Employing Machine Learning and Signature Matching Techniques for Anomaly Detection on the Cloud

Srishti Agarwal
Department of Computer Science
Texas A&M University
agar1605@tamu.edu

Suma Garuda
Department of Computer Engineering
Texas A&M University
suma_garuda@tamu.edu

***Abstract*—Anomaly Detection using machine learning techniques is one of the dynamic research areas in the field of computer security. Traditionally, researchers have used patterns, or signatures stored on the intrusion detection system to detect such anomalous or malicious behavior, but with the popularity of statistical algorithms and newer exploits being found every day, more and more research is getting done on the application of such techniques in designing a host intrusion detection system which is fault proof. We try to build such a system using a mix of traditional and machine learning techniques to detect such kind of patterns on cloud instances. We use Snort, for detecting network attacks through content matching in packets, followed by OSquery, a popular open source framework, for obtaining system activities of a user and detecting anomalous patterns in them.**

***Index Terms*—Anomalies, Intrusion Detection System, Signature Based Detection, Machine Learning, host, network**

I. Introduction

With an increase in the number of cyber attacks and hacking activities, it is clear that traditional signature based intrusion detection systems are very brittle and can be easily evaded. An Intrusion Detection System (or IDS) attempts to identify any intrusions in the system, generate alarms, trigger for any protection defenses, assesses any damage done and provides clues on the attacker which can help an administrator to protect from further similar compromise. An IDS can be classified into two categories: namely Network IDS and Host IDS. A NIDS scans the network traffic and matches every single packet with the library of known patterns of attacks.However, an HIDS (Host intrusion detection system) inspects computing activities happening within a host at regular intervals such as file access and execution of files and alerts when changes are not expected to certain configurations. As stated above, traditional IDS or Misuse IDS are successful against attacks whose characteristics are known a-priori, however with a fast changing attack landscape anomaly IDS appear to be more promising as they can defend against attacks whose characteristics are not known beforehand (zero day exploits). We try to build such an IDS where Snort plays the role of a NIDS and OSquery, an open source framework developed at Facebook, our HIDS. In further sections we will explore these two parts in more detail. The content in this paper is distributed as follows:
In II we describe about the different anomalies in detail and how HIDS detects such anomalies in general. In III we explain different machine learning anomaly detection algorithms, and our approach in detail. In IV we explore OSquery, its different flavors and what features we have considered in detecting anomalous behavior. In V we discuss about our dataset, evaluation technique and results. In VI, we describe our signature model for NIDS and how we can protect our endpoint system from basic network attacks through this first line of defense (Since HIDS, can't protect against the same). We conclude this paper and summarize our thoughts about this approach in VIII and VII. All the code related to this project can be found at [14]

II. Anomalies

Anomalies are referred to as outliers, or patterns that are classified as deviations from normal data. There are three broad categories of anomalies defined:-

Point Based Anomaly: If an individual data point in a stream of data is anomalous, then it is a point based anomaly. This type of behavior could be understood using Bank Based Fraud Detection systems having a single feature like amount spent. If at any instance a transaction above a threshold amount happens, it is flagged and customer is notified.

Contextual Anomaly: If an individual data is anomalous in a specific context then it is a contextual anomaly. This could be understood using a credit card fraud detection system which remembers all average transactions made per month made through an account. In festive seasons this amount might vary significantly from the average amount which is normal, however if same drift in other months is seen, it is an anomaly. This depends on the context.

Collective Anomaly: If a collection of data is anomalous with respect to the entire data we term it as a collective anomaly. We can understand this by using an example of a system

which logs all the sequence of actions like: ...http-web, buffer-overflow, http-web, http-web, smtp-mail, ftp, http-web, ssh, smtp-mail, http-web, **ssh, buffer-overflow, ftp,** http-web, ftp, smtp-mail,http-web. The sequence of actions in bold are indicative of abnormal behavior perhaps a potential remote shell. In figure 1 different type of Anomalies are showcased which we can study using classification techniques.

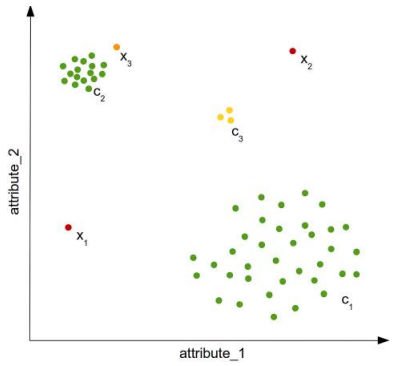

Fig. 1. x_1 and x_2 are global anomalies while x_3 is a local anomaly

III. Detection Algorithms

Most anomaly detection algorithms require a set of purely normal data to train the model and they implicitly assume that anomalies can be treated as patterns not observed before. Since an outlier may be defined as a data point which is very different from the rest of the data, based on same measure, several detection schemes can deal with the problem of anomaly detection. Machine learning techniques can be broadly classified into three categories:-

Supervised Anomaly Detection: Using these techniques we learn a classifier using labeled instances belonging to normal and abnormal class and then assign a normal or anomalous label to a test instance. However, this setup is practically not very relevant due to the assumption that anomalies are known and labeled correctly. For many applications, anomalies are not known in advance as in our case or may occur spontaneously as novelties during the test phase.

