Network Aware Defenses for Intrusion Recognition and Response (NADIR)

Nont Assawakomenkool, Yash Patel, Jonathan Voris
{nassawak@nyit.edu, ypatel20@nyit.edu, jvoris@nyit.edu}
New York Institute of Technology
New York, NY 10023

***Abstract*—It has become increasingly difficult to monitor computer networks as they have grown in scale and complexity. This lack of awareness makes responding to, or even recognizing, attacks a challenge. As a result, organizations' reactions to attacks are delayed, typically leaving them to address the situation long after an incident has taken place. The central idea behind this research is to provide earlier notification of potential network attacks by using deceptive network service information as bait. These "decoy" or "honey-services" will indicate system weak points which do not exist when suspicious network circumstances are detected. That is, although up-to-date versions of the programs will be running on the system at all times, software versions with vulnerabilities will be advertised when a potential attack or reconnaissance effort is detected. Attacks against these services will be unsuccessful because the server running our system is not actually running the vulnerable services. By providing fake vulnerable points, our system is capable of collecting information about attacks earlier in the reconnaissance phase, potentially catching adversaries in the act without exposing any actual system weaknesses. Our solution effectively transforms any legitimate server into a "honeypot" without the added overhead of setting up and maintaining a set of fake network infrastructure.**

I. Introduction

Computer networks have become increasingly important components of people's everyday lives, and this trend seems poised to continue in the future. With this increased pervasiveness of networked systems comes a commensurate increase in the number of people who attempt to break into them. Though a great deal of research effort has gone into preventing the exploitation of software vulnerabilities in order to gain illicit remote access to computer systems, relatively little attention has been paid to interfering with attacks earlier in the attack life cycle, such as during the initial information gathering stage. It has become increasingly difficult to monitor computer networks as they have grown in scale and complexity. This lack of awareness makes responding to, or even recognizing, attacks a challenge. As a result, organizations' reactions to attacks are delayed, typically leaving them to address the situation long after an incident has taken place.

This work introduces the idea of creating a tool by utilizing existing network infrastructure features in order to provide Network Aware Defenses for Intrusion Recognition and Response (NADIR). The central idea behind this research is to provide earlier notification of potential network attacks by using deceptive network service information as bait. These "decoy" or "honey-services" will indicate system weak points which do not exist when suspicious network circumstances are detected. That is, although up-to-date versions of the programs will be running on the system at all times, software versions with vulnerabilities will be advertised when a potential attack or reconnaissance effort is detected. Attacks against these services will be unsuccessful because the server running our system is not actually running the vulnerable services.

By providing fake vulnerable points, our system is capable of collecting information about attacks earlier in the reconnaissance phase, potentially catching adversaries in the act without exposing any actual system weaknesses. Our solution effectively transforms any legitimate server into a "honeypot" without the added overhead of setting up and maintaining a set of fake network infrastructure.

In order to analyze network traffic to gain an understanding of contexts in which decoy service information should be deployed, we use a decision tree machine learning algorithm. Our system uses the Weka machine learning toolkit [1] to analyze, classify, and understand the structure of network traffic in order to infer when conditions are safe to reveal legitimate service details and when such details should be masked with deceptive content.

The overall workflow of our proposed system is shown as Figure 1. Our machine learning based technique is capable of classifying network traffic according to normal and abnormal patterns, but several steps are required to process collected data prior to applying the classification algorithm. Firstly, we use an Intrusion Detection System (IDS) to collect network traffic data. In order to test our system, we constructed a dataset comprised of the legitimate traffic observed by a test server on our university network as well as malicious traffic in the form of attacks we launched at known times to provide ground truth.

Next, the collected network traffic data is converted into a Attribute-Relation File Format (ARFF) file for further processing. Subsequently, the NADIR system takes the network traffic data as input to produces IDS rules based on what traffic was found to be indicative of an attack. Although any machine learning algorithm could potentially be used, we employed a decision tree classification algorithm to produce IDS rules from the selected features automatically [2]. Finally, these rules are used to trigger our deception service description mitigation technique when unusual network circumstances are detected.

