information

MDPI

Article

SDToW: A Slowloris Detecting Tool for WMNs

Vinicius da Silva Faria, Jéssica Alcântara Gonçalves, Camilla Alves Mariano da Silva, Gabriele de Brito Vieira and Dalbert Matos Mascarenhas *

Centro Federal de Educação Tecnológica Celso Suckow da Foseca–CEFET/RJ, Petrópolis 25600-000, Brazil; vinicius.faria@aluno.cefet-rj.br (V.d.S.F.); jessica.goncalves@aluno.cefet-rj.br (J.A.G.); camilla.silva@aluno.cefet-rj.br (C.A.M.d.S.); gabriele.vieira@aluno.cefet-rj.br (G.d.B.V.)
* Correspondence: dalbert.mascarenhas@cefet-rj.br

Received: 3 November 2020; Accepted: 23 November 2020; Published: 25 November 2020

Abstract: Denial of service (DoS) attacks play a significant role in contemporary cyberspace scenarios. A variety of different DoS attacks pollute networks by exploring various vulnerabilities. A group of DoS called application DoS attacks explore application vulnerabilities. This work presents a tool that detects and blocks an application DoS called Slowloris on wireless mesh networks (WMNs). Our tool, called SDToW, is designed to effectively use the structure of the WMNs to block the Slowloris attack. SDToW uses three different modules to detect and block the attack. Each module has its specific tasks and thus optimizes the overall detection and block efficiency. Our solution blocks the attacker on its first WMN hop, reducing the malicious traffic on the network and avoiding further attacks from the blocked user. The comparison results show that SDToW performs with 66.7% less processing consumption and 89.1% less memory consumption than Snort. Our solution does not limit the number of parallel connections per user. Hence, by avoiding this limitation, SDToW has a lower incidence of false positive errors than Snort.

Keywords: DoS; Slowloris attack; WMN; ADoS; HTTP attack

1. Introduction

Currently, denial of service (DoS) attacks create a significant threat to the available resources on the Internet and internal networks. DoS aims to prevent legitimate users from accessing resources available on the network. The attack can exploit vulnerabilities or depletes resources on the target. Consequently, it directly affects the performance of network services [1–3].

There is a group of DoS attacks designed to consume resources from Web and E-mail services. This resource-consuming strategy consequently makes the application services inaccessible on the network. Most DoS attacks create a large number of open or semi-open TCP connections on the target host [4,5]. These open TCP connections disable the server from accepting legitimate requests due to many waiting connections on its sockets. Moreover, the attack consumes memory and processing during its execution. This resource consumption behavior has encouraged studies to design new DoS attack detection systems and damage reduction mechanisms [6].

The literature separates DoS attacks into those that target specific vulnerabilities and those that use flooding attacks [7]. In flooding attacks, the target receives a continuous and excessive volume of packets. Thus, traffic from legitimate sources can be blocked due to congestion and discarding packets. Regarding vulnerability attacks, the target host receives malformed packets [8]. These malformed packets interact with flaws or vulnerabilities in an application or resource hosted by the target.

Another group of DoS attacks specifically exploit the application layer, defined as application layer DoS (ADoS) [9]. Two types of ADoS attacks can be highlighted: flooding and low-rate. The first one produces an excessive flow of traffic, consuming resources on the application level. The second

creates traffic similar to legitimate requests, using vulnerabilities found in the application. Common vulnerabilities exploited by these attacks are the HTTP and HTTPS protocols. This vulnerability permits one to retain connections indefinitely [10]. Low-rate attacks, such as Slowloris, consume resources mainly on the target services without significantly affect the attacker's resources.

The Slowloris attack consists of sending multiple HTTP requests to the Web servers. These requests are similar to the Web browser clients' requests, and therefore it creates challenges to detect the attack. Slowloris send its packets without finishing the HTTP request. Additionally, Slowloris sends periodic incomplete requests to ensure that the server does not finish the connection due to a lack of response. Without knowing if the HTTP requests belong to a legitimate client, the server keeps the connection open. These Slowloris requests are sent in parallel with different source ports to occupy the resources on the server. After consuming the server's resources, it creates an inability to fulfill the new legitimate requests.

The existing literature provides a considerable number of solutions to the Slowloris attack. Some of these solutions use strategies such as limiting the number of connections for each user or defining timeouts for each connection [11]. These limitations can prevent legitimate users from maintaining connections that exceed a pre-established limit. Legitimate users accessing pages with many objects in non-persistent HTTP connection mode can easily exceed this limit. Additionally, strategies that limit the connection timeout can block users that have idle connections. Our solution differentiates from the ones present in the literature because it is designed to work on wireless mesh networks.

Wireless mesh networks (WMNs) have proven to be an alternative to provide low-cost access to the Internet and network resources. The WMNs backbone consists of stationary nodes providing a mesh of connectivity [12]. These wireless nodes play an essential part in traffic routing performance [13]. Clients can access the network resources using the WMNs nodes, thereby accessing Web servers, DNS and the Internet. WMNs are also susceptible to DoS attacks, which can jeopardize network resources [14].

This work presents a Slowloris behavior analysis on WMN. Additionally, we provide a new tool to detect and prevent this attack. Our tool, SDToW, blocks Slowloris attacks without limiting the number of clients' connections. Therefore, our solution avoids blocking legitimate users. We performed our analysis using network traffic logs and by analyzing traffic behavior. This analysis provides useful information concerning the attack behavior. We investigated a real scenario using a Web server and legitimate clients and attackers. Our tool uses filters to extract crucial information from the traffic logs and detect a Slowloris attack. These filters extract the information from the Web server and analyze it on a different node called Concentrator. After that, considering a correct attack identification, our tool blocks the attacker MAC address. The tool blocks the attacker on its associated access point (AP) and avoids further malicious traffic, from that specific attacker, on the WMN. Our tool is separated into three different modules to act on different locations and perform different roles concerning the attack identification process in a WMN. We compared our tool with Snort to analyze its efficiency in blocking attacks and resource consumption.

The work is organized as follows. Section 2 presents related work concerning the defense strategies against Slowloris attacks. Section 3 shows an examination of the Slowloris attack behavior. Section 4 describes the proposed tool and details its modules. Section 5 presents the results of the experiments and compares our tool with Snort. Section 6 presents a discussion regarding how SDToW can work with other security applications. The conclusion is presented in Section 7.

2. Related Works

Slowloris attacks have proven to be worthy of attention regarding the difficulty of countermeasures for their severe effects. The literature provides many works to face this challenge. However, the majority of the solutions present a high level of complexity and time-consuming responses [11]. Some of the solutions uses IPTABLES [15] and Apache modules [16] to provide countermeasures against Slowloris attacks. However, Apache modules such as modevasive [17] and modqos [18] do not efficiently reduce

the attacks' impacts [11]. Furthermore, IPTABLES relies on a threshold to count the number of active connections, which can create false positive errors. An advanced policy firewall (APF) [19] can also be used to counter this attack, but like IPTABLES, its constraints can also lead to false positive errors [20]. The combination of Apache modules and detection tools can also be used in a cloud computing environment [21].

Some studies state that it is possible to use a combination of the timeout and the minimum data rate per request received at the server [22–24]. Therefore, those studies investigated the slow ongoing connections to mitigate the effects of slow DoS attacks, as occurs in Slowloris.

Sousa et al. analyzed two intrusion detection system (IDS) tools to detect Slowloris [25]. The first, called Suricata [26], does not generate an adequate number of alerts to detect Slowloris attack. The second tool, called SNORT [27,28], performs the detection using a trade-off between memory and processing consumption. Snort has been widely used against cybercrimes[29–31]. Lately, in this work, we compare the performances of Snort and our tool. Another tool, called SeVen, provides a defense module based on a selective strategy that uses probability functions to choose among the new requests that will be accepted or rejected when the Web server is saturated [32]. Consequently, when a new request arrives at the application, SeVen checks for availability in the server connections pool. If there are no more available resources for connections, it uses probability to determine which of the established connections must be closed.

Another approach designed to detect attacks on the application layer, such as Slowloris, uses signature-based DoS attack detection and anomaly detection techniques [16,33,34]. Detecting a signature-based DoS attack requires statistical monitoring of the incoming traffic. This strategy produces satisfactory attack detection if the inspected traffic has predefined characteristics of malicious activity. Alternatively, artificial intelligence can be used to detect malicious signatures in application DoS attacks [35–38]. However, these approaches have limitations, given the existing traffic variations concerning the HTTP requests. Therefore, considering that Slowloris creates traffic similar to legitimate connections, this detection strategy could lead to false positive errors. A critical challenge for these approaches relies on determining the majority of legitimate traffic behavior that will be applied to the training sessions [39–41].

