IoT Device Anomaly Detection using Provenance Graphs

Ebelechukwu Nwafor
Howard University
Washington, DC 20059
ebelechukwu.nwafor@bison.howard.edu

Gedare Bloom
Howard University
Washington, DC 20059
gedare.bloom@howard.edu

ABSTRACT

The Internet of Things (IoT) has revolutionized the way we interact with devices. Unfortunately, this technological advancement has been met with privacy and security challenges. One major concern is the notion of malicious intrusions. A common way of detecting a malicious intrusion is by treating an attack as an anomaly and using anomaly detection techniques to pinpoint the source of an intrusion. In a given IoT device, provenance graphs which denotes causality between system events offers immense benefit for intrusion detection. Provenance provides a comprehensive history of activities performed on a system which indirectly ensures device trust. Given a provenance graph, how can we determine if an anomalous activity has occurred? This paper seeks to address this issue. In this paper, we propose two lightweight approach to intrusion detection on IoT devices. The first is a whitelist approach which involves matching patterns of known normal provenance graph. The provenance graph exhibiting normal execution of IoT applications are stored in a knowledge repository. Incoming provenance graphs of similar applications are compared with provenance graph in the knowledge repository to determine if they exhibit anomalous behaviors. The second approach involves the use of a similarity metric to compare provenance graphs from the learning and detection phase. We evaluate the performance of our approach by comparing provenance graphs generated from sample IoT applications and determine the number of false positive, and false negative rates.

CCS CONCEPTS

•Computer systems organization →Embedded systems; •Security and privacy →*Anomaly detection systems;*

KEYWORDS

Internet of Things, Provenance Graphs, Anomaly Detection

ACM Reference format:
Ebelechukwu Nwafor and Gedare Bloom. 2018. IoT Device Anomaly Detection using Provenance Graphs. In *Proceedings of ACM IoTDI Conference, Orlando, Florida USA, July 2018 (IoTDI'18),* 6 pages.
DOI: 10.475/123_4

1 INTRODUCTION

Over the years, IoT devices have become an essential part of our lives, from wearable devices to larger industrial devices such as power plants and smart grids systems. These devices offers immense benefits to consumers by interacting with our physical enironment through sensors and actuators which allows device automation therby improving efficiency. Unfortunately, the prolifiration of IoT devices has led to an increase in the number of intrusions per device which introduces new attack vectors that could have disastrous financial and physical implications. [2]. In 2010, a sophisticated self-replicating virus, stuxnet was discovered which targeted industrial control systems. This virus had the potential to destroy the execution of power plants. In 2015, security researchers demonstrated a vulnerability on the jeep vehicles which allowed remote control of the digital system over the internet. This was achieved this by sending messages through the vehicles internal network, Controller Area Network. This exploit allowed control of components such as the breaks, steering and lights over the internet. Additionally, Researchers at HP [10] reviewed 10 popular IoT devices in which they discovered that 7 did not contain proper encryption for communication, 8 lacked proper strong passwords and 6 raised security concerns with the user interface. One way of detecting intrusions is by the use of anomaly detection techniques. An anomaly, also referred to as an outlier, is defined as data which deviates from the normal system behavior. This enables the detection of malicious attacks. An anomaly could indicate a system fault or that a system has been compromised by a malicious event.

Due to the sensitive nature of safety critical systems, detecting current and future malicious attacks is of utmost importance. Provenance offers a solution that can be used to address this issue. Provenance graph captures a history of transformations that occurred on a data object during its lifecycle. It offers an efficient way to represent relationships that exist between multiple data objects. This in turn can be used to detect system faults or anomalous system behavior such as malicious intrusions. In this paper, we motivate the need for IoT device anomaly detection. Unlike most anomaly detection technique which utilize system call frequencies to detect anomalous system events, we detect anomalies by leveraging the provenance graph of information flow in an IoT application. We identify how provenance graphs can be used to detect anomalous data instances. We propose two lightweight anomaly detection algorithms for identifying anomalous data instances. We evaluate the effectiveness of our approach with dataset from sample IoT application(s) by comparing the false positive, false negative and true positive rate of the proposed approach. Detailed results are presented in sections IV. In summary, our technical contributions are outlined in detail as follows:

- We introduce two anomaly detection algorithms for detecting malicious instances in an IoT device using provenance graphs. The first algorithm involves the use of a whitelist approach in which provenance graphs exhibiting normal application behavior are stored in a knowledge repository.