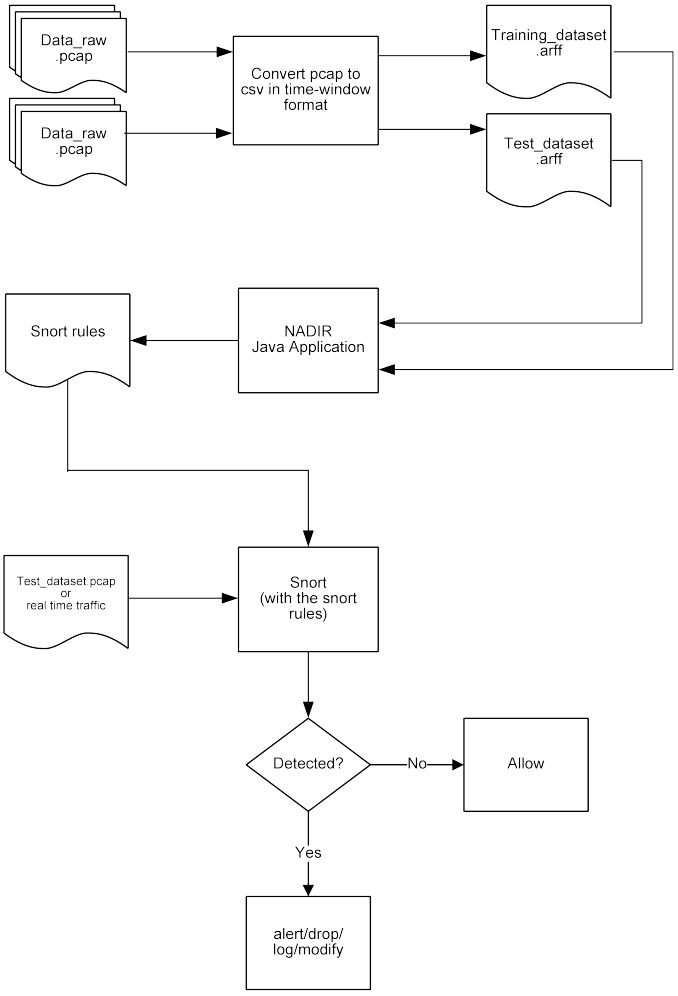


Fig. 1: Over-all processing of NADIR System

The rest of the paper is organized as follows. Section II discusses pertinent past research. Next, we present our threat model in Section III. The experiments we performed with our system are presented in Section IV and their results are provided in V. Finally, we conclude the paper in Section VI.

II. Related Work

Network intrusion detection is a well-studied field, having been the focus of research for over two decades [3], [4]. Approaches to intrusion detection can typically be classified as rule-based, which are efficient but vulnerable to previously unobserved "zero-day" attacks, and anomaly based, which are capable of detecting novel attacks, but can be slow to train and may incur high error rates. For example, a recent approach applied a neural network to classify traffic into normal and abnormal clusters [5]. Other approaches to learning for anomaly-based intrusion detection include Naive Bayes, Nearest neighbor, Decision Tree, and Support Vector Machines [6]. The most widely deployed IDS, Snort, is typically signature-based, but did support anomaly-based detection via the SPADE project [7]. Anomaly based intrusion detection approaches are orthogonal to this work, which instead focuses on leveraging network situational awareness in order to adaptively respond to attacks.

As a result of the research activity in the area of IDS, several venerable network traffic datasets have been collected, including the DARPA [8], KDD-cup'99 [9], and NSL-KDD [10] datasets. The goal of our research was not to improve upon the performance of prior approaches to anomaly detection, but rather to demonstrate how they could be extended to create on-the-fly honeypots by combining them with our decoy service mitigation strategy. As such, we utilized these datasets as a foundation by deriving features from them which could be used to detect network attacks which could be addressed with an adaptively service response.

These datasets included network, system, and user-derived features [11]; we utilized the available network level features which were available from these collections. More specifically, we chooses to focus our attention on three major attack categories: probe attacks and information collection efforts, such as port scans, Denial-of-Service attacks, where a targeted network resource is exhausted, preventing legitimate access, and remote-to-local (R2L) attacks which exploit remote system vulnerabilities in order to gain illicit system access. Although they have been used widely throughout the academic community, the aforementioned datasets are nearly two decades old, and as such are no longer considered representative of current network traffic [12]; this motivated our collection of our own network intrusion testing dataset.

