IoT Device Sensor Data Anomaly Detection using Provenance Graphs

Ebelechukwu Nwafor, Andre Campbell and Gedare Bloom
Department of Electrical Engineering and Computer Science
Howard University, Washington, DC 20059
Email: ebelechukwu.nwafor@bison.howard.edu

***Abstract*—The Internet of Things (IoT) has revolutionized the way humans interact with devices. Unfortunately, this technological revolution 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 detect the source of intrusion. In a given IoT device, provenance which denotes causality between system events offers immense benefit for intrusion, detection. Provenance provides a comprehensive history of activity performed on a system's data, which indirectly ensures device trust. In this paper, we propose an approach to intrusion detection of IoT devices that leverages the observed behavior of sensor data flow as seen through provenance graphs. In this approach, an observed provenance graph is compared to an application's known provenance graph to determine if an anomaly exists. This comparison involves the dimensionality reduction of a provenance graph to a vector space representation and similarity metric. We implemented our approach on an IoT application which simulates a climate control system.**

***Index Terms*—Intrusion detection, Internet of Things, Provenance**

I. Introduction

Over the years, IoT devices have become an essential part of our daily lives in commercial, industrial, and infrastructure systems. These devices offer immense benefits to consumers by interacting with our physical environment through sensors and actuators which allows device automation thereby improving efficiency. Unfortunately, the proliferation of IoT devices has led to an increase in the number of remotely exploitable vulnerabilities leading to new attack vectors that could have disastrous financial and physical implications [1]. In 2015, security researchers demonstrated a vulnerability on the Jeep vehicle which allowed remote control of the digital system over the Internet [2]. In 2016, researchers discovered a vulnerability that allows internet connected smart thermostats be subject to remote ransomware attacks in which an attacker gains sole control of the thermostat until a fee is paid [3]. These are a few example of some of the potential malicious vulnerabilities that could have devastating long lasting impact on an IoT system.

Intrusion detection [4] is the process of discovering malicious exploits that exist in a system (i.e. network, host). One way of detecting an intrusion 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 could be indicative of 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.

Unlike most host-based intrusion detection system which utilize system call frequencies to determine anomalous system events, our approach detects anomalies by leveraging the information flow of sensor events as represented in a provenance graph. Provenance graph captures a comprehensive history of transformations that occurs on a data object during its lifecycle. It offers a way of representing relationships that exist between multiple data objects. This, in turn, can be used to detect system faults or anomalous system events such as malicious intrusions. We propose an approach to identifying anomalous sensor events using provenance graphs. This approach involves the use of a similarity measure to compare observed provenance graphs with provenance graph derived from an application's normal execution. The result of the algorithm generates an anomaly score in which a threshold is set that classifies an observed provenance graph as either anomalous or benign. We evaluate the effectiveness of our approach with a sample IoT application which simulates a climate control system..

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 describe our threat model and assumptions, In section 4, we discuss details of our graph based anomaly detection algorithm. In section 5, we describes our experimental design and evaluation. In section 6 we describe related work on intrusion detection and graph based anomaly detection. Finally, in section 7, we summarize and conclude our findings.

II. BACKGROUND

A. Anomaly Detection

The notion of what constitutes an anomaly is 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" [5]. In computing, an anomaly often indicates the presence of a malicious intrusion 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 rule-based, statistical, clustering or classification techniques to determine normal or anomalous data instances. 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 set from the data to use for detection. Another challenge exists in defining what constitutes as normal system behavior. Most anomaly detection using point based data often fail to include the dependencies that exist between data points.

B. Provenance Graphs

Provenance denotes the origin of an object and all activities that occurred on it. 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 a degree. In 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 [6]. It establishes causality between entities contained in a data object through information flow tracking thereby it allows for the verification of a data source. Provenance is represented by a directed acyclic graph (DAG) in which vertices represent various data entities and the edges correspond to the interaction between them.

