中国矿业大学计算机学院

2017 级本科生课程报告

课程名称 可信无线自组网络技术

报告时间 2020.7.4

学生姓名 袁孝健、杨唯、杨子桢

学　　号 06172151、08172855、08173026

专　　业 信息安全

任课教师 鲍宇

任课教师评语

任课教师评语（①对课程基础理论的掌握；②对课程知识应用能力的评价；③对课程报告相关实验、作品、软件等成果的评价；④课程学习态度和上课纪律；⑤课程成果和报告工作量；⑥总体评价和成绩；⑦存在问题等:）

项目	分值	组得分	个人得分		
1. 模拟平台运行状况---安全功能可靠性模拟。所设计的**安全**功能需要全面运行，否则 0 分。	及格判定基准线20 分				
2. 报告总体格式。是否按照学院模板进行。	5 分				
3. 网络平台运行效果和难度 --个人说明所做的编码和演示功能。	10 分				
4. 报告中的安全性评估。看总体系统。	10 分				
5. PPT 答辩情况。	20 分				
6. 报告内容 50 分 a) 是否围绕主题 b) 是否能够描述并达成目标 c) 真实性、可读性 d) 是否说明不同于文献其它解决方法	 10 10 10 5				
总分 100 分					

成 绩：　　任课教师签字：

年　　月　　日

CATALOGUE

I. INTRODUCTION

The number of attacks on computer networks has been increasing for years. Network Intrusion detection system (NIDS) is a common network security system. NIDS is a device or software that monitors all traffic passing through a policy point to prevent malicious activity. When such activity is detected, an alert is generated and sent to the administrator.

In the past decade, many machine learning techniques have been proposed to improve detection performance. One common method is to use artificial neural network (ANN) for network traffic detection, because it is good at learning complex nonlinear concepts in data.

So neural networks have become an increasingly popular solution for network intrusion detection systems (NIDS). Their capability of learning complex patterns and behaviors make them a suitable solution for differentiating between normal traffic and network attacks. However, a drawback of neural networks is the amount of resources needed to train them. Many network gateways and routers devices, which could potentially host an NIDS, simply do not have the memory or processing power to train and sometimes even execute such models. More importantly, the existing neural network solutions are trained in a supervised manner. Meaning that an expert must label the network traffic and update the model manually from time to time.

In the paper, authors present Kitsune: a plug and play NIDS which can learn to detect attacks on the local network, without supervision, and in an efficient online manner.

II. PROBLEMS

A. The problems of traditions NIDS

Traditional network intrusion detection mainly adopts the detection technology based on feature detection, that is, by comparing the known attack means and the pattern characteristics of system vulnerabilities to determine whether there is an intrusion in the system. This detection method can check out the existing intrusion methods, but it is powerless against new intrusion methods or variants of known attacks. For the intricate network situation, this detection method is obviously not effective to provide guarantee for the network. The current abnormal intrusion detection technology

mostly adopts statistical method, but this kind of statistical intrusion detection also has many disadvantages:

- The sequence of events is not considered, so attacks using the sequence of events are difficult to detect.
- When the attacker is aware of being monitored, he may use the dynamic self-adaptability of the statistical contour to train the normal feature contour by slowly changing its behavior, and finally make the detection system judge its abnormal activity as normal.
- It is difficult to define thresholds for judging normal and abnormal. If the threshold is too low, the detection rate will increase; if the threshold is too high, the misjudgment rate will increase.

B. The problems of NIDS based on ANN

The prevalent approach to using an ANN as an NIDS is to train it to classify network traffic as being either normal or some class of attack. The following shows the typical approach to using an ANN-based classifier in a point deployment strategy:

1) Have an expert collect a dataset containing both normal traffic and network attacks.
2) Train the ANN to classify the difference between normal and attack traffic, using a strong CPU or GPU.
3) Transfer a copy of the trained model to the network/organization's NIDS.
4) Have the NIDS execute the trained model on the observed network traffic.

In general, a distributed deployment strategy is only practical if the number of NIDSs can economically scale according to the size of the network. One approach to achieve this goal is to embed the NIDSs directly into inexpensive routers. We argue that it is impractical to use ANN-based classifiers with this approach for several reasons:

1) Offline Processing

In order to train a supervised model, all labeled instances must be available locally. This is infeasible on a simple network gateway since a single hour of traffic may contain millions of packets. Some works propose offloading the data to a remote server for model training. However, this solution may incur significant network overhead, and does not scale.

