HTTPScout

A Machine Learning based Countermeasure for HTTP Flood Attacks in SDN

Mohammadi, Reza; Lal, Chhagan; Conti, Mauro

DOI
10.1007/s10207-022-00641-3

Publication date
2022

Document Version
Final published version

Published in
International Journal of Information Security

Citation (APA)
Mohammadi, R., Lal, C., & Conti, M. (2022). HTTPScout: A Machine Learning based Countermeasure for HTTP Flood Attacks in SDN. *International Journal of Information Security*, *22 (2023)*(2), 367-379. https://doi.org/10.1007/s10207-022-00641-3

International Journal of Information Security (2023) 22:367–379
https://doi.org/10.1007/s10207-022-00641-3

REGULAR CONTRIBUTION

HTTPScout: A Machine Learning based Countermeasure for HTTP Flood Attacks in SDN

Reza Mohammadi[1] · **Chhagan Lal**[2] · **Mauro Conti**[2,3]

Published online: 3 December 2022

Abstract
Nowadays, the number of Distributed Denial of Service (DDoS) attacks is growing rapidly. The aim of these type of attacks is to make the prominent and critical services unavailable for legitimate users. HTTP flooding is one of the most common DDoS attacks and because of its implementation in application layer, it is difficult to detect and prevent by the current defense mechanisms. This attack not only makes the web servers unavailable, but consumes the computational resources of the network equipment and congests communication links. Recently, the advent of Software Defined Networking (SDN) paradigm has enabled the network providers to detect and mitigate application layer DDoS attacks such as HTTP flooding. In this paper, we propose a defense mechanism named HTTPScout which leverages the benefits of SDN together with Machine Learning (ML) techniques to detect and mitigate HTTP flooding attack. HTTPScout is implemented as a security module in RYU controller and monitors the behavior of HTTP traffic flows. Upon detecting a malicious flow, it blocks the source of the attack at the edge switch and preserves the network resources from the adversarial effects of the attack. Simulation results confirm that HTTPScout brings a significant improvement of 64% in bandwidth consumption and 80% in the number of forwarding rules compared to normal SDN.

Keywords SDN · DDoS · Flooding attack · Machine learning

1 Introduction

In the last few decades, the number of users of the Internet has remarkably increased [1]. Consequently, companies and enterprises deliver their various services to the people around the world over the Internet. Most of these services are delivered to consumers using HTTP protocol, which is a well-known application layer protocol [2], and users take use of it for web surfing, electronic shopping, researching and so forth. Unfortunately, this protocol has some vulnerabilities which are exploited by attackers to make the servers unavailable for legitimate users. HTTP flooding attack is one of the most important Distributed Denial of Service (DDoS) attacks in which a group of attackers sends a huge number of HTTP requests to overwhelm a targeted server [3]. As a result, the targeted server can not deliver services to legitimate users, and finally causes users dissatisfaction. Another critical issue regarding the HTTP flooding attack is that, in some cases, the attackers perform the attack at a slow rate, which makes it difficult to detect. In addition to the adversarial effect on the web servers, this attack also consumes network bandwidth and resources. Unfortunately, due to this attack is being launched in the application layer, most of the current defense mechanisms which are implemented in the network or transport layers are unable to detect it. Besides, most of these mechanisms can detect the attack at the servers' sides [4], [5], which only preserves the web servers and can not prohibit network bandwidth saturation.

Recently, the advent of Software Defined Networking (SDN) paradigm has enabled network administrators to implement their own security mechanisms to defend against attacks [6]. As a matter of fact, the separation between the control and data planes in SDN facilitates network programmability and makes it convenient to leverage the benefits

✉ Reza Mohammadi
r.mohammadi@basu.ac.ir

Chhagan Lal
c.lal@tudelft.nl

Mauro Conti
conti@math.unipd.it

[1] Bu-Ali Sina University, Hamedan, Iran

[2] TU Delft, Delft, The Netherlands

[3] University of Padua, Italy/TU Delft, Padua, Italy

of intelligent algorithms [7], [8]. For this reason, in recent years, applying Machine Learning (ML) techniques to detect security attacks in SDN has attracted much attention by the researchers [9], [10],[11]. Since in SDN, the controller has a global view of the network, it can gather various statistics about the network traffic flows. By applying ML techniques to these statistics, it can be possible to distinguish abnormal traffic flows from the normal ones [12]. After the detection of the abnormal traffic flow, the SDN controller can block the origin of the malicious behavior, and in this way, it mitigates the adversarial effects of the attack on the targeted servers as well as network equipments.

In this paper, we aim to propose HTTPScout as a new security mechanism which leverages the benefits of SDN together with ML algorithms to detect and mitigate HTTP flooding attack. HTTPScout is implemented in the SDN controller and makes use of supervised ML algorithms to train both normal and abnormal behaviors from a dataset. It also monitors all on-the-fly HTTP flows on the network, and extracts some important features of each flow. Then, it applies the ML algorithms to the extracted features. With respect to its knowledge learned from the dataset, if it detects an HTTP flow as an attack, it first identifies and then blocks the origin of the attack by installing rules on the edge switch connected to the attacker(s). HTTPScout not only preserves the targeted web servers from the attack, it also mitigates adverse impact on network bandwidth consumption. In particular, our main contributions can be summarized as follows.

- We propose a new ML based security module named HTTPScout in SDN to detect and mitigate HTTP flooding attacks. In particular, we implement and investigate different ML algorithms for HTTPScout to choose the best suited for detecting the HTTP flooding attack.
- We implement HTTPScout as a security module on RYU controller to make it more secure against HTTP flooding attacks. We also test HTTPScout performance over the different rate of attacks.

The rest of this paper is organized as follows. In Section 2, we review ML techniques related work for HTTP flooding in SDN. Section 3 presents some preliminaries about HTTP flooding attacks and ML algorithms used in HTTPScout. Section 4 presents our proposed solution and explains HTTPScout in detail. In Section 5, simulation settings and the performance evaluation are presented and discussed. Finally, the paper is concluded in Section 7.