Tripathi et al. proposed an anomaly detection system that measures the Hellinger distance between two probability arrangements [42]. The authors obtained the results using a subset of training and tests. During the test phase, the authors compared the attack traffic and the pattern produced in the training phase. They used the most recent HTTP request pattern, using the Hellinger distance to determine the difference between the two probability arrangements. However, the classification method presents false positives, considering the system's probability to differentiate malicious traffic from legitimate traffic. Furthermore, Velan et al. considers that the process of flow data creation is often neglected on Slowloris analysis [43]. Therefore, it leads to negative data analytic results.

Our solution presents improvements, considering its efficiency to detect and block the attack. Furthermore, our solution is designed to work on a WMN and therefore uses the network structure to perform its actions. We briefly summary these improvements below.

- Our tool does not limit the number of parallel connections per user. Additionally, we do no use a timeout limitation for established connections. These behaviors can significantly increase the number of false positive errors. Moreover, limiting the number of parallel connections can block legitimate users from accessing pages with multiple objects or legitimate users behind a NAT.
- The separates modules provide efficient use of the WMN infrastructure to block the attacker—blocking the attacker near its origin, on its first AP node.
- SDToW blocks the attacker using its MAC address instead of the IP. Therefore, we avoid a legitimate user to receive a prior blacklisted IP from the DHCP server.
- Our solution detects malicious traffic without considering the expected probability of choosing between legitimate and malicious traffic. Therefore, we avoid an increase in false positive errors and resource consumption.

- Our tool has less computational complexity and thus promotes less hardware dependency to operate.

3. Analyzing Slowloris Behavior

Slowloris keeps a series of incomplete connections on the Web server [44]. Therefore, keeping connections open will consequently deplete the Web server's resources. Additionally, Slowloris sends several reassembled protocol data unit (PDU) packets in intervals to maintain the ongoing connections. Sending multiple PDU packets is one of the behaviors of Slowloris attacks. Our solution identifies the attack behavior and separates it from legitimate traffic.

Figure 1 presents the impact of a Slowloris attack on a Web server. After complete resource depletion, it uses the available amount of parallel connections. The attack incapacitates the Web server; it cannot accept new HTTP connections. Therefore, once the attack reaches its maximum level, only small PDUs packets occupy the Web traffic.

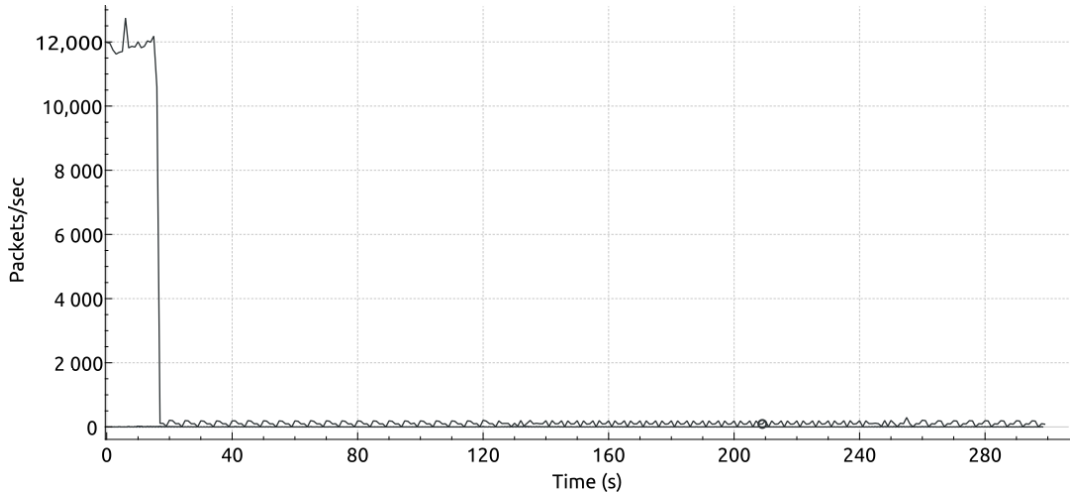

Figure 1. Web server traffic during a Slowloris attack.

Initially, we create filters to distinguish different connections directed to the Web server. These filters aim to gather information and provide more precise identification of Slowloris attack incidence. As mentioned earlier, we design filters based on the attack behavior instead of using the number of connections limitation. Initially, we use Tshark [45] to convert the captured traffic; alternatively, TCPDUMP [46] offers a similar capacity to acquire traffic information [47]. After gathering enough information from the collected traffic, the next step is to analyze the Slowloris attack's detection. Below, we analyze the attack using two different scenarios.

3.1. Slowloris Traffic Analyses

We performed the attack analysis using a scenario with a Web server receiving a direct attack. The use of Tshark provides highly detailed information and helps to identify the attacker's behavior, as shown in Figure 2. During the traffic analysis, we observe that Slowloris uses an initial "GET" containing a reassembled PDU. Then, the following packets have 74 byte size and perform parallel connections to the destination port 80. Furthermore, after establishing a connection with the server, Slowloris sends a sequence of fragmented requests. Using reassembled PDUs, the attack tells the server that more data are coming. These fragmented requests consume the Web server's resources. Consequently, after sending multiple fragmented requests from different source ports, Slowloris occupies all the available resources dedicated to upcoming connections on the server-side. Therefore, the attack prevents the server from attending to new requests from legitimate users, creating a denial of service.

```
3.616749795,192.168.70.51,TCP,74,GET / HTTP/1.1  [TCP segment of a reassembled PDU]
3.820743235,192.168.70.51,TCP,296,[TCP Retransmission] 51092  >  80 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=230
3.828757712,192.168.70.51,TCP,296,[TCP Retransmission] 51098  >  80 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=230
3.828768136,192.168.70.51,TCP,296,[TCP Retransmission] 51096  >  80 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=230
3.828770591,192.168.70.51,TCP,304,[TCP Retransmission] 51094  >  80 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=238
3.828773012,192.168.70.51,TCP,296,[TCP Retransmission] 51100  >  80 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=230
4.268790220,192.168.70.51,TCP,304,[TCP Retransmission] 51094  >  80 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=238
4.268808896,192.168.70.51,TCP,296,[TCP Retransmission] 51100  >  80 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=230
4.268811373,192.168.70.51,TCP,296,[TCP Retransmission] 51098  >  80 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=230
4.268813476,192.168.70.51,TCP,296,[TCP Retransmission] 51096  >  80 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=230
4.268815712,192.168.70.51,TCP,296,[TCP Retransmission] 51092  >  80 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=230
4.428496988,192.168.70.51,TCP,74,[TCP Retransmission] 51102  >  80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460
```

Figure 2. Slowloris attack traffic.

3.2. HTTP Legitimate Traffic Analyses

Our next scenario presents a traffic analysis using legitimate user HTTP requests. We analyze the traffic behavior using only legitimate connections to the Web server. After that, we compare legitimate connections, shown in Figure 3, and Slowloris connections. After establishing a legitimate connection, the first GET is not a reassembled PDU and has a value different than 296 bytes. Additionally, the GET request presents the HTTP protocol set on the traffic. Those two traffic characteristics permit us to identify legitimate traffic. Through conducting an in-depth analysis using Tshark, we observe that the GET request originated in the attack traffic is set as a TCP protocol field.

```
44.085477645,192.168.70.24,HTTP,259,GET /index.html HTTP/1.1 ,33818,80
44.085491071,192.168.70.22,TCP,66,53818  >  80 [ACK] Seq=1029 Ack=2289463 Win=184832 Len=0
44.085711617,192.168.70.22,TCP,66,53818  >  80 [ACK] Seq=1029 Ack=2292359 Win=184832 Len=0
44.086207918,192.168.70.22,TCP,66,53818  >  80 [ACK] Seq=1029 Ack=2296703 Win=184832 Len=0
44.086457796,192.168.70.22,TCP,66,53818  >  80 [ACK] Seq=1029 Ack=2299599 Win=184832 Len=0
44.086708451,192.168.70.22,TCP,66,53818  >  80 [ACK] Seq=1029 Ack=2302495 Win=184832 Len=0
44.086958389,192.168.70.39,TCP,66,54858  >  80 [ACK] Seq=1029 Ack=37868271 Win=997760 Len=0
44.087227804,192.168.70.39,TCP,66,54858  >  80 [ACK] Seq=1029 Ack=37871167 Win=997760 Len=0
```

Figure 3. HTTP traffic.

