specifico che ne definisce le proprietà e permette ai client di creare filtri che indicano a quali eventi sono interessati e per ognuno di essi gli attributi desiderati. Il client utilizzerà i servizi di Subscriptions e di MonitoredItem per ricevere notifiche su eventi e per definire tali filtri.

2.3.6. Methods

I metodi sono funzioni il cui ambito è delimitato da un *Object*, simile ai metodi di una classe nella programmazione orientata agli oggetti o un *ObjectType*, simile ai metodi statici di una classe.

I metodi vengono invocati da un client, procedono al completamento sul server e restituiscono il risultato al client. La durata dell'istanza di chiamata del metodo inizia quando il client chiama il metodo e termina quando viene restituito il risultato.

I metodi possono avere un numero variabile di argomenti di input e restituire gli argomenti risultanti. Ciascun metodo è descritto da un nodo del *Method NodeClass*. Questo nodo contiene i metadati che identificano gli argomenti del metodo e ne descrivono il comportamento.

I metodi vengono richiamati utilizzando il *Call Service*. I client scoprono i metodi supportati da un server cercando i riferimenti agli oggetti proprietari che identificano i metodi supportati.

2.3.7. Roles

Un Role è una funzione assunta da un client quando accede ad un server. I ruoli vengono utilizzati per separare l'autenticazione (che determina chi è un client) dall'autorizzazione (che determina ciò che il client è autorizzato a fare).

Separando queste attività i server possono consentire ai servizi centralizzati di gestire le identità e le credenziali degli utenti mentre il server gestisce solo le autorizzazioni sui suoi nodi.

Quando viene creata una sessione, il server deve determinare quali ruoli sono concessi per quella sessione. Questa specifica definisce le regole di mappatura standard che i server supportano e consente di assegnare ruoli in base a:

- Identità dell'utente: le mappature possono essere basate su nomi utente, certificati utente o gruppi di utenti;
- Identità dell'applicazione: può essere applicata solo se il client dimostra di possedere un certificato attendibile utilizzandolo per creare un SecureChannel o fornendo una firma in ActivateSession;
- Endpoint: le mappature si basano sull'URL utilizzato per connettersi al server e può essere utilizzata per limitare l'accesso ai client in esecuzione su reti particolari.

2.4. Servizi

I servizi che offre OPC UA derivano direttamente da quelli utilizzati per i servizi Web, secondo una comunicazione di tipo richiesta (*Request*) e riposta (*Response*). Per invocare un servizio un client invia un messaggio di *Request* al server che dopo aver elaborato la richiesta invia un messaggio di *Response* al client. Questo tipo di interazione non avviene in real-time ma richiede una tempistica utile alla sua attuazione che porta alla definizione della comunicazione stessa di tipo asincrona; quindi dopo aver mandato il messaggio di *Request* l'applicazione client può continuare la propria esecuzione in attesa del messaggio di *Response* *(Figura 8)*.

Nel seguito vedremo i servizi fondamentali che OPC UA mette a disposizione.

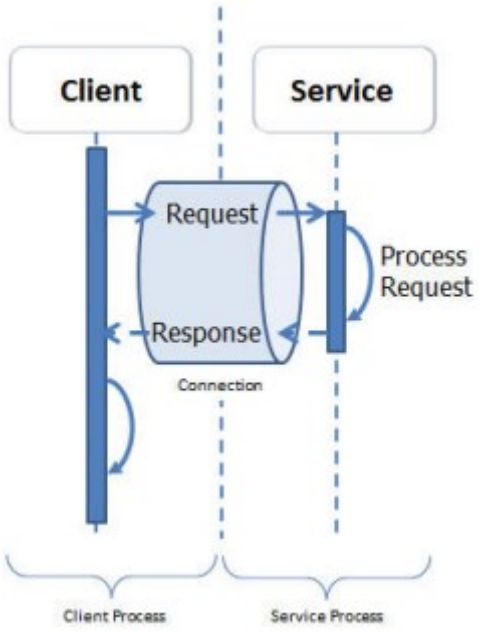

Figura 8 - OPC UA Client-Server Communication

2.4.1. Gestione dei Timeout

In OPC-UA la comunicazione è concepita per poter funzionare tra diversi sistemi, eterogenei tra loro, connessi in una rete; quindi per come è strutturata la comunicazione sono possibili interruzioni di servizio in qualsiasi momento che devono essere rilevate e gestite. Per questo motivo ogni chiamata di un servizio ha come parametri dei *Timeout* definiti dal client, che servono proprio a rilevare problemi nella comunicazione così che in seguito questa possa essere gestita correttamente.

2.4.2. Request & Response Headers

I messaggi di *Request* e *Response* relativi a qualunque servizio contengono il medesimo tipo di *Header*.

Proprietà pubbliche	Descrizione
NodeId AuthenticationToken	Identificatore utilizzato per assegnare la chiamata del servizio alla sessione precedentemente stabilita tra client e server.
DateTime Timestamp	Istante di tempo in cui il client ha inviato la richiesta.
Uint RequestHandle	Identificativo assegnato alla chiamata del servizio, definito dal client
Uint ReturnDiagnostics	Indica se il client richiede al server di fornire, in caso di errore, delle informazioni dettagliate di diagnostica anziché solo un semplice *status code.*
String AuditEntryId	Identifica il client o user che ha iniziato l'azione. Usato in caso di auditing.
Uint TimeoutHint	Timeout settato per la chiamata del servizio lato client, quando termina il server annulla la chiamata ricevuta.

Tabella 2 - Request (namespace: Opc.Ua.RequestHeader)

Proprietà pubbliche	Descrizione
DateTime Timestamp	Istante di tempo in cui il server ha inviato la risposta.
Uint RequestHandle	Identificativo assegnato alla chiamata di servizio definito dal client.
StatusCode ServiceResult	Classe definita dallo standard per descrivere il risultato della chiamata del servizio. Contiene un uint a 32 bit in cui i 16 più significativi rappresentano il valore numerico dell'errore ed i 21 rimanenti contengono informazioni aggiuntive. In particolare, i 2 bit più significativi si riferiscono all'attendibilità del risultato (00=Good, 01=Uncertain, 10=Bad, 11=not used).
DiagnosticInfo ServiceDiagnostics	Indica se il client richiede al server di fornire, in caso di errore, delle informazioni dettagliate di diagnostica anziché solo un semplice status code.

Tabella 3 - Response (namespace: Opc.Ua.ResponseHeader)

2.4.3. Server Discovery

Il Server Discovery viene utilizzato dal client per individuare i server OPC UA connessi alla rete e per leggere le specifiche di sicurezza implementati in essi. Queste informazioni saranno poi utilizzate dal client per creare un *SecureChannel* con il server. Ogni server OPC UA deve esporre almeno un Discovery Endpoint, attraverso cui il client accederà tramite il servizio *GetEndpoints*. L'accesso da parte del client agli endpoint avviene senza che sia necessario stabilire una sessione, inoltre la lettura delle relative informazioni di sicurezza contenute negli endpoint avviene in chiaro senza che ci sia un meccanismo di sicurezza.

La sequenza logica di funzionamento è la seguente:

1. Il server attivo si registra presso un Discovery Server (*RegisterServer*).

2. Il client richiede al Discovery Server la lista dei server attivi (*FindServers*), selezionandone uno.

3. Il client richiede al server l'elenco e il contenuto degli endpoint, tramite il servizio *GetEndpoints* invocato attraverso il Discovery Endpoint del Server. A tal fine, il client specificherà l'URL del Discovery Endpoint quando invocherà il servizio *GetEndpoints*.

4. Sulla base delle configurazioni di sicurezza contenute negli endpoint ricevuti, il client può creare un canale sicuro con il server (*OpenSecureChannel*). Ovviamente può succedere che le configurazioni di sicurezza sono tutte incompatibili con le caratteristiche del client, che dovrà rinunciare alla creazione di un canale sicuro.

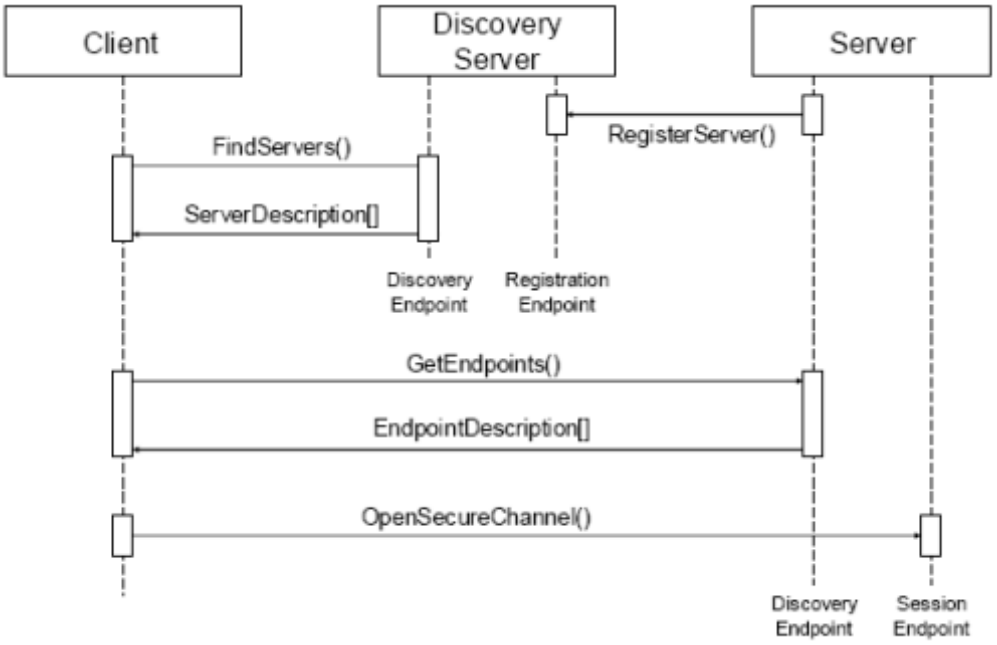

Figura 9 - OPC UA Server Discovery Service

Tutta questa procedura può essere semplificata nel caso in cui il client conosca già il server a cui connettersi (indirizzo pre-configurato, ad esempio); in tal caso le azioni che vengono intraprese sono la numero 3 e 4.

2.4.4. Read & Write

Una delle più importanti caratteristiche di OPC è la lettura e scrittura dei dati mediante meccanismi di scambio dati standard.

Read viene usato per leggere uno o più Attributes di uno o più Nodes e permette inoltre la lettura di sottoinsiemi o singoli valori di array.

Il servizio Read consente al client di definire un tempo massimo di validità dei valori da restituire; tale parametro viene detto *maxAge*. Esso obbliga il server ad accedere alla sorgente dei dati (ad esempio un sensore), se la copia mantenuta dal server è più vecchia rispetto al parametro maxAge specificato dal client.

Figura 10 - OPC UA Read Service

Write viene invece utilizzato per scrivere uno o più Attributes di uno o più Nodes e permette inoltre la scrittura di sottoinsiemi o singoli valori di array. Come la maggior parte degli altri servizi, *Write* è pensato per operazioni generiche e non relative al tipo di dato considerato.

Figura 11 - OPC UA Write Service

2.4.5. Method Call

Lo standard OPC UA permette ai server di esporre nel proprio Address Space dei Methods che possono essere richiamati dai client attraverso il servizio *Call*. Tutte le informazioni necessarie per chiamare un metodo, incluse la descrizione dettagliata dei parametri di input e di output, sono disponibili nella sua descrizione.

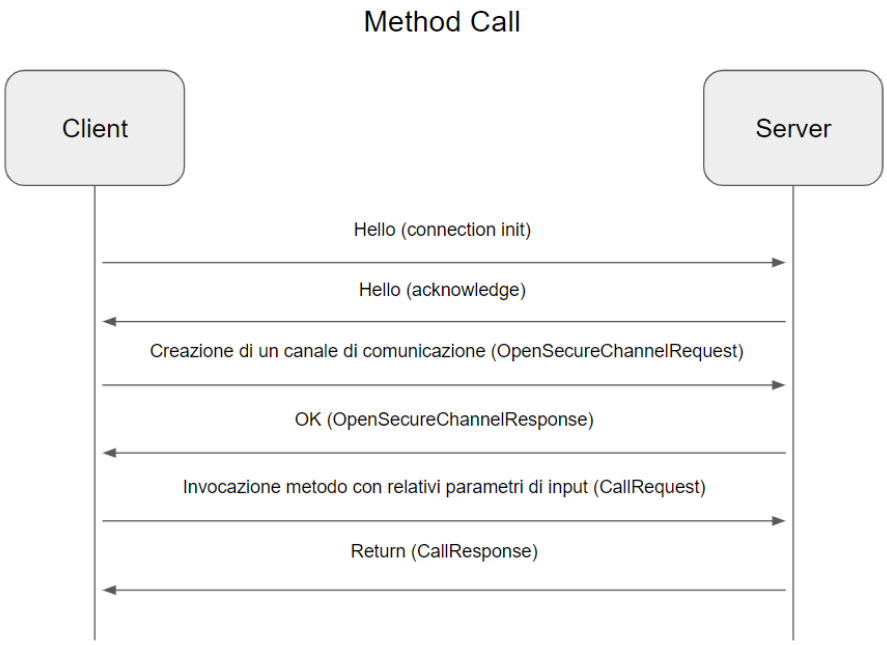

Figura 12 - OPC UA Method Call Service

2.4.6. Subscription Service

Uno degli strumenti più potenti che un client può utilizzare in OPC UA è l'attivazione di un meccanismo che richiama l'aggiornamento automatico dei dati. Un client può creare una o più Subscription e nell'ambito di ciascuna Subscription il client può creare dei Monitored Items. Ciascun Monitored Item è in grado di creare delle Notification, legate a cambiamenti di valori di attributi o di variabili scelte dal client o legate al verificarsi di particolari eventi.

Essenzialmente la Subscription ha il compito di collezionare e raggruppare tutti le Notification prodotte fino a quel momento nell'ambito della stessa Subscription, con cadenza periodica. Tutti le Notification così raggruppate vengono inserite in un NotificationMessage che viene poi inviato al client.

Una Subscription ha due parametri principali:

- il *Publishing Interval*, che definisce l'intervallo di tempo con il quale il server crea un NotificationMessage per la determinata Subscription;

- il *Publishing Enabled* (un valore booleano), che di fatto abilita o meno l'inoltro del NotificationMessage al Client.

Ogni Subscription mantiene un counter *keep-alive* che conta il numero di Publishing Interval che sono trascorsi senza che la Subscription abbia avuto Notifications da inoltrare al client. L'assenza di Notifications prodotte per quella Subscription può rappresentare un problema, perché il client non riceverà alcuna informazione per quella Subscription e dunque potrebbe pensare che la Subscription non sia più attiva. Il keep-alive counter è importantissimo perché al suo scadere, il server dovrà provvedere ad inviare al client un particolare messaggio (*keep-alive Message*) per informarlo che la Subscription è attiva.

Inoltre ogni Subscription mantiene un *lifetime* counter che conta il numero consecutivo di Publishing Interval entro i quali non vi è stata attività da parte del client. L'attività viene monitorata attraverso il controllo dell'invio da parte del client di particolari messaggi di Publish Request. Il contatore viene resettato nel momento in cui viene monitorata attività da parte del client. Quando questo contatore raggiunge il massimo per quella determinata Subscription, allora essa viene chiusa. La chiusura comporta l'eliminazione di tutti i Monitored Items in essa contenuti.

2.4.7. Monitored Item Service

La sezione precedente ha spiegato come una Subscription permetta di realizzare un meccanismo automatico di creazione di particolari messaggi che verranno inviati al client; tali messaggi sono detti *NotificationMessage*. Essi contengono i valori prodotti dai *Monitored Item* creati per quella Subscription.

Un Monitored Item viene definito nell'ambito di una Subscription per monitorare un particolare item del server che può essere:

- Attribute: un qualunque attributo di un qualunque Nodo può essere associato ad un Monitored Item. Gli attributi vengono monitorati unicamente per indicare quando il loro valore cambia. Il cambiamento del valore produce una Notification.

