Università di Perugia
Dipartimento di Matematica e Informatica

Tesi triennale in informatica

Analisi della sicurezza del middleware Data Distribution Service

Relatore
Prof. Francesco Santini

Laureando
Federico Ranocchia

Anno accademico 2023/2024

A mia nonna e alla sua pazienza.

Indice

Capitolo 1

Introduzione

In questo elaborato verrà esposto il middleware Data Distribution Service (DDS) standardizzato dall'Object Management Group (OMG) e verrà effettuata un'analisi sulla sua sicurezza. In particolare nel Capitolo 2 viene spiegato inizialmente il suo funzionamento, per poi passare ai successivi due capitoli dove verranno introdotte due tipologie di vulnerabilità: una basata sullo standard OMG, mentre l'altra sugli applicativi che utilizzano il DDS. Infine nel Capitolo 5 verranno mostrati dei tools in grado di analizzare il traffico di una rete DDS e inoltre verrà introdotto DDSFuzz, un altro strumento per eseguire un test di tipo fuzz.

Il DDS di OMG viene adoperato in numerosi campi grazie alla sua comunicazione efficiente che può essere impostata anche per funzionare in sistemi che hanno bisogno di risposte real-time. L'architettura è basata su un sistema di comunicazione distribuito che si segue su modello publish/subscribe in modo tale da poter rendere flessibile la topologia della rete. Questo vantaggio porta il DDS ad essere utilizzato in vari campi che hanno necessità di aggiornare continuamente la loro struttura. Viene adottato in medicina, in ambienti militari, in controlli industriali, in veicoli autonomi e nelle telecomunicazioni.

Un'applicazione molto nota del DDS è ROS Robot Operating System uno framework che permette a molti ingegneri robotici che hanno bisogno di effettuare uno scambio di informazioni di interfacciarsi con vari dispositivi all'interno di un sistema di sensori che necessitano di comunicare tra di loro. Ad esempio un utente, che ha il compito di automatizzare un robot può ricevere diversi segnali contemporaneamente, inclusi movimento dei giunti (espressi in coordinate), velocità, posizione geografica,

stato del robot, temperatura di un determinato componente e i rimanenti sensori di cui il robot è dotato. Il DDS all'interno di ROS è utilizzato come middleware e si occupa di stabilire e mostrare i dati che vengono mandati dai componenti della rete. In questo modo l'utente ha la possibilità di visualizzare comodamente in un unico punto tutte le informazioni necessarie senza dover creare nuovi canali di comunicazione.

Tuttavia, nonostante le numerose applicazioni il DDS presenta alcune problematiche a livello di sicurezza che possono essere sfruttate da un attore malevolo. Una delle caratteristiche, chiamato processo di discovery, più utilizzate è la possibilità di permettere a nuovi partecipanti di connettersi ad una rete DDS senza l'intervento di un operatore. Diverse vulnerabilità che verranno analizzate successivamente all'interno dell'elaborato, nel Capitolo 3 e nel Capitolo 4, mostrano come il processo di discovery, se non adeguatamente configurato da i vendors delle implementazioni DDS, possa essere sfruttato come vettore di attacco. Durante l'esecuzione dell'autoscoperta è possibile immettere pacchetti malevoli che possono compromettere il corretto funzionamento di uno o più dispositivi all'interno della rete stessa. Inoltre, un attaccante in ascolto che riesce a catturare il traffico generato da questo processo è in grado di effettuare una ricognizione del network in modo da ricevere abbastanza metadati, tra cui il nome del topic, per comprendere al meglio il funzionamento di ogni singolo partecipante collegato.

Un'altra caratteristica vantaggiosa per il DDS che può essere adoperata da un attore malevolo sono le policy QoS. Esse servono per impostare diversi criteri di comportamento che i dispositivi all'interno della rete devono seguire, inclusa la priorità di pubblicazione. Se un attore malevolo riesce ad ottenere l'accesso ai valori di questi parametri, potrebbe acquisire il controllo sui f lussi di dati, manipolando i valori stessi. Tale manipolazione potrebbe consentire all'attaccante di reindirizzare i dati verso entità di ricezione, attraverso una policy alterata, che gestisce le priorità in modo da favorire le sue azioni. Ciò può compromettere una o più operazioni dei dispositivi che elaborano i dati ricevuti

OMG per mitigare queste problematiche relative alla sicurezza ha creato un nuovo standard chiamato DDS security che viene gestito come una estensione per il middleware DDS. Il DDS security introduce una serie di plugin relativi all'autenticazione, crittografia e gestione degli accessi. Questi plugin vengono adoperati come misure di protezione dei messaggi scambiati per garantirne l'integrità e l'impossibi-

lita di effettuare manipolazioni indesiderate. Il processo di autenticazione effettuato dal DDS security ha il compito di garantire che non ci siano partecipanti della rete non autorizzati impedendo ad un attaccante di leggere il contenuto del loro traffico. Invece, la gestione degli accessi si occupa di certificare che i dispositivi all'interno del network dispongano dei permessi necessari per compiere determinate azioni.

Il DDS security in questo elaborato verrà proposto diverse volte come la soluzione ottimale per la mitigazione di certe vulnerabilità. Tuttavia, non viene ancora impiegato in molte implementazioni, data la difficoltà di configurazione iniziale e l'aumento delle risorse richieste durante le operazione crittografiche che risultano molto difficoltose specialmente per dispositivi di tipo IoT (internet of things).

In altri casi il DDS security non viene adoperato perché il network della rete DDS si trova all'interno di un sistema chiuso, isolato da internet o con un accesso limitato (dietro un firewall). Un esempio di questa configurazione si può trovare all'interno di una fabbrica che collega i vari macchinari in modo tale che possano comunicare tra di loro non permettendo la loro disponibilità al di fuori di quell'ambiente.

Nel Capitolo 2 di questo elaborato si introdurrà il funzionamento del middleware DDS partendo dal funzionamento di un modello di tipo publish/subscribe, confrontandolo con il più comune protocollo TCP/IP. Il DDS viene definito data-centric, poichè pone l'importanza sui dati stessi semplificandone la sua configurazione. Verrà inoltre introdotto il concetto delle policy QoS e del Global Space Data per permettere alle entità della rete di comunicare tra di loro senza preoccuparsi dell'accessibilità e disponibilità dei dati. Si analizzerà la sua struttura, tutte le sue entità utilizzate introducendo il protocollo RTPS (Real-Time Publish-Subscribe Protocol) e i moduli che lo compongono. Successivamente verrà introdotto il DDS security insieme ai suoi plugin e al loro funzionamento inclusi l'Authentication Service Plugin e l'Access Control Service Plugin, mostrando nel dettaglio come avviene il processo di autenticazione e di controllo permessi. Infine verranno presentate diverse implementazioni DDS dei vendor, confrontando tra di loro soluzioni gratuite e a pagamento.

Nel Capitolo 3 ci occuperemo di analizzare le vulnerabilità del DDS che si trovano nello standard OMG. Queste non vengono presentate tramite una implementazione a differenza del Capitolo 4, ma solo attraverso l'utilizzo dello standard OMG per evidenziare falle di sicurezza più generali del middleware. Le prime due vulnerabilità impiegano entrambe il protocollo RTPS per essere applicate, in particolare nel pri-

mo caso verrà introdotta una vulnerabilità che permette ad un attore malevolo di compiere una ricognizione della rete, mentre nel secondo caso verrà mostrato come la modifica del valore sequence number all'interno di un pacchetto RTPS potrebbe compromettere il funzionamento di un DataReader. Nel successivo attacco dimostreremo che si può creare un vettori di attacco anche impiegando il DDS security, durante la fase di autenticazione di un'entità all'interno della rete. Infine le ultime due vulnerabilità che verranno spiegate, si basano sulla misconfigurazione di due policy QoS in modo tale da consentire ad un attaccante di manipolare gli aggiornamenti di un flusso di dati e impedire ad una o più entità di ricevere nuove informazioni riguardanti un topic.

Nel Capitolo 3 verranno introdotte altre vulnerabilità, questa volta basate sulle implementazioni software dei vari vendors DDS. Si analizzeranno tre casi che saranno relativi a ROS che impiegherà il DDS come middleware. Inizialmente viene spiegata una libreria, che successivamente verrà fusa con il progetto Scapy di Python, che facilità la costruzione di pacchetti RTPS. La prima vulnerabilità che verrà mostrata consente di effettuare una ricognizione della rete DDS mandando un pacchetto di discovery ai vari applicativi DDS. Il secondo attacco che verrà introdotto consente di dirottare il traffico del processo di discovery verso un entità a scelta dell'attaccante. Questo attacco è possibile dato che lo standard OMG non consente di effettuare un controllo degli indirizzi IP che devono ricevere questo messaggi di tipo discovery. Infine nell'ultimo attacco che verrà mostrato, l'implementazione Fast DDS ha una vulnerabilità data la mancanza di controllo di un parametro che può essere impostato con lunghezza pari a 0. Costruendo un pacchetto RTPS con i giusti parametri sarà possibile mandare in crash ogni entità della rete che lo riceve.

Nel Capitolo 5, l'ultimo capitolo di questo elaborato, verranno presentati degli strumenti di analisi e test del middleware che possono essere compatibili anche con altri protocolli o solamente utilizzabili dal DDS. Il primo tool che verrà introdotto è WireShark, un famoso applicativo che si occupa di sniffare i pacchetti scambiati all'interno di una rete, nel nostro caso siamo interessati a quelli del protocollo RTPS. Infatti successivamente verrà mostrato come si suddivide uno di questi pacchetti tra HEADER e sottomessagi in modo tale da capire come funziona questo tool. Un'alternativa che verrà mostrata è eProsima DDS Record & Replay che ha la caratteristica di catturare i pacchetti RTPS e salvarli in un formato specializzato per la lettura di

dati serializzati. Anche il tool che verrà presentato Fast DDS Spy è stato prodotto dallo stesso sviluppatore, eProsima, e si occupa sempre di analizzare il traffico di rete con la caratteristica di riuscire a capire quali policy QoS siano attive. L'ultimo strumento che verrà indicato è DDSFuzz, un applicativo per effettuare test di tipo fuzz al DDS. Prima di questo tool non esistevano software simili, quindi dopo la sua creazione sono stati scoperte altre vulnerabilità che altri fuzzer più generici non riuscivano a identificare dato che non sfruttavano appieno le caratteristiche del DDS, come la possibilità di cambiare la topologia della rete.

Capitolo 2

Introduzione al DDS

In questo capitolo viene fatta un'introduzione generale dello standard del Data Distribution Service (DDS) gestita dall'Object Management Group (OMG) [33], inizialmente a livello generale per poi andare sempre più nel dettaglio e capire il suo funzionamento. Questo ci sarà utile per comprendere le vulnerabilità che verranno analizzate nei successivi capitoli. Inoltre verrà introdotta la sua estensione DDS security che si occupa di rendere più sicuro lo standard DDS aggiungendo l'autenticazione e una implementazione della cifratura dei pacchetti scambiati tra i vari dispositivi connessi nella rete. Infine verrà mostrato in quali contesti attuali il DDS viene utilizzato con le sue diverse implementazioni.

In questa tesi potrebbe venire omessa la specifica OMG perché faremo riferimento esclusivamente al DDS conforme all'Object Management Group. Questo standard ha delle specifiche tecniche ben precise, consentendo l'interoperabilità tra i diversi vendor che lo rispettano, insieme a tanti altri numerosi vantaggi [17].

2.1 Modello publish/subscribe

Prima di parlare del DDS, è necessario capire il funzionamento del modello publish/subscribe che sta alla base del suo funzionamento. Questo modello di publish e subscribe non funziona come la classica applicazione che siamo abituati a vedere tra server e client tramite protocollo *TCP/IP* nell'ambito delle comunicazioni.

2.1.1 Perché non usiamo una connessione con TCP/IP

Prendiamo l'esempio di un collegamento tra una workstation e un sensore per la temperatura, come mostrato in Figura 2.1. La connessione a livello fisico avverrà tramite un collegamento Ethernet. La workstation e il sensore quindi si trovano nello stesso network e possono ora cominciare a comunicare tra di loro. L'obiettivo è quello di trasferire i dati dal sensore alla workstation in modo tale da poterli visualizzare a schermo. La metodologia più frequente è quella di utilizzare un **socket tramite protocollo TCP/IP**, ma non sempre questa è la soluzione migliore. Alcuni vantaggi del TCP/IP sono la disponibilità di utilizzo in molte applicazioni e nella maggior parte delle connessioni a Internet. Tuttavia, in certe implementazioni, TCP/IP non risulta la soluzione migliore, specialmente quando dobbiamo collegare un numero di dispositivi al network che può variare nel tempo. Se nel nostro network tra workstation e sensore della temperatura, aggiungiamo un altro dispositivo, ad esempio un sensore per la temperatura, bisognerebbe creare un nuovo socket TCP/IP per far comunicare il nuovo sensore con la workstation. Questo è necessario perché TCP/IP supporti una comunicazione di tipo one-to-one (uno a uno). Come vedremo nella prossima sottosezione, il modello publish/subscribe non ha questa limitazione [27].

