

SDN Project Work (Firewall)

Antonio Sirignano

Prof. Giorgio Ventre

Abstract—In tale progetto viene presentato un SDN che utilizza il Machine Learning per la rilevazione di un attacco DDoS. Tutto il codice è presente su [github](#).

Contents

1	Introduzione	1
2	Rilevamento degli attacchi DDoS basato su Support Vector Machine (SVM)	1
2.1	Raccolta dello stato dei flow	1
2.2	Estrazione dei valori caratteristici	1
3	Classificatore	2
3.1	SVM	2
4	Applicazione	2
4.1	Topologia	2
4.2	Controller	2
4.3	Machine Learning	3
4.4	Topologia	4
4.5	Tracker	4
4.6	Esecuzione	6
	<i>Modello SVM • Rilevamento attacco nella topologia</i>	
5	Conclusioni e sviluppi futuri	7
	References	7

1. Introduzione

Gli attacchi DDoS (Distributed Denial of Service) possono causare anomalie nei servizi di rete, portando talvolta ad ingenti perdite economiche e ad altre catastrofiche conseguenze. Obiettivo di tale progetto è quello di trovare un modo per rilevare tali attacchi e gestirli.

La SDN (Software-defined networking) è un architettura che separa la rete in data plane e control plane avente caratteristiche di una rete programmabile, un controllo centralizzato, un'interfaccia aperta. Gli attaccanti mirano alla banda della rete, alle risorse di sistema, alle risorse applicative, in modo da bloccare i servizi di rete. Le difficoltà per rilevare tali attacchi sono le seguenti:

1. Le caratteristiche di tali attacchi sono difficili da identificare;
2. Mancanza di collaborazione tra nodi della rete;
3. Variazione dei tool utilizzati per l'attacco sempre più frequente;
4. L'utilizzo fraudolento degli indirizzi rendono più difficile il tracciamento delle sorgenti dell'attacco;
5. La durata degli attacchi è molto breve il che rende limitato il tempo di risposta.

2. Rilevamento degli attacchi DDoS basato su Support Vector Machine (SVM)

Il controller SDN è responsabile per l'inoltramento, la gestione delle decisioni di quest'ultimo e la raccolta delle informazioni sul traffico degli switch. Nello switch SDN, la principale struttura per il controllo della gestione dell'inoltramento è la flow table. L'SDN gestisce il traffico rilevante dagli ingressi dalla ricerca degli ingressi nella flow table, dove ogni ingresso può inoltrare un pacchetto ad una a più interfacce.

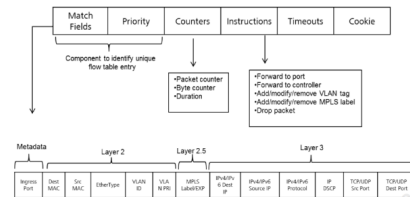


Figure 1. Flow table structure.

Ogni ingresso include un campo per l'header, un contatore e le istruzioni.

Il rilevamento dell'attacco consiste nella raccolta dello stato del flow, l'estrazione dei valori caratteristici e il giudizio del classificatore.

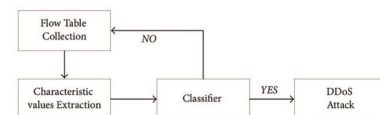


Figure 2. Attack detection process.

Viene inviato periodicamente la richiesta della tabella dei flow allo switch Openflow e invia la risposta con le informazioni. I valori caratteristici estratti sono i principali per la rilevazione degli attacchi DDoS e compongono una sestupla. Tale sestupla viene classificata usando un algoritmo SVM-based in modo da distinguere il traffico normale e quello anormale.

2.1. Raccolta dello stato dei flow

Nell'ambiente SDN, la collezione dell'informazione vengono raccolte attraverso un protocollo Openflow. Lo switch risponde periodicamente al messaggio `onp_flow_stats_request` inviato dal controller.

Per collezionare i dati basta il comando `sudo ovs-ofctl dump-flows s1`.

2.2. Estrazione dei valori caratteristici

Quando vi è un attacco DDoS, la rete crea randomicamente un grande numero di indirizzi IP di origine per l'invio di pacchetti di una certa dimensione per colpire l'obiettivo. Nella rete, il flow dell'attacco mostra una certa similitudine, regolarità, e quindi può essere rilevato tramite l'analisi dei valori caratteristici. I sei valori sono presentati di seguito.

