Network Aware Defenses for Intrusion Recognition and Response (N.A.D.I.R.)

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 [3]. Finally, these rules are used to trigger our deception service description mitigation technique when unusual network circumstances are detected.

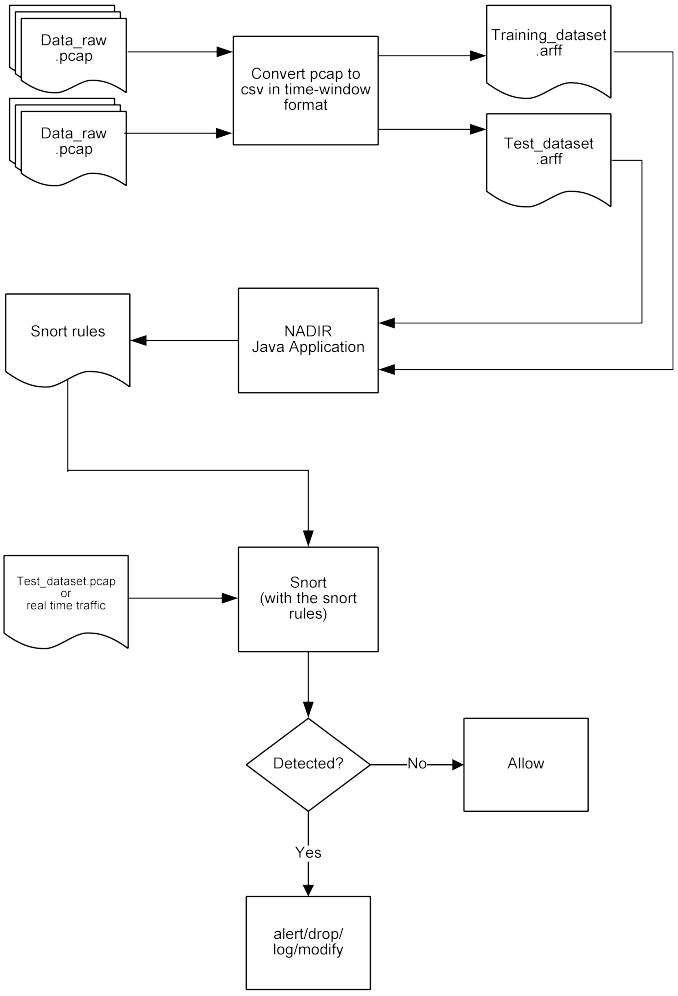


Fig. 1: Over-all processing of NADIR System

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

In this section, we covered our background research related to our development. We began our research by observing and listing useful features for anomaly based intrusion detection used in previous researches and as suggested we collected and listed features by looking at different dataset for intrusion detection. We have started our research by work on DARPA, NSL-KDD, KDD-cup'99 dataset and understand dataset methodology. They have included Network based, System based as well as User based features into the NSL-KDD, KDD-cup'99 dataset [4]. DARPA-pi introduces the dataset for network simulation and we found they collected unlabeled dataset which can be divided into clusters by normal or abnormal behavior of network [5], [6]. Such as, we have focused on network threats only that includes ten different attacks and divided into three major types in our research. In which we covered 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 [7].

As KDD- cup'99 is really basic idea and it also has been long history in performance on anomaly based intrusion detection, but we realized it outdated and not much fit with our model, so finally We collected and built our own dataset for different network attacks with legitimate traffic. Detecting threats against network by searching for anomalous traffic has been an active research area. We found various existing research idea and their suggested approaches include packet inspection, data mining techniques for feature selection, machine learning techniques such as Naive Bayes, Nearest neighbor, Decision tree and Support vector machines to detect intrusion over network [8]. We found snort had also suggested research on anomaly based detection by severity and anomaly score based preprocessor- SPADE which suspended for such reasons [9]. Note that 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. By applying machine learning algorithm to classify threat from unknown data and pick those classifiers and used those features to get IDS rule sets to detect attacks in real-time. We have used decision tree classifier to get snort rule from selected features automatically from past research idea on snort rule generation mechanism by snort rule template using Association Rules Technique [3], [10].