Incoming provenance graphs are compared with provenance graphs in the knowledge repository to determine if they are normal or anomalous. The second approach involves the use of a similarity measure to compare the provenance graphs from the learning and detection phase. The result of the algorithm generates an anomaly score. A threshold is set which classifies provenance in the detection phase as either anomalous or normal.
- We evaluate the effectiveness of our approach to graph based anomaly detection with a dataset from an IoT application by comparing the false positive, false negative and true positive rate of our proposed approach. Detailed results are presented in sections IV.

The remaining portion of this paper is organized as follows. In section 2, we discuss background information on anomaly detection, and provenance graphs. In Section 3, we discuss our approach to anomaly detection by defining the two anomaly detection algorithms. In section 4, we describes our experiment design and evaluation. In section 5, we summarize and conclude.

2 BACKGROUND

2.1 IoT Provenance producer - Consumer Model

2.2 Anomaly Detection

The notion of what consitutes an anomaly is often ambigous and domain specific. Hawkins, defines an anomaly as an "observation which deviates so much from the other observations as to arouse that it was generated by a different mechanism" [7]. We define an anomaly as trends that deviate from the expected normal behavior. Normal behavior can be seen as the average behavior that is depicted where most data points are centered in while an anomaly is in isolation from other data points. Anomaly detection has been researched in a wide variety of fields such as statistics, machine learning, and information theory. It has applications in finance for fraud detection [6], in health care for the monitoring of patient care [18], and in computer security for intrusion detection [11, 19]. An anomaly often indicates the presence of a malicious entity or a system fault. For example, an anomaly could be a sudden increase in web traffic of a web server which could be indicative of a denial of service attack. Additionally, in a critical care health device such as a pacemaker, an anomalous event could be detrimental to the health of a patient which could result in the loss of life.

Anomaly detection involves the use of statistical or machine learning techniques such as clustering and classification to determine normal or anomalous data instances. Some methods deal with assigning a score to determine the anomaly. Details on anomaly detection techniques are outlined below

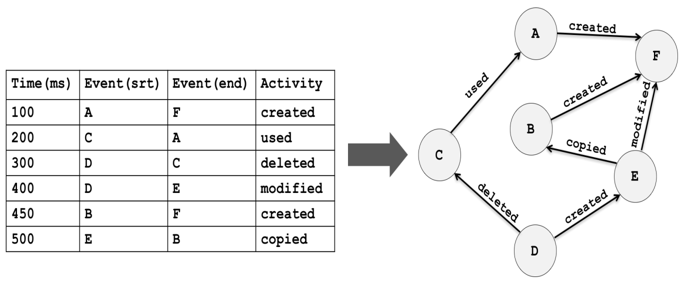
(1) Statistical-based approach: This approach involves the use of parametric or non parametric statistical inference to develop models which are used to determine if a dataset fits a statistical model. Instances that do not fit the defined statistical model are classified as an anomaly.
(2) Classification: The main idea in this approach involves building models which use training data set with predefined labels (i.e normal, anomalous) to classify incoming data. Classification works in two phase: training phase and observation phase. In the training phase, data is collected which contains labels of normal and anomalous system behavior. If the dataset only contains a label of either anomalous or normal behavior, this is considered as a one class classifier. In the observation phase, incoming data is classified by defined data labels.
    - Nearest-Neighbor: It is based on the assumption that normal data occurs in dense neighborhoods and abnormal data in sparse neighborhoods. The main idea is to assign an incoming observation data to a class based on its proximity to the closest data point in the training data set. A distance or similarity measure is used to quantify the distance between points in a dataset. A popular form of nearest neighbor technique is the k-nearest neighbor which groups incoming data based on proximity to k closest data point. Details on k- nearest neighbors is discussed in section 3.5.
(3) Clustering: The main idea is to group similar data instances into clusters. There are various approach to clustering. One approach looks at the density of the clusters, normal data belongs to large dense clusters while abnormal data belongs to small clusters. Another approach as treats clustering as one class which assumes that normal belongs to a cluster and abnormal data does not. Another approach looks at the distance of the data to the centroid. Centroids are seen as the center of the cluster. Normal data are considered to be closer to the centroid than anomalies lie.
    - Density: This approach is used to estimate the density of k nearest neighbors. A data instance in a dense neighborhood is considered normal while data instances in neighborhoods with a sparse density are considered anomalous. The distance from a data instance to a nearest neighbor is seen as the inverse of the density of data instances. This approach faces an issue in which the density approach performs poorly in regions of varying densities. Local outlier factor addresses this issue. Local outlier factor is a measure of the degree of Outlieriness of each data instance contained in a data set. It is achieved by comparing the ratio of local density of k nearest neighbors to the density of a data instance. Data instances with lower density are considered outliers.