2 Related work

In this section, some state-of-art related work in ML based defense mechanisms in SDN are reviewed. Lingfeng and Hui proposed an SDN framework which makes use of ML techniques such as Support Vector Machine (SVM) for detecting and defending against HTTP flood attack [13]. Their framework first collects some features of traffic characteristics and then uses SVM to identify the attack. Finally, the controller adjusts the forwarding policy to resist the DDoS attack's source. One of the weaknesses of their method is that, they have only used DoS and normal items of KDD99 dataset [14] to train SVM algorithm. Generally, in most situations, DoS does not perfectly reflect the behavior of a real and distributed HTTP flooding attack. Moreover, they have only investigated the accuracy of their method and other important metrics such as F1-Score, attack detection time and so forth. are not studied. In [15], Santos et al. have implemented four ML algorithms (SVM, MLP, Decision Tree, and Random Forest) to classify DDoS attacks in SDN. They have produced their own dataset in a test-bed network with 6 clients. Their simulation results showed that among 23 features of traffic flows, only 11 of them are important for ML algorithms. Moreover, they concluded that the Decision Tree algorithm has better performance in general, because of its lowest time to process and good accuracy. However, their proposed method only detects the DDoS attack and can not identify and block the source of the attack for future. Moreover, because their proposed a general solution to detect some other DDoS attacks, it is unclear the accuracy of their solution for HTTP Flooding attack.

Sahoo and Iqbal have proposed a security mechanism which leverages seven different ML techniques to classify and predict DDoS attacks such as Smurf, UDP flood, and HTTP flood [16]. Their proposed method consists of three modules for monitoring, extracting features and classifying the flows, respectively. Simulation results confirmed that in their solution, Linear Regression (LR) algorithm achieves better accuracy compared to other algorithms. However, they have not implemented the solution on an SDN based test-bed, and only applied the algorithms on a dataset. Sen et al. introduced a DDoS attack detection mechanism using AdaBoosting algorithm [17]. They first generated a dataset comprising various attacks and normal traffic, and then applied different ML techniques. They have concluded that AdaBoosting algorithm outperforms other ML algorithms in terms of accuracy. Nevertheless, they have only used a simple test-bed with a single switch and controller. Furthermore, their method only detects the presence of DDoS attack and does not recognize and block the attacker.

Perez Diaz et al. have introduced a modular and flexible architecture to Identify and Mitigate Low-Rate DDoS Attacks using Machine Learning in SDN [18]. They have trained 6 different ML algorithms with the Canadian Institute of Cybersecurity (CIC) DoS dataset [19], and then implemented the algorithms in a separate Intrusion Detection System (IDS). In their proposed architecture, the SDN con-

troller sends the feature of suspicious traffic flows to the IDS using JSON requests. If the IDS detects an anomaly for the request, it responds the flow is anomalous. Then, the controller increases the probability of blocking for that flow's origin. Finally, if the probability of a source is 100%, the controller installs a rule to block the source of the attack. Simulation results showed that Multi-Layer Perceptron (MLP) achieves better accuracy compared to other ML algorithms in their architecture. Although the architecture of [18] is modular and mitigates the HTTP flooding attack, the number of blocked flows and attack detection time have not been studied in their work. In [20], Li et al. proposed a new model to detect DDoS attack in SDN using SVM algorithm. Their proposed method first extracts several key features from the incoming traffic flows to the controller, and measures the distribution of each feature using Shanon's entropy. Then, SVM algorithm is used to detect the DDoS attack. They also implemented other ML algorithms such as Decision Tree, Naive Bayes, KNN and Random Forest. But, simulation results showed that SVM outperforms other ML algorithms. Nonetheless, they have not clearly explained how their method mitigates the attack. Polat et el. proposed a machine learning schema for detecting DDoS attack in SDN [21]. They first generated a new dataset comprising the important features of normal and DDoS attack flows to train the ML algorithms. After training the algorithms, they applied the algorithms on the SDN controller. Simulation results confirmed that KNN algorithm achieves better performance compared to the other ML algorithms. However, the authors have not discussed how their method can locate the source of attack and blocks it. Moreover, some metrics such as attack detection time have not been studied in their work.

In [22], Dong and Sarem proposed an improved KNN algorithm to detect the DDoS attack in SDN. They have implemented their solution on FloodLight controller on an SDN-based test-bed network and concluded that the performance of KNN is better than other ML techniques such as SVM and Naive Bayes. Nonetheless, their proposed method only detects DDoS attack without any mitigation mechanism. Sanjeetha et al. have introduced a DDoS detection and mitigation mechanism which leverages CatBoost classifier in SDN [23]. Their proposed method not only detects the DDoS attacks, but also can identify and block the source of the attacks. Simulation results confirmed that CatBoost outperforms other ML algorithms. However, their proposed method uses source and destination MAC addresses to block the attacker, which can be discarded by using MAC spoofing technique. Jayasri et al. proposed a DDoS attacks detection mechanism for SDN using Naive Bayes and K-means algorithms [24]. The main drawback of their work is that the proposed method has not been investigated for SDN-based test-bed.

We summarize the aforementioned related works in Table 1. We observe that few works have been proposed to both detect as well as mitigate HTTP flooding in SDN. Therefore, we propose HTTPScout as a security module which not only detect, but also identifies and blocks the source of the attack. Moreover, unlike the existing research works, in addition to accuracy, precision and F-measure metrics, we also evaluate HTTPScout for metrics such as attack detection time and Switch Flow table size, which are important to evaluate the effectiveness of such solutions.

3 Preliminaries

In this section, we first describe HTTP flooding attack in detail. Then, we briefly explain four ML algorithms which we have used for HTTPScout in 3.2.

3.1 HTTP flooding attack

HTTP flooding is a well-known application layer volumetric attack which can be launched by sending a huge number of GET or $POST$ requests toward a targeted web server [4]. In most cases, this attack is performed by a group of attackers or compromised hosts to make the web server crash. Generally, the attack can be performed against the victim in various ways. For example, Slowloris [28], is a popular slow rate HTTP flooding attack in which the attackers establish many HTTP connections to the targeted web server and endeavor to hold the connections open for a long time. To prevent closing the connections, upon sending $POST$ or GET requests, the attackers do not complete the header of these requests [29].

If the number of such incomplete requests is high, the server will be crashed, and also network resources will be exhausted.

3.2 ML algorithms

Selecting a suitable ML algorithm to achieve the best performance for detection of a specific attack is a challenge. HTTPScout has been implemented with four popular and well-known ML algorithms which are briefly explained below.