Most provenance collection frameworks developed to track provenance use system level event sequence in an operating system [7]–[9]. For most IoT device, containing limited or no operating system functionality, it is essential to use a provenance collection framework that places less emphasis on an operating system and more emphasis on application level information flow tracking. For our provenance graph generation, we use PAIoT [10], a provenance collection framework which tracks the information flow of sensor based event in an IoT device over its lifecycle.

We formally define a provenance graph as follows:

Definition 1:

A provenance graph is a directed acyclic graph, $p = (v_i, e_i)$ where V is a set of vertices $V = \{v_1, ..., v_n\}$ such that $v_i = (label, type, value)$. Two vertices v_x, v_y are similar (denoted $v_x \sim v_y$) if $v_x.label = v_y.label$. V consists of the following types: Agent, Entity, and Activity. An agent is an actor that is represented by an identifier (e.g. sensor or device name). An entity is an event, which represents data that is produced as a result of an activity. An event is data that is produced as a result of an activity. An activity is an action performed by an agent or entity (e.g. read, update, create, delete). E is a set of edges $E = \{e_1, ..., e_n\}$ where $e_i = (v_s, v_d, label)$ and v_s, v_d are source and destination vertices. $label$ takes on one of the following values: *wasGeneratedBy*, *used*, *wasInformedBy*, *wasDerivedFrom*, *wasAttributedTo*, *wasAssociatedWith*, *actedOnBehalfOf*.

1) Defining an IoT application provenance graph: Many IoT device consist of control systems in which sensor data is used as an input in a feedback loop to an actuator. The operations of most control systems are deterministic. This notion can be leveraged to define an expected provenance graph for each application.

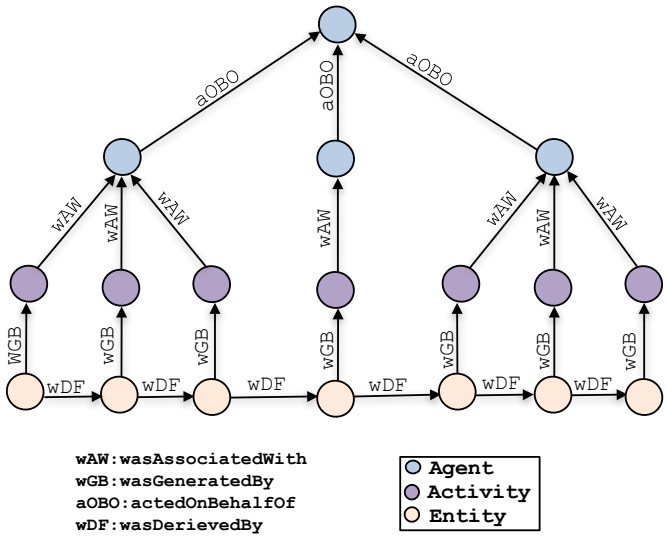

Fig. 1. Components of a provenance graph where nodes represents types (agent, activity, and entity) and edges represents relationship between types

There are two types of control systems: open and closed loop control system. In an open loop control system, the system's output is not affected by the conditions (i.e. there are no feedback loop). An example of an open loop control system is a microwave. The timer is manually set to determine when the microwave stops regardless of the condition of its content (e.g. if the food is properly heated). However, in a closed loop control system, the system's output is used as feedback to influence the input. An example of a closed loop system is a thermostat in which the temperature serves as output which is also used as an input feedback to regulate the temperature based on a predefined set-point. Using feedback as input increases the accuracy of an output.

Each iteration of the control loop sequence generates a path in a provenance graph. The nodes contained along the path denote historical event transformations. For example, in a thermostat application, temperature readings generated might be converted from Celsius to Fahrenheit and utilized as feedback to an actuator or it could be stored locally or transmitted to a remote location. We focus on networked connected control system. Network connection opens doors to potential vulnerabilities

C. Similarity Metric

Similarity metric measures how identical two objects are. It compares the distance between data objects either by 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.

