Relazione AppDetection
Progetto per il corso di Gestione Di Rete A.A 2018/2019

Samuel Fabrizi (Matricola 550097)

Indice

Chapter 1

Introduzione

AppDetection è un'applicazione utilizzata per monitorare e classificare il traffico di rete a livello IP, in base all'applicazione di provenienza. Inoltre, per poter analizzare il traffico di diversi device connessi alla medesima rete, è possibile effettuare un attacco *man in the middle* (MITM) al fine di interporsi tra un insieme di host ed il gateway.

Chapter 2

Struttura del progetto

Il progetto è strutturato in classi, ognuna con una determinata funzionalità. Di seguito verrà brevemente illustrata la struttura delle principali classi ed il loro utilizzo.

2.1 Manager

La classe *Manager* rappresenta il nucleo dell'applicazione. Si preoccupa di effettuare il controllo degli argomenti del programma e di inizializzare strutture dati e thread necessari all'esecuzione dell'applicazione.
Dopo aver creato il *radix-tree* e la struttura dati contenente le statistiche del traffico di rete (vedi Sez. 2.5), crea e manda in esecuzione i thread delegati a:

- catturare i pacchetti su una data interfaccia (vedi Sez. 2.2);
- aggiornare l'interfaccia grafica (vedi Sez. 2.3);
- effettuare l'ARP poisoning (vedi Sez. 2.4), solo se richiesto dall'utente.

2.2 Sniffer

Tale classe definisce il thread delegato alla cattura dei pacchetti sull'interfaccia di rete fornita dall'utente. Lo *Sniffer* è anche il *main thread* dato che rappresenta il ciclo di vita principale dell'intera applicazione.
Per ogni pacchetto catturato viene invocata la funzione *compute_statistics* che, dopo aver estratto l'indrizzo IP sorgente del pacchetto, ricerca un match all'interno del *radix-tree* e, in caso di successo, aggiorna le statistiche.
Considerando che l'obiettivo dell'applicazione è di classificare il traffico a livello IP, tutti i pacchetti di livello inferiore verranno ignorati.

2.3 Drawer

Il thread *Drawer* si occupa dell'aggiornamento dell'interfaccia grafica che evidenzia il traffico catturato e classificato in base all'applicazione di provenienza. Tale interfaccia è composta da un grafico a barre che evidenzia la quantità di traffico catturata (in bytes) e divisa per applicazione.
Per poter rendere pulita e lineare la costruzione e l'aggiornamento di tale grafico ho utilizzato *Dash*, un framework per la costruzione di applicazioni web. Dopo aver avviato l'applicazione, è possibile visualizzare il grafico, aggiornato in tempo reale, all'indirizzo `http://127.0.0.1:8050/`.

2.4 Poisoner

L'attacco MITM, se richiesto dall'utente, è svolto dal thread *Poisoner*.
Dopo aver effettuato una serie di *ARP request* per ottenere il *MAC address* dei vari host e del gateway, inizia ad effettuare l'*ARP poisoning*.
L'*ARP poisoning* è una forma di attacco nella quale l'attaccante invia continue *ARP response* al fine di modificare la *ARP cache* dell'obiettivo. Operando tale tecnica è possibile interporsi tra i vari host e il gateway riuscendo a catturare il traffico di rete dei diversi device.

2.5 Statistics

La classe *Statistics* rappresenta la principale struttura dati. Contiene le statistiche ottenute dalla cattura del traffico di rete e i vari metodi utilizzati per aggiornare o reperire i dati contenuti in tale struttura.
Un oggetto instanza di questa classe inizializza un dizionario i cui elementi sono coppie chiave-valore dove:

- la chiave è il nome di un'applicazione;
- il valore è la somma del numero di bytes dei pacchetti catturati con un indirizzo IP appartenente ad una specifica applicazione.

Il *Manager* si preoccupa di instanziare tale struttura dati per poi passarla come riferimento al main thread *Sniffer* e al thread Drawer.
Al fine di evitare possibili perdite di pacchetti dovute alla sincronizzazione dell'accesso a tale struttura, ho preferito considerare la possibilità di eventuali errori ritenuti ammissibili ai fini del progetto.