2) Supervised Learning

The labeling process takes time and is expensive. More importantly, what is considered to be normal depends on the local traffic observed by the NIDS. Furthermore, in attacks change overtime and while new ones are constantly being discovered, so continuous maintainable of a malicious attack traffic repository may be impractical. Finally, classification is a closed-world approach to identifying concepts. In other words, a classifier is trained to identify the classes provided in the training set. However, it is unreasonable to assume that all possible classes of malicious traffic can be collected and placed in the training data.

3) High Complexity

The computational complexity of an ANN grows exponentially with number of neurons. This means that an ANN which is deployed on a simple network gateway, is restricted in terms of its architecture and number of input features which it can use. This is especially problematic on gateways which handle high velocity traffic.

III. SOLUTION

A. Overview

For these difficulties mentioned above, the author designed a network intrusion detector based on artificial neural network, called **Kitsune**, which is deployed and trained on the router in a distributed manner, but also meets the following conditions：

1) Online Processing.

First, the author replaces offline processing with online processing, which is similar to flow clustering algorithm, and each instance is immediately discarded after training or executing the model. A framework is designed to quickly extract features from dynamic data streams and damped incremental statistics is used to replace the more complex sliding window model. In the damped window model, the weight of the old value decreases with time.

2) Unsupervised Learning

In this paper, the authors propose an unsupervised anomaly detection algorithm based on an integrated automatic encoder called *KitNET*. Figure 1 is a schematic of *KitNET*, first grouping the features of the input instance into a map of the visible neurons in the integration layer, then reconstructing each feature with each automatic encoder and calculating the reconstruction errors based on RMSE. Finally, these

RMSEs serve as inputs to the output layer's automatic encoder, which USES a nonlinear voting mechanism to output abnormal scores. *KitNET* has a parameter m that specifies the maximum number of inputs per autoencoder in the integration(For balancing test speed and performance).

3) Low Complexity

The intrusion detector ensures that the maximum packet processing speed is greater than the maximum packet arrival rate, meaning that there are no packets waiting at any time. Kitsune, designed by the authors, improves packet processing speed by a factor of five, and does not have to perform as poorly as an offline detector.

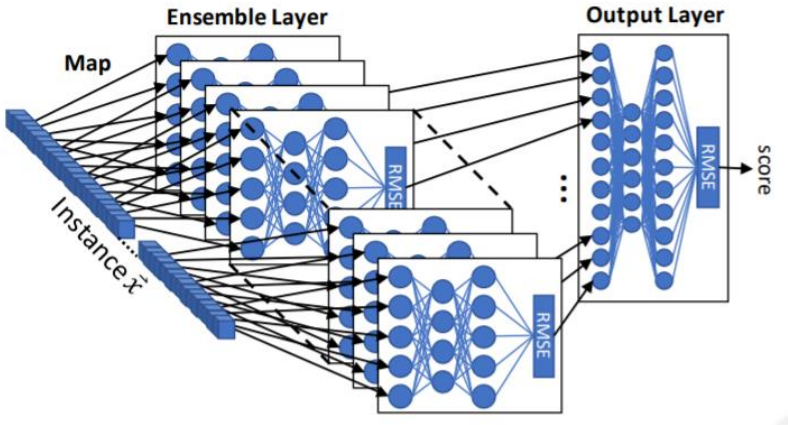

Fig. 1: An illustration of Kitsune's anomaly detection algorithm *KitNET*.

B. The Kitsune NIDS

Kitsune is a plug-and-play NIDS, based on neural networks, and designed for the efficient detection of abnormal patterns in network traffic. It operates by (1) monitoring the statistical patterns of recent network traffic, and (2) detecting anomalous patterns via an ensemble of autoencoders. Each autoencoder in the ensemble is responsible for detecting anomalies relating to a specific aspect of the network's behavior. Since Kitsune is designed to run on simple network routers, and in real-time, Kitsune has been designed with small memory footprint and a low computational complexity.

Kitsune's framework consists of the following components, and its frame diagram is shown in Figure 2:

- **Packet Capturer:** The external library responsible for acquiring the raw packet. Example libraries: NFQueue, fpacket, and tshark(Wireshark's API).
- **Packet Parser:** The external library responsible for parsing raw packets to obtain the meta information required by the Feature Extractor. Example libraries: Packet++, and tshark.

