S. C. Pallaprolu, R. Sankineni, M. Thevar, G. Karabatis and J. Wang, "Zero-Day Attack Identification in Streaming Data Using Semantics and Spark," 2017 IEEE International Congress on Big Data (BigData Congress), Honolulu, HI, USA, 2017, pp. 121-128, doi: 10.1109/BigDataCongress.2017.25.

https://doi.org/10.1109/BigDataCongress.2017.25

Zero-day Attack Identification in Streaming data using Semantics and Spark

Sai C. Pallaprolu, Rishi Sankineni, Muthukumar Thevar, George Karabatis, Jianwu Wang
Department of Information Systems
University of Maryland, Baltimore County, Baltimore, MD, USA
Email: {p39, du83014, ai22698, georgek, jianwu}@umbc.edu

***Abstract*—Intrusion Detection Systems (IDS) have been in existence for many years now, but they fall short in efficiently detecting zero-day attacks. This paper presents an organic combination of Semantic Link Networks (SLN) and dynamic semantic graph generation for the on the fly discovery of zero-day attacks using the Spark Streaming platform for parallel detection. In addition, a minimum redundancy maximum relevance (MRMR) feature selection algorithm is deployed to determine the most discriminating features of the dataset. Compared to previous studies on zero-day attack identification, the described method yields better results due to the semantic learning and reasoning on top of the training data and due to the use of collaborative classification methods. We also verified the scalability of our method in a distributed environment.**

***Keywords*—IDS; Flow Creation, Semantic learning and reasoning, Spark Streaming, Collaborative mining, Zero-day Attack Identification.**

I. Introduction

The development of computing infrastructures has proliferated in our lives and large amounts of personal and sensitive information are hosted on servers which are available through the world wide web [9]. New malicious activities in the Internet also known as intrusions developed by hackers aim to gain unauthorized access to computer systems by exploiting vulnerabilities of computing systems, to compromise the integrity, validity and confidentiality of stored data.

These vulnerabilities used by the attackers weaken the security of systems [2]. An Intrusion Detection System (IDS) is a mechanism that tries to identify a set of actions or disallowed behaviors in a computer system. There are two main types of intrusion detection techniques: misuse detection and anomaly detection. Misuse detection recognizes a suspicious behavior by comparing its signature with a stored database of attacks signatures; the only drawback of this technique is that it cannot detect new attacks. Snort is an example of an IDS using misuse detection techniques. Anomaly detection is another technique, which creates a model of normal behavior, and tries to detect any abnormal deviation from that model resulting in generation of corresponding alerts [10].

Network Security issues are becoming serious with the growth of Internet. Current IDS's use certain detection algorithms to predict the abnormal traffic and guarantee that only benign traffic is allowed to access the system. The main problem of traditional IDS is they take a long time to predict the attacks especially when high volumes of data need to be analyzed.

The key objective of our paper is to create an IDS system which combines the Flow based Intrusion Detection System and Semantic Link Network in the Apache Spark environment and result in real or near-real time computation and analysis of incoming traffic.

II. Related Work

In general, most types of IDSs utilize logic operations, statistical techniques, and machine learning approaches to distinguish between different types of network activities [6], [14], [18], [22], [25]. There have been few studies which address zero-day attack detection problems, most of them utilized unsupervised anomaly detection techniques to discover these types of attacks [18], [22]. Support Vector Machines (SVM) have been utilized by Jungsuk et al. in [25]. Clustering approaches have been used in [14], [22] to discover new attacks types. Hendry et al. in [14] proposed a hybrid supervised and unsupervised clustering algorithm for zero-day attack signature creation. The problem with this approach is the difficulty of creating sufficient and accurate new attack signatures at real-time. Zhichun et al. in [19] proposed a model to detect zero-day worms by analyzing the invariant content of polymorphic worms, making analytical attackresilience guarantees for the signature generation. In [22] Song et al. introduced an approach that can detect zero-day attacks from IDS alerts. The limitation of this approach was the large amount of alerts to be analyzed in order to generate zero day signatures. While such techniques have the capability of detecting zero-day patterns, however, they produce large and unmanageable amounts of false alerts [25].

Our previous work in [3] proposed an efficient approach to detect zero-day attacks using linear data transformation and anomaly detection techniques by processing suspicious network connections which do not match known attack signatures at run-time. The linear data transformation module passes the suspicious connections to discriminant functions, which differentiate between the behavior of known attacks and normal activities [3]. Such functions are used to calculate the estimated probabilities of attack patterns in network connections. These probabilities are compared to a user tunable threshold to raise alerts about zeroday attacks. The anomaly detection technique relies on a modified implementation of the one-class NN algorithm [3]. It has been used to detect anomalies and thus to discover zeroday attack types using an assigned anomaly score. The experimental results indicate that linear data transformation, when applied on network data using discriminant functions attained a detection rate (TPR)

of 0.83. In contrast, this paper, using semantic learning and reasoning on Spark streaming platform attained a detection rate (TPR) of 0.95. It can be also observed from our comparison that semantics based learning is quite effective in term of TP rate compared to other approaches, and it achieves competitive results in term of false positive rate.

III. Approach

Our overall approach is shown in Figure 1. In this section we describe the architecture of the system and its core functionality. This section describes in detail how a semantic link mesh network is formed from a training set, creation of flows, training the classifier with non-zero day attacks, dynamic generation of semantic link network for zero day attacks and classification metrics.

