



UNIVERSITÀ DELLA CALABRIA

DIPARTIMENTO DI
INGEGNERIA INFORMATICA,
MODELLISTICA, ELETTRONICA
E SISTEMISTICA

DIMES

**CdL Magistrale in
Ingegneria Informatica**

Report Progetto - Network Security

Conflict Detector

Docente

Prof. [REDACTED]

Esercitatore

Ing. [REDACTED]

Studenti

Federica Cosenza [REDACTED]

Michele Purrone [REDACTED]

Antonino Vaccarella [REDACTED]

Anno Accademico 2023/2024






Indice

Installazione della Macchina.....	3
Installazione di ONOS	3
Creazione di un'Applicazione	4
Installazione e Attivazione dell'Applicazione	5
Installazione di Mininet	5
Installazione di IntelliJ IDEA	6
Struttura del Progetto	7
Classe AppComponent.....	7
MyRule.....	13
MyScore.....	14

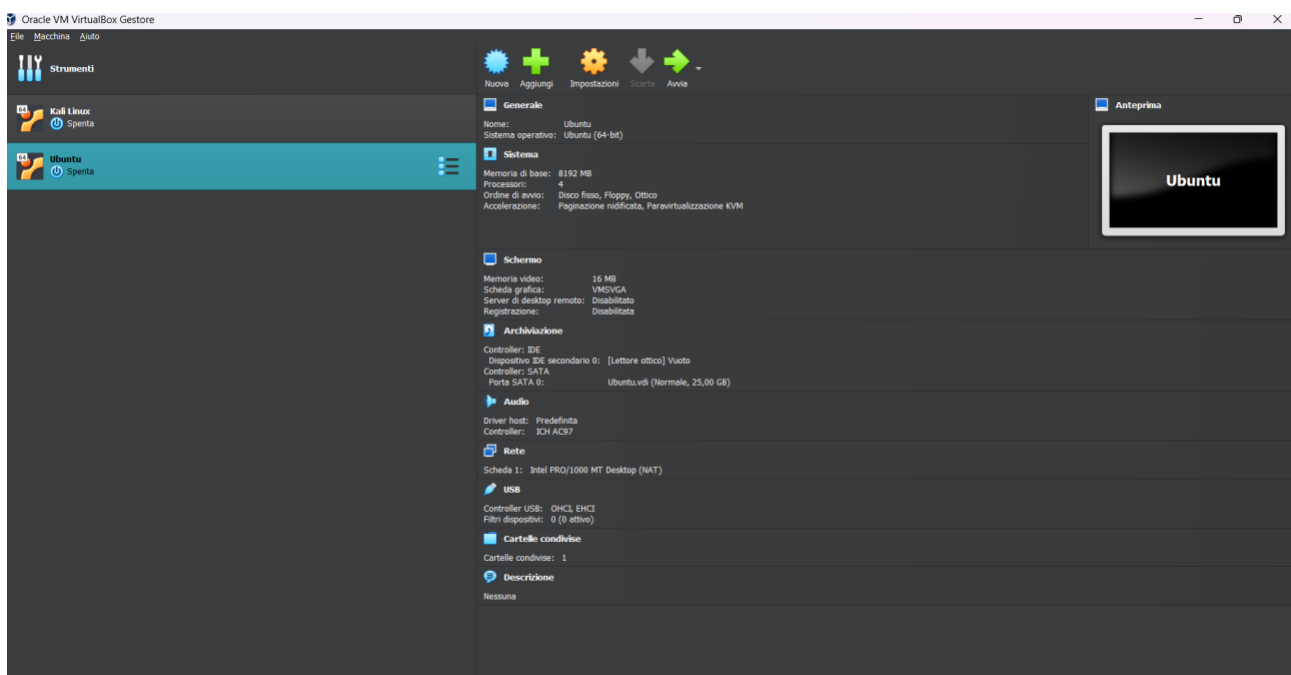
Installazione della Macchina

Per importare la macchina è necessario avere Oracle VirtualBox come software per la virtualizzazione. Vediamo ora le relative procedure per l'installazione della macchina:

Estrarre il file zip, con all'interno:

 Ubuntu.vdi	14/09/2024 15:55	Virtual Disk Image	26.216.448 ...
 Ubuntu.vbox	14/09/2024 15:53	VirtualBox Machin...	6 KB
 Ubuntu.vbox-prev	14/09/2024 15:52	File VBOX-PREV	7 KB
 Logs	14/09/2024 15:53	Cartella di file	
✓ All'inizio dell'anno			
 Snapshots	16/04/2024 18:55	Cartella di file	

Selezionando il file .vbox, VirtualBox effettuerà le operazioni di setup:



La macchina virtuale è già fornita di tutto il software necessario. Per completezza, si riporta comunque la guida su come installare i tool impiegati nella realizzazione dell'idea progettuale.

Installazione di ONOS

La versione di ONOS di riferimento per la realizzazione dell'idea progettuale è la 2.6.0.

La versione di ONOS adottata richiede JAVA 11 (informazione consultabile sulla pagina <https://wiki.onosproject.org/display/ONOS/Requirements>). Si eseguono i seguenti comandi sul terminale:

```
sudo apt update
sudo apt install openjdk-11-jdk
```

e si setta la variabile \$JAVA_HOME:

```
sudo cat >> /etc/environment <<EOL
JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
EOL
```

La pagina <https://wiki.onosproject.org/display/ONOS/Installing+on+a+single+machine> presenta i passi da seguire per l'installazione:

```
sudo mkdir /opt ; cd /opt
sudo wget -c https://repo1.maven.org/maven2/org/onosproject/onos-
releases/2.6.0/onos-2.6.0.tar.gz
```

Una volta scaricato l'archivio tar, occorre decomprimere l'archivio e rinominare la directory estratta in “onos”.

```
sudo tar -xvf onos-2.6.0.tar.gz
sudo mv onos-2.6.0 onos
```

A questo punto è possibile avviare il servizio ONOS e la CLI con i comandi:

```
sudo /opt/onos/bin/onos-service start
sudo /opt/onos/bin/onos -l onos
```

Dalla CLI si attivano i due moduli essenziali per il protocollo OpenFlow e per la gestione del traffico:

```
app activate org.onosproject.openflow
app activate org.onosproject.fwd
```

Creazione di un'Applicazione

Il comando da eseguire per creare una nuova applicazione è:

```
mvn archetype:generate -DarchetypeGroupId=org.onosproject -
DarchetypeArtifactId=onos-bundle-archetype -
DarchetypeVersion=2.2.1-b2
```

avendo cura di cambiare opportunamente la versione *archetype* che potrebbe variare nel tempo.

Maven scaricherà le dipendenze necessarie e chiederà di impostare le proprietà “groupId” e “artifactId” del progetto, per poi finalizzare la creazione.

Installazione e Attivazione dell'Applicazione

È possibile compilare l'applicazione in un archivio di applicazioni ONOS o in un file .oar utilizzando il comando:

```
sudo mvn clean install
```

L'attivazione avviene caricando il file .oar e attivando l'applicazione tramite la GUI disponibile all'indirizzo <http://127.0.0.1:8181/onos/ui>.

