Forensic-Aware Anti-DDoS Device

C.Y. Tseung, K.P. Chow
The University of Hong Kong
Hong Kong
{cytseung, chow}@cs.hku.hk

***Abstract*—When defending DDoS and other types of network attack, most products or service providers perform the protection by dropping the attack traffics. It cures the symptoms but not the disease. To help eliminate network attack, a more proactive approach is to trace back the attack source and stop the attack before it starts. Collecting the attack data is essential in attack trace-back. In this paper, we propose a live capture device to record the attack efficiently without disturbing the original network performance. The device is also integrated with anti-DDoS technique so that forensic data collection when be performed even under DDoS attacks. We made use of a network bridge and utilized packet capturing functionality provided by Linux, plus our packet storing mechanisms to build the forensic aware data collection device. The anti-DDoS protection uses machine learning to extract features of attacks, and then use a customized Bloom filter to defend attacks based on selected features. We implemented and tested the performance of the proposed technique in a lab environment.**

Keywords—live packet capture; data collection; network forensic; live forensic; Bloom filter; DDoS attack

I. INTRODUCTION

Network attack is still one of the major threat in nowadays' Internet world. DDoS protection service provider NexusGuard published a report saying that DDoS reflection attacks in Q2 2016 increased sharply in APAC Region[1]. Kaspersky[2] lab also said that resources in 70 countries were targeted by DDoS attacks in Q2 2016. Another famous attack protection service provider Cloudflare[3] also suggested the network attacks on the Internet is getting larger and larger in scale.

Purchasing such protection service bring a moment of peace, but never able to stop attacks from the source. Almost none of these service providers or firewall products provide attack traffic capture and trace back service. Their services focus on identifying and discarding attack traffics. No action will be taken to find out the hacker behind.

A live attack capturing and forensic-sound solution is in need for law enforcement agencies to analyses and trace the attack, and to take action to stop the attack right from where it starts.

Our contribution is to demonstrate our new light-weight network data capturing forensic device, with functionality of adding extended tailor-made features. We have tested the forensic device in our lab environment to compare and experiment results of our solution will be presented.

II. RELATED WORK

Researches have been done on traffic capturing, attack mitigation and anti-DDoS protection using machine learning and defending with Bloom filter.

Chan[4] proposed a method of packet capturing in his intrusion detection router system. But in his design, victim network architecture has to be changed to deploy the extra router, making it difficult for end users to setup on their own. Also, his design focused mainly on detection instead of data collection. Deri[5] discussed some efficient packet capturing methods though out Linux and Windows based system. However, he did not mention about storing the data.

Lu[6] proposed a robust and efficient detection of DDoS attacks. He tried to apply Bloom filter on incoming packets to extract features to differentiate normal and attack traffic. He also made use of machine learning to analyses traffic in time-varying patterns, and to extract features of network traffic that can be used to identify bad packets. Lee[7] conducted cluster analysis on the 2000 DARPA Intrusion Detection Scenario Specific Data Set. The purpose is to proactively detect DDoS attack in each attack scenario. The selection of the detection parameter applies Shannon entropy to packet header fields of consecutive packets. However, he fixed sampling periods.

Li[8] and Saied[9] both treated DDoS detection as a supervised learning problem. They trained different neural network with normal and different types of simulated attack data. Their experiments show that the detection accuracy is very high. Balkanli[10] even reached a 99% detection rate in his experiments for detecting DDoS using supervised learning. Despite such high detection rate, mitigation of the attack was not discussed in their studies. Seufert[11] proposed using machine learning for both detection and filtering of bad packets. However, his experiment did not show the detailed performance benchmark.

III. FORENSIC-AWARE TRAFFIC COLLECTION

In order to keep the same network topology for users, our device, in the form of a box, is transparent to the existing network. A good place of interception is the link between ISP and victim's network. The box will have two interfaces

DOI 10.1109/SPW.2018.00030

configured as a network bridge connecting the ISP and victim's network entrance point. Our packet capturing and processing happens inside the box while allowing the packet to pass through without delay.

Furthermore, we can place a hook in the incoming direction to manipulate traffic. Attack defense algorithms can be implemented in the hook, such as using a combination of Bloom filter and machine learning algorithms to block certain attack traffic. Figure 1 shows the overall network diagram of our proposed system.

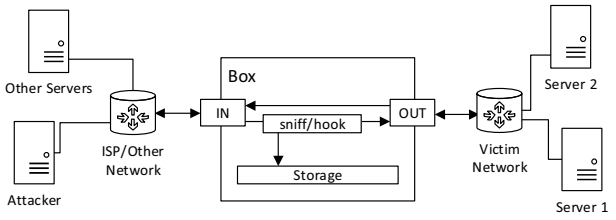

Figure 1 Forensic Box

A. Packet Storing

Before collecting packets, we first consider where to store the collected data.

a) Medium of storage

Many existing tools provided functions to dump network packets into pcap files. However, we do not have full control on these functions. We will implement our own packet dumping mechanism such that we can control the output format and perform selectively dump (e.g. sometimes we want only UDP packets with certain predefined features).

To facility further packet analysis and our proposed "storage-hot-swap" function, we utilize a MySQL database server instance to store raw packet data. A simple table containing a timestamp column and a blob column for storing the binary data of a packet is enough. This table essentially stored all information we need to replay the network traffic, thus making further forensic analysis possible.

b) Storage performance

When under attack, the traffic volume is relatively high, our device must be fast enough in writing packet raw data to storage and avoid causing further degrade to network performance. We conducted tests for both classic mechanical hard-drive(HDD) and solid-state drive(SSD).