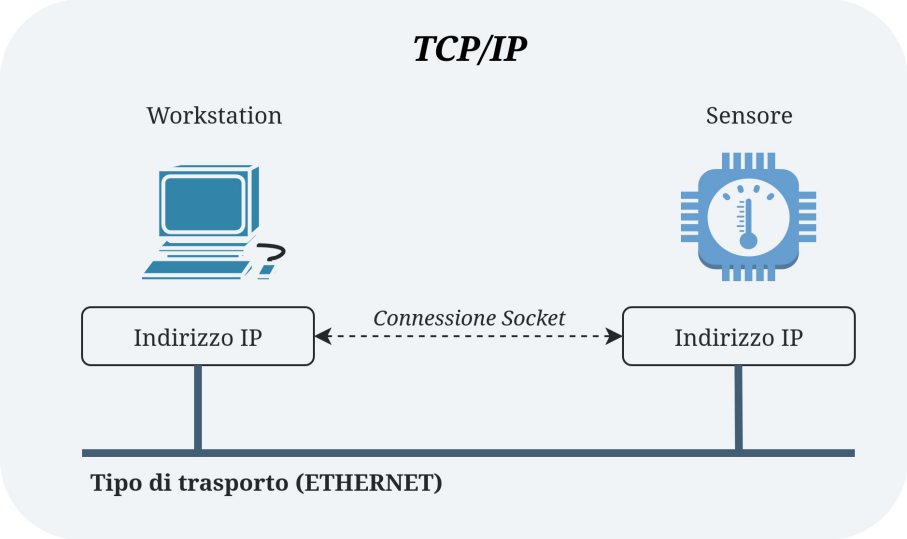

Figura 2.1: Un esempio di collegamento tra una workstation e un sensore della temperatura, utilizzando il protocollo TCP.

2.1.2 Struttura modello publish/subscribe

Per funzionare, il modello publish/subscribe, ha bisogno di due elementi chiamati *publisher e subscriber*, che permettono lo scambio di messaggi all'interno del network. La comunicazione avviene quando hanno un **topic**, che rappresenta una tipologia di dati (ad esempio la temperatura, la distanza e la velocità) è in comune tra di loro.

- **Publisher:** colui che pubblica nuovi dati riguardanti dei topic rendendoli accessibili ai subscriber iscritti. Di solito si tratta di un sensore.
- **Subscriber:** è colui che si iscrive ai topic del publisher, cominciando così a ricevere nuovi dati sul topic scelto. Spesso si tratta di un dispositivo utilizzato per visualizzare informazioni, come uno schermo.

Chiamiamo entità tutti i dispositivi che fanno parte del modello publish/subscribe. Una caratteristica di queste entità è che possono essere aggiunte o rimosse da una rete senza nessun problema, dato che le comunicazioni avvengono in modalità asincrona [17].

Inoltre questo modello risulta molto flessibile e adatto in ambienti real-time dove le informazioni possono cambiare o essere utilizzate da più dispositivi. I subscriber, ad esempio possono cambiare i topic a cui sono iscritti e i publisher possono smettere di pubblicare nuovi aggiornamenti su un determinato topic anche a runtime [20].

2.2 Che cos'è il Data Distribution Service

Il DDS gestito da OMG è un middleware e uno standard API con una gestione dei dati di tipo **data-centric**. Prendendo in considerazione il modello ISO OSI (Open Systems Interconnection), questo middleware è un software che si trova tra l'applicativo e il livello del trasferimento, mostrato in Figura 2.2.

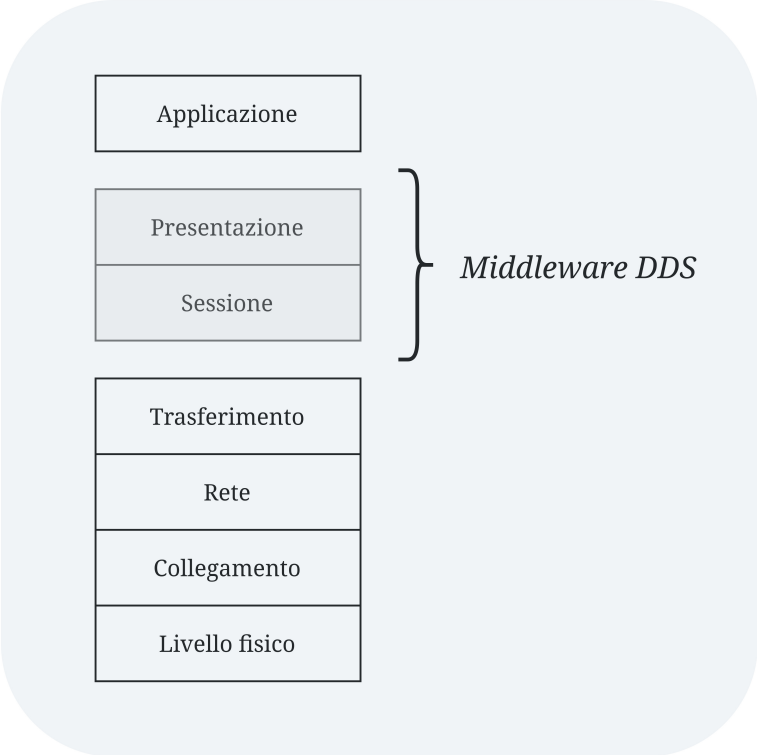

Figura 2.2: Posizione del middleware DDS nel modello ISO/OSI.

Il DDS si basa sul DCPS (Data-Centric Publish-Subscribe) che è un modello di comunicazione simile a quello di tipo publish/subscribe, ma con un approccio più data-centric, in modo tale da semplificare il lavoro del programmatore che si deve solamente occupare di specificare il contesto del dato che deve mandare o ricevere. Con data-centric infatti, specifichiamo che il focus del modello DCPS è incentrato sui dati stessi, anziché sulle entità che li scambiano [22]. Così facendo non bisogna preoccuparsi dell'invio o della ricezione dei messaggi, perché questa parte viene completamente gestita dal middleware.

Altri vantaggi del DDS includono un'**architettura adattabile** che supporti degli elementi di *auto-scoperta (auto-discovery) o Dynamic Discovery* in modo tale da aggiungere o rimuovere dispositivi dalla rete in modo automatico, anche a runtime. Il Dynamic Discovery ha il compito di analizzare quali tipologie di dati ha bisogno questo nuovo dispositivo.

Inoltre ogni nuovo partecipante utilizzerà sempre le stesse API per comunicare con l'applicativo dato che non c'è bisogno di configurare le impostazioni degli indirizzi

IP o preoccuparsi delle diverse architetture [27].

2.2.1 Quality of Service (QoS)

Per soddisfare i diversi requisiti di una trasmissione dati, il Data Distribution Service (DDS) utilizza un insieme di policy *Quality of Service (QoS)*. Queste policy permettono di controllare, regolare e ottimizzare lo scambio di dati tra i vari componenti all'interno del middleware che possono variare significativamente in base al tipo di comunicazione richiesta, offrendo una **gestione altamente flessibile e granulare**. Ogni elemento del middleware può essere configurato con policy specifiche, in modo tale da adattarsi alle esigenze dell'applicazione.

2.2.2 Global Space Data

Il DDS utilizza il Global Data Space (spazio dati globale), *un'area logica condivisa* che consente agli applicativi di accedere a una sorta di memoria locale tramite API. L'applicativo nella scrittura o nella ricezione dei dati utilizzerà questa memoria locale fittizia assumendo il ruolo di un'**unica risorsa centralizzata**. Tuttavia, i dati all'interno di questa memoria possono contenere informazioni provenienti da nodi remoti distribuiti per la rete. L'applicativo *non deve così preoccuparsi dell'accessibilità dei dati*, poiché questi vengono gestiti agendo come se si trovassero tutti in unico punto [22].

2.2.3 Architettura DDS

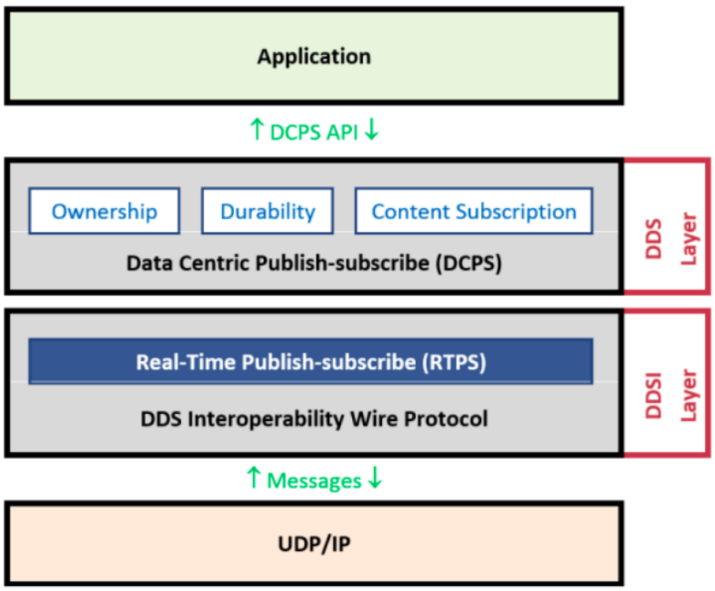

Figura 2.3: Struttura dei layer del DDS [31].

Lo standard DDS definito dall'OMG è composto da due layer: il DDS e il DDSI (DDS Interoperability), mostrati in Figura 2.3.

- **DDS:** è il layer fondamentale in cui troviamo il DCPS (Data-Centric Publish-Subscribe), un modello di comunicazione simile a quello di tipo publish/subscribe, che si occupa di mettere in comunicazione più applicazioni tra di loro. In questo layer vengono inoltre definite le policy QoS [15].

- **DDSI:** è il layer che si occupa di garantire **l'interoperabilità** tra le diverse implementazioni del DDS, ad esempio quando provengono da vendors diversi. All'interno di questo layer troviamo l'RTPS (Real-Time Publish-Subscribe Protocol), un protocollo che permette ai vari dispositivi DDS di comunicare e scoprirsi tra di loro (Dynamic Discovery). RTPS è il *wire-protocol* (un protocollo che permette lo scambio di messaggi del DDS al layer di trasporto di rete del modello ISO OSI) ufficiale del DDS, con standard definito da OMG, che definisce il formato dei messaggi e impone le regole che permettono una trasmissione

standardizzata di scambio dati. Se questo wire-protocol non fosse presente, le diverse implementazioni del DDS non potrebbero comunicare tra di loro [12].

2.3 Le entità del DCPS

Il layer DDS per operare utilizza le entità definite dal DCPS, che rappresentano gli elementi necessari per il funzionamento dell'intero middleware. Queste hanno il compito di gestire i dati scambiati tra i vari partecipanti all'interno del sistema. Le entità principali del DDS sono: il publisher, il subscriber, il DataWriter, il DataReader, il Topic, la Istanza, il Domain e il Domain Participant. Ognuna di queste entità deve tener conto del suo set di policy QoS configurate che ne definiscono il comportamento. Queste verranno analizzate più nel dettaglio nella Sezione 2.4. Nella Figura 2.4 viene mostrato un esempio di come si relazionano le varie entità DCPS tra di loro.

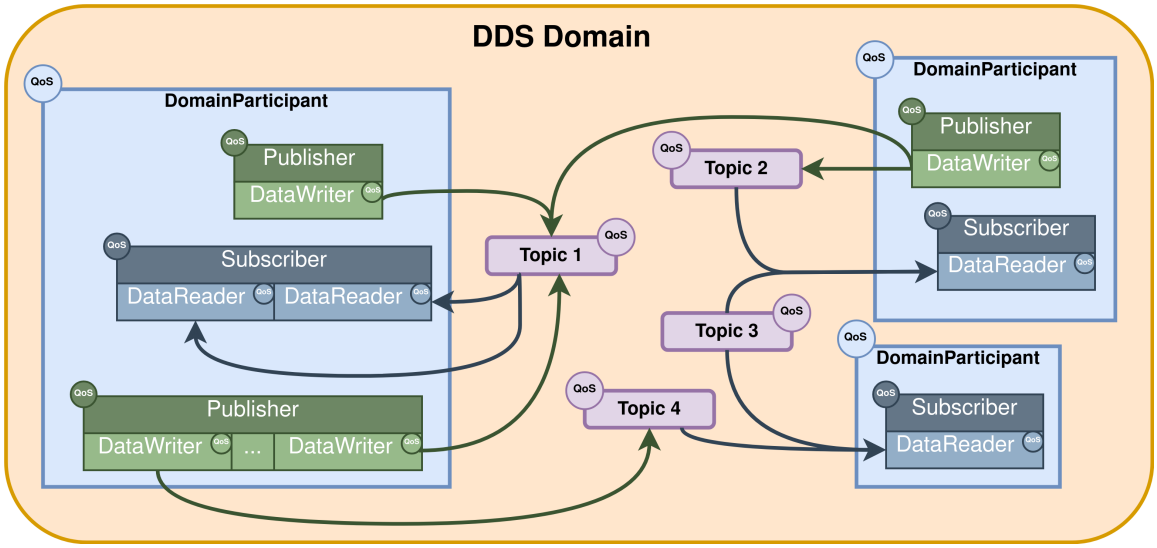

Figura 2.4: Entità DCPS del DDS [6].

Il linguaggio utilizzato da queste entità si chiama *Interface Definition Language (IDL)* e anch'esso è standardizzato da OMG. IDL è un *linguaggio tipizzato* simile a C++ che supporta i seguenti *data types*: char, octet, short, unsigned short, long, unsigned long, long long, unsigned long long, float, double, long double, boolean, enum, array e string [27].

2.3.1 Publisher e Subscriber

Publisher e subscriber sono entità che abbiamo già incontrato in precedenza nel modello publish/subscribe nella Sezione 2.1.2. Esse mantengono lo stesso ruolo all'interno del modello DCPS. Durante l'esecuzione, l'applicativo che si interfaccia con il middleware DDS **non deve occuparsi di gestire le comunicazioni tra publisher e subscriber**, dato che queste verranno gestite automaticamente. Questo consente ai DataWriters e DataReaders di scambiarsi messaggi sfruttando questo canale di comunicazione dinamico, che può cambiare nel tempo a seconda della topologia della rete, senza richiedere alcun intervento dell'applicativo.