- Variable: l'attributo Value dei nodi appartenenti al Variable NodeClass può essere associato ad un MonitoredItem. L'attributo Value di una Variable viene monitorato per indicare un cambiamento nel valore o un cambiamento dello Status Code associato al Value. Vengono specificate delle condizioni su tali cambiamenti: se tali condizioni sono verificate, allora il cambiamento del valore o dello stato associato al Value di una Variabile causa la produzione di un Notification da parte del Monitored Item.
- Object e View: i nodi appartenenti al NodeClass Object e View possono essere associati ad un Monitored Item, al fine di essere monitorati circa il verificarsi di un particolare evento. A tal fine, però, è necessario che il relativo nodo abbia settato un particolare bit denominato SubscribeToEvents dell'attributo EventNotifier. Il verificarsi dell'evento produce la generazione di una Notification da parte del Monitored Item.

Come detto dunque, per ogni cambiamento di attributo, valore, stato o per ogni evento monitorato dal Monitored Item viene prodotta una Notification. Diverse Notifications vengono impacchettate dalla Subscription dentro un NotificationMessage ad intervalli regolari scanditi dal Publishing Interval.

Tutti i Monitored Items hanno alcuni settaggi in comune:

- *Sampling Interval*: definisce la frequenza con cui il server campiona l'item reale al quale si riferisce il Monitored Item (attributo, valore di un nodo oppure evento).
- *Monitoring Mode:* è usato per abilitare e disabilitare sia il campionamento che la produzione delle Notifications.
- *Filter Settings*: definisce i criteri che il server utilizza per determinare se una Notification deve essere generata per un determinato Monitored Item.
- *Queue Parameters*: definiscono le caratteristiche della coda utilizzata per mantenere le Notifications prodotte dai Monitored Items. Tra gli attributi vi è la lunghezza della coda e la politica di gestione della coda.

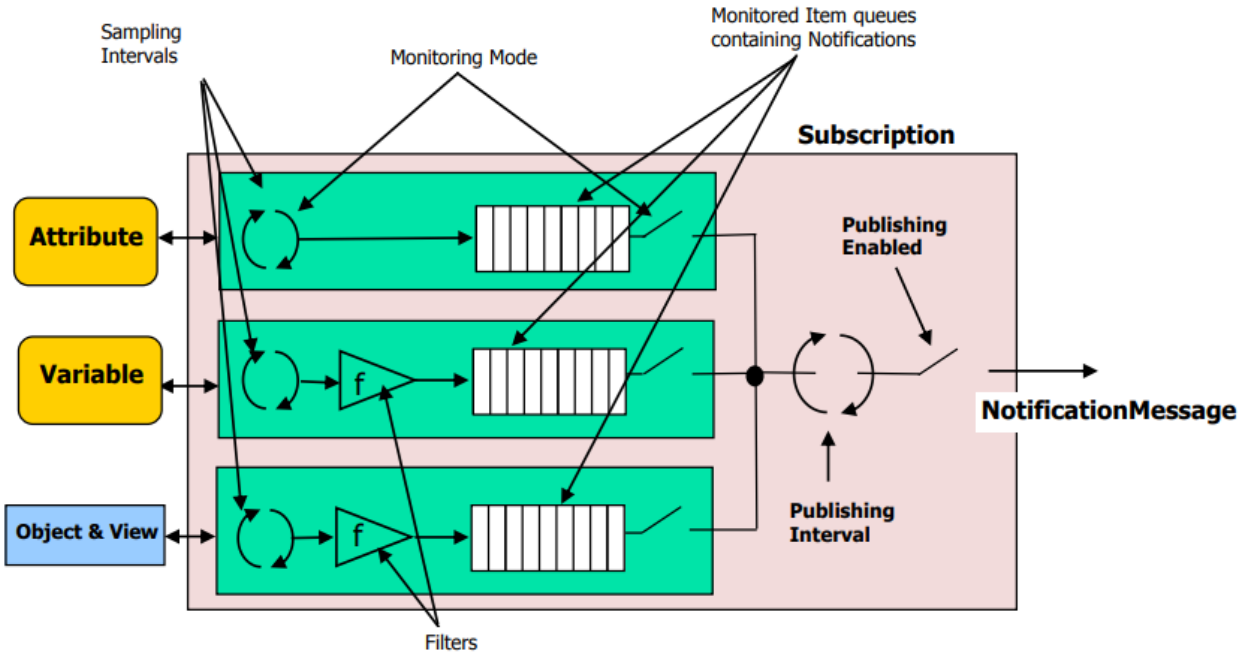

Figura 13 - OPC UA Monitored Item Service

2.4.8. Invio delle Notifications al Client

Il server deve essere in grado di inviare informazioni al client. Come è stato visto nella sezione precedente il client può definire una o più Subscription nell'ambito di una Sessione attiva; per ciascuna Subscription è stato anche spiegato che il Client può creare dei Monitored Item che sono di fatto dei produttori di informazioni (Notifications) accumulate in specifiche code. Ad una frequenza costante (Publish Interval), per ogni Subscription, i contenuti attuali di tutte le code dei Monitored Items vengono inglobati in un NotificationMessage, per essere consegnati al client.

In questa sezione viene illustrato il meccanismo di invio dei NotificationMessage prodotti per ogni Subscription di una Sessione attiva.

In OPC UA, il meccanismo di comunicazione si basa su connessioni unidirezionali, rispettando i requisiti Firewall-Friendly ed immutabilità. Gli elementi principali che caratterizzano l'invio di informazioni da parte del Server sono:

- Invio di NotificationMessage da parte del server regolato da flussi di richieste inviate dal client; un NotificationMessage viene inviato al client in risposta a una Publish Request. Il server mantiene per ogni Sessione attiva una coda di Publish Request ricevute dal client per quella sessione. Per ogni NotificationMessage prodotto da una qualunque delle Subscriptions, una Request viene estratta dalla coda delle Request e il NotificationMessage viene inviato al client tramite una Publish Response.
- Ogni NotificationMessage è identificato in modo univoco da un numero di sequenza specificato dal server nella Publish Response. L'utilizzo di un numero di sequenza permette al client di scoprire l'eventuale perdita del NotificationMessage inviato dal Server.
- Gestione di Ack inviati dal client al server per informare il server circa la corretta ricezione dei NotificationMessage ricevuti.
- Re-invio dei NotificationMessage andati perduti, su esplicita richiesta del client.
- Invio periodico di un "keep-alive" Message, dal server al client, quando non sono disponibili NotificationMessage da inviare; in questo modo il server notifica al client che la sessione è ancora attiva.

Dunque per ogni Publish Request, il server sarà tenuto ad inoltrare al client un NotificationMessage tra quelli prodotti dalle Subscriptions presenti nella Sessione attiva. Ovviamente se non vi sono NotificationMessage, il server attenderà che ne verrà prodotto uno, il quale verrà poi inoltrato al client tramite una Publish Response. Nel caso in cui più NotificationMessage siano stati prodotti e siano pronti per essere trasmessi, il Server dovrà decidere quale NotificationMessage prelevare ed inviare al Client. La politica che viene definita nello standard tiene conto della priorità della Subscription, definita all'atto della creazione della Subscription. Nel caso in cui più NotificationMessage siano pronti per diverse Subscription, il server estrae il NotificationMessage relativo alla Subscription con priorità più alta. Nel caso le Subscriptions abbiano la stessa priorità, il server sceglie secondo una politica round-robin.

La *Figura 14* aiuta a comprendere quanto detto. Come si vede per ciascuna Publish Request pervenuta, il server sceglie una Subscription (in base alla priorità o scelta in base ad una politica round-robin) dalla quale viene prelevato un

NotificationMessage che viene poi spedito dentro una Publish Response. La stessa figura mostra, inoltre, una coda di request in cui vengono accodate tutte le richieste avanzate dal client.

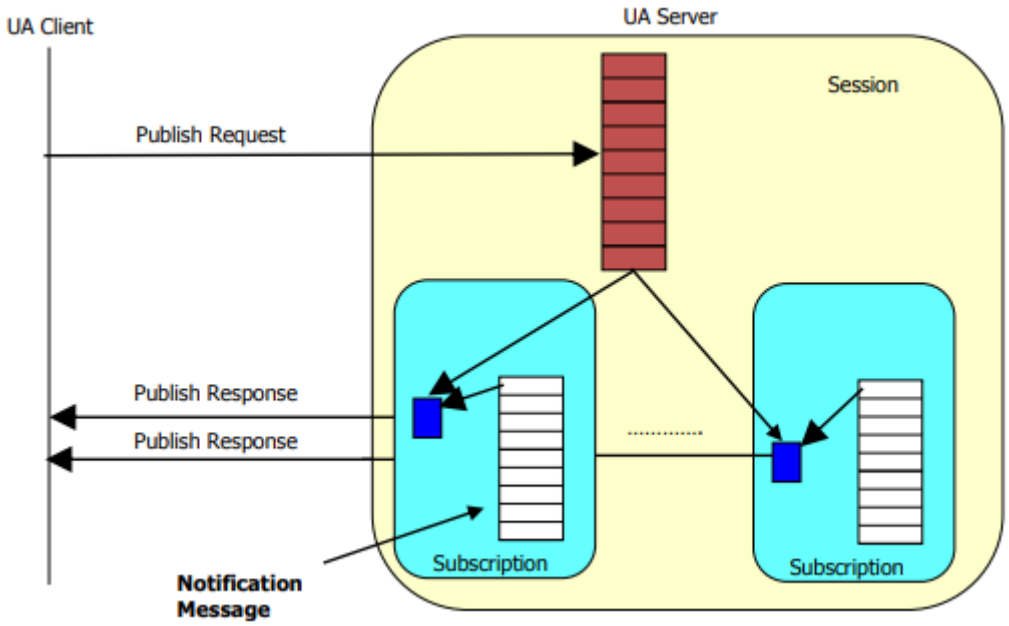

Figura 14 - OPC UA Notification Service

Da notare che l'invio di *NotificationMessage* da parte del server non è arbitrario. La *Publish Response* è subordinata alla precedente ricezione di una *Request*; in altre parole, quando è prodotto un *NotificationMessage* (allo scadere di ogni *Publish Interval*), esso può essere inoltrato ad un client solo se la coda delle *Publish Request* non è vuota.

Si comprende come la frequenza di invio di *Publish Request* da parte del client è un parametro essenziale per le prestazioni dell'intero sistema di scambio dati client/server. Se il client invia request con bassa frequenza, ciò comporterà un accumulo di *NotificationMessage* lato Server con conseguenti ritardi di trasmissione. Se la frequenza di invio di request è, viceversa, molto alta allora c'è il rischio di saturare la larghezza di banda del canale di comunicazione, introducendo allo stesso modo ritardi di comunicazione. In generale la frequenza

di invio di request da parte del client dovrebbe essere legata ed influenzata da questi parametri:

- Numero di sottoscrizioni create dal client e dai valori dei relativi *Publish Interval*;
- Latenza della rete.

2.5. Technology Mapping

Tutte le applicazioni OPC UA devono implementare almeno uno *StackProfile* e possono comunicare solo con altre applicazioni OPC UA che implementano lo stesso *StackProfile*. Questi *StackProfiles* sono una combinazione di diverse mappature (vedi Figura 15) ognuna organizzata in tre gruppi:

1. Data Encodings;
2. Security Protocols;
3. Transport Protocols.

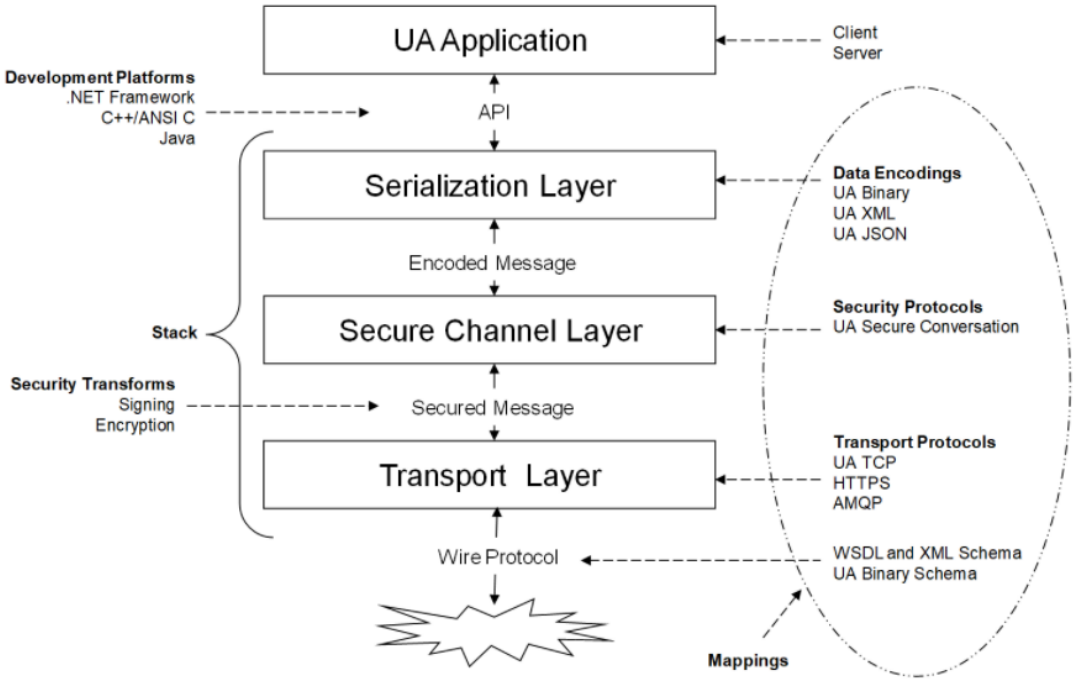

Figura 15 - OPC UA Stack Profile

Ciascun *OPC UA StackProfile* deve essere trattato come un singolo protocollo di livello 7 (*Application*) basato su un protocollo di livello 5, 6 o 7 esistente come TCP/IP, TLS o HTTP. Il livello *SecureChannel* è sempre presente anche se *SecurityMode* è impostato su *None*. In questo caso non viene applicata alcuna sicurezza, ma l'implementazione *SecurityProtocol* deve mantenere un canale logico con un identificatore univoco.

2.5.1. Data Encodings

La codifica dei dati consiste nella serializzazione del messaggio in un formato ottimizzato per la trasmissione in rete. OPC UA specifica tre codifiche: OPC UA Binary, XML e JSON.

Il concetto di base di *OPC UA Binary Encoding* è che un insieme specifico di tipi di dati primitivi (Built-in Data Types) viene tradotto in una rappresentazione binaria utilizzando regole ben definite. Questa codifica viene utilizzata quando è importante mantenere performance elevate e introdurre un overhead minimo nella comunicazione.

5				O	P	C	U	A
05	00	00	00	4F	50	43	55	41

Figura 16 - String example of OPC UA Binary Encoding

I documenti XML, invece, svolgono un ruolo importante poiché la struttura è standardizzata e ciò consente a tutte le applicazioni o piattaforme dotate di un

parser XML di interpretare tali documenti. In questo modo OPC UA permette ad applicazioni di livello operativo di scambiare dati con sistemi a livello aziendale.

Figura 17 - Structure example of XML Encoding

In OPC UA è possibile scambiare dati anche in formato JSON, in particolare per servizi Web o gestionali che importano ed esportano dati JSON. Il funzionamento di tale codifica è molto simile a XML.

2.5.2. Security Protocols

Esistono due protocolli di sicurezza definiti per OPC UA:

1. WS-SecureConversation;
2. UA-SecureConversation.

Entrambi si basano su una connessione certificate-based.

WS-SecureConversation è un'estensione di WS-Security, che definisce concetti e tecnologie per lo scambio dati sicuro attraverso Web Service. Questo protocollo è ottimizzato per lo scambio di dati in formato XML, il che lo rende ideale per le applicazioni OPC UA che interagiscono con sistemi MES e ERP.

Purtroppo non è stato ampiamente adottato dal settore e quindi ad oggi è diventato obsoleto.

OPC UA-SecureConversation (UASC) consente la comunicazione sicura utilizzando messaggi codificati binari ed è progettato per funzionare con diversi TransportProtocol. La struttura del messaggio è mostrata di seguito.