3.2.1 KNN

k-Nearest Neighbor is a type of supervised ML algorithm used to classify a set of data with different labels into various classes [30]. It is a simple and non-parametric classifier in which an object is assigned to the majority of K nearest neighbors. The important parameters in this algorithm are K value and distance metric. Considering large values for K, decreases the side effects of the noise on the classification. On

Table 1 Comparison of existing ML approaches for HTTP flooding in SDN

References	Year	Algorithm(s)	Dataset	Detect/Block the attack origin	SDN based test-bed	Disadvantages
[18]	2020	J48, Random Tree, REP Tree, Random Forest, MLP, and SVM	CIC DoS dataset 2017[19]	Detect and block	Yes (ONOS Controller)	- Some important metrics such as attack detection time has not been investigated
[23]	2021	Catboost, XGBoost, kNN, LR, Decision Tree	Kaggle [25]	Detect and block	Yes (RYU Controller)	- Mitigation mechanism can be discarded by MAC spoofing attack
[15]	2020	SVM, MLP, Decision Tree, and Random Forest	Generated in a test-bed (developed by the authors)	Only Detect	Yes (POX Controller)	- Some important metrics such as attack detection time has not been investigated - The accuracy of HTTP flooding attack detection is unclear - Unable to locate and block the origin of the attack
[16]	2018	kNN, Naive Bayes, Linear Regression, SVM, ANN, Decision Tree, and Random Forest	Dataset [26]	Only Detect	No	- Not implemented in an SDN based test-bed network - Unable to locate and block the origin of the attack
[22]	2019	kNN, Naive Bayes, SVM	Generated in a test-bed (developed by the authors)	Only Detect	Yes (FloodLight Controller)	- The accuracy of HTTP flooding attack detection is unclear - Unable to locate and block the origin of the attack
[21]	2020	k. SVM, NB, ANN, and KNN	Generated in a test-bed (developed by the authors)	Only Detect	Yes (POX Controller)	- Locating the origin of the attack and mitigation have not clearly been discussed - The accuracy of HTTP flooding attack detection is unclear

Table 1 continued

References	Year	Algorithm(s)	Dataset	Detect/Block the attack origin	SDN based test-bed	Disadvantages
[20]	2018	Naive Bayes, SVM, kNN, Decision Tree, and Random Forest	DARPA1999 data set [27]	Detect and block	Yes (FloodLight Controller)	- Mitigation process not explained clearly - The accuracy of HTTP flooding attack detection is unclear
[24]	2021	Naïve Bayes algorithm and K-means	KDD cup99 [14] [26]	Only Detect	No	- Not implemented in an SDN based test-bed network - Unable to locate and block the origin of the attack
[17]	2020	SVM, MLP, AdaBoost, J48 Decision Tree, Bayes Net, and Random Forest	Generated in a test-bed (developed by the authors)	Only Detect	Yes (sFlow-RT)	- Some important features such as attack detection time has not been investigated - The accuracy of HTTP flooding attack detection is unclear - Unable to locate and block the origin of the attack - The test-bed scenario is very simple
[13]	2018	SVM	KDD99 [14]	Detect and block	Yes (RYU Controller)	- Only accuracy of algorithm is investigated - Only DoS and Normal labels have been used for training which does not reflect HTTP Flooding behavior
HTTPScout (Proposed method)	2022	kNN, Decision Tree, Random Forest,Naive Bayes	CIC DoS dataset 2017 [19]	Detect and block	Yes (RYU Controller)	

the other hand, it blurs the distinction between classes [31]. For the distance metric, in most cases the Euclidean distance is chosen, because of its popularity. KNN is a widely used ML algorithm in network traffic classification to detect abnormal behaviors [32], [33].

3.2.2 Naive bayes

It is a type of supervised ML algorithms based on the Bayes theorem in the probability [34]. In fact, it's goal is to make use of conditional probability to predict the belonging of an object or data point to a certain class. The assumption of Naive Bayes algorithm for a given class variable is that the value of a specific feature is unrelated to the value of any other feature for that class [30].

3.2.3 Decision tree

It is a type of supervised ML algorithm that uses a tree to classify data [30]. In the tree, leaf nodes are the classes of the dataset and each internal node expresses an attribute. The incoming edge for each internal node determines the value of the parent node of that internal node. To make an efficient tree in terms of resource consumption, one solution is to create the tree as small as possible. It is commonly used in the classification of the network traffic to find anomaly behaviors [35], [36].

3.2.4 Random forest

This algorithm is an ensemble learning ML technique in which a combination of multiple decision trees is used to provide the solution [37]. In classification problems, the outcome of this algorithm is the class which is selected by most of the decision trees in the forest. In some cases, the outcome is obtained by taking the average of the output of the decision trees. The goal of random forest is to increase the precision by reducing the overfitting in decision trees. Similar to the aforementioned ML algorithms, it is also widely used in the classification and detection of attacks in the computer networks [38], [39].

4 HTTPScout implementation

In this section, we explain HTTPScout in detail. As HTTPScout is using ML algorithms, we first describe about the dataset, data preprocessing, and feature selection phases. Then, we present the architecture of HTTPScout in section 4.2.

4.1 Machine learning in HTTPScout

As mentioned in Section 3, most of the research works have used Kdd99 [14] as a dataset for training their models. Although this dataset is a well-known and popular for training ML models to classify DDoS attacks, it focuses on general DDoS attacks and slow rate HTTP flooding has not been included in this dataset. For this reason, we use CIC DoS dataset [19] to train the ML models as it comprises the most common types of application layer DoS attacks including slow rate HTTP flooding.

Before training the ML models, we perform preprocessing on the dataset to remove some unnecessary data. First, we only keep benign and slow rate HTTP flooding data and removed other types of DDoS attacks from the dataset. Then, we use Gini index method to calculate information gain to find the most important features. Among 76 features in CIC dataset, we selected 33 features to train the models and other non-significant features were removed from the dataset. The removed features are related to other non-HTTP protocols such as UDP and some flag fields of TCP. Table 2 shows the selected features in our proposed work.

After preprocessing, we first train and then generate the ML models described in Section 3 in Scikit-learn [41], a well-known Python machine learning library. After the model creation, we stored them in separate model files. Upon activating HTTPScout, an administrator determines which model should be used to detect the attack. HTTPScout imports the specified model file and uses it for detecting HTTP flooding attack. This mechanism increases the modularity of HTTPScout and helps to add more ML models to it in the future.

4.2 Architecture

As aforementioned, HTTPScout is implemented as a security module on SDN controller and enables network administrators to defend against HTTP flooding attack. For this purpose, HTTPScout consists of two separate phases as follows.