2.3.2 DataWriter e DataReader

Il DataWriter e il DataReader sono definite come **interfacce**, poichè l'applicativo può mandare e ricevere dati tramite queste due entità fondamentali. Questi dati scambiati tra middleware e applicativo sono i *data type e i data-values*. I data type consentono di descrivere la struttura e il formato del dato, mentre i data-value sono i dati veri e propri che rispettano le specifiche data type.

- **DataWriter:** è l'interfaccia usata dagli applicativi per spedire i data-values con un loro specifico data type ai publisher. Ricevuti questi data-values il publisher spedirà le informazioni ricevute dall'applicativo ai relativi subscriber.
- **DataReader:** è l'interfaccia usata dagli applicativi per ricevere i data-values con i rispettivi data type pubblicati in precedenza da un publisher [17].

È possibile associare più DataReaders ad un *unico publisher* e più DataWriters ad un *unico subscriber*. Tuttavia, un DataWriter può essere associato solo ad un Publisher, mentre un DataReader solo ad un Subscriber. Questo avviene perché i DataReaders e DataWriters possono utilizzare **un solo data type**, mentre i publisher e i subscribers, non avendo questa limitazione, possono gestire più DataWriters e DataReaders alla volta.

2.3.3 Topic, Key e Istanza

Nel modello publish/subscribe abbiamo già introdotto i **topic**, ma abbiamo la necessità di approfondirli quando vengono utilizzati con le specifiche del DDS.

I topic vengono utilizzati per identificare il data type scambiato tra i publishers e i subscribers, creando cosi un punto di connessione tra DataWriter e DataReader [21]. All'interno del topic troviamo il nome, i data types e le policy QoS. Il nome del topic corrisponde ad una stringa univoca che serve per identificarlo tra gli altri topic del domain.

All'interno di un topic troviamo sempre **una chiave (key)**, che rappresenta un data type del topic stesso.

Queste chiavi ci permettono di suddividere i data-values di un topic, dividendoli a seconda della chiave. Ogni suddivisione che effettuata tramite una key diversa crea **un'istanza** che al suo interno contiene i data-values di un *flusso di dati (data-stream)* [23].

Per fare un esempio prendiamo il topic velocità in un contesto dove si vogliono analizzare i dati di una gara. La struttura del topic avrà due data types: il primo corrisponde al valore della velocità registrata, mentre il secondo mostra l'id_macchina che identifica da quale vettura i dati provengono, come mostrato nell'istanza rappresentata nel Codice 2.3.3.

```
struct Veicolo { // Nome del topic
    id_macchina; // Key del topic
    velocita;
}
```

Codice 2.1: Esempio di Topic con una key usando il linguaggio IDL.

Creiamo ora due istanze, una con valore 270 per la velocità e con l'id-macchina uguale a 1 e l'altra con un valore di 220 per la velocità e con id_macchina pari a 2. L'id_macchina fungerà da key del topic per distinguere la provenienza dei dati; in questo modo potremo così controllare le due macchine con due istanze ciascuna.

2.3.4 Domain

L'entità del **domain** (dominio) rappresenta uno spazio logico definito che ha lo scopo di **mettere in comunicazione i vari applicativi tra di loro**. All'interno possia-

mo trovare i vari topic che collegano gli applicativi con i loro rispettivi data-types. L'entità domain è caratterizzata dalle seguenti proprietà:

- Ogni domain viene identificato da un id per renderlo **univoco**.
- Ogni entità del DDS può appartenere ad un solo domain.
- Le entità all'interno del domain possono interagire solamente con le altre entità all'interno dello stesso domain.
- Due applicativi DDS per poter comunicare tra di loro hanno bisogno di entrare nello stesso domain.
- Un applicativo può far parte di più domain creando più istanze dell'entità DomainParticipant in ogni domain in cui vuole interagire [25].

2.3.5 DomainParticipant

L'entità del **DomainParticipant** all'interno di un Domain del DDS, viene utilizzata dagli applicativi e rappresenta la prima entità creata da un'applicazione che verrà impiegata per creare DataWriters e DataReaders. Il DomainParticipant ha il **compito di inizializzare le comunicazioni con il Domain** attraverso il processo di discovery che consente alle entità appartenenti allo stesso domain di trovarsi e connettersi automaticamente. Il DomainParticipant è caratterizzato dalle seguenti proprietà:

- Il DomainParticipant come le altre entità, può esistere solo all'interno di un dominio.
- Il DomainParticipant è responsabile della scoperta di altri DomainParticipant all'interno del domain [24].

2.4 Le policy QoS nel dettaglio

A livello logico, le specifiche del DDS definiscono un insieme di policy QoS che le entità del DCPS devono rispettare. Qui di seguito vengono proposte le categorie QoS più rilevanti.

- **Ownership:** questo valore specifica se un topic può essere aggiornato da più (`SHARED` ownership) o da un solo (`EXCLUSIVE` ownership) publisher. Se abbiamo impostato un ownership di tipo `EXCLUSIVE`, per decidere il publisher che ha la possibilità di aggiornare il topic, viene utilizzato l'ownership `STRENGTH`.
- **Liveness:** viene utilizzato per specificare se è necessaria una comunicazione di tipo attivo, rispetto ad una di tipo intermittente.
- **Reliability:** specifica se in una comunicazione tutti i dati trasferiti tra publisher e subscriber devono essere consegnati per intero, oppure se è accettabile anche la perdita di alcuni di essi.
- **Lifespan:** specifica il tempo di scadenza dei dati pubblicati da un publisher.
- **History:** specifica quanti e come i dati devono essere mantenuti in un subscriber dopo averli ricevuti.
- **Durability:** specifica se i dati inviati in precedenza sono disponibili per i nuovi subscriber appena entrati nel domain.

Le configurazioni delle policy vengono trasmesse alle varie entità dal DomainParticipant tra cui i publisher, i subscriber, i topic e i DataWriters. Tuttavia, le policy di un publisher e un subscriber devono essere compatibili. Se così non fosse, la comunicazione tra i due potrebbe essere compromessa [15].

2.5 Il protocollo RTPS

RTPS (Real-Time Publish-Subscribe Protocol) è il wire-protocol nativo utilizzato dal DDS che consente di trasferire i dati provenienti dal layer DDS a quello di trasporto della rete. Solitamente questo viene utilizzato in combinazione con il protocollo best-effort UDP/IP, che risulta ottimale per le comunicazioni di tipo real-time. Tuttavia anche i protocolli connection-oriented incluso il TCP/IP possono essere utilizzati. RTPS include molti vantaggi ideali per il DDS:

- **Connettività plug and play**: le nuove applicazioni possono unirsi o lasciare il domain a proprio piacimento.

- **Tolleranza ai guasti**: non sono presenti singoli punti di guasto perché i dati vengono distribuiti e replicati tra le varie entità DDS che adoperano il Global Data Space.
- **Type-safety**: gli errori di programmazione vengono gestiti in modo tale da non compromettere il funzionamento dei dispositivi remoti.

L'RTPS è suddiviso in quattro moduli differenti: lo structure module, il messages module, behavior module e il discovery module [19].

2.5.1 Structure and behavior module

Lo structure module si occupa di associare le entità DDS alle corrispondenti entità RTPS. Queste ultime sono utilizzate per rappresentare le entità del DDS (come DataReader e DataWriter) all'interno del protocollo RTPS (come l'RTPS Writers e l'RTPS Readers) [19].

Il behavior module, invece, definisce le regole di comunicazione tra due o più entità RTPS, che sono i RTPS Writers e i RTPS Readers (definite dal behavior module), durante una sequenza di messaggi. Esse consentono di mantenere l'interoperabilità tra le varie implementazioni (anche di diversi vendors) del DDS [19].

2.5.2 Messages module

Il messages module si occupa di descrivere il **formato dei messaggi scambiati** tra i RTPS Writers e i RTPS Readers che sono composti da un *HEADER* seguito da dei *sottomessaggi* (Figura 5.1). Nell'header troviamo informazioni relative al protocollo RTPS, cioè la sua versione, il nome del vendor dell'implementazione usata e il mittente. Nel sottomessaggi invece possiamo trovare un header e una serie di metadati. Nell'header del sottomessaggio è presente l'id che ne *identifica il tipo*, eventuali flag e la lunghezza in bytes del sottomessaggio stesso. Le tipologie dei sottomessaggi più importanti, identificate dal suo header, sono:

- **DATA**: in questo sottomessaggio vengono trasferiti dall'RTPS Writers all'RTPS Reader i dati effettivi relativi ad un topic.

- **HEARTBEAT**: viene mandato da un RTPS Writer a un RTPS Reader per comunicare il numero di nuovi (sequence number) aggiornamenti che il Writer ha disponibili.
- **ACKNACK**: utilizzato per comunicare lo stato di un RTPS Reader al corrispondente RTPS Writer, per informarlo riguardo i dati ricevuti e quelli mancanti. Questo sottomessaggio, con la flag `FINAL` impostata, consente di far rimanere il Reader sincronizzato con l'RTPS Writer [19].

2.5.3 Discovery module

Questo modulo garantisce che i nuovi partecipanti DDS (publisher e subscriber) riescano a identificarsi in automatico tra di loro in modo tale da inizializzare una possibile comunicazione. Questo modulo è **responsabile dell'auto-scoperta (Dynamic Discovery)** delle entità del DDS all'interno dello stesso domain. Il Dynamic Discovery utilizza messaggi di tipo *multicast e unicast* per informare gli altri partecipanti di un nuovo dispositivo connesso alla rete, pronto a comunicare con il resto delle entità. Il discovery module è composto da due protocolli chiamati Simple Participant Discovery Protocol (SPDP) e Simple Endpoint Discovery Protocol (SEDP). SPDP ha il compito di scoprire nuovi partecipanti, mentre l'SEPD si occupa di scambiare tra le entità le informazioni di topics, DataWriter e DataReader. In particolare l'SEDP serve per collegare tramite topic i DataReaders ai DataWriters [19].

2.6 DDS Security

Nelle specifiche del DDS **non viene presa in considerazione la sicurezza**, quindi un'implementazione che utilizza il DDS di base può essere esposta a numerosi rischi. Per ovviare a questo problema, OMG ha definito un nuovo standard chiamato **DDS security**. Il DDS security è un'estensione del DDS con l'obiettivo di mitigare una moltitudine di vettori d'attacco come la lettura e la scrittura dei messaggi scambiati tra i partecipanti di un domain DDS. Questa estensione è composta da cinque plugin:

- **Authentication Service Plugin**: serve per effettuare l'autenticazione delle entità DDS. Senza l'autenticazione le entità non possono comunicare tra di loro.

- **Access Control Service Plugin**: ha lo scopo di imporre delle policy alle entità DDS autenticate. Ad esempio limitare la pubblicazione di nuovi dati o la creazione di nuovi topic.
- **Cryptographic Service Plugin**: gestisce tutte le operazioni crittografiche, tra cui la crittografia, la decrittazione e le firme digitali. Ha anche il compito di controllare l'integrità dei messaggi.
- **Logging Service Plugin**: permette di effettuare un audit di tutte le operazioni DDS rilevanti all'interno di un domain.
- **Data Tagging Service Plugin**: fornisce dei metodi per implementare un tag su tutti i dati trasferiti.

Anche se il DDS security riesce a risolvere molti problemi legati alla sicurezza del DDS, non sempre è possibile implementarlo. Spesso la sua configurazione può richiedere molto tempo per essere impostata, soprattutto su sistemi DDS già esistenti e sprovvisti di questa estensione. **Il partecipante più vulnerabile rappresenta la sicurezza complessiva dell'intero sistema**, quindi ogni entità deve essere protetta [15].

Il DDS security può essere usato anche non utilizzando tutti i plugin; gli ultimi due sono facoltativi e vengono raramente usati [12].

2.6.1 Authentication Service Plugin

Senza questo modulo chiunque potrebbe entrare a far parte nel domain del DDS, ponendo un grave rischio alla sicurezza. Per ovviare a questa falla, l'Authentication Service Plugin richiede, a ogni dispositivo che vuole entrare del domain DDS, la **necessità di autenticarsi**. Prima di effettuare l'autenticazione tutti i partecipanti devono avere *un loro certificato e le loro chiavi private*, mentre l'amministratore deve creare un *certificato root* (o Certificate Authority, CA) che deve essere riconosciuto da tutte le entità autorizzate. L'*autenticazione tra le entità avviene in modo reciproco*, in modo tale che ciascun partecipante verifichi l'identità dell'altro tramite un controllo dei certificati.

Processo di autenticazione

L'autenticazione di due entità viene effettuata tramite il protocollo *Diffie-Hellman*, che consente la trasmissione di chiavi in modo sicuro anche in canali di comunicazione non protetti. Le due entità utilizzando Diffie-Hellman otterranno una chiave segreta condivisa da entrambi. Implementando questo protocollo la chiave non verrà mai trasmessa direttamente, **evitando così di essere intercettata** da possibili attaccanti.

Per rafforzare ulteriormente la sicurezza e prevenire attacchi *replay* (riutilizzo di messaggi intercettati) o di impersonificazione, vengono utilizzate le *challenge*. Queste challenge corrispondono a valori casuali che cambiano nel tempo e vengono utilizzati durante il calcolo della firma digitale che si effettua nel protocollo Diffie-Hellman. Dato che che queste challenge cambiano periodicamente, i vecchi messaggi intercettati dagli attaccanti non possono essere più riutilizzati.