1. **Velocità delle risorse IP (SSIP)** è il numero degli indirizzi IP per unità di tempo

$$SSIP = \frac{Sum_IP_{src}}{T},$$

dove Sum_IP_{src} è il numero degli IP e T l'intervallo di campionamento. In caso di attacco, si generano un grande numero di indirizzi IP sorgente per l'invio dei pacchetti e l'incremento di tale numero è esponenziale.

2. **Velocità del porto sorgente (SSP)** è il numero di porte sorgenti per unità di tempo

$$SSP = \frac{Sum_port_{src}}{T},$$

dove Sum_port_{src} è il numero di porti sorgente. Quando vi è un attacco, una grande quantità di numeri di porto viene generato randomicamente.

3. **Deviazione Standard per il Flow Packets (SDFP)**, che altro non è la deviazione standard del numero di pacchetti nel periodi T :

$$SDFP = \sqrt{\frac{1}{N} \sum_{i=1}^N (packets_i - Mean_packets)^2}$$

dove $Mean_packets = \frac{1}{N} \sum_{i=1}^N packets_i$ rappresenta il numero medio dei pacchetti nel periodo T . N è il numero dei flussi di ingresso nel periodo T . N è il numero totale dei flussi in ingresso per periodo. In caso di attacco i pacchetti sono relativamente piccoli e quindi la deviazione standard del flusso è più piccola del normale.

4. **Deviazione del Flow Bytes (SDFB)**, che altro non è che la deviazione standard del numero di bits per periodo:

$$SDFB = \sqrt{\frac{1}{N} \sum_{i=1}^N (bytes_i - Mean_bytes)^2}$$

dove $Mean_bytes = \frac{1}{N} \sum_{i=1}^N bytes_i$ rappresenta il numero medio di bit per periodo. In caso di attacco, per ridurre il carico dei pacchetti, l'attaccante invia pacchetti di pochi bit e così facendo la deviazione diminuisce più del normale.

5. **La velocità dei flussi d'ingresso (SFE)** rappresenta il numero dei flussi per unità di tempo

$$SFE = \frac{N}{T}$$

in caso di attacco, il numero dei flussi in ingresso per unità di tempo cresce esponenzialmente.

6. **Rapporto della coppia di flusso (RFP)**, descrivente il rapporto tra flussi in ingresso che interagenti sul numero totale dei flussi:

$$RPF = \frac{2 \cdot Pair_Sum}{N}$$

dove $Pair_Sum$ è il numero dei flussi interagenti in ingresso. Quando vi è un attacco, i flussi in ingresso inviati all'host destinatario nel periodo T crescono drasticamente; il destinatario non può rispondere in tempo e quindi molti flussi di interazione in ingresso vengono scartati.

3. Classificatore

È possibile vedere il rilevazione di un attacco come un problema della classificazione, ovvero classificare i dati raccolti e giudicare se nella rete è presente un traffico normale o anormale.

Il processo di classificazione quindi consiste nell'estrazione della sestupla e con essa valutare il traffico.

3.1. SVM

Come modello di machine learning viene utilizzato il Support Vector Machine (**SVM**). È un metodo di apprendimento supervisionato usato per la classificazione, regressione e il rilevamento di valori anormali. I vantaggi dell'SVM sono:

1. Efficacia in spazi ad alta dimensione.
2. Efficacia quando il numero di dimensione è maggiore del numero dei campioni.
3. Utilizzo di un sottoinsieme di punti di training nelle funzione di decisione, con un utilizzo efficiente della memoria.
4. Versatilità.

SVM si applica con grande accuratezza al rilevamento di un attacco DDoS.

Prima di tutto, la tabella di flusso in ingresso in uno switch viene campionata con un certo intervallo T e i valori caratteristici di tale tabella vengono calcolati per ogni campione in modo da ottenere l'insieme di campioni Z , il quale si esprime come $Z = (X, Y)$, dove X rappresenta la sestupla ricavata dalla tabella di flusso in ingresso, Y è il vettore delle categorie corrispondenti ad X : 0 rappresenta uno stato normale della rete, e 1 rappresenta uno stato d'attacco della rete.