Another network defense option is to confuse and obstruct potential adversaries by providing deceptive information and decoy resources known as "honeypots" [13]. Honeypots, or decoy servers, are used to collect information on adversarial activity without risking any legitimate network assets. The Beeswarm project is an example of a honeypot deployment framework [14]. A related defensive technology known as "port hoping" hides service information by rapidly altering the ports used by network services; only trusted clients aware of the anticipated pattern are able to gain access [15]. While honeypots and related techniques are powerful tools, they require substantial overhead to deploy and manage. Similarly, altering service characteristics such as port numbers constantly is a resource-intensive process. To address these shortcomings, our approach adaptively alters service characteristics only in response to detected threats.

III. Threat model

For the purposes of this project we assume attacks which are launched over a computer network from a remote host. Attacks requiring physical or local access to exploit are outside of the scope of this work. We also assume an adversary who may be generally aware that the system is equipped with security countermeasures, but is unaware of the specifics of the NADIR system. We assume a typical attack lifecycle consisting of repeated rounds of reconnaissance and exploitation, first to gain access to a network, then to penetrate more deeply into the system once initial access has been established. We consider adversaries who can run any available tools to gain

information about software and services installed on other network hosts. Once they are aware of a vulnerable service on the target server, they can use other available exploits to attack the target through that service.

IV. Experiment

A. Network Setup and Launch Attack

This section presents our experimental testing environment and the steps taken to establish it. All experiments were performed on VMware workstation 12 on live development server with the configuration Intel(R) Core (TM) i7-6500U CPU @ 2.50GHz, 30GB HDD, 2GB RAM for all virtual machines, and the operating system platform is Ubuntu 14.04.5 LTS Desktop 64-bit. We have installed and configured three Virtual Machines - "Victim," "Attacker," and "Vulnerable." All virtual machines are run on the same individual server, but they are separate in logical. The "victim's" VM configured with Snort, the attacker's machine contains Metasploit tool-kit and we used existing vulnerable Ubuntu operating system for testing purposes and make it as our third machine - vulnerable [16].

We have deployed and tested ten different Metasploit attacks: Port Scanning, DoS/ TCP SYN flood, OpenSSH Exploit via NFS Service, Remote Code Execution (Ruby DRb RMI), Remote Procedure Calls (RPC) (JAVA RMI Registry), WordPress XMLRPC DoS, VSFTPD v2.3.4 Backdoor Command Execution, Apache Exploit via PHP CGI, Anonymous login (Samba client) backdoor exploit, and Unrealircd 3.2.8.1 backdoor command execution. These attacks can be categorized into three core categories: Port Scanning (Probe) - gathering network information to bypass security, Denial of Service (Dos) - where some resource is swamped; causing DoS to legitimate users, Remote to local (R2L) - attacks that exploit remote system vulnerabilities to get access to a system. After successfully tested we made all of them as automated handled by Metasploit script, which we used for development of attacker's system [17].

B. Dataset preparation

In this part of research to collect and extract dataset we started data modeling with existing different labeled datasets to understand the model of machine learning. Then we worked on existing datasets to extract usable features from them and we came up with the java parser script which can parse and extract entire data information from Pcap file and saved into Csv file. We put time-window size variable to extract and identify unique combinations of packet transmission. Parser can parse and split the entire Pcap information with respect to the time-window size, as this feature is flexible and it depend on the nodes in our network. While researching on previous suggested approaches we tried to extract the dataset in different time-windows sizes: 10 seconds, 1 seconds, 7 seconds, and 5 seconds. We observe that 4 second time frame is more preferable in our network architecture as it generates more accurate results with weka. We collected and built our own datasets to test our model and accuracy of algorithms for anomaly detection of network attacks [6]. We launch each attack repeatedly and capture them in PCAP format. We extract them into CSV format using a 4 second time window format using ourown CSV-parser.

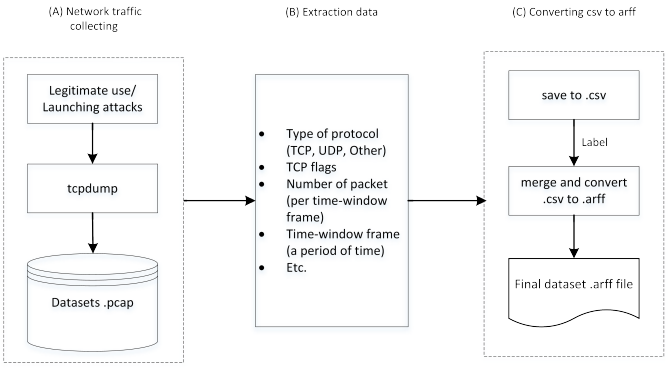


Fig. 2: Process of dataset preparation

Each row in the final CSV represents a unique direction from source_ip to destination_ip for 4sec time window. The List of features which we covered in the dataset is shown in the Table I and the dataset preparation process is shown as Figure 2.

