

Corso di Laurea Magistrale in

Ingegneria Informatica

Progetto di

Network Security

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Studenti**  Di Cello Danny 224527  Seminara Giuseppe 227540  Magurno Roberto 227501  Miriello Raffaele 227564 | |  | |
|  |  |  | |

Anno Accademico 2021/2022

***Indice***

*1 Introduzione 3*

*2 Ambiente di Sviluppo 5*

*3 Attacchi 7*

*3.1 DoS Attack 7*

*3.2 Rank Attack 13*

*3.3 Wormhole Attack 23*

*3.4 Fragment Duplication Attack*

*4 Conclusioni 29*

*5 Bibliografia 31*

***Introduzione***

L’obiettivo del seguente progetto è l’implementazione di un insieme di attacchi, e delle rispettive mitigazioni, in un ambiente IoT simulato. In questo tipo di contesti abbiamo a che fare con dispositivi dotati di bassa potenza di elaborazione che comunicano, ed è proprio a causa di tali caratteristiche che, i protocolli di comunicazioni devono essere leggeri in modo da poter essere eseguiti senza causare un overhead eccessivo. Per tale motivo spesso, tali protocolli, tralasciano alcuni aspetti di sicurezza che richiederebbero un elevato quantitativo di risorse, lasciando uno spazio per potenziali attacchi. Nel nostro caso di studio specifico sfrutteremo questi spiragli di sicurezza in riferimento ai protocolli **6LoWPAN** e **RPL**, implementando in particolare i seguenti attacchi:

* ***DoS Attack.***
* ***Rank Attack.***
* ***Warmhole Attack.***
* ***Fragment Duplication Attack****.*

*L’elaborato ha la seguente struttura:*

*Dopo aver descritto brevemente gli obiettivi del nostro lavoro, concludiamo la seguente sezione fornendo una breve descrizione dei due protocolli analizzati, al fine di rendere maggiormente comprensibile la discussione affrontata nei capitoli successivi.*

*Nel secondo capitolo descriviamo l’ambiente di sviluppo utilizzato per l’implementazione degli scenari descritti.*

*Nel terzo capitolo ci occupiamo di descrivere gli attacchi realizzati e le possibili mitigazioni in riferimento a diversi scenari applicativi.*

*Infine, concludiamo l’elaborato, traendo le conclusioni sul lavoro effettuato, analizzando e descrivendo i punti salienti affrontati ed eventuali effetti collaterali derivanti dagli scenari considerati.*

**6LoWPAN (Ipv6 over Low-Power Wireless Personal Area Networks)**

È un protocollo che definisce meccanismi di incapsulamento e compressione dell’intestazione che consentono l’invio e la ricezione di pacchetti IPv6 su reti basate su IEE 802.15.4, ovvero Low-Rate Wireless Personal Aerea Networks. Ai nodi IPv6 vengono assegnati indirizzi IP gerarchici a 128 bit, attraverso l’uso di un prefisso di lunghezza arbitraria. 6LoWPAN permette di superare i problemi legati alle grandi dimensioni delle intestazioni IPv4 e IPv6, permettendo anche a dispositivi a bassa potenza, con risorse hardware limitate di comunicare. Le specifiche di 6LoWPAN sono definite nei seguenti documenti RFC: **RFC 4919** e **RFC 4944**.

I principali problemi legati ai dispositivi LoWPAN sono i seguenti:

* **Frammentazione e riassemblaggio**: I vincoli di dimensione del pacchetto imposti da IPv6 e 802.15.4 rappresentano un problema di eccessiva frammentazione/riassemblaggio.
* **Compressione dell’intestazione IPv6**: con l’attuale intestazione IPv6 di 40 byte, il payload è ridotto. La compressione dell’intestazione IPv6 rappresenta una necessità per ottimizzare i trasferimenti su una rete 6LoWPAN.
* **Routing**: Le reti LoWPAN sono costituite da moltissimi nodi, è necessario adottare quindi un protocollo di instradamento per supportare tali reti, che al contempo soddisfi i vincoli hardware dei dispositivi stessi.

**RPL (IPv6 Routing Protocol for Low-Power and Lossy Networks)**

È un protocollo di routing per reti wireless a bassa potenza e generalmente soggette a perdita di pacchetti, ottimizzato per la comunicazione multi-hop e la comunicazione molti-a-uno, ma supporta anche la comunicazione uno-a-uno. Il protocollo è definito nel RFC 6550.

RPL crea una topologia simile ad un albero, che prende il nome di DAG (Grafo Diretto Aciclico). Ogni nodo all’interno della rete ha un rank assegnato, che aumenta man mano che i nodi si allontanano dal nodo radice. I nodi scelgono dunque il rank più basso come criterio di selezione della route. I messaggi ICMPv6 sono definiti nel RFC 4443, e in particolare possiamo individuare i seguenti:

* ***DIS (Information Request DODAG)****:* Utilizzato per richiedere informazioni al DODAG nelle vicinanze. È analogo ai messaggi di richiesta inviati dal router per scoprire le reti esistenti.
* ***DIO (Object of Information of the DAG):*** Messaggio che condivide le informazioni del DAG, inviato in risposta ai messaggi DIS, e utilizzato per aggiornare periodicamente le informazioni sulla topologia della rete.
* ***DAO (Object of Update to the Destination):*** Messaggio inviato dai vari nodi per aggiornare le informazioni relative al loro nodo padre in tutto il DAG.

***Ambiente di Sviluppo***

Per l’implementazione degli scenari descritti si è deciso di utilizzare Contiki-NG, come sistema operativo per i dispositivi IoT, e il simulatore COOJA per costruire il nostro ambiente virtualizzato. La principale motivazione che ha influito su tale scelta è che Contiki è il sistema operativo più utilizzato per l’implementazione del protocollo RPL, essenziale per i nostri scopi, inoltre il sistema è Open Source e supporta un elevato numero di dispositivi diversi, dai computer a 8 bit fino ai nodi che formano le reti di sensori.

Di seguito illustriamo il processo di installazione e configurazione dell’ambiente.

Sebbene sia possibile seguire diverse strade per l’installazione del sistema Contiki, abbiamo deciso di seguire l’approccio consigliato nella documentazione ufficiale, utilizzando quindi un’immagine Docker. La scelta effettuata presenta diversi vantaggi, i cui due più rilevanti sono i seguenti:

* ***Facilità di installazione***: l’installazione richiede pochi passi, al termine dei quali si dispone di un ambiente totalmente configurato e funzionante.
* ***Compatibilità:*** l’installazione attraverso un’immagine Docker fornisce una maggiore compatibilità ed un maggior numero di dispositivi simulati, rispetto ad un processo di installazione nativa sul sistema Host.

***Requisiti***

Se non è già presente sul sistema è necessario installare il pacchetto Docker-CE, con il seguente comando:

*$ sudo apt-get install docker-ce*

Successivamente è opportuno aggiungere l’utente corrente al gruppo Docker. Questo passaggio è facoltativo, ma permette di interagire col Docker senza l’utilizzo dei privilegi di amministratore.

*$ sudo usermod -aG docker ns*

***Installazione***

Installato Docker-ce sul sistema host è possibile procedere al download dell’immagine di Contiki-ng digitando

*$ docker pull contiker/contiki-ng*

Tale comando scarica l’ultima versione di Contiki disponibile. Una volta completato il download è necessario clonare il repository GitHub di Contiki-ng e inizializzare tutti i sotto moduli con i seguenti comandi

*$ git clone https://github.com/contiki-ng/contiki-ng.git*

*$ cd contiki-ng*

*$ git submodule update --init –recursive*

Infine, per semplificare il processo di avvio è stato creato il seguente alias:

*export CNG\_PATH=/home/ns/contiki-ng*

*alias contiker="docker run --privileged --sysctl net.ipv6.conf.all.disable\_ipv6=0 --mount type=bind,source=$CNG\_PATH,destination=/home/user/contiki-ng -e DISPLAY=$DISPLAY -v /tmp/.X11-unix:/tmp/.X11-unix -v /dev/bus/usb:/dev/bus/usb -ti contiker/contiki-ng"*

A questo punto l’ambiente di sviluppo è installato ed è possibile lanciare il simulatore digitando il comando

$ contiker cooja

***Struttura Cartelle***

Per la riproduzione degli esperimenti sviluppati è necessario identificare due cartelle importanti:

~/contiki-ng: contiene tutti i file di configurazione di Contiki-NG.

~/attacchi: contiene tutti i file modificati per implementare gli attacchi e le rispettive mitigazioni descritte nel capitolo 3.

La cartella attacchi è suddivisa in sottocartelle ognuna delle quali contiene l’implementazione di un attacco e della rispettiva mitigazione, riferita a due topologie differenti, una media e una più complessa (cartella Complex). Inoltre, è presente uno script Bash che permette di preparate tutti i file necessari e pulire l’ambiente prima di caricare una simulazione, utilizzando i file di backup presenti nella rispettiva cartella.

***Attacchi***

In questo capitolo andremo a descrivere i diversi attacchi implementati, esponendo una trattazione teorica degli stessi e allegando il relativo codice implementativo, opportunamente documentato.

***DOS (Denial of Service)***

L’attacco DoS mira a compromettere la disponibilità di un servizio, sovraccaricando una risorsa con richieste inutili, impedendole così di soddisfare le richieste legittime. L’attacco DoS è un attacco difficile da mitigare perché sfrutta il normale schema di funzionamento del sistema.

Nel nostro caso specifico abbiamo implementato due varianti dell’attacco, nella prima variante abbiamo una topologia semplice, composta da un solo nodo attaccante ed un solo nodo vittima, nella seconda variante, invece, abbiamo implementato una topologia più complessa dove simuliamo un DDoS (Distributed Denial of Service), in cui sono presenti più attaccanti che attaccano lo stesso nodo vittima.

Prima di analizzare i due casi specifichi, per poter effettuare un confronto, mostriamo con la seguente figura, un possibile scenario legittimo.