Detailed information on anomaly detection techniques can be seen in [1, 5, 8, 22]. The process of determining all anomalous instances in a given dataset or system is a complex task. A major challenge in anomaly detection is providing the right set of feature from a dataset to use for detection. Another challenge exists in defining what constitutes as normal system behavior. There often exist a thin line between what is considered normal system behavior and what is considered an anomaly. Most of the research done on anomaly detection are centered on the detection of anomalous behavior using point based dataset. These techniques often fail to include the dependencies that exist between data points. Graphs offers a means of modeling complex structures such as social networks, computer networks, biological DNA sequences.

2.3 Provenance Graphs

Provenance denotes the origin of an object and all activities that occurred on that object. An example of provenance can be seen with a college transcript. A transcript is the provenance of a college degree because it outlines all of the courses needed to be satisfied in order to attain the degree. In the field of computing, data provenance, also known as data lineage, can be defined as the history of all activities performed on a data object from its creation to its current state. Provenance ensures data trust [3]. It establishes causality and dependency between all objects involved in a system and allows for the verification of a data source. In an IoT device, a provenance graph $G = (V, E)$ is a directed acyclic graph (DAG) in which vertices represent device or sensor events data and the edges correspond to the interaction between them.

Graphs provide a means of representing complex relationships that exsist between data objects. Provenance data is represented using a directed acyclic graph (DAG) where nodes denote data objects and the edges represent relationships between data objects. For example, provenance graph from a device can be generated by evaluating the log of system calls. The log information might contain noisy data which is further streamlined by mapping log data to a provenance model. With this information, we are able to build a workflow of device information flow. There has been numerous provenance collection systems developed to track provenance in a computing device most of which deals with tracking system call events [14, 17, 21]. Provenance graph used for experimentation is generated from PAIoT, a provenance-aware framework which tacks the information flow of sensor data event generated in an IoT device over its lifecycle (i.e till the data is evaluated in a decision).



Time(ms)	Event(srt)	Event(end)	Activity
100	A	F	created
200	C	A	used
300	D	C	deleted
400	D	E	modified
450	B	F	created
500	E	B	copied

Figure 1: Provenace data transcribed to Provenance graph which depicts causal dependency between system events. The nodes represents events while the edges represents activities

2.4 Similarity Metric

Similarity metric measures how identical two objects are. It compares some form of distance between data objects by either measuring the angle between the objects (Cosine similarity) or a linear distance between the objects (Euclidean distance). Similarity measures are widely used in document retrieval for selecting a query given a list of documents. The similarity measure used is dependent. There are three well known similarity measure for evaluating data objects. These measures are outlined below:

2.4.1 Cosine similarity. This is a measure of orientation between the two non-zero vectors. It measures the cosine of the angle between the vectors. Two vectors which are at an angle of 90°have a similarity of 0 while two vectors which are similar (with an angle of 0°) have a cosine of 1 and two vectors which are completely opposite (with an angle of 180°) have a similarity of -1. Since we are concerned with the similarity of the vectors, we are only concerned with the positive values bounded in [0,1]. To compute the cosine similarity between two vectors, X and Y, cosine similarity is represented by using the dot product and magnitude of the two vectors.

