Edmund: Entropy based attack Detection and Mitigation engine Using Netflow Data

Mohammad Hashem Haghighat
Department of Automation
Tsinghua University
Beijing, China
l-a16@mails.tsinghua.edu.cn

Jun Li
Research Institute of Information Technology
Tsinghua University
Beijing, China
junl@mail.tsinghua.edu.cn

ABSTRACT

Dozens of signature and anomaly based solutions have been proposed to detect malicious activities in computer networks. However, the number of successful attacks are increasing every day. In this paper, we developed a novel entropy based technique, called Edmund, to detect and mitigate Network attacks. While analyzing full payload network traffic was not recommended due to users' privacy, Edmund used netflow data to detect abnormal behavior.

The experimental results showed that Edmund was able to highly accurate detect (around 95%) different application, transport, and network layers attacks. It could identify more than 100K malicious flows raised by 1168 different attackers in our campus. Identifying the attackers, is a great feature, which enables the network administrators to mitigate DDoS effects during the attack time.

CCS CONCEPTS

•**Computer systems organization** → **Embedded systems;** *Redundancy;* Robotics; •**Networks** → Network reliability;

KEYWORDS

Network Attacks, Entropy, Attack Detection and Mitigation, Malicious Flows

1 INTRODUCTION

Internet bandwidth was improved dramatically, during these last years, in which wider range of online services like P2P file sharing, Voice Over IP (VOIP), e-commerce, internet banking, are used every day. This issue raised a new challenge in network environment.

The number of abnormal network traffic raised drastically. Several illegal activities such as worm propagation, Denial of Service (DoS) Attack, Scan, Botnets, and Flooding Attacks are performed every day. In recent decades, due to botnets and numerous open source attack tools[1], attackers launch attacks especially massive Distributed DoS (DDoS), easily.

[1]Such as Low Orbit Ion Canon (LOIC), XOIC, DDoSIM, DAVOSET, PyLoris, and so on.

ICCNS'18, Qingdao, China
© 2016 Copyright held by the owner/author(s). 123-4567-24-567/08/06...$15.00
DOI: 10.475/123_4

Each successful attack causes huge damage on any enterprises. More than 20% of companies have seen at least one successful DDoS attack against their network, in which a complete DDoS causes 444K USD financial loss on the average [9]. Akamai in [1] reported that the most attractive area for the attackers is gaming (around 80%). There are other considerable reports about the power and impact of network attacks in the world [15, 21, 10].

Generally, network attack detection techniques are categorized into "Signature based" and "Anomally based" detection algorithms. Signature based techniques use a set of pre-defined rules to detect network attacks. In this approach, the network traffic is compared with the rules, so that in case of finding a match, the traffic is considered as malicious [5, 2, 11, 17, 7]. Although, signature based techniques have no false positive, the false negative rate is directly related to its signature database. Namely, the richer the ruleset, the smaller the false negative rate. As a result, it is not possible to detect zero-day attacks by using signature based methods.

Anomaly based detection is another approach to find malicious traffic, in which network and/or host level features are extracted to determine abnormal behavior with a controlled false positive and false negative rates. An example to abnormal behavior is a sudden jump in the rate of packets, which can be caused by worm propagation.

The main challenge of anomaly based detection systems is to minimize false positive and false negative rates in different situations. Network behavior is varying during time. In addition, different networks serve unalike services. Hence, network statistics are changing inconsistently, and proposed anomaly based detection system needs to deal with the changes.

As a practical solution, we proposed a novel technique called Edmund to detect and mitigate various kinds of network attacks using netflow data as soon as possible. Edmund computes the entropy values of Source Network Address, Destination IP Address, Source Port Number, Destination Port Number, TCP flag, Number of Packets per Flow, amd Flow Size. Higher entropy value means less related flows. The extracted values are used to detect malicious flows. Entropy has a great advantage to detect malicous behavior, in which, in the normal case, the entropy values of different network features change smoothly. While, in the attack time, a significant change is seen in the entropy value/s of one or more features [19].

The proposed method consists of learning and detection phases. In the learning phase the system is trained by normal traffic and the detection phase, the trained model is used to detect abnormal traffic. All the malicious are filtered automatically, while those flows considered as normal are used to refine the trained model.