La fase di autenticazione si conclude quando viene completato lo scambio delle chiavi. Queste chiavi verranno poi utilizzate da protocolli di crittazione incluso l'*RSA (Rivest-Shamir-Adleman)* per effettuare comunicazioni in modo sicuro. Infatti non sarà possibile per un attaccante spiare o cambiare il contenuto dei messaggi dato che questi sono criptati [30].

2.6.2 Access Control Service Plugin

Questo plugin **gestisce i permessi delle entità** all'interno di un domain DDS. È possibile configurare questi permessi con una granularità molto fine. Dei possibili permessi possono essere: aggiornare un determinato topic da parte di un DataWriter, far entrare una determinata entità all'interno di un domain, eliminare un topic, iscriversi a un topic, creare un topic con specifici DataReaders e DataWriters e entrare o uscire da determinati domains.

L'Access Control Service Plugin per funzionare ha bisogno di due files in formato XML che devono essere entrambi firmati da un CA: il *governance document* e il *permissions document*. Il governance document rimane uguale per tutti i dispositivi all'interno del domain DDS e si occupa di gestire permessi generali a livello di domain. Il permissions document, invece, è unico per ogni dispositivo e si occupa di gestire i

permessi del singolo partecipante. Questi vengono ricevuti dai partecipanti durante la fase di autenticazione e devono rimanere sempre disponibili [12].

Processo di controllo permessi

Un esempio di processo di controllo permessi avviene quando un DataWriter richiede l'autorizzazione per creare un nuovo topic. Le **altre entità hanno il compito di verificare se l'operazione richiesta dal DataWriter viene consentita dai files di permessi** a loro disposizione. Per effettuare questa operazione il partecipante, che richiede l'autorizzazione , deve mandare il proprio permission document, il topic che intende creare e i propri metadati. Successivamente un altro partecipante, che ha il compito di autorizzare il DataWriter, riceverà il messaggio ed effettuerà le seguenti verifiche:

1. Verifica che la firma digitale del permesso ricevuto sia valida.
2. I metadati forniti devono corrispondere a quelli del del permesso ricevuto.

Il secondo controllo ha lo scopo di verificare che il partecipante, che vuole creare il nuovo topic sia effettivamente quello indicato nel permesso ricevuto in precedenza; eliminando in questo modo gli attacchi di impersonificazione [30].

```
...
    <permissions>
        <grant name="ShapesPermission">
            <subject_name>CN=DDS Shapes Demo</subject_name>
            <validity>
                <not_before>2013-10-26T00:00:00</not_before>
                <not_after>2018-10-26T22:45:30</not_after>
            </validity>
            <allow_rule>
                <domains>
                    <id>0</id>
                </domains>
            </allow_rule>
```

```
<deny_rule>
    <domains>
        <id>0</id>
    </domains>
    <publish>
        <topics>
            <topic>Circle1</topic>
        </topics>
    </publish>
```

...

Codice 2.2: Estratto di permissions document, tratto da documento di riferimento del DDS Security versione 1.1 [18].

Il permission document mostrato nel Codice 2.2 ci indica che ha una validità che parte dal 26 ottobre 2013 fino al 2018 alle 22:45:30. Successivamente viene creata una regola all'interno del dominio con id pari a zero, dove è negato al suo interno ogni possibile pubblicazione sul topic `CIRCLE 1`.

2.7 Implementazioni DDS

Il Data Distribution Service gestito (DDS) dall'Object Management Group (OMG) può essere definito un **open standard**. Questi standard di OMG sono disponibili al pubblico e servono per mantenere una certa *consistenza, portabilità e interoperabilità* tra le varie implementazioni del DDS. Bisogna ricordare che un open standard non equivale però a un software open source, quindi, lo sviluppo dell'applicativo è gestito dalle software house che possono decidere come gestire il codice sorgente. Sul mercato sono comunque presenti soluzioni open source, ma spesso queste presentano degli svantaggi, tra cui la **mancanza di supporto per alcuni linguaggi di programmazione** per interagire con le API del middleware [28].

Nella Tabella 2.1 vengono confrontate le principali implementazioni adoperate oggigiorno. Come si può notare dai dati, le soluzioni proprietarie come RTI Connext DDS dispongono di un più ampio supporto per diversi linguaggi di programmazioni rispetto alle alternative open source, tra cui eProsima FastDDS.

Implementazione	Sviluppatore	Tipo di licenza	Linguaggi supportati	Anno creazione
Fast DDS [4]	eProsima	Apache License 2.0	C++, Python	2014
Connext DDS [11]	Real-Time Innovations	Closed source	C, C#, C++, Python, Java	2005
OpenDDS [2]	Object Computing	Open source (custom)	C++, Java	2005
Cyclone DDS [7]	Eclipse Foundation	Open source (custom)	C, C++, Python	2011
CoreDX [29]	Twin Oaks Computing	Closed source	C, C#, C++, Java	2009

Tabella 2.1: Esempi di implementazioni DDS.

Capitolo 3

Vulnerabilità standard DDS

In questo capitolo ci occuperemo di analizzare e comprendere le vulnerabilità del middleware DDS standard OMG (Object Management Group) [33]. In particolare verrà analizzato il vettore d'attacco, il protocollo utilizzato, il bersaglio dell'attacco e infine verrà proposta una soluzione applicabile per mitigare i possibili attacchi. Queste falle in molti casi, possono essere sfruttate quando un partecipante del domain DDS è sotto il controllo di un attaccante o quando è possibile modificare i file di configurazione delle policy QoS del domain.

Queste problematiche di sicurezza riguardano la versione del DDS 1.4 con le specifiche dello standard OMG.

Nella Tabella 3.1 viene mostrato un riassunto di tutte le vulnerabilità presentate in questo capitolo. Come si può notare dai dati la soluzione più efficace per mitigare questi attacchi è utilizzare l'estensione DDS security che permette di crittografare le diverse comunicazioni scambiate tra le varie entità, rendendo più difficile per un attore malevolo analizzare i contenuti dei pacchetti.

Tipo di attacco	Vettore attacco	Protoc./ Estens.	Bersaglio nella rete	Software	Soluzione
Discovery devices [32]	Verbose nature of RTPS	RTPS-SDPD	Tutti i partecipanti	Sniffer Python	DDS security/ WireGuard
DDos [32]	Heartbeat sequence number	RTPS	DataReader	Sniffer Python	DDS security: Auth Control
DDoS [30]	Authentication challenge	DDS security 1.1 Discovery protoc.	Tutti i partecipanti	Proverif	Scadenza richieste di autenticazione
QoS policy [16]	Policy: Ownership	RTPS	DataReader	RTI shapes	DDS security: Access Control
QoS policy [16]	Policy: Lifespan	RTPS	DataReader	RTI shapes	Controllo per Lifespan scartati

Tabella 3.1: Riassunto delle vulnerabilità del DDS analizzate.

3.1 Enumeration sniff

Prendendo in considerazione, il protocollo RTPS, descritto nella Sezione 2.5.3, e il suo modulo discovery, possiamo notare che i messaggi di default sono molto *verbose*, scambiando le **informazioni in chiaro** durante le comunicazioni tra i vari partecipanti [32]. Il modulo discovery del protocollo RTPS a sua volta si suddivide in altri 2 protocolli, che sono necessari per le specifiche DDS:

- Simple Participant Discovery Protocol (SPDP).
- Simple Endpoint Discovery Protocol (SEDP).

3.1.1 Dettagli attacco e coclusioni

Per questo attacco ci focalizzeremo in particolare sull'SPDP che serve ad individuare la presenza dei partecipanti al resto delle entità nel domain. In particolar modo il funzionamento si basa su messaggi di tipo *multicast* che vengono mandati a tutti i partecipanti attivi. [19].

L'attaccante *sniffando* questi messaggi (all'interno di un domain DDS) di tipo multicast RTPS-SPDP e utilizzando anche un semplice script Python, potrà infatti vedere il loro contenuto in maniera chiara.

All'interno di un pacchetto di questo tipo possiamo trovare: l'indirizzo IP dell'host, il prefisso GUID dell'RTPS, la versione dell'RTPS, l'ID del venditore, i metadati riguardanti la sincronizzazione ed infine il contenuto dei sottomessaggi [32].

Fare una ricognizione della rete DDS senza effettuare veri e propri attacchi di tipo attivo può essere molto utile per un attaccante che ha il compito di penetrare in modo attivo una rete DDS. In molti casi tutto quello che deve fare l'attaccante è osservare i messaggi che vengono scambiati all'interno del network. Successivamente quando si ottengono **informazioni a sufficienza sarà più facile per l'attaccante trovare altre vulnerabilità** [32].

Di solito questo tipo di attacco è **difficile da identificare** e può essere effettuato senza lasciare tracce di nessun tipo, dato che l'attaccante non manda pacchetti. Una soluzione potrebbe essere usare l'estensione DDS security o eseguire la connessione tra i nodi tramite un tunnel con WireGuard per crittare le comunicazioni.

3.2 Blocco DataReader tramite sequence number

Il vettore di attacco si trova nel *messages module* del protocollo RTPS descritto nella Sezione 2.5.2. Questo modulo si occupa di scambiare messaggi tra i DataReader e i DataWriter all'interno un domain DDS.

Per effettuare questi scambi di messaggi vengono utilizzati dei sottomessaggi, in particolare l'HEARTBEAT e l'ACKNACK. L'HEARTBEAT contiene al suo interno il *sequence number* che tiene traccia del numero di aggiornamenti di un topic da parte di DataWriter, mentre l'ACKNACK inviato da un DataReader serve per confermare al DataWriter la ricezione di nuovi dati riguardo un topic. Quando il DataReader riceve il sequence number all'interno di un HEARTBEAT può identificare **se ci sono o no dei pacchetti mancanti e in caso segnalarli** al DataWriter [32]. Inoltre se il parametro `FINAL` è attivo in un sottomessaggio HEARTBEAT, il DataReader deve sempre rispondere al DataWriter con ACKNACK dopo aver ricevuto nuovi aggiornamenti. Il DataWriter, nel frattempo, rimarrà in attesa del sottomessaggio ACKNACK prima di inviare nuovi aggiornamenti al DataReader. Questo sistema aiuta il DataReader a rimanere sempre sincronizzato con il DataWriter.

I controlli del sequence number all'interno dell'HEARTBEAT **non sono però sufficienti** per mitigare questo attacco:

- Un primo controllo viene effettuato per verificare che non ci siano valori negativi;
- Un altro controllo serve a determinare se l'ultimo sequence number appena ricevuto abbia un valore minore rispetto a quello ricevuto in precedenza [32];

3.2.1 Dettagli attacco e conclusioni

Per sfruttare questa vulnerabilità l'attaccante deve utilizzare qualche strumento per sniffare la comunicazione tra il DataReader e il DataWriter, intercettando così i sottomessaggi HEARTBEAT. Dopo aver catturato un HEARTBEAT diretto verso un DataReader e modificato il suo sequence number **assegnandogli un valore molto alto**, l'attaccante lo rinvierà al suo destinatario originario. Una volta ricevuto il sottomessaggio il DataReader si metterà quindi in attesa di un HEARTBEAT con un sequence number **superiore a quello appena ricevuto**. Di conseguenza il DataReader non elaborerà più i messaggi legittimi mandati dal DataWriter, dato che questi hanno sequence number più piccoli. Solo un messaggio HEARTBEAT con un sequence number maggiore a quello del DataReader farà ripristinare la sua esecuzione [32].

Di solito questo tipo di attacco è difficile da identificare. Un messaggio HEARTBEAT riguarda un solo topic, quindi il resto delle comunicazioni che avvengono su topic differenti o anche sullo stesso topic, ma con un DataReader diverso, non subiranno cambiamenti. Questa falla di sicurezza può essere mitigata utilizzando l'estensione DDS security in modo tale da crittografare i messaggi e rendere impossibile per l'attaccante effettuare modifiche al sequence number.

3.3 DDoS sfruttando estensione DDS security

Questo vettore di attacco si trova nell'estensione del DDS chiamata DDS security, descritto nella Sezione 2.6; in particolare la versione utilizzata è la versione 1.1. Il DDS security si occupa di stabilire una connessione sicura tra i vari dispositivi della rete, impiegando dei plugin per effettuare: autenticazione, controllo accesso, crittografia, login e data logging [18].

Ogni partecipante del domain DDS deve essere autenticato dalle altre entità appartenenti allo stesso domain. Successivamente due entità, per iniziare una co-

municazione tra di loro, devono prima scambiarsi le chiavi private in modo sicuro tramite protocollo *Diffie-Hellman* in modo tale da poter crittare i successivi messaggi. Durante l'uso di Diffie-Hellman vengono adoperate le *challenge* (Sezione 2.6.1), che corrispondono a valori che variano nel tempo, inseriti durante il calcolo della firma digitale richiesta dal protocollo. Queste challenge vengono utilizzate per rendere le varie sessioni di autenticazione uniche evitando così attacchi di tipo *replay* [30].