4. Applicazione

4.1. Topologia

La topologia scelta per la sperimentazione è la seguente:

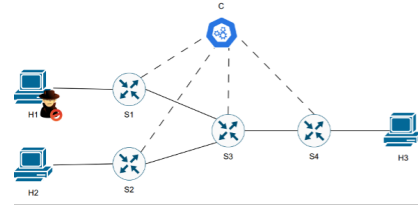


Figure 3. Topology

Gli host sono H1, H2 e H3, dei quali H1 è l'host attaccante. Gli switch sono 4, S1, S2, S3 e S4. Il controller C permette il monitoraggio della nostra rete.

4.2. Controller

Il Controller è realizzato come segue:

```
1 from ryu.base import app_manager
2 from ryu.controller import ofp_event
3 from ryu.controller.handler import
   CONFIG_DISPATCHER, MAIN_DISPATCHER,
   set_ev_cls
4 from ryu.ofproto import ofproto_v1_3
5 from ryu.lib.packet import packet, ethernet,
   ether_types
6
7 from ryu.lib.packet import in_proto, ipv4, icmp,
   tcp, udp
8
9 class Controller(app_manager.RyuApp):
10     OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
11
12     def __init__(self, *_args, **_kwargs):
13         super(Controller, self).__init__(*_args,
14                                         **_kwargs)
15         self.mac_to_port = {}
16
17     @set_ev_cls(ofp_event.EventOFPSwitchFeatures,
18               CONFIG_DISPATCHER)
19     def switch_features_handler(self, ev):
20         datapath = ev.msg.datapath
21         ofproto = datapath.ofproto
22         parser = datapath.ofproto_parser
23
24         match = parser.OFPMatch()
25         actions = [parser.OFPActionOutput(
26                     ofproto.OFPP_CONTROLLER,
27                     ofproto.OFPCML_NO_BUFFER)]
28         self.add_flow(datapath, 0, match,
29                       actions)
30
31     def add_flow(self, datapath, priority, match,
32                 actions, buffer_id=None):
33         ofproto = datapath.ofproto
34         parser = datapath.ofproto_parser
35
36         inst = [parser.OFPInstructionActions(
37                     ofproto.OFPIT_APPLY_ACTIONS,
```

```

actions)]

    if buffer_id:
        mod = parser.OFPFlowMod(datapath=
            datapath, buffer_id=buffer_id, priority=
            priority, match=match, instructions=inst)
    else:
        mod = parser.OFPFlowMod(datapath=
            datapath, priority=priority, match=match,
            instructions=inst)

    datapath.send_msg(mod)

def send_packet_out(self, msg, actions):
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    in_port = msg.match['in_port']

    data = None
    if msg.buffer_id == ofproto.
OFP_NO_BUFFER:
        data = msg.data
        out = parser.OFPPacketOut(datapath=
            datapath, buffer_id=msg.buffer_id, n_port=
            in_port, actions=actions, data=data)
        datapath.send_msg(out)

@set_ev_cls(ofp_event.EventOFPacketIn,
MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    if ev.msg.msg_len < ev.msg.total_len:
        self.logger.debug("packet truncated:
only %s of %s bytes", ev.msg.msg_len, ev.
msg.total_len)

    msg = ev.msg
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    in_port = msg.match['in_port']

    pkt = packet.Packet(msg.data)
    eth = pkt.get_protocols(ethernet.
ethernet)[0]

    if eth.ethertype == ether_types.
ETH_TYPE_LLDP:
        return
    dl_dst = eth.dst
    dl_src = eth.src

    dpid = datapath.id
    self.mac_to_port.setdefault(dpid, {})

    self.logger.info("packet in %s %s %s %s"
, dpid, dl_src, dl_dst, in_port)

    self.mac_to_port[dpid][dl_src] = in_port

    if dl_dst in self.mac_to_port[dpid]:
        out_port = self.mac_to_port[dpid][
dl_dst]
    else:
        out_port = ofproto.OFPP_FLOOD

    actions = [parser.OFPActionOutput(
out_port)]

    if out_port != ofproto.OFPP_FLOOD:
        if eth.ethertype == ether_types.
ETH_TYPE_IP:
            ip = pkt.get_protocol(ipv4.ipv4)
            srcip = ip.src
            dstip = ip.dst
            protocol = ip.proto

            match = parser.OFPMatch(in_port=
in_port, eth_dst=dl_dst, eth_src=dl_src)