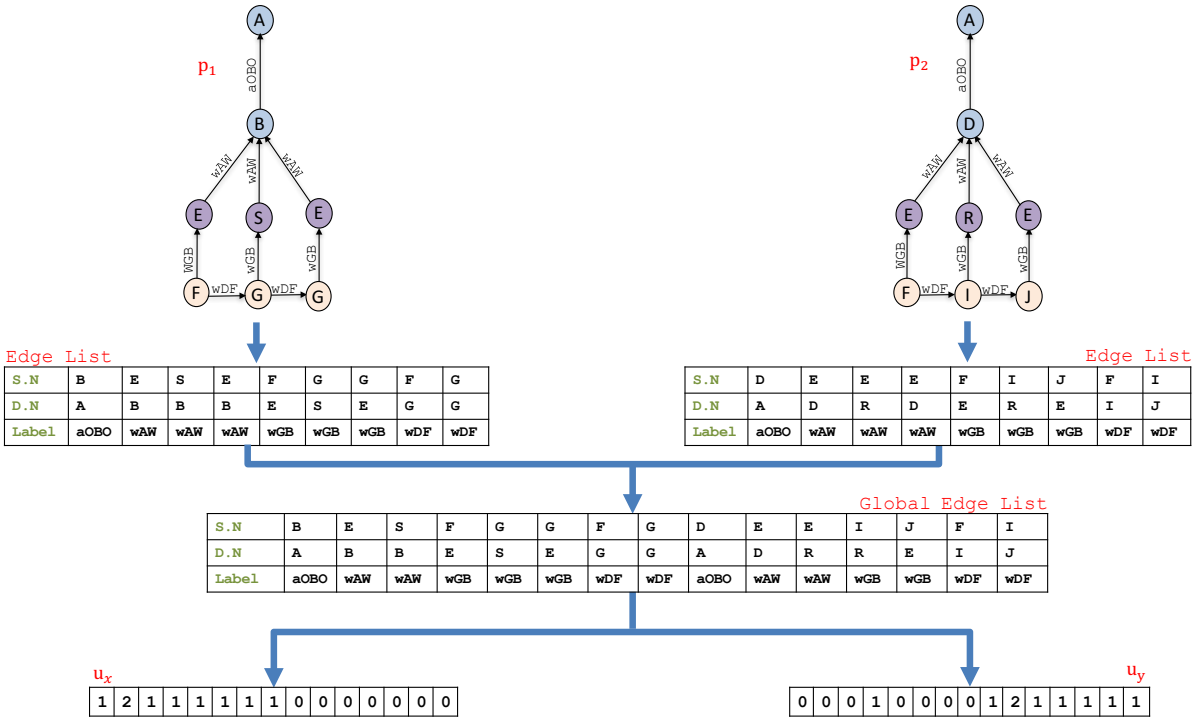

Fig. 2. Provenance graph conversion to vector space. u_x, u_y represents the vectors generated from both provenance graphs

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 identical (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:

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

III. Threat Model and Assumptions

Due to the ubiquitous nature of IoT devices, there are a wide array of vulnerabilities associated with them. It is believed that each application exhibits a unique sequence of events which can be represented as the provenance graph of the application normal data flow. In designing our anomaly detection framework, we expect 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 information flow of sensor events.

IV. Graph Similarity Anomaly Detection

The expected regularity of provenance graphs in IoT applications motivate a graph-based approach to anomaly detection. This approach consists of two phase: observation phase, also known as the training phase and the test or detection phase. In the observation phase, the system collects provenance data considered to be a representation of the normal system behavior. In the detection phase, provenance graph from the observation phase is compared with the provenance graph derived from subsequent observations to determine if an anomaly exists.

A. Provenance Graph to Vector Space Conversion

Our method of comparing the similarity of provenance graphs was inspired by an information retrieval technique for document retrieval. 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 contained in the corpus are converted into a vector space representation which allows documents to be ranked based on some similarity metric.