3.3.1 Dettagli attacco e conclusioni

L'attacco DDoS avviene durante la fase di autenticazione del DDS security 1.1, in particolare quando un nuovo dispositivo tenta di collegarsi alla rete e manda una richiesta di autenticazione all'entità con cui vuole aprire una comunicazione. La richiesta del partecipante viene intercettata dall'attaccante che modifica i valori della challenge crittografica all'interno del pacchetto. **Modificando ripetutamente questi valori, l'attaccante inizia a inviare molteplici richieste crittografiche alla sua vittima.** Il partecipante così comincerà a calcolare le firme digitali per effettuare l'autenticazione, consumando tutte le sue risorse. Dato che, la vittima è probabilmente un dispositivo IoT (Internet of Things) che non possiede di una potenza di calcolo molto elevata, si ritroverà occupata per tutto il tempo necessario a computare diverse firme digitali ricevute dall'attaccante, bloccando così il suo funzionamento [30].

Questo attacco è stato scoperto con *Proverif*, un tool che viene usato per individuare vulnerabilità nei protocolli crittografici. È stato utilizzato in molti studi, ad esempio nell'analisi della posta elettronica certificata e nell'analisi del TLS 1.3 [1].

Una raccomandazione per mitigare questo attacco è quello di cambiare delle policy QoS impostando un tempo limite massimo per effettuare l'autenticazione. Queste policy possono fare in modo che i partecipanti non si trovino sopraffatti dalle troppe richieste di autenticazione. Un allarme potrebbe essere utile per identificare possibili tentativi DDoS di questo tipo, allertando così un amministratore [30].

3.4 Modifica maligna OWNERSHIP_STRENGTH

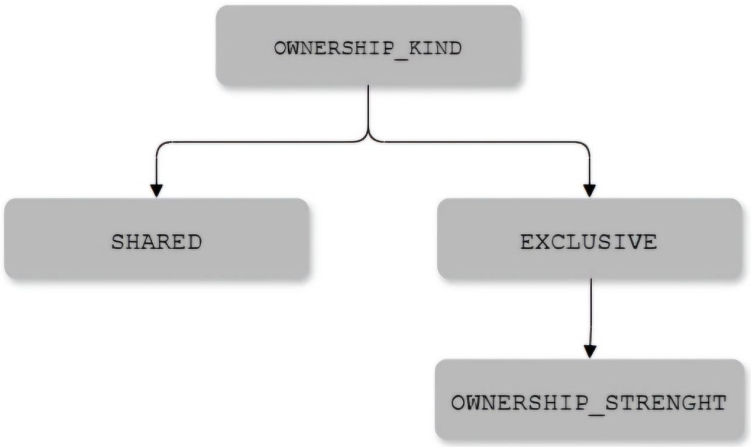

Figura 3.1: Illustrazione policy QoS del DDS

Questo attacco è realizzabile solo se certe policy QoS vengono modificate durante l'esecuzione della rete, specialmente il parametro OWNERSHIP_KIND che gestisce **quanti DataWriter possono scrivere per un determinato topic.** Una descrizione accurata di altre policy QoS viene effettuata nella Sezione 2.4. Questo parametro può essere impostato in due modi diversi:

- **SHARED:** in questo modo più DataWriter possono aggiornare le informazioni di un topic.
- **EXCLUSIVE:** solo un DataWriter può aggiornare le informazioni di un topic. Il DataWriter che ha il permesso di scrittura per il topic è quello che dispone di una OWNERSHIP_STRENGTH con valore più alto.

Nella Figura 3.4 viene proposto uno schema riassuntivo delle policy QoS relative a questo attacco. In una rete dove si adopera un OWNERSHIP_KIND di tipo EXCLUSIVE si consente all'attaccante di utilizzare l'OWNERSHIP_STRENGTH a suo favore. Infatti è possibile far ricevere informazioni a un DataWriter in maniera errata, dato che quest'ultimo non riceverà più dati da una fonte affidabile [16].

3.4.1 Dettagli attacco e conclusioni

L'attaccante, con un DataWriter in suo possesso all'interno di una rete DDS, può sfruttare il fatto che il topic preso di mira può essere aggiornato solo dal DataWriter con l'OWNERSHIP_STRENGTH più alta. Per effettuare questo attacco, tutto quello che serve, è essere a conoscenza del topic che si vuole modificare, delle policy QoS in uso e del valore dell'OWNERSHIP_STRENGTH. L'ultimo passo è quello di **impostare la policy QoS nel DataWriter dell'attaccante con una OWNERSHIP_STRENGTH superiore a quella impiegata dal DataWriter originario.** Ora i DataReader che sono iscritti al topic bersaglio ricevono dati dal DataWriter dell'attaccante [16].

Questa vulnerabilità è stata analizzata con l'ausilio di *RTI Shapes Demo*, un software che emula una rete DDS corrispondente alle specifiche dello standard OMG, sviluppato da Real-Time Innovations (RTI).

L'OWNERSHIP_KIND di tipo EXCLUSIVE è impiegata in contesti dove le informazioni ricevute dal DataReader devono essere accurate dato che un singolo DataWriter (in molti casi si tratta di un sensore) può mandare nuovi aggiornamenti del topic. Se l'attaccante, dovesse riuscire a modificare i valori del topic, potrebbe causare molti danni, specialmente se il DataWriter dell'attaccante riuscirà a mandare degli aggiornamenti al topic senza essere scoperto [16].

Una soluzione utile per mitigare questo attacco potrebbe essere l'applicazione dell'estensione DDS security. Utilizzando i plugin relativi al controllo accesso, come illustrato nella Sezione 2.6.2, permetterà di evitare che un attore malevolo possa modificare le policy QoS inclusa l'OWNERSHIP_STRENGTH.

3.5 Modifica maligna di LIFESPAN QoS

Un'altra policy QoS che può essere usata come vettore di attacco è quella che regola il parametro LIFESPAN. Questo parametro corrisponde al **tempo limite massimo di validità per la lettura di pacchetti** da parte di un DataReader. Per determinare se un pacchetto di un topic è scaduto viene utilizzato il *timestamp* di creazione aggiungendo il valore del LIFESPAN impostato; se questo risultato chiamato *expiration time* risulterà superiore all'orario durante la ricezione del DataReader, allora il pac-

chetto ricevuto sarà ancora valido. Per funzionare, gli orologi interni del DataWriter e del DataReader devono essere sincronizzati tra di loro.

Un'altra policy da considerare riguarda l'affidabilità (RELIABILITY), che può essere impostata in due modi, dei dati riguardanti un topic :

- RELIABLE: questa impostazione costringe il DataReader a farsi ritrasmettere dal DataWriter i pacchetti mancanti o ricevuti in maniera errata. In questo modo le informazioni del DataReader saranno sempre corrette anche se non sempre aggiornate in tempo reale.
- BEST-EFFORT: l'impostazione predefinita non consente il recupero dei pacchetti mancanti o corrotti del DataReader, quindi, quest'ultimo potrebbe anche perdere dei pacchetti che gli sono stati inviati [17].

Se il LIFESPAN dei pacchetti, contenenti i dati del topic, viene impostato con valori molto piccoli, si verificheranno problemi di comunicazione tra DataWriter e DataReader, dato che non sarà possibile effettuare la lettura di certi pacchetti inviati. Impostando il valore RELIABLE tra le policy QoS con RELIABILITY sarà possibile mitigare solo parzialmente questa vulnerabilità [16].

3.5.1 Dettagli attacco e conclusioni

Avendo sotto controllo i parametri LIFESPAN e RELIABLE, l'attaccante modificherà le policy dei DataWriter in modo tale da **avere il valore LIFESPAN molto piccolo.** Così facendo, i pacchetti spediti dal publisher arriveranno già scaduti al DataReader, rendendoli inutilizzabili. In certi casi il pacchetto che deve essere inviato viene distrutto dallo stesso DataWriter all'interno della sua coda prima dell'invio. Questa falla di sicrezza è stato verificata configurando il valore di LIFESPAN $<$ 80ms dove si è visto che nessun pacchetto raggiunge il DataReader. Se si aumenta il valore tra gli 80ms e i 100ms già si può notare che dei pacchetti vengono letti con successo dal DataReader, mentre altri vengono eliminati prima della lettura. Infine impostando un valore LIFESPAN $>=$ 120ms si può notare che la comunicazione tra publisher e subscriber avverrà senza nessun problema.

Un dettaglio da aggiungere è che se si imposta la policy dell'affidabilità (RELIABILITY) con la flag RELIABLE, i millisecondi necessari del LIFESPAN per compromettere le comunicazioni tra DataReader e DataWriter devono essere *moltiplicati*

per un fattore di 0.01. Quindi, ad esempio se si ottiene un completo annullamento delle comunicazioni con un LIFESPAN $<$ 80ms utilizzando la RELIABILITY di tipo BEST-EFFORT, per ottenere lo stesso risultato con RELIABILITY di tipo RELIABLE dobbiamo impostare un LIFESPAN $<$ 0.8ms [16].

Anche questo test è stato dimostrato con RTI Shapes Demo che implementa una soluzione DDS di RTI corrispondente alle specifiche dello standard OMG. Inizialmente molte reti DDS hanno impostato la RELIABILITY di tipo BEST-EFFORT che è l'impostazione predefinita, in modo tale da rendere possibili comunicazioni di tipo real-time. Quindi nella maggior parte dei casi l'attore malevolo non si deve preoccupare di modificare questo parametro.

Una possibile soluzione consiste nell'implementare qualche tipo di controllo in modo tale da avvertire un operatore umano se molti pacchetti vengono scartati perché arrivati con un LIFESPAN scaduto. Questo controllo potrebbe essere anche utile, nel caso in cui il DataWriter e il DataReader si trovassero distanti fisicamente tra di loro, per verificare la qualità del collegamento [16].

Capitolo 4

Vulnerabilità implementazioni DDS

In questo capitolo verranno mostrati dei casi studio di tre tipologie di attacco che vengono effettuati tramite delle implementazioni del DDS dei vari vendors.

L'obiettivo di questi casi studio è di trovare delle vulnerabilità in Robot Operating System (ROS) 2 che utilizza come middleware il DDS e come wire-protocol l'RTPS (Real-Time Publish-Subscribe Protocol). Infatti saranno proprio i pacchetti RTPS a contenere l'*exploit* vero e proprio. Prima di mandare questi pacchetti è stato necessario creare una libreria Python così da poter automatizzare il loro processo di creazione. Questo strumento, successivamente è stato anche unito al progetto Scapy di Python, nella Figura 4.1 è presente un esempio di un pacchetto RTPS costruito con questa libreria.

```
rtps_package = RTPS(
    protocolVersion=ProtocolVersionPacket(major=2, minor=4),
    vendorId=VendorIdPacket(vendor_id=b"\x01\x03"),
    guidPrefix=GUIDPrefixPacket(
        hostId=16974402, appId=2886795266, instanceId=1172693757
    ),
    magic=b"RTPS",
)
```

Figura 4.1: Un pacchetto RTPS costruito con Scapy [14].

4.1 Ricognizione DDS

Le implementazioni DDS, per rispettare lo standard OMG, devono seguire delle **regole di interoperabilità** per far si che i vari applicativi DDS siano compatibili tra di loro. Questo ha portato alla creazione di un sistema molto verbose per effettuare discovery di altri partecipanti all'interno di un network DDS.

La natura molto verbose del processo di autoscoperta fa si che inviando un singolo pacchetto RTPS vuoto a un'entità di una rete DDS all'interno di un dominio, quest'ultima risponderà con un messaggio discovery. La risposta, mostrata in Figura 4.2 ci permetterà di confermare se quel determinato partecipante è attivo.

```
alias@MacBook-Pro-de-alias:~/robot_hacking_manual/1_case_studies/2_ros2$ python3 exploits/footprin
t.py 2> /dev/null
IP / UDP 192.168.1.88:58465 > 192.168.1.85:6666 / RTPS / RTPSMessage
```

Figura 4.2: Esempio di risposta discovery di un'entità DDS [14].

La vulnerabilità è stata individuata adoperando CycloneDDS, ma anche **altre implementazioni presentano lo stesso problema.** Questo accade perché non è possibile mitigare la vulnerabilità senza violare lo standard OMG. L'unica soluzione applicabile consisterebbe nel disabilitare in parte il meccanismo di autoscoperta, ma cosi facendo l'interoperabilità tra i vari vendors DDS andrebbe persa.

Per questo motivo, dato che i vendors preferiscono mantenere questa interoperabilità non esiste una soluzione vera e propria [14].

4.2 Attacco riflesso DDS

Un attacco riflesso consiste nel dirottare il traffico di rete verso un dispositivo vittima, manipolando e in certi casi amplificando anche un flusso di dati.

```
PID_METATRAFFIC_UNICAST_LOCATOR(
    parameterId=50,
    parameterLength=24,
    locator=LocatorPacket(
     locatorKind=16777216, port=47324, address="8.8.8.8"
    ),
),
PID_METATRAFFIC_MULTICAST_LOCATOR(
    parameterId=51,
    parameterLength=24,
    locator=LocatorPacket(
        locatorKind=16777216,
        port=17902,
        address="239.255.0.1",
    ),
),
```

Figura 4.3: Parte di un pacchetto RTPS costruito con Scapy che abilita un attacco riflesso [14].

Il vettore di questo attacco si trova all'interno di più parametri di un sottomessaggio di tipo DATA incluso il parametro `PID_METATRAFFIC_MULTICAST_LOCATOR` che si occupa di specificare gli indirizzi multicast che verranno adoperati successivamente dalle entità per comunicazioni di metatraffico. **Il DDS non prevede filtri per questo specifico valore** e quindi un attaccante può specificare a un partecipante di utilizzare un indirizzo IP multicast per il metatraffico sotto il suo controllo. Ricevuto questo traffico l'attaccante può ricevere informazioni riguardo la rete DDS e rispondere al partecipante con un alto numero di comunicazioni in modo tale da sovraccaricarlo (DDoS).