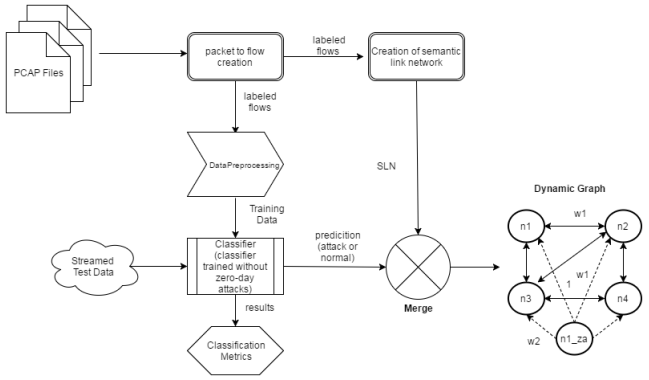

Figure 1. Process Flow of Overall Approach

As shown in Figure 1, information from the input pcap files is converted into flows. Using pre-existing labeled flow data we generate a semantic link network (SLN), which is a graph containing nodes that represent attacks, and edges that represent the semantic relationships between attacks. Once the SLN is completed, it can be used at run-time. In addition, the labeled flow data is used to train a classifier, which is also used at run-time in conjunction with the SLN. These tasks are part of the training phase of the system. It is worth mentioning that some existing attacks in the dataset have been removed from the training part of the dataset, and they have been included in the testing part of the dataset to simulate the existence of zero-day attacks.

At run-time (testing phase) we get streaming incoming flows from the network. These flows contain simulated zero-day attacks which were excluded from the training dataset, therefore the classifier and the SLN do not contain any information about these attacks. When incoming flows are streamed to the system, they pass through a classifier and with information from the SLN, our system decides whether an incoming flow is malicious or benign. If a zero-day attack is identified then the SLN is dynamically updated to reflect this information.

Training phase: Snort [11] is a real time open source network based IDS that has the ability to sense the network traffic and can perform packet analysis. We use snort to collect alerts on incoming data and then label the incoming flows as such.

A. Pcap to CSV converter (Flow Creation with Labels)

We first convert the input pcap file into a csv file using tshark [7], which is used for analyzing packets and also to monitor the network. In order to get the data points that are benign, we used a subtraction module that subtracts two files and gives a resultant file. Our main idea is to subtract the snort alert file from the csv file that is generated from pcap to csv convertor, to get benign data points as shown in figure 2. Then we create a labeled dataset by combining the labeled alert data with the benign data.

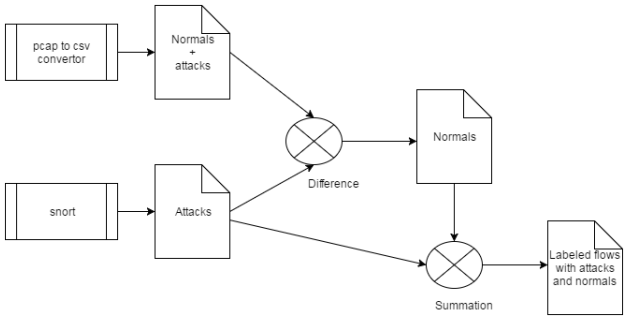

Figure 2. Generation of labeled dataset

Lines 1-7 in Algorithm 1 convert the original pcap file into a csv file using tshark commands in a Linux based system. Lines 8-18 summarize the subtraction of csv file containing alerts from the csv file containing both alerts and benign, resulting in a file that contains only normal data. Finally, we combine both files that contain alerts and normal to form a complete labeled dataset of flows.

Algorithm 1: Algorithm for converting the original pcap file into a csv file using tshark commands.

1 **Return**: Dataset of flows containing attacks and benign.
2 **Features**: A list of common features extracted from packet data to create one flow.
3 **procedure** createFlow(duration, features, packets)
4 Load the pcap file and convert it into csv file.
5 Feed pcap file as an input to snort.
6 Capture the alert pcap file and store it in local db.
7 Convert the alert pcap file into csv.
8 Spark DF df1 $\leftarrow CSV(output\ of\ step\ 1)$.
9 Spark DF df2 $\leftarrow CSV with\ attacks(snort\ output)$.
10 Intersection of two dataframes df1 and df2.
11 **For** i in range (length (df2)):
12 **For** j in range (length(df1)):
13 **if** $df1(i) == df2(j)$ **then**
14 | Delete df2(j);
15 **end**
16 **end-loop**
17 **end-loop**
18 df3 $\leftarrow df2(containing\ only\ normal\ flows)$.
19 Union of df2 and df3.
20 df4 $\leftarrow df2Udf3(attack\ and\ normal\ flows)$.

B. Creation of semantic link network

The main motivation to use semantics is that the classification models do not provide any relationships between the types

of attacks. Improvement in attack prediction and decrease in false positives can be achieved if we capture the contextual relationships between the attacks [5]. Similarity between the nodes is a measure of correlation between the features with respect to the attacks. The construction of the SLN consists of two major steps:

Step 1: Creation of weighted links among the nodes. Semantic link networks contains both attacks and normal activities as nodes. It is expected that the SLN shows a weak relation between attacks and benign activities. The features of the flows dataset fall into three categories as shown in Table 1.