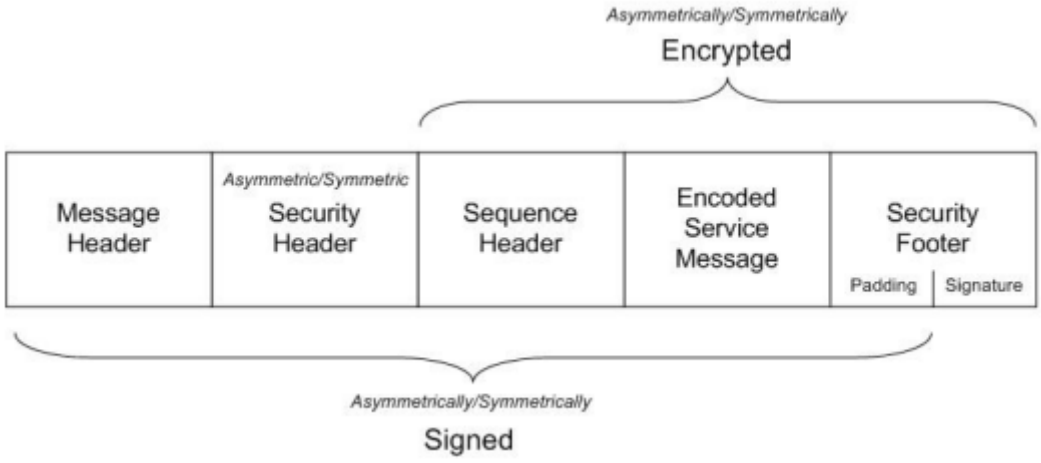

Figura 18 - Tipical structure of OPC UA messages with UA-SecureConversation

Il Message Header contiene informazioni per identificare il tipo di messaggio (ad esempio apertura di un canale o creazione di una sessione).

Questo header è seguito dal Security Header, che può essere di due tipi:

- Asymmetric Security Header: usato per una crittografia asimmetrica basata su chiave pubblica e privata;
- Symmetric Security Header: che contiene un TokenId che identifica il set di chiavi simmetriche usato per la codifica.

Il Sequence Header contiene un numero di sequenza per permettere di ricomporre un messaggio dopo che è stato suddiviso in chunks.

Infine il Security Footer che contiene la Signature del messaggio, permettendo di verificare integrità e autenticità del messaggio.

2.5.3. Transport Protocols

I protocolli per il trasporto hanno lo scopo di stabilire una connessione di rete tra client e server, attraverso l'apertura di Secure Channels e sessioni.

OPC UA definisce due protocolli per il trasferimento dati: UA TCP e HTTPS.

UA TCP definisce un semplice protocollo che opera in congiunzione con TCP aggiungendo alcune funzioni necessarie per OPC UA:

- Stabilimento a livello applicativo delle dimensioni del buffer di comunicazione;
- Condivisione indirizzo IP e numero di porta tra più endpoint OPC che risiedono sullo stesso device;
- Meccanismi di Error Recovery.

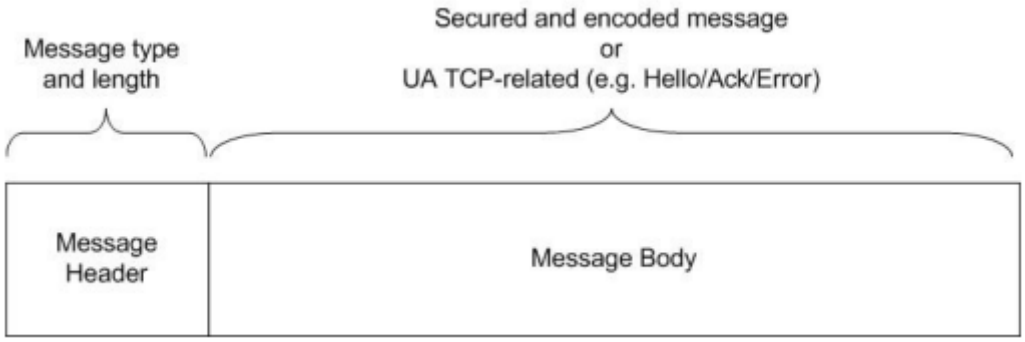

Figura 19 - Structure of OPC UA messages with TCP protocol

Il Message Header contiene informazioni sul tipo e lunghezza del messaggio. Se in combinazione con UA-ServiceConversation questo campo viene condiviso in modo da ridurre l'overhead sulla linea.

Il Message Body può contenere un messaggio relativo ad una service call oppure un messaggio necessario all'uso del protocollo.

UA TCP definisce tre tipologie di messaggi di gestione:

- ***Hello***: inviato dal client per creare un socket verso uno specifico endpoint del server;
- ***Acknowledge***: conferma la creazione del socket e l'accettazione dei parametri relativi alle dimensioni del buffer, messaggi e chunk;
- ***Error***: viene usato per l'invio di informazioni di diagnostica nel caso di rilevamento di errori.

HTTPS fa riferimento ai messaggi HTTP scambiati su una connessione SSL/TLS. La sintassi non cambia, l'unica differenza è che viene creata una connessione TLS invece di una connessione TCP/IP.

Capitolo 3

3. Security

3.1. Perché la Sicurezza Informatica è importante in ambito Industriale?

Negli ultimi anni c'è stata una crescita esponenziale dell'integrazione tra il mondo OT (Operational Technology) e il mondo IT (Information Technology); ma se da un lato l'Industria 4.0 e la digitalizzazione hanno contribuito a migliorare e snellire la gestione dei processi industriali, dall'altro sono aumentati i rischi e le possibilità di subire attacchi informatici.

In ambito industriale, particolarmente a rischio sono i sistemi ICS (Industrial Control Systems) che si occupano di gestire l'interazione e l'interconnessione tra i macchinari industriali di un'azienda.

Ad oggi i sistemi di controllo possono interfacciarsi con dispositivi esterni tramite la rete web, di conseguenza l'intera infrastruttura viene esposta ad attacchi informatici che possono provocare gravi danni alla safety fino a comportare l'interruzione dei processi industriali, con la conseguente perdita economica.

Secondo l'*IBM X-Force's ranking,* l'industria manifatturiera è uno dei settori più colpiti da attacchi di tipo ransomware nel 2021 (vedi Figura 20).

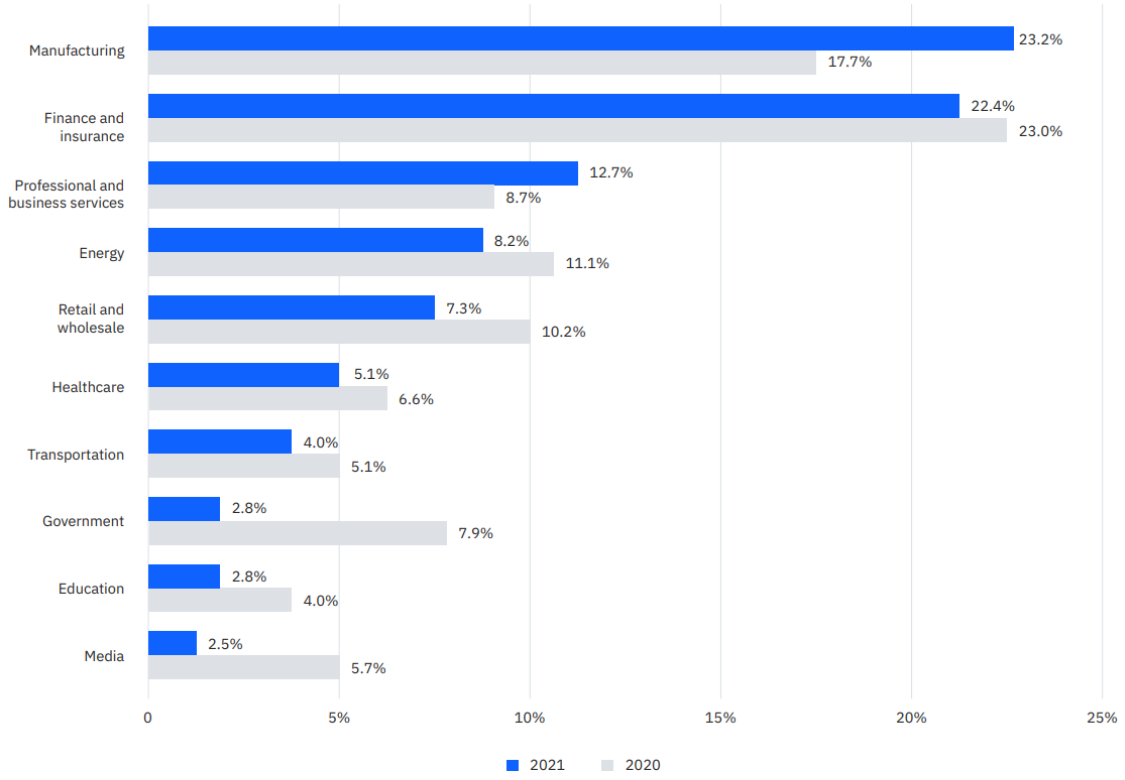

Figura 20 - Breakdown of attacks on the top 10 industries, 2021 vs. 2020

Quindi la Sicurezza Informatica, o Cyber Security, non deve essere sottovalutata ma, al contrario, deve essere parte integrante delle strategie aziendali; perché i danni che un attacco provoca all'azienda non si limita soltanto all'ambito operativo.

Incidenti relativi alla Cyber Security all'interno di un ICS possono:

- diffondere informazioni riguardo processi e progetti mettendo a rischio il segreto aziendale;
- diffondere informazioni sensibili e dati riservati appartenenti a dipendenti o persone esterne all'azienda;
- comportare l'incorrere in sanzioni per la trapelazione delle informazioni in base a quanto stabilito dalla GDPR;
- provocare danni con impatto sull'ambiente a causa del malfunzionamento dei processi;
- compromettere o arrestare la produzione per periodi medio-lunghi;

- danneggiare le strutture a causa di una cattiva gestione dell'attacco.

3.1.1. Principali tipi di Attacco

I principali attacchi informatici ai sistemi di controllo industriale possono avvenire con diverse modalità:

- attraverso un malware, che riesce a penetrare all'interno della rete aziendale grazie ad una disattenzione di un utente che naviga in siti non protetti oppure apre un'email di phishing con allegati malevoli (ad esempio gli attacchi ransomware quali Cryptolocker e simili);
- a causa di attacchi specifici rivolti agli ICS, come Chavez, Stuxnet, Wannacry o Industroyer;
- tramite configurazione errata di alcuni componenti hardware o software dell'ICS;
- a causa di attacchi interni, per mano di dipendenti (o ex-dipendenti) con intenzioni fraudolente allo scopo di danneggiare di proposito il sistema di controllo. Essi non si servono di tecniche hacker invasive, ma utilizzano semplicemente le proprie conoscenze dell'infrastruttura per rubare informazioni o danneggiare la produzione.

3.1.2. Come difendersi

Le contromisure fondamentali per mitigare e ridurre i danni di questo tipo di minacce sono:

- sensibilizzare e formare il personale dipendente sulle tematiche della Cyber Security e sui rischi che l'azienda corre (la maggior parte degli attacchi avviene colpendo le debolezze del personale che non conosce a fondo il problema);
- migliorare la protezione dell'ICS integrando le soluzioni di sicurezza IT con misure di difesa hardware, quali la firma digitale e le tecniche di crittografia (la firma digitale consiste nell'autenticare i messaggi o le parti software che circolano nelle diverse parti del processo, in modo da garantirne

l’origine sicura e bloccando azioni non consentite, invece, le tecniche di crittografia servono a proteggere i dati e le informazioni evitandone la divulgazione);

- assumere uno specialista di Cyber Security che permetta di migliorare la sicurezza complessiva attraverso attività di prevenzione, rilevazione e infine gestione degli attacchi informatici.

3.2. OPC UA Security Model

OPC UA è un protocollo utilizzato tra i componenti nel funzionamento di un impianto industriale a più livelli: dalla gestione aziendale di alto livello al controllo diretto del processo a basso livello di un dispositivo. L’utilizzo di OPC UA per la gestione aziendale comporta rapporti con clienti e fornitori. Può essere un bersaglio interessante per lo spionaggio industriale o il sabotaggio e può anche essere esposto a minacce attraverso malware, come i worm, che circolano sulle reti pubbliche.

L’interruzione delle comunicazioni al controllo di processo potrebbe comportare perdite finanziarie, incidere sulla safety dei dipendenti (o anche pubblica) o causare danni all’ambiente.

OPC UA, essendo distribuito in una vasta gamma di ambienti operativi con ipotesi diverse su minacce e accessibilità e con una varietà di politiche di sicurezza differenti, fornisce un insieme flessibile di meccanismi di sicurezza. La figura seguente mostra una combinazione di tali ambienti.

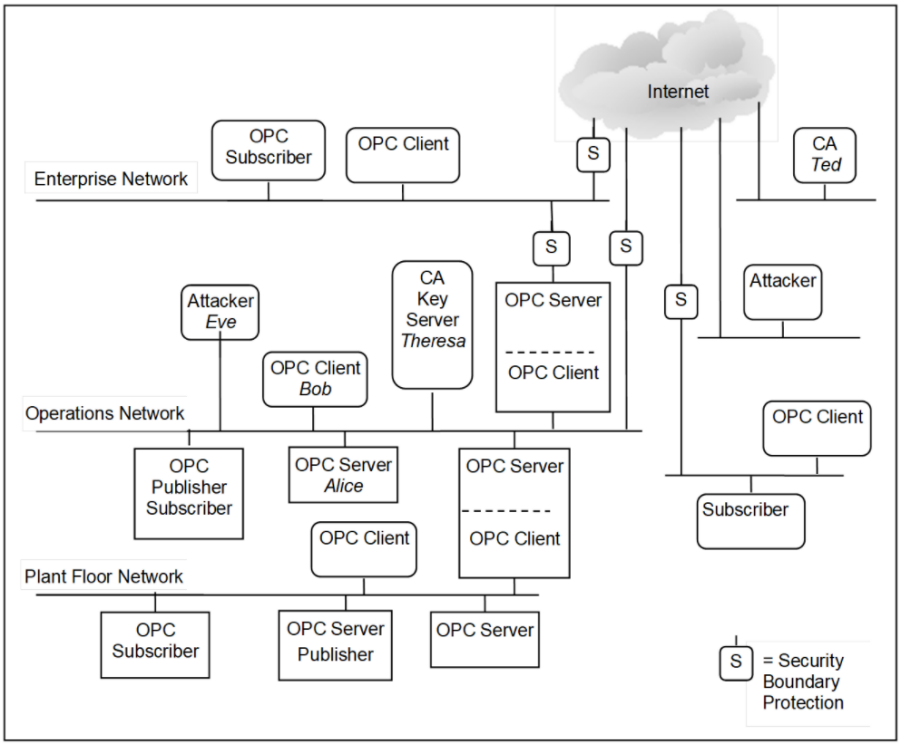

Figura 21 - OPC UA Network example.

Quindi è possibile trovare applicazioni OPC UA sullo stesso host che possono essere facilmente protette da attacchi esterni, oppure applicazioni su host differenti nella stessa rete operativa che possono essere protette installando delle soluzioni di sicurezza perimetrale (Firewall), separando così la rete interna dalla rete esterna; oppure, ancora, applicazioni OPC UA che sono integrate in sistemi di controllo che non hanno una connessione diretta con la rete esterna.

3.2.1. OPC UA Security Objectives

OPC UA soddisfa i requisiti di sicurezza garantendo autenticazione e autorizzazione dei client, nonché protezione dell'integrità e riservatezza dei dati.

Per l'autenticazione del client, OPC UA consente l'accesso anonimo (Anonymous), una combinazione di nome utente e password (Username) o un certificato (Certificate). Inoltre, i server OPC UA possono imporre il controllo dell'accesso per ciascun nodo. È indispensabile una corretta configurazione

dell'autenticazione, ad esempio limitando l'accesso anonimo ai server non critici e modificando le credenziali di default dei produttori.

Per proteggere la comunicazione, OPC UA fornisce diverse modalità di sicurezza dei messaggi (*None, Sign* o *SignAndEncrypt*) nonché politiche di sicurezza che predefiniscono algoritmi crittografici per la crittografia e le firme (*Basic128Rsa15, Basic256* o *Basic256Sha256*). Il client seleziona una politica di sicurezza durante l'handshake per proteggere i messaggi scambiati.

Nel complesso, OPC UA fornisce potenti funzionalità di sicurezza ma la corretta configurazione di questi meccanismi è essenziale. In caso contrario, un attaccante può essere in grado di accedere a dati riservati o compromettere il server OPC UA.

3.2.2. OPC UA Security Architecture

OPC UA supporta più protocolli e tecnologie di comunicazione, che potrebbero richiedere livelli di sicurezza diversi, come le comunicazioni *Client-Server* e *Publisher-Subscriber*.

L'architettura di sicurezza OPC UA, per la comunicazione *Client-Server* è strutturata in diversi layers.