Inoltre, il valore di `PID_METATRAFFIC_MULTICAST_LOCATOR` può essere utilizzato da un attore malevolo per reindirizzare il traffico di molteplici entità verso un unico partecipante che si ritoverà quindi a ricevere un numero elevato di pacchetti indesiderati rallentandone la sua esecuzione.

Un altro parametro simile è il `PID_METATRAFFIC_UNICAST_LOCATOR` che si occupa di specificare un indirizzo IP di tipo unicast per la comunicazioni di metatraffico. Anche in questo caso non è possibile applicare filtri per limitare la scelta di valori per gli indirizzi IP.

Come mostrato nella Figura 4.3 creando un pacchetto con Scapy e modificando i valori dei parametri vulnerabili sarà possibile rispedire le comunicazioni di una identità verso un qualsiasi indirizzo, come l'IP del DNS 8.8.8.8 (Figura 4.4).

```
  127 26.706741635   172.17.0.1 → 172.17.0.2   RTPS 302 DATA(p)
  128 26.707092448   172.17.0.2 → 8.8.8.8      RTPS 306 INFO_TS
, DATA(p)
  129 26.707202239   172.17.0.2 → 239.255.0.1  RTPS 306 INFO_TS
, DATA(p)
  130 26.707478601   172.17.0.2 → 8.8.8.8      RTPS 110 INFO_DS
T, HEARTBEAT
  131 26.707711521   172.17.0.2 → 8.8.8.8      RTPS 110 INFO_DS
T, HEARTBEAT
  132 26.707866988   172.17.0.2 → 8.8.8.8      RTPS 110 INFO_DS
T, HEARTBEAT
```

Figura 4.4: Esempio di reindirizzamento dei messaggi verso il server DNS 8.8.8.8 [14].

Questo problema di sicurezza è presente in tutte le implementazioni, come viene mostrato dai CVE della Figura 4.5, dei vendor dato che **non può essere risolto senza violare lo standard OMG**. Questa opzione però, non è considerata dai vari vendors perché interromperebbe l'interoperabilità tra le diverse implementazioni. Per rimanere compatibili con lo standard OMG i vendor hanno implementato un sistema di controllo che limita il numero di pacchetti scambiati quando viene superata una soglia limite prefissata. Questo sistema non è perfetto, ma in molti casi ha ridotto significativamente l'efficacia di questo attacco [14].

CVE ID	Description	Scope	CVSS	Notes
CVE-2021-38487	RTI Connext DDS Professional, Connext DDS Secure Versions 4.2x to 6.1.0, and Connext DDS Micro Versions 3.0.0 and later are vulnerable when an attacker sends a specially crafted packet to flood victims' devices with unwanted traffic, which may result in a denial-of-service condition.	ConnextDDS, ROS 2*	8.6	Mitigation patch in >= 6.1.0
CVE-2021-38429	OCI OpenDDS versions prior to 3.18.1 are vulnerable when an attacker sends a specially crafted packet to flood victims' devices with unwanted traffic, which may result in a denial-of-service condition.	OpenDDS, ROS 2*	8.6	Mitigation patch in >= 3.18.1
CVE-2021-38425	eProsima Fast-DDS versions prior to 2.4.0 (#2269) are susceptible to exploitation when an attacker sends a specially crafted packet to flood a target device with unwanted traffic, which may result in a denial-of-service condition.	eProsima Fast-DDS, ROS 2*	8.6	WIP mitigation in master

Figura 4.5: Varie CVE per attacco riflesso [14].

4.3 Crash di un partecipante

La seguente vulnerabilità è stata scoperta utilizzando un software in grado di eseguire fuzz tests, incluso DDSFuzz (mostrato nella Sezione 5.3).

I test hanno identificato una vulnerabilità in un **mancato controllo di lunghezza del valore del parametro** `PID_BUILTIN_ENDPOINT_QOS` che si trova all'interno di un sottomessaggio utilizzato durante la fase di autoscoperta del DDS [14].

Questo parametro viene utilizzato nella fase di discovery tra i partecipanti della rete per trasmettere alla nuova entità, che vuole unirsi alla rete, le policy QoS attive. La nuova entità, confrontate le proprie impostazioni QoS con quelle appena ricevute può determinare con quali partecipanti può iniziare le comunicazioni [19].

Data la mancanza di un controllo per la lunghezza del valore del parametro è possibile costruire un pacchetto RTPS, come mostrato in Figura 4.6 in modo tale da mandare in crash il partecipante che lo riceve. **Osserviamo che il valore `parameterLength` è pari a zero**, mentre il parametro `parameterData` contiene dei bytes (nel nostro caso nulli). Questa discrepanza causa un crash durante la lettura di `parameterData` dato che l'indice di lettura venendo impostato (`parameterLength`) a zero denota che non dovrebbe essere presente alcun dato al suo interno. Di con-

seguenza, viene eseguita una porzione di codice al di fuori dei limiti della memoria causando un errore di segmentazione.

Inoltre, non viene esclusa la possibilità che questa vulnerabilità possa essere sfruttata per eseguire del codice arbitrario malevolo.

```
PID_BUILTIN_ENDPOINT_QOS(
                    parameterId=119,
                    parameterLength=0,
                    parameterData=b"\x00\x00\x00\x00",
                ),
```

Figura 4.6: Pacchetto RTPS con il valore di parameterLength incorretto [14].

Questa vulnerabilità è stata individuata solamente nell'implementazione Fast DDS. Successivamente è stata registrata una segnalazione CVE, mostrata in Figura 4.7, che ha portato al rilascio di una patch per correggere il problema [14].

CVE ID	Description	Scope	CVSS	Notes
CVE-2021-38445	OCI OpenDDS versions prior to 3.18.1 do not handle a length parameter consistent with the actual length of the associated data, which may allow an attacker to remotely execute arbitrary code.	OpenDDS, ROS 2*	7.0	Failed assertion >= 3.18.1

Figura 4.7: CVE per mancato controllo di lunghezza di `PID_BUILTIN_ENDPOINT_QOS` [14].

Capitolo 5

Tools di analisi

In questo capitolo verranno esposti vari tools che hanno lo scopo di analizzare il funzionamento del DDS. Questi tools, a volte, vengono anche utilizzati con altri protocolli o altri sistemi di rete, mentre ci sono altri tool inclusi DDSFuzz che sono compatibili solamente con il middleware DDS. Molti strumenti sono stati impiegati anche per esaminare il software ROS (Robotic Operation System) che utilizza come sistema di comunicazione un'implementazione del DDS (di solito Fast DDS [4]) per effettuare lo scambio di comunicazione tra i vari dispositivi connessi.

5.1 WireShark

WireShark è un tool di tipo *packet-sniffing* che ci consente di interagire e visualizzare il contenuto dei pacchetti scambiati da diversi protocolli di rete. Esso ha ottenuto un grande successo grazie alla sua interfaccia user-friendly e al suo codice sorgente open-source con licenza di utilizzo libero. I pacchetti possono essere analizzati sia in real-time che in modalità statica utilizzando un file di tipo *PCAP* (packet capture).

Questo tool è anche compatibile con il protocollo RTPS che viene adoperato dalle entità del DDS per comunicare tra di loro come descritto nella Sezione 2.5.2. Nella Figura 5.1 possiamo trovare un esempio di un pacchetto RTPS suddiviso dall'interfaccia di WireShark.

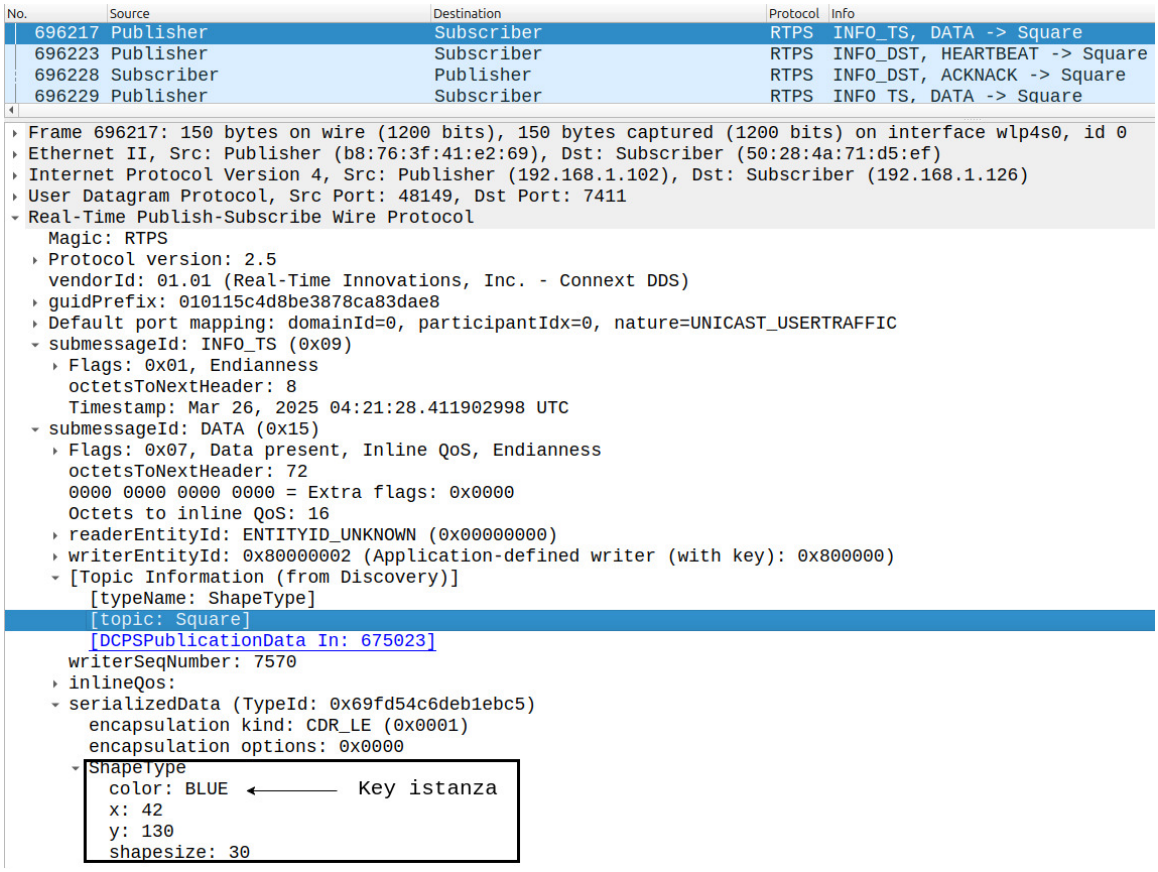

Figura 5.1: Esempio di cattura di un pacchetto RTPS con topic Square tramite WireShark.

5.1.1 Pacchetti RTPS

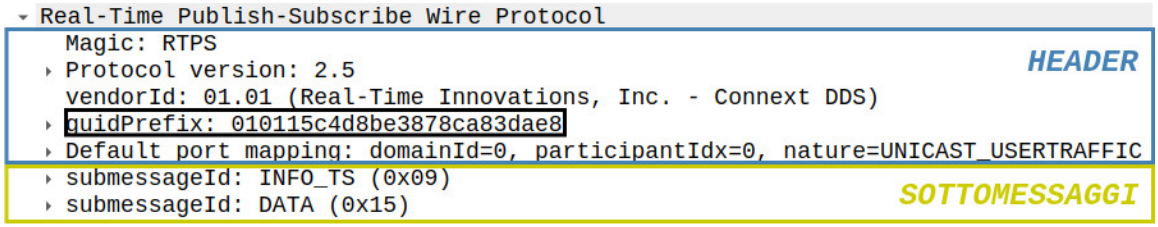

Figura 5.2: Pacchetto RTPS contente l'HEADER (in blu) e due sottomessaggi (in giallo).

Nella Figura 5.2, con l'ausilio di WireShark, controllando il contenuto di pacchetto RTPS possiamo osservare la presenza di un HEADER e due diversi sottomessaggi.

- All'interno dell'HEADER sono presenti diversi metadata, tra cui il guidPrefix, un identificativo univoco che rappresenta il partecipante che ha inviato il pacchetto.
- Il sottomessaggio INFO_TS serve a fornire un timestamp ai destinatari del pacchetto, quindi non contiene un guidPrefix per il suo destinatario al suo interno. Al contrario, un sottomessaggio **INFO_DST include il guidPrefix**, in quanto il messaggio è diretto ad un'unica entità specifica, come avviene, ad esempio, per un sottomessaggio ACKNACK destinato a un DataWriter [10].
- Il secondo sottomessaggio, che nel nostro caso è di tipo DATA, serve a identificare il tipo di pacchetto.

Nella Figura 5.3 viene evidenziato con l'ausilio dell'interfaccia di WireShark la differenza tra NFO_TS e INFO_DST.

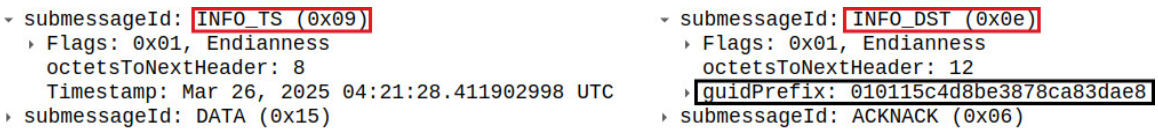

Figura 5.3: A sinistra sono mostrati i sottomessaggi DATA e INFO_TS, mentre a destra sono presenti i sottomessaggi ACKNACK e INFO_DST che includono il guidPrefix del DataWriter, lo stesso riportato in Figura 5.2.