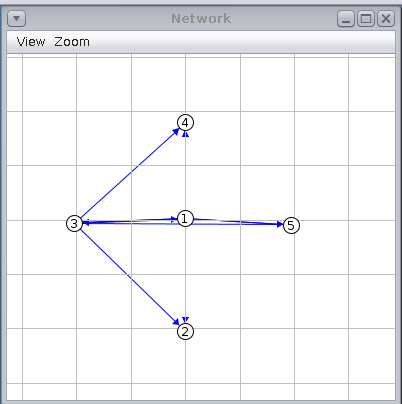


Figura a: Topologia base

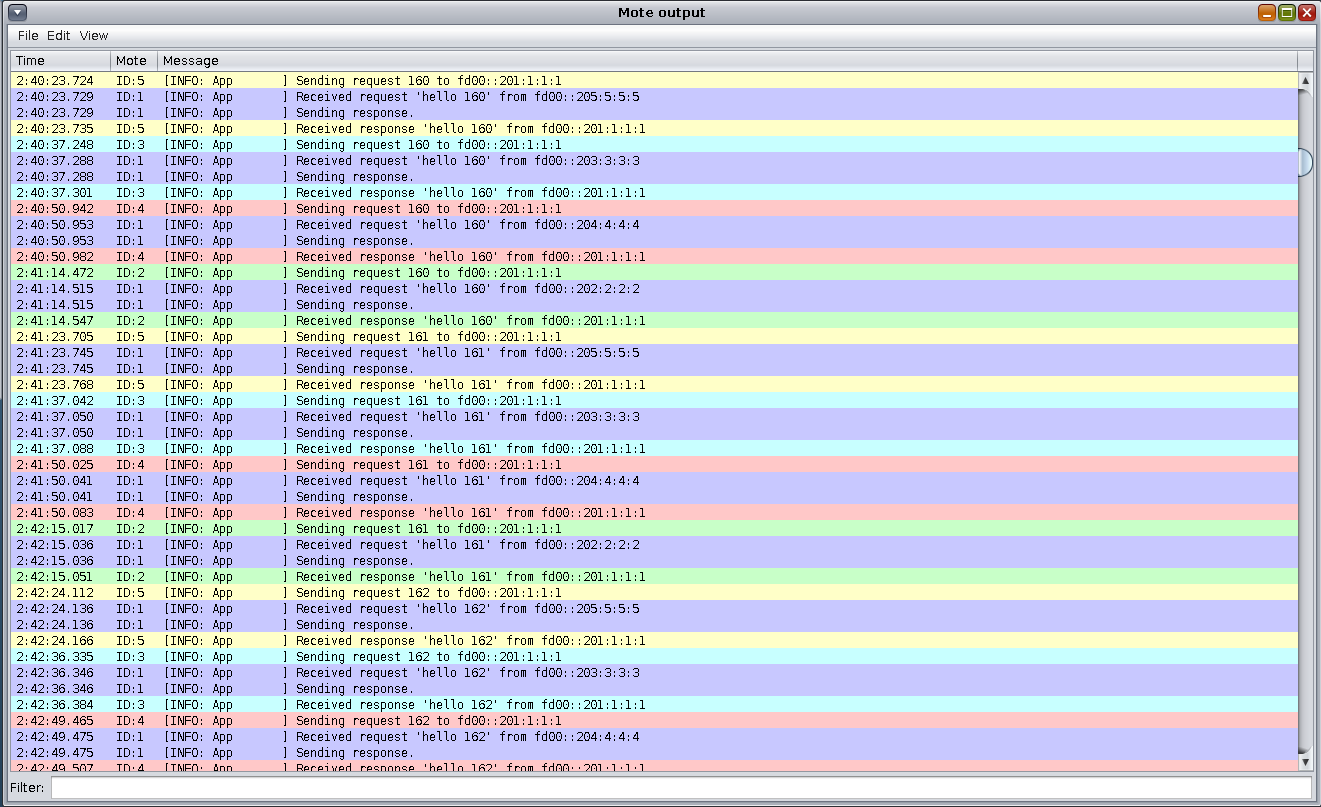
La figura 1a mostra la topologia della rete considerata, che consiste in un server UDP, identificato da ID=1, e di quattro nodi client che inviano richieste ad intervalli casuali.

Figura 1b: Log Scenario Legittimo

La figura 1b mostra il log dei messaggi scambiati ed è possibile osservare che subito dopo che un client fa la richiesta, il server la riceve e invia una risposta che viene ricevuta correttamente dal client.

Andiamo a descrivere ora le due varianti di attacco implementate.

***Variante 1***

Questa prima variante è un semplice attacco DoS, con una topologia relativamente semplice, ma funzionale per la comprensione dell’attacco, in cui troviamo un solo nodo attaccante che inonda il server di richieste impedendogli di rispondere a richieste legittime.

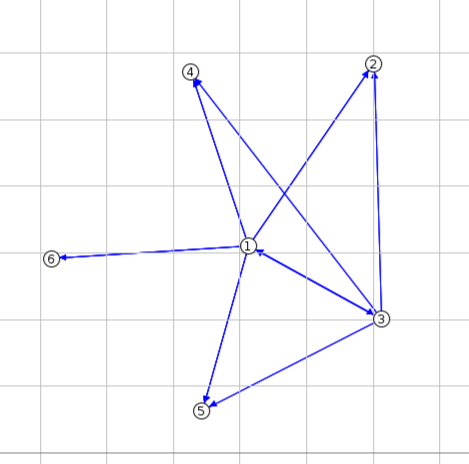


Figura 2a: Topologia DoS

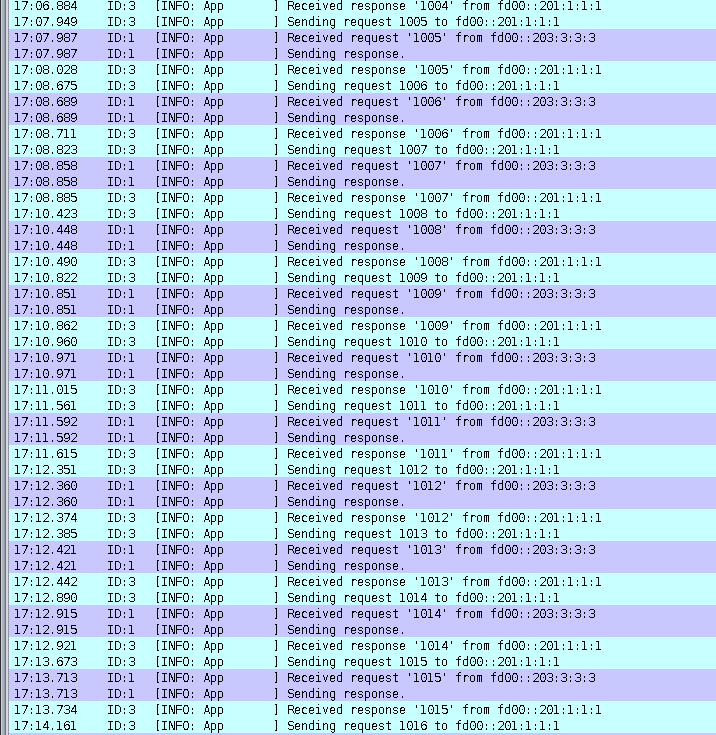


Figura 2b: Log attacco DoS

La figura 2a mostra la topologia di questo attacco, in cui abbiamo 1 server, identificato sempre da ID=1 lì dove non esplicitamente indicato diversamente, e 5 nodi client di cui 4 legittimi e uno malevolo indicato con ID=3. Come è possibile osservare dal log il nodo 3 inonda di richieste il server consumando tutte le sue risorse, costringendolo a scartare le richieste provenienti dai nodi legittimi.

*Mitigazione*

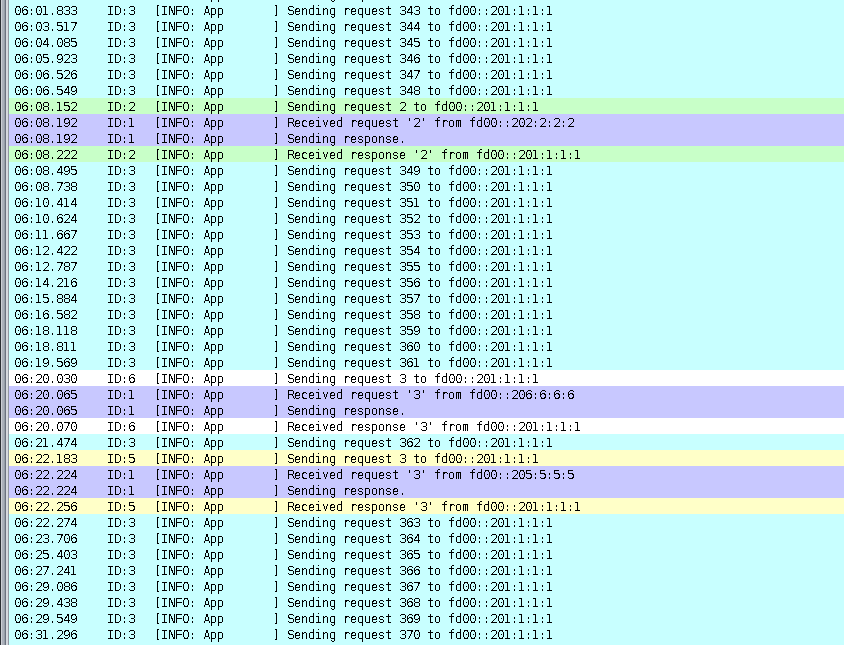
*La mitigazione dell’attacco si basa sul rate di ricezione dei messaggi da parte del server. In particolare, si fissa una soglia, stabilita in modo empirico sulla base della topologia della rete, e si vanno a scartare i messaggi provenienti da nodi che hanno un rate più elevato rispetto alla soglia stabilita. Il valore della soglia ricopre un’importanza cruciale, fissando un valore troppo basso, infatti, anche i nodi legittimi potrebbero essere classificati come malevoli, viceversa fissando un valore troppo elevato potrebbe passare troppo tempo prima di accorgersi dell’attacco consumando tutte le risorse a disposizione e iniziando a scartare richieste legittime. Per implementare tale mitigazione è stato creato un thread sul server che calcola e aggiorna periodicamente il rate di ricezione dei pacchetti per ogni nodo presente nella rete e che comunica con lui. Sulla base di tale valore calcolato, ogni volta che arriva un pacchetto si decide se elaborarlo o scartarlo.*

Figura 2c: Mitigazione DoS

*Come è possibile osservare dalla figura 2c, applicando tale mitigazione, il server non risponde più ai messaggi di richiesta del nodo malevolo 3, e al contempo riesce a rispondere correttamente alle richieste dei nodi legittimi.*

*L’elevata presenza di messaggi da parte del nodo 3 rispetto ai nodi legittimi è legata al fatto che ha un rate di invio molto elevato rispetto agli altri nodi.*

***Variante 2***

In questo secondo scenario si vuole simulare un attacco DDoS, abbiamo quindi una topologia più complessa in cui abbiamo 4 nodi malevoli (ID: 3, 16, 9, 6) che attaccano il nodo server. Mostriamo in modo analogo al caso precedente prima l’attacco e subito dopo lo scenario con la mitigazione attiva, quest’ultima è la medesima del caso precedente, ma come si può osservare dalle figure seguenti risulta efficace.

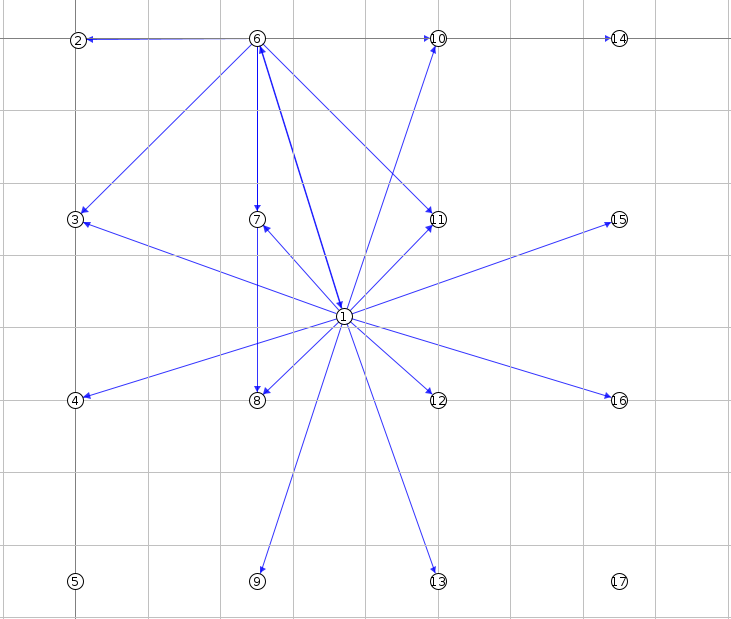
**

Figure 3a: Topologia DDoS

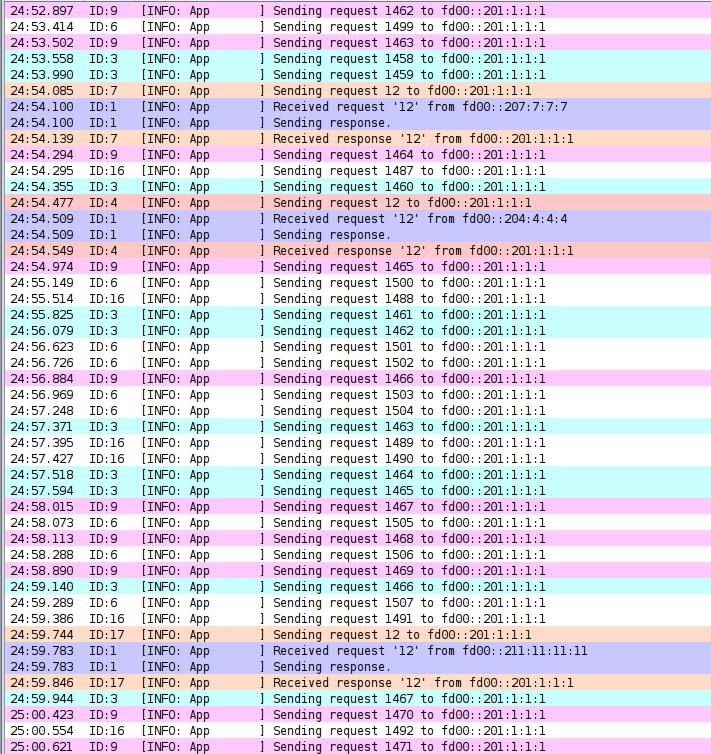


Figura 3b: Mitigazione DDoS

Come è possibile osservare in questo secondo scenario descritto manca il log riferito all’attacco senza la mitigazione attiva. Questo è dovuto al fatto che nel momento in cui si esegue il DDoS, i nodi malevoli consumano tutte le risorse del Server, facendo terminare, quasi immediatamente, la simulazione con un errore. La figura 3b, oltre la corretta esecuzione dello scenario con la mitigazione attiva, permettono di visualizzare l’efficacia della soluzione proposta.