As a result, the learned model is changing during time according to the network behavior. This is the main contribution of Edmund model.

2 RELATED WORK

There has been several entropy based solutions to detect abnormal behavior in computer networks. Zhang et al. in [22] proposed an entropy based method aiming to avoid false positive and false negative. Although, the proposed method provided accurate results, their method needed more resources to analyze traffic and was very time consuming.

Mehdi et al. in [13] used maximum entropy estimation to observe benign traffic distribution and then, utilized it as a baseline to detect network attacks. However, the authors focused on low rate bandwidth traffic like home environment to test their method. Ma and Chen [12] employed Lyapunov exponent and computed the entropy values of source and destination IP addresses in different time intervals to detect DDoS attacks, with a low false positive rate.

In [16] Terzi et al. proposed an unsupervised learning detection system based on analyzing netflow data to detect UDP flooding attacks. Vidal et al. in [18] also, proposed an entropy based model called AIS to predict anomalies, which provides the ability to detect flooding attacks. Although, AIS achieved reasonable results in simulated environment, the scenarios of real world may affect the results.

David and Thomas in [4] focused on decreasing the overhead of detection and proposed a fast entropy approach to detect DDoS attacks. Flow based analysis as well as fast entropy approach reduced the detection time, significantly, while having reasonable accuracy.

Jun et al. [8] proposed a DDoS detection technique using both entropy of packet header fields and traffic volume, in which, at the step, they used the traffic volume information to detect suspicious behavior. Then, they further analyzed suspicious traffics using the entropy of packet header fields to identify attacks.

In [6], Giotis et al. proposed an entropy based anomaly detection technique. They extracted header fields including source/destination MAC addresses and 5-tuples. Then, they detected network anomalies based on entropy of the distribution value of each field. They labeled these anomalies as DDoS attack or worm propagation. In addition, Rui Wang et al. in [20] employed entropy based technique in SDN to detect DDoS attacks. The proposed solution runs over OpenFlow to reduce the heavy communication between controller and switches, and detect DDoS attacks locally. However, identification of attackers during the attack time provides a powerful mechanism to mitigate the attack costs effectively. As a result, Edmund method was proposed in this paper as an entropy based DDoS detection engine to find and filter malicious flows during the attack time, automatically.

3 EDMUND METHOD

This section is aimed to describe the proposed novel technique (Edmund) in detail. Edmund uses a machine learning model to detect network attacks. It includes "learning" and "detection" phases, in which, at first, the system is trained by normal traffic to generate expected behavior. Then, in the detection phase, Edmund analyzes incoming flows and assign them benign or malicious labels, according to the trained model. The overall procedure of Edmund is described by Figure 1.

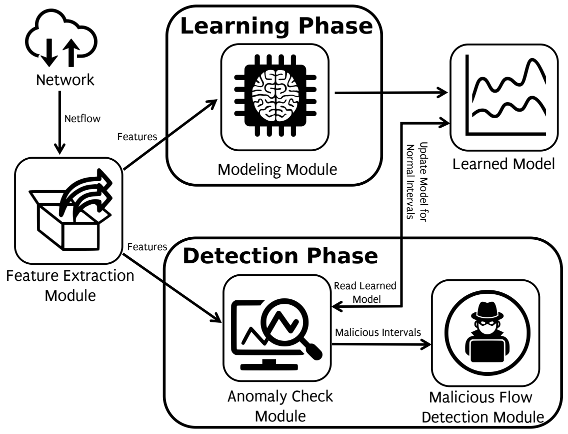


Figure 1: Edmund Architecture.

As illustrated in Figure 1, our proposed model consists of four important modules: "Feature Extraction", "Learning", "Anomaly Check" and "Malicious Flow Detection". The rest of this section describes each module in detail.

3.1 Feature Extraction Module

Extracting statistical features from netflow data is the first step in both learning and detection phases. We used Shannon's Entropy proposed in [14] as it provides a better insight compared to traditional anomaly detection systems.

Definition 3.1. Let $X = \{x_1, x_2, ..., x_n\}$ be the set of qualitative variable X and $P = \{p_1, p_2, ..., p_n\}$, be their probabilities. The Shannon's Entropy of given set is computed as follows.