Assume a network traffic rate of 1Gbps:

$$1\ Gbps = \left(\frac{1}{8}\right) GB/s = 125\ MB/s$$

Take a common product on market, WD BLUE WD1003FZEX[12] offers 150MB/s theoretical data transfer rate with 1TB storage and 7200rpm. While a SSD, take SAMSUNG SSD 850 PRO[13] for example, offers 520MB/s write speed. A normal HDD seems to be good enough to handle a network traffic rate of 1Gbps, however, the overheads are yet to be calculated. The actual performance will be tested later.

c) Storage hot-swaping

Assume a 1Gbps network attack and we only capture incoming packets, for a 1TB device, by a rough calculation, can hold data for:

$$\frac{1TB}{1Gpbs} = \frac{1000000MB}{125MB/s} = 8000\ s = 133min = 2.2hr$$

For an attack of higher traffic, the time will be lesser. We cannot assume a single storage device can withstand and store the whole attack, as some attacks last for even a whole day long.

Taking advantage of SATA3's hot-plug feature, we can program our system to alert user when the storage is full and then notify user to replace the existing storage with a new empty one to continue collecting data.

B. Packet Capture

Our forensic box is placed between the ISP network and customer's network. It can be placed between any two networks to monitor and capture packet data passing through them.

The two network interfaces on the box are configured as a "bridge", this configuration makes the box transparent to the network.

C. Packet Extraction

There are many kernel level networking tools available for Linux-based system that can fulfil our needs for capturing packets passing through the bridge. After capturing the packet, we have a "data collector" background process to write the packet data to storage.

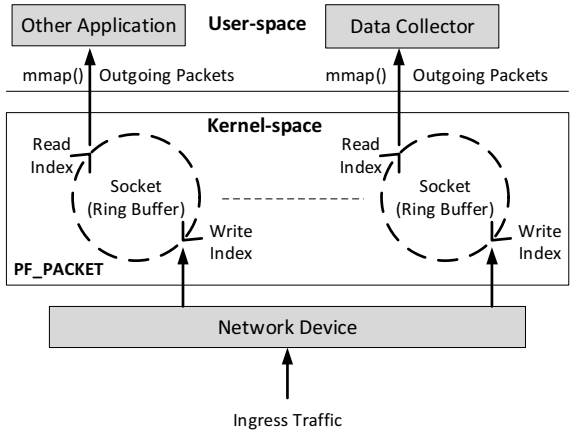

Figure 2 netsniff workflow

a) netsniff

The open source tool "netsniff-ng"[14] utilize the interfaces RX_RING and TX_RING in Linux to provider a

zero-copy packet extracting functionality. It offers the fastest performance for capturing network flows in user space. Based on the original source code, we can also modify it to add our own feature like basic filtering of unwanted packets. All packet data will then be sent to the data collector running on user-space via mmap function.

b) netfilter

Another way to capture packet is to use the hooking functionality provided by netfilter[15]. A memory buffer sk_buff containing all data of a network frame will be passed to a custom hook function registered when loading the module. In the hook function we then can examine and send the packet data to the data collector running on user-space via netlink socket.

The netfilter is also where we can implement our packet filtering features. It makes discarding network packets in the middle of the network possible. As long as we introduce algorithms like Bloom filter and machine learning, we are able to identify attack packets when they pass through the network, and discard them to protect the affected victim network.

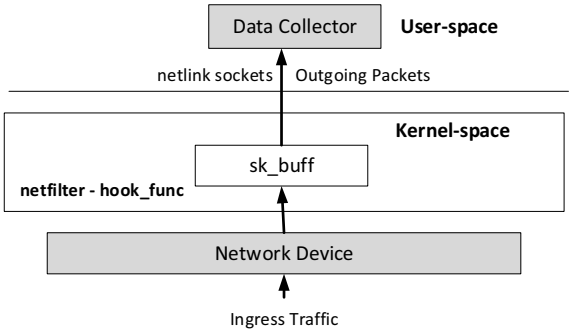

Figure 3 netfilter workflow

c) Data collector

The collector can be configured to receive packet data from either method above. We implement a "double buffer" in our data collector. Memory space for buffers of same fixed size A and B are reserved, incoming packet will be written to buffer A first, once buffer A is full, new packet data will be written to buffer B. At the same time, data on buffer A will start writing to database for permanent storage. Buffer A will be cleared after all data are written to storage. When buffer B is full, buffer A will be receiving new packets in turn, then data on buffer B will be written to storage.

Depends on the writing speed from buffer to storage, packet lost will happen if a buffer is full while the other buffer hasn't yet finished its data writing, thus new arriving packets have to be discarded. Note that this "packet lost" means only we are not able to store lost packets into our storage, "lost packets" will still pass through the bridge and network.

IV. ANTI-DDOS TECHNIQUE

In order to protect against DDoS attack while capturing forensic data, we integrated the self-learning anti-DDoS technique into the box, which combines the Bloom filter and machine learning algorithms, which as shown in Figure 4.

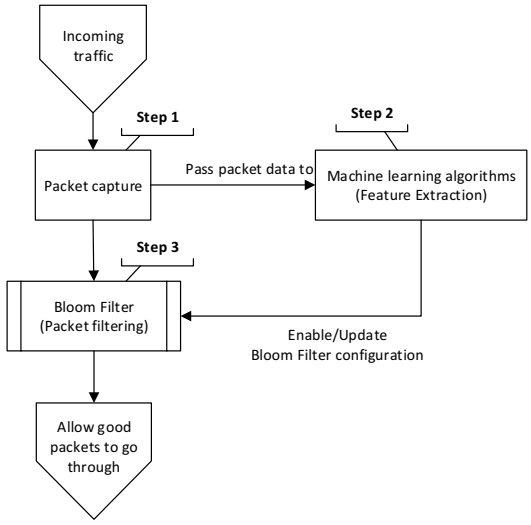

Figure 4 Anti-DDoS brief defense flow

Following are the details of the steps:

A. Step 1 – Capture and manipulation of incoming packets

To avoid changing the topology of victim's network, we make our interception transparent to the network. At the same time, we must be able to inspect, modify and block every packet coming through the network. We utilize the built-in network bridge and hook of Linux system to inspect and modify network flow transparently.