$$\cos(\theta) = \frac{X \cdot Y}{\|\mathbf{X}\| \cdot \|\mathbf{Y}\|} = \frac{\sum_{i=1}^{n} X_i Y_i}{\sqrt{\sum_{i=1}^{n} X_i^2}\sqrt{\sum_{i=1}^{n} Y_i^2}}$$

2.4.2 Jaccard Similarity. This similarity measure evaluates the intersection divided by the union of two non zero vectors.

$$J(X, Y) = \frac{|X \cap Y|}{|X \cup Y|}$$

2.4.3 Euclidean distance. This measures calculates the line distance between two data objects in an euclidean space. The euclidean distance between vectors X and Y , $d(X, Y)$ is defined by:

$$d(X, Y) = \sqrt{\sum_{i=1}^{n} (X_i - Y_i)^2}$$

3 THREAT MODEL AND ASSUMPTIONS

Due to the ubiquitous nature of IoT devices, there are a wide array of potential vulnerabilities associated with them. It is believed that each application exhibits a unique sequence which can be represented as the provenance graph of the normal application's data flow. In designing our anomaly detection scheme, we assume an attacker's footprint is reflected through the data flow as depicted in the provenance graph. Our algorithm detects attacks such as false data injection, modifications to the application code and the compromise to the integrity of data that exploits normal information flow of events. We do not guaranty security due to hardware tampering and our approach does not detect eavesdropping attacks. Our approach also cannot detect intrusion that are not represented in the provenance graph such as system call hacks, memory leaks, and network anomaly detection.

A vector is a dimensionality reduction of provenance graphs

4 WHITELIST APPROACH

The whitelist approach consist of three phase: the data collection phase, the knowledge generation phase and the detection phase. The data collection phase involves generating provenance graphs. Provenance graphs are generated at normal program execution. The knowledge generation phase involves generating a repository of provenance graphs of various IoT applications with normal executions, free from malicious activities. The knowledge repository contains an exhaustive list of known provenance graphs for various IoT applications. Each, IoT Application exhibits a unique data flow path which is represented in their provenance graph. The knowledge repository is updated by knowledge experts pertaining to the

IoT application. Finally, in the detection phase, the application's provenance graph being observed is compared to the provenance graph contained in the knowledge repository. If there is a direct match, the running application is considered to be running as expected, otherwise an alert such as a warning is generated and the application is flagged for possible intrusion.

5 GRAPH SIMILARITY APPROACH

This approach consists of two phase: learning phase also known as the training phase and the test also known as the detection phase. In the training phase, the system collects provenance data considered to be a representation of the system's normal daily activity and free from malicious events. Once learning data been collected, the system's activities are further observed. This phase is known as the testing phase or detection phase. In the test phase, observed provenance graph is compared to the learning phase to determine if an anomaly exists between the two.

5.1 Graph to Vector Space Conversion

5.1.1 Feature Extraction. Similarity function is a measure of how identical provenance graphs from the learned and observed graph are in a vector space. In order to apply a similarity function between provenance graphs, we compute a vector representation of our provenance graphs. In this approach, provenance graphs undergoes dimensionality reduction into vector space and this representation is used as an input to our anomaly detection algorithm. Selecting features from provenance graphs is an important task. We need to select features which preserves the order of the graphs. Our approach utilizes the frequency of unique edges contained in both graphs. We achieve this by identifying identical edges contained in both graphs. To determine if an edge contained in both graphs are identical, we utilize a combination of the edge label and the type of node in which the event originates from. For example, an event originating from a temperature sensor might generate a similar provenance graph to an event originating from a pressure sensor, however both edges are not considered similar since they originate from different sensors.