1) Provenance Graph Feature Extraction: A similarity function measures how identical two provenance graphs are in a vector space. In order to apply a similarity function, we compute a vector representation. In this approach, a provenance graph undergoes dimensionality reduction into an n-dimensional vector space where n represents the number of unique edges (global set of edges) contained in the provenance graph set. This representation is used as input to our anomaly detection algorithm.

Selecting the most appropriate feature from provenance graphs is an important task. We need to utilize features that preserves the order of edges and nodes contained in the

provenance graph. Our approach utilizes the occurrence of unique edges contained in both graphs.

We provide a means of classifying such edges as identical. To determine identical edges, we utilize a combination of edge labels and source and destination node. That is, two edges e_x and e_y are similar (denoted $e_x \sim e_y$) if $e_x.v_s \sim e_y.v_s$, $e_x.v_d \sim e_y.v_d$, and $e_x.label = e_y.label$

An edge list E_p consists of all of the edges contained in p. A global edge list E_G consists of all non-similar edges contained in P. E_G preserves the ordering of edges contained in P in the vector space.

Definition 2:

Let $P = \{p_1, ..., p_n\}$ such that $p_i = (V, E)$. $E_G = \{e_1, ..., e_n\} \mid E_G \in P$ where $e_i \neq e_{i+1}, 1 \leq i \leq n$. $E_p \leftarrow \{e_1, ..., e_n\} \mid E_p \in p$ where $e_{pi} \neq e_{pi+1}, 1 \leq i \leq n$

Finally, we define our provenance graph to vector approach as follows:

Definition 3: Let $P = \{p_1, ..., p_n\}$ where $p_i = (V, E)$. The vector space representation of $p_i, \boldsymbol{u_i}$, is the number of times edges contained in E_p occurs in E_G. $\boldsymbol{u_i}$ has a length of k which consists of all of the unique edges contained in the global edge list E_G.

Figure 2 displays the vector space conversion of provenance graphs. $\boldsymbol{p_1}$, and $\boldsymbol{p_2}$ consists of vertices $A, B, E, F, G, I, J, K, M$ and edges $aOBO, wAW, wGB, wDF$. The vector representation of graphs, u_1, and u_2 is the occurrence of edges contained in the edge list that is found in the global edge list.

B. Graph Based Similarity Detection Algorithm

Given u_x, u_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.

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

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

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 application 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 paper.

V. Experiment

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

procedure GRAPHSETTOEDGELIST(P)
  INPUT: $P = \{p_0, ..., p_n\} \mid p_i \leftarrow (V_i, E_i), 0 \leq i < n.$
  $E_G \leftarrow \{\}$
  for $p_i = (V_i, E_i) \in P, 0 \leq i < n$ **do**
    for $e_j \in E_i$ **do**
      Found $\leftarrow$ False
      for $e_g \in E_G$ **do**
        if $e_j \sim e_g$ **then**
          Found $\leftarrow$ True
        end if
      end for
      if *Found* = False **then**
        $E_G \leftarrow E_G \cup e_j$
      end if
    end for
  end for
  return E_G
end procedure

procedure GRAPHTOVECTOR(E, E_G)
  $k = |E_G|$
  $Q[k] \mid Q[i] \leftarrow 0, 0 \leq i < k$
  for $e_j \in E$ **do**
    for $e_g \in E_G \mid 0 \leq g < k$ **do**
      if $e_j \sim e_g$ **then**
        $Q[g] \leftarrow Q[g] + 1$
      end if
    end for
  end for
  return Q
end procedure

procedure GRAPHANOMALY(P, p)
  $E_G \leftarrow GraphSetToEdgeList(P \cup p)$
  $Q \leftarrow GraphtoVector(p, E_G)$
  for $p_i \in P$ **do**
    $N_i \leftarrow GraphtoVector(p_i, E_G)$
    $z \leftarrow sim(Q, N_i)$
    if z $\geq$ threshold **then**
      return normal
    end if
  end for
  return anomaly
end procedure