4.2.1 Attack detection phase

To detect the presence of an attack, the network administrator first determines two parameters: Time window and ML algorithm. The prior denotes the period of the investigation of attack by HTTPScout, and the latter determines the ML algorithm taken by HTTPScout to detect the attack. During each period, HTTPScout monitors the HTTP connections in the network and records the specifications of each connection in a table named *LogTable*. This periodical observance has two benefits: First, it causes less overhead, because instead of checking each HTTP flow separately upon receipt at the controller, HTTPScout checks all flows together at the end

Table 2 List of the selected features from CIC dataset for training [40]

Number	Feature	Description
1	Flow duration	Duration of the flow in Microsecond
2	Packet Length Max	Maximum length of a packet
3	total Fwd Packet	Total packets in the forward direction
4	total Bwd packets	Total packets in the backward direction
5	total Length of Fwd Packet	Total size of packet in forward direction
6	total Length of Bwd Packet	Total size of packet in backward direction
7	Fwd Packet Length Min	Minimum size of packet in forward direction
8	Fwd Packet Length Max	Maximum size of packet in forward direction
9	Fwd Packet Length Mean	Mean size of packet in forward direction
10	Fwd Packet Length Std	Standard deviation size of packet in forward direction
11	Bwd Packet Length Min	Minimum size of packet in backward direction
12	Bwd Packet Length Max	Maximum size of packet in backward direction
13	Bwd Packet Length Mean	Mean size of packet in backward direction
14	Bwd Packet Length Std	Standard deviation size of packet in backward direction
15	Flow Bytes/s	Number of flow bytes per second
16	Flow Packets/s	Number of flow packets per second
17	Flow IAT Mean	Mean time between two packets sent in the flow
18	Flow IAT Std	Standard deviation time between two packets sent in the flow
19	Flow IAT Max	Maximum time between two packets sent in the flow
20	Flow IAT Min	Minimum time between two packets sent in the flow
21	Fwd IAT Min	Minimum time between two packets sent in the forward direction
22	Fwd IAT Max	Maximum time between two packets sent in the forward direction
23	Fwd IAT Mean	Mean time between two packets sent in the forward direction
24	Fwd IAT Std	Standard deviation time between two packets sent in the forward direction
25	Fwd IAT Total	Total time between two packets sent in the forward direction
26	Bwd IAT Min	Minimum time between two packets sent in the backward direction
27	Bwd IAT Max	Maximum time between two packets sent in the backward direction
28	Bwd IAT Mean	Mean time between two packets sent in the backward direction
29	Bwd IAT Std	Standard deviation time between two packets sent in the backward direction
30	Bwd IAT Total	Total time between two packets sent in the backward direction
31	FWD Packets/s	Number of forward packets per second
32	Bwd Packets/s	Number of backward packets per second
33	Packet Length Min	Minimum length of a packet

of a time window to detect the presence of the attack. Second, as described in Table 2, some features of the dataset are related to the time elapsed from the beginning of the flow. Hence, considering the time window in HTTPScout helps to compute accurate values for the features of each HTTP flow.

Typically, in SDN, upon receiving a packet which is not matched with any of the installed forwarding rules on a switch, the switch encapsulates it in an OpenFlow message named *PACKET_IN* and forwards it to the SDN controller [42]. This behavior notifies the controller about incoming packets from a new flow in the network. As shown in Fig. 1, HTTPScout leverages the benefits of SDN capabilities to detect and prevent HTTP flooding attacks. Upon receiving a new packet, it first checks whether the packet is an HTTP request or response. If it is neither, then HTTPScout computes and installs a bidirectional path for that flow on the relevant switches. Otherwise, some important header fields such as input port of the connected switch, source and destination IP addresses, source and destination layer 4 port addresses, and the size and arrival time of the packet are extracted and stored as a record *(In_Port, Switch, Src_IP, Dst_IP, Src_port, Dst_port, Size, Time)* in the *LogTable*. For example, *(3, S5, 10.0.0.1, 10.0.0.5, 9653, 80, 52, 12:36:72.5)* means that an HTTP packet with source IP address (10.0.0.1), destination IP address (10.0.0.5), source transport layer port number (9653), destination transport layer port number (80),

and length (52 byte) at time 12:36:72.5 has been received in input physical port 3 of switch $S5$. Later it is forwarded by the switch toward HTTPScout module in the SDN controller. Then, HTTPScout computes and installs a bidirectional path for this flow on the relevant switches. The installed forwarding rules on the edge switches of each path have two actions. One action determines the output port of the switch for the flow, and the other enforces the switch to send a copy of the incoming packets of that flow toward the controller. This policy enables the HTTPScout to pursue all the packets belonging to a specific HTTP flow. In fact, in addition to the first packet of each HTTP flow, the consequent packets of that flow will be received by the controller. Once the timer expired, HTTPScout extracts the mentioned features in the Table 2 from the *LogTable* for each flow, and then executes the ML model which is previously determined by the administrator for the extracted features. If the ML model detects the attack, HTTPScout starts the mitigation phase.

4.3 Attack mitigation phase

In this phase, HTTPScout identifies the source of the attack. To do this, it retrieves the input port number of the switch which is connected to the source of the attack from *LogTable*. Then, it removes all the relevant forwarding rules belonging to the source of the attack from all network switches. This mechanism empties the useless forwarding rules from the flow tables of the network switches. Thus, it avoids wasting of TCAM memory on the switches. Finally, HTTPScout installs a new rule to block the origin of the attack at the edge switch. As a matter of fact, by applying this policy, HTTPScout prohibits the network resources consumption by the attacker hosts. It is worth emphasizing that, the block rule contains MAC, IP and physical input port number of the attacker. Considering these three match fields in the block rules has two benefits. First, in situations in which the attackers and benign hosts access the same gateway, it correctly distinguishes the attackers. Second, it disarms the attackers if they use spoofing techniques because in this case they only can spoof MAC or IP address, and will not be able to spoof or change their physical input port which is connected to the edge SDN switch. Algorithm 1 shows the pseudocode of HTTPScout.

Algorithm 1 HTTPScout: Detection and Prevention Phase

Input: $PACKET\_IN(P)$
$PACKET\_IN(P)$: New packet arrived to the controller

```
1: LogTable ← []; //A table for logging the features of incoming packets
2: while true do
3:     P ← New_Packet()
4:     if P==HTTP then
5:         LogTable ← LogTable ∪ Extract_Packet_Features();
6:         Install_BiDirectionalPath(P.SrcIP, P.DstIP)  // On edge switches, it installs two rules
7:     end if
8:     if T exceeded then // Timer Expiration
9:         for each f ∈ LogTable do
10:            FE_f ← Extract_Flow_Features(f)
11:            RES ← ML_Algorithm(FE_f);
12:            if RES is Attack then
13:                In_Port ← Find_Connected_Port(f_Switch)
14:                RemoveRulesFromAllSwitches(f_SrcIP)
15:                InstallBlockRule(In_port, f_SrcIP)
16:            end if
17:        end for
18:    end if
19: end while
```

4.4 Computational and space complexity of HTTPScout

The time complexity of HTTPScout depends on the number of malicious hosts and the number of HTTP requests of each host. According to algorithm 1, if we consider n is the number of malicious hosts, m is the number of the request of each host, and k is the number of network switches, with the assumption that all the hosts are attacker. Then, the computational complexity of HTTPScout at the worst case is $O(nm.(1+nm+k+1)) \sim O(n^2m^2+nmk)$. This implies that HTTPScout has polynomial complexity and applying it to defense against HTTP flooding does not lead to considerable overhead on SDN controller. As mentioned in Section 4, HTTPScout stores the important header fields and other related information for each HTTP request as a record in *LogTable*. If we consider each record is s byte, the space complexity of HTTPScout at the worst case will be $O(nms)$. It is worth to mention that although the space and time complexity of HTTPScout is polynomial, in some circumstances where the computational capacity of the SDN controller is limited, and the number of malicious hosts or their requests is high, the overhead might be significant.