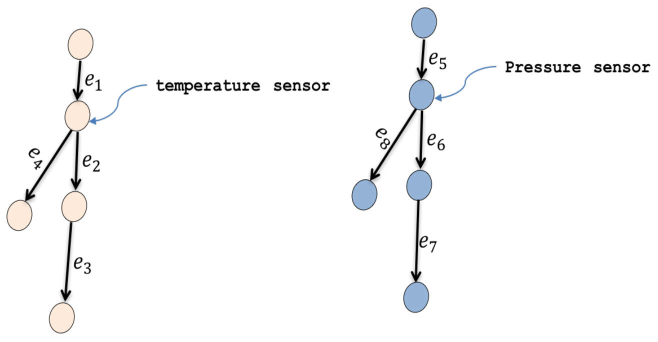


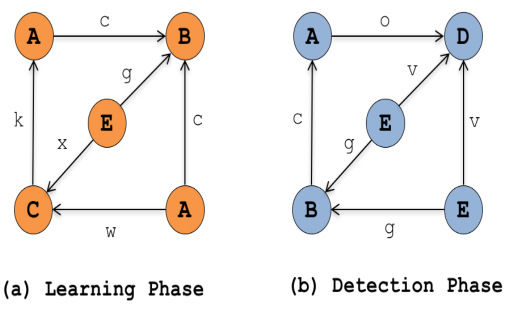
Figure 2: Edge identification between two graphs. Identical events from Different sensor nodes corresponds to different edges

Given a set of provenance graphs $P = \{p_1, ...p_n\}$, where $p_x = (V, E)$. P consists of graphs both in the learning and detection phase. We denote the occurrence of edges and nodes contained in set P. Each graph in P is represented as a vector by using our graph dimensionality reduction approach which preserves the order of edges and allows updates of edges. We formally define our approach as follows:

Definition 5.1. Let $P = \{p_1, ..., p_n\}$ where $p_i = (V_i, E_i)$ where $V_x = n_x, ...n_k$. n_s is the sensor node, the vector space representation of p_i, v_i is the number of times each edges, E_i contained in P appears in the graph , p_i if $n_s \in V_x$ and $n_s \in V_y$.

$$\boldsymbol{v_x} = (freq(E_i, p_i))$$

where $freq$ denotes the occurrence of E_i in graph p_i



edge	v_x	v_y
c	2	1
g	1	1
k	1	1
o	0	1
v	0	2
w	1	0

Figure 3: Provenance graphs generated from the learning and detection phase

For example, Figure 3 displays provenance graphs generated in the Learning and detection phase respectively. Both graphs, $\boldsymbol{p_1}$, and $\boldsymbol{p_2}$ consists of vertices A, B, C, D, and E and edges $c, g, k, o, v, andw$. The vector representation of the two graphs, v_1, and v_2 is the occurrence of unique edges contained in both graphs.

5.2 Graph Based Similarity Detection Algorithm

The method of comparing the similarity of provenance graphs based on a similarity metric was inspired by a document retrieval technique. Given a corpus $D = \{d_1, ..., d_n\}$, and query, q, How do we find document(s) $d_x, ...d_y$ which are similar to q and rank them by order of importance. To achieve this, documents are converted into a vector space representation which allows document to be ranked based on some similarity metric. Figure 4 depicts the overall goal of the similarity approach in detecting anomalies.

Given v_x, v_y which denotes the vector representation of provenance graphs p_x, p_y. The similarity of p_x, p_y is found by calculating the cosine similarity between the two vectors where 1 denotes similarity between the two vectors and 0 denotes non-similarity between the two vectors. A threshold value is set which is used to classify the behavior of provenance graphs in the detection phase as normal or an anomaly.

$$sim(v_x, v_y) = \frac{v_x \cdot v_y}{\|\mathbf{v_x}\| \cdot \|\mathbf{v_y}\|} = \frac{\sum_{i=1}^{n} v_{xi} v_{yi}}{\sqrt{\sum_{i=1}^{n} v_{xi}^2} \sqrt{\sum_{i=1}^{n} v_{yi}^2}} \in [0, 1]$$