5.1.2 Un'alternativa a WireShark: eProsima DDS Record & Replay

eProsima DDS Record & Replay è un software open-source con licenza Apache 2.0 sviluppato da eProsima (gli stessi creatori di Fast DDS) che ci consente di catturare i contenuti del traffico di un network DDS all'interno di un file di tipo MCAP, a differenza del formato PCAP utilizzato da WireShark [3].

MCAP è un database open-source che serve a salvare il contenuto di più istanze DDS (quindi provenienti da differenti flussi di dati, Sezione 2.3.3), associandolo ad

un timestamp in modo tale da creare dei **dati serializzati** [9]. Con questi dati serializzati è possibile vedere con più facilità il loro contenuto in modo tale da poter analizzare al meglio il comportamento del traffico della rete, controllando nel dettaglio uno o più partecipanti.

MCAP è stato creato per *loggare* dati provenienti da applicazioni che utilizzano un modello publish/subscribe, tra cui il DDS e ROS. Quindi diversi applicativi possono interpretare questo formato, inclusa la piattaforma foxglove [8], che consente una visualizzazione diretta dal browser e da eProsima DDS Record & Replay.

5.2 eProsima Fast DDS Spy

eProsima Fast DDS Spy è un tool, sviluppato da eProsima, open-source con licenza Apache 2.0 accessibile tramite *interfaccia a riga di comando* (CLI) che serve a monitorare i partecipanti e i loro messaggi scambiati all'interno di una rete DDS, mostrato in Figura 5.4 [5].

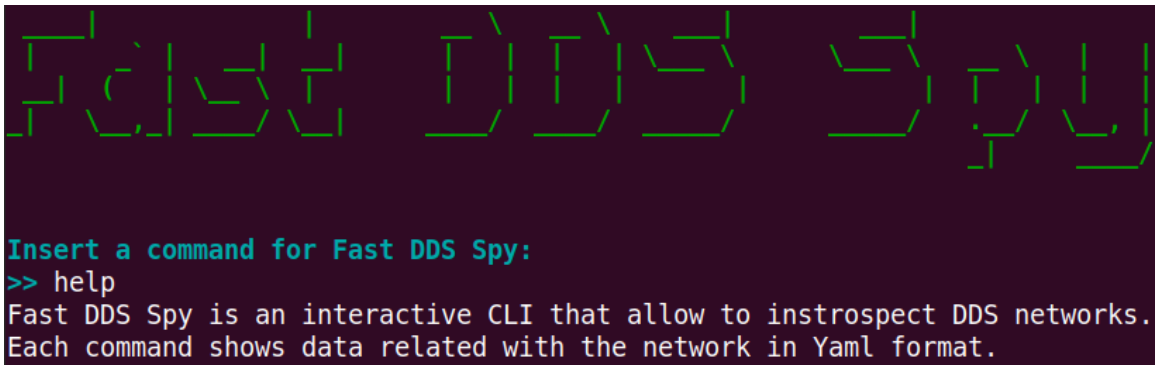

Figura 5.4: Screenshot di Fast DDS Spy appena avviato.

Facile da utilizzare, può essere molto efficace per comprendere al meglio la topologia di una rete DDS. Un attaccante può usare questo tool per capire quali siano le varie entità del network, inclusi il DataWriter, il DataReader i DomainParticipant e i topic. Per ciascuna di queste entità è anche possibile **ottenere il loro GUID (guidPrefix)**, una stringa univoca che ci permette di identificare un partecipante. Il GUID ottenuto può essere impiegato per filtrare il traffico real-time scambiato all'interno della rete.

Un'altra importante funzionalità particolarmente utile per effettuare una ricognizione del network è offerta dai comandi `READER <GUID>` e `WRITER <GUID>` (o in alternativa, `READER VERBOSE` e `WRITER VERBOSE`), eseguiti nella Figura 5.5, che **mostrano una parte delle policy QoS adoperate** dai DataReader e i DataWriter.

```
Insert a command for Fast DDS Spy:
>> reader 01.01.d5.b6.2f.5b.fe.f5.93.6a.bb.3c|80.0.1.7
guid: 01.01.d5.b6.2f.5b.fe.f5.93.6a.bb.3c|80.0.1.7
participant: RTI Shapes Demo
topic:
  name: Square
  type: ShapeType
qos:
  durability: volatile
  reliability: best-effort

Insert a command for Fast DDS Spy:
>> writer 01.01.2d.9a.1b.de.fc.87.e2.0c.02.65|80.0.0.2
guid: 01.01.2d.9a.1b.de.fc.87.e2.0c.02.65|80.0.0.2
participant: RTI Shapes Demo
topic:
  name: Square
  type: ShapeType
qos:
  durability: volatile
  reliability: reliable
```

Figura 5.5: Esecuzione dei comandi `READER <GUID>` e `WRITER <GUID>`.

5.3 DDSFuzz

DDSFuzz è un tool di analisi open source con l'obiettivo di effettuare test di **tipo fuzz** in modo da testare vari input che il middleware DDS può ricevere durante la sua esecuzione. Prima di questo strumento non esistevano (o non erano disponibili al pubblico) soluzioni per effettuare questo tipo di test specifico sul DDS, mancanza che ora DDSFuzz cerca di colmare. Infatti anche se esistono tool inclusi RoboFuzz e Deng, questi ultimi vengono impiegati solamente per testare applicativi ROS (Robotic Operation System), che sfruttano solo una parte delle funzioni del DDS, **tralasciando funzionalità del middleware** che non vengono impegate da ROS [26].

5.3.1 Fuzz testing

Prima di parlare del tool DDSFuzz dobbiamo comprendere il significato di fuzzing (chiamato anche fuzz test) che è alla base del funzionamento di questo tool. Il fuzzing è un test automatizzato che serve a **trovare e creare casi limite** utilizzando dati di input invalidi in modo tale da esplorare più vulnerabilità possibili all'interno di un software. Inizialmente i primi tool di fuzzing non erano molto semplici da utilizzare e poco conosciuti, ma con il tempo sono migliorati perchè resi più user-friendly aumentando così la loro frequenza di uso per testare software. Oggi, il fuzzing viene applicato a diversi tipi di applicativo, tra cui compilatori, applicazioni, protocolli di rete e kernel [13].

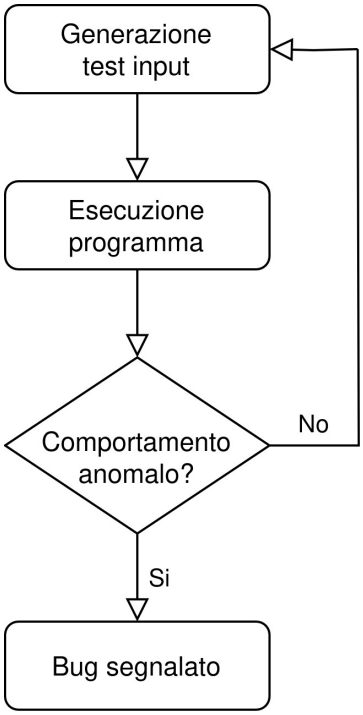

Figura 5.6: Diagramma di flusso del funzionamento di un fuzzer.

Questi fuzzing test **simulano un attacco** mandando al software che si vuole testare input regolari e irregolari, in modo tale da poter accedere ad ogni parte del codice con ogni tipo di input possibile. In questo modo è possibile analizzare il comportamento del software e distinguere casi anomali o insicuri che non dovrebbero essere impiegati. Infatti quando questo test è attivo il fuzzer riceverà varie informazio-

ni riguardanti lo stato e l'output del programma, in modo tale da poter distinguere il suo comportamento. Se viene scoperto qualche comportamento anomalo (ad esempio un segmentation fault) il fuzzer segnalerà in un log gli input utilizzati in modo tale da poter segnalare il bug, nella Figura 5.6 viene mostrato un diagramma di flusso per spiegare il suo funzionamento.

Tuttavia, dato che questi test sono generalizzati per essere compatibili con vari software, si crea una sorta di imprecisione nel mandare i vari input. Questi **input a volte non sono variegati a sufficienza per analizzare ogni singola parte del software** che stiamo analizzando. Inoltre in molti casi può anche succedere che vengano creati dei test che non sono necessari consumando così risorse e tempo inutilmente.

Per migliorare la qualità di questi test è stato creato il *DDSFuzz che essendo un fuzzer specifico solamente per il middleware DDS* rimuove molte delle imprecisioni di un fuzzer più generico.

5.3.2 Punti di forza del DDSFuzz

DDSFuzz essendo specializzato per il DDS prende in considerazione molte sue caratteristiche che non sono presenti in altri software o protocolli. La prima considerazione che viene effettuata da DDSFuzz e che avviene durante l'esecuzione del middleware DDS è la topologia della rete che può mutare nel tempo creando problemi per i fuzzer tradizionali che non si adattano ai suoi cambiamenti. Infatti certi dispositivi si possono connettere o disconnettere dalla rete a loro piacimento rendendo quindi necessario cambiare i destinatari degli input di test evitando così di trasmettere informazioni verso un'entità che si è disconnessa e quindi impossibilitata a ricevere ulteriori aggiornamenti. DDSFuzz **prende anche in considerazione le policy QoS e le funzionalità abilitate dal DDS security**, come l'autenticazione e il controllo accessi, in modo tale da poter creare test più accurati.

Finita l'esecuzione di DDSFuzz ci ritroviamo con due tipologie di bug che possiamo riscontrare in un applicativo DDS:

- **BUG TRADIZIONALI:** racchiude tutti i classici bug che sono presenti anche in altri software, un esempio può essere un buffer overflow;

- **BUG SEMANTICI:** questo bug avviene quando viene violata una norma definita dallo standard DDS, ad esempio un bypass dell'autenticazione. Dato che il loro comportamento non è facilmente categorizzabile risulta più difficile individuarli;

Uno dei maggiori punti di forza di DDSFuzz sta proprio nel trovare questi bug semantici, quasi impossibili da riconoscere per gli altri fuzzer. Durante l'esecuzione di DDSFuzz vengono eseguite in parallelo tre implementazioni (Fast DDS [4], Cyclone DDS [7], OpenDDS [2]) vendor del DDS in modo tale da identificare se i risultati dei test input rimangono consistenti tra di loro in ognuna delle esecuzioni. Se otteniamo una soluzione differente in una sola degli applicativi impiegati possiamo dire che quest'ultima ha un comportamento anomalo generato da un bug di tipo semantico [26].

5.3.3 Composizione di DDSFuzz

DDSFuzz è composto da tre elementi principali chiamati: DDS input generator, DDS program executor e bug detector.

- **DDS INPUT GENERATOR:** si occupa di generare degli input specifici per il DDS prendendo in considerazione la topologia della rete, delle policy QoS e dei parametri di sicurezza del DDS security. In questo modo non vengono generati test inutili che non servono per l'analisi dell'esecuzione. Questi test poi successivamente verranno adattati per essere compatibili con il protocollo RTPS in modo tale che il DDS program executor potrà utilizzarli immediatamente;

- **DDS PROGRAM EXECUTOR:** ricevuti gli input generati li invierà alle tre implementazioni del DDS, mandando feedback al DDS input generator sulla validità ed efficacia dei test generati;

- **BUG DETECTOR:** a differenza di altri bug detector provenienti da fuzzer tool generici, questo componente del DDSFuzz riesce a distinguere tra bug di tipo tradizionale e bug semantici, grazie all'analisi degli output delle implementazioni del DDS [26];

Capitolo 6

Conclusione

Questa tesi ha analizzato in dettaglio il Data Distribution Service (DDS), iniziando con un'introduzione in cui viene spiegato il suo funzionamento e le sue caratteristiche principali, incluse l'interoperabilità tra le diverse implementazioni dei vendors, la possibilità di cambiare la topologia della rete a runtime e una gestione di policy QoS che rendono il middleware estremamente flessibile per ogni tipo di utilizzo. Successivamente è stato eseguito un confronto tra i vari applicativi del DDS per elencare e comparare i loro punti di forza e svantaggi.

Nei successivi capitoli, dopo l'introduzione, sono state analizzate delle vulnerabilità relative allo stesso standard OMG che il DDS deve rispettare. Sono state anche mostrate delle possibili soluzioni per mitigare certe problematiche di sicurezza, tra cui il DDS security che tramite i suoi plugin permette di crittare le comunicazioni tra le varie entità e bloccare certe loro azioni se non hanno determinati permessi. Specialmente la mancanza di un sistema di autenticazione permette ad un autore malevolo di intercettare il traffico scambiato tra entità, dato che di base le comunicazioni avvengono in chiaro.

Un'altra tipologia di vulnerabilità che è stata presentata, è relativa al software DDS utilizzato dai vendors per implementarlo. Queste falle di sicurezza hanno evidenziato che soluzioni open source come FastDDS hanno maggiori possibilità di avere problematiche di sicurezza dato che la maggior parte dei CVE compilati sono relativi a questa implementazione di eProsima. Da qui possiamo dedurre che anche se il costo iniziale di una soluzione a pagamento è superiore, alla lunga garantisce una maggiore copertura.

Queste vulnerabilità hanno evidenziato come lo standard OMG può creare diverse problematiche legate alla sicurezza. Anche se molte delle vulnerabilità che sono state presentate sono ormai note, in molti casi vengono ignorate dato che i vendors cercano di mantenere il loro applicativo in regola con lo standard OMG. Questa scelta è guidata dal fatto che lo standard garantisce l'interoperabilità tra le diverse implementazioni. Purtroppo la collaborazione di queste software house sono ancora molto limitate tra di loro, se in futuro ci fosse più collaborazione si potrebbe arrivare a creare standard più rigidi in modo tale da aumentare la sicurezza complessiva del middleware.