Il codice, opportunamente commentato, relativo a questo attacco e le rispettive simulazioni avviabili con Cooja sono presenti all’interno delle seguenti cartelle:

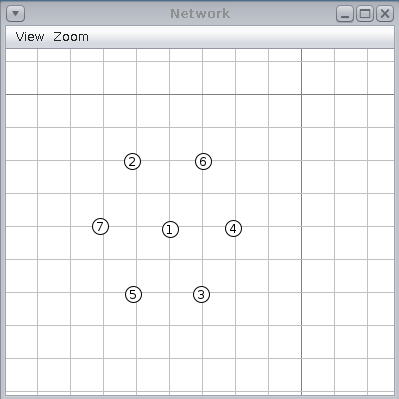
* Attacchi/DOS.
* Attacchi/Complex/DOS.

***Rank Attack***

L’introduzione del concetto di rank all’interno del protocollo RPL risulta utile a diverse applicazioni come l’ottimizzazione del percorso o la prevenzione di cicli, ma al contempo rappresenta un efficace vettore di attacco, che può incidere sulle performance della rete. Un nodo malevolo può infatti falsificare il valore del rank, e gli altri nodi non hanno modo di verificare l’autenticità di tale valore.

In questa tipologia di attacco un nodo malevolo modifica il proprio rank in modo da attirare e manipolare l’intera rete, causando, ad esempio, ritardi nella consegna dei messaggi, perdita dei pacchetti, creazione di loop che rendono la topologia instabile e rendendo necessario un elevato numero di messi di controllo per correggere la struttura della rete.

Di seguito mostriamo un possibile scenario legittimo e successivamente proponiamo tre varianti dell’attacco, con due topologie differenti.



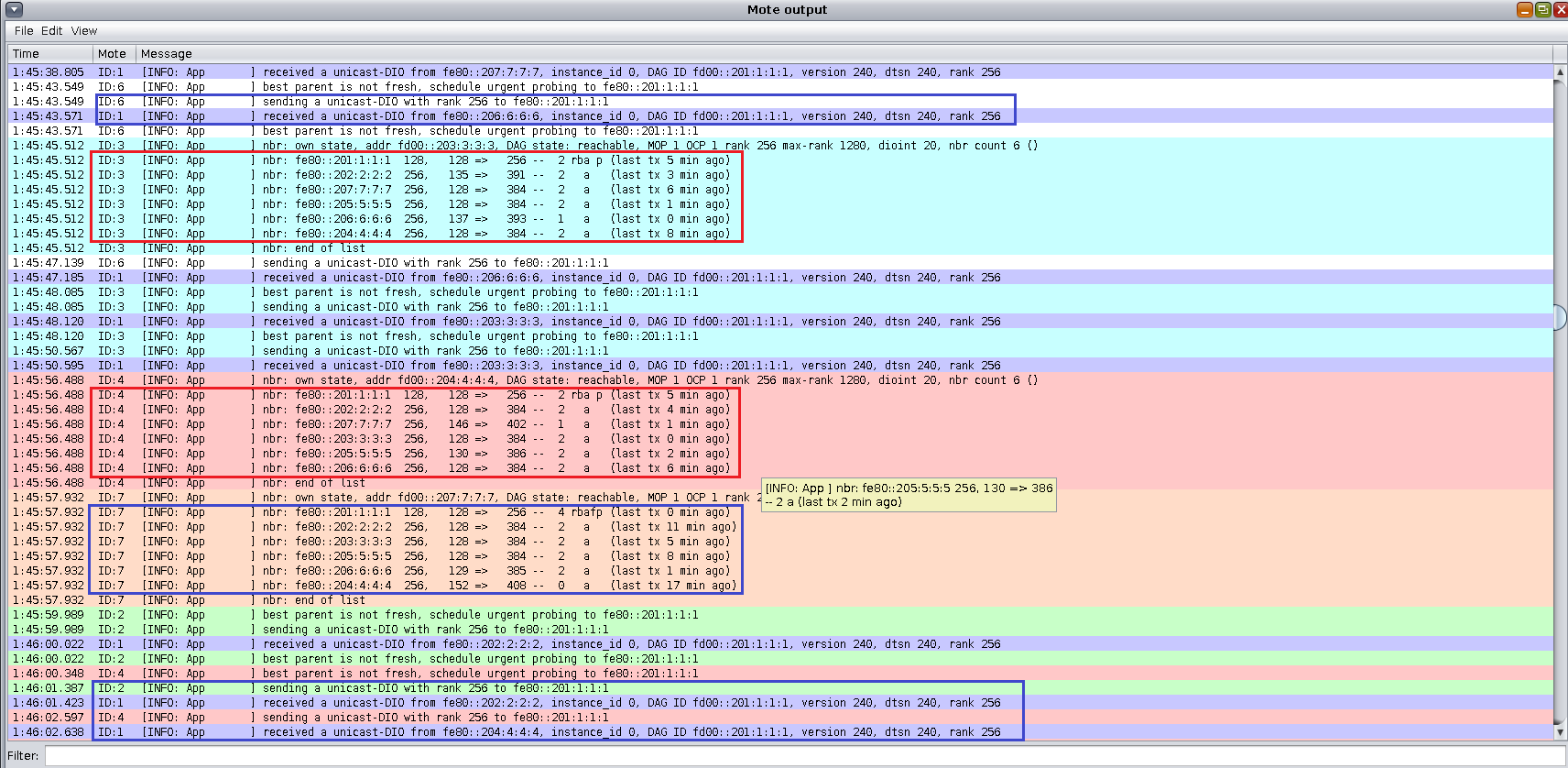


Figura 4b: Log scenario Legittimo

Figura 4a: Topologia Base

In questo scenario base abbiamo 7 nodi (Figura 4a), di cui 1 svolge il ruolo di server, mentre i restanti 6 svolgono il ruolo di client. Nella figura 4b possiamo vedere i log in riferimento a tale scenario legittimo, in cui vengono evidenziati, in blu, alcuni messaggi DIO inviati/ricevuti, e in rosso, i DAG dei nodi 3, 4, 7.

***Attacchi***

***Variante 1***

In questo scenario implementiamo la topologia rappresentata in figura 4a, in cui si assume che il nodo 4 sia malevolo. Tale nodo effettuerà un poisoning del suo valore di rank nei confronti del solo nodo 3, designato come vittima. In particolare, dall’analisi dei log (Figura 5a) è possibile osservare come il nodo attaccante con ID=4 invii dei messaggi DIO con rank posto pari a 1000 solo al nodo 3, il quale riceve e registra tale valore di rank associato al nodo attaccante nel proprio DAG.

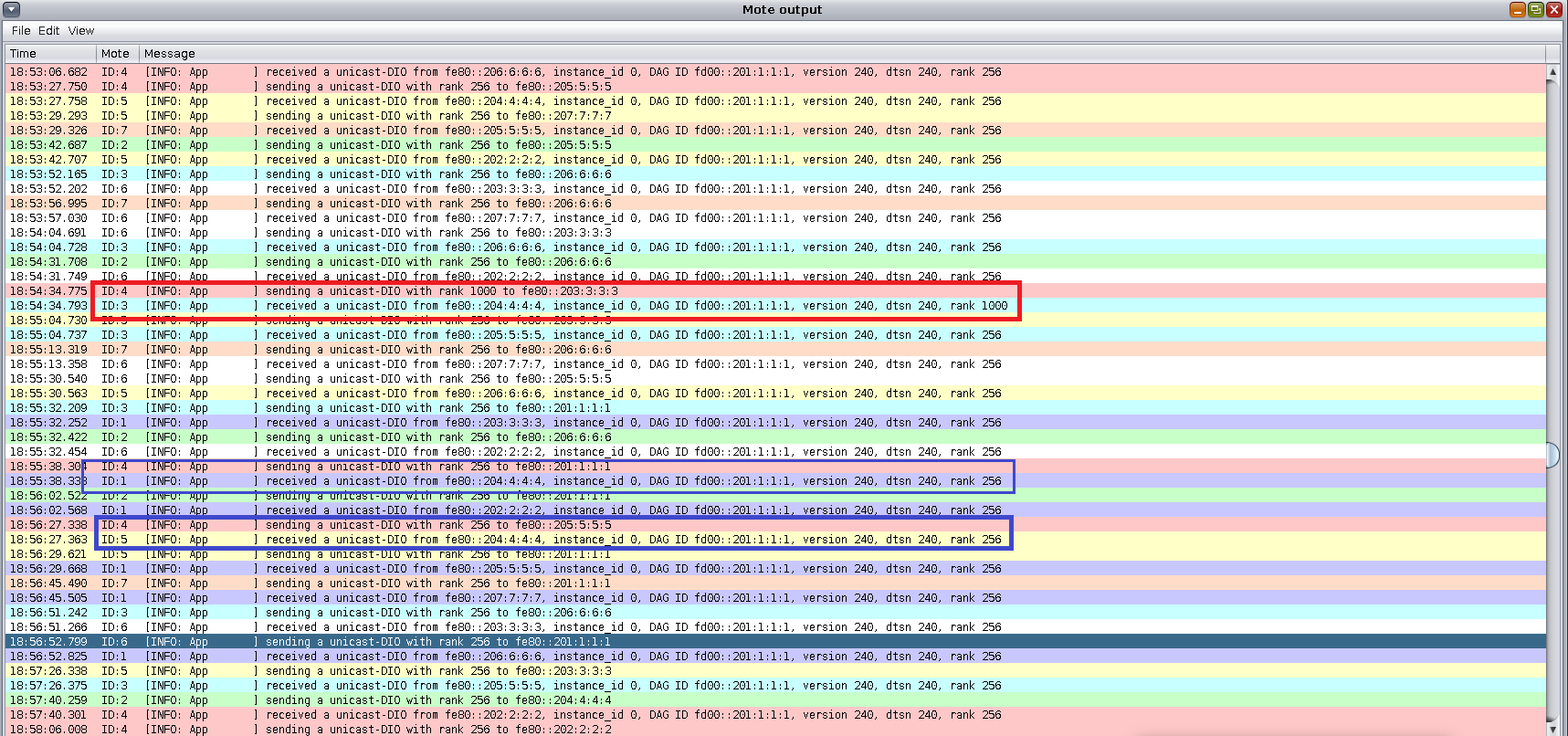
Per semplificare la visualizzazione è stato cerchiato in blu il messaggio DIO indirizzato al nodo 3 con rank alterato, mentre in rosso è stato evidenziato i messaggi DIO che ricevono i nodi 1 e 5 contente il valore di rank reale del nodo 4 (Figura 5a). Nella figura 5b invece in verde è stato evidenziato il DAG del nodo 3 in cui è possibile osservare la riuscita dell’attacco e in viola i DAG dei nodi che ricevono il valore di rank corretto (Le frecce indicano la riga del DAG riferita al nodo malevolo 4).