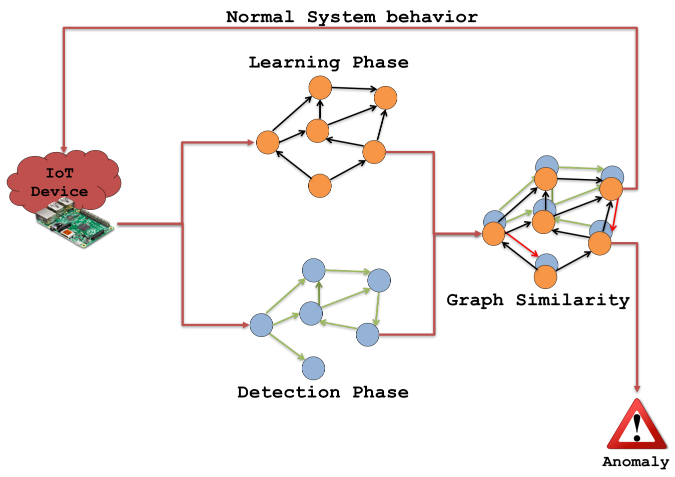


Figure 4: Graph Similarity Approach

Figure 5: Graph based anomaly detection algorithm

```
1: procedure GraphAnomaly($G_x, G_y$)
2:     $v_x \leftarrow GraphtoVector(G_x)$          ▷ returns a vector of $G_x$
3:     $v_y \leftarrow GraphtoVector(G_y)$          ▷ returns a vector of $G_x$
4:     $z \leftarrow sim(v_x, v_y)$                 ▷ similarity function
5:     if z < threshold then
6:         classify as normal
7:     else
8:         classify as anomaly
9:     return $z$                                   ▷ anomaly score
```

Algorithm 1 depicts a formal definition of the graph-based similarity algorithm.

5.2.1 Defining Anomaly Threshold. An anomaly threshold t is a score that defines at what point a provenance graph contained in the test data is considered anomalous. Ensuring a proper threshold score is used for detection is an important task that requires extensive knowledge of the attack domain. The threshold is manually set to a value t, which is defined by domain experts. For automatic anomaly threshold detection, one can use prediction methods to define the anomaly score. Prediction techniques are beyond the scope of this research.

6 EXPERIMENTAL EVALUATION

This section outlines experimental procedures involved in evaluating our proposed algorithms.

6.1 Evaluation

In order to test the effectiveness of our proposed approach, evaluation was performed on Raspberry Pi 3 with 1GB of memory running ubuntu mate (Linux v4.9). Provenance graph were generated using an IoT application which simulates a Heating Ventilation, AC (HVAC) system. For our implementation, we utilized Adafruit DH22 temperature and humidity sensor connected to the Raspberry Pi 3 via GPIO. To build our knowledge repository, we ran our IoT application executing in a normal environment and the resultant provenance graph was stored in Neo4j, a graph database for efficient graph processing. To generate our malicious attack dataset, we utilized simulated data which represents provenance graphs in an event of an attack. The provenance graphs generated were agreed upon by domain experts.

We evaluate the performance of our proposed algorithms by calculating five performance metrics: false positive, true positive and false negative, precision and recall rate. False positive indicates when an intrusion that does not exist is detected, false negative indicates when the IDS classifies a provenance graph as normal when it is malicious, finally for true positive an IDS classifies a provenance graph as anomalous which is actually an intrusion. precision and recall rate are calculated as follows:

$$Precision = \frac{tp}{tp + fp}$$

$$Recall = \frac{tp}{tp + fn}$$

7 RELATED WORK

There has been a considerable amount of research done on anomaly detection. Most of the work involves the analysis of system call sequence. Liao et al [12] characterizes a system's normal behavior by denoting the frequency of unique system calls which are converted into a vector space representation. A classification algorithm such as k-NN is used to classify training data set. Our approach also deals with system events by converting a provenance graph to a vector representation. We use a more sophisticated clustering algorithm for our detection phase. Stephanie forest's group [9] defines a link between the human immune system and intrusion detection systems. They develop a normal system behavior repository by analyzing system call sequences of an application. The application's system call sequence is stored in a normal database which is queried during observation in which all behaviors are judged. If an application executes a sequence of system calls that are not found in the normal database, a mismatch is recorded. If the mismatch for that application exceeds a threshold, an anomaly is detected. Yoon et al. [20] developed a technique for intrusion detection in embedded systems by analyzing system call frequencies. This is achieved by learning normal system profile of observed patterns in the system call frequency distribution.They hypothesize that applications follow a known frequency pattern which is centered around the centroid. . Data from the training set is clustered using k-means which groups legitimate system behavior. Observation at run-time are compared with the clusters in the detection phase, if the incoming observation does not fit into a cluster, it is considered an anomaly.