Semi-supervised Anomaly Detection: This technique also uses training and test datasets, whereas training data only consists of normal data without any anomalies. The basic idea is, that a model of the normal class is learned and anomalies can be detected afterwards by deviating from the model. The idea is also known as ”one-class” classification. Some of the well known algorithms are One-Class Support Vector Machine and Autoencoders. In general any density estimation method can be used to model the probability density function of the normal classes, such as Gaussian Mixture Models or Kernel Density Estimation.

Unsupervised Anomaly Detection: When using this technique, a classifier is built without knowing the true labels. Such a classifier should be able to take in the input data and come up with a structured answer based on the patterns which it is trained to recognize. Depending upon the problem statement, a model could be trained in a supervised or an unsupervised manner. Generally, Anomaly Detection algorithms learn in an unsupervised manner as the labeled data is hard to get and the data is generally imbalanced. Most of the time the abnormal data has a ratio of 1: million in a dataset. Thus, unsupervised learning is preferred in such cases. The techniques used in this category assume that normal data is more frequent than abnormal data. If this assumption is not true then there are chances of high false positive rate. In the following section we explain such techniques that have been used to solve this kind of detection problem.

A. Support Vector Machines

SVM first maps the input vector into a higher dimensional feature space and then obtains the optimal separating hyper-plane in the higher dimensional feature space. Moreover, a decision boundary, i.e. the separating hyper-plane, is determined by support vectors rather than using the entire training samples for this purpose and thus is extremely robust to outliers. In particular, this classifier is designed for binary classification. That is, to separate a set of training vectors which belong to two different classes. Note that the support vectors are the training samples close to a decision boundary. It also provides a user specified parameter called penalty factor. Thus, one can make a trade-off between the number of misclassified samples and the width of a decision boundary.

We have used One-Class Support Vector Machine (OCSVM) which can be considered as a regular two-class SVM where all the training data lies in the first class and the origin is the only member of the second class. The basic idea of the OCSVM is to map the input data into a high dimensional feature space using an appropriate kernel function and construct a decision function to best separate one class data from the second class data with the maximum margin. In this paper, both the origin and the data that close enough to the origin belong to the second class and they are considered as the anomalies. Figure 2 gives the geometrical interpretation of the OCSVM.

Although one-class SVMs are heavily used as a semi-supervised anomaly detection method, it is an unsupervised algorithm by design when using a soft-margin. In the unsupervised anomaly detection scenario, the one-class SVM is trained using the dataset and afterwards, each instance in the dataset is scored by a normalized distance to the determined decision boundary. Additionally, one-class SVMs have been modified such that they include further robust techniques for explicitly dealing with outliers during training. The basic idea is that anomalies contribute less to the decision boundary as normal instances.

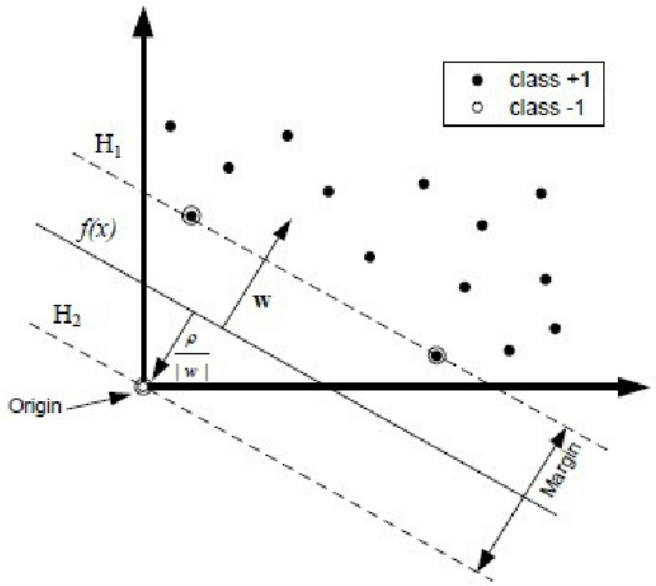

Fig. 2. One-Class Support Vector Machine

B. K-Nearest Neighbour

The k-nearest-neighbor global unsupervised anomaly detection algorithm is a straightforward way for detecting anomalies. As the name already implies, it focuses on global anomalies and is not able to detect local anomalies. First, for every record in the dataset, the k-nearest-neighbors have to be found. Then, an anomaly score is computed using these neighbors, where two possibilities have been proposed: Either the distance to the kth-nearest-neighbor is used (a single one) or the average distance to all of the k-nearest-neighbors is computed. The choice of the parameter k is of course important for the results. If it is chosen too low, the density estimation for the records might be not reliable. On the other hand, if it is too large, density estimation may be too coarse. Thus, this algorithm will not be able to detect anomalies close to clusters and will assign small scores.

C. Clustering Based Approach

K-means is a clustering analysis algorithm that groups objects based on their feature values into K disjoint clusters. Objects that are classified into the same cluster have similar feature values. K is a positive integer number specifying the number of clusters. There are five simple steps of the K-means clustering algorithm:

- Find the number of clusters.
- Initialize the K cluster centroids randomly.
- Assign each data point to the closet centroid.
- Update the centroids using current cluster memberships.
- Repeat if the convergence criterion is not met, and repeat the assignment.