4. SDToW

The traffic analysis provides an essential step towards the Slowloris attack detection. Our tool uses this traffic analysis to detect the attack behavior.

Our previous traffic analysis provides the development of filters. We use these filters to extract relevant information and thus identify the Slowloris attack in Web servers. The filters created were:

- (GET) filter.
- (Reassembled PDU) filter .
- (Packets with 296 bytes and TCP set on protocol field) filter.

The filters were applied using a python3 script. This script picks only the lines that match the information required to identify the attack. Initially, from the traffic logs connections directed to the Web server, our tool applies the GET Filter to select connections containing GETs. In the next step, we combine the first filter with the second filter, thereby extracting connections also marked as reassembled PDU. The last filter identifies connections marked as TCP on the protocol field and a 296 byte packet size. After applying the three combined filters, SDToW creates a list containing the connections that meet all the pre-defined requirements. Therefore, our tool separates only traffic that fulfills these requirements.

By analyzing the Slowloris traffic and the HTTP legitimate traffic described in Section 3, we observed that after establishing the connection, the first request to the Web server is not a default GET request; see Figure 4. This behavior occurs because Slowloris does not send complete GET requests. Thus, Slowloris avoids the request conclusion by inserting a "\n" at the request data's end. Conversely, legitimate requests send a complete request within the packet. Furthermore, the Slowloris traffic requests use a TCP value in the fourth traffic logs column, thereby differentiating from the legitimate requests that present the same column's HTTP value.

```
3.616749795,192.168.70.51,TCP,74,GET / HTTP/1.1  [TCP segment of a reassembled PDU]
6.656491721,192.168.70.51,TCP,74,GET / HTTP/1.1  [TCP segment of a reassembled PDU]
9.720134739,192.168.70.51,TCP,74,GET / HTTP/1.1  [TCP segment of a reassembled PDU]
10.744195225,192.168.70.51,TCP,74,GET / HTTP/1.1  [TCP segment of a reassembled PDU]
13.687925009,192.168.70.51,TCP,74,GET / HTTP/1.1  [TCP segment of a reassembled PDU]
```

Figure 4. Tracers from the Slowloris attack after applying the filters.

4.1. SDToW Modules

After identifying the Slowloris-type malicious behavior, our tool initiates countermeasures to block the malicious traffic. Therefore, it performs the attack identification using a node to analyze the traffic log, here called Concentrator. The correct attack identification prevents legitimate users from being affected by restrictive measures, thereby generating fewer false positives. Figure 5 presents the three modules, described below:

- CM: collection module (CM), which acts in the Web server.
- AFM: analysis and filtering module (AFM), which works in a different node called Concentrator.
- BM: blocking module (BM), which acts in the access points.

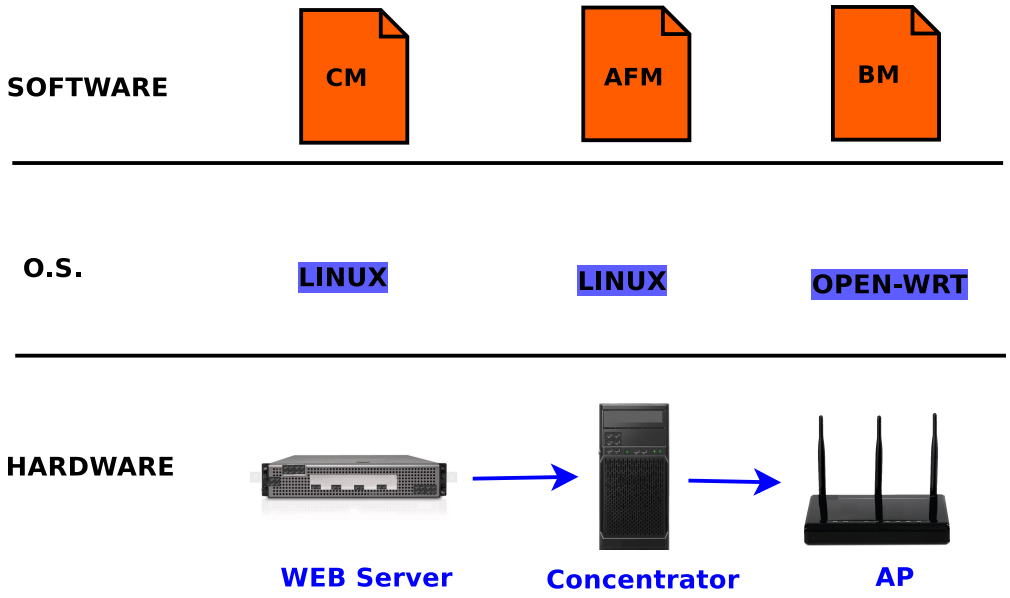

Figure 5. SDToW architecture.

Figure 6 presents the collection module (CM) flowchart working on the Web server. Initially, the CM collects traffic logs from HTTP requests. After that, it collects the traffic for five seconds. We choose to collect for 5 s because it was enough to collect relevant traffic information without delaying too much to send the traffic list to the Concentrator. In previous tests, a number between 5 and 7 s did not produce a relevant delay. Additionally, it provided enough time to detect and stop the attack before it depletes the server resources. It is essential to mention that this traffic collection occurs continuously. Therefore, at every 5 s, a new traffic list is created and sent to the Concentrator. After that, it creates a traffic log using the tool Tshark. We set Tshark to convert the received traffic to a pcap extension. Subsequently, CM removes irrelevant information concerning HTTP connections and creates the traffic list. Next, CM sends the traffic list to the Concentrator.

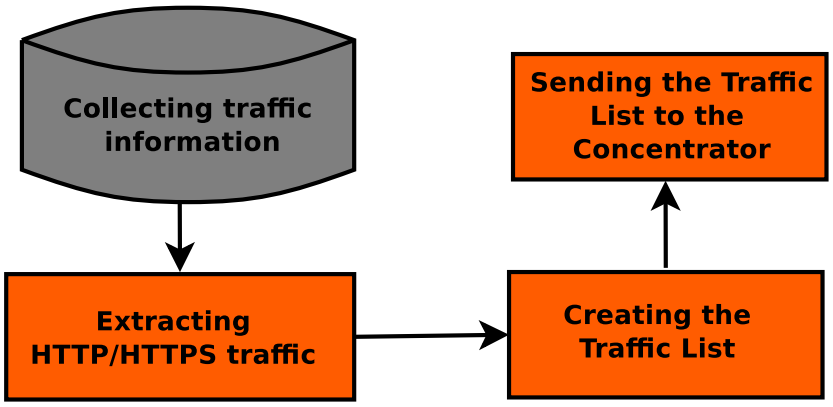

Figure 6. Collection module (CM) flowchart.

Algorithm 1 explains the AFM operation, which runs in the Concentrator machine. AFM uses the previously defined filters to identify attacks. This attack identification uses the traffic list received from the CM. Therefore, AFM selects connections that contain GETs and containing reassembled PDU. It will then perform two analyses to obtain connections whose packet size is 296 bytes, and includes TCP defined on the protocol field. AFM considers a decisive attack when connection information fulfills the filters mentioned above. Upon a positive attack identification, AFM extracts the source IP from the malicious connection. After that, it includes the attacker source IP on a blacklist file. Subsequently, the module sends the blacklist file to the APs.

Algorithm 1: Analysis and filtering module (AFM)

```
1  Blacklist file
2  begin
3      TL ⟵ TrafficList
4      Line ⟵ Pointer to the first TL element
       /* EoF is the end of TL                                          */
5      while Line ≠ EoF do
6          if Line contains "GET" AND "ReassembledPDU" then
7              if Line contains Packetsize == 296 then
8                  Extract the "Source IP"
9                  Insert "Source IP" on the Blacklist
10             end
11         end
12     end
13     if Blacklist ≠ empty then
14         send Blacklist to the APs
15     end
16 end
```

Algorithm 2 presents the steps concerning the BM, which runs on the APs. Initially, the APs receive the blacklist generated by the Concentrator. Then, it compares the IP address of the received blacklist and the IPs on its ARP table. This comparison verifies the occurrence of IPs connected to the AP and also present on the received blacklist. By using the APs ARP table, BM acquires the associated MAC with the attacker IP. Subsequently, BM uses IPTABLES to block traffic from the attacker MAC addresses. After these steps, BM waits for new blacklists in order to block new attackers. We are considering that our tool works on internal networks. Thus we opted to block the MAC address because it generates fewer false positives than blocking based on IP address. Blocking MAC address avoids problems using dynamic IP allocation performed by DHCP (Dynamic Host Configuration

Protocol). DHCP may designate an IP once leased to an attacker to a new user. Therefore, if our tool performs the block based on IP address, it could perform a denial of service for legitimate users.