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log_n P(x_i) \tag{1}$$

For each time interval, Edmund computes the entropy values of seven features containing source network address, destination IP address, source and destination port numbers, TCP flag, number of packets per flow, and flow size. Higher entropy value means less related flows. In the learning phase, Edmund uses these seven entropy values to build expected behavior, while in the detection phase, these values reveals malicious activities. In fact, the entropy values should be close to the expected results. However, in the attack time, there are several reasons to have significantly less (or sometimes more) entropy values. The attack can be performed against a single victim, which increases the number of flows with the same destination IP address. Moreover, DDoS can be launched by a botnet that creates a huge number of flows with the same "number of packets" or "flow size" features. Another situation is SYN Flood attack, which netflows' TCP flag are the same. As a result, the entropy values goes down incredibly.

3.2 Modeling Module

This module is responsible to predict the expected result by computing the "average" and "standard deviation" values of each individual feature during time, incrementally.

Definition 3.2. Let $F_{n+1} = \{f_{1_{n+1}}, f_{2_{n+1}}, ..., f_{7_{n+1}}\}$ explains the entropy values of extracted seven features of the $n+1^{th}$ time interval. Also, $M_n = \{m_{1_n}, m_{2_n}, ..., m_{7_n}\}$ and $\Sigma_n = \{\sigma_{1_n}, \sigma_{2_n}, ..., \sigma_{7_n}\}$, be the set of average and standard deviation of extracted features based on the first n time intervals, respectively. The average and standard deviation values of $n+1^{th}$ interval is computed incrementally, by the following equations.

$$m_{i_{n+1}} = \frac{(m_{i_n} \times n) + f_{i_{n+1}}}{n+1} \qquad 1 \le i \le 7 \quad (2)$$

$$\sigma_{i_{n+1}} = \sqrt{(\frac{n-1}{n} \times \sigma^2_{i_n}) + \frac{(f_{i_{n+1}} - \sigma^2_{i_n})^2}{n+1}} \qquad 1 \le i \le 7 \quad (3)$$

The computed average and standard deviation values of the features are used to calculate "Expected Range" as follows.

Definition 3.3. Let $M_n = \{m_{1_n}, m_{2_n}, ..., m_{7_n}\}$ and $\Sigma_n = \{\sigma_{1_n}, \sigma_{2_n}, ..., \sigma_{7_n}\}$, be the set of average and standard deviation of the features based on the first n time intervals, respectively. The Expected Range of seven extracted features, $ER_n = \{ER_{1_n}, ER_{2_n}, ..., ER_{7_n}\}$, is computed as below:

$$ER_{i_n} = [m_{i_n} \pm (\alpha \times \sigma_{i_n})] \qquad 1 \le i \le 7 \quad (4)$$

where α is configured to manage system false positive and false negative rates.

3.3 Anomaly Check Module

The next step is to check the feature values with the computed expected results, in which the entropy values are checked by their corresponding expected range. It is important to note that the network behavior is changing during time. Hence, when the system determines that the input interval is normal, the learned model is updated. The whole procedure is described in Algorithm 1.

Algorithm 1: Anomaly Check Module

```
input: $F$ as feature set and $ER$ as expected range
output: Label of seven extracter features
foreach feature in F do
    if (feature is in $ER_{feature}$)
        label[feature] ⟵ "normal"
    else
        label[feature] ⟵ "malicious"
return label
```

It is noteworthy that the time interval is considered as normal when all its seven features are placed in their corresponding expected range.

3.4 Malicious Flow Detection

As described before, more similar behavior, caused by running distributed attacks, leads to lower entropy value. As a result, one can detect similar (malicious) records by grouping all flows, based on the out-of-range feature(s). In addition to detecting abnormal flows, this module assigns malicious probability to each flow based on the number of similar features. The more similar the features, the higher probability of the malicious flow.

As an example assume that a SYN Flooding attack using IP Spoofing technique is happening in the network, in which, so many flows with the same destination IP address and port number, TCP Flag, number of packets, and flow size are seen in the netflow data. As a result, the entropy values of these five features go down significantly. Edmund considers this time interval as malicious. Then it tries to detect malicious flows by grouping them according to these five features. Edmund reports all malicious flows according to the rules provided in Figure 2, in which the malicious probability is $\frac{5}{7} \times 100 = 71.4\%$.