Figura 5a: Attacco base – Messaggi DIO

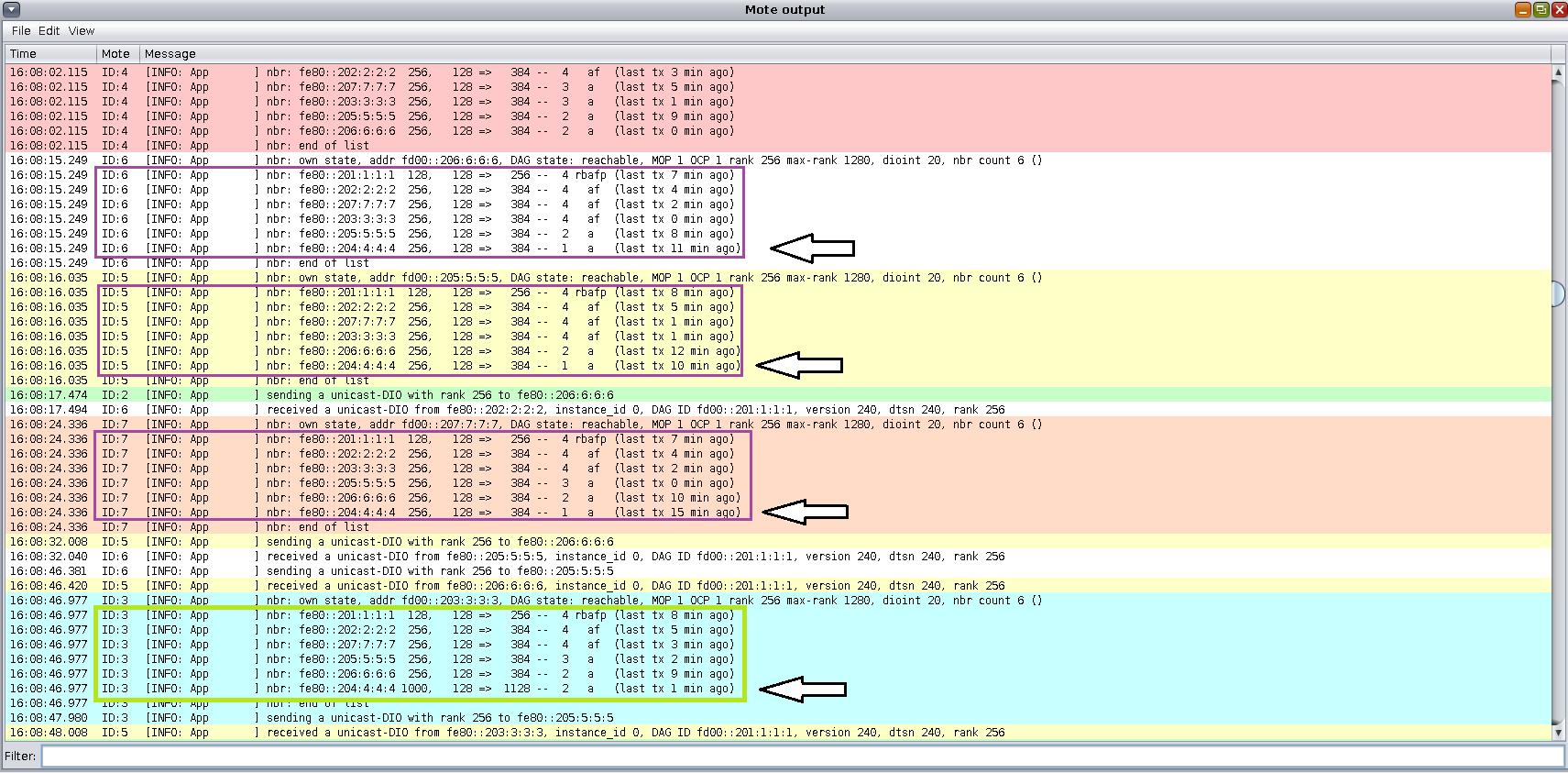


Figura 5b: Attacco base – DAG

Per implementare tale attacco è stato inserito il seguente codice all’interno del file rpl-icmp6.c nell’implementazione del metodo rpl\_icmp6\_dio\_output.

char \*vit="fe80::203:3:3:3"; // Nodo Vittima

uip\_ipaddr\_t ip;

uiplib\_ip6addrconv(vit, &ip);

if(node\_id == 4){ // Nodo attaccante

if(uip\_ip6addr\_cmp(&ip, addr))

curr\_instance.dag.rank=1000; //rank poisoning

}

**Mitigazione**

Esistono diverse tecniche per mitigare tale tipologia di attacco, nel nostro caso di studio abbiamo deciso di adottare una tecnica di analisi statistica basata sul valore di rank dei nodi vicini. L’idea alla base di tale tecnica risiede nella convinzione che generalmente il valore di rank di un nodo, non può essere eccessivamente diverso rispetto a quello degli altri vicini. Il nostro obiettivo, quindi, è stabilire una soglia e considerare corretto il valore del rank solo se inferiore a tale soglia. In [4] viene proposto di calcolare tale valore di soglia nel seguente modo:

Dove:

* **MaxRank:** è il massimo valore di rank tra i vicini del nodo corrente.
* **N**: è il numero di vicini del nodo corrente.
* **K**: è una costante da determinare in modo sperimentale in funzione della topologia della rete.
* **RankMed**: è il rank medio dei vicini.

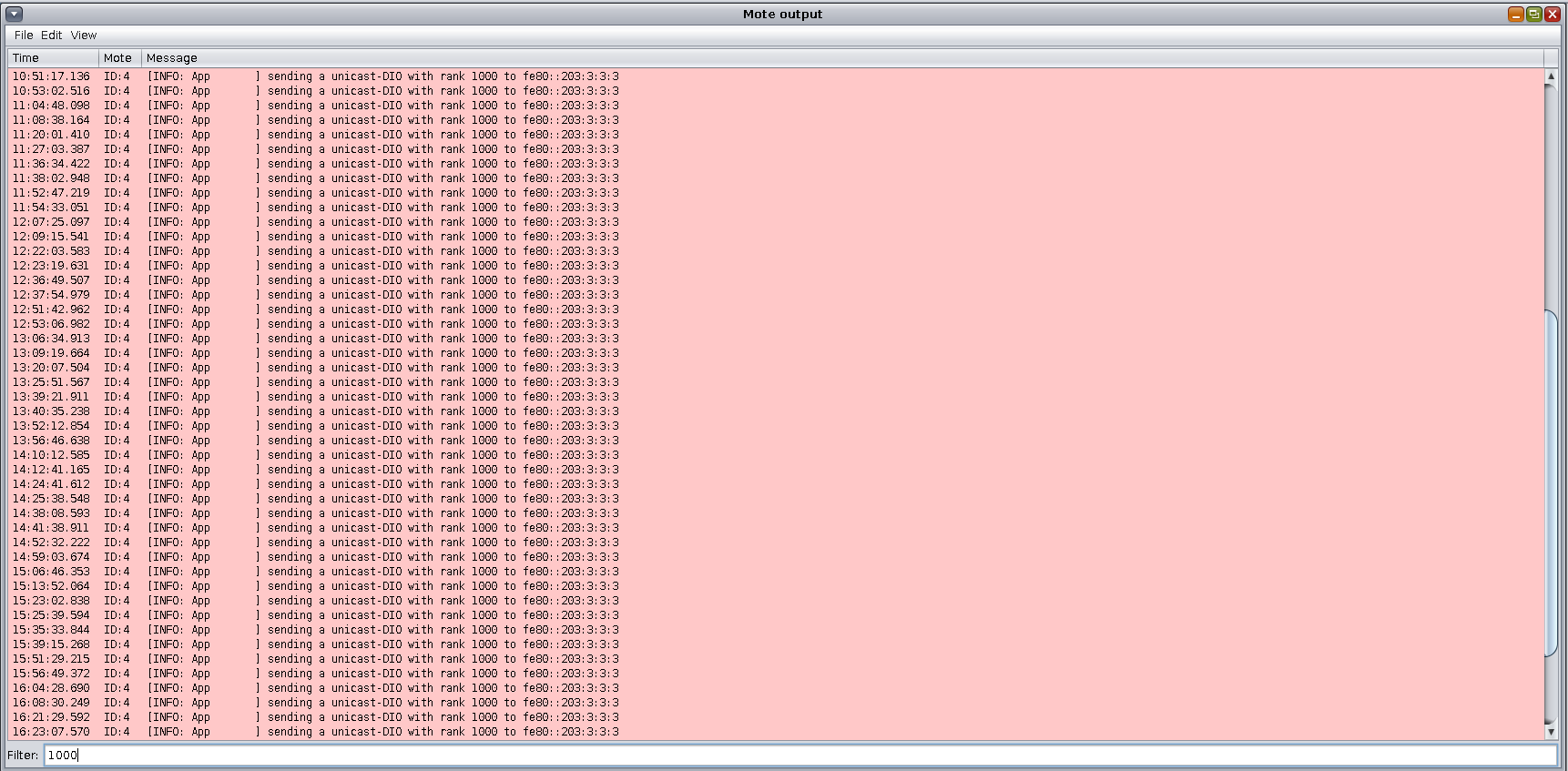
La figura 5c mostra il log, in cui sono state filtrate tutte le righe che hanno valore 1000, come è possibile osservare il nodo 4 continua ad inviare alla vittima (3), i messaggi DIO con rank fasullo, ma il nodo 3 non dice di averli ricevuti perché vengono scartati e non elaborati. La conferma del corretto funzionamento della mitigazione lo si ha nella figura 5d in cui viene mostrato il DAG del nodo vittima, il quale associa al nodo 4 il valore di rank corretto propagato dai suoi vicini.