Making use of "bridge-utils"[16] provided by the Linux kernel, we are able to build a network bridge between two network interfaces, make it like a normal network switch. By this configuration, our device is transparent to victim's network and no extra changes of network structure needed to be made.

In order to examine and manipulate network packets, we use the hook provided by netfilter[16]. As we are only interested in the traffic coming into the victim network, we place a hook at NF_BR_PREROUTING[16] on the traffic incoming interface. A memory buffer sk_buff will be passed to our hook and we can examine the buffer to extract packet information. The packet will then be judged, if Bloom filter is later turned on, it will either get "allowed to pass through", "discarded" or "to be determined by the next hook" depending on our hook's return value.

In short, making use of these functionalities, we can inspect and modify network flow transparently.

B. Step 2 – Self-learning Feature Extraction

We apply machine learning algorithms in a background process to monitor network traffic to identify whether the victim is being attack. Once an attack is detected, our algorithm will learn from incoming traffics, and information of features will be extracted and automatically applied to the Bloom filter to block attack packets.

One may say that if we already know the features of attack packets, for example they all come from a fixed IP range and have the same TTL value for IP packets, we can simply apply these filtering rules in a simple rule-based firewall system to block attack traffics. Modern DDoS attacks are not as simple as a brutal force flooding. Moreover, the old-fashioned method requires lots of human interactions and rule-based firewall does not support advance filtering based on customized features of packets. We therefore automate the learning process of different attack types and judge packets using more advanced algorithms instead of simple rules.

In the next section, we will show the feasibility and details of using machine learning algorithms to detect abnormal traffic and extract features to be used in defending the attack.

C. Step 3 – Defend with Bloom filter

When an attack is happening and features of packets are extracted, such as network protocol, source address, TTL. The final step is to use our tailor-made Bloom filter to block attack traffics based features given.

Bloom filter can test if an object, which in our scenario is a packet, is inside a set. Using features selected by machine learning, we can define a set of evil packets, then easily use Bloom filter to tell whether the next packet is inside the set, i.e. the next packet is evil. For example, in a TCP SYN flood, all evil packets will have the same set of features, namely SYN bit set to on, same TTL value and same source IP. Our machine learning is expected to find out these features.

Our implementation of Bloom filter can discard spam packets having selected features in common exceeding a certain volume over a certain period of time.

Based on an implementation proposed by Chan[4], we use a multi-layer counting Bloom filter to tell whether an object with similar features is encountered multiple times. We implement N layers, and M bins per layer, then we have $N \times M$ bins in total. Each bin is simply a non-negative counter. For each packet, it will be hashed and increment one of the bins in each layer. Once a bin excess a certain threshold (overflow threshold) after a packet is hashed in, we can conclude that we have received many similar packets before. Counters cannot be increased indefinitely, thus a leaky bucket algorithm is also used to decrease all bin counters as a "cool-down" method, allowing legitimate packets to pass through continuously.

Given the basic design of our Bloom filter, the key point is what packet properties or features we use and how to do hash them. It directly affects how the Bloom filter judges whether a packet is similar with previous packets. For each packet, we first construct a byte-array, namely characteristic array, which combines the selected features of the packet. For example, for a TCP packet, we make first two bytes in the array the source port, the following byte indicates whether a SYN bit is set and so on. Thus, packets have the same features will result in the same characteristic array, which can later be judged by the Bloom filter. We hash the array by iterating N times (number of layers) through a simple shift-register random number generator and get N random value. The random values indicate which bin in a layer is hit and should be incremented. Once a bin counter exceeds the overflow threshold after a packet's hash is added, we can judge that this packet is an attack packet and thus discard it. The overflow threshold and leaky bucket algorithm together volume of spam packets that can pass the blocking. We use a parameter named *init_of_th*(initial value of the threshold) to store the overflow threshold, which can be adjusted automatically or manually during operation. In our settings, we will test the suitable value of *init_of_th* later for different network environments and situations.

D. Self-leaning Automatic Defense

Bloom filter is implemented as a kernel module, giving it the highest system privilege, also allowing us to dynamically insert or remove the filter. As minor delays are expected when analyzing packet data with machine learning, a background process will be responsible for the job to avoid degradation of network performance. This technique is implemented inside the forensic device.

The forensic device is programed to:

1. Collect normal traffic periodically when victim network is not under attack to train the attack-detection machine learning algorithms and determine suitable configuration for Bloom filter, e.g. "*init_of_th*" thresholds.
2. Activate the Bloom filter, use the packet features learnt from the machine learning analysis.
3. Adjust the option of Bloom filter from time to time during the attack.

By using the methodologies above, we have an effective technique which helps mitigate the DDoS attack.

V. ATTACK DETECTION

Before Bloom filter can function, we have to teach it what features of a packet to look at. In this section, we will discuss how to make machine learning algorithms be aware of an attack. We will show that given a traffic data input, our algorithms can extract features of the traffic automatically, which can be used in Bloom filter.