Per rendere più semplice l'avvio di ONOS e il processo di compilazione, è stato creato uno script "run.sh" che automatizza il processo sopra descritto:

```
#!/bin/bash

PASSWORD="Arabella"

# Avvio di ONOS
gnome-terminal --tab -- bash -c "echo $PASSWORD | sudo -S
/opt/onos/bin/onos-service start; exec bash"

# Attesa di 15 secondi
sleep 15

# Avvio della CLI
gnome-terminal --tab -- bash -c "/opt/onos/bin/onos -l onos; exec
bash"

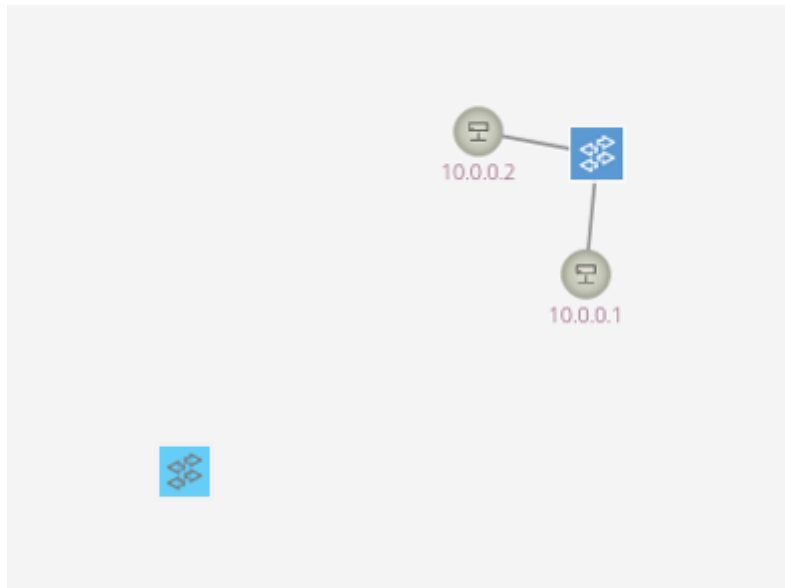
# Compilazione
gnome-terminal --tab -- bash -c "echo $PASSWORD | sudo -S mvn
clean install; exec bash"
```

Installazione di Mininet

La guida per l'installazione di Mininet è disponibile sulla pagina GitHub al link:

<https://github.com/mininet/mininet/tree/master>

La topologia predefinita è quella minima, che comprende uno switch kernel OpenFlow (identificato con "s1") collegato a due host ("h1" e "h2"), oltre al controller di riferimento ("c0").



Con il comando:

```
sudo mn --mac --switch ovsk,protocols=OpenFlow13 --  
controller=remote, ip=127.0.0.1, port=6653 --link=tc
```

è possibile avviare la CLI di Mininet. La topologia è adesso visualizzabile con il comando net.

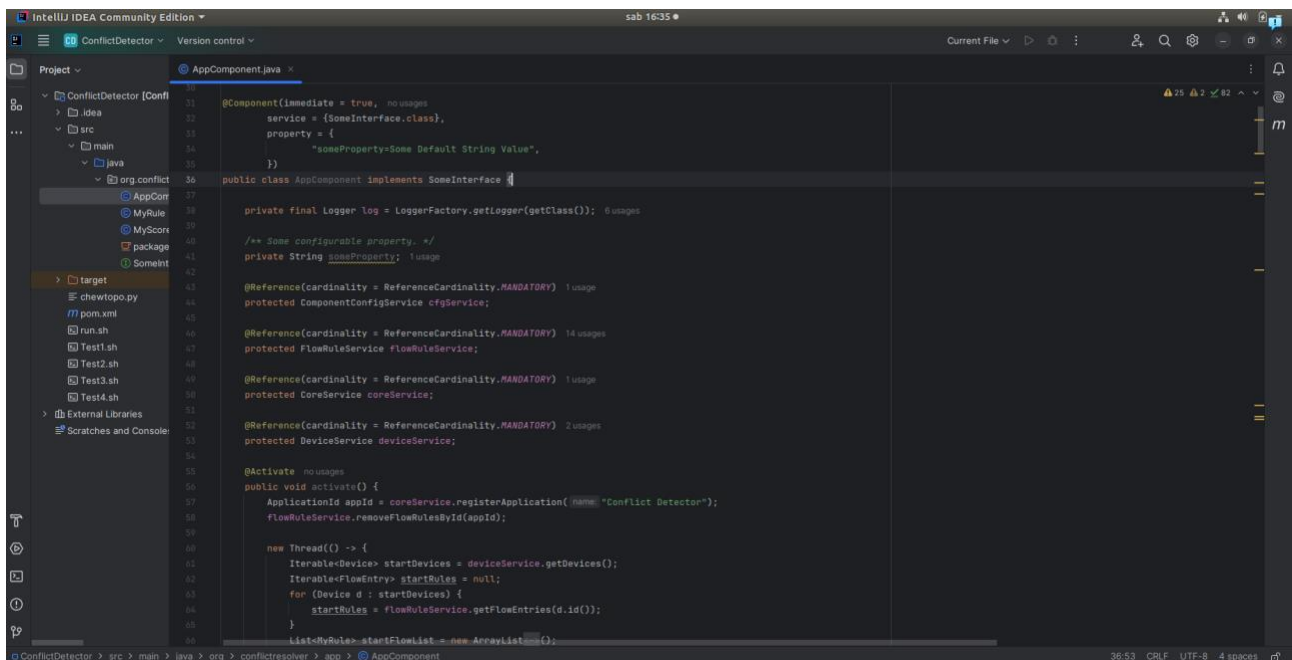
```
mininet> net  
h1 h1-eth0:s1-eth1  
h2 h2-eth0:s1-eth2  
s1 lo: s1-eth1:h1-eth0 s1-eth2:h2-eth0  
c0
```