Feature Type	Feature Names
Nominal(categorical features)	Protocol, flags
time-based(continuous)	Start time, end time
Location based(continuous)	Source ip ,destination ip, source port,destination port
Statistical features(continuous)	Packets, octets

Table I. CATEGORIES OF FEATURES IN THE DATASET

To find the weighted links among the nodes, we create a feature-node similarity matrix. Before creating the node-similarity matrix we perform some preprocessing on the numerical features, by applying binning since the similarity measures we use in the SLN require binary values.

Source port	Protocol	Attack label
40	TCP	back attack
60	UDP	smurf
80	UDP	DOS
99	HTTP	Neptune

Table II. SUBSET OF RAW DATA FOR SIMILARITY CALCULATION

A node similarity matrix consists of features in rows and attack types as in columns. It also contains normalization frequency which is in the binary domain [0, 1] of each feature f with each node n_i.

To calculate the normalized frequency we followed a series of steps as shown below:

1) Calculation of feature frequency with respect to particular class ff_c.
2) Calculation of total frequency of that feature in entire dataset f_t.
3) Calculation of weight, $W = ff_c/f_t$.
4) Weight discretization (conversion of continuous weights into binary with threshold=0.5)

After applying the above steps on the subset of raw data in Table 2, the binary values are formed as shown in Table 3.

	back attack	smurf	DOS	Neptune
TCP	1	0	0	0
UDP	0	1	1	0
HTTP	0	0	0	1
Source port[40,60)	1	0	0	0
Source port[60,80)	0	1	0	0
Source port[80,100)	0	0	1	1

Table III. BINARY VALUES FOR THE SIMILARITY MATRIX

We then apply Anderberg's similarity [13] on the preprocessed data to generate the Node Similarity Matrix (Table 4).

	back attack	smurf	DOS	Neptune
backattack	1	0.1	0.2	0.02
smurf	0.1	1	0.78	0.3
DOS	0.2	0.78	1	0.0123
Neptune	0.02	0.3	0.0123	1

Table IV. UN-NORMALIZED SIMILARITY MATRIX

To normalize the data in the Similarity Matrix we use Min-Max normalization technique [17] by considering the maximum value as the second largest value in each feature. After normalization, we get an initial SLN as shown in Figure 3.

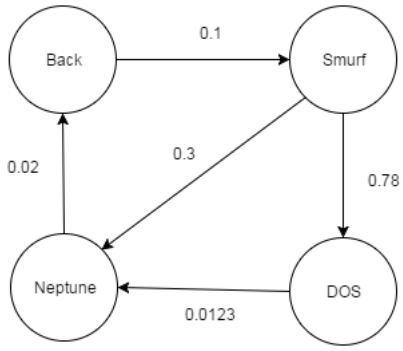

Figure 3. Initial SLN

Step 2: Reasoning on such links to progress the semantics. It was observed in the initial SLN that some nodes are not connected and do not have any semantic relationships [4]. To get the implicit semantic relation between the non-connecting nodes we applied logical reasoning [5].

Let us assume that Node-Similarity-Matrix is represented as 'P', so that p_{ij} represents the weight of semantic link from p_i to p_j and p_{ji} represents the weight of semantic link from p_j to p_i. If there are no direct semantic links from p_i to p_j then p_{ij} = p_{ji}=0 [6].

For a given Node-Similarity-Matrix P, the result of $(p_{ij}Xp_{jr})$ means that the p_i node can reach the p_r node through semantic links in one reasoning step from two links $p_i \rightarrow p_j$ and $p_j \rightarrow p_r$. Reasoning steps can be performed by raising P to the power of v (i.e. $P^v + 1 = P^vXP$), where $p_{ir}^v + 1$ means that the node p_i reaches the node p_r in V+1 steps.

C. Data Preprocessing

In real world the data is incomplete, noisy and inconsistent. To deal with these issues a proper data pre-processing techniques should be performed on top data beforehand. We performed several data pre-processing techniques as follows:

Filling missing values and normalization: Filling missing values in random biases the classification model. So we filled the missing values in a smarter way by replacing the missing value of a particular attribute by the attribute mean of rest of the values of that particular feature which belong to the same class.

We normalized the feature values of all the features vectors present in our dataset in order to not get influenced by features

with wide range of values while computing distances. For example, if a feature has range in [-0.5,+0.5] and another feature has range in [-100; 100], a small change in the second feature is probably more influencing when computing the distance of two feature vectors. This large variation may lead to inconsistency in the mining model especially in distance based models such as KNN. We normalized each value in the feature vector by dividing it with the maximum value in that particular feature value. If F is the feature with length n, then f_i will be the instance of vector and the formula we followed for normalization is

$$Normalized\ Feature\ F - norm = f_i / f_{i-max}$$

where f_{i-max} = max value in that feature vector and i ranges from (1-n).

Feature selection and dataset splitting: We have used the KDD CUP' 99 data set to prove our methods, which is one of the most widely used dataset in cyber research. The data set contains a total of 23 attacks. A critical issue in data pre-processing is feature selection: instead of using all the features (attributes or variables) in the data, one can selectively choose a subset of features. There are a number of benefits of feature selection: (1) dimensionality reduction to reduce the computational cost; (2) noise reduction to revamp the classification accuracy; (3) more interpretable features or characteristics that can help identify and monitor the target variables or class types. These advantages are typified on KDD CUP'99 dataset. Out of 42 features, only a smaller number of them shows strong correlation with the targeted network attacks, meaning that computation is reduced, while prediction accuracy is increased via effective feature selection.