1. **Application Layer**

La comunicazione tra un'applicazione client e un'applicazione server, per trasmettere informazioni, impostazioni e comandi, avviene tramite la creazione di una sessione (*Session*). L'application layer gestisce anche la parte di autenticazione e autorizzazione (*Authentication & Authorization*) di un utente. Una sessione comunica attraverso un canale sicuro (*Secure Channel*) creato a livello comunicazione (*Communication Layer*) e si basa su di esso per una trasmissione sicura.

In caso di interruzioni del canale, la sessione rimarrà valida per un periodo di tempo consentendo al client di ristabilire la connessione con il server tramite un

nuovo canale. In caso contrario la sessione si chiude allo scadere della sua durata.

2. Communication Layer

Il livello comunicazione fornisce meccanismi di sicurezza per soddisfare i requisiti di *Riservatezza, Integrità* e *Autenticazione* dell'applicazione. È quindi essenziale stabilire un canale sicuro (*Secure Channel*) che viene utilizzato per proteggere la comunicazione tra un client e un server. Il canale fornisce la crittografia per mantenere la riservatezza, le firme dei messaggi per garantire l'integrità e i certificati per fornire l'autenticazione dell'applicazione.

I meccanismi di sicurezza forniti dai servizi *Secure Channel* sono implementati da uno stack di protocollo come spiegato in sezione *2.5 Technology Mapping*.

Se viene utilizzato *l'OPC UA Connection Protocol* (*UACP*), le funzionalità per *Riservatezza, Integrità, Autenticazione* dell'applicazione e *Secure Channel* sono simili alle specifiche SSL/TLS.

Per sopravvivere alla perdita delle connessioni del livello di trasporto (*Transport Layer*), ad esempio connessioni TCP, e riprendere una nuova connessione, il livello di comunicazione è responsabile di ristabilire la connessione del livello di trasporto senza interrompere il canale logico di sicurezza.

3. Transport Layer

Il livello di trasporto gestisce la trasmissione, la ricezione e il trasporto dei dati forniti dal livello di comunicazione (*Communication Layer*).

Il livello di trasporto può essere utilizzato anche per implementare riservatezza e integrità utilizzando HTTPS. È importante notare che i certificati HTTPS possono essere condivisi da più applicazioni su una piattaforma e che possono essere compromessi al di fuori dell'utilizzo di OPC UA.

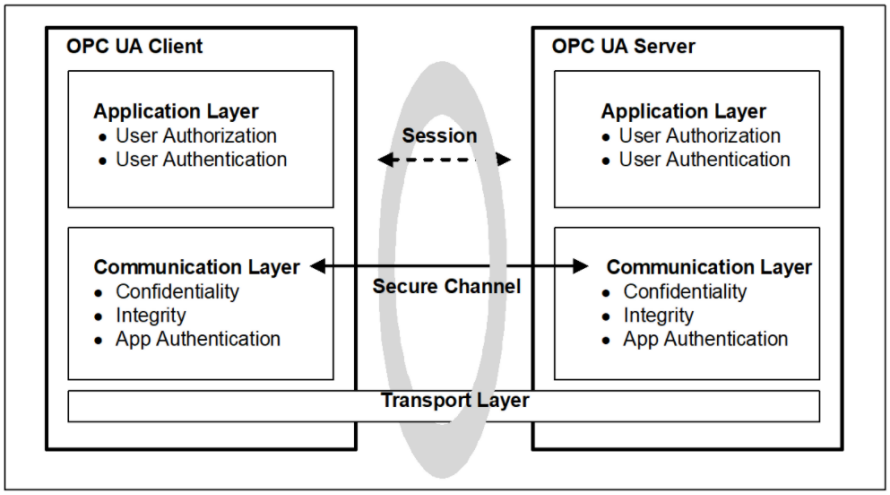

Figura 22 - OPC UA Security Architecture, Client-Server.

OPC UA supporta anche un'architettura di comunicazione *Publisher-Subscriber* (*PubSub*) che fornisce riservatezza e integrità. Ciò viene ottenuto utilizzando la *crittografia simmetrica* e gli algoritmi di firma.

Le chiavi simmetriche sono distribuite da un *Security Key Server* (SKS) che utilizza la sicurezza *Client-Server* standard, descritta nella sezione precedente, per stabilire l'autenticazione dell'applicazione e dell'utente. Questo approccio consente a tutte le applicazioni (Publisher e/o Subscriber) in un *SecurityGroup* di condividere informazioni.

Un vantaggio dell'utilizzo di chiavi simmetriche condivise sono le prestazioni elevate che offrono, ma uno svantaggio è che tutte le applicazioni del gruppo hanno gli stessi diritti. Quindi qualsiasi applicazione (Publisher o Subscriber) nel gruppo può pubblicare un messaggio e qualsiasi applicazione nel gruppo può decodificarlo.

Ad esempio supponiamo un sistema composto da un controller (Publisher) e da tre Human Machine Interface, detti anche HMI (Subscribers). Il controller sta pubblicando i messaggi e gli HMI li stanno ricevendo. Se uno degli HMI fosse compromesso, potrebbe iniziare a pubblicare messaggi e gli altri due HMI non sarebbero in grado di capire se il messaggio sia stato inviato dal controller o meno. Una possibile soluzione a questo problema potrebbe essere che la chiave

fosse condivisa da un gruppo composto solo dal controller e da un HMI. Quindi nessun HMI potrebbe influenzare gli altri HMI.

La configurazione di *SecurityGroups* richiede un'attenta considerazione per garantire i requisiti di sicurezza.

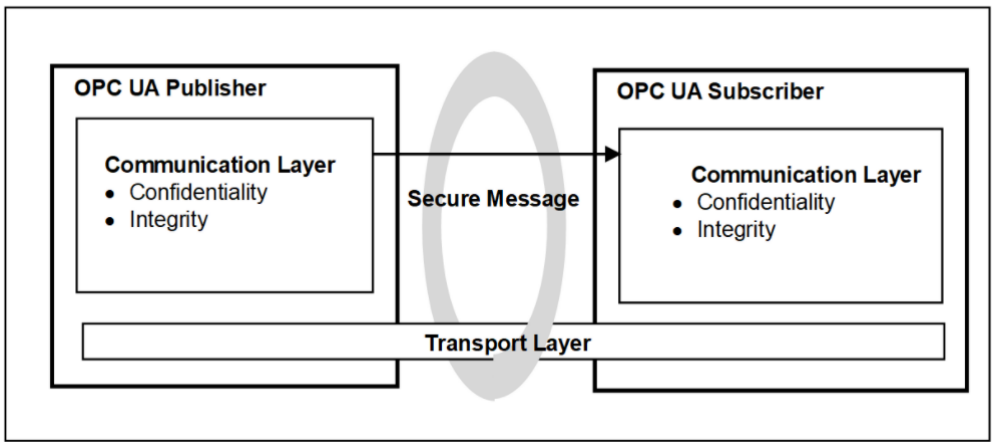

Figura 23 - OPC UA Security Architecture, Publisher-Subscriber.

3.2.3. SecureChannel Service Set

Questo set definisce i servizi utilizzati per creare un canale di comunicazione che garantisce Riservatezza e Integrità dei dati scambiati tra client e server.

OpenSecureChannel	
Parametri	**Descrizione**
Request	
ClientCertificate: ApplicationInstanceCertificate	Certificato che identifica il client. La richiesta di OpenSecureChannel dovrebbe essere firmata con la chiave privata di questo certificato. Se la SecurityPolicyUri è impostata su 'None', il server dovrebbe ignorare l'ApplicationInstanceCertificate.
SecurityToken Request Type	ISSUE: crea un nuovo SecurityToken per un nuovo SecureChannel RENEW: crea un nuovo SecurityToken per un SecureChannel già esistente.
SecureChannelId	L'identificatore per il SecureChannel.
MessageSecurityMode	Il tipo di sicurezza da applicare al messaggio.
SecurityPolicyUri	Identifica la SecurityPolicy da utilizzare per proteggere i messaggi inviati tramite SecureChannel.
ClientNonce	Un numero casuale generato ogni volta che viene creato o rinnovato un SecureChannel.
Response	
ChannelSecurityToken	Descrive il nuovo SecurityToken emesso dal server.
ChannelId	Identificatore univoco per il SecureChannel
TokenId	Identificatore univoco per un singolo SecurityToken
ServerNonce	Un numero casuale generato ogni volta che viene creato o rinnovato un SecureChannel.

Tabella 4 - OpenSecureChannel parameters, this service is used to open or renew a SecureChannel.

3.2.4. Session Service Set

Questo set definisce i servizi per una connessione a livello Application nel contesto di una sessione.

CreateSession	
Parametri	Descrizione
Request	
Application Description	Informazione che descrive l'applicazione client.
SessionName	Stringa che identifica il nome della sessione.
ClientNonce	Un numero random di lunghezza minima 32 byte. Il Server utilizzerà questo valore per dimostrare il possesso del proprio ApplicationInstanceCertificate nella response.
ClientCertificate: ApplicationInstanceCertificate	Certificato rilasciato al Client. Se la SecurityPolicyUri è impostata su 'None', il server dovrebbe ignorare l'ApplicationInstanceCertificate Il server dovrebbe verificare che questo certificato sia lo stesso utilizzato per creare il SecureChannel.
Response	
SessionId: NodeId	Identificatore univoco assegnato dal Server alla sessione.
AuthenticationToken: SessionAuthenticationToken	Identificatore univoco da utilizzare per ogni richiesta e determina se un Client ha accesso ad una sessione o meno.
ServerNonce	Un numero random di lunghezza minima 32 byte. Il Client utilizzerà questo valore per dimostrare il possesso del proprio ApplicationInstanceCertificate nella richiesta ActivateSession.
ServerCertificate: ApplicationInstanceCertificate	Certificato rilasciato al Server. Il Server ne proverà il possesso utilizzando la chiave privata per firmare il 'Nonce' fornito dal Client nella richiesta. Per i SecureChannels che utilizzano l'ApplicationInstanceCertificate, il Client dovrà verificare che questo certificato sia lo stesso utilizzato per creare il SecureChannel.
ServerEndpoint: Endpoint Description	Elenco di Endpoint supportati dal Server.
ServerSignature	Firma generata con la chiave privata associata al ServerCertificate. Viene calcolata aggiungendo il ClientNonce al ClientCertificate e firmando la sequenza di byte risultante.

Tabella 5 – CreateSession parameters, this service is used by a Client to create a Session.

ActivateSession	
Parametri	**Descrizione**
Request	
ClientSignature	Firma generata con la chiave privata associata al ClientCertificate. Viene calcolata aggiungendo il ServerNonce al ServerCertificate e firmando la sequenza di byte risultante.
UserIdentityToken	Le credenziali dell'utente associato all'applicazione client. Il Server utilizza queste credenziali per determinare se al Client deve essere consentito attivare una sessione e a quali risorse ha accesso.
UserTokenSignature	Se il Client ha specificato un token di identità utente che supporta le firme digitali, deve crearne una firma e passarla come parametro.
Response	
ServerNonce	Un numero random di lunghezza minima 32 byte. Il Client utilizzerà questo valore per dimostrare il possesso del proprio ApplicationInstanceCertificate nella chiamata successiva alla richiesta di ActivateSession.

Tabella 6 - ActivateSession parameters, this service is used by a Client to specify the User's identy for the Session.

3.2.5. Secure Communication Channel

OPC UA può adottare certificati di tipo *X.509v3* per creare delle connessioni garantendo autenticità e autorizzazione. OPC UA ne distingue tre tipi:

- **Application Instance Certificate**: identifica l'istanza di un'applicazione OPC UA in esecuzione su un determinato host ed è ottenuto dalla Certification Authority responsabile;
- **Software Certificate**: attesta che un prodotto ha superato determinati test e supporta quindi un elenco di profili e servizi ed è rilasciato da un laboratorio di certificazione autorizzato OPC UA;
- **User Certificate**: verifica che un utente sia veramente chi afferma di essere e che abbia i permessi necessari per le operazioni che intende eseguire. L'adozione di questi certificati non è obbligatoria in quanto OPC UA supporta altri metodi di autenticazione, come la coppia username e password.

La connessione tra un client e un server OPC UA consiste nella creazione di un *Secure Channel* e di una sessione al suo interno. Questo avviene in quattro step:

1. Il client recupera, se necessario, le informazioni relative agli endpoint del server al quale vuole collegarsi, incluse le configurazioni di sicurezza. Sceglie poi un endpoint con una Security Policy adeguata al contesto e ne verifica la validità tramite una *Validation Authority* (due applicazioni OPC UA possono comunicare tra di loro soltanto se hanno almeno una Security Policy in comune);

2. Se il certificato dell'endpoint è considerato affidabile viene creato un Secure Channel, si definisce quindi una Security Mode (None, Sign o SignAndEncrypt). La creazione di un Secure Channel viene usata principalmente per scambiare informazioni segrete che vengono poi utilizzate per derivare chiavi simmetriche con cui verranno cifrati e firmati i dati. Questo per evitare l'uso della crittografia a chiave pubblica, meno efficiente in termini prestazionali. Ogni Secure Channel ha una durata limitata. Una volta scaduta è necessario avviare una nuova procedura di apertura del canale e generare delle nuove chiavi simmetriche.

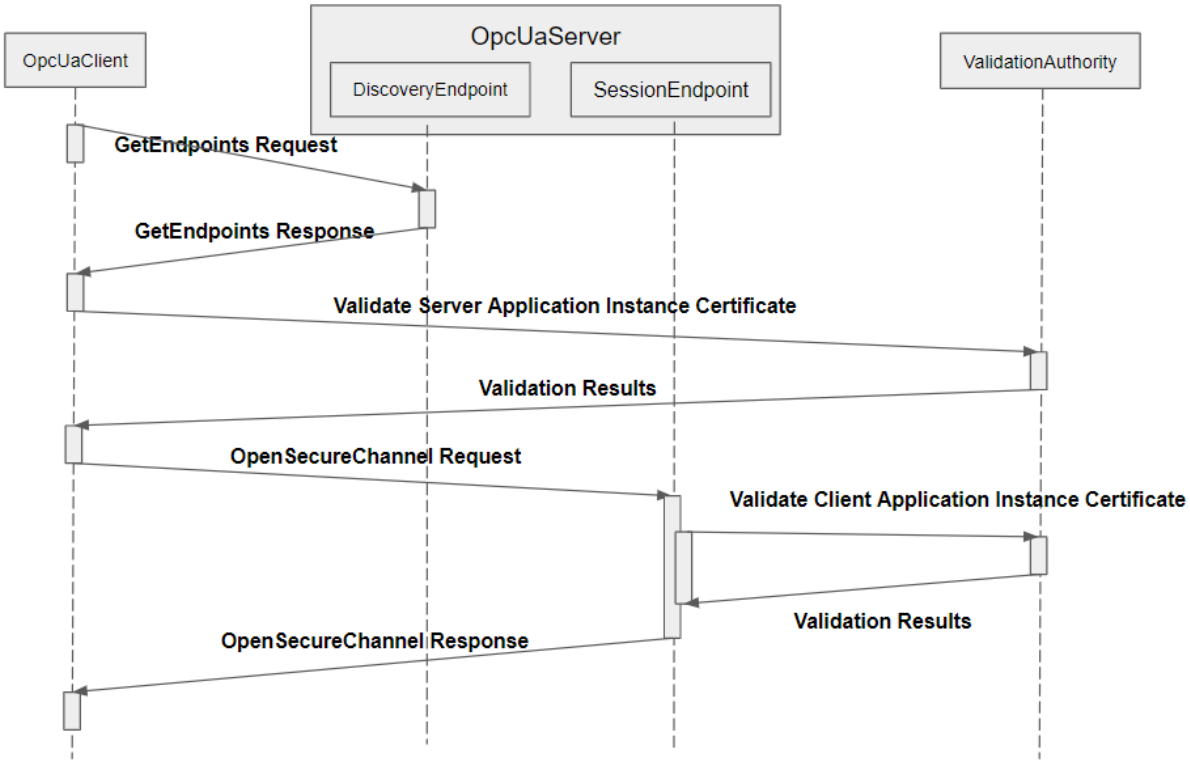

Figura 24 - Creating an OPC UA Secure Channel

3. Il client instaura una session con il server attraverso l'invio di una richiesta *CreateSession* che risponde fornendo i propri Software Certificates per comunicare le proprie funzionalità e per dimostrare il possesso del certificato utilizzato nella creazione del *Secure Channel* (channel response);

4. Infine il client invia una richiesta di *ActivateSession*, che include le credenziali dell'user corrente e i suoi Software Certificates, in modo da attivare la sessione. Le credenziali possono essere un certificato X.509 (validato da un VA) oppure una coppia username-password. Una volta validate le credenziali e i Software Certificates viene attivata la session e il client può accedere ai dati di processo del server.