Figura 5c: Mitigazione Rank Attack log

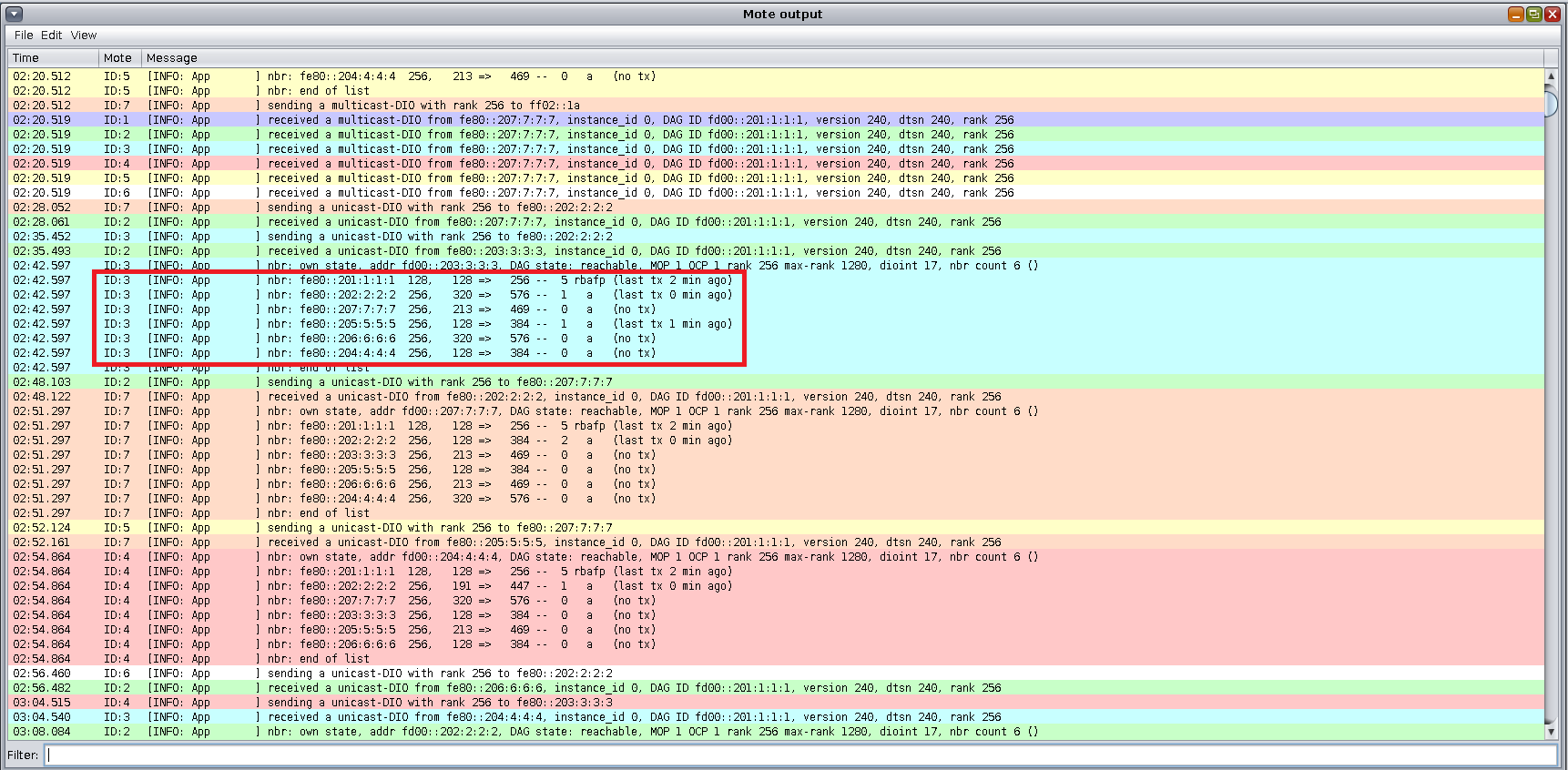


Figura 5d: Mitigazione Rank Attack – DAG nodo vittima

Di seguito il codice necessario per implementare la mitigazione. Il file di riferimento è sempre rpl-icmp6.c, mentre il metodo è dio\_input.

rpl\_nbr\_t \*nbr =rpl\_parent\_get\_from\_ipaddr(&UIP\_IP\_BUF->srcipaddr); //nodi vicini

rpl\_rank\_t somma=0, max=0;

int count=0;

while (nbr!= NULL){

rpl\_rank\_t t= rpl\_neighbor\_rank\_via\_nbr(nbr);

somma =somma+t;

count++;

if(t>max)

max=t;

nbr = nbr\_table\_next(rpl\_neighbors, nbr);

}

if(count!=0){

rpl\_rank\_t med= (rpl\_rank\_t) somma/count;

rpl\_rank\_t soglia= (rpl\_rank\_t) max \* (1/count - K) + med; // K = 0.3

if(dio.rank >= soglia){

goto discard;

}

}

Il file e/o metodo all’interno del quale sono implementati attacchi e mitigazioni del rank è comune a tutti e tre gli scenari rappresentati. Si eviterà quindi di riportalo nuovamente negli scenari successivi.

***Variante 2***

In questa seconda variante dell’attacco, affrontiamo uno scenario più complesso rispetto al precedente, troviamo infatti, due nodi malevoli (ID= 4 e 9) che inviano dai rank fasulli a tutti gli altri nodi della rete, con lo scopo di propagare il poisoning a tutta la rete. In questo scenario, al contrario del caso precedente, applichiamo un approccio graduale per simulare meglio un ambiente reale. I due nodi, infatti, entrano nella rete come nodi legittimi che inviano valore di rank corretto e solo dopo un certo periodo di tempo iniziano ad effettuare il poisoning.

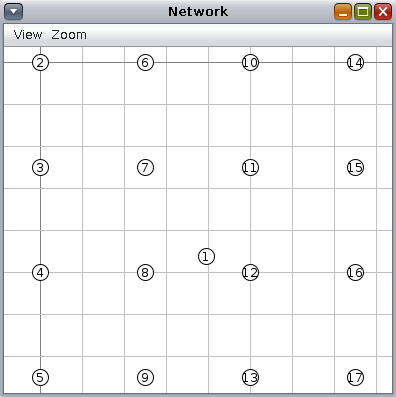


Figura 6a: Topologia rete

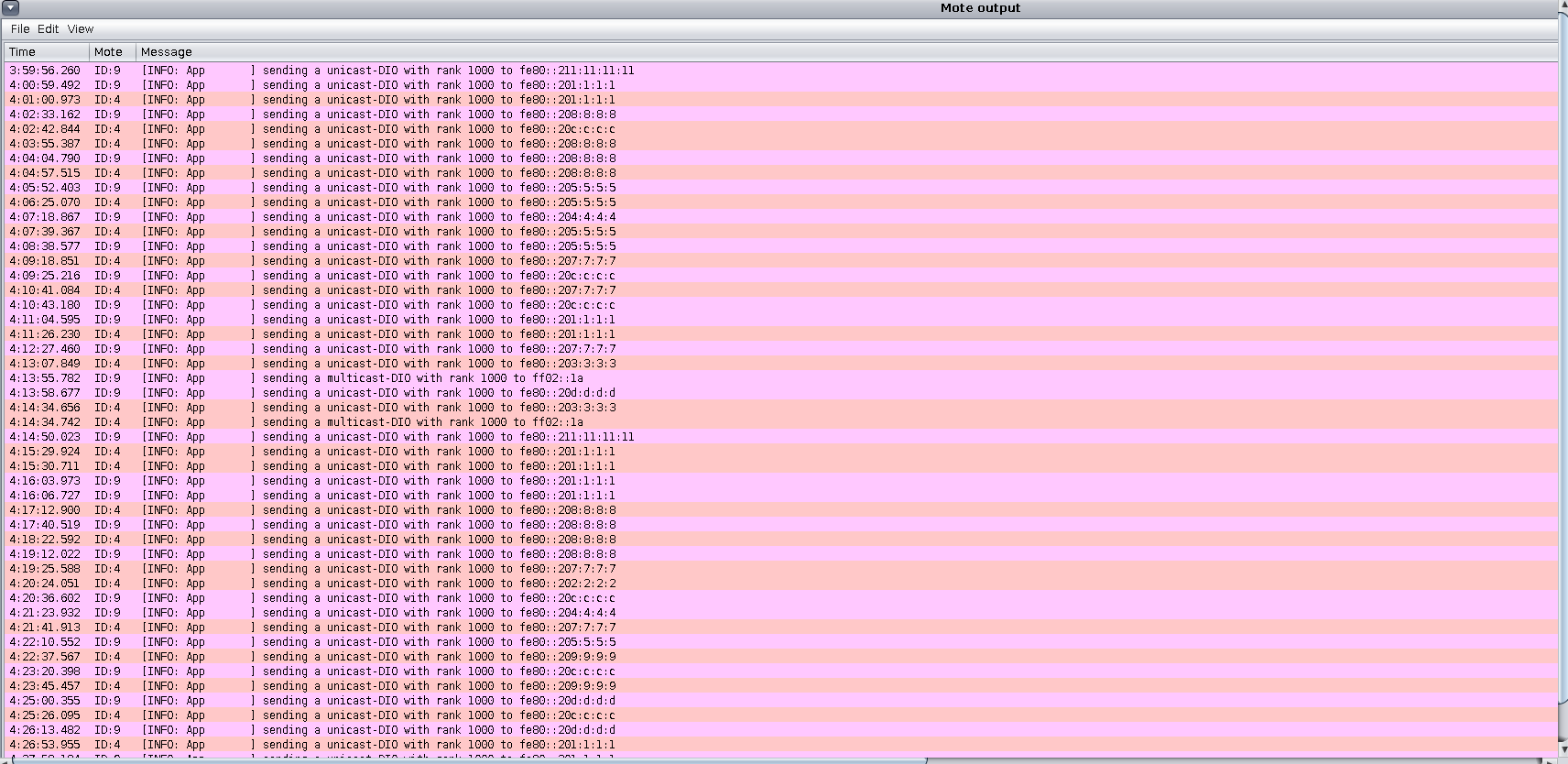


Figura 6b: Rank Attack - Log

In figura 6b si può osservare come tutti i messaggi DIO inviati dai nodi 4 e 9 contengano un rank falso.

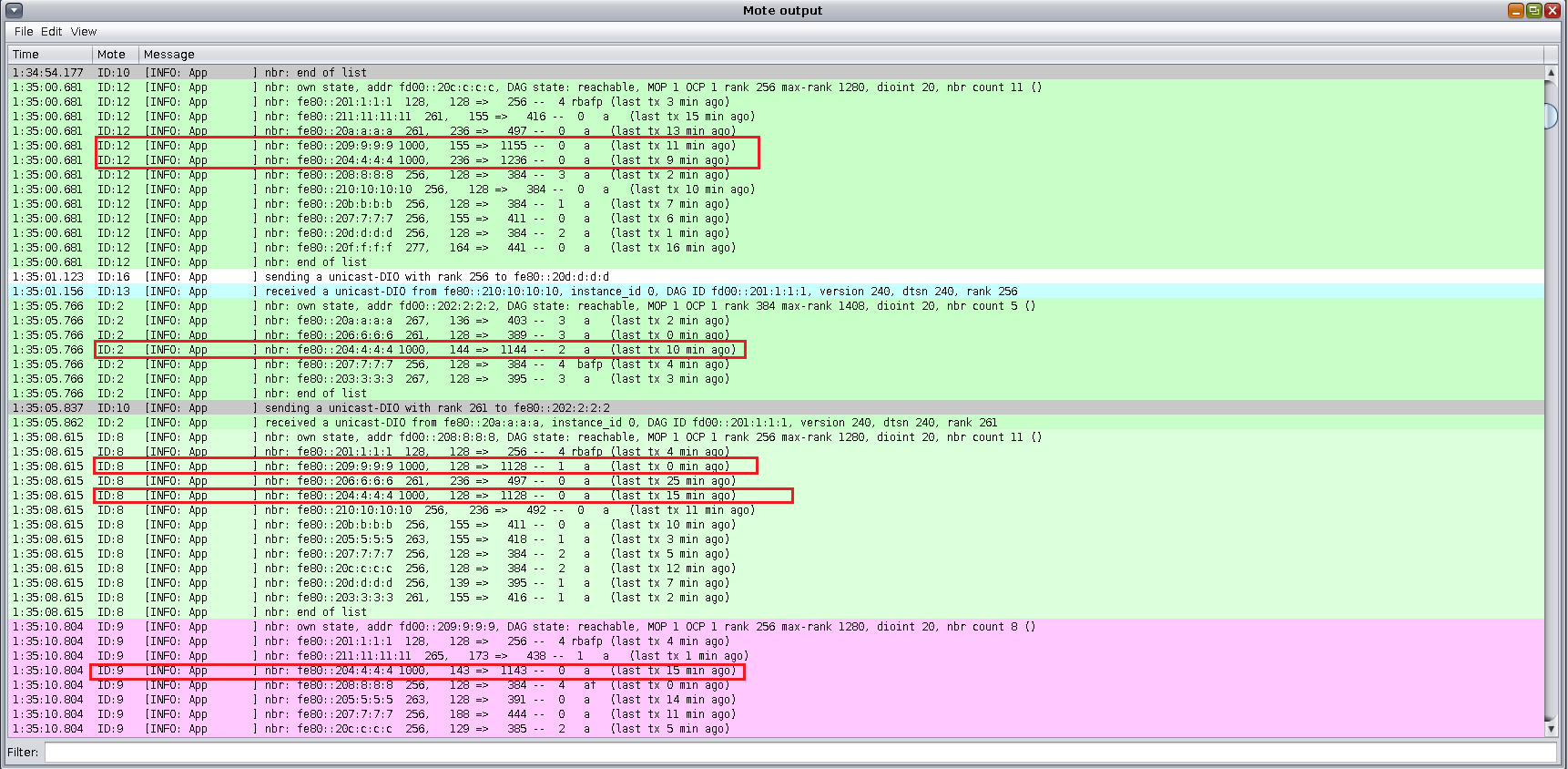


Figura 6c: Rank attack DAG

Nella figura 6c è possibile osservare come l’attacco vada a buon fine. Sono state evidenziate in rosso le righe dei DAG dei diversi nodi alterate dall’attacco.