- **Feature Extractor (FE):** The component responsible for extracting n features from the arriving packets to create creating the instance $\vec{x} \in \mathbb{R}^n$. The features of $\vec{x}$ describe the a packet, and the network channel from which it came.
- **Feature Mapper (FM):** The component responsible for creating a set of smaller instances (denoted as v) from $\vec{x}$, and passing v to the in the Anomaly Detector (AD). This component is also responsible for learning the mapping, from $\vec{x}$ to v.
- **Anomaly Detector (AD):** The component responsible for detecting abnormal packets, given a packet's representation v.

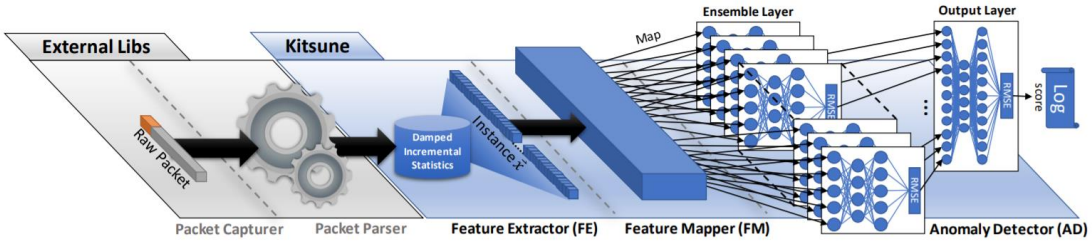

Fig. 2: An illustration of Kitsune's Architecture.

IV. RELATED WORK

The domain of using machine learning (specifically anomaly detection) for implementing NIDSs was extensively researched in the past. However, these solutions usually do not have any assumption on the resources of the machine running training or executing the model, and therefore are either too expensive to train and execute on simple gateways, or require a labeled dataset to perform the training process. Several previous works have proposed online anomaly detection mechanisms using different lightweight algorithms. For example, the PAYL IDS which models simple histograms of packet content or the KNN algorithm. These methods are either very simple and therefore produce very poor results, or require accumulating data for the training or detection.

A popular algorithm for network intrusion detection is the ANN. This is because of its ability to learn complex concepts, as well as the concepts from the domain of network communication [1]. In [2], the authors evaluated the ANN, among other classification algorithms, in the task of network intrusion detection, and proposed a solution based on an ensemble of classifiers using connection-based features. In [3],

the authors presented a modification to the back-propagation algorithm to increase the speed of an ANN's training process. In [4], the authors used multiple ANN-based classifiers, where each one was trained to detect a specific type of attack. In [5], the authors proposed a hierarchal method where each packet first passes through an anomaly detection model, then if an anomaly is raised, the packet is evaluated by a set of ANN classifiers where each classifier is trained to detect a specific attack type.

All of the above methods are some solutions for intrusion detection and have certain functions at some time. But most of them which use ANNs, are either supervised, or are not suitable for a simple network gateway. In addition, some of the works assume that the training data can be stored and accumulated which is not the case for simple network gateways. However, this paper's solution enables a plug-and-play deployment which can operate at much faster speeds than the aforementioned models.

V. SIMULATION

A. Overview

Our simulation process is mainly aimed at Kitsune, an NIDS, whose packet acquisition process is as follows, and We also recorded a video demonstrating the process and briefly explaining the code.

The process of our simulation experiment:

1) Read a PCAP packet from the local path and pass it to the Packet Parse.

2) Packet Parse receives the original binary file and parses the packet, and then sends the meta-information of the packet to the Feature Extractor (FE).

3) The FE receives this information, and uses it to retrieve over 100 statistics which are used to implicitly describe the current state of the channel from which the packet came. These statistics form the instance $\vec{x} \in \mathbb{R}^n$, which is passed to the Feature Mapper (FM).

4) After the FM receives $\vec{x}$, there are two modes:

- **Train-mode:** Uses $\vec{x}$ to learn a feature map. The map groups features of $\vec{x}$ into sets with a maximum size of m each. Nothing is passed to the AD until the map is complete. At the end of train-mode, the map is passed to the AD, and the AD uses the map to build the ensemble architecture (each set forms the inputs to an autoencoder in the ensemble).

- **Execute-mode:** The learned mapping is used to create a collection of small instances v from $\vec{x}$, which is then passed to the respective autoencoders in the ensemble layer of the AD.