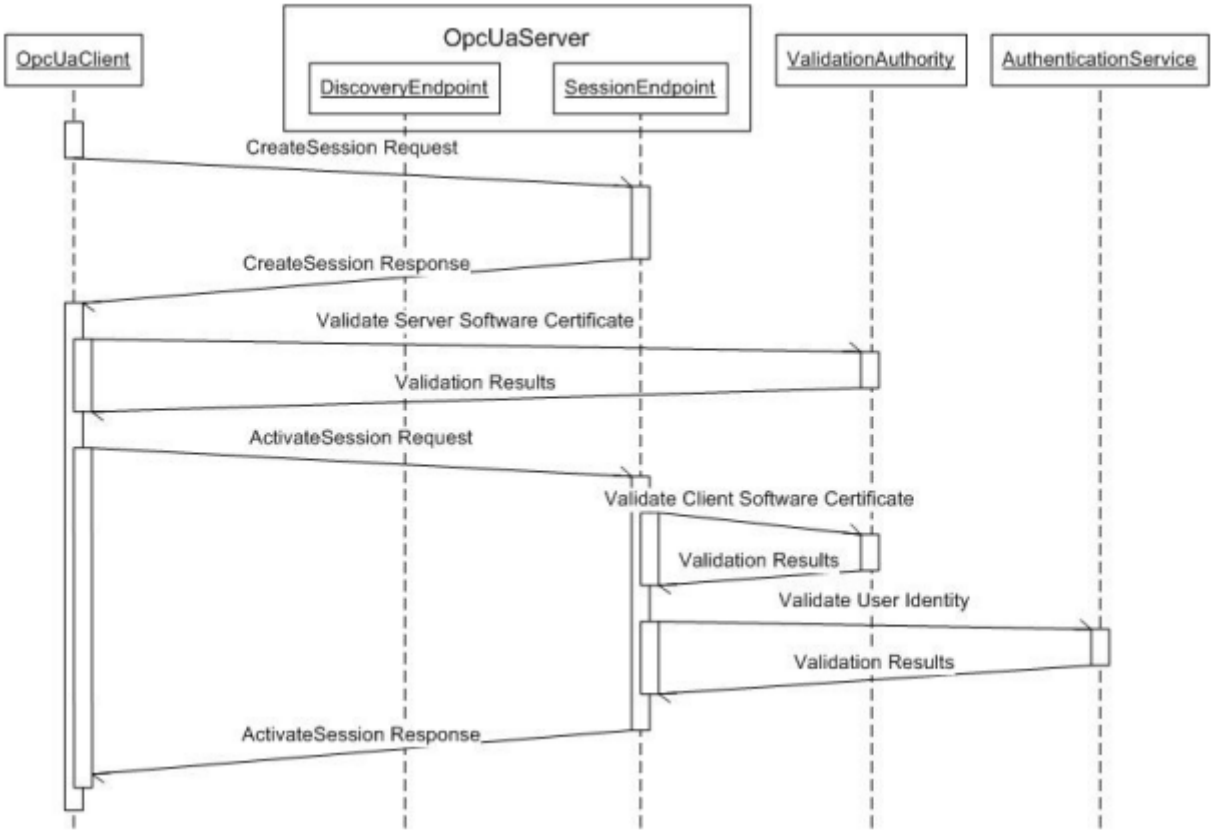

Figura 25 - Establishing an OPC UA Session.

L'intera connessione viene terminata scambiando dei messaggi di chiusura, *CloseSession* e *CloseSecureChannel*. Entrambi i messaggi sono protetti con chiavi simmetriche, tuttavia, OPC UA specifica che i messaggi *CloseSecureChannel* devono essere solo firmati poiché non vengono trasmessi dati sensibili.

Quando si verifica un errore, il server invia un messaggio di errore al client e chiude la connessione senza problemi. Quando un client riceve un messaggio di errore lo segnala all'applicazione e chiude la connessione. Se, invece, un client riscontra un errore, lo deve segnalare all'applicazione e inviare al server un messaggio *CloseSecureChannel*. Il server terminerà così la comunicazione.

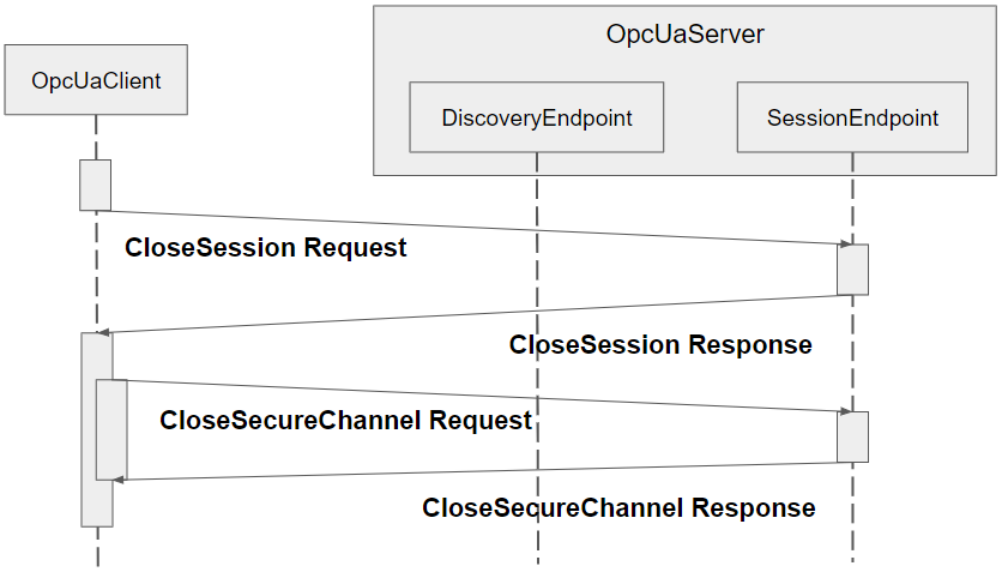

Figura 26 - Closing an OPC UA communication

Capitolo 4

4. OPC UA Security Assessment

OPC UA è stato esplicitamente progettato con l'idea di security by design, tuttavia richiede una configurazione ben studiata per soddisfare i requisiti di sicurezza che il mondo industriale richiede.

Infatti, la complessità intrinseca di OPC UA e un'ampia gamma di modelli implementativi rendono questa configurazione un compito laborioso e soggetto a errori. Di conseguenza, esiste la necessità di valutare la sicurezza delle implementazioni OPC UA, e in particolare le implementazioni che sono (non) intenzionalmente connesse a Internet, dove diversi migliaia di bot (sonde) sono alla ricerca di vulnerabilità sfruttabili.

In questo capitolo si andrà prima ad analizzare il traffico di rete scambiato tra due applicazioni OPC UA, utilizzando Wireshark per la cattura del traffico dati, e poi a studiare le configurazioni di sicurezza di un tipico server OPC UA con *Metasploit Framework*, uno strumento di Vulnerability Assessment.

L'ambiente di test è stato realizzato in locale su un Sistema Operativo Linux, il Server OPC UA è stato sviluppato con uno script in Python seguendo lo stack di "FreeOpcUa" che è possibile trovare sulla piattaforma GitHub. Di seguito si riportano gli strumenti utilizzati con le rispettive versioni.

Applicativo – Strumento	Tipo - Versione
Server OPC UA	Free OpcUa Python GitHub [commit hash: 67f15551884d7f11659d52483e7b932999ca3a75]
Kali Linux	Release: 2022.1
Python	Version: 3.9.10
Wireshark	Version: 3.6.0
Metasploit Framework	Version: 6.1.31-dev

4.1. Analyzing OPC UA Communications with Wireshark

La cattura e l'analisi dei pacchetti (messaggi) tra un client e un server OPC UA permette di capire meglio il funzionamento di creazione di un canale di comunicazione (Secure Channel), nonché la comunicazione stessa, e che tipo di informazioni è possibile derivare; soprattutto se l'encryption non è attiva.

A tale scopo si utilizza *Wireshark*, un software per analisi di protocollo o packet sniffer, che permette di osservare tutto il traffico scambiato sulla rete. Per la visualizzazione dei pacchetti specifici OPC UA basta utilizzare il filtro `opcua` nella "barra di ricerca". Inoltre per assicurarsi che *Wireshark* acquisisca effettivamente i dati è necessario specificare quale porta viene utilizzata dall'endpoint del server OPC UA.

In un primo caso si analizzano i pacchetti scambiati con un server privo di impostazioni di sicurezza, nel secondo, invece, si analizzano i pacchetti di un server con encryption attiva. In entrambi i casi è presente l'autenticazione del client tramite credenziali (username e password).

- **CASO 1: NO ENCRYPTION**

Nel nostro caso l'endpoint è "`opc.tcp://127.0.0.1:4840`" quindi la porta è la `4840`, che in Wireshark è già impostata di default.

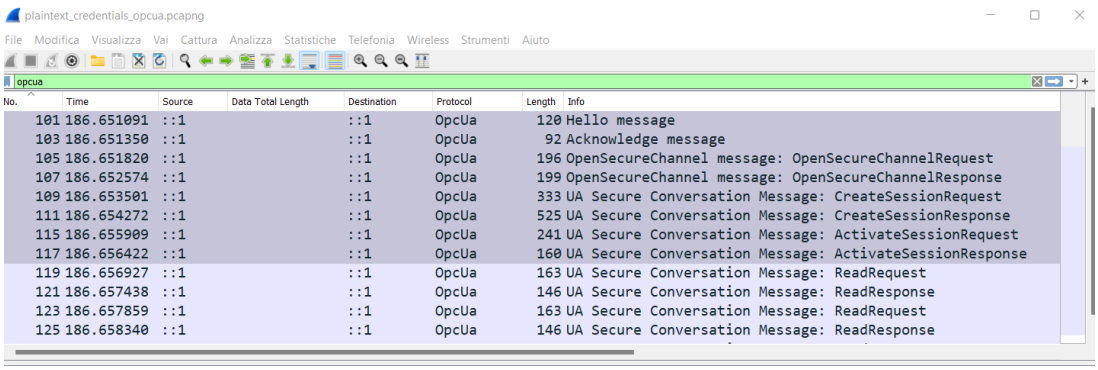

Figura 27 - Opening OPC UA SecureChannel message sequence, captured with Wireshark, no-Encryption.

La comunicazione si apre sempre con l'invio di un messaggio "`Hello`" da parte del client, in cui vengono specificate le dimensioni del buffer supportate. Il `SendBufferSize` specifica la dimensione dei chunks per lo scambio dei messaggi tramite la connessione.

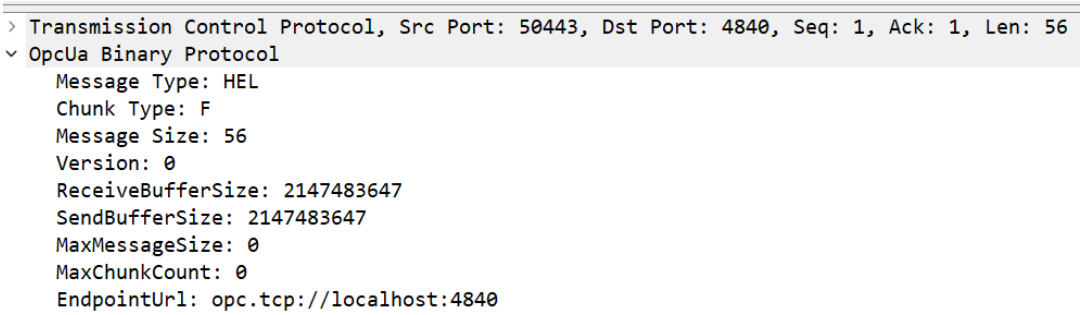

Figura 28 - Informations exchanged in a 'HELLO' message

Il server risponde con un messaggio di "`Acknowledge`" che completa la negoziazione del buffer. La dimensione del buffer verrà poi segnalata al livello SecureChannel.

Il Client una volta ricevuto l'ACK dal Server, invia un messaggio di `OpenSecureChannelRequest` con il quale richiede l'apertura di un canale sicuro. Qui è possibile individuare la richiesta del `SecurityToken`.

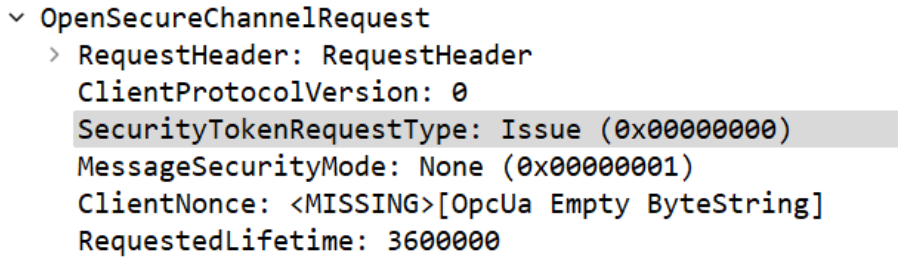

Figura 29 - Parameters in a OpenSecureChannel Request.

Il Server risponde con un messaggio di `OpenSecureChannelResponse` attraverso il quale comunica al Client il `SecurityToken`. Successivamente quindi Client e Server comunicheranno con un `SecureChannelId: 6` utilizzando un `SecurityTokenId: 13`.

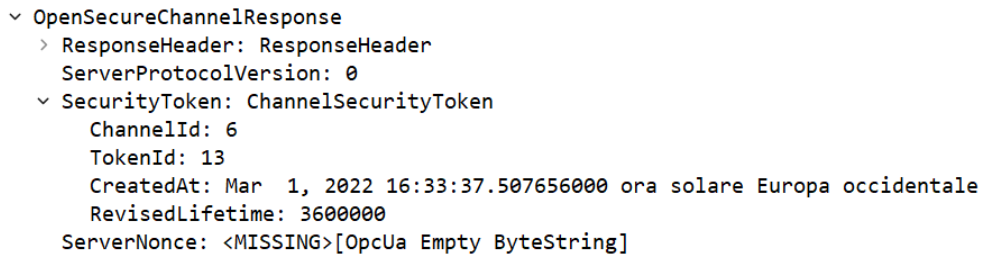

Figura 30 - Parameters in a OpenSecureChannel Response.

I pacchetti successivi riguardano la creazione e attivazione di una sessione con cui le due applicazioni, Client e Server, comunicheranno.

Nella richiesta `CreateSession` è possibile individuare, oltre all'`ApplicationDescription` (Figura 31), il `ClientNonce` e il `ClientCertificate` se presente (Figura 32).

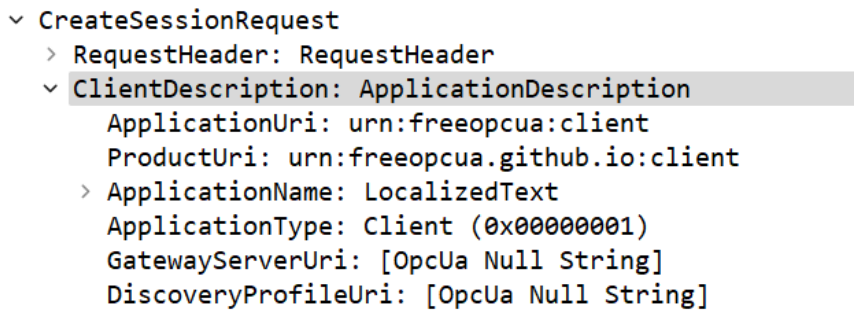

Figura 31.

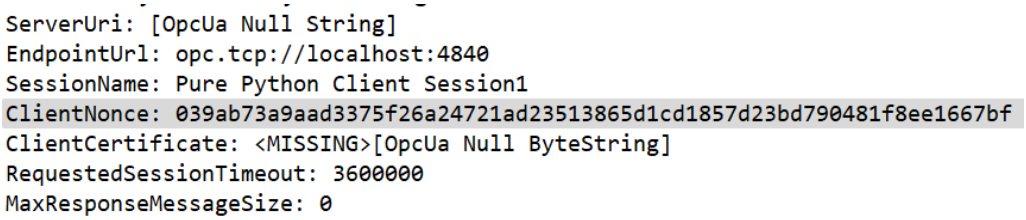

Figura 32.