Detected Attack Types
- Network Layer
  - Port Scan ... number of flows with the same dst_ip and different dst_port is high
  - Range Scan ... number of flows with the same dst_port and different dst_ip is high
- Transport Layer
  - TCP
    - Land ... src_ip = dst_ip and src_port = dst_port
    - TCP Flooding ... packet_count is large and flow_size is large
    - SYN Flood ... number of SYN packets is large
  - UDP
    - Ping Pong ... dst_port = reflecting port and src_port = reflecting port
    - Fraggle ... dst_port = reflecting port and dst_ip = broadcast
    - UDP Flooding ... packet_count is large and flow_size is large
  - ICMP
    - Smurf ... echo request and dst_ip = broadcast
    - Ping of Death ... flow_size/packet_count is too high
    - ICMP Flooding ... packet_count is large and flow_size is large
- Application Layer
  - TCP Anomalies ... FTP, HTTP, Telnet, SMTP, SSH, DNS, HTTPS, POP3, NTP, RPC, IRC, IMAP, NetBIOS, MySQL, RDP, MSSQLServer, Oracle, and SIP Anomaly
  - UDP Anomalies ... DNS, IRC, IMAP, RPC, RDP, and SIP Anomaly

Figure 2: DDoS Attack Type Detection Rules.

4 EVALUATION

In this section, the Edmund method was evaluated with the help of two different netflow data as below.

(1) Two days real network traffic of the campus, captured from 2017-07-10 00:00:00 to 2017-07-11 23:59:59. The total number of flows were around 1.1 million.
(2) The first day of CTU-13 botnet traffic dataset, captured in CTU university in 2011 [3]. The dataset have a large real botnet traffic[2] mixed with normal and background data, including about 2.8 million flows, which around 41K are malicious.

Apache Spark framework was used to create and analyze the data using the Edmund model. The evaluation part served three purposes:

(1) To highlight the important need to update the model during detection phase.
(2) To show the usage of malicious flow detection module, which tries to detect the attackers, so that in the attack time, they can be filtered to mitigate the attack effects.
(3) To observe the accuracy of Edmund.

4.1 System Learning

As described in the previous section, system learning plays the most important role in the accuracy of the model. Also, the computed entropy value of the features should be around one. Figure 3 shows the entropy values of the features during the learning phase.

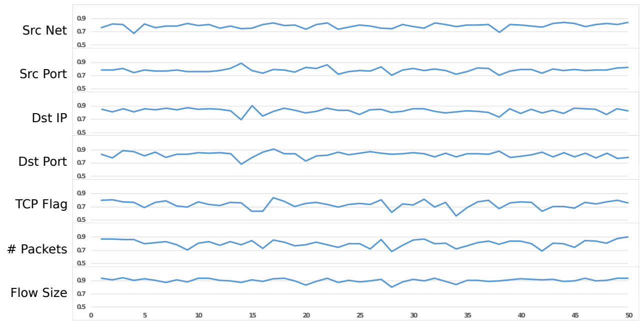


Figure 3: The Entropy values during the Learning Phase.

We computed the expected range of each feature, according to the average and standard deviation of their entropy values during the learning phase.

4.2 Malicious Behavior Detection

Edmund model used the expected range to distinguish abnormal traffic from normal one. Figure 4 illustrates the result of detection during our experience, while the expected range's lower and upper bounds are highlighted by red and green colors respectively. It is worth mentioning that, the network behavior is changing during time, which the learned model was updated during detection phase, in case of considering the interval as normal. Figure 4 highlights this important requirement as well.

As depicted in Figure 4, most of the time intervals were normal because their entropy values were in the expected range (around %89). However, the entropy values of some intervals were significantly lower than the starvation range. An example can be the time interval of the second day, 03:57 A.M. The entropy values of "Source Network", "Source Port", "destination IP", and "TCP Flags" were close to 0.05 which were magnificently lower than the lower boundary of this range. In fact, most captured traffic during this time interval was sent by some attackers via the same network, using one specific source port, and against one victim server. Table 1 summarizes the results of this module.