5) After the AD receives v, there are also two modes:

- **Train-mode:** Receive v and uses v to train the ensemble layer. The RMSE of the forward-propagation is then used to train the output layer. The largest RMSE of the output layer is set as ϕ and stored for later use.
- **Execute-mode:** Receive v and executes v across all layers. If the RMSE of the output layer exceeds $\phi\beta$, then an alert is logged with packet details.

6) The original packet $\vec{x}$ and v are discarded.

B. Feature Extractor

The author designed a framework for high speed feature extraction of temporal statistics, over a dynamic number of data streams (network channels). The framework has a small memory footprint since it uses incremental statistics maintained over a damped window. We will not introduce the specific principle of damped increment statistics here. A total of 20 features can be extracted from a single time window λ in our simulation. The FE extracts the same set of features from a total of five time windows: 100ms, 500ms, 1.5sec, 10sec, and 1min into the past (λ =5, 3, 1, 0.1, 0.01), thus totaling 100 features.

In the code, the get_next_vector function of FE class in the FeatureExtractor.py is used to extract the Kitsune features of a package one at a time from a given PCAP file. If wireshark (tshark) is installed, it will be parsed by it, otherwise using the Scapy library will be slow. If parsed with tshark, the results are also generated into a TSV file that can be used directly next time.

```
Parsing with tshark...
tshark parsing complete. File saved as: mirai_part.pcap.tsv
counting lines in file...
There are 120001 Packets.
```

Fig.3: Operation results of program FE.

frame.time_epoch	frame.len	eth.src	eth.dst	ip.src	ip.dst	tcp.srcport	tcp.dstport	udp.srcport	udp.dstport	icmp.type	icmp.code	arp.d
1540446382.933899000	60	5c:cf:7f:05:f2:c6	4c:09:d4:c6:12:7b	192.168.2.108	52.24.43.67	21074	80					
1540446382.933904000	60	5c:cf:7f:05:f2:c6	4c:09:d4:c6:12:7b	192.168.2.108	52.25.66.250	20532	8280					
1540446382.934426000	86	4c:09:d4:c6:12:7b	5c:cf:7f:05:f2:c6	192.168.2.1	192.168.2.108	21074	80			3	0	
1540446382.934636000	86	4c:09:d4:c6:12:7b	5c:cf:7f:05:f2:c6	192.168.2.1	192.168.2.108	20532	8280			3	0	
1540446383.291054000	60	48:02:2e:01:83:15	ff:ff:ff:ff:ff:ff						1	48:02:2e:01:83:15	192.168.2.109	0
1540446383.367591000	74	40:8d:5c:4b:99:1d	3c:33:00:98:ee:fd	192.168.2.101	192.168.2.110					8	0	
1540446383.383526000	74	3c:33:00:98:ee:fd	40:8d:5c:4b:99:1d	192.168.2.110	192.168.2.101					0	0	
1540446383.391651000	117	00:16:6c:7f:82:20	4c:09:d4:c6:12:7b	192.168.2.115	192.168.2.1			2440	53			
1540446383.393709000	117	4c:09:d4:c6:12:7b	00:16:6c:7f:82:20	192.168.2.1	192.168.2.115			53	2440			
1540446383.435821000	60	5c:cf:7f:05:f2:c6	4c:09:d4:c6:12:7b	192.168.2.108	52.24.43.67	21074	80					
1540446383.435884000	60	5c:cf:7f:05:f2:c6	4c:09:d4:c6:12:7b	192.168.2.108	52.25.66.250	20532	8280					

Fig.4: Part of the TSV file generated from the extracted features.

C. Feature Mapper

The purpose of the FM is to map $\vec{x}$'s n features (dimensions) into k smaller sub-instances, one sub-instance for each autoencoder in the Ensemble Layer of the AD.

In order to ensure that the ensemble in the AD operates effectively and with a low complexity, we require that the selected mapping $f(\vec{x}) = v$:

1) Guarantee that each $\vec{v}_i$ has no more than m features, where m is a user defined parameter of the system. The parameter m affects the collective complexity of the ensemble.

2) Map each of the n features in $\vec{x}$ exactly once to the features in v. This is to ensure that the ensemble is not too wide.

3) Contain subspaces of X which capture the normal behavior well enough to detect anomalous events occurring in the respective subspaces.