Il Client avanza quindi la richiesta di `ActivateSession`, nella quale si distinguono i campi relativi a `ClientSignature`, `UserIdentityToken` e `UserTokenSignature`. Se l'autenticazione avviene tramite username e

password, nel campo `UserNameIdentityToken` è possibile leggere in chiaro le credenziali che il Client invia al Server.

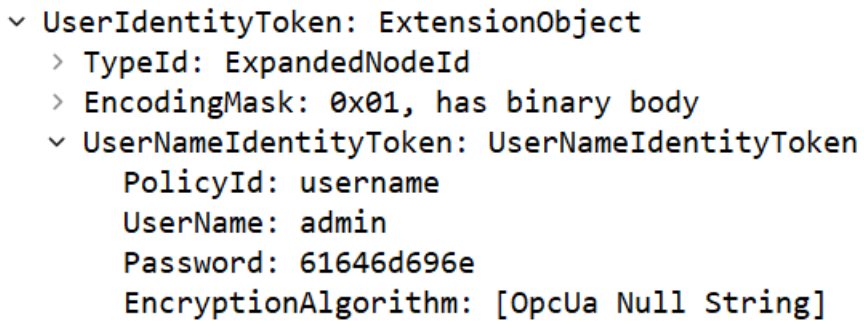

Figura 33 - Credentials passed by the Client to the Server when activating the Session.

Nella risposta l'unico campo di reale interesse è il `ServerNonce` (Figura 34).

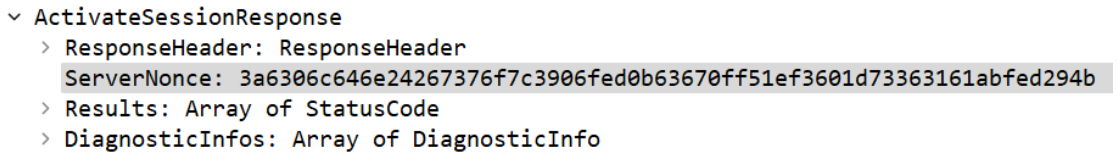

Figura 34.

Per quanto riguarda le richieste di scrittura e lettura è facile individuare il `NodeId` nella `ReadRequest` (Figura 35) e il valore richiesto nella `ReadResponse` (Figura 36).

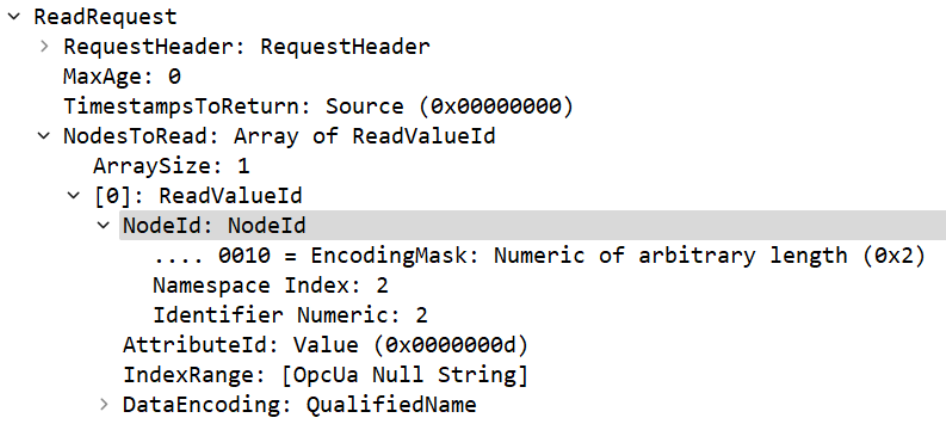

Figura 35 - Parameters in a ReadRequest.

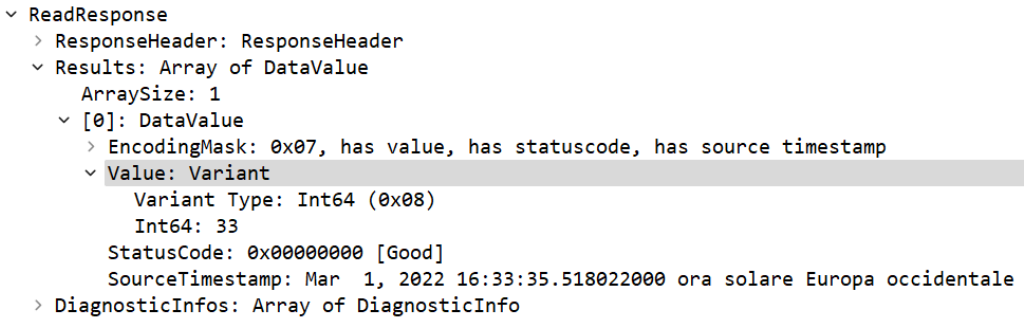

Figura 36 - Parameters in a ReadResponse.

- **CASO 2: WITH ENCRYPTION**

Per questo caso è stata utilizzata una policy di tipo `Basic256Sha256_SignAndEncrypt` che utilizzerà dei certificati e delle chiavi private per criptare i messaggi e scambiarli in modo sicuro. Si nota subito che la sequenza dei messaggi per la creazione di un canale di comunicazione, rispetto a prima, è molto differente.

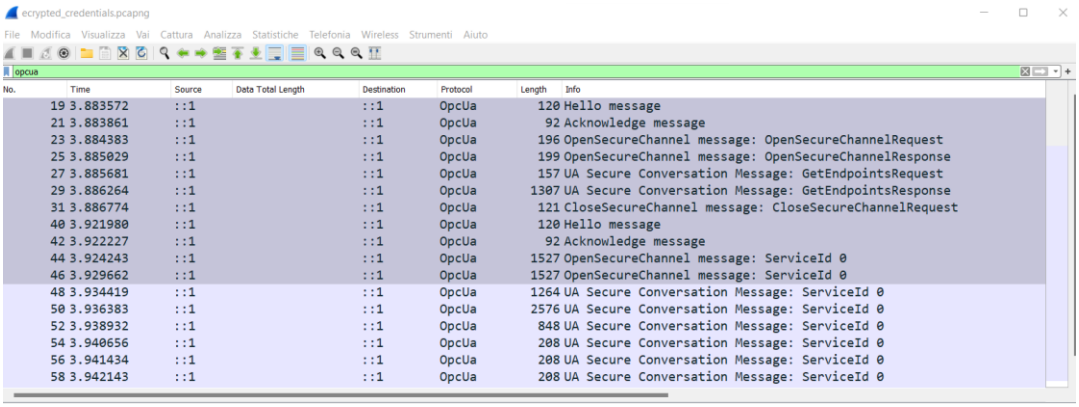

Figura 37 - Opening a SecureChannel message sequence, captured with Wireshark, Encrypted.

In particolare si nota che il Client richiede l'accesso agli Endpoints per capire quali sono quelli che supportano la stessa `SecurityPolicy` (`Basic256Sha256_SignAndEncrypt`); altrimenti sarà costretto a rinunciare alla comunicazione. L'informazione più rilevante è la descrizione degli Endpoints (`EndpointDescription`) che il Server restituisce al Client nella `GetEndpointsResponse`, da cui è possibile ricavare il `ServerCertificate` utilizzato per la cifratura dei messaggi (Figura 38).

```
Endpoints: Array of EndpointDescription
  ArraySize: 1
  [0]: EndpointDescription
    EndpointUrl: opc.tcp://::1:4840
    Server: ApplicationDescription
      ApplicationUri: urn:freeopcua:python:server
      ProductUri: urn:freeopcua.github.io:python:server
      ApplicationName: LocalizedText
      ApplicationType: ClientAndServer (0x00000002)
      GatewayServerUri: [OpcUa Null String]
      DiscoveryProfileUri: [OpcUa Null String]
      DiscoveryUrls: Array of String
    ServerCertificate: 3082034e30820236a0030201020209009b1c24d61fb16147300d06092a864886f70d0101…
    MessageSecurityMode: SignAndEncrypt (0x00000003)
    SecurityPolicyUri: http://opcfoundation.org/UA/SecurityPolicy#Basic256Sha256
    UserIdentityTokens: Array of UserTokenPolicy
      ArraySize: 1
      [0]: UserTokenPolicy
        PolicyId: username
        UserTokenType: UserName (0x00000001)
        IssuedTokenType: [OpcUa Null String]
        IssuerEndpointUrl: [OpcUa Null String]
        SecurityPolicyUri: [OpcUa Null String]
    TransportProfileUri: http://opcfoundation.org/UA-Profile/Transport/uatcp-uasc-uabinary
    SecurityLevel: 0
```

Figura 38

Una volta individuati gli Endpoints il Client chiude il canale per poi ricrearne un altro in cui viene impostata la `SecurityPolicy` su `Basic256Sha256`. Il valore del `ServerCertificate` se decriptato restituisce il certificato originale utilizzato.

ServerCertificate:3082034e30820236a0030201020209009b1c24d61fb16147300d060
92a864886f70d01010b05003023121301f06035504030c18707974686f6e2d6f70637
5612e6578616d706c652e6f7267301e170d3136303130343233303832335a170d32353
039323333233303832335a30233121301f06035504030c18707974686f6e2d6f7063756
12e6578616d706c652e6f726730820122300d06092a864886f70d01010105000382010
f003082010a02820101009f68307ba4afdf98ffc8ec8bac48af254c20673d9821a886d
52edd901d800ca0417509a178178af1a36be5d4cb7c830b2ae13de8b1015932f9cb607
6fb28dc94a098138cbc494000efdaee10dd4dfc85e0332b94e9e2bcb83029eec8922a0
0f2b51a2aeec04ab8cd0af1f96cd58b1f28b4edfb4ae5bdb73925aa534632201804bd1
304a1ef4b95cc6e33f589d04c07c7ab333dbdc3f4d54271012a3d2f09d663adb54f2b2
829b6de64712acbe5fe31099c14fae7f3edca7418bc2944c27eb211bd6092450716f48
ad3c5b32e75bc0c84b5d042ee380053de9955e434ea7af20ead43d84a6854c125a8dc8
be3a268de59ca6c6aae18acd7cd061404ddf6fec7030203010001a38184308181301d0
603551d0e04160414e28fdbc59ba038f2864f5af7b8e95a8acd2be954301f0603551d2
3041830168014e28fdbc59ba038f2864f5af7b8e95a8acd2be954300c0603551d13040
530030101ff30310603551d11042a3028862675726e3a6578616d706c652e6f72673a4
67265654f706355613a707974686f6e2d6f70637561300d06092a864886f70d01010b0
50003820101000057a803a22d49eed2dae7691f6b2779b4f6deaadc8c2888c538b5a5f
a6ef0bd1ae1094fea20c376050dfd079f57ffb51a285e1235bcffa4d7d50b1cb86c156
d5f15ec6ff5784aabd32a6f73b601cff75d37a565b55390ddb5623a03e0970a7e5a053
1927bd1d5189033f8f834d5f13d28ec67c53b4d97a32897743bf1bbdf71b8af9943c14
1556fb28175d87e3ed605506390de9aa6497ab5cfae2c909202c87661e8c75c15613e9
171aae8b7db78c324648b074af3ec02d5743777176870dca4fa6c4f1e27b11ca3e9677
921be4c4300c18534e5a4bc52f829e90573e837ec2ef977baf31994089efe8cfd5d6c8
14fd7295e40bd1c307109babe79f61a78d2

Parse X.509 certificate – Input format: DER Hex

```
Version: 3 (0x02)
Serial number: 1117684887730004819 9 (0x009b1c24d61fb16147)
Algorithm ID: SHA256withRSA
Validity
Not Before: 04/01/2016 23:08:23 (dd-mm-yyyy hh:mm:ss) (160104230823Z)
Not After: 23/09/2025 23:08:23 (dd-mm-yyyy hh:mm:ss) (250923230823Z)
Issuer
CN = python-opcua.example.org
Subject
CN = python-opcua.example.org
Public Key
Algorithm: RSA
Length: 2048 bits
Modulus: 9f:68:30:7b:a4:af:df:98:ff:c8:ec:8b:ac:48:af:25:
         4c:20:67:3d:98:21:a8:86:d5:2e:dd:90:1d:80:0c:a0:
         41:75:09:a1:78:17:8a:f1:a3:6b:e5:d4:cb:7c:83:0b:
         2a:e1:3d:e8:b1:01:59:32:f9:cb:60:76:fb:28:dc:94:
         a0:98:13:8c:bc:49:40:00:ef:da:ee:10:dd:4d:fc:85:
         e0:33:2b:94:e9:e2:bc:b8:30:29:ee:c8:92:2a:00:f2:
         b5:1a:2a:ee:c0:4a:b8:cd:0a:f1:f9:6c:d5:8b:1f:28:
         b4:ed:fb:4a:e5:bd:b7:39:25:aa:53:46:32:20:18:04:
         bd:13:04:a1:ef:4b:95:cc:6e:33:f5:89:d0:4c:07:c7:
         ab:33:3d:bd:c3:f4:d5:42:71:01:2a:3d:2f:09:d6:63:
         ad:b5:4f:2b:28:29:b6:de:64:71:2a:cb:e5:fe:31:09:
         9c:14:fa:e7:f3:ed:ca:74:18:bc:29:44:c2:7e:b2:11:
         bd:60:92:45:07:16:f4:8a:d3:c5:b3:2e:75:bc:0c:84:
         b5:d0:42:ee:38:00:53:de:99:55:e4:34:ea:7a:f2:0e:
         ad:43:d8:4a:68:54:c1:25:a8:dc:8b:e3:a2:68:de:59:
         ca:6c:6a:ae:18:ac:d7:cd:06:14:04:dd:f6:fe:c7:03
Exponent: 65537 (0x10001)
Certificate Signature
Algorithm: SHA256withRSA
Signature: 00:57:a8:03:a2:2d:49:ee:d2:da:e7:69:1f:6b:27:79:
           b4:f6:de:aa:dc:8c:28:88:c5:38:b5:a5:fa:6e:f0:bd:
           1a:e1:09:4f:ea:20:c3:76:05:0d:fd:07:9f:57:ff:b5:
```

```
            1a:28:5e:12:35:bc:ff:a4:d7:d5:0b:1c:b8:6c:15:6d:
            5f:15:ec:6f:f5:78:4a:ab:d3:2a:6f:73:b6:01:cf:f7:
            5d:37:a5:65:b5:53:90:dd:b5:62:3a:03:e0:97:0a:7e:
            5a:05:31:92:7b:d1:d5:18:90:33:f8:f8:34:d5:f1:3d:
            28:ec:67:c5:3b:4d:97:a3:28:97:74:3b:f1:bb:df:71:
            b8:af:99:43:c1:41:55:6f:b2:81:75:d8:7e:3e:d6:05:
            50:63:90:de:9a:a6:49:7a:b5:cf:ae:2c:90:92:02:c8:
            76:61:e8:c7:5c:15:61:3e:91:71:aa:e8:b7:db:78:c3:
            24:64:8b:07:4a:f3:ec:02:d5:74:37:77:17:68:70:dc:
            a4:fa:6c:4f:1e:27:b1:1c:a3:e9:67:79:21:be:4c:43:
            00:c1:85:34:e5:a4:bc:52:f8:29:e9:05:73:e8:37:ec:
            2e:f9:77:ba:f3:19:94:08:9e:fe:8c:fd:5d:6c:81:4f:
            d7:29:5e:40:bd:1c:30:71:09:ba:be:79:f6:1a:78:d2
Extensions
subjectKeyIdentifier: e28fdbc59ba038f2864f5af7b8e95a8acd2be954
authorityKeyIdentifier: kid=e28fdbc59ba038f2864f5af7b8e95a8acd2be954
basicConstraints:cA = true
subjectAltName: uri:urn:example.org:FreeOpcUa:python-opcua
```

Per quanto riguarda la creazione della sessione di comunicazione e lo scambio delle credenziali per l'autenticazione, invece, avviene tutto in maniera criptata; come anche la scrittura e lettura di variabili.

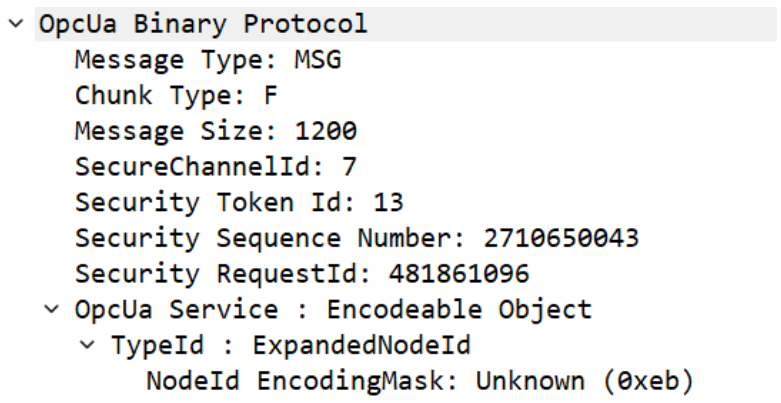

Figura 39 - With encryption no data can be read.