Threat detection and vulnerability patches are important though the easiest way to defend machine is to hide information or make the machine invisible into the network. Port hoping is the defense technology which hides service information of the system and confused attackers when they try to reconnaissance by altering services ports [11]. Honeypot is basically used to collect network information and analysis the network behavior to improve its performance. The machine placed where the inbound network traffic comes in through router, so the honeypot collects and keeps track network traffic and its statistical behavior. It also has lots of research with threat detection used by honeypot service [12]. Honeypot the open source project for Ubuntu was the basic idea to reply any request with fake packet information [13].

Snort IDS is one of active research area from last some decades and snort has some features which can be changed and that useful to change IDS behavior and make it honeypot temporarily using snort rules [14]. Adaptive network response is most important part into the research our core goal for our research effort seeks to address this oversight by developing the novel technique for detecting and responding to adversarial efforts to collect information about system they have targeted for attack [15]. We have not just focused on stop an attack but also catch attacker with sufficient evidence and take an action on machine to overcome performed attack. In contrast, the main idea behind our approach is to create a feedback mechanism whereby hosts, routers, and other devices can

utilize current state information of network and alter network-facing properties to the attacker. By keep tracking network behavior and look at those anomalies and if the attack happens then dynamically change firewall rules and also change its state and also capture attack events is our core goal end of this research, By checking logs and detect and mitigate threat would be an advanced step which we could find in our application [16]. This how collected all important information and made background research more accurate and bidirectional in a way that we could work on the actual experiment and its results and accuracy which we discussed in next part of this research paper.

III. Threat Model

In our research, we assume that all conditions of attack are based on remote attacks. It means no adversaries can get into a target server physically, but they can exploit the target server remotely. The attackers know nothing about the N.A.D.I.R. system so that they attack the target server with no suspect. The attackers use some tools for scanning the target server's running services to learn the information of those services and look for a vulnerable service(s). Once they know the vulnerable service(s) on the target server, they can use other tools to attack the target through that service(s) to down the service or to gain an access so they can do more business. Which this threat model, the N.A.D.I.R. system takes an action at the first place where the attackers scan the running services on the server. The N.A.D.I.R. system hides the true information of running services then sends the decoy services' information back to the attackers. Thus, we can detect and catch the attackers who attack the decoy services.

IV. Experiment

A. Network Setup and Launch Attack

In this part of research, we cover introduction of our testing environment and also experimental setup of those machines. 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 traffic flow between each machines are shown in Figure 2. In which victim's VM configured with snort, 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 [17]. We have deployed and tested ten different Metasploit attacks and we divided it into three types of attack including 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 [18].

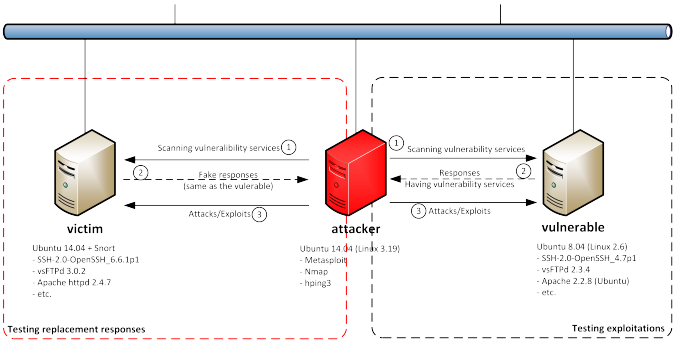


Fig. 2: Logical connection model

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 it 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. While researching on previous suggested approaches we tried to extract dataset by different time-windows size by changing its value by 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 generate more accurate result with weka. As this feature is flexible and it is depends on the nodes in our network. We collected and built our own datasets to test our model and accuracy of algorithms for anomaly detection of network attacks [8]. We launch each attack repeatedly and capture those Pcap and after merging those Pcap we extract that into Csv format using our own Csv-parser by 4 second time window size.