A. Preliminaries of attack dataset for machine learning

Datasets of DDoS attacks are needed to replay an attack for us to study. They are usually in format of packet captures (PCAP file) of a real or simulated DDoS attacks. There are many open source tools for simulating a DDoS attack. Our focus is on analyzing the attack traffic; we therefore captured some simulated attack traffics as PCAP files to replay the attack. We also sourced datasets from variety of places. Table I shows the datasets we used.

TABLE I. ATTACK TRAFFIC DATASETS

Source	Attack Type	Size	Description
CAIDA 2007 DDoS[16]	ICMP	21 GB	Approximately one hour of anonymized traffic traces from an ICMP Flooding attack on August 4, 2007.
ISCX IDS 2012 Data Set[17]	TCP + Normal traffic	23.4 GB	DDoS using botnet on TCP.
DARPA 2000 DDoS[18]	TCP + Normal traffic	0.1 GB	Contains two scenarios of DDoS attack on TCP on AFRL network testbed.
Lab-simulated	NTP amplification	0.3 GB	Simulated attacks on our own testbed.
Lab-simulated	UDP amplification	22 GB	Simulated attacks on our own testbed using CHARGEN protocol.
Lab-simulated	DNS DrDoS	8 GB	Simulated DrDoS attacks on our own testbed.
ISCX 2012 Data Set[17]	Normal traffic	37.9 GB	7 days of network activity of the testbed of University of New Brunswick

B. *Features of network frame*

Taking advantages of netfilter and sk_buff, we are able to examine every network frame passing through and extract their properties. Table II is a list of features we are interested in.

TABLE II. USEFUL FEATURES OF NETWORK FRAMES

Index	0	1	2	3	4	5	6
Feature	frame.len	ip.src	ip.ttl	ip.proto	ip.srcport	ip.dstport	tcp.flags

Note that ip.* means if the frame contains an IP packet, tcp.*/udp.* means if the packet is a tcp/udp packet. We are not interested in the packet's destination IP because we are only intercepting packets coming into the victim's network, thus all destination IPs will be the victim's server IPs which makes these properties not useful.

C. Entropy calculation for features

A packet alone will provide no discriminating info for us to determining whether it is an attack packet or a normal packet. Only statistical feature of consecutive packets will provide such information. Among different statistical measures, we find that entropy is an effective one in describing the change of distribution of a certain feature. The occurrence of DDoS attack will cause a difference in distribution of certain features (e.g. source IP will become dispersed in the case of IP spoofing) of network traffic.

The existing entropy-based method in literature divide the PCAP file into bins of equal time intervals (e.g. Lee[7] uses 1-min interval), and calculate the entropy using the following formula:

$$H(X) = -\sum_{i=1}^{N} \left(\frac{n_i}{S}\right) log_2 \left(\frac{n_i}{S}\right),$$

where $X = \{n_i : i = 1, \dots, N\}$, means feature X takes on N possible value, with each value occurs n_i times in the sample. And $\sum_{i=1}^{N} n_i = S$ equals to packet number in the sample. The maximum value of $H(X)$ is $log_2(S)$ and is achieved when every packet in the sample has different value. If S is different in each equal-interval bin, there scale of $H(X)$ will be different. And S is the measure of packet rate. As a consequence, the packet rate will be a dominant factor and make entropy less informative. To overcome the flaw, we fix S, so that the entropy is all of the same scale and thus comparable. The choice of S is highly heuristic. Large S will lead to stable result but incur more detection delay and less data point in training phase. Small S will cause high variance but less detection delay and more data point in training phase. In our setting, we find $S = 1000$ is a proper choice.

We gather consecutive 1000 packets to form one *record*, and then calculate the entropy of our interested features.

D. *Principal component analysis and SVM*

Principal component analysis (PCA) gives us some ideas of what learning algorithm to use. The data is drawn in scatter plot. The co-ordinate is formed by 3 most weighted principle components, they explain 65.79%, 20.10%, 9.42% of the total variance. Table III is the mapping of color to traffic for the scatter plot in Figure 5.

TABLE III. ATTACK TYPES COLOR INDICATION AND RECORDS

Attack Type	Normal	ICMP	NTP	UDP	DNS	TCP	DARPA
Color	Green	Red	Blue	Cyan	Magenta	Black	Yellow
Count	1359	359648	602	19039	11221	3769	124

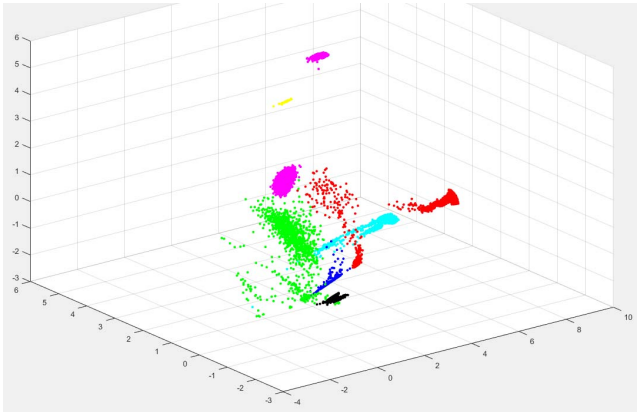

Figure 5 PCA scatter plot of different types of attack traffic

From the scatter plot, we find that different types of traffic will form different clusters. New type of attack is predicted to form clusters different than existing ones. Ideally, we can draw the boundary of normal cluster. If a data point is within the boundary, it is normal. If it is outside the boundary, it is attack. One-class Support Vector Machine (One-Class SVM) is the learning algorithm we are looking for to determine whether the network is being attack.