Additionally, anomaly detection on graphs has also been explored. Manzoor et al [13] proposed a centroid based clustering anomaly detection for instances of streaming heterogeneous graphs in real time. This algorithm is able to accommodate incoming edges in real time. They propose a method of comparing similarity between heterogeneous graphs by comparing the similarity of two graphs by their relative frequency. Each graphs is represented as a vector known as shingles. Since all of the graph is stored in memory, they also accommodate an efficient representation of the shingles in memory in what is known as streamhash. Papadimitriou et al [16] proposed five similarity algorithms namely Signature similarity, vertex/edge vector similarity, vertex ranking, vertex edge overlap, for comparing the similarity of web graphs for the detection of anomalies inspired by document similarity method namely shingling and random projection based method. Nodes represents a webpage. An anomaly could be a missing link (edge) or a web node. Out of the five similarity measures proposed, Signature similarity which compares two graphs based on a set of features (signatures) using a scheme known as *simHash* performed the best followed by vector similarity. By comparing instances between snapshot of crawled webpages, this enables the detection of inconsistencies in crawled web content. Noble et al [15] proposed two algorithms for anomaly detection in graphs The first approach, looks at unusual substructures in graphs. This is achieved by inverting the subdue score of patterns that occurs frequently in a graph. Subdue is a method for detecting substructures within graphs. The values that produce high scores are flagged as anomalies. The second approach examines unusual subgraphs contained in a graph by iteratively using subdue algorithm to compress the graph. The idea behind this method is that subgraphs containing many common substructures are considered less anomalous that one with less substructures. Some graph approach involve the use of a community-based approach in which dense regions of connected nodes are considered normal and nodes with high sparse regions which do not belong to any community are considered anomalous. *AUTOPART* [4] consist of nodes with similar neighbors are clustered together and the edges which do not belong to any cluster is considered as an anomaly. To find communities it achieves this task by reorganizing the rows and the columns of the adjacency list. Our approach of graphs similarity is similar to graph kernels and graph edit distance. graph kernels involves measuring the similarity between two graphs, graph edit distance looks at the number of operations required for a graph G_1 to be identical to G_2.

8 DISCUSSION

TODO

9 SUMMARY AND CONCLUSION

In this paper, we proposed and evaluate two classes of anomaly detection algorithms for the detection of malicious intrusion on IoT devices. Our approach is lightweight and efficient in detecting anomalies that exists using provenance graphs. We evaluate the functionality of our approach through implementation and comparison of the various algorithms described.

10 ACKNOWLEDGMENT

This research has been supported in part by US National Science Foundation (CNS grant No. 1646317), and by Leidos. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSF or Leidos.

REFERENCES

[1] Leman Akoglu, Hanghang Tong, and Danai Koutra. 2015. Graph based anomaly detection and description: a survey. *Data Mining and Knowledge Discovery* 29, 3 (01 May 2015), 626–688. DOI:http://dx.doi.org/10.1007/s10618-014-0365-y

[2] Mario Ballano Barcena and Candid Wueest. Insecurity in the Internet of Things. (????).

[3] Elisa Bertino. 2015. *Data Trustworthiness—Approaches and Research Challenges.* Springer International Publishing, Cham, 17–25. DOI:http://dx.doi.org/10.1007/978-3-319-17016-9_2

[4] Deepayan Chakrabarti. 2004. Autopart: Parameter-free graph partitioning and outlier detection. In *European Conference on Principles of Data Mining and Knowledge Discovery.* Springer, 112–124.

[5] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly Detection: A Survey. *ACM Comput. Surv.* 41, 3 (July 2009), 15:1–15:58. DOI:http://dx.doi.org/10.1145/1541880.1541882

[6] Sushmito Ghosh and Douglas L Reilly. 1994. Credit card fraud detection with a neural-network. In *System Sciences, 1994. Proceedings of the Twenty-Seventh Hawaii International Conference on*, Vol. 3. IEEE, 621–630.