As a result, selecting the relevant features and ignoring the irrelevant and redundant features has become indispensable. However, when dealing with large amounts of data, most existing feature selection algorithms do not scale well, and their efficiency may significantly deteriorate to the point of becoming inapplicable. For these reasons, we propose a distributed approach for partitioned data using Minimum Redundancy Maximum Relevance [21] (MRMR) feature selection algorithm on the Apache Spark platform. MRMR feature selection, as a preprocessing step on the dataset, is highly efficient for dimensionality reduction, removing unrelated data, and improving learning accuracy.

The main benefit of MRMR feature set is that by reducing mutual redundancy within the feature set, these features capture the class characteristics in a broader scope. Features selected within the MRMR framework are independent of class prediction methods, and thus do not directly aim at producing the best results for any prediction method. The fact that MRMR features improve prediction for all four methods we tested confirms that these features have better generalization property. This also implies that with fewer features the MRMR feature set can effectively cover the same class characteristic space as more features in the baseline approach. Nevertheless, the recent surge in dimensionality of data raises a serious challenge to multiple prevailing feature selection methods with respect to coherence and efficacy.

In our dataset we have total of 42 features including the class attribute. Most of the data mining algorithms show inefficiency and are not effective due to high dimensionality [26]. To deal with this problem we performed feature selection process using information-gain [26] to reduce the features. We performed feature selection and achieved success in ranking the attributes based on information gain. After the dimensionality reduction, we split the dataset into two parts training and testing sets to validate our model. The composition of training set from the dataset is 60 percent and the rest is the testing set.

score	Features	score	Features
0.9630	service	0.5531	logged_in
0.9452	same_srv_rate	0.4068	dst_host_count
0.9119	count	0.3708	dst_host_srv_diff_host_rate
0.8747	flag	0.2134	srv_count
0.8498	dst_host_diff_srv_rate	0.1999	srv_diff_host_rate
0.8226	dst_host_same_srv_rate	0.1546	dst_host_rerror_rate
0.7932	dst_host_srv_count	0.1489	protocol_type
0.6735	dst_host_serror_rate	0.1338	dst_host_srv_rerror_rate
0.6554	serror_rate	0.0959	rerror_rate
0.6302	dst_host_srv_serror_rate	0.0783	hot
0.6158	srv_serror_rate	0.0704	wrong_fragment
0.6158	num_access_files		

Table V. FEATURE SCORE OF TOP 25 ATTRIBUTES

Algorithm 2: Algorithm summarizing the preprocessing technique used to clean the dirty data

1 **Inputs**: Dirty data Dataframe(DF).
2 **Return**: Preprocessed data.
3 **Begin**
4 Locate all missing values in dataframe.
5 **if** $x = missing\ value$ **then**
6 | sum (values in same column per that x's class)/length of values in same column per x's class)
7 **end**
8 f_{i-max} = max value in that feature vector.
9 i ranges from (1-n) , n=length of the feature.
10 Normalized feature $f - norm = f_i / f_{i-max}$.
11 Import Info Gain.
12 **For** i in 0 to length of feature:
13 Information-Gain $f_i = IG(f_i)$.
14 **End-loop**
15 User defined threshold t = θ.
16 **For** i in 0 to length of columns:
17 **if** *Information-Gain* $f_i < \theta$ **then**
18 | *Remove* f_i
19 **end**
20 ***End-loop***
21 ***End***

Algorithm 2 summarizes the preprocessing technique we followed to clean the dirty data. Lines 2-5 summarize the process of removing missing values by replacing them the average of the values in that same feature per that particular class. Lines 6-8 summarize the process of normalization. Lines 9-18 summarize dimensionality reduction by removing certain columns with the information gain less than the user provided threshold value.

D. On the fly analytics and classification metrics

The run-time component of our approach, which uses Spark Streaming, is shown in Figure 6. Spark is known for

its in memory computing and provides a streaming framework called Spark Streaming [30].

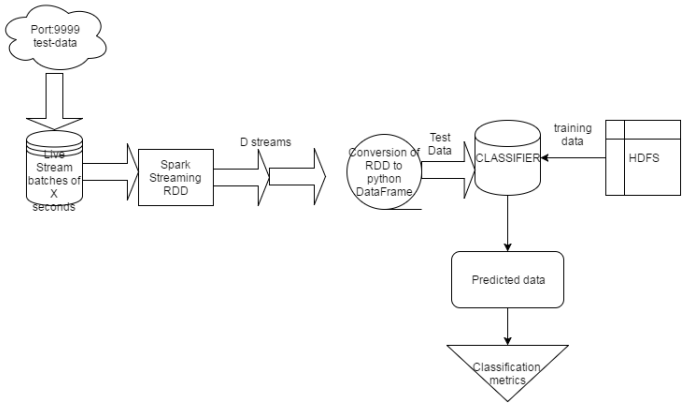

Figure 4. Discovery of zero-day attacks on streaming test data.

Spark ecosystem and streaming: We performed flow-based intrusion detection and prediction for network traffic streaming by using Apache spark streaming framework [29]. Spark Streaming is an extensible spark core API and the main properties of Spark-Streaming are scalability, high throughput and fault tolerant stream processing [16]. Spark is known for its feature called Resilient Distributed Dataset (RDD) [28] which is collection of elements distributed across the memory nodes of the cluster and can be operated on in parallel.