We trained a classifier using the library OneClassSVM from a tool scikit-learn in Python with the parameters (nu=0.001, kernel="rbf", tol=1e-5, gamma=0.05). Then verify the accuracy using 5, 3, 2-fold cross validation and obtained the following table:

TABLE IV. DETECTION ACCURACY OF NORMAL AND ATTACK RECORDS

"Normal" records	Normal	ICMP	NTP	UDP	DNS	TCP	DARPA
1088	96.3%	100%	98.4%	100%	100%	93.2%	100%
906	97.9%	100%	98.3%	100%	100%	92.5%	100%
680	95.4%	100%	99.0%	100%	100%	99.7%	100%

To reduce the chance of getting false alarm, we only trigger the "Under-attack" alarm when a number of records are being labelled as attack packets in a fixed period of time. Later when in another period of time, when less than a number of attack records are labelled, the system will back to "Normal" status. The experiment also shows that 1000 records (i.e. 1,000,000 normal packets) is enough to build the normal profile.

E. Feature selection

We assume that, when the victim network is under attack, for a short period of time, the attack type is the same. Feature selection algorithm has to be applied on incoming packets, to get the best determining feature(s) to be used in Bloom filter hashing and filtering.

When the alarm is raised (i.e. 3 consecutive records are labelled as attack), the median of each feature of the 3 attack records are chosen to form a representative of an attack record. Also, from the normal profile, we use the median of each feature of the normal records to generate the representative of a normal record.

$$Attack\ record\ \mathcal{A} = (A_0, A_1, A_2, A_3, A_4, A_5, A_6)$$

$$Normal\ record\ \mathcal{N} = (N_0, N_1, N_2, N_3, N_4, N_5, N_6)$$

The index from 0 to 6 is the same as in Table II (i.e. A_0 is median of entropy of frame.len of the 3 attack records, N_1 is median of entropy of ip.src of all normal records in normal profile etc.) Given a feature, say frame.len, if $A_0 > N_0$, the attack traffic is more dispersed than normal traffic. So, if frame.len is used as a determining feature, normal traffic will be blocked more than attack traffic. Thus, only features that Attack record is smaller than Normal record should be selected. Based on this, a subset of feature is selected. The subset of features can be ranked based on the difference of attack record and normal record. For example, if the feature subset is (0, 1, 2) and $N_2 - A_2 > N_0 - A_0 > N_1 - A_1$, the feature ranking is (2, 0, 1) in descending order. Intuitively, the bigger the difference of entropy, the more expressive the feature is. The tradeoff is that, adding more features to the string for hashing will make each packet more unique, thus less normal traffic and less attack traffic will be blocked. In order to achieve a balance between block attack traffic and pass through normal traffic, a threshold of ranking score need to be set. We find 0.7 is a proper threshold. Consider only the selected subset of features, let the difference with respect to each feature be the ranking score. Scale the scores so that the highest score is 1 and lowest score is 0. The features with scaled scores higher than 0.7 should be chosen as a determining feature, which is reliable for identifying attacking packets. For feature selection, we have tested with median, ANOVA and Linear SVM. Experimental results will be discussed in the later section.

To summarize, we can say that the attack packets are similar or even the same for in terms of some determining features. Bloom filter can identify an attack packet based on these features.

VI. EXPERIMENTS

We have conducted two sets of experiment: one set on performance of forensics traffic capture and another set on performance of defending against DDoS attacks.

A. Experiments on Forensic Traffic Capture

The purpose of these experiments is to test the performance and actual data collecting ability. These experiments also help us find out the best configuration and hardware requirement for different network environment.

a) Experimental Settings

A testbed, as shown in Figure 6, was built to test the performance of our proposed system. A network bandwidth measurement tool "iPerf"[19] was used to generate and receive network traffic. The traffic generation rate is adjustable though start up parameters "-b". Our forensic box is placed between two servers, one for sending and one for receiving the traffic. The link speed of network interfaces are all 1Gbps. Experiment data are collected from the sender, receiver and our box.

By calculating the packets sent, packets received and packets stored on database over a certain period of time, we can calculate how many packets we are able to capture and store, i.e. the performance of our packet capture system.

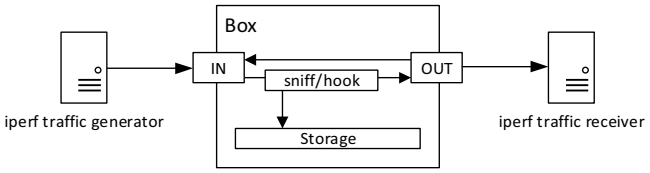

Figure 6 Forensic traffic capture testbed setup

Bitrate to be tested are (Mbps):

100, 200, 300, 400, 500, 600, 700, 800, 900, 1000

Storage device used are respectively:

- A 7200rpm HDD claimed to have a maximum of 150MB/s write speed
- A SSD claimed to have 520MB/s write speed

The following properties are measured in our experiments:

- Duration – the period of time the test was performed
- Packets Sent/Received/Dropped – the number of packets sent by the generator, received by the receiver, and the different between these two due to network congestion or malfunction of the network bridge in the box, etc.)
- Packets Stored/Lost – the number of packets successfully/failed to store into database, however it doesn't necessarily mean it can't reach the receiver
- Lost Rate – the percentage of packets passing through the network but not being able to store in database

b) Experimental Results

Appendix A and Appendix B showed the testing result for using HDD as storage and SSD.

From the result, we find that even the theoretical speed of HDD is 150MB/s, it is far from enough to use as a storage device in our scenario. It may due to the high latency and high seek time of classic mechanical hard drive. A SSD performs good when the network traffic is below 1Gbps. As in our testbed setup, we use direct link between interfaces, the good linking quality resulted in no packet drop throughout our experiments.