Installazione di IntelliJ IDEA

L'ambiente di sviluppo scelto è IntelliJ IDEA. La guida per l'installazione è disponibile al link:

<https://www.jetbrains.com/help/idea/installation-guide.html#toolbox>

Struttura del Progetto



In `src/main/java/org/conflictresolver/app` ci sono le tre classi che definiscono il sistema: *AppComponent.java*, *MyScore.java* e *MyRule.java*.

Classe AppComponent

La classe AppComponent racchiude la logica dell'applicazione, essa è formata dai seguenti metodi:

Activate

L'annotazione `@Activate` indica che il metodo viene chiamato quando il componente viene attivato. L'applicazione viene registrata presso il *CoreService* con il nome "*Conflict Detector*": ogni applicazione ONOS ha bisogno di un *ApplicationId* univoco per gestire regole di flusso e altre operazioni di rete.

```
ApplicationId appId = coreService.registerApplication("Conflict Detector");
flowRuleService.removeFlowRulesById(appId);
```

Un nuovo thread viene avviato per eseguire le operazioni di monitoraggio e gestione dei flussi.

```
new Thread(() -> {
    // codice interno
}).start();
```

Viene recuperata una lista di tutti i dispositivi presenti nella rete utilizzando *deviceService.getDevices()*.

Per ogni dispositivo, vengono recuperate le regole di flusso (*FlowEntry*) associate tramite *flowRuleService.getFlowEntries(d.id())*.

```

Iterable<Device> startDevices = deviceService.getDevices();
Iterable<FlowEntry> startRules = null;
for (Device d : startDevices) {
    startRules = flowRuleService.getFlowEntries(d.id());
}

```

Per ogni regola di flusso trovata in precedenza, viene creata una nuova istanza di *MyRule*, una classe personalizzata che gestisce i dettagli della regola di flusso.

La lista memorizza tutte le regole attuali nel sistema, così da poterle confrontare successivamente con nuove regole.

```

List<MyRule> startFlowList = new ArrayList<>();
for (FlowEntry f : startRules) {
    startFlowList.add(new MyRule(f.id().value(), f, false));
}

```

Successivamente vengono inizializzate le strutture dati:

- **conflictList:** contiene le regole di flusso che risultano in conflitto e che potrebbero essere rimosse.
- **flowList:** contiene le regole di flusso attualmente attive nel sistema.
- **users:** una mappa per tenere traccia degli utenti (identificati tramite IP) e i loro punteggi.
- **bannedIp:** un insieme che contiene gli indirizzi IP che sono stati bannati a causa di comportamenti malevoli.
- **scheduler:** un esecutore programmato che gestisce il reinserimento degli IP bannati dopo un certo periodo di tempo.

```

List<MyRule> conflictList = new ArrayList<>();
List<MyRule> flowList = new ArrayList<>();
HashMap<String, MyScore> users = new HashMap<>();
ScheduledExecutorService scheduler =
    Executors.newScheduledThreadPool(1);
Set<String> bannedIp = new HashSet<>();

```

L'intero ciclo che monitora e gestisce le regole di flusso è implementato nel thread.

Prima di eseguire nuove operazioni, il sistema rimuove tutte le regole di flusso contrassegnate come "rimuovibili" (metodo *isRemovable()*).

```

while(true) {
    try {
        // Rimozione delle regole in conflitto
    }
}

```



```

        for (MyRule f : conflictList) {
            if (f.isRemovable()) {
                flowList.remove(f);
            }
        }
        conflictList.clear();

```

Il sistema aggiorna la lista dei dispositivi e delle regole di flusso a ogni ciclo. Le regole che erano già presenti all'inizio dell'esecuzione vengono rimosse, in modo che solo le nuove regole vengano analizzate per eventuali conflitti.

```

Iterable<Device> devices = deviceService.getDevices();
Iterable<FlowEntry> rules = null;
for (Device d : devices) {
    rules = flowRuleService.getFlowEntries(d.id());
}
for (FlowEntry f : rules) {
    if(!flowList.contains(new MyRule(f.id().value(), f, false))){
        flowList.add(new MyRule(f.id().value(), f, false));
    }
}
flowList.removeAll(startFlowList);

```

Il cuore del metodo sta nella gestione dei conflitti (*Overlap*, *Redundancy*, *Shadowing*, *Generalization*, *Correlation*) attraverso la logica di incremento del *punishment*:

```

if (!overlap(startFlowList, f).isEmpty() && !f.isRemovable()) {
    punishment += 0.05;
    flowRuleService.removeFlowRules(f.getFlow());
    users.get(ip).startTimer();
    conflictList.add(f);
}

```

Ogni tipo di controllo verifica se la regola corrente ("*f*") entra in conflitto con quelle iniziali.

- **Punizione:** Se la regola è in conflitto, viene aumentato il livello di "*punishment*" associato all'utente che ha inserito la regola.
- **Rimozione della regola:** Se viene rilevato un conflitto, la regola viene rimossa dal sistema.

La fiducia di un utente viene aggiornata a ogni ciclo in base alla formula per il calcolo della *trustness*.

```

users.get(ip).setPunishment(punishment);

```

```

if(punishment < 0.0) {
users.get(ip).setPunishment(0.0);
}
else if(punishment > 1.0) {
users.get(ip).setPunishment(1.0);
}

double oldTrustness = users.get(ip).getTrustness();
double newTrustness = 1.0;

double oldAlpha = users.get(ip).getAlpha();
double newAlpha = 0.0;

if(oldTrustness < 0.0) {
oldTrustness = 0.0;
}
else if(oldTrustness > 1.0) {
oldTrustness = 1.0;
}

// Calcolo della trustness
double beta = 0.001;
double lambda = 0.7;
newAlpha = oldAlpha * (1 + beta * oldTrustness);
double reward = newAlpha * (users.get(ip).getTime()) + 0.001;
if(reward > 1.0) {
reward = 1.0;
}
else if(reward < 0.0) {
reward = 0.0;
}
newTrustness = oldTrustness + (reward - punishment);

if(newTrustness < 0.0) {
newTrustness = 0.0;
}
else if(newTrustness > 1.0) {
newTrustness = 1.0;
}

```

```

}

users.get(ip).setTrustness(newTrustness);
users.get(ip).setPunishment(0.0);
users.get(ip).setAlpha(newAlpha);