On top of RDD spark streaming provides DStreams which are a sequence of RDDs that are captured in certain interval of time. In our experiment we read the data from the open ports and capture data to the spark RDD and further converts that a python dataframe. One of the main reason we choose Spark ecosystem is its extensive feature to combine different formats of data processing tasks, which is needed for our cyber-attack detection application. We built our application using Spark ecosystem to bolster real-time streaming process and classification tasks. In our experiments to stream the test data, we sent the test data to one of the open ports by using awk scripts [1]. And from the ports we used spark streaming to get the data and fed as in test data to the classifier in continuously streamed batch fashion.

Attack prediction and Data Storage: In our experimentation we used KNN [27] classifier to identify whether an incoming test stream data point is an attack or benign. KNN classifier predicts an incoming flow as an attack or benign by using nearest neighborhood heuristics. For every incoming streamed test flow, KNN classifier calculates the Euclidean distance to all the flows in the training set and assign its label by taking voting amongst its 'k' number of nearest neighbor data points. In our experimentation we used the k value as 3.

To handle such large amounts of data one should adopt big data technologies as the traditional relational databases could not easily withstand the volume [23]. To handle these different varieties of data one requires a centralized platform like Hadoop file systems. In our experiments we stored the files in Hadoop file system [20]. We accessed HDFS by a user interface called Hue and it provided an access to load the files into HDFS.

Algorithm 3: Algorithm for Streaming test data and prediction of zero-day attacks

1 **Inputs**: Training set and streaming test data.
2 **Return**: Prediction and classification metrics.
3 Initialize spark context as SC.
4 Initialize streaming context as ssc.
5 Split the data set into training and testing set (60, 40).
6 Awk script to send the data to a port.
7 Collect the Dstreams into spark RDD with batch interval of 1 sec using streaming context.
8 Let i be the number of Dstreams.
9 **For** I in range (i):
10 Euclidean Measure (trainRDD(i) , testRDD(i)).
11 Assign labels for instance of testRDD.
12 Return the predictions.
13 **End-loop**
14 Calculate the accuracy and classification metrics.

Algorithm 3 summarizes the streaming of test data and attack prediction. Lines 1-5 depicts the important packages installed and initializing the spark and streaming context. Lines 6-10 summarizes the training of KNN classifier and testing the incoming Dstreams to predict whether the incoming data point is an attack or benign. Line 11 summarizes the calculation of classification metrics.

Dynamic node updating upon zero-day hit: As shown in Figure 4, we tested the data that is streamed from an open port in the client system and we dynamically generated the relevant attacks with similarities for the zero day attacks along with non-zero day attacks as in nodes of dynamically updated graph.

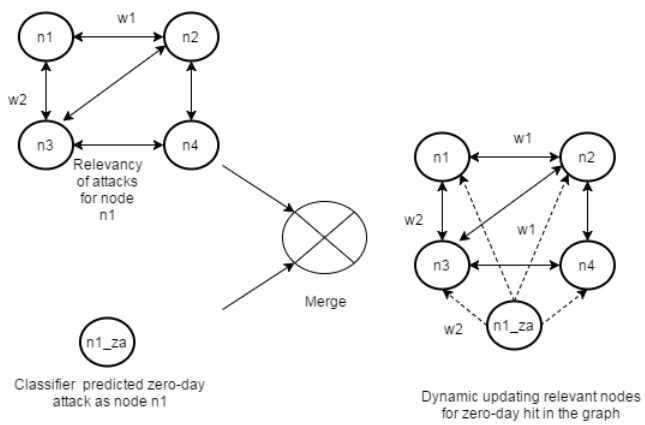

Figure 5. Dynamic Graph Updating.

In Figure 5, the classifier predicted the zero-day attack as node n1. So our algorithm traverses through semantic link network and add zero-day attack dynamically with relevancy scores equals to node n1. When a zero-day attack node n_{1za} is discovered it is added to the graph. The closed pre-existing node n_3 is also found based on similarity of the features. Then an edge w_2 is drawn from the n_{1za} to its most similar one n3, with the similarity score as its weight. This process is repeated for all nodes connected to the n3 that are dynamically

The above query summarizes how the graph is updated

dynamic_graph =
merge(left=test_stream,right=Node-Similarity-Matrix,
left_on='Classifier predicted node', right_on='Node1 of
Node-Similarity-Matrix')

Figure 6. Dynamic Graph generation query.

dynamically upon the hit of zero-day attack. Here the merge is happening between the two data frames test-stream with predicted label and Node-Similarity-Matrix. The query update the node of zero-day attack to the relevant nodes that are same as the predicted node. Hence the table dynamic-graph contains the newly updated nodes of zero-day attacks dynamically to study the relevancy nodes.

IV. Implementation and Validation

In this section we discuss the implementation of our proposed approach and study the results which were noticed. Most of our experiments have been carried out on the MAYA computing cluster [8], which is a community-based, interdisciplinary core facility for scientific computing and research on parallel algorithms at UMBC.

To evaluate our proposed approach we used metrics such as precision, accuracy, recall and ROC curve [12] in the following subsections.