B. Experiments on Anti-DDoS Attacks

The purpose of these experiments is to test the performance and blocking accuracy of our proposed technique. These experiments also help us find out the best machine learning algorithm to be used for feature extraction and the suitable configuration for Bloom filter for different network environment. We will use different types of attack data as source, i.e. the input of our machine learning algorithms, to do feature extraction then apply these features in Bloom filter for blocking.

a) Experiment Settings

A testbed, as shown in Figure 7, was built to test our proposed technique. PCAP files were replayed to simulate the attacking scenario. However, if we play both normal traffic and attack traffic to the same destination, it will be hard to identify which packet is normal and which comes from the attack traffic. Thus, we use 2 servers for receiving traffic of normal and traffic of attack to calculate the blocking rate of attack traffic and passing rate of normal traffic. As all traffics will go through the SmartBox, it simulates the situation that the victim is being attack during normal business operations.

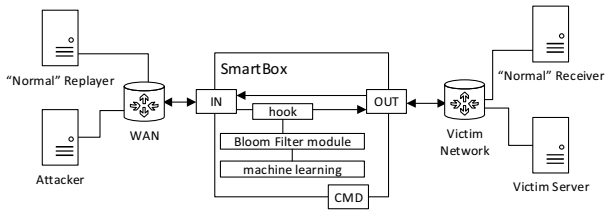

Figure 7 Anti-DDoS attack testbed setup

"Normal" Replayer is the PC plays the normal traffic to the "Normal" Receiver. Attacker is the PC that plays the attacking PCAP to the Victim Server. Wireshark is used in both receiving server to calculate the packet volume received. Compare with the packets sent, which is known and controllable, we can measure the blocking efficiency of our technique.

Chan's[4] implementation of Bloom filter uses fixed feature list: ip.dst, ip.proto, ip.dstport in hashing, which is in the case of a router. In our case, since the destination IP, protocol and port will mostly be like the same (traffics coming in to a single or limited victim servers and services), we use ip.src, ip.proto, ip.srcport as the enabled Bloom filter determining feature for the baseline comparison.

b) Experiment 1

We set both normal packet rate and attack packet rate to 15000/s. And bloom filter parameter init_of_th = 20000. This is to simulate the DDoS attack when the attack traffic does not lead to a significant increase in the traffic volume. In this experiment we calculate the followings: Attack Block Rate, the percentage of attack traffic is blocked; Normal Pass Rate, the percentage that normal traffic passes through. The higher the better for both values. The result shows that the Baseline option only works for NTP attack, and ANOVA, SVM, Median improves a lot in Attack Block Rate and Normal Pass Rate. ANOVA and Median outperforms SVM in most types except TCP attacks.

Appendix C showed the experiment result. We note from the result that Normal Pass Rate is very low, this is due to the configuration of init_of_th parameter. The higher init_of_th is, the more traffic volume will be allowed to pass by Bloom

filter. In our next experiment we set init_of_th to 80000, so that normal traffic will pass through when Bloom filter is on Baseline Option. The proposed SVM feature selection algorithm gives poor "normal pass rate" compare to the other two, thus, we skip the test for SVM in next experiments.

c) Experiment 2

We set both normal packet rate and attack packet rate to 15000/s. And bloom filter parameter init_of_th = 80000.

The result in Appendix D showed that Bloom filter let almost all normal traffic pass through and blocks very limited amount of attack traffic. This is because init_of_th determines the threshold packet rate Bloom filter let pass through. And it does not differentiate between attack packet and normal packet, it will let attack packet pass through if attack packet rate is similar to normal packet rate.

In many cases, DDoS attack will incur a bit rate much higher than the victim's normal traffic. We set attack packet rate = 75000/s to simulate brutal force DDoS attack, normal packet rate = 15000/s and bloom filter parameter init_of_th=80000.

From the result in Appendix E, we find that both ANOVA (with feature selection threshold = 0.4) and Median (with feature selection threshold=0.7) improves Attack Block Rate and Normal Pass Rate significantly. ANOVA outperforms Median in UDP attack. In other types of attack, the performance of ANOVA and Median are almost the same. We can the configuration of Bloom filter option follows the following procedure:

1. Alarm is raised when One-Class SVM detects 3 consecutive attack records, Bloom filter is on and use the features according to the output of Median Approach.
2. Compute the output of Median Approach for every 3 new incoming attack records. Update features enabled when 3 consecutive feature selection output is different than the current option.
3. When 500 attack records are generated, use ANOVA to select the features accordingly. Update features selected every 500 attack records.

The configuration procedure uses Median approach as "warm-up" and ANOVA when there are enough records. The advantage over using Median approach at all time is twofold, first ANOVA produce more stable results than Median approach, second the update of ANOVA is less frequent than Median approach and thus reduces the computation overhead.

The feature selection algorithm is based on the statistics of normal profile and incoming attack records and does not rely on any attack profile. It will work for both known and unknown attacks.

VII. Conclusion And Future Works

We proposed a forensic box that is able to capture live traffic and perform basic attack defense. We then conducted experiment to test the completeness of network packets we captured. We further conclude that for such a network traffic capturing device, it's essential to use a SSD as the storage device.

A lot of work needed be done in the future, including:

1. Latency measurement. Average delays after inserting our forensic box into the network should be measures.
2. Testing for higher bandwidth. As per our result, a single SSD can handle 1Gbps traffic, for higher traffic rate, we may need to implement RAID configuration into the box.
3. Implementation of storage hot-swapping feature.
4. Find out the other minimum hardware requirement besides storage device, e.g. CPU and memory.