Di seguito il codice necessario per l’implementazione dell’attacco.

if(var <=0){//var inizializzato a 50. Simula attesa prima dell’attacco

if(node\_id == 4 || node\_id == 9){ // attaccanti

curr\_instance.dag.rank=1000; //rank poisoning

}} else{

var--;

}

***Mitigazione***

La mitigazione è comune a quella descritta nello scenario precedente.

Nella figura 6d è possibile osservare il corretto funzionamento della mitigazione, infatti nei DAG dei diversi nodi, non compare mai il valore di rank 1000, ma è presenta il valore reale che i nodi malevoli hanno iniviato nella prima fase.

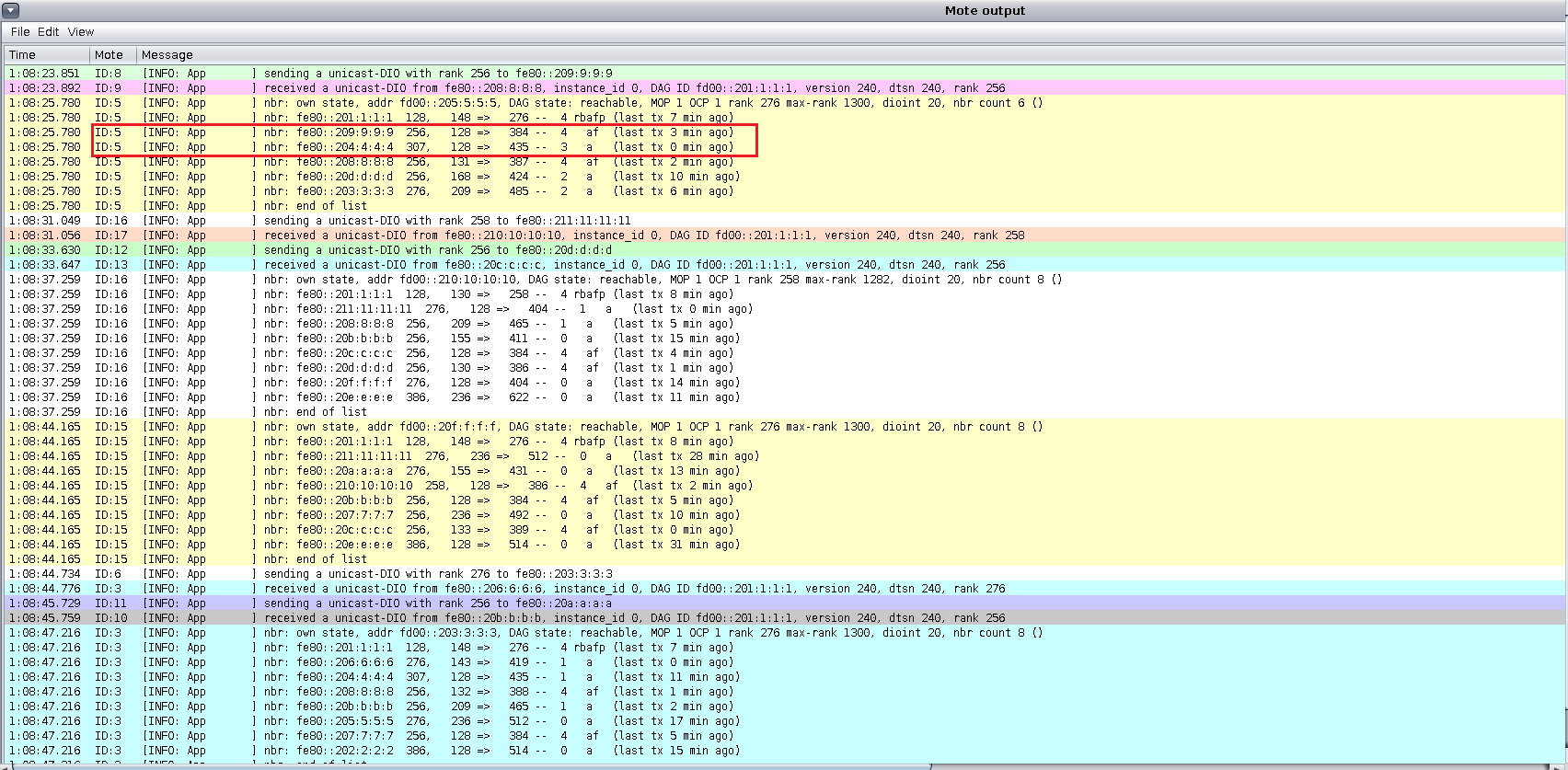


Figura 6d: Mitigazione rank attack

***Variante 3***

In quest’ultimo scenario, pur mantenendo la topologia mostrata nella figura 6a, si mostra un utilizzo alternativo del rank attack rispetto al semplice poisoning visto nei casi precedenti. Il rank attack può infatti essere utilizzato come attacco di apertura per ulteriori attacchi come ad esempio un man in the middle, che ha sua volta può portare ad molti altri scenari come drop dei pacchetti o lo sniffing. Di seguito vogliamo utilizzare il rank attack per realizzare un man in the middle, quindi vogliamo far in modo che il nodo vittima scelga il nodo malevolo come suo padre nel DAG, in questo modo quando la vittima deve inoltrare un pacchetto sulla rete, lo farà attraverso il nodo malevolo che quindi potrà controllare tale pacchetto. Per implementare ciò è necessario che il nodo malevolo effettui un rank poisoning nei confronti della vittima, inserendo all’interno del messaggio DIO un valore di rank ottimale, quindi più basso, rispetto a quello di tutti i vicini del nodo vittima.

In questo scenario abbiamo due nodi malevoli i nodi 4 e 15. In figura 7a possiamo vedere come come il poisoning riesca correttamente ed in più è possibile vedere come i nodi vittima scelgano i nodi malevoli come loro padri, attraverso la stringa **‘*bafp’*** riportata nelle righe dei DAG in riferimento ai nodi malevoli.

La stringa *‘bafp’* va interpretata nel seguente modo:

* **b**: Best rank. Indica che il nodo ha il miglior valore di rank rispetto agli altri vicini.
* **a**: Il rank è accettabile (valido) e il nodo è un possibile candidato ad essere scelto come genitore nel DAG.
* **f**: Indica che il collegamento è aggiornato.
* **p**: Indica che il nodo è stato scelto come padre nel DAG.

La presenza quindi dei due caratteri *b* e *p* nei DAG dei nodi vittima 14 e 2, mostrano come l’attacco sia riuscito correttamente. In particolare, si puo osservare come il nodo 2 scelga il nodo malevolo 4 come suo padre nel DAG, e come il nodo 14 scelga il nodo malevolo 15.

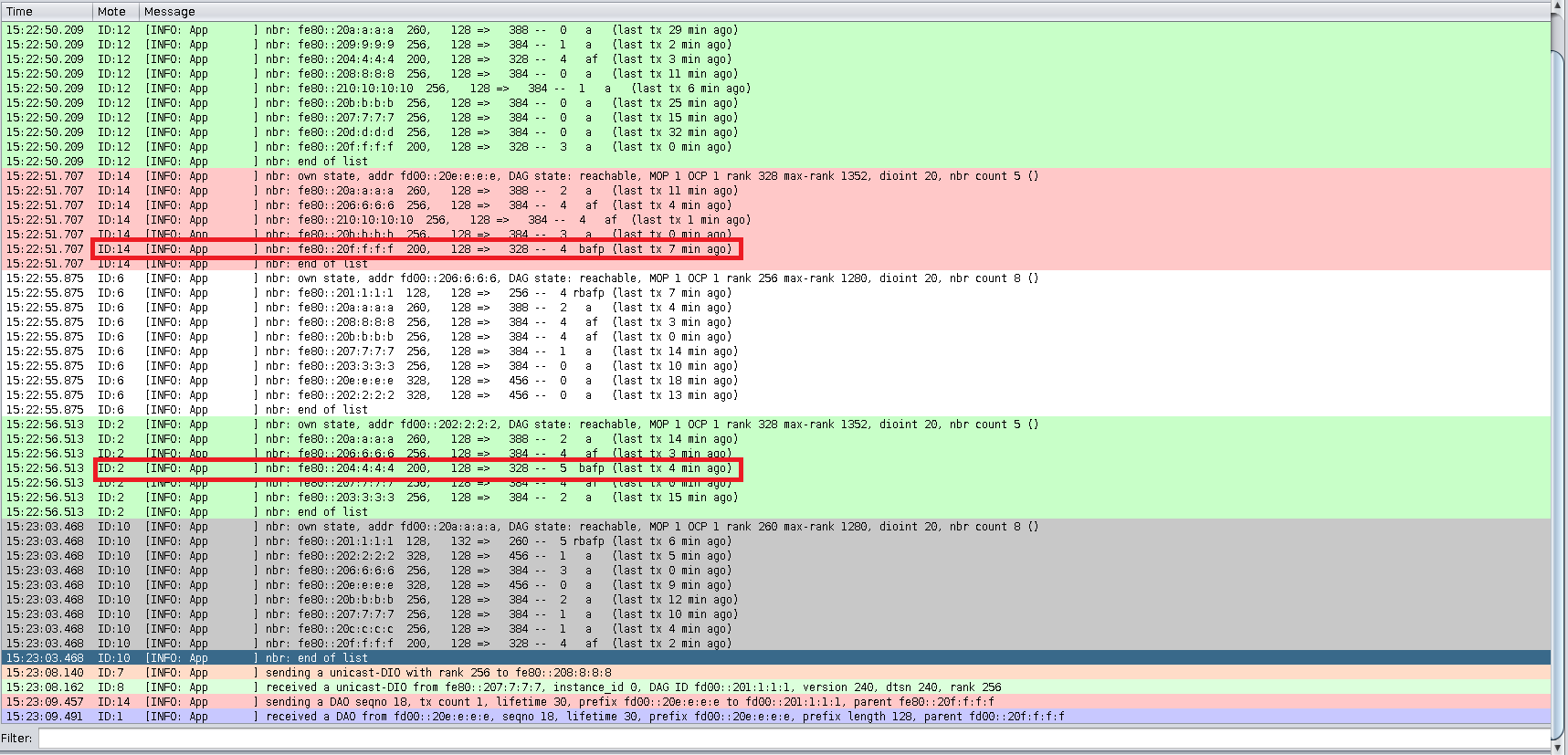


Figura 7a: Rank Attack – DAG

Di seguito il codice necessario per l’implementazione di tale attacco.

if(var <=0){ //Inizializzato a 50. Simula attesa prima dell’attacco

if(node\_id == 4 || node\_id == 15){ // attaccanti

curr\_instance.dag.rank=200; //rank poisoning

}} else{

var--;

}

***Mitigazione***

La mitigazione è comune ai due casi trattati in precedenza, l’unica differenza è che la soglia in questo caso essa funge da lower bound, rispetto al ruolo di upper bound ricoperta nei due casi precedenti. La formula per il calcolo della soglia viene modificata nel seguente modo:

I cui i vari termini hanno lo steso significato del caso precedente, ad eccezione di MinRank che rappresenta il valore di rank minino tra i vicini del nodo in considerazione.