Fig. 3. Graph based anomaly detection algorithm

A. Experimental Setup

We evaluate our intrusion detection algorithm by implementing an IoT application which simulates a climate control system. Constant irregularities in temperature could have devastating effects on industrial machinery. Climate control systems ensures a proper functional environment for people and machinery. This system consist primarily of a Heating Ventilating and Air Conditioning (HVAC) System which uses temperature, humidity data to regulate environmental conditions within a building. The temperature is set within a

bounded limit such that when the temperature exceeds a certain threshold, a cooling phase is activated and the air conditioning is switched on until the temperature decreases below a threshold in which the heating phase is activated in which the heater is turned on.

For the implementation of our climate control system, we utilized a publicly available dataset [11] which consists of a year's worth of longitudinal data on the thermal conditions, related behaviors, and comfort of twenty-four occupants in a medium-sized building. This dataset consist of temperature, humidity, air velocity generated at a duration of fifteen minutes. We utilize the temperature and humidity data as input to our aforementioned simulation program.

We performed experiments to see if we are able to detect anomalous instances in temperature and humidity data. We generate provenance graphs for each week of the year. We compare the provenance graph generated in various weeks to see how they differ using our graph similarity approach (e.g week 1 compared to week 2, week 2 compared to week 3).

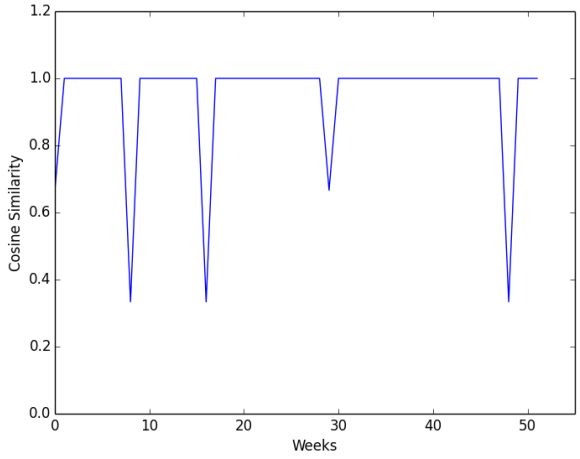

Fig. 4. Provenance graph comparison of climate control system by week

Figure 4 shows a graph of the cosine similarity values generated from the comparison between the provenance graphs by various weeks. Most of the provenance graphs generated are similar (i.e having a cosine similarity of 1). This might be as a result of less variation in temperature change.

VI. 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-Nearest Neighbors is used to classify the training data set. Stephanie et al [13] 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 where 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. [14] developed a technique for intrusion detection on embedded systems by analyzing system call frequencies. This is achieved by learning normal system profile of observed patterns in the system call frequency distribution.Their hypothesis is 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 runtime 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 [15] 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. Papadimitriou et al [16] proposed five similarity algorithms for comparing the similarity of web graphs namely signature similarity, vertex/edge vector similarity, vertex ranking,and vertex edge overlap. These algorithm are inspired by document similarity method namely shingling and random projection based method. Nodes represents a webpage and an anomaly could indicate 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 [17] 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 than 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* [18] consist of nodes with similar neighbors are clustered together and the edges which do not belong to any cluster are considered as an anomaly. To detect communities, it achieves this task by reorganizing the rows and the columns of the adjacency list.

VII. Summary and Conclusion

In this paper, we propose an anomaly detection algorithm for detecting anomalous instances of sensor based events in an IoT device using provenance graphs. We implemented our approach on an IoT application which simulates a climate control system. We plan on conducting detailed experiments to identify false positives, true positives, and false negative rates.

VIII. Acknowledgment

This research has been supported in part by US National Science Foundation (CNS grant No. 1646317). 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.