We apply the K-means clustering algorithm to training datasets which may contain normal and anomalous data without being labeled as such in advance. The rationale behind this approach is the assumption that normal and anomalous data form different clusters in the features space. Of course, the data may contain outliers which do not belong to a bigger cluster, yet this does not disturb the K-means clustering process as long as the number of outliers is small.

D. More Algorithms

There are other algorithms available as well which are popular for detecting anomalies in an intrusion detection system. Some of them are using Artificial Neural Networks, Self-Organizing Maps, Decision Trees, Naive bayes, Genetic Algorithms, Fuzzy Logic, Hybrid Classifiers and Ensemble classifiers. However, we will limit our scope to the above algorithms only as these are some algorithms that we have experimented and obtained significant results with.

IV. HIDS USING OSQUERY

OSquery is one of the FOSS developed at Facebook and is popularly used as an instrumentation framework for Windows, OSX and Linux. This tool makes low-level operating system analytics and monitoring both performant and intuitive by exposing an OS as a high-performance relational database, thus allowing us to write SQL queries to explore operating system data. Different tables represent abstract concepts such as running processes, loaded kernel modules, open network connections, browser plugins, hardware events or file hashes. In the next section we will describe about the different concepts and some interesting queries. In the upcoming sections we will describe our approach in building a model that can predict anomalies.

A. Extracting Features from OSquery

In order to understand how we have extracted features from training our model, we will first explain different modes of osquery, its concepts and query syntax. We will outline some interesting queries that we have used in aggregating events.

Modes of Osquery: This framework provides two flavors, one which exposes a interactive console to obtain information instantly, try out new queries and explore our operating system and other allows to schedule queries and can act as host monitoring daemon. They are explained below with suitable examples:

osqueryi: This mode provides a real time console for inputting queries and fetching useful information. With the power of a complete SQL language and dozens of useful tables built-in, osqueryi is an invaluable tool when performing incident response, diagnosing a systems operations problem, troubleshooting a performance issue, etc. Figure 3 shows an example of osqueryi, where we use this console to simply query the name of the top 5 processes and pid in an OS.

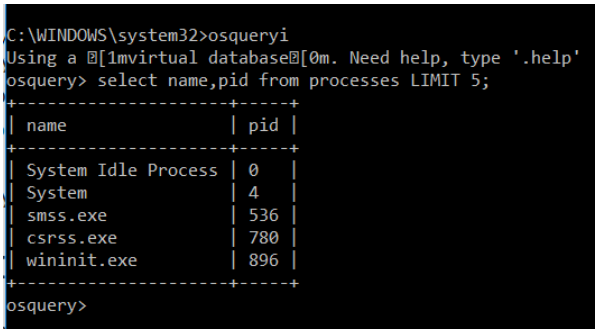

Fig. 3. OSqueryi Interactive Console

osqueryd: This is a daemon which reads the queries from the configuration file (osquery.conf) and logs the events accordingly into the osqueryd.results.log file. More information about these files and format of the configuration file can be found in official docs [1]. Figure 4 shows a typical configuration file which is used for querying the database. Each of the query is generally given a name, a short description, interval at which it has to be repeated and additionally we can provide custom packs which can be very specific searching for indicators of compromise (IOCs) in the system like registry values created by a trojan.

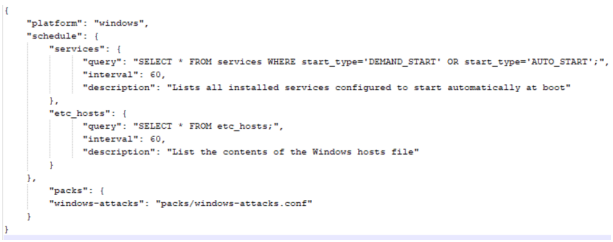

Fig. 4. osquery configuration file

Concepts of OSquery: As explained above, everything in osquery is a SQL table, hence our entire operating system could be considered as a series of tabular concepts. Each concept thus becomes a SQL table. There are tables associated with concepts like processes, sockets, the filesystem, a host alias, scheduled services, running kernel modules or a registry entry etc. There are some other informational things like OS version, CPU features or memory details that are not tabular but rather a body of details with labeled data. We can select all this into a table with a single row and many columns. When we want to inspect a concept, we can SELECT the data and in real-time the associated OS Apis are called. As seen in figure 3 we inspect the processes concept and obtain columns: name and pid for top 5 processes. There is a complete list of the schemas available [3] which can be used to obtain a variety of information.

SQL Syntax for OSquery: The OSquery SQL language is a superset of SQLite's. We can SELECT any column from the concept table. All mutation-based verbs like INSERT, UPDATE, DELETE and ALTER are valid as well.It was interesting to see that osquery also supports Math functions like sqrt, log, cos, tan etc. Now we will outline some interesting queries that can be used to keep track of anomalous behavior.

