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 from smart grids, to healthcare and home automation systems. Unfortunately, this technological advancement has been met with privacy and security challenges. One major challenge is the notion of malicious intrusions. A common way of detecting a malicious attack 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 anomaly detection. Provenance provides a comprehensive history of activities performed on a system which indirectly ensures trust. Given a provenance graph, how do we determine if anomalous activities exist? 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 known normal provenance graph of IoT applications. The normal provenance graph is stored in a whitelist repository. Incoming provenance graphs from the detection phase of similar applications are compared with known application provenance graphs in the whitelist repository to determine if they exhibit anomalous behaviors. The other 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.

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, there has been an increase the number of connected devices. Cisco projects over 50 million devices will be connected to the internet by 2020 [6] . This increase has led to an increase in the number of intrusions on IoT devices [2]. Researchers at HP [11] reviewed 10 popular IoT devices. The result showed 7 did not contain proper encryption for communication, 8 lacked proper strong passwords and 6 raised security concerns with the user interface. With the pervasive nature and heterogeneity of IoT devices, the complexity of intrusions are further exacerbated. Malicious intrusions on IoT devices could have disastrous financial consequences. For example, a vulnerability on a consumer device such as a smart watch or mobile card reader could lead to the theft of sensitive financial and personal information.

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 that deviates from the normal system behavior. This enables the detection of known and unknown malicious attacks. Anomaly detection has applications in domains such as intrusion detection, fraud detection, medical health devices, sensor fault detection, and web spam. An anomalous event could indicate that a system fault exists. It could also indicate that a system is being used as a botnet in a distributed denial of service attack (DDOS). For the purpose of this paper, we focus on anomaly detection on memory constrained IoT devices. Due to the sensitive nature of safety critical systems, detecting current and future malicious attacks is of utmost importance to the security of IoT devices. Provenance can be used to address this issue. Provenance graph captures a holistic history of system events and also offer an efficient way of representing relationships between multiple data objects which can be used to detect system faults or anomalous system behaviors. In an IoT enabled smart home, provenance can be used to detect a point of intrusion in an event of a system hack.

In this paper, we motivate the need for device anomaly detection on memory constrained IoT devices. Unlike most anomaly detection system which utilize system call frequencies to detect anomalous system behaviors, We take a different approach to anomaly detection by detecting anomalous system events in IoT devices based on provenance data generated by these devices. We identify how provenance graphs can be used to detect anomalous data instances. We propose two lightweight intrusion detection algorithms for the detection of anomalous data instances that might exist in IoT devices. graph similarity algorithm by measuring the similarity of provenance graphs in the learning and detection phase. We evaluate the effectiveness of our approach to graph based anomaly detection with data set from sample IoT application(s). Detailed results are presented in sections IV.

Our technical contributions are outlined in detail as follows:

The remaining portion of this paper is organized as follows. In Section 2, we discuss background information on anomaly detection, provenance graphs, text based query ranking and a review of k-NN. In Section 3, we discuss how provenance data is converted in vector space using tf-idf. Section 5 explains about our classification approach using k-NN. Section 6 describes our experiment and gives a detailed evaluation. Section 7 contains further discussions. Finally, We summarize and conclude in Section 8.

2 BACKGROUND

2.1 Anomaly Detection

The notion of what is considered an anomaly is ambigous and often 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" [8]. We define an anomaly as trends that deviate from the normal behavior. Normal behavior is defined by 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 [7], in health care for the monitoring of patient care [19], and in computer security for intrusion detection [12, 20]. 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. This 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.

Most of the research on anomaly detection is centered on the detection of anomalous behavior in point based dataset. These techniques might not include the dependent relationships that exist between data points. Graphs provides a means of modeling complex structures such as social networks, computer networks, DNA sequences. 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 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. In addition, normal system behavior is constantly evolving. The issue of generating training or test dataset which classifies anomalous and normal system behavior is a major challenge since not all known anomalous system behavior can be recreated.

Anomaly detection 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 training dataset. This data is considered to be a representation of the system's normal daily activity and free from malicious events. Once training dataset has been collected, the system's activities are further observed. This part is known as the testing phase. In the testing phase, observed system behavior is compared to the Learning phase to determine is an anomaly exists between the two. A threshold as defined by domain experts is used to determine if the observed data is considered an anomaly.

Data labels are grouped into three major categories. Supervised anomaly detection, semi-supervised anomaly detection, and unsupervised anomaly detection. In supervised anomaly detection approach, training data contains instances of normal and anomalous data. Incoming data is classified based on the training data category. In semi-supervised approach, only one class of training data is collected, normal data. Incoming data that does not fit the normal class as specified by a threshold is regarded as an anomalous. In the training phase, most anomaly detection techniques use data derived from the normal system behavior. This is referred to as one class classification. In unsupervised approach, there are no training dataset. it is believed that normal data is clustered around each and occurs more frequently than an anomaly. This is a widey used form of anomaly detection since training data which is hard to get is not requited.

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, 9, 23]

2.2 Provenance Graphs

Provenance denotes the origin of an object and all other 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 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 trust of data [3]. It establishes causality and dependency between all objects involved in the 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 and is a good choice for modeling provenance because provenance data contains information with dependency relationships which denotes causality between multiple data objects. Provenance data is represented by directed acyclic graphs 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 contains noisy data and is further streamed by mapping it to a provenance model. With this information, we are able to build a workflow of device data execution. There has been numerous provenance collection systems developed to track provenance in a computing device most of which deals with tracking system calls [15, 18, 22]. Provenance graph used for experimentation is generated from PAIoT, a provenance-aware framework [] . We chose PAIoT because it captures dependencies between device and sensor events.