5 Performance evaluation

In this section, we conduct a comprehensive simulation study, and analyze the results to evaluate the performance of HTTPScout with different ML algorithms. We compare the performance of HTTScout to normal SDN in the cases with/without the presence of attack. In normal SDN, there is no defense mechanism in place against HTTP flooding attacks, and we will show that how the use of HTTPScout can improve the security of normal SDN.

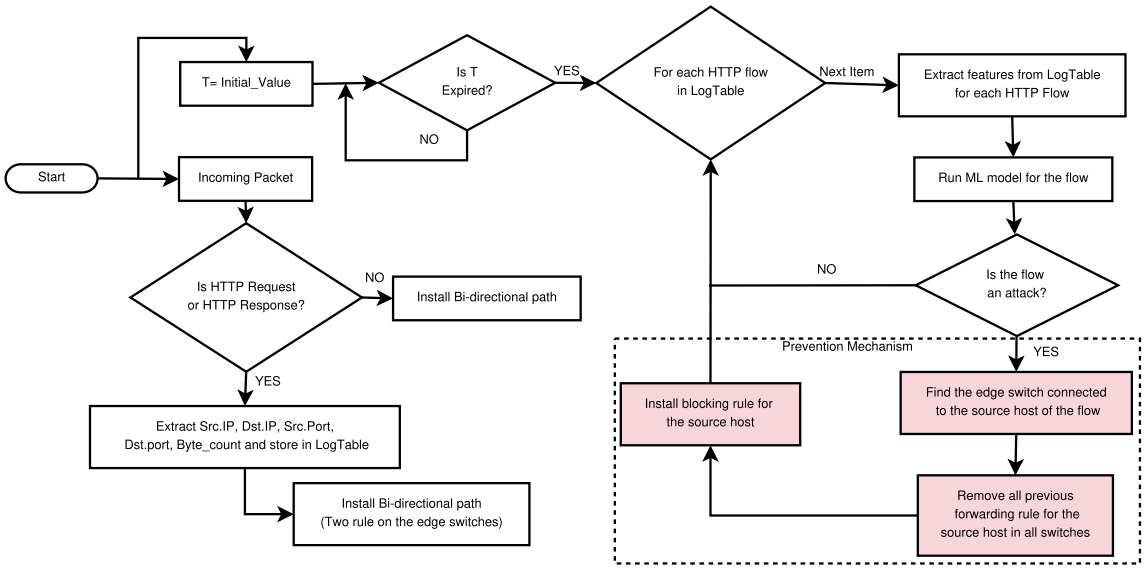

Fig. 1 Detection and Prevention phases in HTTPScout

5.1 Emulation setup

We implement HTTPScout as a security module upon the RYU controller [43], and to emulate the data plane of the test-bed network, we use Mininet [44], a popular SDN emulation tool. We run our experiments on a PC with four processing cores and 3GB RAM. To achieve accurate evaluation, we applied K-fold cross validation [45] with $k=10$ in the training phase of ML models. Moreover, we considered the 30% of the dataset for test, and 70% of the data to train the ML models.

Simulations are performed on the test-bed network which is shown in Fig. 2. The test-bed network consists of six OpenFlow switches, three legitimate hosts (H1-H3), six attackers (H4-H9), and two flash crowds (H10, H11). We have considered 10 different HTML files, each of which has 51Byte size. Benign hosts send complete HTTP GET requests for one of these HTML files to a web server randomly between 10 and 15 seconds. To investigate the reaction of HTTPScout against flash crowd hosts, we set these hosts to send incomplete HTTP requests for a short time (10 seconds). We have used SlowHTTPtest [46] to perform HTTP flooding attack. The attacker hosts use this tool to send incomplete HTTP requests with different rates. In order to evaluate the performance of HTTPScout in different load of attacks, we have configured five different scenarios in which the attacker hosts send from 10 to 50 incomplete HTTP requests per second. Moreover, we have considered time window $T=30$ seconds. Simulation time for all scenarios is set to 600 seconds, and the attack begin time in all scenarios is set to 220 seconds.

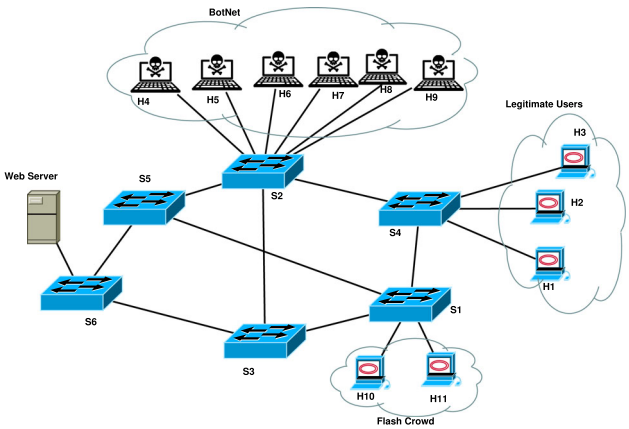

Fig. 2 The test-bed network

5.2 Result analysis

In this section, we evaluate the experimental results using the following performance metrics.

5.2.1 Attack detection time

It denotes the time between sending the first incomplete HTTP request by an attacker and detecting the attack by HTTPScout. Fig. 3 shows that RF method detects the presence of HTTP flooding attack sooner than other ML algorithms. It is because of using multiple decision trees by RF. This mechanism also fortifies RF to be accurate in detection of the attack. Moreover, Fig. 3 shows that by increasing the rate of the attack, detection time decreases. It is because in high rate of the attack, HTTPScout monitors more incomplete HTTP connections for each period of time,