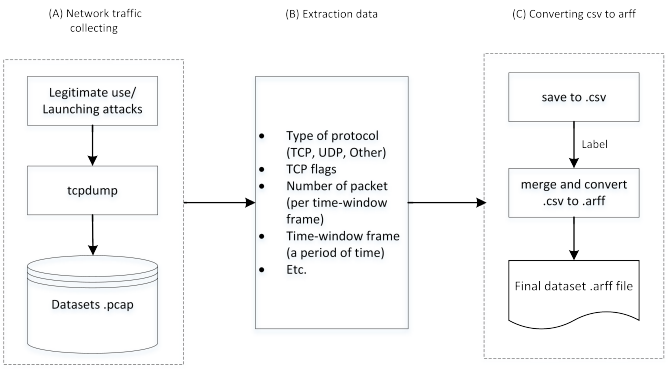


Fig. 3: Process of dataset preparation

Each row into the final Csv represents a unique direction from source_ip to destination_ip for 4sec time window. List of features which we covered into dataset is shown in the Table I and the dataset preparation process is shown as Figure 3.

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
ANOMALY_SCORE	Class feature to label pattern (NORMAL/NMAP/DOS/R2L)

TABLE I: Table of feature extraction

We added time window size parameter flexible that in future one can change this parameter for their use by observing network traffic. We launched each attacks from attacker's machine and we assume that all the packets which are coming from that attacker's machine as an attack and then we labeled our dataset manually attacks and normal 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 removed those features we are just looking at network behavioral features and extracting those to classify and generate snort rule dynamically for matched patterns.

C. Data modeling and Machine learning

Weka library is one of the famous open source library for data mining and machine learning [20]. It provides various algorithm and methods to work on any types of dataset to extract useful features which can applies to identify more accurately and our research Began with dataset preparation to build and train our model. We used ANOMALY_SCORE class variable to predict a specific pattern for normal or attack into 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 [10]. We used weka.jar library to implement J48 decision tree java program. Conditional program generated through weka by taking labeled training dataset as an input. It uses program template in which we designed 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 that model [21]. 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 snort rule template which we used into our Java application and as an output of that program we get snort rules.

D. Snort configuration

Intrusion Detection System (IDS) is a device which monitors packets on your network. IDS reports attack behaviors based on rules and signatures applied to the machine. 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 [22].

We converted Snort IDS into INLINE QUEUE mode by using Iptables rule and Snort configurations. Now, it receives packets sent from the Net filter Queue with the help of the nfnetlink_queue library, compares them with Snort signature rules and alert or drop if they match a rule, then finally sends them back to Net filter Queue where the Snort Inline tagged packets are dropped [22]. An IPS (Intrusion Prevention System) where a packet matching a signature rule is blocked or modified.

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.

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, Snort just only monitor the traffic and try to match with the snort rule. Once the input packets are match with

the rule, snort will replace the payload of the original output 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 N.A.D.I.R. system is show as Figure 1. As development phase of this research, we rebuild whole system by reconfiguring snort and adding snort rules to replace payloads by checking TCP flags of incoming requests [23]. 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 maintain by our Java application. At the end we come up with the overall system setup with our java application which takes training label set and once it will regenerate J48 decision tree algorithm by taking unlabeled testing sets or label instead and it can generate snort rules dynamically. Java application generates snort rules and updates my-snort.rules file as per the training/testing dataset. Java application keep tracking on snort logs and it generates drop rule and update my-drop.rules file if an attack happened and detected. After successfully update snort rules, we managed deamon to restart snort service without missing any packet. We used two different deamon services and created two snort instances and handled two snort instances with two bridge interfaces. Here, if first snort instance is running and once snort rule updated, Java application runs another snort instance with updated rule sets and kill the first snort instance. The system flush entire drop rule file every night and keep the all blacklist IPs into backup file automatically. We tested our model step by step as well in real-time network traffic.

V. Results

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.

This section we divided into two part of evaluation: in first part we evaluate machine learning algorithm and its results for anomaly detection by Intrusion detection system and in the next part we focused on actual production and its results with snort experiment and attack detection and its respond. As summarized the java application converts decision tree into java program by taking J48 tree and the classified tree looks like Figure 4. Table II and Figure 5 show the accurecy results of J48 algorithm and ROC curve for each type of traffic respectively.