- ARP Spoofing:
  SELECT * FROM (SELECT COUNT(1) AS mac_count, mac FROM arp_cache GROUP BY mac) WHERE mac_count > 1;
  This query checks if there are multiple entries against a single mac address. If the count is greater than 1, it provides a table of such address and count from arp_cache concepts.
- Processes listening on Network Ports:
  SELECT DISTINCT process.name, listening.port, listening.address, process.pid FROM processes AS process JOIN listening_ports AS listening ON process.pid = listening.pid;
  This query uses two concepts and joins both the tables (processes and listening_ports) to provide a list of ports and listening process on them.
- Executables hiding as svchost:
  SELECT * FROM processes WHERE LOWER (name)= 'svchost.exe' AND LOWER(path)!= 'c:\windows\system32\svchost.exe' AND LOWER (path)!= 'c:\windows\syswow64 \svchost.exe' AND path!='';
  This query detects processes which masquerade as legitimate Windows processes (for stealth) by checking where they are stored on the host.

We have used such queries and some simple features to build our feature list as shown in table I. Similar features can be used for linux systems. One can also use syslog and shell history in the feature list. However we have limited ourselves to these features only, for now.

TABLE I
FEATURE LISTS

Feature Group	Concepts Used for osquery
Basic Features	Number of schedules tasks at startup, shared_resources, different wmi consumers, logged in users, services running, etc_hosts
Network Features	Number of open ports, listening processes, mac_count from arp cache against address
Registry Features	Number of run keys in registry hive.
Miscellaneous Features	Number of system process like svchost running with incorrect system path, registry entries checking the artifacts for a malware.

B. Model for HIDS

After we gathered all the features from the user machine, we can pre-process it and train our model based on any detection algorithms described in III. In order to detect anomalies in the system we worked with two detection algorithms namely

k-means clustering and OCSVM. The overall procedure for Anomaly Detection can be understood by studying figure 5. The logs that are obtained, containing events corresponding to queries in the configuration file are first pre-processed to obtain the vector set. Every row in this vector set corresponds to the events happening per minute in the user system. However, this can be varied, by changing the interval field in the queries of the configuration file. A model is trained using this data. Similarly test data is processed to a similar feature vector and inputted to the detector. All the events are grouped with predicted value being either -1 or +1. All the events for which the values predicted are < 0, are considered as anomalies for an OCSVM model.

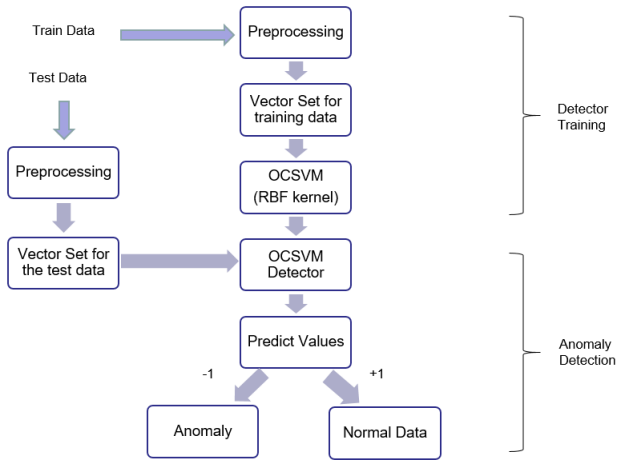

Fig. 5. Model Training and Detection of Anomalies

V. Evaluation

For an anomaly detection system, a thorough evaluation is particularly crucial to perform, as experience shows it is far too easy to say than done. Devising a sound evaluation schemes is not easy. We will discuss some of the evaluation challenges that we faced and later provide results from our detection model.

A. Challenges in Evaluation

Difficulty of Data: The most significant challenge an evaluation faces is the lack of appropriate data. There are no public sets available for assessing such types of anomalous detection systems. This might be due to the fact that such datasets might have the risk of exposing sensitive data for an organization and a user. Hence most of the time data is self-collected and activities are simulated to assemble any training and testing data. However, in general this is not an easy task, as most lack access to an appropriately sized organizational network. It is crucial to realize that activity found in a small laboratory network differs fundamentally from the aggregate traffic. Conclusions drawn from analyzing a small environment cannot be generalized to settings of larger scale.Thus, lacking a sense of realism.

Interpretation of Results: In addition to correctly identifying attacks, an anomaly detection system also needs to support the operator in understanding the activity and enabling a quick assessment of its impact. In other applications of machine learning, we do not see a comparable problem, as results tend to be intuitive there.

B. Setup and Dataset

Evaluation of this model is done on cloud virtual machines hosted as AWS instances. In order to make the setup vulnerable we allowed all inbound and outbound traffic. The figure 7 shows all AWS instances used for testing. For evaluation of our model we used a Windows instance, and simulated an attack using kali. We collected 72 hours of activity as our

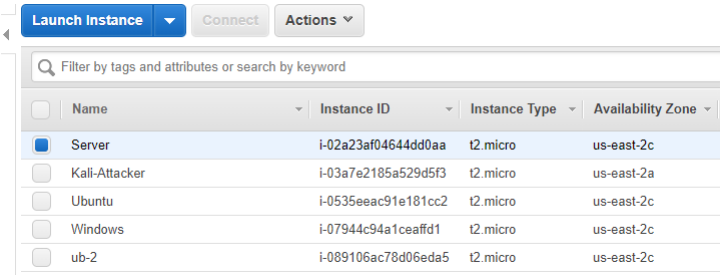

Fig. 6. lab Setup

training set, most of the events that we received in this set were some routine checkups by aws service like starting of some services, port scanning etc. We did not simulate any attacks while we were training data. We assume that this data is representative of benign activity. Please read on VII where we explain pitfalls of this assumption. In order to test our model we performed an attack using metasploit on kali and opened a reverse meterpreter session using the metasploit module: /multi/script/web_delivery. This module fires up a web server that serves a payload. This allows an attacker to embed a script in a web application, as soon as a user clicks on the link, a web server is started which opens up a meterpreter session at the attacker end.
We simulated the typical activity of an attacker, as shown below.