We assume that all the packets which the direction from the Attacker to the target are abnormal then we labeled our dataset manually attacks and normal. After that, we merged and converted the dataset and converted into ARFF file format for development. We ignore and remove SOURCE_IP, DESTINATION_IP, and FIRST_TIMESTAMP because we observe it somehow classified pattern by taking those specific values. So after removing those features we are just look at network behavioral features and extracting those to classify and generate Snort rule dynamically from the matched patterns.

C. Data modeling and Machine learning

Weka library is one of the famous open source library for data mining and machine learning [18]. It provides various algorithm and methods to work on any types of dataset to extract useful features which can be applied to identify more accurately. Our research began with dataset preparation to build and train our model. We used ANOMALY_SCORE class variable to predict a specific pattern between normal and attack in the dataset. Decision tree is used where dataset has some predictor by which pattern can be classified into appropriate class. In our research we used J48 algorithm which is easy to convert into conditional algorithm for automatic Snort rule generation [19]. We used weka.jar library to implement J48 decision tree java program. Conditional program generated using weka by taking labeled training dataset as an input. It uses program template that we designed the Snort rule template with weka static method call which takes labeled/unlabeled testing dataset and generates Snort rules by looking at matched attack patterns and its classified feature values.

We used J48 decision tree machine learning algorithm to train the model as well as Snort rule generation by using

Name of Feature	Description
TYPE_PACKET	Which type of packet tcp, udp and other into string
FIRST_TIMESTAMP	Arriving timestamp when the first packet received into numeric
NO_OF_PACKETS	No of packets how much received into numeric
DEST_IP	Destination IP into string
SOURCE_IP	Source IP into string
TCP_FIN	Sum of TCP FIN flags into numeric
TCP_SYN	Sum of TCP SYN flags into numeric
TCP_RST	Sum of TCP RST flags into numeric
TCP_PSH	Sum of TCP PSH flags into numeric
TCP_ACK	Sum of TCP ACK flags into numeric
TCP_URG	Sum of TCP URG flags into numeric
TCP_ECE	Sum of TCP ECE flags into numeric
TCP_CWR	Sum of TCP CWR flags into numeric
ETHER_TYPE_AVG	Average of Ethernet type into numeric
WIRELEN_AVG	Average of Ethernet Wirelen into numeric
IP_OFFSET_AVG	Average of IP Offset value into numeric
IP_LENGTH_AVG	Average of IP Length value into numeric
IP_VER_AVG	Average of IP Version value into numeric
IP_HLEN_AVG	Average of IP hlen value into numeric
IP_TTL_AVG	Average of IP ttl value into numeric
IP_FLAG_AVG	Average of IP flag value into numeric
IP_TYPE_AVG	Average of IP type value into numeric
TCP_OFFSET_AVG	Average of TCP Offset value into numeric
TCP_LENGTH_AVG	Average of TCP Length value into numeric
TCP_HLEN_AVG	Average of TCP hlen value into numeric
TCP_RESERVE_AVG	Average of TCP Reserve value into numeric
TCP_WINDOW_AVG	Average of TCP Window size value into numeric
TCP_URGENT_AVG	Average of TCP Urgent value into numeric
PAYLOAD_OFFSET_AVG	Average of Payload Offset value into numeric
PAYLOAD_LENGTH_AVG	Average of Payload Length value into numeric

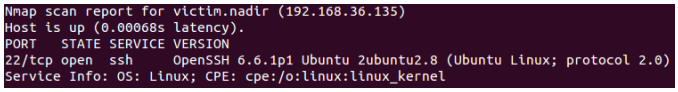
TABLE I: Table of feature extraction

that model [20]. J48 decision tree classifier classify pattern by number of feature from our dataset which we than converted into Snort rule using Snort rule template. For example if model classified a specific attack pattern by tcp_syn, no_packets and type_packet features than it will put those values of that specific features into the Snort rule template which we used in our NADIR application. Thus, we get the Snort rules as an output of the application.