La figura 7b mostra come la mitigazione funzioni correttamente, si può osservare infatti, come nel dag del nodo vittima 2 non sia più presente la stringa bafp, si riconosce dunque il nodo 4 come attaccante e si impedisce l’attacco man in the middle.

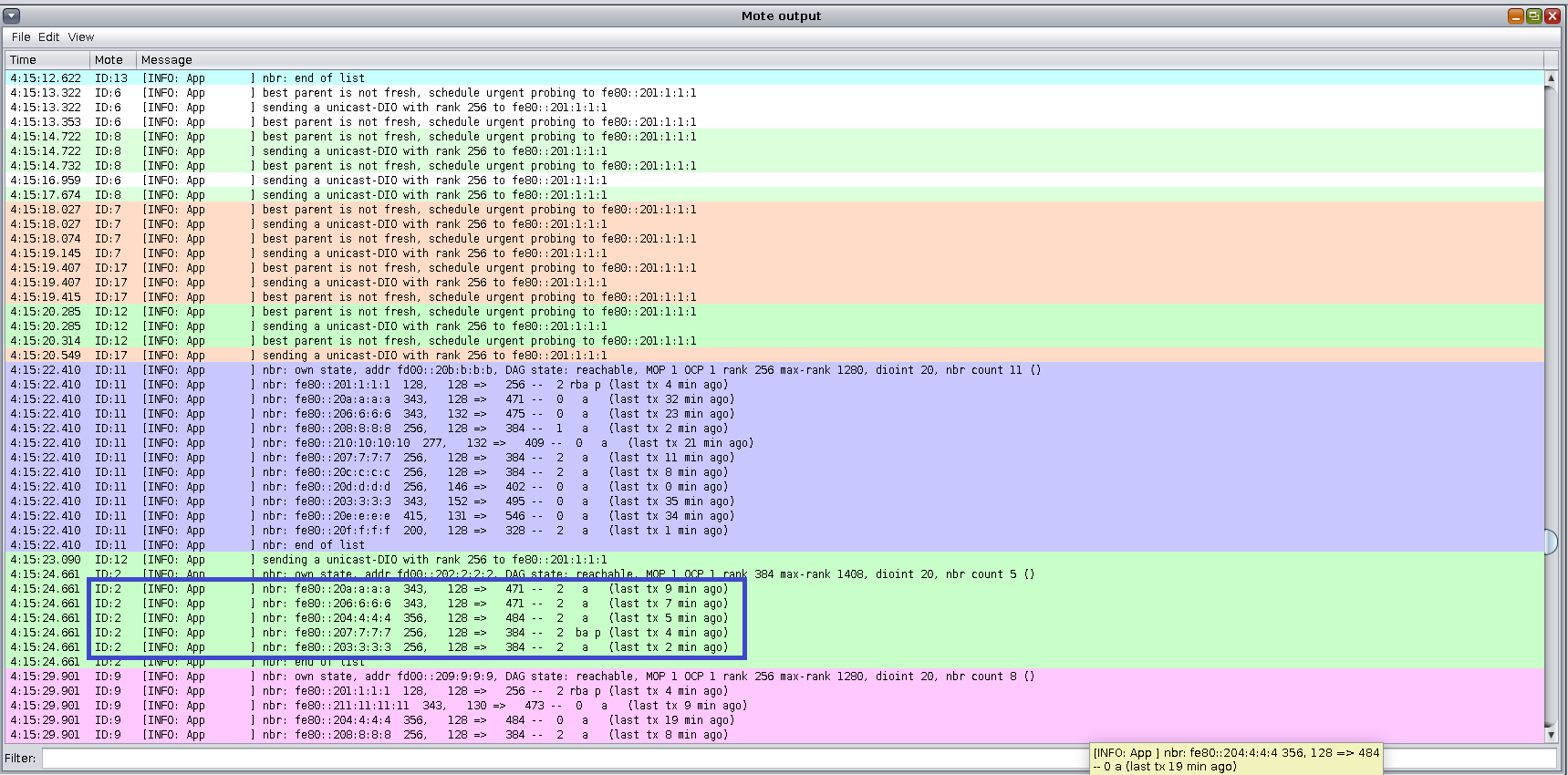


Figura 7b: Mitigazione Rank

Di seguito il codice necessario all’implementazione della mitigazione.

rpl\_nbr\_t \*nbr =rpl\_parent\_get\_from\_ipaddr(&UIP\_IP\_BUF->srcipaddr);

rpl\_rank\_t somma=0, min=10000;

int count=0;

while (nbr!= NULL){

rpl\_rank\_t t= rpl\_neighbor\_rank\_via\_nbr(nbr);

somma =somma+t;

count++;

if(t<min)

min=t;

nbr = nbr\_table\_next(rpl\_neighbors, nbr);

}

if(count!=0){

rpl\_rank\_t med= (rpl\_rank\_t) somma/count;

rpl\_rank\_t soglia= (rpl\_rank\_t) (med - (min \* (1/count - K)));

if(dio.rank <= soglia){

goto discard;

}

}

Il codice sorgente e le simulazioni avviabili in cooja di questo attacco sono presenti all’interno della cartella rank.

***Wormhole Attack***

L'attacco wormhole è uno degli attacchi più comuni e severi che vengono effettuati nelle reti RPL. È basato sulla presenza di due nodi malevoli, M1 e M2, i quali si adoperano per creare un tunnel tra nodi appartenenti alla rete, A e B, e distanti tra loro; la presenza di questo tunnel punta ad ingannare i due nodi A e B facendo credere loro di essere più vicini di quanto non lo siano in realtà. Ciò mira a modificare il normale transito dei pacchetti all'interno della rete; prendendo in considerazione la topologia mostrata nella *Figura 1*, il wormhole cerca di ingannare il nodo B facendogli credere di essere vicino al nodo radice A; questo porta il nodo B a scegliere come best parent la radice anziché il nodo N.

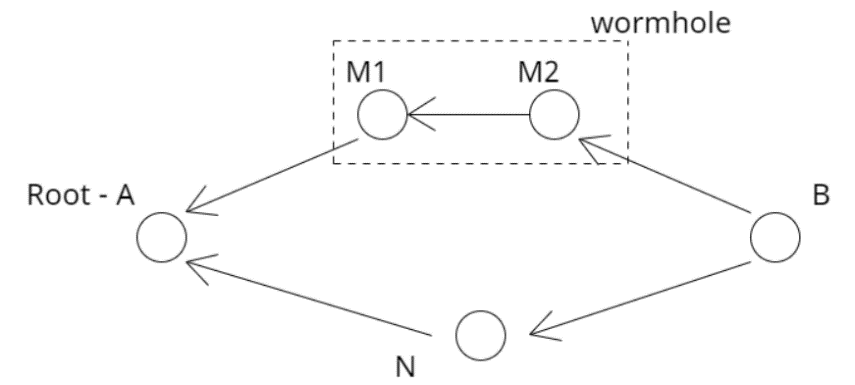


Figura 2

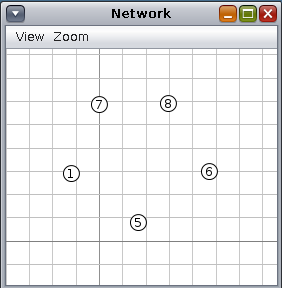
Per questo attacco sono state eseguite quattro simulazioni. La prima è stata eseguita a puro scopo esplicativo prendendo come riferimento una semplice rete, mostrata nella *Figura 2*, composta da 5 sensori: un nodo radice contrassegnato dall'ID 1 e quattro nodi client con IDs 5, 6, 7 e 8:

Figura 3

|  |  |
| --- | --- |
| Id Nodo | Id Best Parent |
| 5 | 1 |
| 6 | 5 |
| 7 | 1 |
| 8 | 7 |

In questa simulazione i nodi client provvedono ad inviare, ogni minuto, un pacchetto alla radice facendolo passare per il loro best parent. Di seguito vengono mostrati gli output dei nodi:

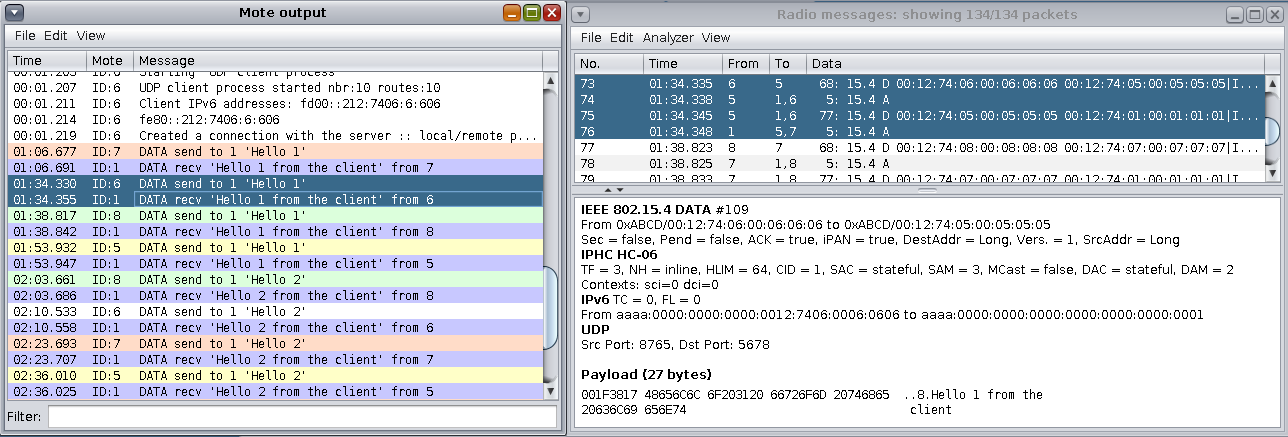


Figura 4a Figura 3b

In particolare, si vuole porre l’attenzione sul nodo con ID 6; in questo caso, il suo best parent, come già anticipato, risulta essere il nodo con ID 5. Il nodo 6, quindi, nel tentativo di recapitare il pacchetto alla radice lo trasmette inizialmente al nodo 5 il quale risponde con un ack; successivamente, il pacchetto viene recapitato direttamente al server; ciò si evince dalla serie di pacchetti, mostrati nella *Figura 3b,* che vanno dal numero 73 al numero 76 e dall’output del nodo server (*Figura 3a*) che conferma di aver ricevuto il pacchetto del nodo 6.

Viene considerata, adesso, una variante della precedente topologia in cui i due nodi contrassegnati dagli IDs 7 e 8 assumono ruolo di nodi malevoli e vengono incaricati di eseguire l'attacco wormhole; i nodi client, come sempre, provvedono ad inviare, ogni minuto, un pacchetto alla radice facendolo passare dal loro best parent. Riponendo l’attenzione sul nodo con ID 6 si evince come il suo best parent sia cambiato; infatti, a causa del wormhole creato dai nodi 7 e 8 il nodo 6 sceglie come best parent proprio il nodo 8 anziché il nodo 5:

Immagine che contiene testo

Descrizione generata automaticamente

Figura 5

Immagine che contiene testo

Descrizione generata automaticamente

Figura 6

Nel dettaglio, il nodo 7 si adopera per trasmettere il rank estrapolato dal messaggio dio della radice al nodo 8 che, a sua volta, lo trasmette al nodo con ID 6. L'effetto del wormhole è quello di far credere al nodo 6 di essere vicino alla radice; a questo punto il nodo 6 sceglie come best parent il nodo malevolo con ID 8 anziché il nodo 5. Il nodo 6, quindi, invia normalmente i suoi pacchetti alla radice facendoli passare, inconsapevolmente, per il wormhole.

