Problema affrontato

Si vuole progettare una botnet (a scopo didattico) in grado di effettuare attacchi di tipo “Distributed Denial-of-Service”, con la possibilità di controllare individualmente ciascun nodo della rete.

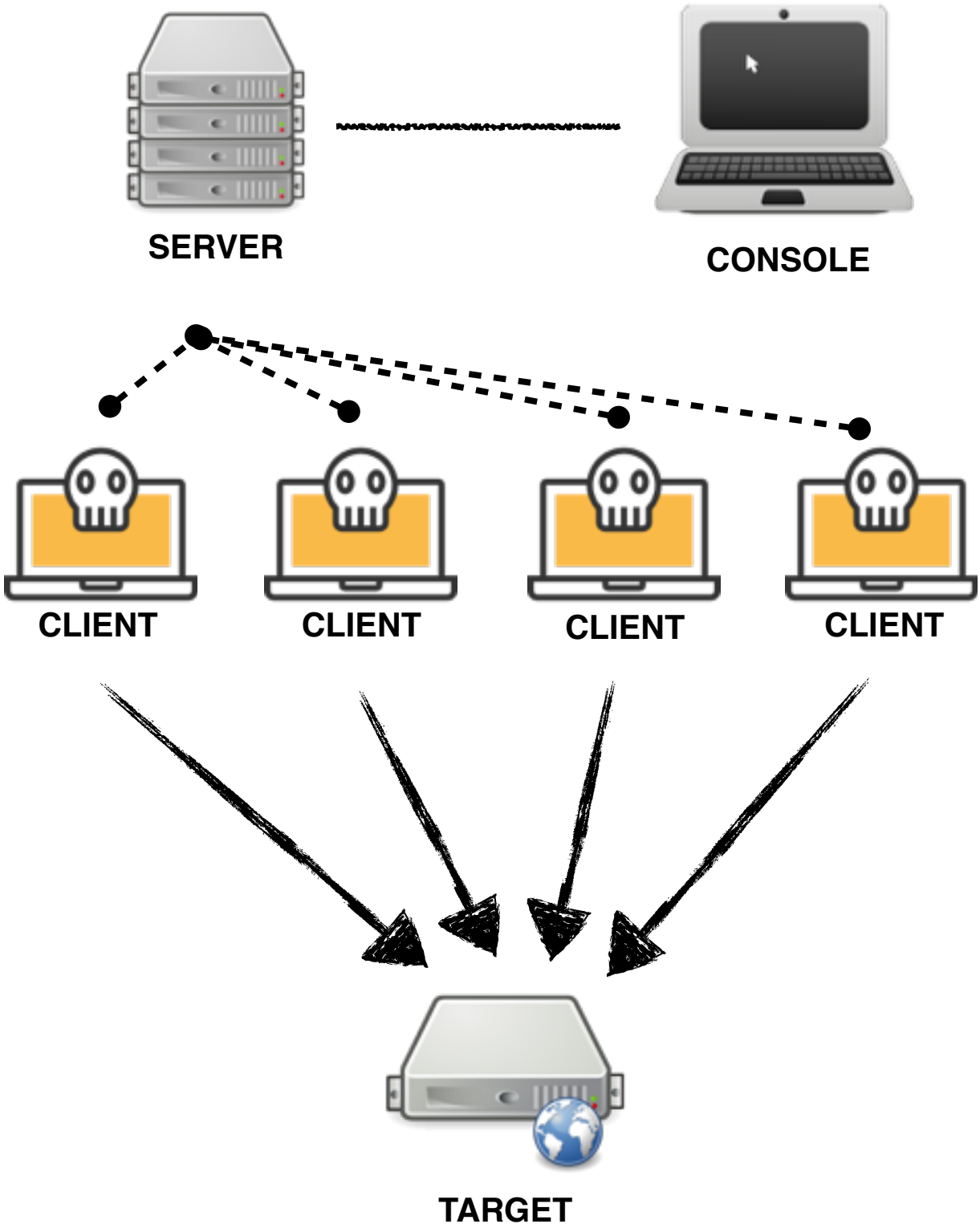
Requisiti progettuali

- E’ necessario utilizzare le seguenti tecniche di programmazione: Functional Programming, Object Orientation e Generic Programming.