```

```

        if protocol == in_proto.
IPPROTO_ICMP:
            icmp_info = pkt.get_protocol
(icmp.icmp)
            print(icmp_info.type)
            match = parser.OFPMatch(
eth_type=ether_types.ETH_TYPE_IP, ipv4_src=
srcip, ipv4_dst=dstip, eth_src=dl_src,
eth_dst=dl_dst, in_port=in_port, ip_proto=
protocol)

        elif protocol==in_proto.
IPPROTO_UDP:
            u = pkt.get_protocol(udp.udp
)
            match = parser.OFPMatch(
eth_type=ether_types.ETH_TYPE_IP, ipv4_src=
srcip, ipv4_dst=dstip, eth_src=dl_src,
eth_dst=dl_dst, in_port=in_port, ip_proto=
protocol, udp_src=u.src_port, udp_dst=u.
dst_port)

        elif protocol==in_proto.
IPPROTO_TCP:
            t = pkt.get_protocol(tcp.tcp
)
            tcp_src = t.src_port
            tcp_dst = t.dst_port
            match = parser.OFPMatch(
eth_type=ether_types.ETH_TYPE_IP, ipv4_src=
srcip, ipv4_dst=dstip, eth_src=dl_src,
eth_dst=dl_dst, in_port=in_port, p_proto=
protocol, tcp_src=tcp_src, tcp_dst=tcp_dst)

            if msg.buffer_id != ofproto.
OFP_NO_BUFFER:
                self.add_flow(datapath, 1,
match, actions, msg.buffer_id)
                return
            else:
                self.add_flow(datapath, 1,
match, actions)

            data = None
            if msg.buffer_id == ofproto.
OFP_NO_BUFFER:
                data = msg.data

            out = parser.OFPPacketOut(datapath=
datapath, buffer_id=msg.buffer_id, in_port=
in_port, actions=actions, data=data)
            datapath.send_msg(out)

```

Code 1. controller.py

Tale codice implementa un semplice controller della libreria RYU.

4.3. Machine Learning

Per il modello SVM il codice è il seguente:

```

1 import numpy as np
2 import pandas as pd
3 from sklearn.model_selection import
train_test_split
4 from sklearn.preprocessing import StandardScaler
5 from sklearn.svm import SVC
6 from sklearn.metrics import confusion_matrix,
classification_report
7 import joblib
8
9 dataset = pd.read_csv('/home/antonio/Desktop/NCI
/code/data/dataset.csv') #import dataset
10
11 X = dataset.drop('Class', axis=1) #splitting
into features and label
12 y = dataset['Class']
13
14 # Splitting del datasetn in train_set e test_set
15 X_train, X_test, y_train, y_test =

```

```

16     train_test_split(X, y, test_size=0.15,
17                      random_state=0)
18 # Scaling
19 scaler = StandardScaler()
20 X_train = scaler.fit_transform(X_train)
21 X_test = scaler.fit_transform(X_test)
22 # SVM fitting with train set
23 classifier = SVC(kernel='linear', random_state
24                 =0)
25 classifier.fit(X=X_train, y=y_train)
26 # Testing
27 y_pred = classifier.predict(X_test)
28 # Cofusion matrix
29 cm = confusion_matrix(y_test, y_pred)
30 cr = classification_report(y_test, y_pred)
31 # Print
32 print(y_pred)
33 print(cm)
34 print(cr)
35 # Exprot model
36 filename = '/home/antonio/Desktop/NCI/code/ml/
37           model.sav'
38 joblib.dump(classifier, filename)
39 print('Model exported!')

```

Code 2. svm.py

Tale codice implementa il codice per la generazione di un modello per l'identificazione di un attacco.

Il modello viene addestrato su un dataset (dataset.csv nella cartella data) il quale presenta una per ogni sestupla di valori di flusso, indica una classe il cui valore corrisponde a 0 se i valori corrispondenti rappresentano uno stato normale della rete, mentre corrisponde ad 1 se i valori corrispondenti rappresentano una rete sotto attacco.