[2]Neris Botnet

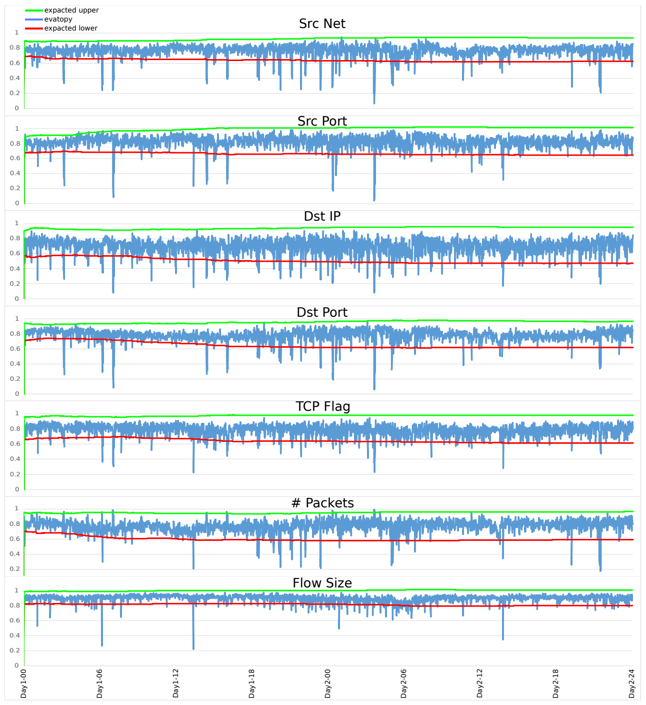


Figure 4: The Result of Detection during our Experience

Table 1: Summary of Malicious Intervals.

	Intervals		
Item	Total	Normal	Malicious
Quantity	2880	2563	317

4.3 Malicious Flow Detection

The next step is to deeply analyzed malicious time intervals to extract attack flows, which causes the entropy values of the features to become lower than the expected/starvation range. This procedure is done in the malicious flow detection module. The result is summarized in Table 2.

As highlighted in Table 2, the number of captured flows during these two days was 1108156, while more than 120K were labeled as malicious, and just a bit lower than one million were normal. The result revealed that 1168 attackers tried to send around 673 MB data to 42 victims during the test time using various techniques. The detected attack types were categorized into network layer and

Table 2: Summary of Detecting Malicious Flows.

Number of Flows	Total	1108156
	Malicious	121470
	Normal	986686
Transfered Data (Byte)	Total	40.83 GB
	Malicious	673.17 MB
	Normal	40.16 GB
Number of Transfered Packets	Total	57657933
	Malicious	1839563
	Normal	55818370
Number of Attackers		1168
Number of Victims		42

application layer classes. Network layer contained DDoS Flooding, Port and Range Scan, and Slowloris, while in the application layer, different attacks to DBMSes (like Oracle, MSSQLServer, and MySQL), mail servers (SMTP, POP3, IMAP), and other protocols (HTTP, FTP, SSH, DNS, Telnet, SIP, NetBIOS, RPC, and RDP) were detected. Figures 5 and 6 compare the number of network layer and application layer attacks, and different attack types, respectively.

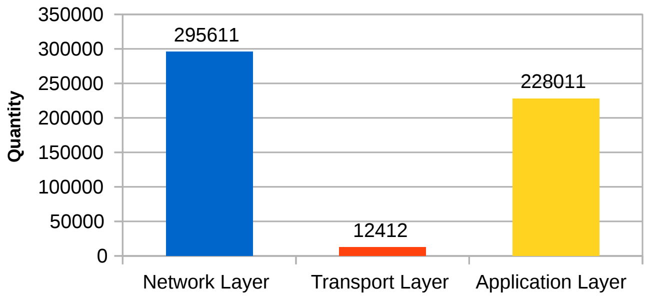
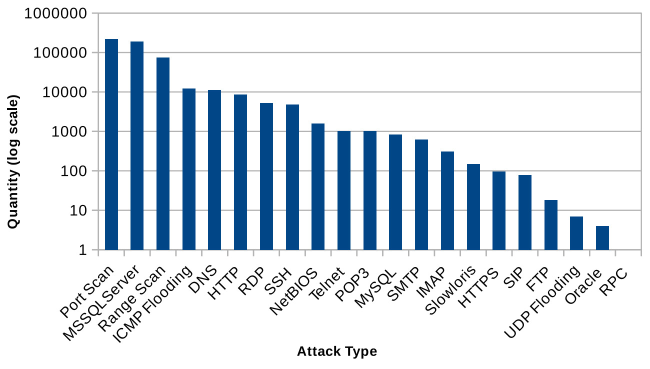


Figure 5: The Comparison of Network, Transport, and Application Layers Attacks.