Dataset and Source Description: For our experimentation we have used 10 percent KDD CUP 1999 dataset [15]. This dataset contains half a million data points. It contains 42 features including label feature which comprises of 22 attack types and a normal. To train and test the classifier we had split the dataset into 60 percent training and 40 percent training set. To identify the zero-day attacks we trained the classifier with 8 attacks out of 22 attack types. In test dataset we had marked 14 attacks that were removed from training dataset as zero-day attacks. The below table gives a list of the attacks that were marked as zero-day attack and non-zero-day attacks. For our experimental graphical representation we had given unique number to each attack as shown in Table 6.

Attack Number	Attack Name	Zero Day/ Non Zero-day
1	back	Non Zero-day
2	buffer-overflow	Non Zero-day
3	ftp-write	Zero-day
4	guess-passwd	Non Zero-day
5	imap	Zero-day
6	ipsweep	Zero-day
7	land	Zero-day
8	land-module	Zero-day
9	multihop	Zero-day
10	neptune	Non Zero-day
11	nmap	Zero-day
12	perl	Zero-day
13	phf	Zero-day
14	pod	Zero-day
15	portsweep	Zero-day
16	rootkit	Zero-day
17	satan	Non Zero-day
18	smurf	Non Zero-day
19	spy	Zero-day
20	teardrop	Zero-day
21	warez-client	Non Zero-day
22	warez-master	Non Zero-day

Table VI. Attack classification as zero day/non-Zero day

The total number of samples in training dataset are 107,369 and in testing dataset are 36,398. The testing set consists of data samples related to all the zero-day attacks.

Prediction of Zero-day Attacks and graph expansion: After creation of Node-Similarity-Matrix, we deployed a KNN classifier and set k value as 3. We split the whole dataset into two parts, training set and testing set in the ration of 3:2. After the creation of training set, we had removed the zero-day attacks which are mentioned in the above table 1. After training we had tested the classifier using test data that is streamed continuously using spark streaming context. Out of 14 zero-day attacks our classifier had predicted 10 zero-day attacks and for rest of them it predicted as benign. Classifier prediction and actual nodes are depicted in Table 7.

Zero-day Attack	Classifier Predicted Node
teardrop	satan
nmap	satan
portsweep	neptune
ftp-write	normal
rootkit	satan
ipsweep	neptune
land	neptune
land-module	buffer-overflow
pod	warez-client
Imap	normal
perl	normal
multi-hop	buffer-overflow
phf	normal

Table VII. Identification of Zero-day Attacks by KNN

Results: Table 8 gives a clear picture of total number of test instances, number of attack samples, number of normal samples.

Sample type	Number of instance
Test data	36398
Zero day attacks	5683
Non-Zero attacks	6665
Attack samples	12348
Normal samples	24050

Table VIII. Distribution of samples in test dataset

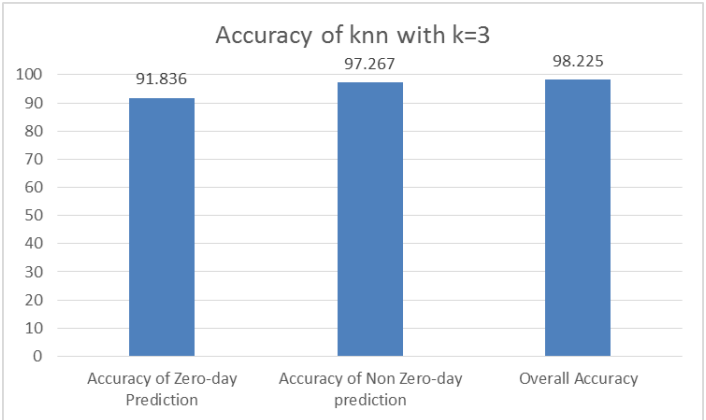

Figure 7. Accuracy of KNN Classifier.

Figure 7 depicts the variation of accuracy between the correctness in prediction of zero-day attacks, Non Zero-day attacks and overall prediction rate. Figure 8 depicts the performance of the classifier.

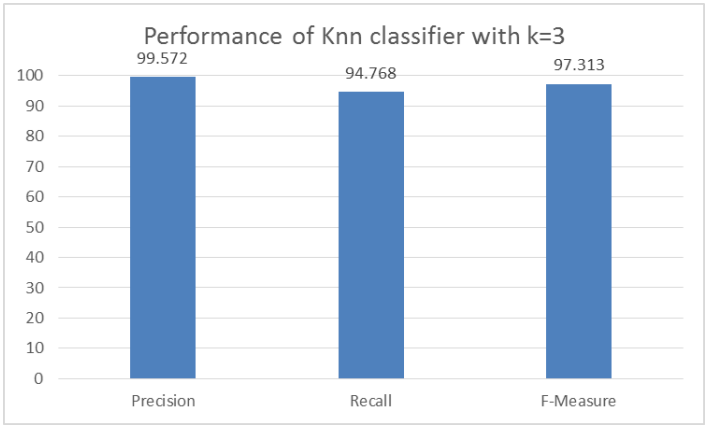

Figure 8. Classifier performance.

It is shown that the precision [24] is 99.572 percent, which means our system had predicted the normal samples correctly with almost 0 percent error rate. After plotting the accuracy and classifier performance we had calculated true positive rate and false positive rate for different values of k ranging from 0 to 3 and then we had plotted ROC curve to make sure that our system is stable in identification of zero day attacks as in Table 9.