Requisiti funzionali

- Il software ha un architettura di tipo “Command and Control”.
- Ogni client deve essere in grado di riconnettersi al server nel caso in cui la connessione venga interrotta.
- Il server deve essere in grado di gestire più client contemporaneamente.
- Il server deve gestire comandi inviati a client multipli.
- La console deve fornire un interfaccia semplice e completa di tutte le funzionalità disponibili.
- Tutte le comunicazioni vengono cifrate per aggirare un ipotetico “Intrusion Detection System”.
- E' possibile sfruttare l'intera rete di client per eseguire attacchi ddos.
- Il client permette all’attaccante, tramite server e console, di accedere alla shell di ogni singolo client.
- Il client ha la possibilità di scattare foto dalla webcam o catturare uno screenshot e inviarlo alla console.
- Il client ha la possibilità di registrare un clip audio dal microfono e inviarlo alla console.
- Il client è predisposto per comportarsi come un cryptolocker (cifrando i file presenti su disco)
- Il client ha la possibilità di essere lanciato all'avvio del sistema (persistenza)
- E' possibile compilare il codice usando la libreria pyinstaller.

Schema funzionale



Descrizione della Console

La console è la parte dell'applicativo che fornisce un'interfaccia per la gestione dei client e degli attacchi. Di seguito viene approfondito il funzionamento della console.

All'avvio, come operazione preliminare, verrà chiesto all'utente di fornire una password per l'autenticazione al server. Nel caso in cui la password fornita non fosse corretta, il server provvederà a interrompere immediatamente la connessione.

Le uniche funzioni della console sono la connessione/comunicazione con il server (tramite il protocollo precedentemente descritto) e la lettura/scrittura di dati su disco.

Per rendere semplice ed intuitivo l'utilizzo della console è previsto un sistema di auto-completamento dei comandi (consultabili con il comando "cmd_help"):

Descrizione dei comandi:

COMANDO	DESCRIZIONE
cmd_help	Mostra la lista di tutti i comandi disponibili.
cmd_clear	Pulisce la schermata del terminale senza inviare alcun comando al server.
cmd_lpwd	Mostra la directory locale.
cmd_lls	Mostra i file nella directory locale.
cmd_lcd	Cambia la directory locale.
cmd_list	Visualizza a schermo la lista di tutti i client connessi al server, pronti per l'interazione.
cmd_interact	Seleziona il client e inizia un interazione singola con esso.
cmd_stop	Interrompe un interazione singola con un client.
cmd_exit	Interrompe l'esecuzione della console.

cmd_put	Legge un file nella directory locale, lo invia e lo salva sulla directory del client.
cmd_get	Il client legge un file nella propria directory e lo invia. La console lo salva nella directory locale.
cmd_download	Scarica un file remoto nella directory del client
cmd_reverse_shell	Lancia una reverse shell del client su un listner esterno al programma (Si consiglia l'uso di netcat)
cmd_flood_add	Aggiunge un target nella pila degli attacchi da eseguire.
cmd_flood_remove	Rimuove l'intera pila di attacchi da eseguire.
cmd_flood_start	Scorre la pila degli attacchi e li esegue
global_flood_add	Aggiunge un target nella pila degli attacchi da eseguire. (Per ogni client della botnet)
global_flood_remove	Rimuove l'intera pila di attacchi da eseguire. (Per ogni client della botnet)
global_flood_start	Scorre la pila degli attacchi e li esegue. (Per ogni client della botnet)
cmd_screenshot	Invia il comando di catturare uno screenshot al client, attende un immagine e la visualizza.
cmd_screenshot_save	Invia il comando di catturare uno screenshot al client, attende un immagine e la salva su disco.
cmd_camera	Invia il comando di catturare una foto da webcam al client, attende un immagine e la visualizza.
cmd_camera_save	Invia il comando di catturare una foto da webcam al client, attende un immagine e la salva su disco.
cmd_mic	Invia il comando di catturare un clip audio al client, attende un file audio e lo salva su disco.
cmd_ransome	Invia il comando e la chiave per cifrare i files in una directory sul client.
cmd_unransome	Invia il comando e la chiave per decifrare i files in una directory sul client.
cmd_msgbox	Invia un messaggio al client che verrà visualizzato come "message box".

Descrizione del Server

Il server rappresenta il cuore della botnet, esso gestisce la comunicazione tra le varie parti del software, la coordinazione degli attacchi e l'interazione con uno o più client. Di seguito viene approfondito il funzionamento del server.

Connessione:

La connessione al server avviene tramite una porta stabilità in precedenza. Per praticità si è scelto di utilizzare un unica porta sia per i client che per la console.

Al momento della connessione per distinguere i client dalla console, il server attende una di queste tre stringhe identificatrici:

<_CONSOLE_>	Questa stringa identifica una console.
<_CLIENT_>	Questa stringa identifica un client della botnet.
<_TESTER_>	Questa stringa è utilizzata per effettuare un test di connessione al server.

Nel caso in cui venga fornita una stringa differente il server provvederà a chiudere istantaneamente la connessione.