Una volta addestrato il modello, tramite la libreria joblib, viene salvato, per il futuro utilizzo.

(dataset from : [Link](#))

4.4. Topologia

```

1  #!/usr/bin/python
2  import threading
3  import random
4  import time
5  from mininet.log import setLogLevel, info
6  from mininet.topo import Topo
7  from mininet.net import Mininet, CLI
8  from mininet.node import OVSKernelSwitch, Host
9  from mininet.link import TCLink, Link
10 from mininet.node import RemoteController #
11     Controller
12 import time
13
14 class Environment(object):
15     def __init__(self):
16         "Create a network."
17         self.net = Mininet(controller=
18             RemoteController, link=TCLink)
19         info("*** Starting controller\n")
20
21         c1 = self.net.addController( 'c1',
22             controller=RemoteController, ip = '127.0.0.1
23             ', protocol='tcp', port=6633) #Controller
24         c1.start()
25
26         info("*** Adding hosts\n")
27         self.h1 = self.net.addHost('h1', mac='
28             00:00:00:00:00:01', ip='10.0.0.1') #host
29         malware
30         self.h2 = self.net.addHost('h2', mac='
31             00:00:00:00:00:02', ip='10.0.0.2')

```

```

26         self.h3 = self.net.addHost('h3', mac='
27             00:00:00:00:00:03', ip='10.0.0.3')
28
29         info("*** Adding switches\n")
30         self.s1, self.s2, self.s3, self.s4 = [
31             self.net.addSwitch(s, failMode="standalone")
32             for s in ('s1', 's2', 's3', 's4')]
33
34         info("*** Adding links\n")
35         for h, s in [(self.h1, self.s1), (self.
36             h2,self.s2), (self.h3, self.s4)]:
37             self.net.addLink(h, s)
38
39         self.net.addLink(self.s1, self.s3)
40         self.net.addLink(self.s2, self.s3)
41         self.net.addLink(self.s3, self.s4)
42
43         info("*** Starting network\n")
44         self.net.build()
45         self.net.start()
46
47 ...
48 if __name__ == '__main__':
49
50     setLogLevel('info')
51     info('starting the environment\n')
52     env = Environment()
53
54     info("*** Running CLI\n")
55     CLI(env.net)
56     setLogLevel('info')
57
58     env.net.stop()

```

Code 3. topology.py

Utilizzando Mininet, è stata realizzata la topologia da analizzare. Il codice fa quanto segue:

1. Viene aggiunto il controller, visto nel precedente codice.
2. Vengono aggiunti gli host, dove utilizziamo h1 come attaccante della rete (anche se, con il comando utilizzato per simulare l'attacco, è possibile scegliere come attaccante uno qualsiasi dei tre host).
3. Vengono aggiunti gli switch, s1, s2, s3 e s4.
4. Si effettuano i collegamenti tra i vari elementi della rete.

Eseguendo tale codice, verrà costruita e inizializzata la topologia in questione, con la quale potremmo interagire tramite il terminale di Mininet.

Eseguendo il comando exit, verrà eliminata l'intera topologia e l'esecuzione termina.

4.5. Tracker

4.5.1. warning.py

```

1  import warnings
2
3  warnings.filterwarnings('ignore', category=
4      DeprecationWarning)
5  def warn(*args, **kwargs):
6      pass
7  warnings.warn = warn

```

Code 4. warning.py

Questo primo codice ci permette di ignorare tutti i warning che si hanno durante l'esecuzione del tracking in modo da avere un terminale più pulito e chiaro.