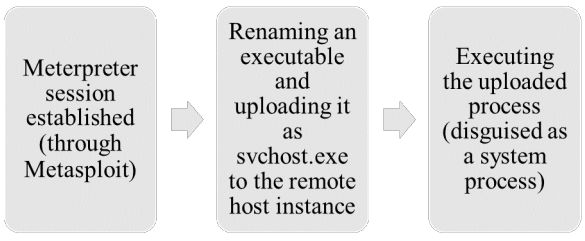

Fig. 7. Attack

Our model was able to detect the persistent step where a malicious binary was launched under the pretext of a system process.

C. Results

A sample of events obtained from the user system is shown in the figure 8.


```
{"name":"logged_in_users","hostIdentifier":"WIN-4H48I8HVEP4","calendarTime":"Wed Apr 25 16:17:45 2018
{"name":"logged_in_users","hostIdentifier":"WIN-4H48I8HVEP4","calendarTime":"Wed Apr 25 16:17:45 2018
{"name":"logged_in_users","hostIdentifier":"WIN-4H48I8HVEP4","calendarTime":"Wed Apr 25 16:17:45 2018
{"name":"open_ports","hostIdentifier":"WIN-4H48I8HVEP4","calendarTime":"Wed Apr 25 16:18:48 2018 UTC",
{"name":"open_ports","hostIdentifier":"WIN-4H48I8HVEP4","calendarTime":"Wed Apr 25 16:20:55 2018 UTC",
{"name":"open_ports","hostIdentifier":"WIN-4H48I8HVEP4","calendarTime":"Wed Apr 25 16:25:10 2018 UTC",
{"name":"open_ports","hostIdentifier":"WIN-4H48I8HVEP4","calendarTime":"Wed Apr 25 16:27:17 2018 UTC",
{"name":"services","hostIdentifier":"WIN-4H48I8HVEP4","calendarTime":"Wed Apr 25 16:27:17 2018 UTC","u
{"name":"services","hostIdentifier":"WIN-4H48I8HVEP4","calendarTime":"Wed Apr 25 16:27:17 2018 UTC","u
{"name":"open_ports","hostIdentifier":"WIN-4H48I8HVEP4","calendarTime":"Wed Apr 25 16:32:36 2018 UTC",
{"name":"open_ports","hostIdentifier":"WIN-4H48I8HVEP4","calendarTime":"Wed Apr 25 16:34:43 2018 UTC",
{"name":"services","hostIdentifier":"WIN-4H48I8HVEP4","calendarTime":"Wed Apr 25 16:38:00 2018 UTC","u
```


Fig. 8. sample osquery logs

A complete list of training data can be obtained from the github link as well [4]. These logs are processed to obtain the train and test vectors. Once the model is trained, it is saved and is now ready for testing purposes. Figure 9 shows a sample test vector. This test vector is fed into the model and results are predicted. As explained in 2 a value of -1

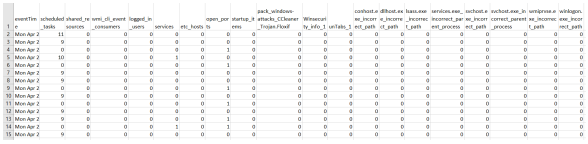

Fig. 9. sample pre-processed vector

indicates an anomaly. This concept is shown in figure 10. We evaluated the events for the anomalous behavior and obtained a feature vector which corresponded to the launch of malicious process. Although our prediction rate of anomalous events is

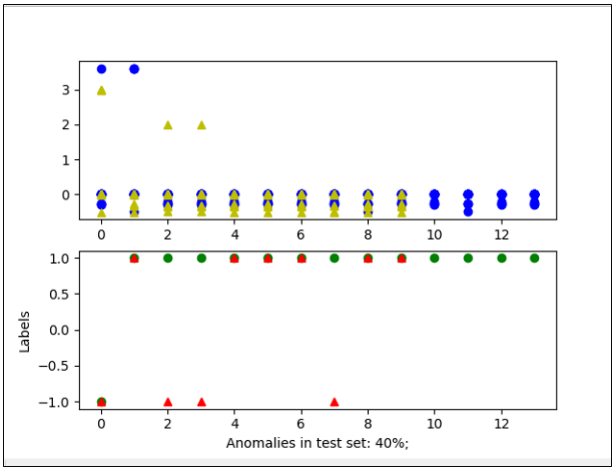

Fig. 10. Output of OCSVM detector: In the figure above, blue represents the pre-processed training data and yellow represents the preprocessed test data. In the figure below Red shows the predicted test data and green shows the predicted train data