4) Be discovered in a process which is online, so that no more than one at time is stored in memory.

To respect the above requirements, we find the mapping f by incrementally clustering the features (dimensions) of X into k groups which are no larger than m. We accomplish this by performing agglomerative hierarchal clustering on incrementally updated summary data.

The feature mapping algorithm of the FE performs the following steps:

1) While in train-mode, incrementally update summary statistics with features of instance $\vec{x}$.

2) When train-mode ends, perform hierarchal clustering on the statistics to form f.

3) While in execute-mode, perform $f(\vec{x}_t) = v$, and pass v to the AD.

In the code, Feature Mapper is implemented through a helper class called corClust in corClust.py for KitNET which performs a correlation-based incremental clustering of the dimensions in X. Several functions in this class and their functions are as follows:

- update：update a params in a class.
- corrDist: creates the current correlation distance matrix between the features.
- cluster: clusters the features together, having no more than maxClust features per cluster.
- __breakClust__(self, dendro, maxClust): recursive helper function which breaks down the dendrogram branches until all clusters have no more than maxClust elements.

In our simulation, 5000 traffic packets were selected to train the Feature Mapper, and the 100-dimension features were finally mapped to 19 automatic encoders in the integration layer, as shown in Figure 5:

```
Feature-Mapper: train-mode, Anomaly-Detector: off-mode
Running Kitsune:
1000
2000
3000
4000
5000
The Feature-Mapper found a mapping: 100 features to 19 autoencoders.
```

Fig.5: The train-mode of the Feature Mapper.

D. Anomaly Detector

The anomaly detector (AD) contains a special neural network called *KitNET*. It is an unsupervised artificial neural network for online anomaly detection tasks, consisting of an integration layer and an output layer of two automatic encoders.

In train-mode, *KitNET* differs slightly from the common ANN network in that it signals RMSE reconstruction errors between the two main layers of the network. In addition, *KitNET* is trained using stochastic gradient descent (SGD). In execute-mode, *KitNET* does not update any of its internal parameters. Instead, *KitNET* performs forward propagation through the entire network, and returns $L^{(2)}$'s RMSE reconstruction error. The output of *KitNET* is the RMSE anomaly score $s \in [0, \infty)$. The larger the score s, the greater the anomaly. To use s, one must determine an anomaly score cutoff threshold ϕ. The common approach is to set ϕ to the largest score seen during train-mode (Figure 6), so we took this approach in our simulation. We used 50,000 packets to train the Anomaly Detector, as shown in Figure 7.

```
Feature-Mapper: execute-mode, Anomaly-Detector: train-mode
6000
7000
8000
9000
10000
11000
12000
```

Fig.6: Start training the Anomaly Detector.

In the code of KitNET.py, first check if it has received the feature map mapped by FM. If not, it needs to train FM first, otherwise the integration and output layers of the automatic encoder will be built using the function __createAD__ (Denoising

Autoencoders code has been implemented in file dA.py. The process function is an interface function for external calls. When it is called, if AD has been trained, the execute function will be executed directly, otherwise the train function will be executed. The train function will return the anomaly score of x during training (do not use for alerting) and the execute function will return the final abnormal score. We can judge the size of the anomaly by this score.

```
51000
52000
53000
54000
55000
Feature-Mapper: execute-mode, Anomaly-Detector: execute-mode
50001
The max rmse during train-mode: 0.5205736372044542
```

Fig.7: The Anomaly Detector outputs the largest RMSE in train-mode.

E. Result

In our simulation, we use the Miria botnet's traffic packet. The original data set consists of 760,000 streams, and we select 120,000 streams from it. We use the previous 55,000 traffic to train FM and AD, and output RMSE for each traffic, as shown in Figure 8.