References

[1] NexusGuard: DDoS Threat Report Q2 2016 - APAC Region, https://www.nexusguard.com/hubfs/Nexusguard_Q2_2016_Threat_Report_APAC.pdf, 2016

[2] Kaspersky Lab: Kaspersky DDoS Intelligence Report for Q2 2016, https://securelist.com/analysis/quarterly-malware-reports/75513/kaspersky-ddos-intelligence-report-for-q2-2016/, 2016

[3] Cloudflare: 400Gbps: Winter of Whopping Weekend DDoS Attacks, https://blog.cloudflare.com/a-winter-of-400gbps-weekend-ddos-attacks/ 2016

[4] Eric Y. K. Chan, H. W. Chan, K. M. Chan, P. S. Chan, Samuel T. Chanson, M. H. Cheung, C. F. Chong, K. P. Chow, Albert K. T. Hui, Lucas C. K. Hui, S. K. Ip, C. K. Lam, W. C. Lau, K. H. Pun, Y. F. Tsang, W. W. Tsang, C. W. Tso, D. Y. Yeung, S. M. Yiu, K. Y. Yu, Weihua Ju: Intrusion Detection Routers: Design, Implementation and Evaluation Using an Experimental Testbed, IEEE Journal on Selected Areas in Communications, vol. 24, no. 10, 2006

[5] Deri, Luca & S P A Via, Netikos & Km, Brennero & La Figuretta, Loc.: Improving Passive Packet Capture: Beyond Device Polling, http://luca.ntop.org/Ring.pdf, 2017

[6] K. Lu, D. Wu, J. Fan, S. Todorovic, A. Nucci: Robust and efficient detection of DDoS attacks for large-scale internet, Computer Networks, vol 51, pp. 5036-5056, 2007

[7] Keunsoo Lee, Ki Hoon Kwon, Younggoo Han, Sehun Kim: DDoS attack detection method using cluster analysis, Expert Systems with Applications: An International Journal, vol 34, issue 3, pp. 1659-1665, 2008

[8] J. Li, Y. Liu, G. Lin.: DDoS attack detection based on neural network, Aware Computing (ISAC), IEEE 2010 2nd International Symposium, pp. 196-199, 2010

[9] Alan Saied, Richard E. Overill, Tomasz Radzik: Detection of known and unknown DDoS attacks using Artificial Neural Networks, Neurocomputing, vol 172, pp. 385-393, 2016

[10] E. Balkanli, J. Alves and A. N. Zincir-Heywood: Supervised learning to detect DDoS attacks, 2014 IEEE Symposium on Computational Intelligence in Cyber Security (CICS), Orlando, FL, 2014, pp. 1-8, 2014

[11] S. Seufert and D. O'Brien: Machine Learning for Automatic Defence Against Distributed Denial of Service Attacks, 2007 IEEE International Conference on Communications, Glasgow, 2007, pp. 1217-1222, 2007

[12] WD: WD Black – Desktop Performance Internal Storage Hard Drive, https://www.wdc.com/products/internal-storage/wd-black-desktop.html, 2017

[13] Samsung: 850 PRO | Consumer SSD | Samsung V-NAND SSD, http://www.samsung.com/semiconductor/minisite/ssd/product/consumer/850pro.html, 2017

[14] SysTutorials: netsniff-ng - the packet sniffing beast - Linux Man Page, https://www.systutorials.com/docs/linux/man/8-netsniff-ng/, 2017

[15] Richard Kelly: Common Functionality in the 2.6 Linux Network Stack, http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/linux-2-6-network-stack-paper.pdf, 2010

[16] CAIDA: The CAIDA UCSD "DDoS Attack 2007" Dataset, https://www.caida.org/data/passive/ddos-20070804_dataset.xml, 2007

[17] UNB: UNB ISCX Intrusion Detection Evaluation DataSet, http://www.unb.ca/research/iscx/dataset/iscx-IDS-dataset.html, 2010

[18] Lincoln Laboratory: 2000 DARPA Intrusion Detection Evaluation Data Set, https://www.ll.mit.edu/ideval/data/2000data.html, 2000

[19] iPerf - The TCP, UDP and SCTP network bandwidth measurement tool, https://iperf.fr/, 2017

VIII. APPENDICES

Appendix A

Performance benchmarking using HDD as storage

Bitrate (Mbps)	Duration (s)	Packets Sent	Packets Received	Packet Dropped	Packets Stored	Packet Lost	Lost Rate
100	950.7	729818	729818	0	704801	25017	3.40%
200	967.8	1234423	1234423	0	921370	313053	25.00%
300	960.7	1722670	1722670	0	971666	751004	43.60%
400	973	2159463	2159463	0	1022420	1137043	52.70%
500	986.3	2597232	2597232	0	1026000	1571232	60.50%
600	968.4	3039786	3039786	0	1044755	1995031	65.60%
700	970.2	3562648	3562648	0	1076640	2486008	69.80%
800	980.9	3969160	3969160	0	1070134	2899026	73.00%
900	991.2	4363349	4363349	0	1105341	3258008	74.70%
1000	971.5	4799481	4799481	0	1112474	3687007	76.80%