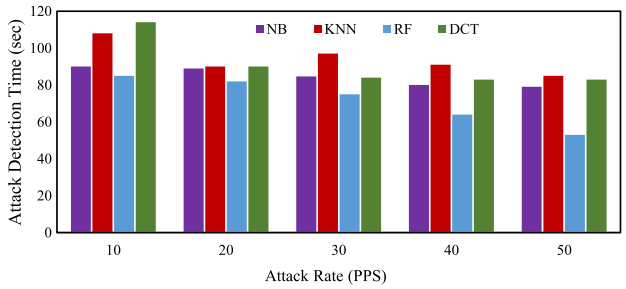

Fig. 3 Attack detection time vs. different attack rates

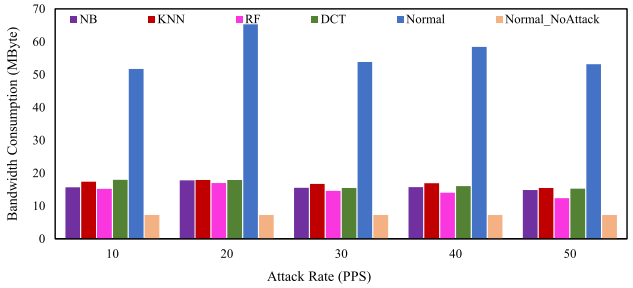

Fig. 4 Average bandwidth consumption vs. different attack rates

thus the attack is detected sooner. One of the most significant parameter that affects the attack detection time is T because HTTPScout checks the presence of the attacks each T seconds. For this reason, considering low value for T decreases the attack detection time and leads to sooner protection of network from adversary effects of the attacks. On the other hand, it imposes more overhead for the SDN controller or might lead to inaccurate results in attack detection.

5.2.2 Bandwidth consumption

As mentioned in Section 1, one of the main adverse effect of HTTP flooding attack is the bandwidth consumption. In this paper, we have considered this metric to denote the amount of traffic traversed via OF switches during the emulation run time. As shown in Fig. 4, for the Normal scenarios in which there is no defense mechanism, the bandwidth consumption is considerably high. Contrary, by using HTTPScout, the bandwidth consumption is significantly decreased. Fig. 4 illustrates that when using HTTPScout as a security mechanism, the bandwidth usages are nearly similar to Normal scenario. As a matter of fact, the figure shows that HTTPScout decreases the adverse impacts of HTTP flooding on the network resources. Moreover, from Fig. 3 and Fig.4, we can conclude that sooner detecting the attack in RF method and consequently blocking the attack leads to prohibiting more network bandwidth consumption by the attackers.

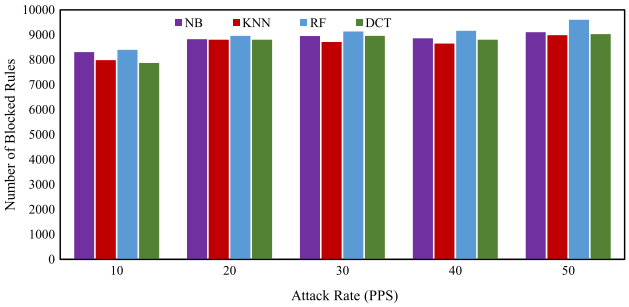

Fig. 5 Number of blocked malicious flows vs. different attack rates

5.2.3 Number of Blocked Malicious flows

It is the number of incomplete HTTP connections that are detected and blocked by HTTPScout. In other words, this metric denotes the number of incomplete requests after the origins of the attack have been blocked at the edge switches. According to Fig. 5, it can be concluded that the best results are achieved when using RF method in HTTPScout. As shown in Fig. 3, we conclude that RF outperforms other ML methods in terms of the attack detection time. It blocks more malicious flows compared to the others. Hence, using RF also improves the performance of HTTPScout in terms of the number of blocked malicious HTTP flows. Consequently, the result of this behaviour preserves the data plane switches from TCAM saturation, which leads to install forwarding rules for benign traffic flows without memory space constraints.

5.2.4 Accuracy, F1-Measure and AUC

Since HTTPScout uses ML algorithms to detect HTTP flooding attack, we investigate the performance of ML algorithms which have been chosen for HTTPScout in terms of three key measures used in machine learning. This includes accuracy, F1-Measure, and Area Under the Curve (AUC). Accuracy focuses on the proportion of correct predictions and defined as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

where TP is True Positive, TN is True Negative, FP is False Positive and FN is False Negative, respectively.

F1-Measure tests the accuracy of a defense mechanism in detection of an attack, especially when the False Negatives and False Positives are critical. As a matter of fact, higher values for F1-Score denote higher correctness in classification. F1-Score is defined as follows:

$$F1 - Measure = \frac{TP}{TP + \frac{FP+FN}{2}} \tag{2}$$

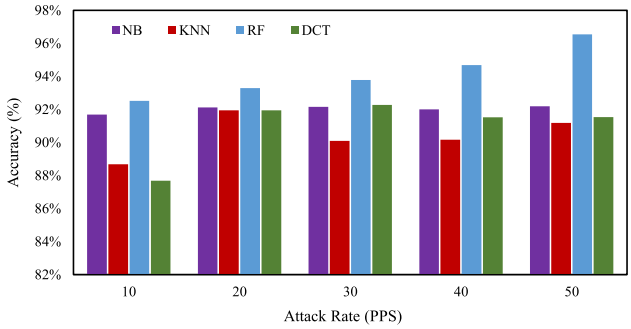

Fig. 6 Accuracy vs. different attack rates

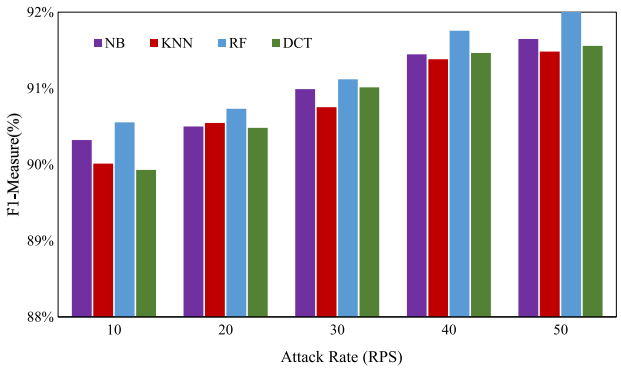

Fig. 7 F-Measure vs. different attack rates

Figures 6 and 7 show the accuracy and F1-Measure values for HTTPScout with different ML algorithms, respectively. According to these two figures, it can be concluded that RF outperforms other ML algorithms because of its ensemble nature. In contrast to the other ML techniques in HTTPScout, only RF is ensemble which means it constructs multiple DCT which considerably improves its performance in the detection of HTTP flooding attack.

AUC for an ML algorithm in network security is a measure that determines the ability of the algorithm to distinguish between the normal and abnormal network traffics. In a nutshell, AUC is used as a summary for Receiver Operator Characteristic (ROC), and higher values for it mean higher capability in correct attack detection. Table 3 shows AUC for different ML algorithms in HTTPScout for various rates of attack. It confirms that using RF method in HTTPScout increases its ability to detect attacks compared to KNN, NB, and DCT.