Algorithm 2: Blocking module (BM)

```
1  Block attacker IP begin
2      BL ⟵ Traffic List
3      Line ⟵ Pointer to the first BL element
4      ARPIP ⟵ list of all ARP table IPs and MACs
       /* EoF is the end of BL                                   */
5      while Line ≠ EoF do
6          if ARPIP contains any IP from BL then
7              Extract the corresponding MAC from ARPIP
8              Block the MAC
9          end
10     end
11 end
```

All the WMN nodes contain the blocking module. Therefore, when the AFM sends the blacklist, it is stored on all WMN nodes. Storing the blacklist on each WMN node permits that it compares its new ARP table whenever a new client connects to the AP. Therefore, if a blocked attacker moves from one AP to another, it will be blocked. Comparing its new ARP table with the blacklist, the current AP will block the attacker by its MAC.

Our tool excludes addresses from the blacklist after a configured period. In our experiments, we maintain the address for 72 h. Whenever a previously blocked attacker tries to access the network during the blocking period, our tool doubles the exclusion period. Network administrators can change this period according to their policy.

5. Results

Figure 7 presents our test scenario running on a network laboratory. All the APs in our scenario do not use Network Address Translation (NAT). The APs operate with two different interfaces, one providing client connection, and the other interface provides connections between the APs. For client connection, we use the IEEE 802.11ac/n/a 5GHz interface.

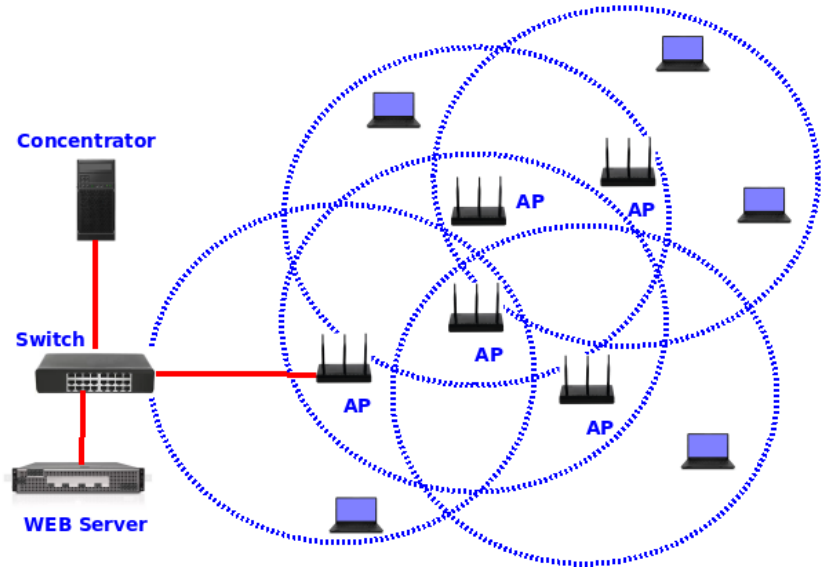

Figure 7. Experimental scenario.

The connection between the APs utilizes the IEEE 802.11b/g/n 2.4 GHz creating an ad-hoc WMN. The WMN uses the Optimized Link State Routing Protocol (OLSR) [48]. The parameters of OLSR used were the following: the interval between HELLO messages was equal to 1.5 s, and the Link state messages were sent every 5 s. The multipoint relays use these Link state messages to calculate the topology map. Hence, we installed two packages in the wireless routers: Openwrt_luci_app_olsr and Openwrt_luci_app_olsr_services. These packages permit the creation of ad-hoc routes using the OLSR protocol. We also use the Expected Transmission Count (ETX) metric to optimize the routing selection. The ETX metric can be defined as the expected number of transmissions required to deliver a packet over a given link [49]. This metric calculates the weight of a given route by using the sum of all ETX weights. Each ETX weight informs the value for every link on a path. This ETX weight guarantees the objective proposed by the metric of choosing routes that decrease the total number of retransmissions at the link level along the way. We also use the following hardware:

- 5 TP-Link-Archer-C20.v4 wireless routers.
- 2 I5-4590 computers with 8 GB RAM
- 5 AMD A8-4500M notebooks with 8 GB of RAM.

Concerning the operating system, we use Ubuntu 18 for the Web server and the notebooks. We replace the original APs firmware with the Open-WRT [50]. Open-WRT is a Linux operating system targeting embedded devices. Performing this replacement allowed our module BM to run directly on the APs. Our Web application server uses the Apache2 version 2.4.7.

We collect information regarding the required time to transfer the traffic list and the blacklist. Additionally, we measure the processing time of our modules. That information allowed us to obtain the total blocking time. The combination of the information collected is explained below.

- Collection count (C_C)—time in which the CM, running on the Web server, collects information regarding the network traffic and creates the traffic list.
- Transfer list time (TL_T)—the required time to send the traffic list from CM to the Concentrator.
- Traffic list processing time (TLP_T)—the required time to analyze the traffic list using the AFM filters. If it finds malicious traffic, it creates the blacklist.
- Blacklist transfer time (BL_T)—the time needed to transfer the blacklist from the AFM to the APs.
- Blocking time (B_T)—the time required to process and block malicious IPs address from the blacklist by the BM.

In our experiments, C_C uses 5 s to collect traffic information and generates the traffic list. The parameters described above define the total blocking time (TB_T) on the equation below:

$$TB_T = C_C + TL_T + TLP_T + BL_T + B_T \quad (1)$$

Equation (1) presents the sum of all related time values that directly impacts the total blocking time TB_T. Concerning the blacklist transfer, we observe that the number of hops produces jitter variation regarding the required time to transfer the list. This jitter variation motivates the second experiment to analyze it on multiple hops scenario. The transfer list time (TL_T) and traffic list processing time (TLP_T) vary according to the traffic list size. The traffic list size and how it interferes with the blocking time motivated us to conduct experiments to analyze it. Below, we describe two experiments to analyze the blocking attack performance.

5.1. Experiment 1: Measuring How the Traffic List Size Affects the Blocking Delay

We start the experiment using one attacker client on the first AP near the Web server. Therefore, the attack occurs at a one-hop distance from the target. We use the experiment to analyze how much the traffic list size influences the blocking time.

After starting the attack, the collection module CM running on the Web Server captures the network traffic and creates a traffic list (TL). CM sends the TL to the Concentrator, which through

its analysis and filtering module (AFM), applies the pre-established filters to detect the attacker's IP address. Upon identifying the malicious IP, it will be added to a blacklist and later sent to the AP. Once the blacklist is received, the BM compares the received IPs with its ARP table, obtaining the respective MAC address. After that, the blocking module (BM) prevents new traffic from the attacker by blocking its MAC address.

In this experiment, we varied the amount of HTTP traffic to change the traffic list size. By changing the traffic list size, we analyzed its impact on the total blocking time. The traffic list size directly influences the transfer list time (TL_T) and the traffic list processing time (TLP_T). Therefore, by increasing the time to send and process the traffic list, this consequently raises the total blocking time (TB_T); see Figure 8.

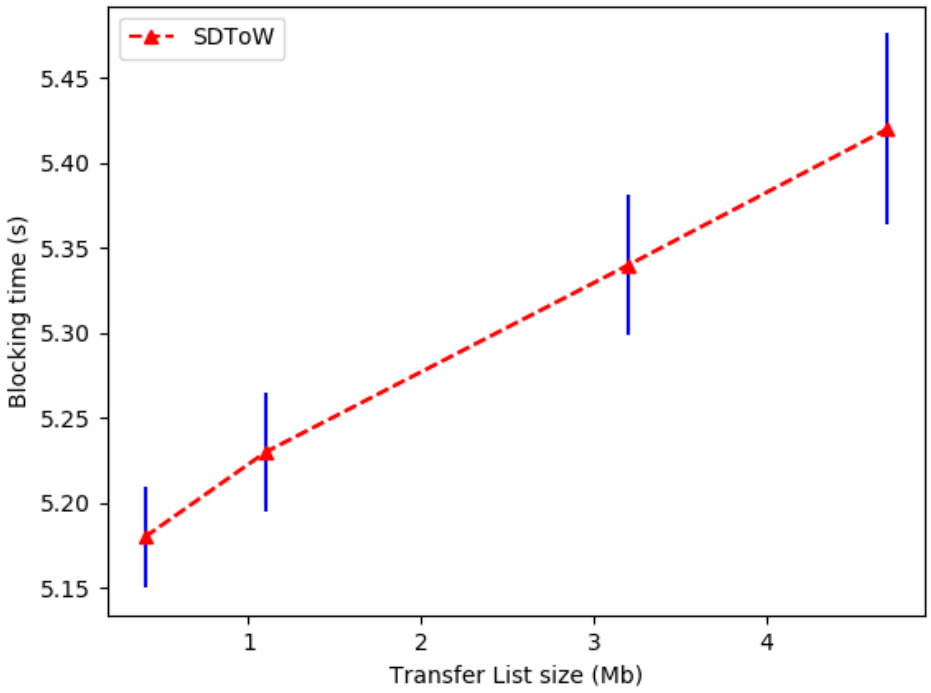

Figure 8. The influence of the traffic list size on the total blocking time.