Il nodo 8 riceve il pacchetto dal nodo 6 e risponde inviandogli un ACK ed illudendolo, così, di aver preso in carico il suo pacchetto; a questo punto il wormhole esegue il drop del pacchetto del nodo 6 il quale è del tutto ignaro dell’accaduto [2]. La radice, quindi, a causa dei drops eseguiti dal wormhole, non riceve nessuno dei pacchetti del nodo 6; ciò si evince dagli output dei nodi mostrati nella *Figura 5*.

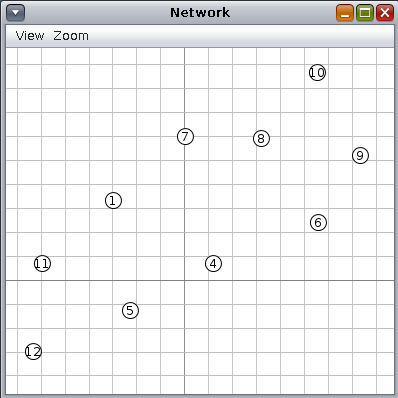
Di seguito viene mostrato il codice, inserito nel file rpl-icmp6.c, eseguito del nodo 8 per ingannare il nodo 6:

|  |
| --- |
| […]  /\* Attacco -------------------------------------------------------------\*/  int node\_id\_attaccante = 8;  /\* il rank della radice viene ricevuto dal nodo 7 \*/  int rank\_dio\_radice = ROOT\_RANK(instance);  /\* modifica del dio che il nodo 8 invierà ai suoi nodi vicini. \*/  /\* il wormhole effettuerà il replay del dio creato dalla radice, \*/  /\* in questo modo, nodi lontani crederanno di essere vicini alla radice \*/  if (node\_id == node\_id\_attaccante) {  /\* modifica del rank del dio che verrà inviato \*/  dag->rank = rank\_dio\_radice;  set16(buffer, pos, dag->rank);  /\* invio del dio ai nodi vicini \*/  goto finalize\_and\_send\_dio;  }  /\* Attacco -------------------------------------------------------------\*/  […] |

Per le simulazioni vere e proprie dell’attacco sono state prese in considerazione tre topologie composte, rispettivamente, da 10, 50 e 100 nodi; in ogni simulazione, così come nell’esempio, ogni nodo client aveva il compito di inviare alla radice un pacchetto ogni minuto; la durata di ogni simulazione è stata impostata a 10 minuti.

L'impatto dell'attacco è stato misurato, per ogni simulazione, in termini di pacchetti persi e di *wormholed* *nodes*, definiti come il numero di nodi che inviano pacchetti che passano per il wormhole.

Di seguito vengono mostrate le topologie, denominate: small (10 nodi), medium (50 nodi) e large (100 nodi):

Immagine che contiene tavolo

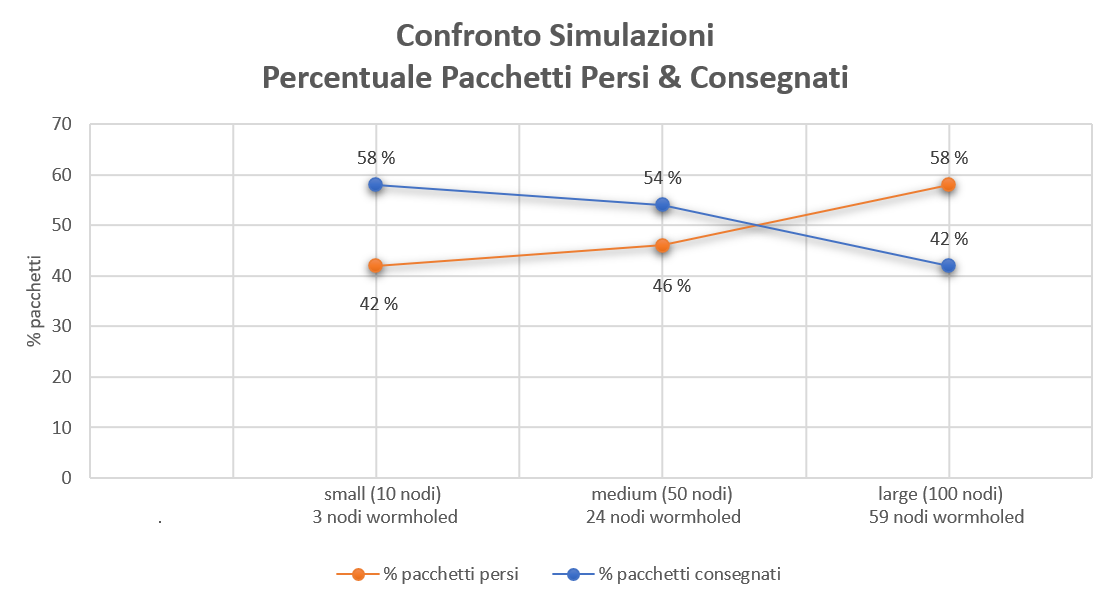
Descrizione generata automaticamenteImmagine che contiene tavolo

Descrizione generata automaticamente

Figura 7

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| topologia | nodi | wormholed nodes | % pacchetti persi | % pacchetti consegnati |
| small | 10 | 3 | 42 % | 58 % |
| medium | 50 | 24 | 46 % | 54 % |
| large | 100 | 59 | 58 % | 42 % |

Da questi dati si evince come all’aumentare dei wormholed nodes aumenti anche la percentuale dei pacchetti persi e diminuisca la percentuale di pacchetti consegnati: [2]



La mitigazione attuata come contromisura per questo attacco è stata quella basata sulla conoscenza delle posizioni dei nodi ritenuti sicuri; questa soluzione risulta essere economica ed efficiente quando si ha a che fare con nodi caratterizzati da posizioni fisse.

L’implementazione della mitigazione è stata eseguita creando una whitelist di nodi ritenuti sicuri; ogni nodo, quindi, accettava messaggi dio solo dai nodi contenuti in questa whitelist. Dal punto di vista del codice, la mitigazione è stata inserita nel file rpl-dag.c.

Immagine che contiene testo

Descrizione generata automaticamente

Figura 8

Come mostrato dagli output dei nodi (*Figura 7*), in cui viene considerata la stessa topologia rappresentata in *Figura 2*, nonostante il tentativo di attacco dei nodi 7 e 8, il nodo 6 sceglie come best parent il nodo con ID 5 e il pacchetto viene correttamente ricevuto dal nodo radice.

**Altri attacchi...**

***Conclusioni***

Con il seguente elaborato sono stati implementati i seguenti attacchi: DoS Attack, Rank Attack, Fragment Duplication Attack e Warmhole Attack; in un ambiente IoT simulato attraverso l’uso del sistema operativo Contiki-NG e del simulatore Cooja. È stata fornita una breve trattazione dei principali protocolli trattati, evidenziando la loro importanza ed il ruolo assunto nei diversi scenari applicativi in riferimento sia ad un ambiente sicuro che ad un ambiente esposto a rischi. È stata descritta inoltre la procedura di installazione e configurazione dell’ambiente di sviluppo. Per concludere l’elaborato è bene fare presente ad eventuali assunzioni effettuate riguardo l’implementazione delle soluzioni proposte e l’ambiente di sviluppo:

* Per quanto riguarda l’attacco DoS, la mitigazione assume che la rete, intesa in numero di nodi, rimanga costante. Quest’assunzione è dovuta dall’impossibilità di passare argomenti al simulatore a tempo di esecuzione. Quest’assunzione non mira però in alcun modo l’efficacia o la generalità della soluzione proposta, in un ambiente reale, infatti, è sufficiente sostituire le allocazioni di memoria statiche, con delle allocazioni dinamiche (esempio usando la funzione malloc) in modo da poter allocare nuovo spazio, per memorizzare le informazioni suoi nuovi nodi, qualora fosse necessario.
* Sempre in riferimento al DDoS con mitigazione, è bene precisare che sono state effettuate simulazioni anche su reti più complesse, formate da 30+ nodi con diversi attaccanti e diverse vittime. Sebbene il tutto continui a funzionare correttamente, è stato scelto di non presentare tale scenario. L’elevato numero di messaggi scambiati, infatti, rende poco chiaro il contesto e richiede un’attenta analisi del log completo, che per questioni di leggibilità, si è deciso di non allegare vista la sua eccessiva lunghezza.
* Il rank attack può essere utilizzato come vettore di partenza per la costruzione di altri attacchi, come ad esempio il Sinkhole o il DoS. Analizzando quest’ultimo esempio, infatti, un nodo malevolo combinando Rank Attack e IP Spoofing, potrebbe inviare pacchetti DIO sulla rete, contenenti un valore di rank ottimale e l’indirizzo IP del nodo vittima. In tal modo gli altri nodi della rete sceglieranno il nodo vittima come nodo di inoltro dei pacchetti e questo potrebbe causare un DoS su tale nodo, se il numero di pacchetti è eccessivamente elevato.
* Il wormhole attack risulta essere molto potente e può essere eseguito anche in reti che fanno uso di tecniche crittografiche in quanto è un attacco che non genera nuovi pacchetti ma si limita ad effettuarne il replay; quindi, l’attacco, permette di effettuare un denial of service senza la necessità di rubare chiavi private. Per quanto concerne la mitigazione, esistono molte contromisure per mitigare l’attacco ma nessuna di queste sembra essere definitiva. La contromisura scelta per questo progetto, basata sulla conoscenza delle posizioni dei nodi ritenuti sicuri, risulta essere praticabile e non dispendiosa per dispositivi “fissi” ma risulta inefficace nel caso in cui i nodi della rete siano dispositivi mobili.

Aggiungere descrizione script avvio…..

**Simulazioni Wormhole Attack**

Per poter avviare le simulazioni relative al wormhole attack è necessario avviare *contiker* e poi *cooja* dal terminale.

Una volta avviata la GUI è necessario caricare la simulazione da “File” > “Open simulation” > “Browse”;

Da qui bisogna poi dirigersi nella cartella “contiki-ng” > “wormhole” e scegliere la simulazione da avviare entrando nella corrispondente cartella e aprendo il file “simulation.csc”:

Immagine che contiene testo

Descrizione generata automaticamente

**Bibliografia**

[1] K. K. Rai and K. Asawa, "Impact analysis of rank attack with spoofed IP on routing in 6LoWPAN network," 2017 Tenth International Conference on Contemporary Computing (IC3), 2017, pp. 1-5, doi: 10.1109/IC3.2017.8284340.

[2] P. Perazzo, C. Vallati, D. Varano, G. Anastasi and G. Dini, "Implementation of a wormhole attack against a rpl network: Challenges and effects," 2018 14th Annual Conference on Wireless On-demand Network Systems and Services (WONS), 2018, pp. 95-102, doi: 10.23919/WONS.2018.8311669.

[3] Hummen, René & Hiller, Jens & Wirtz, Hanno & Henze, Martin & Shafagh, Hossein & Wehrle, Klaus. (2013). 6LoWPAN Fragmentation Attacks and Mitigation Mechanisms. 10.1145/2462096.2462107.

[4] Iuchi, Kenji & Takumi, Matsunaga & Toyoda, Kentaroh & Sasase, I.. (2015). Secure Parent Node Selection Scheme in Route Construction to Exclude Attacking Nodes From RPL Network. IEICE Communications Express. 4. 10.1587/comex.4.340.