In second part, by evaluating snort system which generates snort rule and push them into existing snort configuration and restart snort service. Finally we got a whole working machine

```
J48 pruned tree
------------------

TCP_SYN <= 28
|   WIRELEN_AVG <= 1392
|   |   TCP_PSH <= 0
|   |   |   TCP_SYN <= 0
|   |   |   |   TCP_FIN <= 0
|   |   |   |   |   TCP_RST <= 0
|   |   |   |   |   |   IP_FLAG_AVG <= 1
|   |   |   |   |   |   |   WIRELEN_AVG <= 61
|   |   |   |   |   |   |   |   PAYLOAD_OFFSET_AVG <= 21: NORMAL (544.0)
|   |   |   |   |   |   |   |   PAYLOAD_OFFSET_AVG > 21: Probe (20.0)
|   |   |   |   |   |   |   WIRELEN_AVG > 61
|   |   |   |   |   |   |   |   IP_TTL_AVG <= 51: NORMAL (2017.0)
|   |   |   |   |   |   |   |   IP_TTL_AVG > 51
|   |   |   |   |   |   |   |   |   IP_TTL_AVG <= 119
|   |   |   |   |   |   |   |   |   |   WIRELEN_AVG <= 144: NORMAL (274.0)
|   |   |   |   |   |   |   |   |   |   WIRELEN_AVG > 144
|   |   |   |   |   |   |   |   |   |   |   IP_TTL_AVG <= 81
|   |   |   |   |   |   |   |   |   |   |   |   IP_TTL_AVG <= 62: Probe (5.0)
|   |   |   |   |   |   |   |   |   |   |   |   IP_TTL_AVG > 62: NORMAL (37.0)
|   |   |   |   |   |   |   |   |   |   |   IP_TTL_AVG > 81: Probe (6.0)
|   |   |   |   |   |   |   |   |   IP_TTL_AVG > 119: NORMAL (1891.0/4.0)
|   |   |   |   |   |   IP_FLAG_AVG > 1
|   |   |   |   |   |   |   TCP_HLEN_AVG <= 6: NORMAL (939.0)
|   |   |   |   |   |   |   TCP_HLEN_AVG > 6
|   |   |   |   |   |   |   |   WIRELEN_AVG <= 97
|   |   |   |   |   |   |   |   |   WIRELEN_AVG <= 94
|   |   |   |   |   |   |   |   |   |   TCP_WINDOW_AVG <= 236: NORMAL (105.0)
|   |   |   |   |   |   |   |   |   TCP_WINDOW_AVG > 236
|   |   |   |   |   |   |   |   |   |   IP_TTL_AVG <= 89: R2L (2.0)
|   |   |   |   |   |   |   |   |   |   IP_TTL_AVG > 89: NORMAL (8.0)
|   |   |   |   |   |   |   |   WIRELEN_AVG > 94: R2L (4.0)
|   |   |   |   |   |   |   WIRELEN_AVG > 97: NORMAL (516.0)
```

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

which works on J48 machine learning algorithm for detecting intrusion and generating snort rule in real-time and restart snort service with any packet loss. If it detects any attempt to exploit machine can respond by itself by dropping all incoming packets coming from attacker's IP. We uses one common rule for DOS and DDOS detection so machine can detect both of them but can't mitigate DDOS attack though it can detect successfully.

In accuracy results, ROC- space shows area under the curve (AUC) is 98% of accurate for Normal traffic. It classified Probe, R2L, Dos attacks with 99%, 95% and 98% accurate results respectively. Here, ROC Curves can be used to evaluate the tradeoff between TP (true-positive) and FP (false-positive) rates of classification algorithm. We got some false prediction in which actual attack predicted as normal and same as some normal pattern classified as an attack and got some false alarm by the machine learning classifier. Dos results shows that there is 0.0 FP rate with the classification algorithms.