5.2. Experiment 2: Measuring How the Number of Hops Affects the Blocking Delay

We ran the experiments ten times to obtain the average values. The collection time and the traffic list size directly affect the delay to detect an attacker.

In the second scenario, we increase the number of hops between the attacker and the Web server. Therefore, we analyze the delay to block attackers who are more than one hop from the target. We positioned them at one hop, two hops, and three hops distance from the server. This experiment uses the same process to detect the attacker as the first experiment. The number of hops directly impacted the blocking delay as we varied the number of hops from 1 to 4, as shown in Figure 9. We kept the traffic list at the same size during the experiment. Consequently, it permitted us to isolate the impact concerning the number of hops on countering the attack. The increasing delay to counter the attack relies on the time needed to send the blacklist through multiple hops.

The attacker distance regarding the number of hops increases the time needed for an effective response from SDToW. The blacklist contains only source IPs, and hence it is a file that generally does not need to be fragmented to fit on the network Maximum Transmission Unit (MTU). Therefore, the leading cause of delay for multiple hops derives from passing through multiple routers on our WMN.

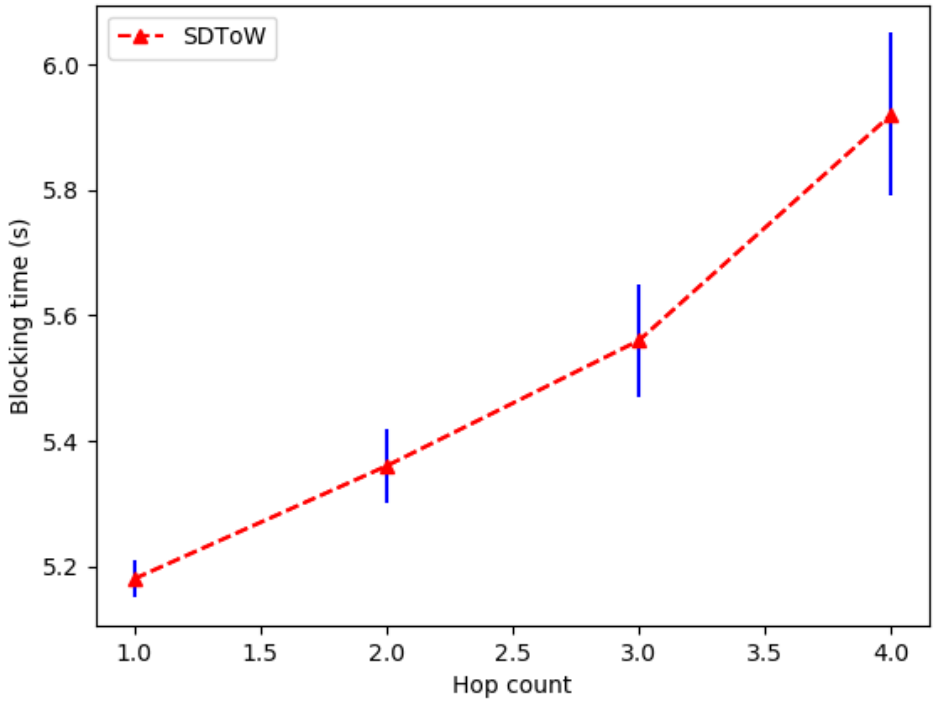

Figure 9. The influence of the number of hops on the total blocking time.

5.3. Experiment 3: Comparing SDToW with Snort

After conducting the experiments mentioned above, we set up a new experiment comparing our tool with Snort. We chose Snort because it is a widespread tool with multiple functions to detect and block attacks. Additionally, the literature presents Snort as a tool to counter the Slowloris attack.

Snort was initially created to be an open-source IDS and later become an IPS. It permits real-time protocol analyses and permits investigations concerning attack behavior. Snort also provides an extensive attack database providing many filters to detect attacks, i.e., fingerprint attacks, buffer overflow, DoS and port scans. Snort initially scans the pre-selected interfaces, acting like a sniffer. During this sniffer task, Snort can capture packets from the network using a passive capture or by inspecting pre-collected traffic. After collecting traffic data, it uses its decoder module on the collected packets. The decoder module inspects packets, searching for anomalies that differ from the default protocol behavior. Then it uses the pre-processor engine that provides in-depth attack analyses on the received traffic. After detecting an attack that matches its internal attack rules, Snort can provide two responses: an attack alert and an action based on the alert. The attack alert consists of generating logs detailing the traffic associated with the alert on the library rules. The action based on the alert provides prevention action during an attack—i.e., blocks an attack. Snort also permits the creation of new detection rules based on the predefined attack behavior. In this work, we used a Snort rule that verifies the number of ongoing connections for each network device. Therefore, whenever a device reached a maximum connection threshold, it would generate an alert and subsequently block the attacker.

Figure 10 presents the Snort scenario gathering information from the incoming traffic. We placed Snort between the AP and the Web server. We designed this scenario to permit a fast attack response by Snort. Therefore, it can block the attacker whenever the malicious traffic matches the anti-Slowloris rule.

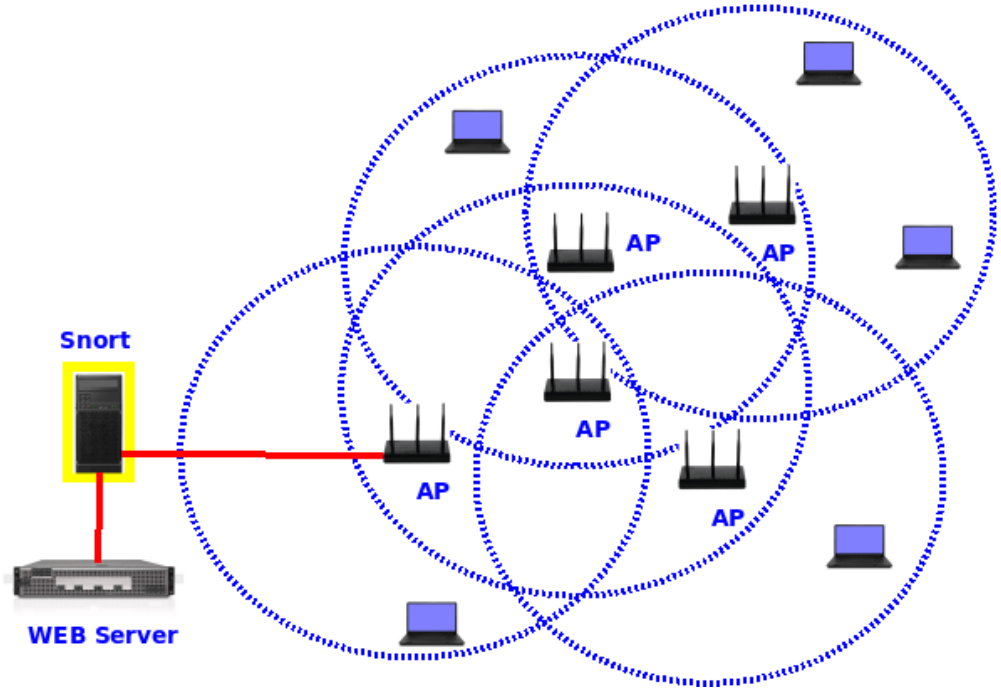

Figure 10. Experimental scenario using Snort.

In this scenario, Snort blocks the attacker by its IP address instead of the MAC address as our tool does. The present scenario imposes Snort to block malicious traffic only by its IPs because it has no information regarding the attacker's MACs addresses. Only the APs that directly provide WiFi access to the clients identify their MAC address. Therefore, considering that Snort does not operate inside the APs, we set Snort to block only the attacker's IP.

We set up a rule to block attackers with a specific number of ongoing connections to the Web server, thereby blocking whenever the number of connections from the same IP exceeded the specified number of ongoing connections.