```

Se la fiducia di un utente scende sotto una certa soglia, l'indirizzo IP viene bannato temporaneamente o permanentemente. Il ban può essere temporaneo (gestito dal “scheduler”) o permanente.

```

//Meccanismo di blocco dell'IP sospetto
for (Map.Entry<String, MyScore> pair : users.entrySet()) {

if (pair.getKey().equals(ip) && pair.getValue().getTrustness() <
0.7) {
    pair.getValue().setAttempts(pair.getValue().getAttempts() +
1);
if (pair.getValue().getAttempts() == 1) {
    //log.info("L'utente con IP: {}" + ip + " viene riconosciuto
come attaccante per la prima volta!", ip);
flowRuleService.removeFlowRules(f.getFlow());
users.get(ip).setBannedRules(users.get(ip).getBannedRules() + 1);
//log.info("L'utente con IP: {}" + ip + " ha {} " +
users.get(ip).getBannedRules() + " regole bannate");
conflictList.add(f);
f.setRemovable(true);
bannedIp.add(pair.getKey());

scheduler.schedule(() -> {
bannedIp.remove(pair.getKey());
}, 2, TimeUnit.SECONDS);
break myLoop;
}
if (pair.getValue().getAttempts() == 2) {
//log.info("L'utente con IP: {}" + ip + " viene riconosciuto come
attaccante per la seconda volta!", ip);
flowRuleService.removeFlowRules(f.getFlow());
users.get(ip).setBannedRules(users.get(ip).getBannedRules() + 1);

```

```

//log.info("L'utente con IP: {}" + ip + " ha {} " +
users.get(ip).getBannedRules() + " regole bannate");
conflictList.add(f);
f.setRemovable(true);
bannedIp.add(pair.getKey());

scheduler.schedule(() -> {
bannedIp.remove(pair.getKey());
}, 4, TimeUnit.SECONDS);
break myLoop;
}
if (pair.getValue().getAttempts() == 3) {
//log.info("L'utente con IP: {}" + ip + " viene riconosciuto come
attaccante per la terza volta!", ip);
flowRuleService.removeFlowRules(f.getFlow());
users.get(ip).setBannedRules(users.get(ip).getBannedRules() + 1);
//log.info("L'utente con IP: {}" + ip + " ha {} " +
users.get(ip).getBannedRules() + " regole bannate");
conflictList.add(f);
f.setRemovable(true);
bannedIp.add(pair.getKey());

scheduler.schedule(() -> {
bannedIp.remove(pair.getKey());
}, 8, TimeUnit.SECONDS);
break myLoop;
}
if (pair.getValue().getAttempts() >= 4) {
//log.info("L'utente con IP: {}" + ip + " viene riconosciuto come
attaccante, sarà bloccato per sempre!", ip);
flowRuleService.removeFlowRules(f.getFlow());
users.get(ip).setBannedRules(users.get(ip).getBannedRules() + 1);
log.info("L'utente con IP: {} " + ip + " ha " +
users.get(ip).getBannedRules() + " regole bannate");
conflictList.add(f);
f.setRemovable(true);
bannedIp.add(pair.getKey());
break myLoop;
}
}

```

```
}  
}
```

longToIp

Questo metodo converte un indirizzo IP rappresentato come stringa in un formato leggibile.

Subnet

Gestisce il controllo di sottoreti, applicando maschere di rete a indirizzi IP per determinare se appartengono alla stessa rete.

MyRule

La classe *MyRule* rappresenta una regola di flusso personalizzata. I campi caratterizzanti sono

```
public Long id;  
private FlowEntry flow;  
private boolean removable;
```

- *id*: Un identificatore univoco per la regola.
- *flow*: Un oggetto di tipo *FlowEntry* che rappresenta una regola di flusso nella rete.
- *removable*: Un booleano che indica se la regola può essere rimossa o meno.