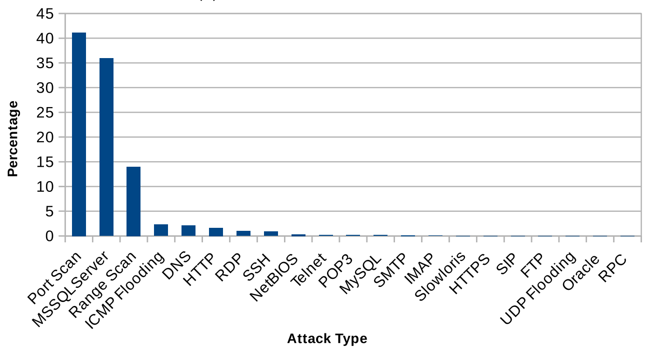
4.4 Observing the Accuracy

The CTU-13 botnet traffic dataset was used to observe the accuracy of Edmund. The dataset contains around 2.8 million labeled netflow version 5 records including normal, botnet, and background labels. The analysis result is provided in Table 3.

According to the table, around 98.55% of the total traffic were normal, while the rest of 1.45% malicious flows generated by just one attacker. Edmund could detect more than 99.72% of malicious flows. However, it marked around 5% normal traffic as malicious. Different measurements of the experiment including False Positive and Negative Rates, Accuracy, Precision, Recall, and F_Score are computed as described in Table 4.



(a) The Number of Attacks.



(b) The Percentage of Attacks.

Figure 6: The Comparison of Different Attack Types.

Table 3: Analysis Result of CTU-13 Dataset

(a) Flow

		Real Data		
		Normal	Malicious	Total
Edmund	Normal	2637565	116	2637681
	Malicious	143124	40831	183955
	Total	2780689	40947	2821636

(b) Source IP Address

		Real Data		
		Benign	Attacker	Total
Edmund	Benign	531254	0	531254
	Attacker	10667	1	10668
	Total	541921	1	541922

$$FPR = \frac{FP}{FP+TN} \qquad FNR = \frac{FN}{FN+TP}$$

$$Precision = \frac{TP}{TP+FP} \qquad Recall = \frac{TP}{TP+FN}$$

$$Accuracy = \frac{TP+TN}{All\ Data} \qquad F\_Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Table 4: **Edmund False Positive and Negative Rates, Accuracy, Precision, Recall, and F_Score.**

	FPR	FNR	Accuracy	Precision	Recall	F_score
Flow	0.003	0.051	0.949	0.999	0.948	0.974
Source IP	0	0.02	0.98	1	0.98	0.99

5 CONCLUSION

This paper proposed a novel entropy based DDoS detection and mitigation engine using netflow data, called Edmund, to detect and mitigate DDoS attacks. It was not practical to analyze the whole network traffic including packet headers and payload due to several reasons, including user privacy, encrypted data, and huge analysis cost. Hence, Edmund chose netflow to detect abnormal behavior in the network traffic.

The experimental results show that Edmund provides a high accurate framework to do online attack detection. It also detects attackers during the attack time. Therefore, the gateway can filter incoming packets sent by detected attackers early at network entry point.

Although network attacks make the traffic abnormal, some other reasons such as flash crowd can change flow statistics, as well. It is important to distinguish between attacks and flash crowd, so that only the attackers are filtered during the attack. In the future, we plan to study different scenarios that change the flow behavior in order to find a better solution to handle these benign traffics.

REFERENCES

[1] Akamai. 2017. Akamai's Quarterly Reports on State of the Internet Security. (2017). https://www.akamai.com/us/en/our-thinking/state-of-the-internet-report/global-state-of-the-internet-security-ddos-attack-reports.jsp

[2] A. Bakshi and Y. B. Dujodwala. 2010. Securing cloud from DDoS attacks using intrusion detection system in virtual machine. In *Communication Software and Networks, 2010. ICCSN'10. Second International Conference on.* IEEE, 260–264.