We compared Snort and our tool by measuring the processing consumption, memory consumption and false positive errors. By measuring the processing consumption and memory consumption, we analyzed both tools regarding their scalability. The amount of analyzed traffic data directly influences the scalability of the tools. After that, we measured the incidence of false positives affecting legitimate users. The wrong classification of legitimate users as attackers creates problems that can jeopardize the solution. That problem emerges when many legitimate clients are classified as attackers and are thus blocked.

Firstly, to measure processing and memory consumption, we set up a scenario varying the number of clients: Figures 11 and 12. The first test uses only one client, and this client is the attacker. The subsequent tests present three and six clients, including among them one attacker client. Increasing the number of clients boosts the amount of traffic data and thus raises the resource consumption. In the experiment with six clients, our tool achieved 66.7% less processing consumption and 89.1% less memory consumption than Snort.

Two factors directly enhanced our tool's performance. The first is the distributive operations by the modules. Our tool uses three modules on different machines, hence distributing the resource consumption between the modules. The second factor is the strategy of blocking the attacker on its first hop. Our tool blocks the attacker on the first AP, where it is connected. Therefore, it reduces the amount of malicious traffic running on the network. Snort does not block the attacker near its origin. Therefore, by using Snort, Slowlois will keep sending connection requests to the network. These requests will pass through the APs until they reach Snort, where they will be dropped.

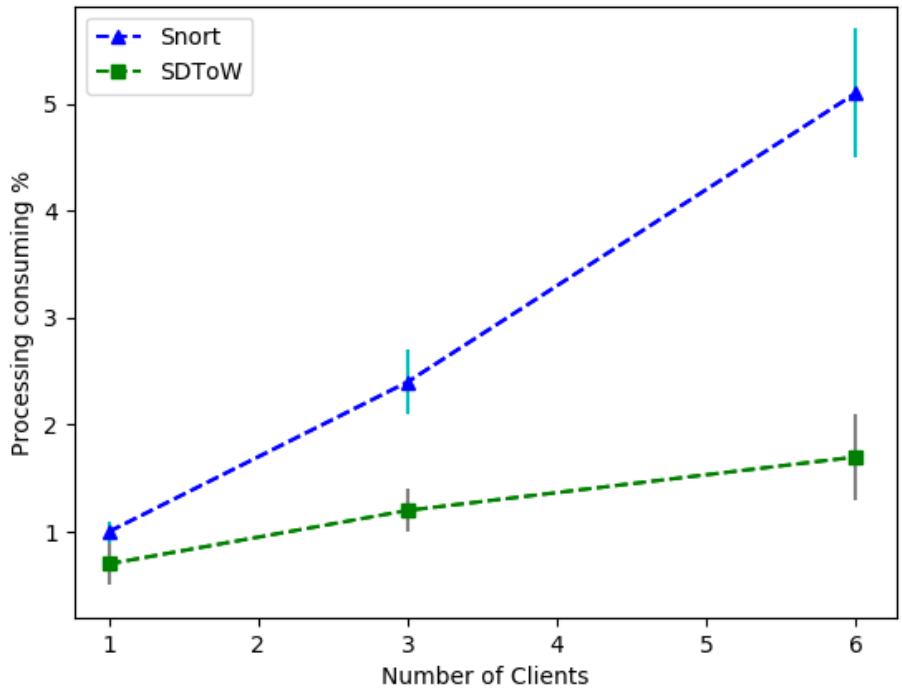

Figure 11. Processing consumption comparison for Snort and SDToW.

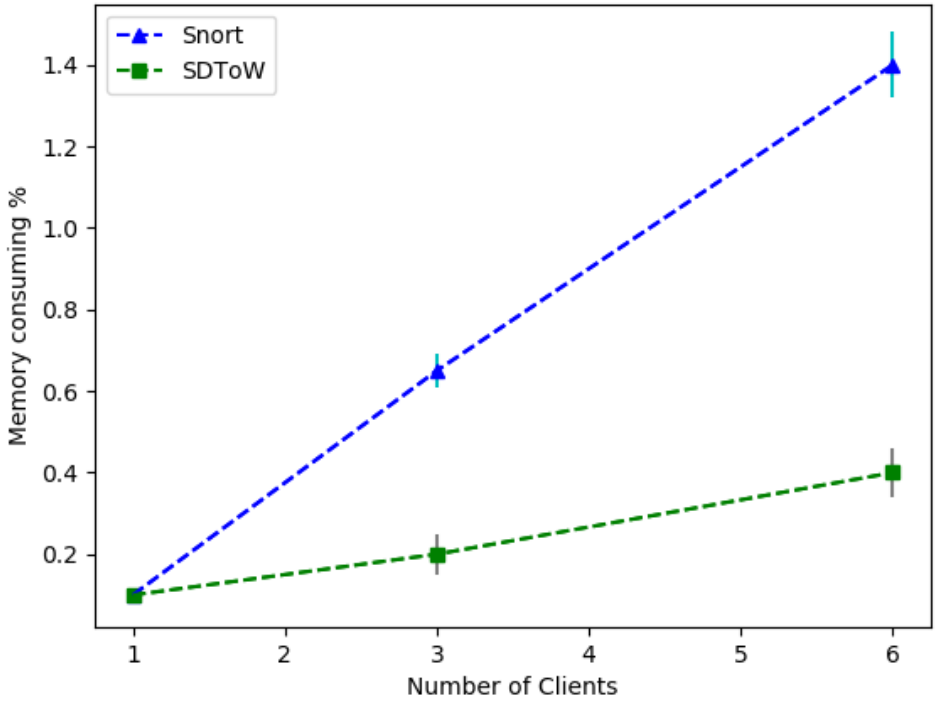

Figure 12. Memory consumption comparison between Snort and SDToW.

After the resource consumption analysis, we measured the number of false positive errors; see Figure 13. We conducted the tests with five clients. One of the five clients was an attacker, and the other clients made legitimate HTTP requests.

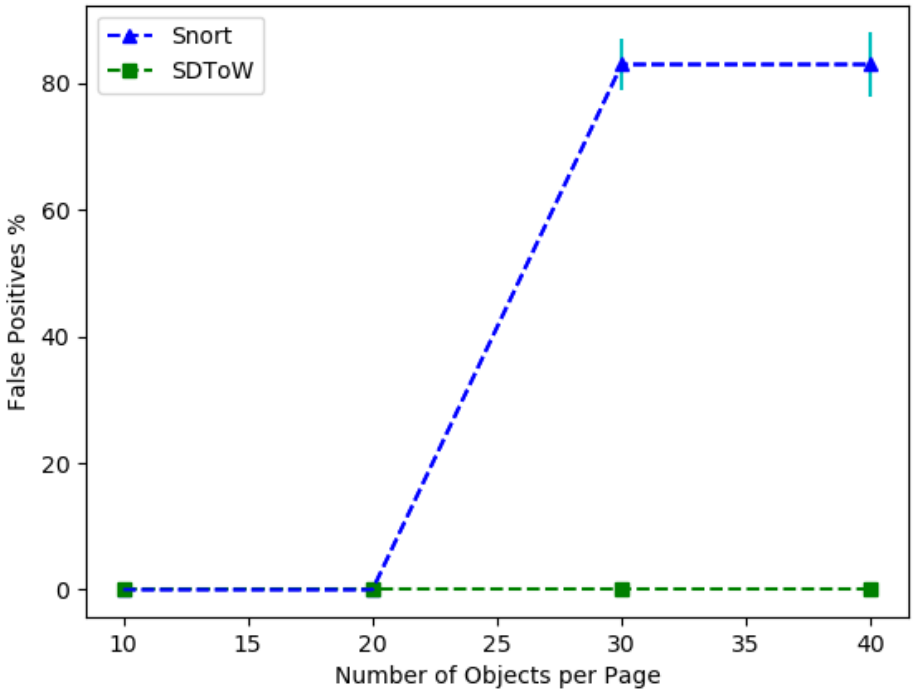

Figure 13. False positive errors comparison between Snort and SDToW.

Snort presents weaknesses considering the occurrence of false positive errors. These weaknesses rely on the number of ongoing connections permitted by each user. In our tests, we used a Snort rule that verifies if the number of connections exceeds 20 parallel connections per user. Therefore, whenever a user exceeds this limit, Snort will drop packets from this user. Considering the page with 10 and 20 objects per page, Snort did not create any false positive blocks.

However, when we increased the number of objects per page to 30, Snort blocked all the clients. Snort blocked the clients because they exceed the maximum limit of 20 parallel connections per user. Therefore, Snort erroneously blocked four of the five clients that were not attackers, creating 80% false positives. The same number of false positives occurred with the 40 objects per page. It is possible to increase this limit for parallel connections, but at any point, this limitation will create a challenge between the number of false positives and accuracy. Therefore, raising the limit of ongoing connections reduces the number of false positives, but it permits Slowloris to consume a higher number of connections.