[7] D. M. Hawkins. 1980. *Identification of outliers.* Chapman and Hall, London [u.a.]. http://gso.gbv.de/DB=2.1/CMD?ACT=SRCHA&SRT=YOP&IKT=1016&TRM=ppn+02435757X&sourceid=fbw_bibsonomy

[8] Victoria Hodge and Jim Austin. 2004. A Survey of Outlier Detection Methodologies. *Artificial Intelligence Review* 22, 2 (Oct. 2004), 85–126. DOI:http://dx.doi.org/10.1023/B:AIRE.0000045502.10941.a9

[9] Steven A. Hofmeyr, Stephanie Forrest, and Anil Somayaji. 1998. Intrusion Detection Using Sequences of System Calls. *J. Comput. Secur.* 6, 3 (Aug. 1998), 151–180. http://dl.acm.org/citation.cfm?id=1298081.1298084

[10] HP. Internet of things research study. (????). http://h20195.www2.hpe.com/V4/getpdf.aspx/4aa5-4759enn

[11] Terran Lane, Carla E Brodley, and others. 1997. Sequence matching and learning in anomaly detection for computer security. In *AAAI Workshop: AI Approaches to Fraud Detection and Risk Management.* 43–49.

[12] Yihua Liao and V. Rao Vemuri. 2002. Using Text Categorization Techniques for Intrusion Detection. In *Proceedings of the 11th USENIX Security Symposium.* USENIX Association, Berkeley, CA, USA, 51–59. http://dl.acm.org/citation.cfm?id=647253.720290

[13] Emaad Manzoor, Sadegh M. Milajerdi, and Leman Akoglu. 2016. Fast Memory-efficient Anomaly Detection in Streaming Heterogeneous Graphs. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16).* ACM, New York, NY, USA, 1035–1044. DOI:http://dx.doi.org/10.1145/2939672.2939783

[14] Kiran-Kumar Muniswamy-Reddy, David A. Holland, Uri Braun, and Margo Seltzer. 2006. Provenance-aware Storage Systems. In *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference (ATEC '06).* USENIX Association, Berkeley, CA, USA, 4–4. http://dl.acm.org/citation.cfm?id=1267359.1267363

[15] Caleb C. Noble and Diane J. Cook. 2003. Graph-based Anomaly Detection. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '03).* ACM, New York, NY, USA, 631–636. DOI:http://dx.doi.org/10.1145/956750.956831

[16] Panagiotis Papadimitriou, Ali Dasdan, and Hector Garcia-Molina. 2010. Web graph similarity for anomaly detection. *Journal of Internet Services and Applications* 1, 1 (01 May 2010), 19–30. DOI:http://dx.doi.org/10.1007/s13174-010-0003-x

[17] Thomas Pasquier, Xueyuan Han, Mark Goldstein, Thomas Moyer, David Eyers, Margo Seltzer, and Jean Bacon. 2017. Practical Whole-System Provenance Capture. In *Symposium on Cloud Computing (SoCC'17).* ACM, ACM.

[18] Michal Valko, Gregory Cooper, Amy Seybert, Shyam Visweswaran, Melissa Saul, and Milos Hauskrecht. 2008. Conditional anomaly detection methods for patient–management alert systems. In *Proceedings of the... International Conference on Machine Learning. International Conference on Machine Learning*, Vol. 2008. NIH Public Access.

[19] Christina Warrender, Stephanie Forrest, and Barak Pearlmutter. 1999. Detecting intrusions using system calls: Alternative data models. In *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on.* IEEE, 133–145.

[20] Man-Ki Yoon, Sibin Mohan, Jaesik Choi, Mihai Christodorescu, and Lui Sha. 2017. Learning Execution Contexts from System Call Distribution for Anomaly Detection in Smart Embedded System. In *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation (IoTDI '17).* ACM, New York, NY, USA, 191–196. DOI:http://dx.doi.org/10.1145/3054977.3054999

[21] Robert H'obbes' Zakon (Ed.). 2012. *28th Annual Computer Security Applications Conference, ACSAC 2012, Orlando, FL, USA, 3-7 December 2012.* ACM. http://dl.acm.org/citation.cfm?id=2420950

[22] Y. Zhang, N. Meratnia, and P. Havinga. 2010. Outlier Detection Techniques for Wireless Sensor Networks: A Survey. *IEEE Communications Surveys Tutorials* 12, 2 (2010), 159–170. DOI:http://dx.doi.org/10.1109/SURV.2010.021510.00088