Il resto della classe presenta il costruttore, i metodi accessori, *l'equals* e *l'hashCode*.

```
public MyRule(Long id, FlowEntry flow, boolean removable) {  
    this.id=id;  
    this.flow = flow;  
    this.removable = removable;  
}  
  
public FlowEntry getFlow() {  
    return this.flow;  
}  
  
public void setFlow(FlowEntry flow) {  
    this.flow = flow;  
}  
  
public boolean isRemovable() {
```

```

        return this.removable;
    }

    public void setRemovable(boolean removable) {
        this.removable = removable;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return
false;
        MyRule myRule = (MyRule) o;
        return Objects.equals(id, myRule.id);
    }

    @Override
    public int hashCode() {
        return Objects.hashCode(flow);
    }

```

MyScore

La classe *MyScore* definisce un oggetto che tiene traccia di una serie di metriche relative al comportamento di un utente. I campi caratterizzanti sono:

```

double trustness;
double alpha;
double punishment;
int attempts;
int loadedRules;
int bannedRules;
long startTime;

```

- **trustness:** il valore rappresenta il "livello di fiducia" dell'utente.
- **alpha:** un parametro che rappresenta il coefficiente di guadagno che influenza il calcolo della fiducia.
- **punishment:** un valore che rappresenta la quantità di "punizione" inflitta all'utente.

- **attempts**: tiene traccia del numero di tentativi da parte dell'utente di caricare regole in conflitto o regole dannose.
- **loadedRules**: conta il numero di regole che l'utente ha caricato con successo.
- **bannedRules**: conta il numero di regole che sono state rimosse dal sistema.
- **startTime**: il tempo di inizio relativo all'inizio del monitoraggio dell'utente.

Il resto della classe presenta i costruttori, i metodi accessori, e un metodo *getTime()* che calcola il tempo trascorso in secondi dall'ultimo reset del timer, impiegato per tenere traccia del tempo che passa da quando è stato penalizzato per l'ultima volta.

```
public MyScore() {
    this.trustness = 1.0;
    this.alpha = 0.01;
    this.punishment = 0.0;
    this.attempts = 0;
    this.loadedRules = 0;
    this.bannedRules = 0;
    this.startTimer();
}

    public MyScore(double trustness, double alpha, double
punishment) {
        this.trustness = trustness;
        this.alpha = alpha;
        this.punishment = punishment;
        this.attempts = 0;
        this.loadedRules = 0;
        this.bannedRules = 0;
        this.startTimer();
    }

        public MyScore(double trustness, double alpha, double
punishment, int attempts, int loadedRules, int bannedRules) {
            this.trustness = trustness;
            this.alpha = alpha;
            this.punishment = punishment;
            this.attempts = attempts;
            this.loadedRules = loadedRules;
            this.bannedRules = bannedRules;
            this.startTimer();
        }
```

```
public double getTrustness() {
    return this.trustness;
}

public void setTrustness(double trustness) {
    this.trustness = trustness;
}

public double getAlpha() {
    return this.alpha;
}

public void setAlpha(double alpha) {
    this.alpha = alpha;
}

public double getPunishment() {
    return this.punishment;
}

public void setPunishment(double punishment) {
    this.punishment = punishment;
}

public int getAttempts() {
    return this.attempts;
}

public void setAttempts(int attempts) {
    this.attempts = attempts;
}

public int getLoadedRules() {
    return loadedRules;
}

public void setLoadedRules(int loadedRules) {
    this.loadedRules = loadedRules;
}
```



```
}

public int getBannedRules() {
    return bannedRules;
}

public void setBannedRules(int bannedRules) {
    this.bannedRules = bannedRules;
}

public void startTimer() {
    this.startTime = System.currentTimeMillis();
}

public long getTime() {
    long currentTime = System.currentTimeMillis();
    return (currentTime - this.startTime) / 1000;
}
```