

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*

*3.2 Rank Attack*

*3.3 Warmhole Attack*

*3.4 Fragment Duplication Attack*

*4 Conclusioni?*

***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 (DODAG). 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. Inoltre, è presente uno script Bash che permette di preparate tutti i file necessari e pulire l’ambiente prima di caricare una simulazione.

***Attacchi***

In questa sezione andremo a descrivere gli 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, una prima variante abbiamo una topologia semplice, con un solo nodo attaccante ed un solo nodo vittima, mentre nella seconda variante abbiamo implementato una topologia più complessa dove abbiamo diversi nodi attacchi (per simulare un DdoS) e diversi nodi vittima?.

Per l’implementazione dell’attacco, è stato modificato il codice del seguente file: …… rpl-udp/client.c

Caso 1

In particolare, col nodo 3, si vuole simulare il comportamento di un nodo malevolo che inonda di richieste il server identificato da id=1 e da IP= …:1:1:1

Per fare ciò è stata introdotta la seguente modifica al file quello di sopra

Codice dos ttacco

La figura x mostra lo scenriao di attacco descritto, in cui è possibile vedere come il nodo 3 inonda di messaggi il server UDP, il quale non riesce a rispondere correttamente alle richieste legittime.

Mitigazione

Per mitigare l’attacco costruiamo una sorta di allowlist basata sull’analisi sulla frequenza di ricezione dei pacchetti da parte del server. Per fare ciò abbiamo bisogno di un Thread che calcola la frequenza di ricezione, e di una funzione che permetta di elaborare solo le richieste leggittime, ovvero le richieste provenienti da nodi che hanno un rate di invio inferiore rispetto alla soglia specificata. Il valore di soglia è stato ricavato in modo empirico cercando di ottimizzare il trade off tra identificazione repentina del attacco e assenza di falsi positivi. Fissando una soglia troppo bassa infatti anche i nodi leggittimi verrebbero bloccati, viceversa il nodo vittima potrebbe accorgersi troppo tardi dell’attacco e potrebbe iniziare a scartare richieste leggittime prima di accorgesi dell’attacco.

Di seguito riportiamo il codice relativo alla mitigazione

Vista l’impossibilità di passare dei parametri alle classi in fase di invocazione, si è assunta una topologia statica, con un numero di nodi prefissato. È bene osservare che tale assunzione non è restrittiva e non perde di generalità, in un ambiente dinamico, in cui è possibile individuare e aggiornare il numero di nodi a tempo di esecuzione è sufficiente sostituire le allocazioni di memoria statiche, con delle allocazioni dinamiche (ricorrendo alla funzione malloc, ad esempio) ed il tutto continua a funzionare correttamente.

Codice thread

Codice mitigazione vero e proprio

Scenario complesso

In questo secondo scenario possiamo individuare una topologia più complessa in cui ritroviamo diversi nodi atatccanti e diverse vittime per simulare un attacco DdoS (distribuited Denial of Service). Le figure sequenti mostrato lo scenario di attacco con una topologia complessa.

Foto topologia complessa

Con con mitigazione piccolo commento

Rank Attack

L’introduzione del concetto di rank all’interno del protocollo RPL, risulta utile per diversi scenari come l’ottimizzazione dei percorsi, la prevenzione di cicli, ma può costituitre anche un vettore di attacco, che può incidere sulla performace della rete.

In questa tipologia di attacco, l’attaccante modifica volonatariamente il proprio rank, manipolando cosi la rete. Anche in questo caso sono state presentate due topologie diverse una più semplice in cui si mostra un solo nodo attaccante e una sola vittima, e un secondo scenario in cui abbiamo diversi