Con questa prima analisi si conferma la necessità di implementare nei dispositivi, oltre alle features di autenticazione, l'utilizzo di certificati e di chiavi private per l'encryption. In questo modo si garantisce la riservatezza dei dati e per un attaccante sarà difficile interpretare i messaggi scambiati. È sconsigliata, dunque, la `SecurityPolicy` di tipo `None` a meno che il dispositivo non si trovi su una rete isolata.

4.2. OPC UA Vulnerability Assessment: Metasploit Framework

Metasploit mette a disposizione diversi moduli per effettuare un Vulnerability Assessment, che vanno da moduli per PLC, ad esempio Schneider Modicon o Siemens S7, su software SCADA, a protocolli industriali, ad esempio Modbus, Profinet, ecc; ed è ampiamente utilizzato per le valutazioni in ambienti industriali. Lo stesso NIST (*National Institute of Standards and Technology*) lo ha utilizzato per valutare le prestazioni di sistemi industriali.

A questo scopo è necessario integrare degli script specifici per OPC UA, in quanto Metasploit non contiene alcun modulo relativo a questo tipo di applicazioni. Cercando sul web è possibile trovare degli script realizzati da alcuni ricercatori dell'Università di Aachen, Germania (*Linus Roepert, Markus Dahlmanns, Ina Berenice Fink, Jan Pennekamp, Martin Henze*) che permettono di:

- Scoprire i server OPC UA in una rete;
- Verificare la presenza di credenziali di accesso anonime, predefinite o deboli;
- Recuperare informazioni sul server e sulle configurazioni delle policy di sicurezza, nonché diritti di accesso;

- Verificare la suscettibilità a determinate vulnerabilità note (*CVE – Common Vulnerabilities and Exposures*).

4.2.1. Discovery OPC UA Servers

Come primo passo è necessario scansionare la rete per scoprire i server OPC UA. Per fare questo è possibile utilizzare il comando *Nmap,* presente su Metasploit, per effettuare un port scanning, che individua tutte le porte aperte su un dispositivo bersaglio, o anche su un range di indirizzi IP, e determina quali servizi di rete sono attivi (Figura 40).

Di default, OPC UA utilizza la porta TCP 4840 per il suo protocollo binario e le porte TCP 80 (HTTP) e 443 (HTTPS) quando funge da servizio Web.

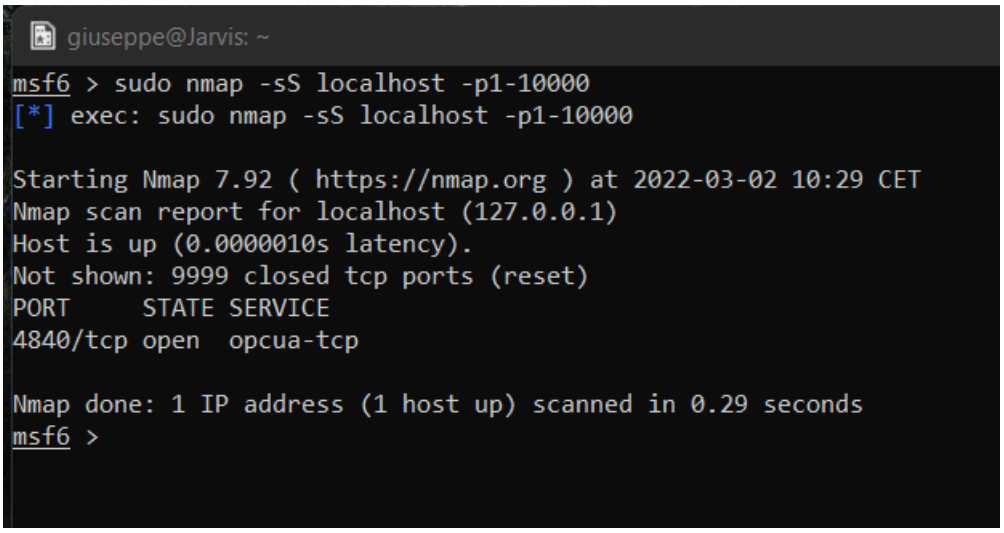

Figura 40 - Scanning the network with Nmap command.

Per verificare che i dispositivi rilevati abbiano effettivamente un servizio OPC UA attivo su dette porte, si utilizza l'estensione `opcua_hello`, con la quale si effettua un semplice handshake con il server (Figura 41).

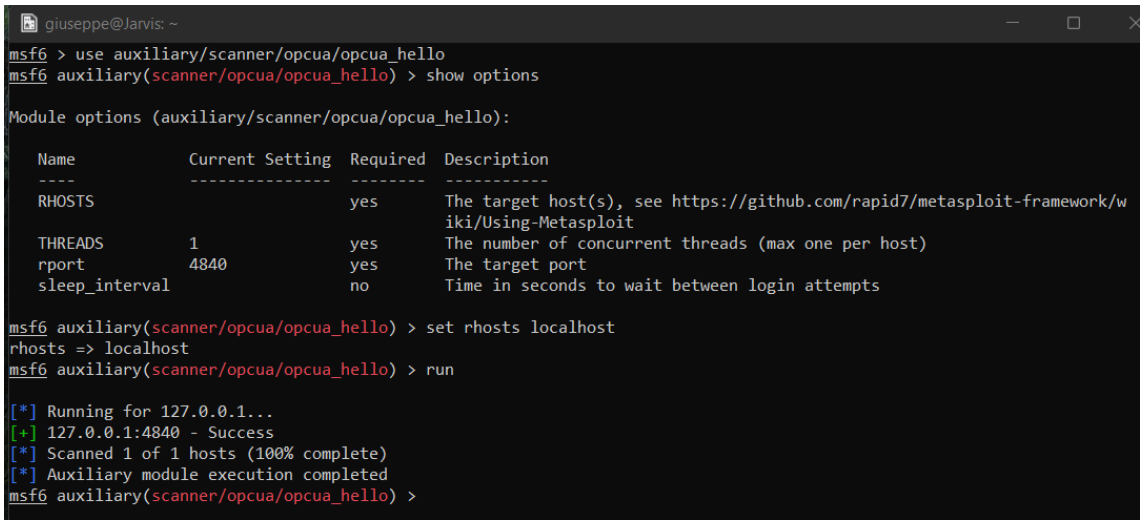

Figura 41 - Setting the opcua_hello options and running the script.

4.2.2. Testing OPC UA Authentication

Una volta individuato un server OPC UA, il passaggio successivo consiste nel verificare se è presente un'autenticazione e se essa è configurata in modo sicuro. Come abbiamo visto, OPC UA consente diverse modalità di autenticazione. In particolare, l'accesso anonimo e l'autenticazione con username e password sono soggetti a errori di configurazione.

L'accesso anonimo (credenziali di accesso vuote) potrebbe rivelare informazioni sensibili sulla configurazione del server e permettere l'accesso ai dati con conseguente perdita di informazioni o addirittura il rischio di operazioni di scrittura non autorizzate.

Gli accessi basati su username e password rischiano di essere deboli in quanto vengono utilizzate spesso credenziali di accesso predefinite (documentate sui manuali) oppure facilmente intuibili e soggette ad attacchi di tipo brute force.

Per verificare tali problemi è sufficiente lanciare l'estensione `opcua_login_scanner` integrata per le implementazioni OPC UA. È possibile inoltre utilizzare un file contenente le credenziali predefinite, specifiche di OPC UA e liberamente accessibili da istruzioni e manuali di configurazione, oppure un

qualsiasi altro file contenente un dizionario con le credenziali più utilizzate e facilmente disponibile sul web (Figura 42). Similmente si può verificare se il server accetta un certificato client self-signed senza ulteriore convalida.

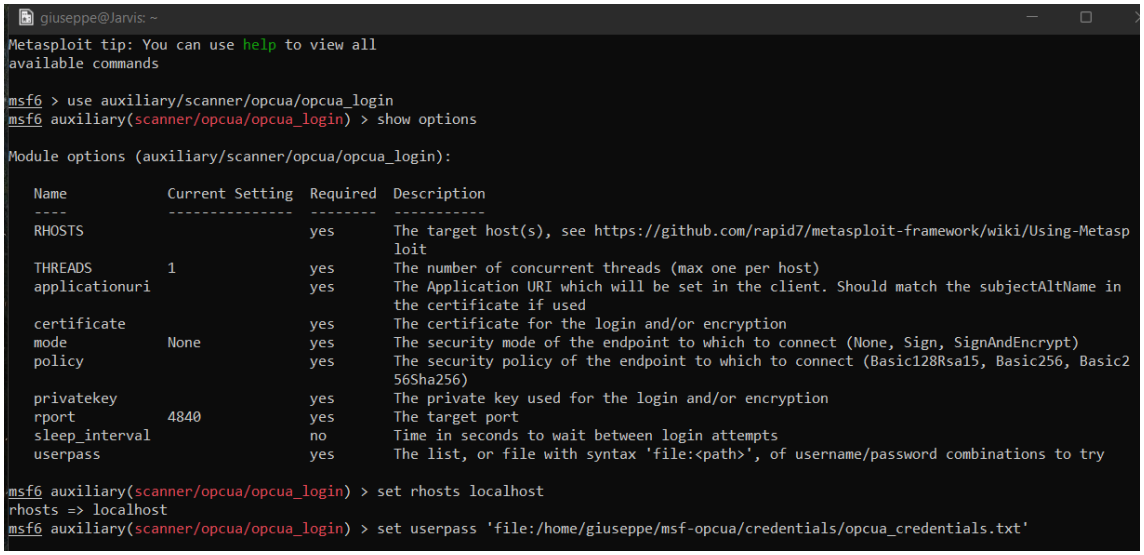

Figura 42 - Setting options for opcua_login script.

Una volta impostate le opzioni desiderate e lanciato lo scanning, Metasploit restituisce l'esito dell'autenticazione. In particolare si nota che è stato possibile connettersi con successo e quali credenziali sono state utilizzate per l'autenticazione (Figura 43). Ovviamente le credenziali sono state volutamente scelte "deboli" per la dimostrazione, ma il framework utilizza un set di credenziali vario che ricopre la maggior parte di quelle predefinite dai principali rivenditori OPC UA sul mercato. Inoltre è sempre possibile modificare il file e aggiungere nuove usernames e password per testare credenziali più solide. È importante rimarcare che nel web sono facilmente reperibili repository costantemente aggiornate che contengono credenziali di autenticazione di sistemi in uso, talvolta sottratte in modo fraudolento. A tal proposito è bene ricordare che una adeguata policy per la gestione delle credenziali nei sistemi industriali, ad esempio il cambio periodico delle password, è di vitale importanza.

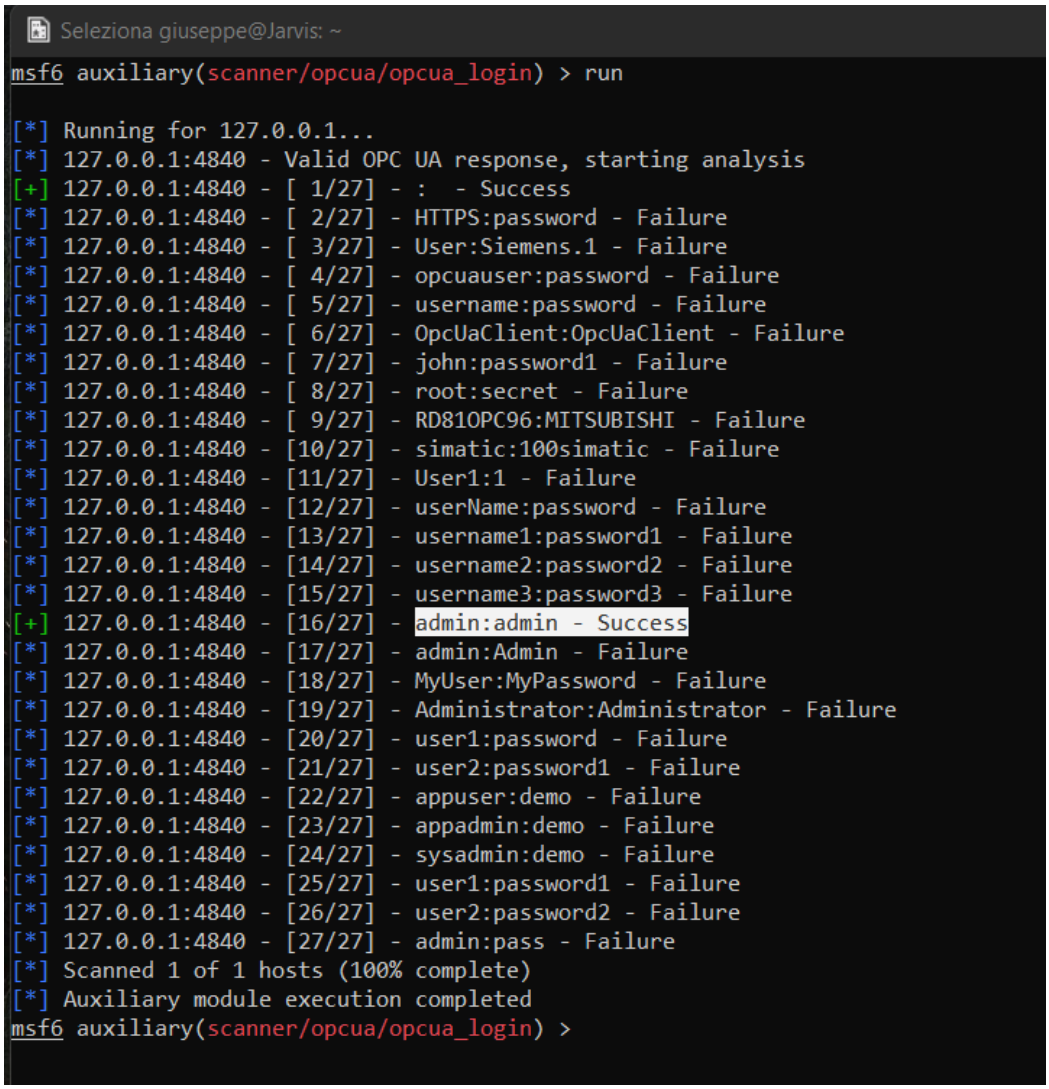

Figura 43 - Running the opcua_login script.

4.2.3. Deriving OPC UA Server Information & Configuration

Se è possibile accedere a un server OPC UA (in modo anonimo, utilizzando username e password o tramite certificato), possiamo connetterci e ottenere informazioni attraverso l'estensione `opcua_server_config` (Figura 44).

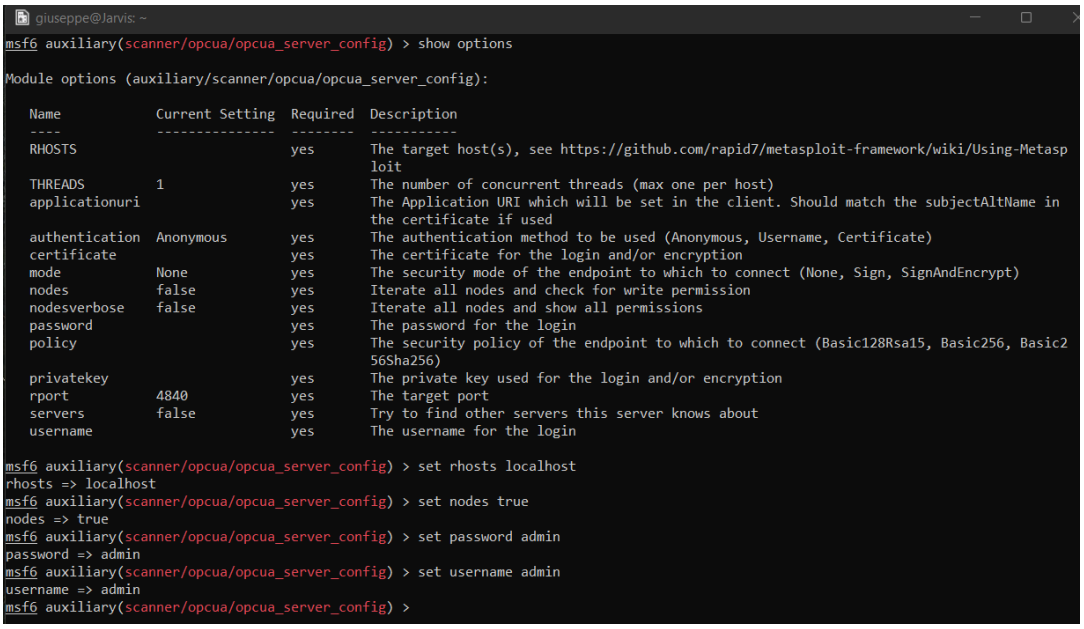

Figura 44 - Setting options for opcua_server_config script with the credentials obtained from the previous step.