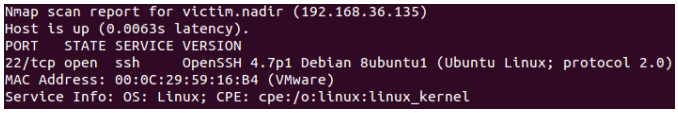
D. Snort configuration

Intrusion Detection System (IDS) is a device which monitors packets on the computer networks. IDS reports attack behaviors based on rules and signatures applied to the machine. Intrusion Prevention System (IPS) could achieve Real-time intercepting by leveraging in-line deployment in the network. It analyzes all network traffic passing through interface and takes actions to suspicious packets immediately [21].

We converted Snort IDS into INLINE QUEUE mode using Iptables rules and Snort configurations. Now, it receives packets sent from the Net filter Queue with the help of the nfnetlink_queue library [21]. Snort compares the packets with Snort signature rules, takes action such as alert or drop if the packets are matched to a rule. Then, it finally sends them back



(a) The real running service information on the server



(b) The attacker gets the fake information

Fig. 3: The figure show the service information that run on the server (a)Actual service version and (b)Fake service version

to Net filter Queue where the Snort Inline tagged packets are processed. This effectively converts NADIR from passive IDS into an active IPS.

In this case, the attacker will get the same information or fingerprints as the responses from the Vulnerable when the Attacker scans vulnerability services on the victim which are fake responses (replaced payload packets). For example, the attacker gets a fingerprint of version of SSH service from the Vulnerable as SSH-2.0-OpenSSH-4.7p1 and gets the same information, SSH-2.0-OpenSSH-4.7p1, from Victim as well even though the corrected information is SSH-2.0-OpenSSH-6.6.1p1 as shown in Figure 3.

In order to forge responses to service fingerprinting attempts, the first step is to queue all the packet not only from the attacker but also all the ingress/egress traffic by iptables. Then, all the traffic will pass through the Snort which run as inline mode. The input traffic packets will not be replaced the payload yet, Snort just only monitors the traffic and tries to match with the Snort rule. Once the input packets are match with the rule, Snort will replace the content of the payload of the original response packet to be pre-payload which is same as vulnerable response. Lastly, the replaced packets that contain a fake response will be sent back to the attacker. Thus, the attacker will learn and get an incorrect information.

E. Overall system setup

The over-all process of NADIR system is show as Figure 1. As development phase of this research, we rebuild the whole system by reconfiguring Snort and adding Snort rules to replace payloads by checking the TCP flags of incoming requests [22]. We divide Snort rules into four different rule files: (1) my-flowbits.rules, (2) my-print.rules, (3) my-snort.rules, and (4) my-drop.rules

The (1) my-flowbits.rules and (2) my-print.rules which we developed at beginning are used to control the process of replacement of any string or payload. The other rule files will be maintained by the NADIR application. At the end, we come up with the overall system setup with our NADIR application which takes training labeled dataset and unlabeled or labeled testing dataset to generate the decision tree by using J48 algorithm then generating Snort rules dynamically based on the decision tree. The NADIR application generates Snort rules

and updates my-snort.rules file by inputting the training/testing dataset. The NADIR application tracks the Snort logs and it generates filtering rules and update my-drop.rules file if an attack happened and detected. After successfully updating the Snort rules, we managed daemon to restart the Snort service without missing any packets by using two different daemon services and two Snort instances To handle two Snort instances we used two bridge interfaces. If the first Snort instance is running and the Snort rules need to be updated, the NADIR application runs another Snort instance with updated rule sets and halts the first Snort instance. NADIR also flushes the entire drop rule file every night and keep the all blacklist IPs into backup file automatically so that we can have the benefit of temporary blocking.