References

[1] M. B. Barcena and C. Wueest, "Insecurity in the internet of things." [Online]. Available: https://www.symantec.com/content/dam/symantec/docs/white-papers/insecurity-in-the-internet-of-things-en.pdf

[2] A. Greenberg, "The jeep hackers are back to prove car hacking can get much worse," 2015. [Online]. Available: https://www.wired.com/2016/08/jeep-hackers-return-high-speed-steering-acceleration-hacks/

[3] D. Raywood, "Defcon: Thermostat control hacked to host ransomware," 2016. [Online]. Available: https://www.infosecurity-magazine.com/news/defcon-thermostat-control-hacked/

[4] A. Lazarevic, V. Kumar, and J. Srivastava, *Intrusion Detection: A Survey*. Boston, MA: Springer US, 2005, pp. 19–78. [Online]. Available: https://doi.org/10.1007/0-387-24230-9_2

[5] D. Hawkins, *Identification of outliers*, ser. Monographs on applied probability and statistics. London [u.a.]: Chapman and Hall, 1980. [Online]. Available: http://gso.gbv.de/DB=2.1/CMD?ACT=SRCHA&SRT=YOP&IKT=1016&TRM=ppn+02435757X\&sourceid=fbw_bibsonomy

[6] E. Bertino, *Data Trustworthiness—Approaches and Research Challenges*. Cham: Springer International Publishing, 2015, pp. 17–25. [Online]. Available: https://doi.org/10.1007/978-3-319-17016-9_2

[7] T. Pasquier, X. Han, M. Goldstein, T. Moyer, D. Eyers, M. Seltzer, and J. Bacon, "Practical whole-system provenance capture," in *Symposium on Cloud Computing (SoCC'17)*, ACM. ACM, 2017.

[8] R. H. Zakon, Ed., *28th Annual Computer Security Applications Conference, ACSAC 2012, Orlando, FL, USA, 3-7 December 2012*. ACM, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2420950

[9] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer, "Provenance-aware storage systems," in *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference*, ser. ATEC '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 4–4. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267359.1267363

[10] E. Nwafor, D. Hill, A. Campbell, and G. Bloom, "Towards a provenance aware framework for internet of things devices," in *Proceedings of the 14th International Conference on Ubiquitous Intelligence and Computing*, ser. UIC '17. San Fransisco, CA, USA: IEEE Computer Society, 2017.

[11] J. Langevin, P. L. Gurian, and J. Wen, "Tracking the human-building interaction: A longitudinal field study of occupant behavior in air-conditioned offices," *Journal of Environmental Psychology*, vol. 42, no. Supplement C, pp. 94 – 115, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0272494415000225

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

[13] S. A. Hofmeyr, S. Forrest, and A. Somayaji, "Intrusion detection using sequences of system calls," *J. Comput. Secur.*, vol. 6, no. 3, pp. 151–180, Aug. 1998. [Online]. Available: http://dl.acm.org/citation.cfm?id=1298081.1298084

[14] M.-K. Yoon, S. Mohan, J. Choi, M. Christodorescu, and L. Sha, "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*, ser. IoTDI '17. New York, NY, USA: ACM, 2017, pp. 191–196. [Online]. Available: http://doi.acm.org/10.1145/3054977.3054999

[15] E. Manzoor, S. M. Milajerdi, and L. Akoglu, "Fast Memory-efficient Anomaly Detection in Streaming Heterogeneous Graphs," in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: ACM, 2016, pp. 1035–1044. [Online]. Available: http://doi.acm.org/10.1145/2939672.2939783

[16] P. Papadimitriou, A. Dasdan, and H. Garcia-Molina, "Web graph similarity for anomaly detection," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 19–30, May 2010. [Online]. Available: https://doi.org/10.1007/s13174-010-0003-x

[17] C. C. Noble and D. J. Cook, "Graph-based anomaly detection," in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '03. New York, NY, USA: ACM, 2003, pp. 631–636. [Online]. Available: http://doi.acm.org/10.1145/956750.956831

[18] D. Chakrabarti, "Autopart: Parameter-free graph partitioning and outlier detection," in *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer, 2004, pp. 112–124.