Per evitare abusi sia il client che la console devono fornire una password nel momento successivo all'identificazione, nel caso in cui la password fornita non fosse corretta, la connessione verrà interrotta immediatamente.

Ad ogni console o client connesso verrà dedicato un thread con funzioni specifiche. Verranno quindi salvati indirizzi e socket in array differenti per permettere al server di gestire le comunicazioni.

Thread di tipo client:

Il thread utilizzato per i client ha due funzioni principali:

1) Stabilisce (mediante una password) se il client connesso è abilitato all'interazione con il server. In caso contrario si provvederà a interrompere immediatamente la connessione.
2) Verifica, istante per istante, che la connessione con il client sia attiva. In caso contrario verrà chiuso il socket e rimossi riferimenti dagli array di gestione.

Thread di tipo console:

Il thread dedicato alla console permette la comunicazione diretta con il server, con un client specifico o con l'intero array di client.
Nell'istante in cui la console si collega, il thread resta in attesa di una password necessaria a stabilire se il client (di tipo console) è abilitato all'interazione, in caso contrario si provvederà a interrompere immediatamente la connessione.

Ad identificazione avvenuta, la console interagirà direttamente con il server. Sarà quindi possibile ottenere una lista dei client connessi o inviare attacchi globali coinvolgendo tutti i client.

Una volta avviata l'interazione con un singolo client, il thread si comporterà da intermediario per la comunicazione con la console.

Nel caso in cui si scelga di avviare un attacco coinvolgendo tutti i client, il thread provvederà in un primo momento a ricevere il comando d'attacco dalla console, successivamente esso verrà inoltrato ad ogni singolo client verificandone la corretta ricezione.

CLIENT - Panoramica generale

Il client rappresenta la parte dell'applicativo installata su un host remoto (precedentemente compromesso), che permette ad un attaccante di controllare il sistema, ottenere dati o lanciare attacchi di tipo ddos.
Di seguito viene approfondito il funzionamento.

Una volta avviato il client cerca di connettersi al server tramite la porta prestabilita.
A connessione avvenuta, vengono inviate la stringa identificatrice e la password, successivamente il client resta in attesa di istruzioni.

Sono stati previsti alcuni comandi eseguibili direttamente dal client:

- **<_AREYOUALIVE_>** stabilisce se la connessione è attiva.
- **cd** cambia la directory.
- **<_GETPATH_>** invia l'attuale directory.
- **cmd_get** scarica un file dalla console.
- **cmd_put** invia un file alla console.
- **cmd_download** scarica un file remoto.
- **cmd_reverse_shell** lancia una reverse shell su un host remoto.
- **cmd_flood_add** aggiunge un target alla pila degli attacchi.
- **cmd_flood_remove** rimuove l'intera pila degli attacchi.
- **cmd_flood_start** scorre la pila ed esegue gli attacchi.
- **cmd_screenshot** cattura uno screenshot e lo invia alla console.
- **cmd_camera** cattura una foto e la invia alla console.
- **cmd_mic** cattura un clip audio e lo invia alla console.
- **cmd_ransome** cifra una directory e visualizza un msgbox
- **cmd_unransome** decifra una directory e visualizza un msgbox
- **cmd_msgbox** mostra un messaggio a video tramite un msgbox

Nel caso in cui il comando inviato non sia fra quelli elencati, il client proverà ad eseguirlo sulla shell dell'host.
Non potendo gestire comandi interattivi e per evitare che un comando

impedisca di continuare a controllare la shell, è stata predisposta una funzione che interrompe l'esecuzione del comando dopo 10 secondi.
Sono state previste eccezioni a più livelli, così in caso di errore durante l'esecuzione di una funzione, il client annullerà l'operazione in corso. Resterà quindi in attesa di istruzioni o interromperà la connessione, cercando nuovamente di riconnettersi al server.

CLIENT - Panoramica Funzionalità

Il client possiede cinque principali funzionalità:

Screenshot: utilizzando il modulo ImageGrab della libreria PIL (pillow) in combinazione con la libreria cStringIO, sul client è stata inserita una funzione che permette di scattare uno screenshot, salvarlo su un oggetto di tipo cStringIO e ritornare l'immagine sotto forma di stringa. Questo permette di catturare uno screenshot e inviarlo alla console come una stringa di testo.

Camera: è stata predisposta una funzione che utilizzando la libreria cv2 (importata parzialmente utilizzando tramite il file cv2.pyd) in combinazione con la libreria cStringIO, scatta e salva su file una foto dalla webcam. Successivamente il file viene letto e in inserito su un oggetto di tipo cStringIO, da esso è possibile estrarre una stringa che rappresenta l'immagine originale. Come operazione finale è prevista la sovrascrittura e la cancellazione del file contenente la foto catturata in precedenza.