4.5.2. tuples.py

```

1  import numpy as np
2  import csv
3
4  #Flow table information extracted

```

```

5 #NXST_FLOWreply(xid = 0      4) : cookie = 0      0,
6 #duration = 21.098s,
7 #table = 0,
8 #n_packets = 1,
9 #n_bytes = 42,
10 #idle_age = 21, #
11 #priority = 65535, tcp,
12 #in_port = 1,
13 #dl_src = 00 : 00 : 00 : 00 : 00 : 01,
14 #dl_dst = 00 : 00 : 00 : 00 : 00 : 02,
15 #nw_src = 95.141.141.242,
16 #tp_src=3786,
17 #tp_dst=0
18
19 # SDFP
20 packets_csv = np.genfromtxt('packets.csv',
21                             delimiter=",")
22 dt_packets = packets_csv[:,]
23 sdfp = np.std(dt_packets)
24
25 #SDFB
26 bytes_csv = np.genfromtxt('bytes.csv', delimiter
27                             =",")
28 dt_bytes = bytes_csv[0]
29 sdfb = np.std(dt_bytes)
30
31 # nIP & SSIP
32 n_ip = np.prod(dt_bytes.shape)
33 ssid = n_ip // 3
34
35 # SFE
36 sfe = n_ip // 3
37
38 # RFIP
39 file_one = None
40 file_two = None
41
42 with open('ipsrc.csv', 'r') as f1, open('ipdst.
43 csv', 'r') as f2:
44     file_one = f1.readlines()
45     file_two = f2.readlines()
46
47 with open('intflow.csv', 'w') as f:
48     for line in file_one:
49         if line not in file_two:
50             f.write
51
52 with open('intflow.csv') as f:
53     reader = csv.reader(f, delimiter=',')
54     dt = list(reader)
55     row_count_non_int = len(dt)
56
57 rfip = abs(float(n_ip - row_count_non_int)/n_ip)
58
59 headers = ['SSIP', 'SDFP', 'SDFB', 'SFE', 'RFIP'
60 ]
61 features = [ssid, sdfp, sdfb, sfe, rfip]
62
63 with open('/home/antonio/Desktop/NCI/code/
64 tracker/rt_data.csv', 'w') as f:
65     cursor = csv.writer(f, delimiter=',')
66     cursor.writerow(headers)
67     cursor.writerow(features)
68
69 f.close()

```

Code 5. tuples.py

Tale file di codice durante l'esecuzione preleva i dati dai rispettivi files per il calcolo della sestupla. Una volta effettuato il calcolo, i dati vengono scritti in un file denominato `rt_data.csv`.

4.5.3. inspector.py

```

1 import numpy as np
2 import pandas as pd
3 from sklearn.model_selection import
4     train_test_split
5 from sklearn.preprocessing import StandardScaler
6 from sklearn.svm import SVC

```

```

6 from sklearn.metrics import confusion_matrix,
7     classification_report
8 import joblib
9 import warning
10
11 filename = '/home/antonio/Desktop/NCI/code/ml/
12     model.sav'
13 classifier = joblib.load(filename)
14 dt_realtime = pd.read_csv('/home/antonio/Desktop
15 /NCI/code/tracker/rt_data.csv')
16 dt_realtime.fillna(0, inplace=True)
17 result = classifier.predict(dt_realtime)
18
19 with open('/home/antonio/Desktop/NCI/code/
20     tracker/.result', 'w') as f:
21     f.write(str(result[0]))

```

Code 6. inspector.py

Tale codice si occupa del prelevare il modello ed utilizzarlo per la classificazione dei dati pervenuti dal controller. Il risultato ottenuto viene salvato in un file denominato `.result`.

4.5.4. collector.py

```

1 import subprocess
2 import time
3 import csv
4 import os
5
6 n = 4 # number of switches
7 iterations = 2000
8
9 #Flow table information extracted
10 #NXST_FLOWreply(xid = 0      4) : cookie = 0      0,
11 #duration = 21.098s,
12 #table = 0,
13 #n_packets = 1,
14 #n_bytes = 42,
15 #idle_age = 21, #
16 #priority = 65535, tcp,
17 #in_port = 1,
18 #dl_src = 00 : 00 : 00 : 00 : 00 : 01,
19 #dl_dst = 00 : 00 : 00 : 00 : 00 : 02,
20 #nw_src = 95.141.141.242,
21 #tp_src=3786,
22 #tp_dst=0
23
24 del_flows = []
25
26 def run_command(command):
27     try:
28         result = subprocess.run(command, shell=
29             True, capture_output=True, text=True, check=
30             True)
31         return result.stdout
32     except subprocess.CalledProcessError as e:
33         print(f"Command '{command}' failed with
34             error: {e.stderr}")
35         return None
36
37 def extract_field(data, field_name, delimiter=","
38                 , field_delimiter=""):
39     extracted = []
40     for row in data.splitlines():
41         fields = row.split(delimiter)
42         for field in fields:
43             if field_name in field:
44                 extracted.append(field.split(
45                     field_delimiter)[1])
46     return extracted
47
48 def save_to_csv(data, file_path):
49     with open(file_path, 'w', newline='') as
50         file:
51         writer = csv.writer(file)
52         writer.writerow(data)
53
54 def main():
55     for i in range(1, iterations + 1):