2.3 Similarity Measures

Similarity 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:

2.3.1 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.

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

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

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

2.3.3 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 GRAPH BASED ANOMALY DETECTION

3.1 Potential Anomalies

Due to the ubiquitous nature of IoT devices, there are a wide array of potential vulnerabilities associated with them. We focus on select vulnerabilities of inconsistent sensor output. Inconsistency are analogous to spikes in sensor readings, constant data values, or faulty sensors, inconsistent device attributes and denial of service attacks on sensor data.

3.2 Graph to Vector Space Conversion

3.2.1 Feature Extraction. In order to apply clustering algorithms or similarity measure to provide anomaly detection, we compute a vector space representation of our provenance graphs. This representation is used as input parameters to our hybrid anomaly detection algorithm. Selecting features from graph properties is an important task because we need to select features that preserves the order and details of each node and edges contained in the graph. We focus on the frequency of the nodes and edges contained in both graphs. Selecting the right features is important in ensuring optimal performance of our anomaly detection algorithms. Our approach not only utilizes the frequency of nodes but also consists of a damping factor which regulates the weight of nodes or edges that occur frequently and increases the weight of nodes and edges that occurs less frequently.

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 represents a vector by using our graph embedding approach which draws emphasis from document query retrieval. This approach preserves the order of edges and allows updates of edges of incoming graphs. We formally define our approach as follows:

Definition 3.1. Let $P = \{p_1, ..., p_n\}$ where $p_i = (V, E)$, the vectorial representation of p_i, v_i is the number of times each vertice, V_i and edges, E_i contained in P appears in the graph , p_i.

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

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

The order of Edges and vertices can be found by taking a breadth first search or depth first search transversal of the graphs.

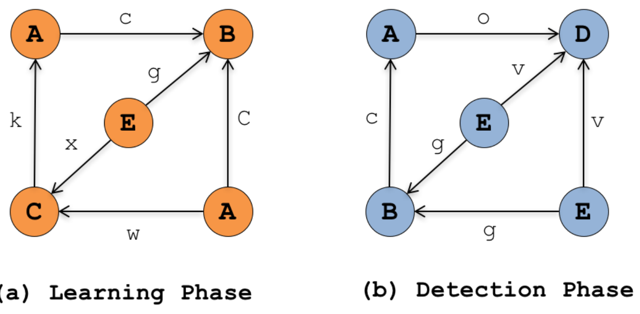

Figure 1: Provenance graphs generated from the learning and detection phase

For example, Figure 1 displays provenance graph 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. The vector representation of the two graphs, v_1, and v_2 are

$$\boldsymbol{v_1} = (1, 1, 1, 1, 1, 0, 0)$$
$$\boldsymbol{v_2} = (1, 2, 0, 1, 1, 1, 0)$$

3.3 Graph Based Similarity Detection Algorithm

The method of comparing the similarity of provenance graphs based on a similarity metric is 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 2 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

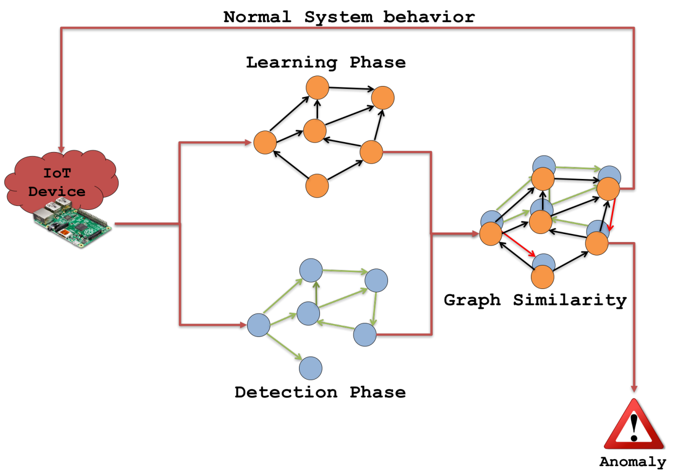

Figure 2: Graph Similarity Approach

classify the behavior of provenance graphs in the detection phase as normal or an anomaly.

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

3.3.1 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.

3.3.2 Computational Complexity: TODO

3.4 Hybrid Detection Algorithm

TODO...

4 EXPERIMENTAL EVALUATION

TODO...

4.1 Threat Model

We assume the attacker has access to the physically or remotely device either by malicious code injection. Network anomaly detection is out of the scope of this research. We do not guaranty security due to hardware tampering. Our approach does not dettect eavesdropping attacks.

5 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 [13] 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 [10] 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. [21] 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 [14] 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 [17] 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 [16] 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.

6 DISCUSSION

TODO

7 SUMMARY AND CONCLUSION

In this paper, we proposed two anomaly detection algorithms for intrusion detection on IoT devices. Our approach is lightweight and efficient in detecting anomalies that exists using provenance graphs. We evaluated the functionality of our approach through implementation. In conclusion, we believe that creating an efficient anomaly detection algorithm is an essential to ensuring an end to end security of IoT devices.

8 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] Dave Evans. 2011. The Internet of Things How the Next Evolution of the Internet Is Changing Everything. https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf, (Apr 2011). Accessed: 2016-10-01.

[7] 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.

[8] 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

[9] 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

[10] 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

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

[12] 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.

[13] 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

[14] 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

[15] 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

[16] 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

[17] 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

[18] 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.

[19] 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.

[20] 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.

[21] 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

[22] 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

[23] 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