5.2.5 Number of installed forwarding rules

This metric denotes the total number of remained forwarding entries on the OF switches at the end of emulation. In fact, this metric shows whether a defense mechanism can effectively remove unnecessary forwarding rules from the relevant switches. Moreover, it is a useful metric to estimate the amount of TCAM memory exhaustion caused by the HTTP flooding attack. As it can be seen in Figure 8, using ML techniques to detect and prevent the attack significantly reduces the number of installed forwarding rules. It is because after detection of attack, HTTPScout blocks the source of the attack at the edge switch and prohibits them to send further HTTP packets. Moreover, Figure 8 confirms that among ML algorithms, RF achieves better results compared to others. The reason is that because it achieves more accuracy according to Figure 6, thus it removes more unnecessary rules from the switches.

Table 3 AUC of ML algorithms in HTTPScout for different attack rates

Algorithm	NB	KNN	RF	DCT
10	84%	84%	85%	84%
20	85%	85%	85%	85%
30	85%	85%	86%	85%
40	86%	85%	86%	86%
50	86%	86%	87%	86%

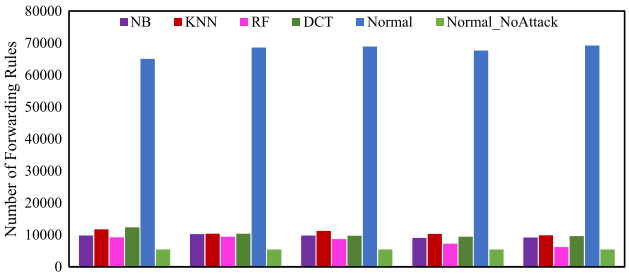

Fig. 8 Number of forwarding rules at the end of simulation time vs. different attack rates

5.2.6 Training time

It is a useful metric to show the quickness of an ML algorithm in training phase. In Section 5, we showed that the time complexity of HTTPScout is low. In addition to the time complexity, since the HTTPScout takes use of ML algorithms, it is essential to investigate the training time of the ML algorithms used by HTTPScout. Table 4 illustrates that RF algorithm has the highest training time among other ML algorithms. The reason is that it uses multiple Decision Trees to solve a classification problem, and thus consumes more time compared to other non-ensemble algorithms. It is worth mentioning that although RF has the highest training time, it has superiority over other ML algorithms investigated in this paper for HTTPScout. Moreover, as we mentioned in Section 4, each ML model is created once and then used by HTTPScout. To update the model, for example, with a newer version of the dataset, the administrator can train the model again and then HTTPScout loads the model at the initiation

Table 4 Training time for ML algorithms used in HTTPScout

Algorithm	NB	KNN	RF	DCT
Training time in Milliseconds	610 ms	2540 ms	15570 ms	1850 ms

phase. Therefore, it is not required to train the model every time HTTPScout begins to detect the attack. In nutshell, the training time does not affect the performance of HTTPScout in attack detection.

6 Discussion

Emulation results in Section 5 showed that using HTTPScout is beneficial in the mitigation of HTTP flooding attack. However, the success of HTTPScout depends on the accurate output of ML algorithms. In other words, if an ML algorithm correctly detects an attack, then HTTPScout can identify the origin of that attack and block it. For this reason, to achieve more accurate results, a network administrator should use suitable ML algorithms and datasets which cover different patterns of HTTP flooding attack. Moreover, as the results show in table 4, using ML algorithms imposes some computational overheads for training that needs to be considered in the environment which faces computational resources constraints. The other issue with HTTPScout is that it cannot be used to mitigate HTTPS attacks. Because HTTPS packets are encrypted at the source, HTTPScout is unable to decrypt them in the controller to distinguish between complete and incomplete HTTP requests.

7 Conclusion

In this paper, we introduce HTTPScout as an ML-based security module for SDN controllers to mitigate one of the popular application layer attack called HTTP flooding. HTTPScout observes ongoing HTTP flows and extracts important features from them, and later it checks their behavior using ML algorithms. Upon detecting an abnormal HTTP request, it first eliminates useless forwarding rules related to the origin of the attack, and then blocks the attacker at the edge switch. HTTPScout not only preserves the targeted web servers from HTTP flooding, but it also reduces the waste memory space of OpenFlow switches, and prohibits network resource exhaustion. One of the benefits of HTTPScout is that, it is implemented in a modular fashion, and it allows more ML models to be added in the future. To investigate the performance of HTTPScout, we carried out a comprehensive evaluation with various ML algorithms in different attack scenarios. Experimental results show that HTTPScout with Random Forest ML algorithm achieves better accuracy and F1-measure compared to KNN, Decision Tree, and Naive Bayes algorithms.

Declarations

Compliance with Ethical Standards Disclosure of potential conflicts of interest Research involving Human Participants and/or Animals Informed consent

Research Data Policy and Data Availability Statements Data sharing not applicable to this article as no datasets were generated or analysed during the current study.

References

1. https://ourworldindata.org/internet. Last visited 9 Mar, 2021
2. https://www.baeldung.com/cs/popular-network-protocols. Last visited 27 Mar, 2021
3. Sreeram, I., Vuppala, V.P.K.: Http flood attack detection in application layer using machine learning metrics and bio inspired bat algorithm. Appl. Comput. inform. **15**(1), 59–66 (2019)
4. Verma, A., Xaxa, D.K.: A survey on http flooding attack detection and mitigating methodologies. Int. J. Innov. Adv. Comput. Sci. **5**, 5 (2016)
5. Jaafar, G.A., Abdullah, S.M., Ismail, S.: Review of recent detection methods for http ddos attack. J. Comput. Netw. Commun. 2019 (2019)
6. Chica, J.C.C., Imbachi, J.C., Vega, J.F.B.: Security in sdn: a comprehensive survey. J. Netw. Comput. Appl. **159**, 102595 (2020)
7. Benzekki, K., El Fergougui, A., Elbelrhiti Elalaoui, A.: Software-defined networking (sdn): a survey. Secur. Commun. Netw. **9**(18), 5803–5833 (2016)
8. Xu, G., Mu, Y., Liu, J.: Inclusion of artificial intelligence in communication networks and services. ITU J. ICT Discov. **1**, 1–6 (2017)
9. Sultana, N., Chilamkurti, N., Peng, W., Alhadad, R.: Survey on sdn based network intrusion detection system using machine learning approaches. Peer-to-Peer Netw. Appl. **12**(2), 493–501 (2019)
10. Xie, J., Yu, F.R., Huang, T., Xie, R., Liu, J., Wang, C., Liu, Y.: A survey of machine learning techniques applied to software defined networking (sdn): research issues and challenges. IEEE Commun. Surv. Tutor. **21**(1), 393–430 (2018)
11. Nikoloudakis, Y., Kefaloukos, I., Klados, S., Panagiotakis, S., Pallis, E., Skianis, C., Markakis, E.K.: Towards a machine learning based situational awareness framework for cybersecurity: an sdn implementation. Sensors **21**(14), 4939 (2021)
12. Singh, J., Behal, S.: Detection and mitigation of ddos attacks in sdn: a comprehensive review, research challenges and future directions. Comput. Sci. Rev. **37**, 100279 (2020)
13. Yang, L., Zhao, H.: Ddos attack identification and defense using sdn based on machine learning method. In: 15th International Symposium on Pervasive Systems, Algorithms and Networks (I-SPAN). IEEE 2018, pp. 174–178 (2018)
14. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html. Last visited 3 Sep, 2021