Conversely, if we reduce the limit of ongoing connections for each user in the Snort rule, it will increase false positive errors. Considering a server hosting a page with many objects per page, it will raise the number of ongoing connections per user. Therefore, if the page has more objects than the connection limit, it will create a false positive error in Snort detection. The connection limit does not affect SDToW because its detection mechanism does not use this limitation parameter. Hence, raising the number of objects per page does not lead our tool to create false positive errors.

6. Discussion

Although SDToW blocks the attacker using its MAC address, it is possible to change the blocking module to perform the block using the source's IP address. Blocking an attacker by its IP address permits Web server protection against attacks that come from the Internet. During a Slowloris attack coming from the Internet, it is not appropriate to block the attacker by its MAC address. This limitation occurs because the source MAC address is not the attacker's address but is the MAC address from the gateway or is the MAC address from the one-hop device directly connected to the blocking module. Therefore, if the blocking module operates in gateway devices, it should be configured to block the attacker by its IP address.

Another question that arises regards IP or MAC spoofing. SDToW is not able to detect IP or MAC spoofing. However, it is possible to combine SDToW with a firewall or IPs, thereby protecting from MAC spoofing and IP spoofing.

SDToW can work with load balancing and a reverse proxy. However, it creates two possible changes to SDToW architecture. The first one is to install the collection module (CM) on the server that provides the reverse proxy or the server that provides load balancing. This change would permit SDToW the ability to capture the traffic dedicated to the Web servers. The second possibility is to use a promiscuous switch port that grants full access to the same network that hosts the Web servers. Therefore, the collection module could be installed on a different node and connected with the promiscuous switch port. Consequently, the collection module would be able to collect the Web traffic.

Slowloris can be mitigated or prevented using restrictions on the Web server access. It is possible to limit the number of parallel connections per user and discard additional connections from the same user. Additionally, it is possible to restrain the amount of low-rate traffic on the Web server, thereby finishing connections that take a long time to have their requests completed. The drawback of these restrictions relies on the difficulty to distinguish legitimate Web traffic from the attacker traffic. Therefore, restrictions created to mitigate or prevent Slowloris attacks can also lead to user access problems. SDToW does not create any traffic restrictions and it blocks the attacker near its first AP. Hence, it mitigates the attack with a lower network impact.

7. Conclusions

This work presents a detection and blocking tool for the Slowloris attack on WMNs. We performed a behavior analysis of the referred attack to extract relevant information from the Web server traffic. We developed filters to collect traffic information. Those filters permitted the correct attack identification and subsequently blocked the ongoing attack.

We used the WMN structure, creating different modules with predefined objectives. The modules work in different locations inside the network, executing different tasks. These modules reduce the resource consumption overhead and permit one to block the attacker using its physical address. In the experiment using six clients, our tool achieved 66.7% less processing consumption and 89.1% less memory consumption than Snort. SDToW blocks the attacker near its origin and thus avoids unnecessary traffic on the network.

We conducted experiments to analyze the impacts of traffic list size and the number of hops. The results of these experiments provide a more precise understanding of the required time to block attacks. Therefore, the experiments reveal that SDToW can mitigate an ongoing Slowloris attack before the Web server's resource depletion. Moreover, by blocking the attacker on its first hop, SDToW limits the attacker from trying new attacks on different servers on the WMNs.

Our solution does not limit the number of parallel connections per user. Hence, by avoiding this limitation, SDToW has a lower incidence of false positive errors. In further research, we intend to analyze the impact of DDoS on our solution and enhance our tool to provide efficient detection of distributed Slowloris attacks.

Author Contributions: Conceptualization, D.M.M.; data curation, V.d.S.F. and C.A.M.d.S.; formal analysis, J.A.G. and G.d.B.V.; investigation, V.d.S.F., J.A.G., C.A.M.d.S., G.d.B.V. and D.M.M.; project administration, D.M.M.; software, V.d.S.F., J.A.G., C.A.M.d.S., G.d.B.V. and D.M.M.; writing—original draft, J.A.G., G.d.B.V. and D.M.M.; writing—review and editing, C.A.M.d.S. and D.M.M. All authors have read and agreed to the published version of the manuscript.

Funding: CEFET/RJ and CNPq.

Acknowledgments: The authors would like to thank CNPq and CEFET/RJ for the financial support given to this research and development work.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Gu, Q.; Liu, P. Denial of service attacks. In *Handbook of Computer Networks: Distributed Networks, Network Planning, Control, Management, and New Trends and Applications*; Wiley: Hoboken, NJ, USA, 2007; Volume 3, pp. 454–468.
2. Tripathi, N.; Hubballi, N. Slow rate denial of service attacks against HTTP/2 and detection. *Comput. Secur.* **2018**, *72*, 255–272. [CrossRef]
3. Singh, M.P.; Bhandari, A. New-flow based DDoS attacks in SDN: Taxonomy, rationales, and research challenges. *Comput. Commun.* **2020**, *154*, 509–527. [CrossRef]
4. Goncalves, J.A.; Faria, V.S.; Vieira, G.B.; Silva, C.A.; Mascarenhas, D.M. WIDIP: Wireless distributed IPS for DDoS attacks. In Proceedings of the 2017 1st Cyber Security in Networking Conference (CSNet), Rio de Janeiro, Brazil, 18–20 October 2017; pp. 1–3.
5. Yuan, H.; Xia, Y.; Yang, H.; Yuan, Y. Resilient control for wireless networked control systems under DoS attack via a hierarchical game. *Int. J. Robust Nonlinear Control.* **2018**, *28*, 4604–4623. [CrossRef]
6. Karapoola, S.; Vairam, P.K.; Raman, S.; Kamakoti, V. Net-Police: A network patrolling service for effective mitigation of volumetric DDoS attacks. *Comput. Commun.* **2020**, *150*, 438–454. [CrossRef]
7. Carl, G.; Kesidis, G.; Brooks, R.R.; Rai, S. Denial-of-service attack-detection techniques. *IEEE Internet Comput.* **2006**, *10*, 82–89. [CrossRef]
8. Sameera, N.; Shashi, M. Deep transductive transfer learning framework for zero-day attack detection. *ICT Express* **2020**, *6*, 361–367. [CrossRef]
9. Jazi, H.H.; Gonzalez, H.; Stakhanova, N.; Ghorbani, A.A. Detecting HTTP-based application layer DoS attacks on Web servers in the presence of sampling. *Comput. Netw.* **2017**, *121*, 25–36. [CrossRef]
10. Toklu, S.; Şimşek, M. Two-Layer Approach for Mixed High-Rate and Low-Rate Distributed Denial of Service (DDoS) Attack Detection and Filtering. *Arab. J. Sci. Eng.* **2018**, *43*, 7923–7931. [CrossRef]
11. Papadie, R.; Apostol, I. Analyzing websites protection mechanisms against DDoS attacks. In Proceedings of the 2017 9th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), Târgoviște, Romania, 29 June–1 July 2017; pp. 1–6.
12. Matos Mascarenhas, D.; Monteiro Moraes, I. PIF and ReCiF: Efficient Interest-Packet Forwarding Mechanisms for Named-Data Wireless Mesh Networks. *Information* **2018**, *9*, 243. [CrossRef]
13. Deng, X.; He, T.; He, L.; Gui, J.; Peng, Q. Performance analysis for IEEE 802.11 s wireless mesh network in smart grid. *Wirel. Pers. Commun.* **2017**, *96*, 1537–1555. [CrossRef]
14. Vijayanand, R.; Devaraj, D.; Kannapiran, B. Intrusion detection system for wireless mesh network using multiple support vector machine classifiers with genetic-algorithm-based feature selection. *Comput. Secur.* **2018**, *77*, 304–314. [CrossRef]
15. Sharma, R.K.; Issac, B.; Kalita, H.K. Intrusion detection and response system inspired by the defense mechanism of plants. *IEEE Access* **2019**, *7*, 52427–52439. [CrossRef]
16. Sikora, M.; Gerlich, T.; Malina, L. On Detection and Mitigation of Slow Rate Denial of Service Attacks. In Proceedings of the 2019 11th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), Dublin, Ireland, 28–30 October 2019; pp. 1–5.
17. Sivabalan, S.; Radcliffe, P.J. Feasibility of Eliminating IDPS Devices from a Web Server Farm. *Int. J. Netw. Secur.* **2018**, *20*, 433–438.
18. Giunta, R.; Messina, F.; Pappalardo, G.; Tramontana, E. Augmenting a Web server with QoS by means of an aspect-oriented architecture. In Proceedings of the 2012 IEEE 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, Toulouse, France, 25–27 June 2012; pp. 179–184.
19. Jyothi, V.; Wang, X.; Addepalli, S.K.; Karri, R. Brain: Behavior based adaptive intrusion detection in networks: Using hardware performance counters to detect ddos attacks. In Proceedings of the 2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID), Kolkata, India, 4–8 January 2016; pp. 587–588.
20. Labonne, M.; Olivereau, A.; Polve, B.; Zeghlache, D. Unsupervised protocol-based intrusion detection for real-world networks. In Proceedings of the 2020 International Conference on Computing, Networking and Communications (ICNC), Big Island, HI, USA, 17–20 February 2020; pp. 299–303.
21. Agrawal, N.; Tapaswi, S. Low rate cloud DDoS attack defense method based on power spectral density analysis. *Inf. Process. Lett.* **2018**, *138*, 44–50. [CrossRef]