100%, we have a high false positive rate of 45% (as out of the 40% anomalous activities we obtained, approximately half of them are false alerts). A similar analysis was obtained for other evaluations.


```
Name: 2, dtype: int64)
('Errorneous Anamoly Detected in test set: ',

scheduled_tasks                                 0
shared_resources                                0
wmi_cli_event_consumers                         0
logged_in_users                                 0
services                                        0
etc_hosts                                       0
svchost.exe_incorrect_path                      1
open_ports                                      0
startup_items                                   0
pack_windows-attacks_CCleaner_Trojan.Floxif     0
pack_Winsecurity_info_1                         0
pack_unTabs_1                                   0
pack_conhost.exe_incorrect_path                 0
pack_dllhost.exe_incorrect_path                 0
pack_lsass.exe_incorrect_path                   0
pack_services.exe_incorrect_parent_process      0
pack_svchost.exe_incorrect_path                 0
pack_svchost.exe_incorrect_parent_process       0
pack_wmiprvse.exe_incorrect_path                0
pack_winlogon.exe_incorrect_path                0
```


Fig. 11. Predicted Anomalous test vector

VI. Rebooting Snort

Snort is an open source NIDS software used to sniff traffic in real time and perform protocol analysis, content searching and matching (based on libpcap). The signature based approach detects attacks by looking for specific patterns such as byte sequences in network traffic, flags or known malicious instructions used by the attacker. Snort also has the ability to deploy any rule and hence they are not included with the software. However, when we used the rules from the Emerging Threats (ET) library or community rules developed by the Snort community on the cloud (AWS instance), most attacks were misclassified and some were not detected. Thus, we make a small set of rules to protect against common network attacks.

For analyzing the characteristics or behavior of each attack, it was first performed through a Kali Linux instance with the host running Wireshark for packet monitoring. After the useful features are extracted through Wireshark, they were modeled into a signature by combining protocol information, source and destination IP's, ports involved and action to be taken when an intrusion attempt is detected. These rule sets are then added to Snort configured on the AWS instance, where we simulate an attack and check if we can detect the same.

Attack Simulation and Detection:

A. SSH attacks

Automated programs often tests combinations (either dictionary based or brute force) of possible usernames and passphrases to break the SSH authentication mechanism. We simulate the same through Hydra. The identified features are: Connection to server, fixed destination port: 22, SYN flag set with a combination of threshold - 3 attempts in 60s. Through

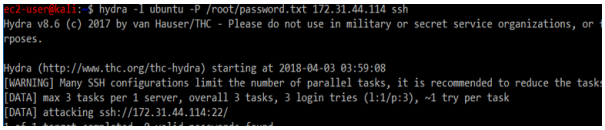

Fig. 12. Simulation of SSH attack through Hydra

this rule, we could successfully detect an SSH attack on the AWS instance, as shown below.

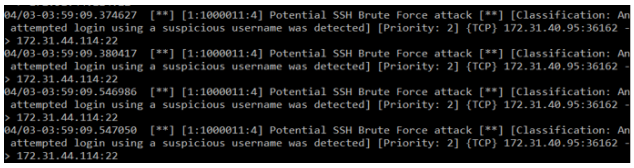

Fig. 13. Alert generated on AWS instance

B. Denial of Service attacks

DoS attacks are attempts to make an online service unavailable by overwhelming it with traffic.

1. HTTP Flood: In this attack, a large number of GET/POST requests are sent to a web server to consume all it's available resources. We used HULK (HTTP Unbearable Load King) to simulate this attack. The Wireshark capture for the same is displayed below. However, we notice here that the flood

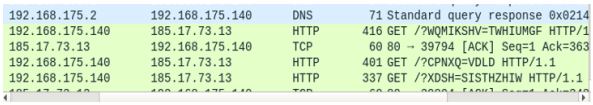

Fig. 14. HTTP flood monitored on Wireshark

has a very low threshold - 10 in 60s with a connection to the server. The attack can also be detected by performing payload content matching for GET/POST strings.

2. Amplification Attacks, Smurf and Fraggle Attack: The attacker spoofs the source address and sends a large number of ICMP and UDP packets respectively to the broadcast network. The responses received from all components on the network, then overload the targeted victim machine. The unique characteristic used for detection is packet threshold which are tracked by source IP with certain specific port activity (7,9,13,17) for UDP or echo type set as 8 (reply) for ICMP.

3. Generic Floods, UDP/ICMP: Tools such as hping3 and nping are used to make and send custom TCP/IP packets. These attacks can only be detected through terribly high thresholds for activity on specific ports. However, these are very broad and thus might misclassify some legitimate traffic as malicious. However, malicious traffic is always detected accurately.

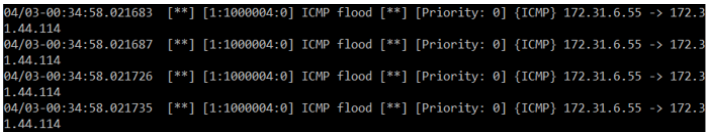

Fig. 15. AWS alert for ICMP flood

4. Land Flood: These attacks are caused by sending a packet to a machine with the source host/port same as the destination host/port i.e. by spoofing the source IP. However, these attacks not only have a high threshold with the TCP SYN flag set, they also use the sameip feature in the packets (for same source and destination IP's).

5. Teardrop Attack: The attacker sends mangled IP packets over-sized and loaded with payloads. Due to a bug in the TCP/IP fragmentation reassembly code in various OS, it leads to a crash of the OS. These attack packets not only have the Fragment flag set, but also a very specific id of 242.