K value	TPR	FPR
1	0.92	0.12
2	0.939	0.09
3	0.94768	0.03

Table IX. ROC METRICS

By using the above values in Table 9, we plotted the ROC curve as shown in Figure 9.

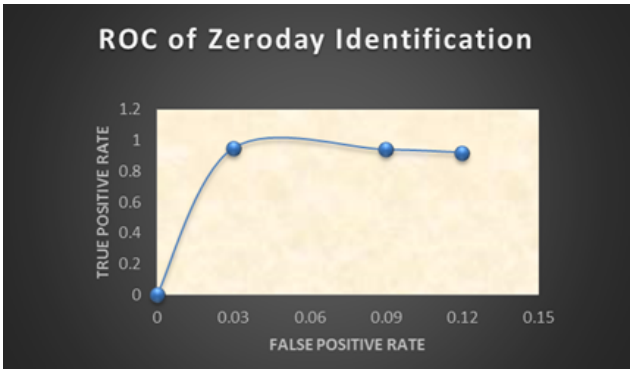

Figure 9. Experimental Results using ROC.

As shown in Figure 10, the classifier predicted the zero-day attack "tear drop" as a non zero day attack "Satan", so based on the similarity values of satan with different attacks, the semantic web was dynamically developed for the Zero-day attack.

We have also calculated the Program Execution Time by collecting the JSON logs and by changing the number of nodes in the cluster. Figure 11 shows details of the execution times and we found that if we increase the number of nodes in the cluster, as expected the program execution time decreases.

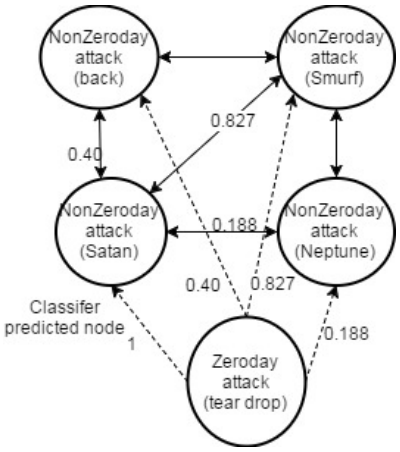

Figure 10. Dynamic graph update with zero day attack.

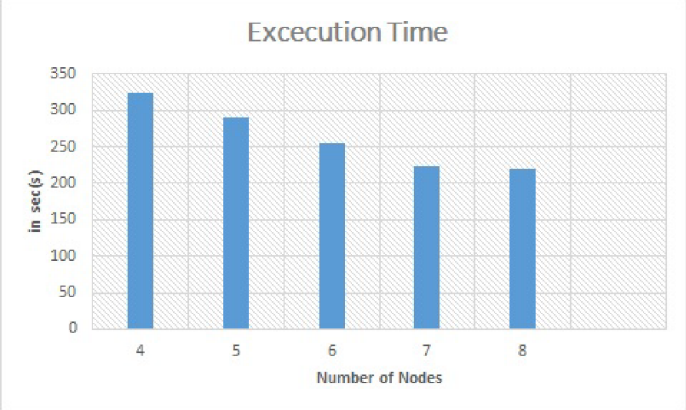

Figure 11. Average execution time with varying nodes.

We also calculated the throughput of our systems which means number of batched processed per min which tells us about the scalability of our system which is shown in Figure 11 and the throughput of our system in Figure 12.

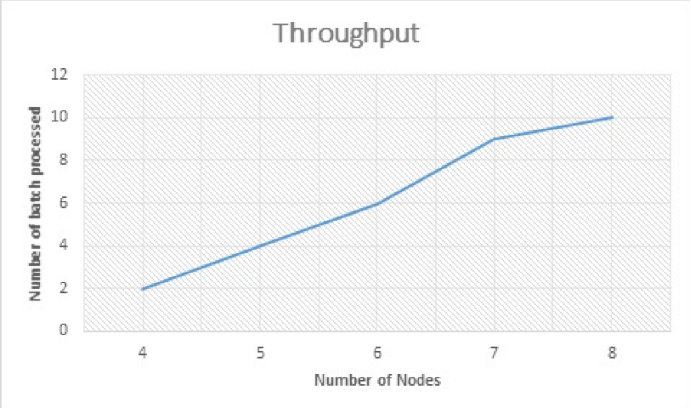

Figure 12. Throughput of the prototype system.

As we increase the number of nodes in our spark environment, the number of batch processed per minute increases gradually.

V. CONCLUSION AND FUTURE WORK

In this research, we addressed the problem of time complexity, handling large amount of data and parallel processing by implementing our whole approach in the Apache spark ecosystem. We used a flow-based approach and implemented a classifier that could identify zero-day attacks. We performed on the fly analytics and achieved near real-time predictions, by categorizing the incoming stream flows into either attack or benign activity without using any computer file system. We applied semantics on the training data to find the relevancy scores between the attacks. In our future work, instead of depending on the result of single classifier we would like to create a voting system between a series of classifier like KNN, SVM, J-48, Naïve Bayes and random forest.

REFERENCES

[1] Alfred V Aho, Brian W Kernighan, and Peter J Weinberger. *The AWK programming language*. Addison-Wesley Longman Publishing Co., Inc., 1987.