Le informazioni sui server OPC UA disponibili e conosciuti possono essere utilizzate per ampliare l'ambito della valutazione; informazioni come `ApplicationUri` e `ProductUri` forniscono dettagli riguardo al software utilizzato e alla sua versione, che permette la verifica delle policy delle patch e il controllo di potenziali vulnerabilità.

Altre informazioni riguardano quale `SecurityLevel` viene utilizzato, la `MessageSecurityMode` utilizzata per la crittografia, la politica di sicurezza specificata da `PolicyUri` e il `TokenType` utilizzato per l'autenticazione.

L'estensione (`opcua_server_config`) permette anche di iterare tutti i nodi disponibili, derivando quali sono leggibili e quali scrivibili, e quindi identificare i permessi configurati in modo errato (Figura 44).

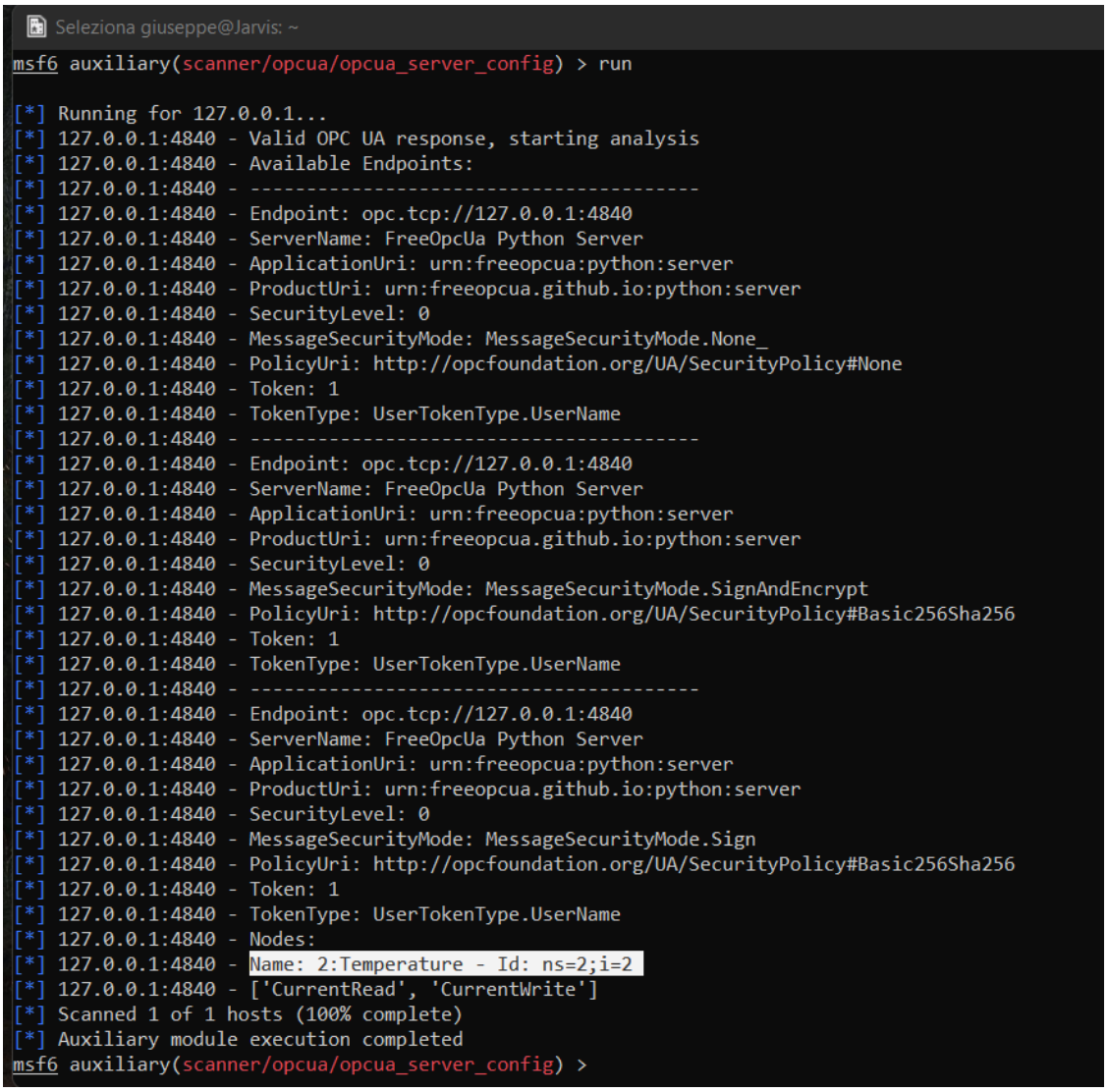

Figura 45 - Informations obtained from the Server using the opcua_server_config.

4.2.4. Checking for Potential Vulnerabilities

Una volta ottenute le informazioni di configurazione di un server OPC UA si è in grado di verificare la presenza di potenziali vulnerabilità note (CVE). A tale scopo, Metasploit consente di eseguire exploit per CVE specifici su un server; inoltre, alcuni server OPC UA (soprattutto quelli che permettono l'accesso anonimo) sono soggetti a un attacco di Denial-Of-Service (DOS), ovvero un attacco informatico in cui l'attaccante cerca di impedire l'accesso alle risorse del server da parte dei client.

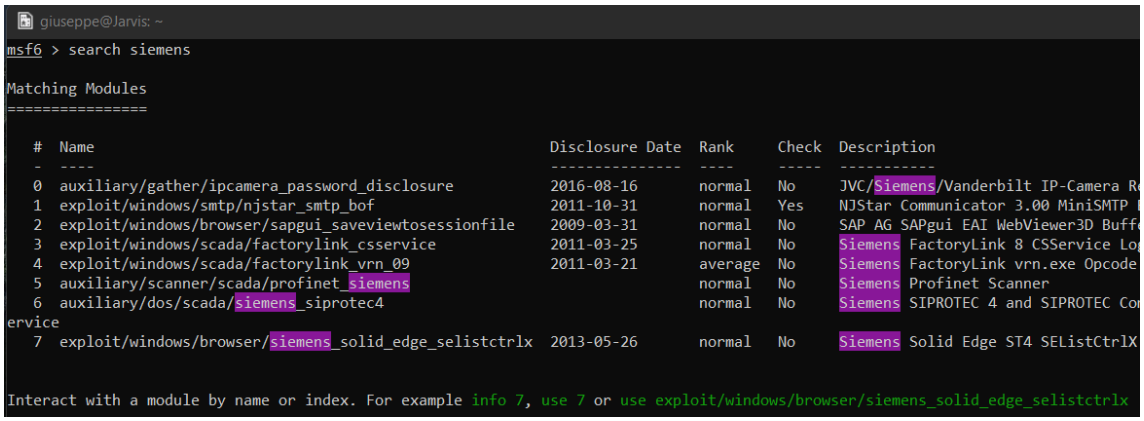

Figura 46 - Exploit examples offered by Metasploit that can be used in Industrial Environment.

Inoltre, eseguendo una ricerca su Shodan, un motore di ricerca studiato appositamente per individuare i dispositivi collegati ad internet, chiamato anche il motore di ricerca del mondo Internet of Everything, è facile trovare dispositivi OPC UA connessi ad internet che, probabilmente, possono presentare delle vulnerabilità. Inoltre, il fatto che siano indicizzati e quindi facilmente identificabili ne aggrava il rischio di compromissione.

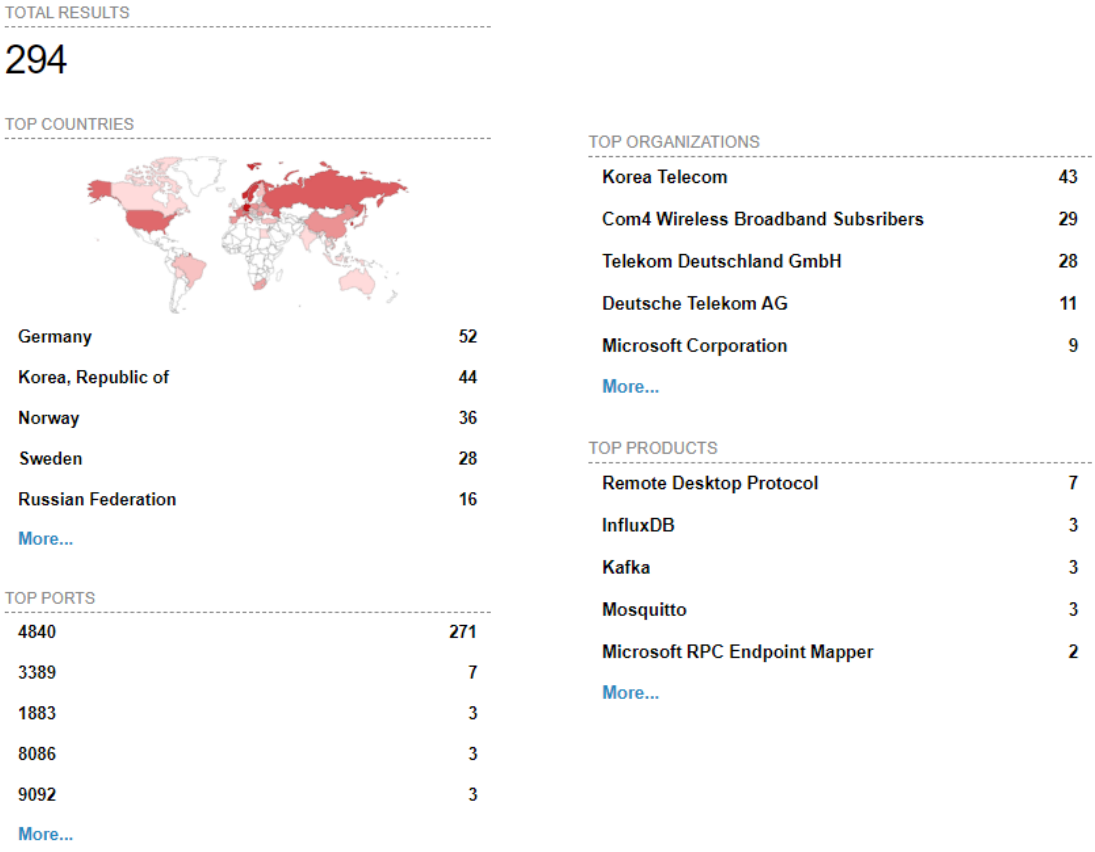

Figura 47 - A research on Shodan shows-up OPC UA devices exposed on the Internet.

Capitolo 5

5. Conclusioni

Il mondo dell'Industria ad oggi ha bisogno di protocolli di comunicazione in grado di rispondere a diverse necessità, quali la flessibilità, la scalabilità e, soprattutto, la Sicurezza. Si è visto che OPC UA è capace di adattarsi a diverse piattaforme grazie alla sua struttura ed è in grado di soddisfare i requisiti di sicurezza su diversi livelli, supportando features di autenticazione, autorizzazione e encryption; garantendo quindi confidenzialità e integrità dei dati.

Il tema della Cyber Security però, rimane aperto. È vero che OPC UA permette di configurare i dispositivi con diverse funzionalità di sicurezza, ma il fattore umano è sempre il punto debole principale. Una configurazione minima e una povera conoscenza del protocollo inducono a commettere errori di implementazione, che consentono ad un attaccante di individuare facilmente vulnerabilità che possono essere sfruttate per creare danni, anche gravi. La sensibilizzazione delle persone sulle tematiche della Cyber Security è un'attività che deve essere presente in qualsiasi realtà aziendale ed è anche necessario investire sulle attività di analisi per la prevenzione delle minacce informatiche.

In questo documento si è voluto fornire un quadro generale di quello che il protocollo OPC UA mette a disposizione, presentare un metodo di analisi delle implementazioni OPC UA, in grado di assistere gli operatori nella configurazione ottimale di tali applicativi, e formare i lettori sulle principali minacce informatiche in ambito manifatturiero con alcuni esempi di eventuali conseguenze.

L'analisi della Security è stata condotta su un prototipo in esecuzione su un personal computer. Ciononostante, le considerazioni, metodologie e i risultati ottenuti rimangono validi anche per un ambiente industriale reale. L'attività si è concentrata particolarmente sull'information gathering, ovvero l'acquisizione su larga scala di informazioni rilevanti per la sicurezza in merito al sistema in esame. Le informazioni ottenute risultano particolarmente importanti per una corretta scelta della configurazione di sicurezza o per lo sviluppo di eventuali exploit

specifici. Infatti, ad oggi, la mancanza di exploit specifici per applicazioni OPC UA in Metasploit non ha permesso di effettuare degli attacchi veri e propri sull'ambiente di test utilizzato.

Le future attività prevedono dunque la possibilità di integrare tale funzionalità e di estendere l'analisi ad applicazioni industriali reali.

Riferimenti Bibliografici

[1] Wolfgang Mahnke, Stefan-Helmut Leitner, Matthias Damm, *OPC Unified Architecture*, Springer, 2009.

[2] *Online version of OPC UA specifications and information models*, *https://reference.opcfoundation.org/#Core*.

[3] Rocket Systems, *OPC UA Tutorial Series,* gennaio 2021, https://www.youtube.com/playlist?list=PLWw98q-Xe7iGf-c4b6zF0bnJA9avEN_mF.

[4] *GitHub – mailrocketsystems/OPCUA*, https://github.com/mailrocketsystems/OPCUA, [commit hash: e296a407534e71eba2b35adb004afa0b1915277f].

[5] *GitHub – FreeOpcUa/python-opcua*, https://github.com/FreeOpcUa/python-opcua,

[commit hash: 67f15551884d7f11659d52483e7b932999ca3a75].

[6] *GitHub – FreeOpcUa/opcua-asyncio*, https://github.com/FreeOpcUa/opcua-asyncio,

[commit hash: 04a85e2c5bd16c353e445ad8b691bd3868a69954].

[7] GitHub – *Publicly available OPC UA Servers and Clients,* https://github.com/node-opcua/node-opcua/wiki/publicly-available-OPC-UA-Servers-and-Clients, 14 Mar. 2021.

[8] Giulia Madeddu, *La sicurezza informatica nei Sistemi di Controllo Industriale delle aziende manifatturiere*, 5 gennaio 2019, https://www.dgroove.it/la-sicurezza-informatica-nei-sistemi-di-controllo-industriale-delle-aziende-manifatturiere/3974/.

[9] *IBM – X-Force Threat Intelligence Index*, 2022, https://www.ibm.com/security/data-breach/threat-intelligence/.

[10] Redazione Data Manager Online, *L'importanza della sicurezza informatica nei Sistemi di Controllo Industriale*, 23 gennaio 2018, https://www.datamanager.it/2018/01/limportanza-della-sicurezza-informatica-nei-sistemi-controllo-industriale/.

[11] GitHub – COMSYS/msf-opcua, https://github.com/COMSYS/msf-opcua, [commit hash: a3d4fedf91ca59055a083c8047cdc7a1de3cbe7e].

[12] L. Roepert, M. Dahlmanns, Ina B. Fink, J. Pennekamp, M. Henze, *Assessing the Security of OPC UA Deployments*, 2020, https://www.comsys.rwth-aachen.de/fileadmin/papers/2020/2020-roepert-opcua-security.pdf.

[13] OPC Foundation, *OPC UA Profile Reporting Application*, https://profiles.opcfoundation.org/.

[14] K. Rutherford, OPC Connect – News and Views from the OPC Foundation, *Analyzing OPC UA Communications with Wireshark*, Febbraio 2017, https://opcconnect.opcfoundation.org/2017/02/analyzing-opc-ua-communications-with-wireshark/.