22. Shorey, T.; Subbaiah, D.; Goyal, A.; Sakxena, A.; Mishra, A.K. Performance Comparison and Analysis of Slowloris, GoldenEye and Xerxes DDoS Attack Tools. In Proceedings of the 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Bangalore, India, 19–22 September 2018; pp. 318–322.
23. Damon, E.; Dale, J.; Laron, E.; Mache, J.; Land, N.; Weiss, R. Hands-on denial of service lab exercises using slowloris and rudy. In Proceedings of the 2012 Information Security Curriculum Development Conference, Kennesaw, GA, USA, 12–13 October 2012; pp. 21–29.
24. Sultana, N.; Bose, S.; Loo, B.T. An extensible evaluation system for DoS research. In Proceedings of the 2019 11th International Conference on Communication Systems & Networks (COMSNETS), Bangalore, India, 7–11 January 2019; pp. 344–351.
25. De Sousa Araújo, T.E.; Matos, F.M.; Moreira, J.A. Intrusion detection systems' performance for distributed denial-of-service attack. In Proceedings of the 2017 CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON), Pucón, Chile, 18–20 October 2017; pp. 1–6.
26. Park, W.; Ahn, S. Performance comparison and detection analysis in snort and suricata environment. *Wirel. Pers. Commun.* **2017**, *94*, 241–252. [CrossRef]
27. Day, D.; Burns, B. A performance analysis of snort and suricata network intrusion detection and prevention engines. In Proceedings of the Fifth International Conference on Digital Society, Gosier, Guadeloupe, France, 23–28 February 2011; pp. 187–192.
28. Habib, B.; Khurshid, F.; Dar, A.H.; Shah, Z. DDoS Mitigation in Eucalyptus Cloud Platform Using Snort and Packet Filtering—IP-Tables. In Proceedings of the 2019 4th International Conference on Information Systems and Computer Networks (ISCON), Mathura, India, 21–22 November 2019; pp. 546–550.
29. Roldán, J.; Boubeta-Puig, J.; Martínez, J.L.; Ortiz, G. Integrating complex event processing and machine learning: An intelligent architecture for detecting IoT security attacks. *Expert Syst. Appl.* **2020**, *149*, 113251. [CrossRef]
30. Ujjan, R.M.A.; Pervez, Z.; Dahal, K.; Bashir, A.K.; Mumtaz, R.; González, J. Towards sFlow and adaptive polling sampling for deep learning based DDoS detection in SDN. *Future Gener. Comput. Syst.* **2020**, *111*, 763–779. [CrossRef]
31. Hu, Q.; Yu, S.Y.; Asghar, M.R. Analysing performance issues of open-source intrusion detection systems in high-speed networks. *J. Inf. Secur. Appl.* **2020**, *51*, 102426. [CrossRef]
32. Corrêa, J.H.G.; Junior, E.A.S.; Fonseca, I.E.; Nigam, V.; Ribeiro, M.R.; Villaça, R.S. Selectivity and Autoscaling as Complementary Defenses for DDoS Protection to Cloud Services. In Proceedings of the 2019 IEEE 8th International Conference on Cloud Networking (CloudNet), Coimbra, Portugal, 4–6 November 2019; pp. 1–3.
33. Durcekova, V.; Schwartz, L.; Shahmehri, N. Sophisticated denial of service attacks aimed at application layer. In Proceedings of the 2012 ELEKTRO, Rajeck Teplice, Slovakia, 21–22 May 2012; pp. 55–60.
34. Kim, J.; Kim, H.S. Intrusion Detection Based on Spatiotemporal Characterization of Cyberattacks. *Electronics* **2020**, *9*, 460. [CrossRef]
35. Singh, K.J.; De, T. MLP-GA based algorithm to detect application layer DDoS attack. *J. Inf. Secur. Appl.* **2017**, *36*, 145–153. [CrossRef]
36. Chiba, Z.; Abghour, N.; Moussaid, K.; Rida, M. Intelligent approach to build a Deep Neural Network based IDS for cloud environment using combination of machine learning algorithms. *Comput. Secur.* **2019**, *86*, 291–317. [CrossRef]
37. Yao, Y.; Su, L.; Zhang, C.; Lu, Z.; Liu, B. Marrying graph kernel with deep neural network: A case study for network anomaly detection. In *International Conference on Computational Science*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 102–115.
38. Rosay, A.; Carlier, F.; Leroux, P. Feed-forward neural network for Network Intrusion Detection. In Proceedings of the 2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring), Antwerp, Belgium, 25–28 May 2020; pp. 1–6.
39. Ferrag, M.A.; Maglaras, L.; Moschoyiannis, S.; Janicke, H. Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study. *J. Inf. Secur. Appl.* **2020**, *50*, 102419. [CrossRef]
40. Tang, T.A.; McLernon, D.; Mhamdi, L.; Zaidi, S.A.R.; Ghogho, M. Intrusion detection in sdn-based networks: Deep recurrent neural network approach. In *Deep Learning Applications for Cyber Security*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 175–195.

41. Chastikova, V.; Sotnikov, V. Method of analyzing computer traffic based on recurrent neural networks. *J. Phys. Conf. Ser.* **2019**; *1353*, 012133. [CrossRef]
42. Tripathi, N.; Hubballi, N.; Singh, Y. How secure are Web servers? An empirical study of slow HTTP DoS attacks and detection. In Proceedings of the 2016 11th International Conference on Availability, Reliability and Security (ARES), Salzburg, Austria, 31 August–2 September 2016; pp. 454–463.
43. Velan, P.; Jirsik, T. On the Impact of Flow Monitoring Configuration. In Proceedings of the NOMS 2020—2020 IEEE/IFIP Network Operations and Management Symposium, Kissimee, FL, USA, 20–24 April 2020; pp. 1–7.
44. Vishnu, N.; Batth, R.S.; Singh, G. Denial of Service: Types, Techniques, Defence Mechanisms and Safe Guards. In Proceedings of the 2019 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE), Dubai, UAE, 11–12 December 2019; pp. 695–700.
45. Montagud, M.; De Rus, J.A.; Fayos-Jordan, R.; Garcia-Pineda, M.; Segura-Garcia, J. Open-source software tools for measuring resources consumption and DASH metrics. In Proceedings of the 11th ACM Multimedia Systems Conference, Istanbul, Turkey, 8–11 June 2020; pp. 261–266.
46. Goyal, P.; Goyal, A. Comparative study of two most popular packet sniffing tools-Tcpdump and Wireshark. In Proceedings of the 2017 9th International Conference on Computational Intelligence and Communication Networks (CICN), Girne, Cyprus, 16–17 September 2017; pp. 77–81.
47. Langthasa, B.; Acharya, B.; Sarmah, S. Classification of network traffic in LAN. In Proceedings of the 2015 International Conference on Electronic Design, Computer Networks & Automated Verification (EDCAV), Shillong, India, 29–30 January 2015; pp. 92–99.
48. Jain, R.; Kashyap, I. An QoS aware link defined OLSR (LD-OLSR) routing protocol for MANETs. *Wirel. Pers. Commun.* **2019**, *108*, 1745–1758. [CrossRef]
49. Jevtic, N.J.; Malnar, M.Z. Novel ETX-Based Metrics for Overhead Reduction in Dynamic Ad Hoc Networks. *IEEE Access* **2019**, *7*, 116490–116504. [CrossRef]
50. Fainelli, F. The OpenWrt embedded development framework. In Proceedings of the Free and Open Source Software Developers European Meeting, Bengaluru, India, 4–8 January 2008; p. 106.