L'OMG in futuro dovrebbe aggiornare i propri standard in modo da mitigare le problematiche legate a queste regole di interoperabilità specialmente durante il processo di discovery. Lo standard OMG proposto per rispondere alla problematica è il DDS security che purtroppo in molti casi non viene applicato dato che è difficile da configurare in una rete già esistente e sprovvista di questa estensione.

Una soluzione potrebbe essere quella di aumentare solamente la sicurezza del processo di discovery che in molti casi è il principale vettore d'attacco, magari utilizzando una parte dei plugin di sicurezza già impiegati dal DDS security.

Nell'ultima parte dell'elaborato sono stati descritti degli strumenti in grado di analizzare il traffico all'interno di una rete DDS. Per questo motivo introdotto WireShark, un potente strumento in grado di catturare pacchetti del protocollo RTPS insieme ad un'alternativa sviluppata da eProsima chiamata eProsima DDS Record & Replay. Mentre questi due strumenti sono unici e non sono stati sviluppati da differenti software house, lo stesso non si può dire per il tool Fast DDS Spy. Anche se questi strumenti garantiscono il funzionamento con più applicativi del middleware creati dai vari vendors, essi presentano dei distinti punti di forza.

Se OMG e i diversi vendors in futuro collaborassero tra di loro potrebbero creare un applicativo definitivo per monitorare una rete DDS avendo tutti i vantaggi possibili. Questo applicativo eviterebbe di creare differenze di analisi del traffico che normalmente si possono trovare quando vengono adoperati i tool ad oggi disponibili. Un altro sistema aggiuntivo per migliorare l'efficacia del controllo del network potrebbe essere quello dell'utilizzo di un'intelligenza artificiale che monitora i pacchetti scambiati tra le varie entità. Ad esempio essa potrebbe monitorare se i valori ricevuti dai sensori rimangono all'interno di un intervallo predefinito e se mantentengono una

consistenza nel tempo. Se venisse rilevato un problema, l'IA potrebbe avvisare un operatore che analizzerà più nel dettaglio l'inconsistenza individuata.

Inoltre questa AI potrebbe essere impiegata durante la configurazione iniziale della rete, specialmente per la creazione delle policy QoS per evitare di creare delle misconfigurazioni che potrebbero essere successivamente sfruttate da un attore malevolo.

L'ultimo strumento che è stato presentato all'interno di questo elaborato è DDSFuzz, un software che ci permette di effettuare test di tipo fuzzing per ogni implementazione del DDS. Infatti una delle vulnerabilità che è stata presentata è stata scoperta proprio con l'utilizzo di un tool simile. Questo dimostra che questi modelli di software sono utili per trovare falle di sicurezza che possono compromettere anche il funzionamento dei dispositivi all'interno della rete stessa. DDSFuzz è stato descritto in maniera dettagliata perché rappresenta il primo applicativo di tipo fuzzer ad essere solamente compatibile con il middleware DDS portando l'analisi dei suoi test di input ad una maggiore accuratezza. Come è stato indicato, i bug identificati si dividono in due categorie distinte, la prima relativa ai tradizionali bug, ma nella seconda il vero punto di forza di DDSFuzz sono i bug semantici. Solo grazie all'esecuzione parallela di diversi applicativi DDS sarà possibile individuare quest'ultima categoria di bug che avvengono quando una o più delle norme del DDS, come ad esempio l'obbligo di autenticazione, potrebbe essere bypassato con uno specifico input.

DDSFuzz è relativamente nuovo e ci dimostra una mancanza di strumenti d'analisi per trovare problematiche di funzionamento del DDS e della sua sicurezza, dato che sono subito state trovate nuove CVE dopo la sua esecuzione.

In conclusione, questo elaborato ha dimostrato come, in futuro, sia necessario avanzare nuovi studi per la sicurezza del middleware DDS. Molte aziende che adoperano questa soluzione non sono interessate ad creare nuove misure di protezione dato che mantengono il network DDS in un ambiente protetto senza possibilità di accesso da parte di esterni, rendendolo un sistema chiuso. Tuttavia, con la crescente domanda di interconnessioni e diffusione di reti sempre più complesse, potrebbe rivelarsi insufficiente nel lungo termine.

Il DDS è una tecnologia ancora in sviluppo che, con il tempo, potrà aiutare a risolvere tante problematiche attuali legate alla comunicazione in ambienti distribuiti.

Se prendiamo il 5G, un insieme di tecnologie che permette di connettere ad

Internet un grande numero di dispositivi inclusi quelli di tipo IoT, possiamo osservare come la domanda di interconnettere più dispositivi tra di loro stia crescendo. Questa interconnessione in futuro non troppo distante, potrebbe portare a creare un sistema in grado di permettere a una vasta rete di sensori di comunicare in tempo reale tra di loro.

In questo scenario, il DDS potrebbe rappresentare una efficiente soluzione per permettere l'interconnessione di questa vasta rete di sensori, evitando di impiegare tecnologie come il protocollo TCP/IP, che non è adatto a gestire un ambiente con un network dinamico. Tuttavia, affinché questa evoluzione possa realizzarsi è necessario incrementare misure di protezione più efficaci, capaci di risolvere almeno le vulnerabilità che sono state mostrate in questo elaborato.

Bibliografia

[1] Bruno Blanchet, Ben Smyth, Vincent Cheval, and Marc Sylvestre. *ProVerif 2.05: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial*, 2023. URL `https://bblanche.gitlabpages.inria.fr/proverif/manual.pdf`. [Accesso: 2 febbraio 2025].

[2] Object Computing. OpenDDS, February 2025. URL `https://opendds.org`. [Online; accessed 20. Mar. 2025].

[3] eProsima. DDS Record & Replay, October 2024. URL `https://www.eprosima.com/middleware/tools/dds-record-replay`. [Online; accessed 28. Mar. 2025].

[4] eProsima. eProsima Fast DDS, March 2025. URL `https://www.eprosima.com/middleware/fast-dds`. [Online; accessed 20. Mar. 2025].

[5] eProsima. eProsima Fast DDS Spy, March 2025. URL `https://www.eprosima.com/middleware/tools/fast-dds-spy`. [Online; accessed 28. Mar. 2025].

[6] eProsima. 1.1. What is DDS? Fast DDS 3.1.2 documentation, February 2025. URL `https://fast-dds.docs.eprosima.com/en/latest/fastdds/getting_started/definitions.html`. [Online; accessed 24. Mar. 2025].

[7] Eclipse Foundation. Eclipse Cyclone DDS - Home, March 2025. URL `https://cyclonedds.io`. [Online; accessed 20. Mar. 2025].

[8] Inc. Foxglove Technologies. Foxglove - Visualization and observability for robotics developers., March 2025. URL `https://foxglove.dev`. [Online; accessed 28. Mar. 2025].

[9] Inc. Foxglove Technologies. MCAP, March 2025. URL https://mcap.dev. [Online; accessed 28. Mar. 2025].

[10] Real-Time Innovations. Using Wireshark with RTI Connext DDS Systems documentation, July 2020. URL https://community.rti.com/static/documentation/wireshark/2020-07/doc. [Online; accessed 26. Mar. 2025].

[11] Real-Time Innovations. RTI Connext Overview, April 2024. URL https://www.rti.com/products/connext-professional. [Online; accessed 22. Mar. 2025].

[12] Jesse Rengers. Dds in a zero trust cloud native environment in the naval domain, November 2022. URL http://essay.utwente.nl/93639/.

[13] Hongliang Liang, Xiaoxiao Pei, Xiaodong Jia, Wuwei Shen, and Jian Zhang. Fuzzing: State of the art. *IEEE Transactions on Reliability*, 67(3):1199–1218, Sep. 2018. ISSN 1558-1721. doi: 10.1109/TR.2018.2834476.

[14] Víctor Mayoral-Vilches. Robot hacking manual (rhm). *arXiv preprint arXiv:2203.04765*, 2022.

[15] Michael James Michaud, Thomas R. Dean, and Sylvain P. Leblanc. MALICIOUS USE OF OMG DATA DISTRIBUTION SERVICE (DDS) IN REAL-TIME MISSION CRITICAL DISTRIBUTED SYSTEMS, April 2017. URL https://espace.rmc-cmr.ca/jspui/handle/11264/1241. [Online; accessed 11. Feb. 2025].

[16] Michael James Michaud, Thomas R. Dean, and Sylvain P. Leblanc. Attacking OMG data distribution service (DDS) based real-time mission critical distributed systems. In *13th International Conference on Malicious and Unwanted Software, MALWARE 2018, Nantucket, MA, USA, October 22-24, 2018*, pages 68–77. IEEE, 2018. doi: 10.1109/MALWARE.2018.8659368. URL https://doi.org/10.1109/MALWARE.2018.8659368.

[17] Object Management Group. OMG Data Distribution Service, April 2015. URL http://www.omg.org/spec/DDS/1.4/PDF. [Accesso: 2 febbraio 2025].

[18] Object Management Group. DDS Security, July 2018. URL https://www.omg.org/spec/DDS-SECURITY/1.1/PDF. [Accesso: 2 febbraio 2025].

[19] Object Management Group. About the DDS Interoperability Wire Protocol Specification Version 2.5, February 2025. URL https://www.omg.org/spec/DDSI-RTPS/2.5/About-DDSI-RTPS. [Online; accessed 12. Feb. 2025].

[20] Sangyoon Oh, Jai-Hoon Kim, and Geoffrey Fox. Real-time performance analysis for publish/subscribe systems. *Future Generation Computer Systems*, 26 (3):318–323, 2010. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2009.09.001. URL https://www.sciencedirect.com/science/article/pii/S0167739X09001344.

[21] OMG. Topic [DDS Foundation Wiki], February 2025. URL https://www.omgwiki.org/ddsf/doku.php?id=ddsf:public:guidebook:06_append:glossary:t:topic. [Online; accessed 5. Feb. 2025].

[22] OMG. What is DDS?, February 2025. URL https://www.dds-foundation.org/what-is-dds-3. [Online; accessed 6. Feb. 2025].

[23] RTI. Instance | data distribution service (dds) community rti connext users, February 2025. URL https://community.rti.com/glossary/instance. [Online; accessed 2025-02-05].

[24] RTI. DomainParticipant | Data Distribution Service (DDS) Community RTI Connext Users, February 2025. URL https://community.rti.com/glossary-term/domainparticipant. [Online; accessed 6. Feb. 2025].

[25] RTI. Domain | Data Distribution Service (DDS) Community RTI Connext Users, February 2025. URL https://community.rti.com/glossary/domain. [Online; accessed 6. Feb. 2025].

[26] Dohyun Ryu, Giyeol Kim, Daeun Lee, Seongjin Kim, Seungjin Bae, Junghwan Rhee, and Taegyu Kim. Differential fuzzing for data distribution service programs with dynamic configuration. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ASE '24, page 807–818, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400712487. doi: 10.1145/3691620.3695073. URL https://doi.org/10.1145/3691620.3695073.

[27] J.M. Schlesselman, Gerardo. Pardo-Castellote, and Bert. Farabaugh. Omg data-distribution service (dds): architectural update. In *IEEE MILCOM 2004. Military Communications Conference, 2004.*, volume 2, pages 961–967 Vol. 2, 2004. doi: 10.1109/MILCOM.2004.1494965.

[28] Dave Seltz. Comparing Open Source DDS to RTI Connext DDS: Considerations in Picking the Right DDS Solution to Run Your Distributed System, March 2025. URL `https://www.rti.com/blog/picking-the-right-dds-solution`. [Online; accessed 21. Mar. 2025].

[29] Inc Twin Oaks Computing. CoreDX DDS Data Distribution Service Middleware, March 2025. URL `https://www.twinoakscomputing.com/coredx`. [Online; accessed 22. Mar. 2025].

[30] Binghan Wang, Hui Li, and Jingjing Guan. A formal analysis of data distribution service security. In Jianying Zhou, Tony Q. S. Quek, Debin Gao, and Alvaro A. Cárdenas, editors, *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security, ASIA CCS 2024, Singapore, July 1-5, 2024*. ACM, 2024. doi: 10.1145/3634737.3656288. URL `https://doi.org/10.1145/3634737.3656288`.

[31] Nanbor Wang, Douglas C. Schmidt, Hans van't Hag, and Angelo Corsaro. Toward an adaptive data distribution service for dynamic large-scale network-centric operation and warfare (ncow) systems. In *MILCOM 2008 - 2008 IEEE Military Communications Conference*, pages 1–7, 2008. doi: 10.1109/MILCOM.2008.4753364.

[32] Thomas White, Michael N. Johnstone, and Matthew Peacock. An investigation into some security issues in the dds messaging protocol, 2017. URL `https://api.semanticscholar.org/CorpusID:52840449`.

[33] He Yuefeng. Study on data transmission of dcps publish-subscribe model. In *2018 2nd IEEE Advanced Information Management,Communicates,Electronic and Automation Control Conference (IMCEC)*, pages 1–2172, May 2018. doi: 10.1109/IMCEC.2018.8469351.

Ringraziamenti

Vorrei ringraziare il mio professore relatore, Francesco Santini, per la sua disponibilità e il suo supporto durante la stesura di questa tesi. Grazie ai miei genitori per il loro sostegno continuo, ai miei amici per avermi accompagnato in questo percorso con la loro vicinanza e alla mia ragazza per la pazienza e l'incoraggiamento costante. Questa tesi è anche merito vostro.

Federico