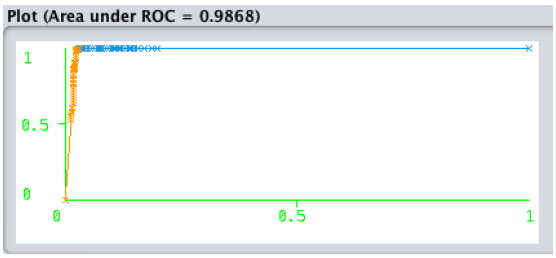
VI. Conclusion

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 the tool to generate the dynamic Snort rules automatically by feeding dataset or network traffic. We first work on hiding the real running service information on the Victim and sending back the bogus information back to the Attacker. Then, we collected the dataset in two main different types of traffic which are legitimate and malicious packets. Next, we developed the application to extract features from captured packets and convert to the particular format. After we have the final datasets, we use them to feed our NADIR application to

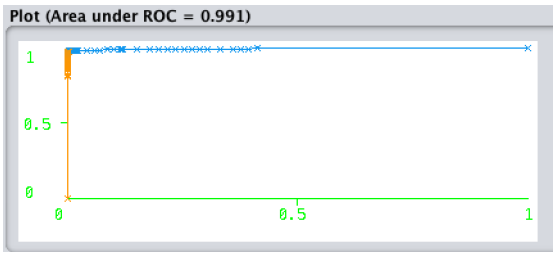
=== Detailed Accuracy By Class ===

	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	

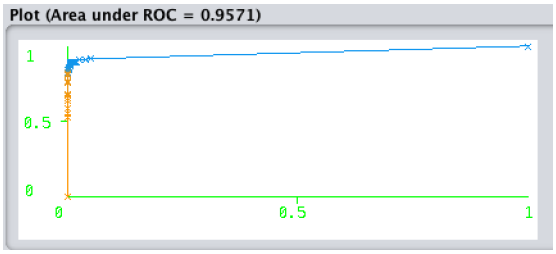
TABLE II: Accuracy Result of J48 algorithm (based on our dataset)



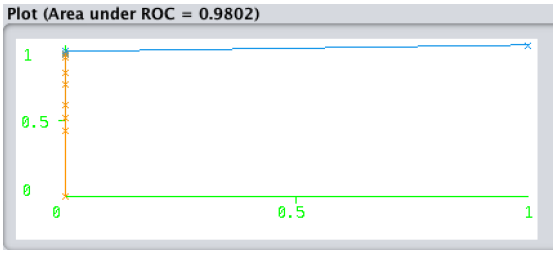
(a) ROC-NORMAL



(b) ROC-Probe



(c) ROC-R2L



(d) ROC-DoS

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

generate the dynamic Snort rules. NADIR learns and classifies the relational dataset by J48 decision tree algorithm then generates the Snort rules to detect the malicious pattern. Once Snort-IDS gets the updated rules, application keeps track snort logs and takes one more forward step to create the drop rules in my-drop.rules to drop network traffic from the specific source IP address which violates the alert rules as it detects threat real-time.

REFERENCES

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

[2] Snort. Available at https://www.snort.org.

[3] 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.

[4] 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

[5] DARPA Intrusion Detection Data Sets Available at https://www.ll.mit.edu/ideval/data/.

[6] 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.

[7] S. Paliwal and R. Gupta, "Denial-of-service, probing & remote to user (r2l) attack detection using genetic algorithm," *International Journal of Computer Applications*, vol. 60, no. 19, pp. 57–62, 2012.

[8] 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.

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

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

[11] L. Spitzner, "Honeypots: Catching the insider threat," Honeypot Technologies Inc.

[12] 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.

[13] "Beeswarm honeypot on ubuntu 14.04 lts," http://www.rationallyparanoid.com/articles/beeswarm-honeypot.html, 2014.

[14] V. Kumar and D. O. P. Sangwan, "Signature based intrusion detection system using snort," *International Journal of Computer Applications & Information Technology*, vol. 1, pp. 35–41, 2012.

[15] M. Couture, B. Ktari, M. Mejri, and F. Massicotte, "A declarative approach to stateful intrusion detection and network monitoring."

[16] J. Breier and J. B. sova, "A dynamic rule creation based anomaly detection method for identifying security breaches in log records," *Wireless Personal Communications*, 2015.

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

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

[19] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *Comput. Secur.*, vol. 28, no. 1-2, pp. 18–28, Feb. 2009. [Online]. Available: http://dx.doi.org/10.1016/j.cose.2008.08.003

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

[21] 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.
[22] "Netfilter & snort_inline," https://openmaniak.com/inline_final.
[23] "Snort tracking exploit progress with flowbits," http://resources.infosecinstitute.com/snort-tracking-exploit-progress-with-flowbits/#gref.