Microfono: utilizzando le librerie pyaudio e wawe è possibile salvare su file (nella directory %TEMP%) un clip audio di arbitraria lunghezza.
La funzione predisposta dopo aver acquisito l'audio e salvato il file, si preoccuperà di leggerlo e restituirlo in formato binario. Come operazione finale, anche in questo caso è prevista la sovrascrittura e cancellazione del file.

Ransomware: sono state predisposte due funzioni, una per cifrare e l'altra per decifrare (con chiave arbitraria) il contenuto di una directory.
A seguito dell'operazione di cifratura verrà richiesto, tramite un messaggio, il pagamento di un compenso per decifrare i files.

Persistenza: compilando un eseguibile del client, è possibile utilizzare la funzionalità di auto-avvio. Lanciando l'eseguibile senza specificare il parametro d'avvio (precedentemente stabilito dall'attaccante), il client copierà se stesso in una directory nascosta, creerà uno script di avvio (che verrà inserito fra le chiavi di avvio automatico del registro di sistema) e imposterà un cronjob per riavviarsi dopo 30 secondi in modalità stealth. E' prevista anche la creazione di uno script di "disinfezione" il quale eliminerà completamente il client dal sistema.

CLIENT - Modulo d'attacco DoS

Come accennato in precedenza, il client possiede un modulo d'attacco che gli permette di eseguire attacchi di tipo DoS.
I parametri di ogni singolo attacco vengono conservati in una pila, in attesa che l'attaccante invii il comando d'avvio. Di seguito viene descritta la procedura:

1) Il client riceve il comando **cmd_flood_add** seguito dal tipo di protocollo, indirizzo dell'host, porta del servizio, numero di thread e durata dell'attacco in secondi.
2) Viene generato un oggetto specifico (in base al tipo di protocollo scelto) e viene inserito in una pila che contiene tutti gli attacchi che il client dovrà eseguire.
3) Nel momento in cui il client riceve il comando **cmd_flood_start** la pila viene passata ad un oggetto di classe "Attack", il quale scorrerà la pila e eseguirà i vari attacchi genericamente rispetto al tipo di oggetto estratto.

4) La pila viene automaticamente rimossa al termine degli attacchi o a seguito della ricezione del comando **cmd_flood_remove.**

Sfruttando a livello server i comportamenti descritti in precedenza, sono stati inseriti nella console altri tre comandi che hanno le seguenti corrispondenze:

LATO SERVER	**->**	**LATO CLIENT**
global_flood_add	**->**	**cmd_flood_add**
global_flood_start	**->**	**cmd_flood_start**
global_flood_remove	**->**	**cmd_flood_remove**

Quando il server riceve uno di questi comandi con i relativi parametri, scorre la lista dei client e inoltra a ciascuno il comando corrispondente. Questo permette all'attaccante di sfruttare tutti i client contemporaneamente per eseguire un attacco di tipo DDoS.

Protocollo di comunicazione

L'applicativo utilizza un protocollo personalizzato basato su stringhe di testo per la trasmissione delle informazioni.
Si è scelto di creare un protocollo personalizzato in primo luogo per garantire la privatezza delle informazioni scambiate, ma soprattuto per aggirare un ipotetico "Intrusion Detection System". Di seguito vengono elencate le caratteristiche.

Spedizione dei dati

1) L'informazione viene codificata in base64
2) La stringa codificata viene cifrata
3) Viene misurata la lunghezza della stringa cifrata
4) Viene effettuato il padding del valore precedentemente calcolato
5) Viene formata la stringa da inviare combinando il valore della lunghezza della stringa cifrata e la stringa cifrata.

Esempio con informazione ("comando_1") e password ("prova"):

1) Y29tYW5kb18x
2) \0xcb\0x14\0xfa\0x3d\0x09\0xa8\0xf5\0x65\0x63\0xe5\0x2f\0x06\0xb3\0xd5\0x28\0xc5
3) \0xcb\0x14\0xfa\0x3d\0x09\0xa8\0xf5\0x65\0x63\0xe5\0x2f\0x06\0xb3\0xd5\0x28\0xc5 ha una dimensione di 77 byte.
4) Padding di 77 -> 0000000077
5) 0000000077\0xcb\0x14\0xfa\0x3d\0x09\0xa8\0xf5\0x65\0x63\0xe5\0x2f\0x06\0xb3\0xd5\0x28\0xc5

Ricezione dei dati

La ricezione dei dati avviene ricevendo i primi 10 caratteri della stringa in modo da poter determinare quanti caratteri sarà necessario ricevere per ottenere la stringa (cifrata) completa, successivamente i dati verranno prima decifrati e poi decodificati.