```



```

50     for j in range(1, n + 1):
51         print(f"Inspection no. {i} at s{j}")
52
53         # Extract essential data from raw
54         data
55         raw_data = run_command(f"sudo ovs-
56         ofctl dump-flows s{j}")
57         if raw_data is None:
58             continue
59
60         with open('raw', 'w') as raw_file:
61             raw_file.write(raw_data)
62
63         flowentries = [line for line in
64         raw_data.splitlines() if "nw_src" in line]
65         with open('flowentries.csv', 'w') as
66         flowentries_file:
67             flowentries_file.write("\n".join
68             (flowentries))
69
70         packets = extract_field(raw_data, "
71         packets")
72         bytes_data = extract_field(raw_data,
73         "bytes")
74         ipsrc = extract_field(raw_data, "
75         nw_src")
76         ipdst = extract_field(raw_data, "
77         nw_dst")
78
79         # Check if there are no traffics in
80         the network at the moment
81         if not packets or not bytes_data or
82         not ipsrc or not ipdst:
83             state = 0
84         else:
85             save_to_csv(packets, 'packets.
86             csv')
87             save_to_csv(bytes_data, 'bytes.
88             csv')
89             save_to_csv(ipsrc, 'ipsrc.csv')
90             save_to_csv(ipdst, 'ipdst.csv')
91
92             subprocess.run("python3 tuples.
93             py", shell=True)
94             subprocess.run("python3
95             inspector.py", shell=True)
96
97             with open('.result', 'r') as
98             result_file:
99                 state = int(result_file.read
100                 ()).strip())
101
102             if state == 1:
103                 print(f"Network is under attack
104                 occurring at s{j}")
105                 default_flow = run_command(f"
106                 sudo ovs-ofctl dump-flows s{j} | tail -n 1")
107                 run_command(f"sudo ovs-ofctl del
108                 -flows s{j}")
109                 run_command(f"sudo ovs-ofctl add
110                 -flow s{j} \"{default_flow.strip()}\")
111                 run_command(f"killall hping3")
112                 print(default_flow)
113                 time.sleep(3)
114
115 if __name__ == "__main__":
116     os.makedirs("data", exist_ok=True)
117     main()

```

Code 7. collector.py

Tale file, può essere visto come il main del tracker, in quanto è quello che esegue anche gli script precedenti.

Viene scelto come numero di iterazioni 2000, ma è un valore puramente arbitrario, mentre n è il numero degli switch.

Vengono definite tre funzioni inizialmente:

1. `run_command`: permette di eseguire un comando da terminale, passandolo come parametro a tale funzione.
2. `extract_field`: permette di estrarre i singoli parametri dai

rispetti campi una volta interrogato il controller.

3. `save_to_csv`: salva i dati in un file .csv.

Il main esegue due cicli `for` innestati, in modo da eseguire i controlli per ogni switch tante volte quanto il numero di iterazioni scelto.

Eseguendo il comando `sudo ovs-ofctl dump-flows s{j}` si effettua vengono estratti i dati rilevati dal controller e poi salvati su un file `raw`.

Vengono poi selezionati solo i flussi in ingresso e salvati nel file `flowentries.csv`. Vengono da qui estratti e creati i file relativi a pacchetti, bytes, ipsrc e ipdst.

Eseguendo gli script `tuples.py` e `inspector.py` può essere letto il risultato della classificazione.

Nel caso di un attacco vengono eseguiti i comandi per eliminare il flusso nel rispettivo switch, facendo ritornare la rete ad uno stato normale.