[3] CTU. 2011. CTU-13 Botnet Traffic dataset. (2011). https://mcfp.weebly.com/the-ctu-13-dataset-a-labeled-dataset-with-botnet-normal-and-background-traffic.html

[4] J. David and C. Thomas. 2015. DDoS attack detection using fast entropy approach on flow-based network traffic. *Procedia Computer Science* 50 (2015), 30–36.

[5] C. Douligeris and A. Mitrokotsa. 2004. DDoS attacks and defense mechanisms: classification and state-of-the-art. *Computer Networks* 44, 5 (2004), 643–666.

[6] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris. 2014. Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments. *Computer Networks* 62 (2014), 122–136.

[7] J. Ioannidis and S. M. Bellovin. 2002. Implementing Pushback: Router-Based Defense Against DDoS Attacks.. In *NDSS*.

[8] J.-H. Jun, C.-W. Ahn, and S.-H. Kim. 2014. DDoS attack detection by using packet sampling and flow features. In *proceedings of the 29th annual ACM symposium on applied computing*. ACM, 711–712.

[9] Kaspersky-Labs. 2014. GLOBAL IT SECURITY RISKS SURVEY 2014 DISTRIBUTED DENIAL OF SERVICE (DDoS) ATTACKS. (2014). http://media.kaspersky.com/en/B2B-International-2014-Survey-DDoS-Summary-Report.pdf

[10] D. Linthicum. 2013. As cloud use grows so will rate of DDoS attacks. *InfoWorld. February 5th* (2013).

[11] A. M. Lonea, D. E. Popescu, and H. Tianfield. 2013. Detecting DDoS attacks in cloud computing environment. *International Journal of Computers Communications & Control* 8, 1 (2013), 70–78.

[12] X. Ma and Y. Chen. 2014. DDoS detection method based on chaos analysis of network traffic entropy. *IEEE Communications Letters* 18, 1 (2014), 114–117.

[13] S. A. Mehdi, J. Khalid, and S. A. Khayam. 2011. Revisiting traffic anomaly detection using software defined networking. In *International workshop on recent advances in intrusion detection.* Springer, 161–180.

[14] C. E. Shannon. 1948. A mathematical theory of communication. *Bell system technical journal* 27, 3 (1948), 379–423.

[15] SPAMfighter-News. 2015. Survey - With DDoS Attacks Companies Lose around 100k/Hr. (2015). http://www.spamfighter.com/News-19554-Survey-With-DDoS-Attacks-Companies-Lose-around-100kHr.htm

[16] D. S. Terzi, R. Terzi, and S. Sagiroglu. 2017. Big data analytics for network anomaly detection from netflow data. In *Computer Science and Engineering (UBMK), 2017 International Conference on.* IEEE, 592–597.

[17] V. Vaidya. 2001. Dynamic signature inspection-based network intrusion detection. (Aug. 21 2001). US Patent 6,279,113.

[18] J. M. Vidal, A. L. S. Orozco, and L. J. G. Villalba. 2018. Adaptive artificial immune networks for mitigating DoS flooding attacks. *Swarm and Evolutionary Computation* 38 (2018), 94–108.

[19] J. Wang, X. Yang, and K. Long. 2010. A new relative entropy based app-DDoS detection method. In *Computers and Communications (ISCC), 2010 IEEE Symposium on.* IEEE, 966–968.

[20] R. Wang, Z. Jia, and L. Ju. 2015. An entropy-based distributed DDoS detection mechanism in software-defined networking. In *Trustcom/BigDataSE/ISPA, 2015 IEEE*, Vol. 1. IEEE, 310–317.

[21] Z. Xiao and Y. Xiao. 2013. Security and privacy in cloud computing. *IEEE Communications Surveys & Tutorials* 15, 2 (2013), 843–859.

[22] J. Zhang, Z. Qin, L. Ou, P. Jiang, J. Liu, and A. X. Liu. 2010. An advanced entropy-based DDOS detection scheme. In *Information Networking and Automation (ICINA), 2010 International Conference on*, Vol. 2. IEEE, V2–67.