C. Stealth Port Scans

The attacker employs basic security scanners like Nmap or develops their own custom tools to discover hosts and services on a network in the reconnaissance stage.

1. Xmas/TCP/Ping Scan: One of the most basic scans (Xmas) is to find out which ports are open and which are closed. The most distinguishable features observed were the

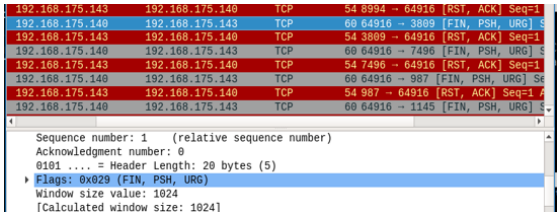

Fig. 16. Wireshark monitoring for Xmas Scan

the FIN, PSH and URG flags being set. This rule set helps us to identify a scanning attempt and pushes an alert to the console.

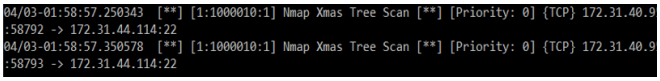

Fig. 17. AWS alert for Xmas Scan

A similar analysis is also done for the Ping and TCP scans. Some of the defining features are packet size = 0 or the ACK flag being set. The following are also successfully detected as shown below:

2. Nmap OS Fingerprinting: The Nmap scanner identifies the underlying OS by analyzing the SYN/ACK responses sent by the open ports. Each OS uses a different algorithm to calculate the timestamp value (like linear extrapolation for

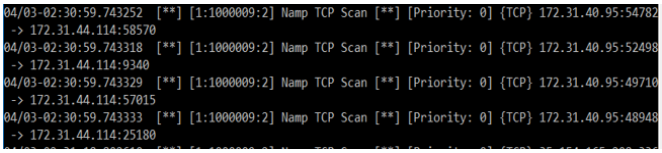

Fig. 18. AWS alert for TCP Scan

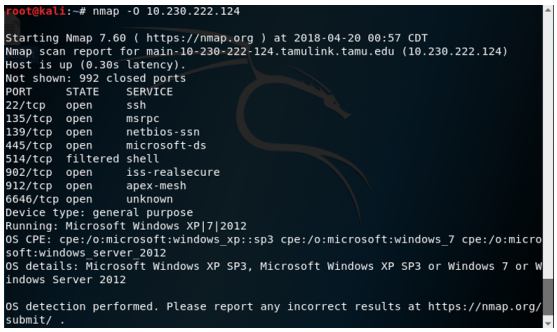

Fig. 19. OS Fingerprint Scan

example). These attack packets also have the FIN, SYN, PSH and URG flags set and thus help in detection.

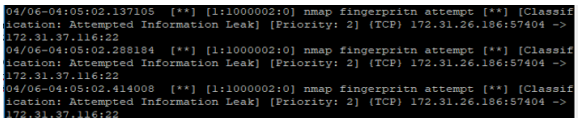

Fig. 20. AWS alert for Nmap OS Fingerprint Scan

D. ARP Spoofing

This technique is used by attackers for performing man-in-the-middle attacks by convincing the victim that the destination MAC address is associated with their IP address. Snort has built in functionality in the configuration file to avoid such attacks (wth hardcoded MAC's and their associated IP's).
Example: preprocessor arpspoof
preprocessor arpspoof_detect_host: IP MAC
However, there is very little automatic about this rule and hence a better detection technique is employed through OS-Query. However the hardcoded values also help us in identifying ARP cache overwrite attacks.

E. Web Based attacks

1. SQL injection: It refers to the technique of inserting SQL meta characters and commands into Web-based input fields in order to manipulate the execution of back-end SQL queries. These are attacks directed primarily against another organization's Web server. Some basic features used in signatures are the single-quote and its hex equivalent (eg. (%27)/(')), keyword's - select, union, insert, update, delete and the word 'or' with various combinations of its upper and lower case hex equivalents.

2. Cross Site Scripting attacks: These attacks work by embedding script tags in URLs and enticing unsuspecting users to click on them, ensuring that the malicious Javascript gets executed on the victim's machine. These attacks leverage the trust between the user and the server and the fact that there is no input/output validation on the server to reject Javascript characters. The distinguishing features include: </script> tags and an established connection to the server.

F. Remote Shell Access/ Remote Code Execution/ Malware Injection through Metasploit

Metasploit is a penetration testing tool employed by script kiddies to execute built in exploits for vulnerable machines. Earlier versions of Metasploit could be detected through payload content matching techniques as the data was unobfuscated. (string matching: stdapi_sys_config_getuid, stdapi_sys_config_sysinfo, stdapi_sys_process_execute etc.) However, in newer versions of Metasploit, data is being

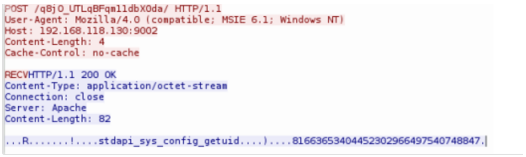

Fig. 21. Wireshark monitoring of earlier versions of Metasploit packets

encoded through techniques like Base-64, UTF-8 or simple XOR approaches.These packets can hence evade detection through signatures.