V. Results

This section is divided into two parts. In the first, we evaluate the machine learning algorithm and its results for anomaly detection by Intrusion detection system. In the second, we focus on the overall NADIR system by using Snort to run an experiment in which network attacks are detected and response to with forged honey-service information. NADIR uses the J48 decision tree based learning algorithm to create Snort rules on the fly in response to previously observed network features. An example of a subsection of a classification tree is provided in Figure 4. Table II and Figure 5 show the accuracy results of the J48 algorithm and ROC curve for each type of traffic.

```
TCP_SYN > 28
|   IP_FLAG_AVG <= 1
|   |   IP_TTL_AVG <= 63: NMAP (1046.0)
|   |   IP_TTL_AVG > 63
|   |   |   TCP_LENGTH_AVG <= 21: DOS (11.0)
|   |   |   TCP_LENGTH_AVG > 21: NMAP (15.0)
|   IP_FLAG_AVG > 1
|   |   PAYLOAD_OFFSET_AVG <= 31
|   |   |   TCP_RST <= 22
|   |   |   |   TCP_HLEN_AVG <= 5: NMAP (6.0)
|   |   |   |   TCP_HLEN_AVG > 5: NORMAL (8.0)
|   |   |   TCP_RST > 22: NORMAL (17.0)
|   |   PAYLOAD_OFFSET_AVG > 31: DOS (4.0)
```

Fig. 4: Sub part of pruned tree J48 Decision (based on our dataset)

We evaluated the end-to-end NADIR system which collects network features, uses them to generate Snort rules, and then pushes them into the configuration of an existing Snort service, which is then restarted. This results in an overall system which uses the J48 machine learning algorithm to detect network anomalies. In response, it generates Snort rules dynamically and replaces them without incurring any lost packets. If NADIR detects any attempt to attack the machine, it can respond by taking action such as dropping incoming packets coming from the suspected attacker's IP address.

We tested the complete NADIR system by launching attacks against a computer with it installed at known times to provide ground truth. We then calculated area under the ROC curve (AUC) values to evaluate the tradeoff between true positive (TP) and false positive (FP) rates of our system. The AUC for detecting these attacks was 0.987 for Normal traffic. NADIR classified Probe, R2L, DoS attacks with AUC values of 0.991, 0.950, and 0.980 respectively. Although some errors of both types were made by NADIR's classifier, rates were low; for example, the prediction results of DoS attacks show a 0% false positive rate.

VI. Conclusion

This paper presented NADIR, a system for defending against network reconnaissance efforts. NADIR works by providing deceptive service information when suspicious network contexts are detected. As a part of this effort we created a tool to generate Snort rules dynamically by observing network traffic in real time. NADIR is capable of hiding information pertaining to real running service and sending back decoy information to potential adversaries when suspicious network circumstances are detected. As part of this effort, we collected a network intrusion dataset by observing legitimate traffic on our university computer network and injecting real attack data. NADIR is capable of extracting features from observed network packets. As a proof of concept, we implemented NADIR using the J48 decision tree algorithm to create an anomaly based IDS, though NADIR is flexible enough to support any approach to intrusion detection. Our experimental results demonstrate that NADIR is capable of providing decoy service information as a mitigation strategy to defend against real attacks.

References

[1] Weka. Available at http://www.cs.waikato.ac.nz/ml/weka/.

[2] N. Khamphakdee, N. Benjamas, and S. Saiyod, "Network traffic data to arff converter for association rules technique of data mining," pp. 89–93, Oct 2014.

[3] M. Roesch *et al.*, "Snort: Lightweight intrusion detection for networks." in *Lisa*, vol. 99, no. 1, 1999, pp. 229–238.

[4] K. Wang and S. J. Stolfo, "Anomalous payload-based network intrusion detection," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2004, pp. 203–222.

[5] B. Subba, S. Biswas, and S. Karmakar, "A neural network based system for intrusion detection and attack classification," in *2016 Twenty Second National Conference on Communication (NCC)*, March 2016, pp. 1–6.

[6] Z. Dewa and L. A. Maglaras, "Data mining and intrusion detection systems," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 1, pp. 62–71, 2016.

[7] S. Biles, "Detecting the unknown with snort and the statistical packet anomaly detection engine (spade)," Computer Security Online Ltd.

[8] MIT Lincoln Labs Information Systems Technology Group, "DARPA Intrusion Detection Data Sets," https://www.ll.mit.edu/ideval/data/, 1998.

[9] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*. IEEE, 2009, pp. 1–6.

[10] S. Lakhina, S. Joseph, and B. Verma, "Feature reduction using principal component analysis for effective anomaly–based intrusion detection on nsl-kdd," 2010.