4.6. Esecuzione

4.6.1. Modello SVM

Per la creazione del modello SVM, ci si reca nella cartella tramite terminale nella cartella `m1` e si esegue il seguente comando:

```
python3 svm.py
```

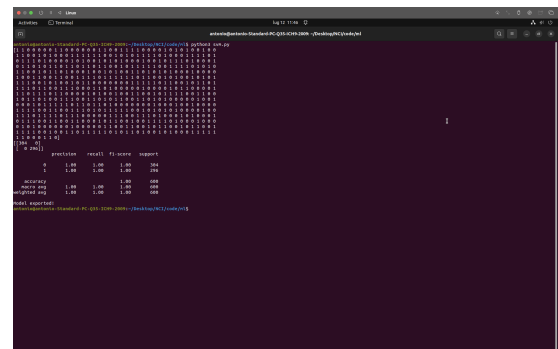


Figure 4. svm.py execution

Una volta eseguito, verrà salvato il modello: `model.sav`

4.6.2. Rilevamento attacco nella topologia

Dapprima si esegue lo script per creare la topologia e per poi aprire la shell di Mininet. Il comando da eseguire è:

```
sudo python3 topology.py
```

Si esegue il controller lanciandolo tramite il comando:

```
ryu-manager controller.py
```

E infine si esegue il tracker:

```
sudo python3 collector.py
```

Per simulare un generico un normale traffico nella rete utilizziamo il comando `ping`: nella shell di Mininet si esegue:

```
h2 ping h3
```

Si può vedere nell'immagine come il tracker esegue i vari cicli ma non rileva nulla di anomalo.

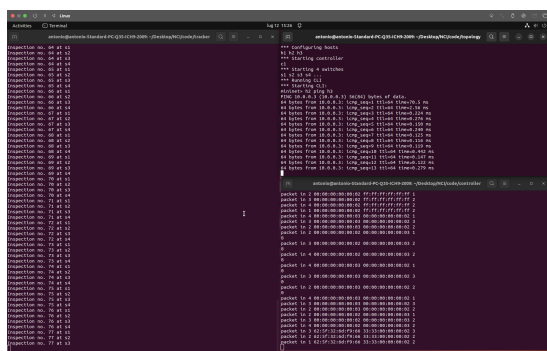


Figure 5. Normal traffic

Per simulare un attacco DDoS è possibile eseguire tale comando nella shell di Mininet:

```
h1 hping3 --rand-source --flood h2
```

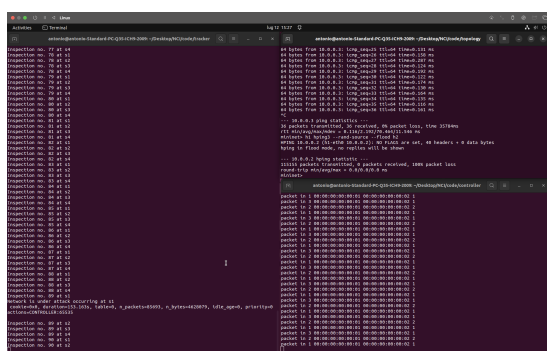


Figure 6. Attack

Come si vede nell'immagine, il tracker rileva l'attacco e cancella il flusso.

5. Conclusioni e sviluppi futuri

In tale progetto si è visto come è possibile applicare l'intelligenza artificiale nel mondo della sicurezza informatica: rappresenta un approccio molto innovativo con il quale rilevare gli attacchi ad una qualsiasi rete. Infatti sia che la topologia cambi, sia che l'attaccante effettui azioni per celare la sua identità, il sistema riesce ad individuare comunque il traffico malevolo e eseguire un'azione per la salvaguardia della rete.

Per tale progetto è possibile migliorare la gestione della rete dopo aver rilevato un attacco e la possibilità di tracciare l'host attaccante.

References

- [1] J. Ye, X. Cheng, J. Zhu, L. Feng, and L. Song, "A ddos attack detection method based on svm in software defined network," *Security and Communication Networks*, vol. 2018, no. 1, p. 9 804 061, 2018. DOI: <https://doi.org/10.1155/2018/9804061>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1155/2018/9804061>. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1155/2018/9804061>.
- [2] [Online]. Available: <https://scikit-learn.org/stable/modules/svm.html>.