[2] Omar Al-Jarrah and Ahmad Arafat. Network intrusion detection system using neural network classification of attack behavior. *Journal of Advances in Information Technology Vol*, 6(1), 2015.

[3] Ahmed Aleroud and George Karabatis. Toward zero-day attack identification using linear data transformation techniques. In *Software Security and Reliability (SERE), 2013 IEEE 7th International Conference on*, pages 159–168. IEEE, 2013.

[4] Ahmed Aleroud and George Karabatis. Context infusion in semantic link networks to detect cyber-attacks: a flow-based detection approach. In *Semantic Computing (ICSC), 2014 IEEE International Conference on*, pages 175–182. IEEE, 2014.

[5] Ahmed Aleroud, George Karabatis, Prayank Sharma, and Peng He. Context and semantics for detection of cyber attacks. *International Journal of Information and Computer Security 7*, 6(1):63–92, 2014.

[6] Daniel Barbara, Ningning Wu, and Sushil Jajodia. Detecting novel network intrusions using bayes estimators. In *Proceedings of the 2001 SIAM International Conference on Data Mining*, pages 1–17. SIAM, 2001.

[7] Gerald Combs et al. Wireshark. *Web page: http://www. wireshark. org/last modified*, pages 12–02, 2007.

[8] Adam Cunningham, Gerald Payton, Jack Slettebak, and Jordi Wolfson-Pou. Pushing the limits of the maya cluster.

[9] Solane Duque and Mohd NIzam bin Omar. Using data mining algorithms for developing a model for intrusion detection system (ids). *Procedia Computer Science*, 61:46–51, 2015.

[10] Zyad Elkhadir, Khalid Chougdali, and Mohammed Benattou. Intrusion detection system using pca and kernel pca methods. In *Proceedings of the Mediterranean Conference on Information & Communication Technologies 2015*, pages 489–497. Springer, 2016.

[11] Pedro Garcia-Teodoro, J Diaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, 28(1):18–28, 2009.

[12] James A Hanley and Barbara J McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1):29–36, 1982.

[13] Peng He and George Karabatis. Using semantic networks to counter cyber threats. In *Intelligence and Security Informatics (ISI), 2012 IEEE International Conference on*, pages 184–184. IEEE, 2012.

[14] Gilbert R Hendry and Shanchieh J Yang. Intrusion signature creation via clustering anomalies. In *SPIE Defense and Security Symposium*, pages 69730C–69730C. International Society for Optics and Photonics, 2008.

[15] S Hettich and SD Bay. Kdd cup 1999 data. *The UCI KD Archive, Irvine, CA: University of California, Department of Information and Computer Science*, 1999.

[16] George Karabatis, Jianwu Wang, and Ahmed. AlEroud. Towards adaptive big data cyber-attack detection via semantic link networks. In *The first Workshop of Mission-Critical Big Data Analytics workshop (MCBDA 2016)*, 2016.

[17] SB Kotsiantis, D Kanellopoulos, and PE Pintelas. Data preprocessing for supervised leaning. *International Journal of Computer Science*, 1(2):111–117, 2006.

[18] Kingsly Leung and Christopher Leckie. Unsupervised anomaly detection in network intrusion detection using clusters. In *Proceedings of the Twenty-eighth Australasian conference on Computer Science-Volume 38*, pages 333–342. Australian Computer Society, Inc., 2005.

[19] Zhichun Li, Manan Sanghi, Yan Chen, Ming-Yang Kao, and Brian Chavez. Hamsa: Fast signature generation for zero-day polymorphic worms with provable attack resilience. In *Security and Privacy, 2006 IEEE Symposium on*, pages 15–pp. IEEE, 2006.

[20] Glenn K Lockwood. Conceptual overview of map-reduce and hadoop. *Retrieved March*, 13:2016, 2015.

[21] Hanchuan Peng, Fuhui Long, and Chris Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on pattern analysis and machine intelligence*, 27(8):1226–1238, 2005.

[22] Leonid Portnoy, Eleazar Eskin, and Sal Stolfo. Intrusion detection with unlabeled data using clustering. In *In Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001*. Citeseer, 2001.

[23] Manas Srivastava, C Sabarinathan, Rishi Sankineni, and TM Manoj. Mining of big data using map-reduce theorem. *IOSR Journals (IOSR Journal of Computer Engineering)*, 1(17):49–55.

[24] Kai Ming Ting. Precision and recall. In *Encyclopedia of machine learning*, pages 781–781. Springer, 2011.

[25] Giovanni Vigna, William Robertson, and Davide Balzarotti. Testing network-based intrusion detection signatures using mutant exploits. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 21–30. ACM, 2004.

[26] Guohua Wu and Junjun Xu. Optimized approach of feature selection based on information gain. In *Computer Science and Mechanical Automation (CSMA), 2015 International Conference on*, pages 157–161. IEEE, 2015.

[27] Hui Yu, Patrick PK Chan, Wing WY Ng, and Daniel S Yeung. Apply randomization in knn to make the adversary harder to attack the classifier. In *Machine Learning and Cybernetics (ICMLC), 2010 International Conference on*, volume 1, pages 179–183. IEEE, 2010.

[28] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

[29] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.

[30] Matei Zaharia, Tathagata Das, Haoyuan Li, Scott Shenker, and Ion Stoica. Discretized streams: An efficient and fault-tolerant model for stream processing on large clusters. *HotCloud*, 12:10–10, 2012.