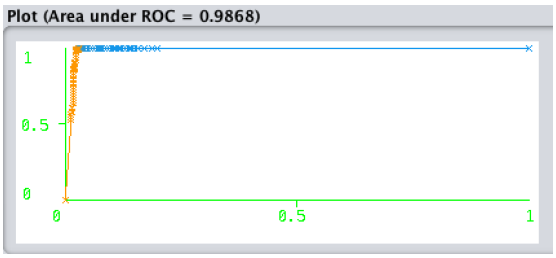
[11] F. Iglesias and T. Zseby, "Analysis of network traffic features for anomaly detection," *Machine Learning*, vol. 101, no. 1, pp. 59–84, 2015. [Online]. Available: http://dx.doi.org/10.1007/s10994-014-5473-9

[12] Terry Brugger, "KDD Cup '99 dataset (Network Intrusion) Considered Harmful," http://www.kdnuggets.com/news/2007/n18/4i.html, 2007.

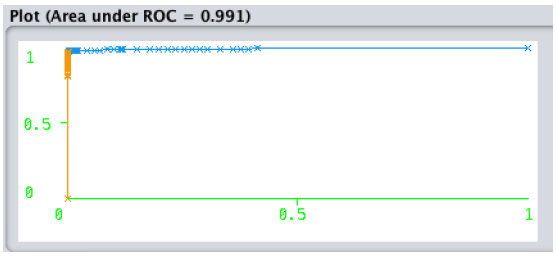
=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.996	0.033	0.994	0.996	0.995	0.967	0.987	0.995	NORMAL
	0.976	0.003	0.981	0.976	0.979	0.975	0.991	0.977	Probe
	0.820	0.001	0.890	0.820	0.854	0.853	0.957	0.777	R2L
	0.934	0.000	0.959	0.934	0.947	0.946	0.980	0.950	DoS
Weighted Avg.	0.991	0.029	0.991	0.991	0.991	0.967	0.987	0.990	

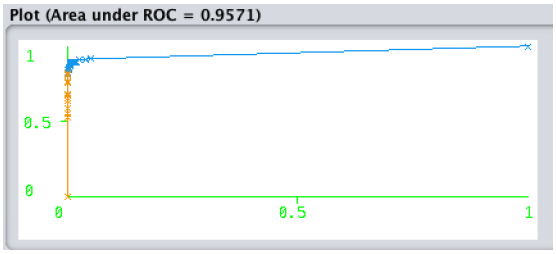
TABLE II: Accuracy Result of J48 algorithm (based on our dataset)



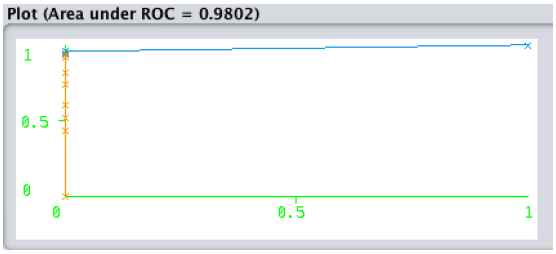
(a) ROC-NORMAL



(b) ROC-Probe



(c) ROC-R2L



(d) ROC-DoS

Fig. 5: The ROC Curves for each type of traffic

[13] L. Spitzner, "Honeypots: Catching the insider threat," in *Computer Security Applications Conference, 2003. Proceedings. 19th Annual*. IEEE, 2003, pp. 170–179.

[14] "Beeswarm: Active ids made easy," http://www.beeswarm-ids.org/, 2014.

[15] Y.-B. Luo, B.-S. Wang, and G.-L. Cai, "Analysis of port hopping for proactive cyber defense1," *SERSC International Journal of Security and Its Applications*, vol. 9, no. 2, pp. 123–134, 2015.

[16] Metasploitable. Available at https://information.rapid7.com/metasploitable-download.html.

[17] "Metasploit scripts," https://www.offensive-security.com/metasploit-unleashed/existing-scripts/.

[18] S. S. Aksenova, "Machine learning with weka weka explorer tutorial for weka version 3.4.3," 2004.

[19] "Use weka in your java code," https://weka.wikispaces.com/Use+Weka+in+your+Java+code.

[20] K. Swamy and K. V. Lakshmi, "Network intrusion detection using improved decision tree algorithm," *International Journal of Computer Science and Information Technologies*, vol. 3, no. 5, 2012.

[21] "Netfilter & snort_inline," https://openmaniak.com/inline_final.

[22] "Snort tracking exploit progress with flowbits," http://resources.infosecinstitute.com/snort-tracking-exploit-progress-with-flowbits/#gref.