2.6 Config

Il file *config.py* contiene un insieme di costanti che definiscono la configurazione dell'applicazione.

Di particolare rilevanza è la variabile *DIR_FILES_APP* contenente il path della directory nelle quale sono presenti i file utilizzati per la costruzione del *radix-tree*.

Chapter 3

Istruzioni per l'esecuzione

3.1 Opzioni a riga di comando

Di seguito saranno elencate le opzioni disponibili e una breve descrizione per ognuna di esse.

-i <interface> (obbligatorio) Specifica l'interfaccia su cui catturare il traffico di rete.

-f <file1 ... filen> (obbligatorio) Specifica una lista di file, ognuno contenente un insieme di indirizzi IP in notazione CIDR per classificare il traffico di quella determinata applicazione. Il nome del file deve essere della forma *nomeapp_ ip.txt*.

-ah <ip1 ... ipn> Specifica un insieme di indirizzi IP sui quali effettuare l'*ARP poisoning*.

-g <gateway> Specifica l'indirizzo IP del gateway. Tale argomento risulta necessario solo nel caso in cui venga fornito l'argomento *-ah*.

-h Visualizza l'help.

3.2 Esecuzione e terminazione

Per avviare il programma basta digitare:
sudo python3 AppDetection.py [-h] -i INTERFACE -f FILES [FILES ...] [-ap [ADDRHOST [ADDRHOST ...]]] [-g GATEWAY]
Dopodiché è necessario aprire la pagina web all'indirizzo `http://127.0.0.1:8050/` per poter visualizzare il grafico relativo alla classificazione del traffico catturato.
Premendo la combinazione di tasti CTRL-C è possibile interrompere il programma.

Chapter 4

Test e dipendenze

4.1 Test

Il progetto è stato sviluppato in *Python 3.5.3* e testato su *Debian 9*.
Nella figura 4.1 sono illustrati due screenshot che mostrano l'applicazione in esecuzione.

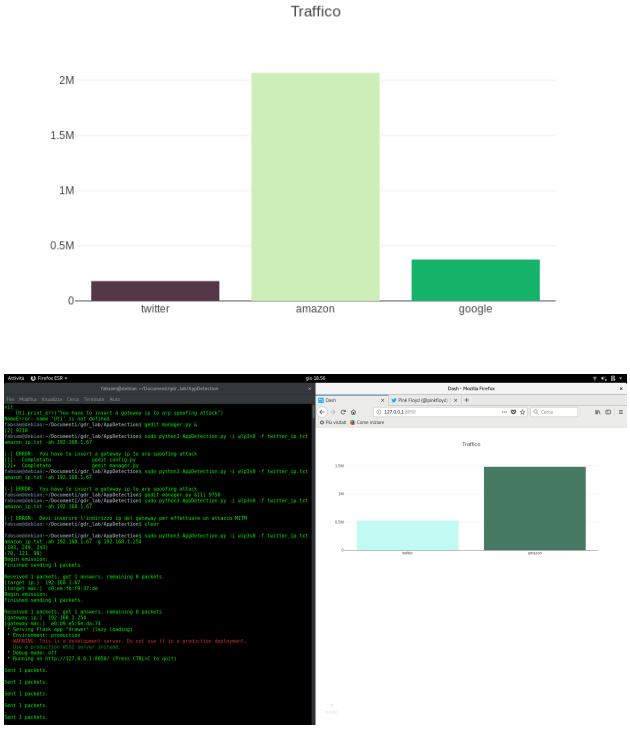

Figure 4.1: Esempi di test effettuati

4.2 Dipendenze

Per poter eseguire *AppDetection* è necessario installare le seguenti librerie:

- *scapy* (pip3 install scapy)
- *dash* (pip3 install dash)
- *dash-core-components* (pip3 install dash-core-components)
- *dash-html-components* (pip3 install dash-html-components)
- *pyshark* (pip3 install pyshark)
- *numpy* (pip3 install numpy)
- *radix* (pip3 install py-radix)
- *plotly* (pip3 install plotly)

Inoltre sono state utilizzate anche le librerie *argparse* e *threading* presenti nella libreria standard di Python.
Il codice è stato commentato utilizzando le *docstring* di Python in modo da potervi accedere in una sessione interattiva dell'interprete.