15. Santos, R., Souza, D., Santo, W., Ribeiro, A., Moreno, E.: Machine learning algorithms to detect ddos attacks in sdn. Concurr. Comput. Pract. Exp. **32**(16), e5402 (2020)
16. Sahoo, K.S., Iqbal, A., Maiti, P., Sahoo, B.: A machine learning approach for predicting ddos traffic in software defined networks. In: 2018 International Conference on Information Technology (ICIT), pp. 199–203. IEEE (2018)
17. Sen, S., Gupta, K.D., Ahsan, M.M.: Leveraging machine learning approach to setup software-defined network (sdn) controller rules during ddos attack. In: Proceedings of International Joint Conference on Computational Intelligence, pp. 49–60. Springer (2020)
18. Pérez-Díaz, J.A., Valdovinos, I.A., Choo, K.-K.R., Zhu, D.: A flexible sdn-based architecture for identifying and mitigating low-rate ddos attacks using machine learning. IEEE Access **8**, 155859–155872 (2020)
19. Cic dos dataset (2017). https://www.unb.ca/cic/datasets/dos-dataset.html. Last visited 3 Sep, 2021
20. Li, D., Yu, C., Zhou, Q., Yu, J.: Using svm to detect ddos attack in sdn network. IOP Conf. Ser. Mater. Sci. Eng. **466**(1), 012003 (2018)
21. Polat, H., Polat, O., Cetin, A.: Detecting ddos attacks in software-defined networks through feature selection methods and machine learning models. Sustainability **12**(3), 1035 (2020)
22. Dong, S., Sarem, M.: Ddos attack detection method based on improved knn with the degree of ddos attack in software-defined networks. IEEE Access **8**, 5039–5048 (2019)
23. Sanjeetha, R., Raj, A., Saivenu, K., Ahmed, M.I., Sathvik, B., Kanavalli, A.: Detection and mitigation of botnet based ddos attacks using catboost machine learning algorithm in sdn environment. Int. J. Adv. Technol. Eng. Explor. **8**(76), 445 (2021)
24. Jayasri, P., Atchaya, A., Sanfeeya Parveen, M., Ramprasath, J.: Intrusion detection system in software defined networks using machine learning approach. Int. J. Adv. Eng. Res. Sci. **8**, 4 (2021)
25. https://www.kaggle.com/datasets. Last visited 3 Sep, 2021
26. Alkasassbeh, M., Al-Naymat, G., Hassanat, A., Almseidin, M.: Detecting distributed denial of service attacks using data mining techniques. Int. J. Adv. Comput. Sci. Appl. **7**(1), 436–445 (2016)
27. https://www.ll.mit.edu/r-d/datasets/1999-darpa-intrusion-detection-evaluation-dataset. Last visited 3 Sep, 2021
28. Sabri, S., Ismail, N., Hazzim, A.: lowloris dos attack based simulatio. IOP Conf. Ser. Mater. Sci. Eng. **1062**(1), 012029 (2021)
29. Suroto, S.: A review of defense against slow http attack. JOIV Int. J. Inf. Visual. **1**(4), 127–134 (2017)
30. Alpaydin, E.: Introduction to Machine Learning. MIT Press, New York (2020)
31. Everitt, B.S., Landau, S., Leese, M., Stahl, D.: Miscellaneous clustering methods. Clust. Anal. 215–255 (2011)
32. Kachavimath, A.V., Nazare, S.V., Akki, S.S.: Distributed denial of service attack detection using naïve bayes and k-nearest neighbor for network forensics. In: 2020 2nd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA), pp. 711–717, IEEE (2020)
33. Elbaghdadi, A., Mezroui, S., El Oualkadi, A.: K-nearest neighbors algorithm (knn): an approach to detect illicit transaction in the bitcoin network. In: Integration Challenges for Analytics, Business Intelligence, and Data Mining, pp. 161–178. IGI Global (2021)
34. Berrar, D.: "Bayes' theorem and Naive Bayes classifier. In: Encyclopedia of Bioinformatics and Computational Biology: ABC of Bioinformatics; Elsevier Science Publisher: Amsterdam, The Netherlands, pp. 403–412 (2018)
35. Chen, Y., Pei, J., Li, D.: Detpro: a high-efficiency and low-latency system against ddos attacks in sdn based on decision tree. In: ICC 2019–2019 IEEE International Conference on Communications (ICC), pp. 1–6. IEEE (2019)
36. Lakshminarasimman, S., Ruswin, S., Sundarakantham, K.: Detecting ddos attacks using decision tree algorithm. In: 2017 Fourth International Conference on Signal Processing, Communication and Networking (ICSCN), pp. 1–6. IEEE (2017)
37. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
38. Chen, L., Zhang, Y., Zhao, Q., Geng, G., Yan, Z.: Detection of dns ddos attacks with random forest algorithm on spark. Procedia Comput. Sci. **134**, 310–315 (2018)
39. Idhammad, M., Afdel, K., Belouch, M.: Detection system of http ddos attacks in a cloud environment based on information theoretic entropy and random forest. Secur. Commun. Netw. 2018 (2018)
40. https://github.com/ahlashkari/CICFlowMeter/blob/master/ReadMe.txt. Last visited 27 Mar, 2021
41. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: machine learning in python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
42. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J.: Openflow: enabling innovation in campus networks. ACM SIGCOMM Comput. Commun. Rev. **38**(2), 69–74 (2008)
43. https://ryu-sdn.org/. Last visited 27 Mar, 2021
44. http://mininet.org/. Last visited 27 Mar, 2021
45. Michalski, R.S., Carbonell, J.G., Mitchell, T.M.: Machine Learning: An Artificial Intelligence Approach. Springer, Berlin (2013)
46. https://github.com/shekyan/slowhttptest. Last visited 27 Mar, 2021