Appendix B

Performance benchmarking using SSD as storage

Bitrate (Mbps)	Duration (s)	Packets Sent	Packets Received	Packet Dropped	Packets Stored	Packet Lost	Lost Rate
100	963.6	714391	714391	0	714380	11	0.00%
200	962.8	1201667	1201667	0	1201657	10	0.00%
300	986.9	1674583	1674583	0	1674556	27	0.00%
400	970.8	1796763	1796763	0	1796742	21	0.00%
500	1002.9	2206004	2206004	0	2205956	48	0.00%
600	942.4	2954443	2954443	0	2953432	1011	0.03%
700	973.4	3412400	3412400	0	3407392	5008	0.15%
800	950.2	3975492	3975492	0	3937488	38004	0.96%
900	1005.6	4610815	4610815	0	4464809	146006	3.20%
1000	950.1	5147195	5147195	0	4598193	549002	10.70%

Appendix C

Experiment 1 results

Input Attack Type	Learning Algorithm	Feature Automatically Selected	Attack Block Rate	Normal Pass Rate
ICMP	Baseline	ip.src, ip.proto, ip.srcport	0.05%	67.21%
	ANOVA	frame.len, ip.srcport, tcp.flags	94.09%	80.33%
	SVM	frame.len, ip.proto	94.29%	44.26%
	Median	frame.len, ip.srcport	94.28%	72.13%
NTP	Baseline	ip.src, ip.proto, ip.srcport	96.04%	75.81%
	ANOVA	frame.len, ip.srcport,ip.dstport, tcpflags	94.53%	85.48%
	SVM	ip.src, tcpflags	96.30%	33.87%
	Median	frame.len, ip.srcport	96.38%	79.03%
UDP	Baseline	ip.src, ip.proto, ip.srcport	0.00%	66.13%
	ANOVA	tcp.flags	100.00%	16.13%
	SVM	tcp.flags	100.00%	16.13%
	Median	ip.srcport	41.26%	67.74%
DNS	Baseline	ip.src, ip.proto, ip.srcport	0.00%	67.21%
	ANOVA	ip.srcport, tcp.flags	90.84%	62.30%
	SVM	tcp.flags	100.00%	26.23%
	Median	ip.srcport	91.11%	60.66%
TCP	Baseline	ip.src, ip.proto, ip.srcport	0.00%	66.13%
	ANOVA	frame.len, ip.ttl, ip.proto, ip.dstport	75.74%	70.97%
	SVM	ip.ttl, ip.dstport	99.99%	70.97%
	Median	frame.len, ip.dstport	75.59%	70.97%
DARPA	Baseline	ip.src, ip.proto, ip.srcport	0.00%	66.13%
	ANOVA	frame.len, tcp.flags	99.99%	40.32%
	SVM	tcp.flags	99.99%	13.57%
	Median	frame.len	99.99%	40.32%

Appendix D

Experiment 2 results

Input Attack Type	Learning Algorithm	Feature Automatically Selected	Attack Block Rate	Normal Pass Rate
ICMP	Baseline	ip.src, ip.proto, ip.srcport	0.00%	100.00%
	ANOVA	frame.len, ip.srcport, tcp.flags	0.02%	98.39%
	Median	frame.len, ip.srcport	0.00%	100.00%
NTP	Baseline	ip.src, ip.proto, ip.srcport	0.00%	100.00%
	ANOVA	frame.len, ip.srcport,ip.dstport, tcpflags	0.00%	98.39%
	Median	frame.len, ip.srcport	0.00%	100.00%
UDP	Baseline	ip.src, ip.proto, ip.srcport	0.00%	100.00%
	ANOVA	tcp.flags	49.65%	95.16%
	Median	ip.srcport	0.00%	100.00%
DNS	Baseline	ip.src, ip.proto, ip.srcport	0.00%	100.00%
	ANOVA	ip.srcport, tcp.flags	0.00%	100.00%
	Median	ip.srcport	0.00%	100.00%
TCP	Baseline	ip.src, ip.proto, ip.srcport	0.00%	98.39%
	ANOVA	frame.len, ip.ttl, ip.proto, ip.dstport	0.00%	98.39%
	Median	frame.len, ip.dstport	0.00%	100.00%
DARPA	Baseline	ip.src, ip.proto, ip.srcport	0.00%	100.00%
	ANOVA	frame.len, tcp.flags	34.99%	98.39%
	Median	frame.len	48.27%	96.77%

Appendix E

Experiment 2 results with updated parameters

Input Attack Type	Learning Algorithm	Feature Automatically Selected	Attack Block Rate	Normal Pass Rate
ICMP	Baseline	ip.src, ip.proto, ip.srcport	0.00%	98.39%
	ANOVA	frame.len, ip.srcport, tcp.flags	94.09%	100.00%
	Median	frame.len, ip.srcport	94.07%	100.00%
NTP	Baseline	ip.src, ip.proto, ip.srcport	95.70%	98.39%
	ANOVA	frame.len, ip.srcport,ip.dstport, tcpflags	95.70%	98.39%
	Median	frame.len, ip.srcport	95.70%	100.00%
UDP	Baseline	ip.src, ip.proto, ip.srcport	0.55%	100.00%
	ANOVA	tcp.flags	100.00%	91.94%
	Median	ip.srcport	41.16%	100.00%
DNS	Baseline	ip.src, ip.proto, ip.srcport	1.34%	100.00%
	ANOVA	ip.srcport, tcp.flags	91.59%	91.94%
	Median	ip.srcport	91.20%	93.55%
TCP	Baseline	ip.src, ip.proto, ip.srcport	0.00%	100.00%
	ANOVA	frame.len, ip.ttl, ip.proto, ip.dstport	71.48%	100.00%
	Median	frame.len, ip.dstport	72.13%	100.00%
DARPA	Baseline	ip.src, ip.proto, ip.srcport	0.00%	98.39%
	ANOVA	frame.len, tcp.flags	100.00%	96.77%
	Median	frame.len	100.00%	96.77%