Fig. 22. Wireshark monitoring of Metasploit packets

VII. Limitations

In some sense, anomaly detection is also a type of classification problem: there are two classes, "normal"and "abnormal" and the objective is to determine which of the two more likely matches an observation. However, a basic rule of machine-learning is that one needs to train a system with specimens of all classes, and, crucially, the number of representatives found in the training set for each class should be large [6].
Yet for anomaly detection aiming to find novel attacks, by definition one cannot train on the attacks of interest, but only on normal traffic, and thus having only one category to compare new activity against. In other words, one often winds up training an anomaly detection system with the opposite of what it is supposed to find in a setting certainly not ideal, as it requires having a perfect model of normality for any reliable decision. We have tried to model this (was running

as daemon for several days on the system). However, this is far from ideal as this doesn't mimic the true user experience. We would certainly like to collect organizational data and run our evaluation on them.

Also in intrusion Detection, the relative cost of any misclassification is extremely high as compared to other machine learning applications. A false positive requires spending expensive analyst time examining the reported incident only to find that it is representing benign activity. False negatives on the other hand have the potential to cause serious damage to organization.

One of the other limitation or challenge that is evident is transferring the results into actionable reports for the admin. This is challenging as one has to find the difference between "abnormal activity" and "attacks". As stated in V we have some false positives corresponding to the system stop/start activity. While these activities may not be exactly malicious but they do have some characteristic features which distinguish it from other normal activity. We particularly got a large number of services starting at this point. This is certainly not an "attack" but our system characterized it as a "abnormal point", hence stressing the fact that machine learning algorithms do not make any mistakes within their model of normality; yet for an admin it is the interpretation of results that matters.

Lastly one of the limitations that we see is the diversity of activities or traffic on a user system, which leads to misconceptions about what anomaly detection technology can actually achieve in operational environments.

Future work includes combining the defenses proposed: network attack detection through Snort and host intrusion detection through OSQuery so that they can be employed as a single tool for intrusion detection or as a service.

VIII. Conclusion

In this paper, we demonstrate our approach for identifying anomalous activities - through traditional signature based techniques combined with the host intrusion detection capabilities of OSQuery. Attacks on the cloud are imminent from outside, where the attacker tries to gain control of the services being provided leading to breakdown of infrastructure. This includes DoS attacks, ARP poisoning, Man-in-the-Cloud, Port scans, web delivery exploits through Phishing attacks among others. Our rule sets help in securing the cloud infrastructure from such attacks. However, in scenarios where detection is not possible through Snort, we employ a novel host based behavior monitoring system through OSQuery and use OCSVM classifier techniques to distinguish between benign and malicious activities. This combination helps us not only to drop network packets but also identifies remote users (logged_in users), new files introduced in the system (registry keys through Trojan's) and malicious executables hiding in plain sight (svchost) providing comprehensive security to the cloud and protection against known and unknown attacks.

Acknowledgment

It is our immense pleasure to express gratitude to Dr. Guofei Gu (Department of Computer Science - Texas A&M University) as a guide who provided us with constructive feedback and encouragement during the project helped us in completing the same.

References

[1] https://osquery.readthedocs.io/en/stable/

[2] W. Hu, Y. Liao, and V. R. Vemuri, "Robust Anomaly Detection Using Support Vector Machines, in Proc. International Conference on Machine Learning, 2003

[3] https://osquery.io/schema/3.2.4

[4] Hu Jiankun and Yu Xinghuo 2009 A Simple and Efficient Hidden Markov Model Scheme for Host-Based Anomaly Intrusion Detection IEEE Network Journal

[5] Eskin Eleazar, Andrew A., Michael P., Leonid P. and Sal S 2002 A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data

[6] Outside the Closed World: On Using Machine Learning For Network Intrusion Detection, 2010 IEEE Symposium on Security and Privacy

[7] ShikhaAgrawal and JitendraAgrawal 2015 Survey on Anomaly Detection using Data Mining Techniques Procedia Computer Science , vol 60 pp 708 - 713

[8] V. Mahajan and S K Peddoju, "Integration of Network Intrusion Detection Systems and Honeypot Networks for Cloud Security, " International Conference on Computing, Communication and Automation (ICCCA 2017)

[9] SNORT Users Manual - Amazon AWS. Retrieved from http://manual-snort-org.s3-website-us-east-1.amazonaws.com/node27.html

[10] About Snort (n.d.), Retrieved from https://www.snort.org/

[11] Z. Al Haddad, M. Hanoune, A. Mamouni, "A Collaborative Network Intrusion Detection System (C-NIDS) in Cloud Computing, " in International Journal of Communication Networks and Information Security (IJCNIS), vol.8, No.3, December 2016.

[12] Emerging Threats Rule set (updated July 2016). Retrieved from https://rules.emergingthreats.net/open/snort-2.9.0/emerging-all.rules

[13] Metasploit for the Aspiring Hacker (February 2,2016). Retrieved from https://null-byte.wonderhowto.com/how-to/hack-like-pro-metasploit-for-aspiring-hacker-part-12-web-delivery-for-linux-mac-0168734/

[14] github: https://github.com/srishti-agarwal/OSQueryOutlierDetection