Traccia 1: Angelo Pio Pompeo MATRICOLA: 2014615

**Descrizione del codice e della configurazione:**

Il codice del progetto è realizzato tramite il linguaggio di programmazione C, il codice può essere diviso in tre sezioni a livello concettuale:  
- file di configurazione Docker : Dockerfile, ControllerDockerfile e docker-compose.yml

- file del programma C&C denominato controller per semplicità: controller.c utils.c common.h controller.h utils.h

- file programma bot: bot.c bot.h

**Comunicazione**

Controller e bot comunicano tramite scambio di messaggi http, l’ip del controller è statico e deciso a priori.  
  
Nel progetto ho deciso di usare la tecnologia di basata sui container fornita da Docker, questa porta vari vantaggi tra I quali la possibiltà di poter eseguire instanze multiple del programma bot ognuna con un suo indirizzo IP univoco (vedi sotto paragrafo Docker per maggiori chiarimenti).

Per permettere la comunicazione client-server tramite il protocollo http si fa uso della libreria : “libcurl” , lato client per l’invio di richieste http e della libreria “microhttpd” lato server per ricevere richieste http ed inviare risposte. Sia bot che controller usano entrambe queste librerie allo stesso tempo per scopi differenti:

controller usa libcurl per inviare comandi ai suoi bot e microhttpd per ricevere notifiche sull’esecuzione dei comandi inviati. Bot invece usa libcurl per fare la prima connessione al controller e quindi essere registrato nella botnet e microhttpd per rimanere costantemente in modalità di ricezione di comandi dal controller, una volta ricevuto un comando e portato a termine bot invia una notifica al controller tramite libcurl per segnalare l’avvenuto eseguimento del comando.

**Docker**

Nel file docker-compose.yml viene definita una custom network bridge: “frontend” a cui appartengono bot e controller, alla rete frontend viene assegnata una subnet: 172.20.0.0/24 così da poter assegnare indirizzi ip univoci a tutti I bot e al controller, al quale però viene impostato un indirizzo ip statico data la sua natura di server (tale indirizzo ip è 172.20.0.2).   
L’ip del bot viene assegnato automaticamente dal daemon docker rispettando il vincolo di appartenenza alla subnet prima citata, la porta di connessione del bot invece è randomica, viene selezionata una porta libera del sistema dalla quale il bot può ricevere comandi dal C&C, ciò viene fatto tramite la funzione definita setBotPort.

Tramite il comando < docker-compose up –build –scale bot=x -d> eseguito dalla cartella in cui sono presenti I file del progetto (cartella src) viene nell’ordine :

* fatta la build dei due file Dockerfile
* creati ed eseguiti x container test\_bot\_x e un container test\_controller\_1

a questo punto tramite il comando <docker exec -it test\_controller\_1 bash> si può entrare nella shell di sistema del test\_controller\_1 e lanciare il programma tramite <./controller>

Il flag –scale bot =x permette di creare ed eseguire x instance del programma bot, x deve essere un numero minore o uguale di 254 in quanto la subnet “frontend” ha maschera di rete /24 la quale permette l’asssegnazione di al più 254 indirizzi ip.  
  
Il file Dockerfile crea un’immagine docker chiamata “bot” a partire da un’immagine ubuntu in cui vengono installate le opportune dependencies, copiati e compilati I file bot.c e bot.h ed infine lanciato il programma bot.   
ControllerDockerfile si comporta in maniera analoga ad eccezion del fatto che il il programma controller non viene lanciato in automatico benì sarà necessario connettersi al terminal bash per farlo.  
  
**Bot**

Il programma bot svolge le seguenti funzioni:

* Connessione al C&C con passaggio dei propri IP e PORT NUMBER
* Ricezione di comandi dal C&C, I comandi supportati sono email – http\_req – sys-info

Al’’avvio del programma bot le prime istruzioni invocate sono: setBotIP e setBotPort, queste due funzioni si occupano di inizializzare le variabili globali IP e port.   
In seguito il bot tenta di instaziare una connessione con il C&C tramite l’invio di una get request per inviare I dati relativi al proprio IP e PORT NUMBER , se la connessione fallisce ritenta ogni 15 secondi finchè non ha successo.  
Una volta inviate le informazioni necessarie al C&C il bot resta indefinitivamente in ascolto sulla propria porta.

Al momento della ricezione di una richiesta http dal server il bot analizza l’url di quest’ultima, se l’url contiene gli endpoint (“http\_req”, “sys\_info”, “email”) il bot eseguirà il relativo comando.

Una volta terminata l’esecuzione di un comando il bot invia tramite la funzione notifyController() una http request per notificare al controller il termine dell’esecuzione.

**C&C (controller)**

Il programma controller presenta una struct active\_bots per tener traccia dei bot afferenti alla botnet,ed usa un paradigma di programmazione multithreading, in particolare usa due thread: il primo invoca la funzione child() usata per rimanere sempre in ricezione di richieste http sulla porta 8081 mentre il secondo thread viene usato per interagire da riga di comando con il C&C.

In aggiunta ogni connessione http in entrata viene gestita tramite la generazione di un nuovo thread, ciò viene impostato tramite la costante *MHD\_USE\_THREAD\_PER\_CONNECTION* nel metodo MHD\_start\_daemon() usato da microhttpd per restare in modalità di ricezione.

typedef struct bot

{

int bot\_id;

long port;

struct in\_addr bot\_address;

struct in\_addr target\_address;

char action[10];

struct bot \*next;

} active\_bots;

La struct active\_bots è nell’ambito del programma una critical section in quanto due stessi bot potrebbero notificare allo stesso momento il C&C dell’avvenuto completamento di un comando; poichè il C&C aggiorna le informazioni riguardanti comando eseguito e host target di ogni bot afferente alla botnet c’è necessità di sincronizzare I vari thread, per fare ciò tutte le funzioni di utility definite in utils.c usate all’interno del programma per gestire la lettura e scrittura alla botnet fanno uso di due semafori: ***r – w****.*

**Esempio di esecuzione:**