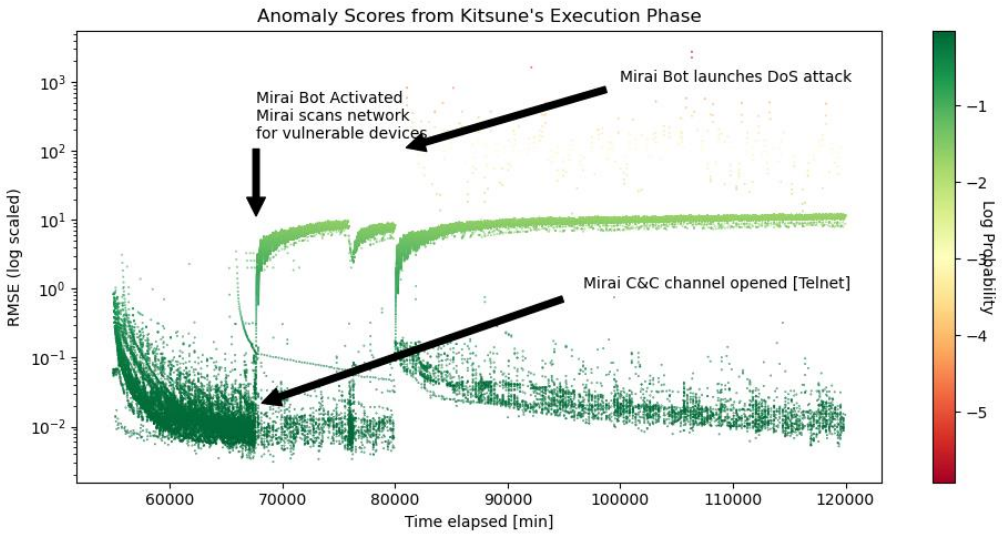

Fig.8: Abnormal score distribution of Miria botnet's traffic

The position of the first arrow is the change in traffic when Miria opens the telnet channel of the C&C server to control the device. At the same time, Mirai Bot activated Mirai scans network for vulnerable devices. From the position indicated by the upper arrow, it can be seen that the flow has changed greatly and the abnormal score is

significantly higher than the previous normal value. Starting with the third arrow, Mirai Bot launches a DoS attack. The abnormal score also becomes very high, and we use different colors to show it more intuitively and this also suggests that the anomaly is become very large.

VI. DEFECT & IMPROVEMENT

A. Evade the detection

When first installed, Kitsune assumes that all traffic is benign while in train-mode. Therefore, a preexisting adversary may be able to evade Kitsune's detection. However, during execute-mode, Kitsune will detect new attacks, and new threats as they present themselves.

In order to improve this defect, we can first execute and see if there is a high anomaly score. If there is, then we will not train from the instance (since we only want to learn from benign instances). By doing so, we can potentially remain in train-mode indefinitely. And if the target network has been contaminated, we can add a signature-based malicious traffic filtering mechanism, such as Snort, alongside the Kitsune system.

B. Attack the system

Another threat to Kitsune is a DoS attack launched against the FE. In this scenario, the attacker sends many packets with random IP addresses. By doing so, the FE will create many incremental statistics which eventually consume the device's memory. In fact, this kind of attack is obvious because it can cause big anomalies, but it can make the system very unstable.

In order to improve this defect, we can limit the number of incremental statistics which can be stored in memory. And a good solution to maintaining a small memory footprint is to periodically search and delete incremental statistics with $w_i \approx 0$. In practice, the majority of incremental statistics remain in this state because we use relatively large λs (quick decay).

VII. WORK DISTRIBUTION

Name	Id	Works
Yuan Xiaojian	06172151	Translate part of the paper PPT making and presentation Do the simulation Demonstrate the system in class Record the video in English Write the report
Yang Wei	08172855	Translate part of the paper Assist in PPT making Do the simulation Assist in report writing
Yang Zizhen	08173026	Translate part of the paper Assist in PPT making Do the simulation Assist in report writing

VIII. REFERENCES

[1] Anna L Buczak and Erhan Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. IEEE Communications Surveys & Tutorials, 18(2):1153–1176, 2016.

[2] Srinivas Mukkamala, Andrew H Sung, and Ajith Abraham. Intrusion detection using an ensemble of intelligent paradigms. Journal of network and computer applications, 28(2):167–182, 2005.

[3] Reyadh Shaker Naoum, Namh Abdula Abid, and Zainab Namh AlSultani. An enhanced resilient backpropagation artificial neural network for intrusion detection system. International Journal of Computer Science and Network Security (IJCSNS), 12(3):11, 2012.

[4] Nidhi Srivastav and Rama Krishna Challa. Novel intrusion detection system integrating layered framework with neural network. In Advance Computing Conference (IACC), 2013 IEEE 3rd International, pages 682–689. IEEE, 2013.

[5] Chunlin Zhang, Ju Jiang, and Mohamed Kamel. Intrusion detection using hierarchical neural networks. Pattern Recognition Letters, 26(6):779–791, 2005