Automatic Identification of Critical Data Items in a Database to Mitigate the Effects of Malicious Insiders

Jonathan White and Brajendra Panda

Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville, Arkansas, 72703, USA
{jlw09, bpanda}@uark.edu

Abstract. A major concern for computer system security is the threat from malicious insiders who target and abuse critical data items in the system. In this paper, we propose a solution to enable automatic identification of critical data items in a database by way of data dependency relationships. This identification of critical data items is necessary because insider threats often target mission critical data in order to accomplish malicious tasks. Unfortunately, currently available systems fail to address this problem in a comprehensive manner. It is more difficult for non-experts to identify these critical data items because of their lack of familiarity and due to the fact that data systems are constantly changing. By identifying the critical data items automatically, security engineers will be better prepared to protect what is critical to the mission of the organization and also have the ability to focus their security efforts on these critical data items. We have developed an algorithm that scans the database logs and forms a directed graph showing which items influence a large number of other items and at what frequency this influence occurs. This graph is traversed to reveal the data items which have a large influence throughout the database system by using a novel metric based formula. These items are critical to the system because if they are maliciously altered or stolen, the malicious alterations will spread throughout the system, delaying recovery and causing a much more malignant effect. As these items have significant influence, they are deemed to be critical and worthy of extra security measures. Our proposal is not intended to replace existing intrusion detection systems, but rather is intended to complement current and future technologies. Our proposal has never been performed before, and our experimental results have shown that it is very effective in revealing critical data items automatically.

1 Introduction

Intrusion detection systems are important tools in the fight against malicious attackers on computing systems. These tools monitor different system activities and report on activities that can be construed as malicious. The system administrator or security engineers look at this information, and based on experience to some extent, determine which of these actions are indeed malicious. In this work, we propose a more quantitative approach to help these system administrators make sound judgments about where to focus their security efforts and to better identify data items that are

critical to the system. When these critical data items are automatically identified, the security engineers can provide better and more thorough protection to these items.

Existing intrusion detection systems suffer shortcomings in this regard. First, not many of them do a good job in handling threats from malicious insiders [1]. These attacks, which are often considered to cause the majority of major security breaches, are a significant threat to all computing systems. The insider threat is of paramount importance when designing security systems, and more work needs to be done on classical intrusion detection systems to counteract the malicious insider [7].

A second concern with existing intrusion detection systems is that they do not typically allow system administrators the ability to focus on mission critical data [8]. It is very hard, if not impossible, to protect all data on a system against all forms of misuse. Without guidance on what is and is not critical to the system, it is difficult to focus preventative and detection measures on those assets that are deemed critical, and without such focus the efforts at data collection and analysis are likely to overwhelmed by the "noise" produced in classical intrusion detection systems. These sources of noise produced by classical intrusion detection systems include lesser offenses that will not cause extensive damage, mistakes made unintentionally by non-malicious users, and the like.

As a corollary to this, modern information systems contain a constantly shifting collection of gigabytes, if not terabytes, of data and information. It is unlikely that static lists of critical files and processes will remain relevant over time as the system changes. The identification of critical data in a system must be dynamic, and the information about what is critical in the system must be able to be altered as the system changes [5].

Also, classical intrusion detection systems tend to rely on the fact that security engineers are generally familiar with the data systems that they are charged to protect. However, this assumption is a dangerous one as data systems continue to expand, sometimes exponentially. A malicious insider may have intimate knowledge about what files or databases are important to their part of the business, but this expert knowledge does not necessarily transfer to other layers of the organization, even to the group tasked with computer security. Security engineers, though insiders themselves, may not be familiar with what data items are critical as they might be outside of their normal scope of knowledge.

These several factors lead us to propose a new method that can be used to automatically identify data items that are critical. In our research, we have identified two approaches to this process. One involves identifying critical data items by content alone, and the second method involves examining the usage of the data system and detecting which data items influence a large number of other data items and at what frequency this influence occurs. As the identification of critical data items by content typically requires expert knowledge, we have chosen not to focus on that at this time. However, it is an area of future work, and we will be examining this method later.

The second proposal uses statistical data relationship oriented models. This method attempts to locate critical data without necessarily being concerned about the content of the data. Attributes that identify critical data items would include ownership, file size, view count, time of last access, file location, and what data items were written to after reading this data item (influence). These methods may identify

data items that are not necessarily the most popular in the computing system but nonetheless influence a large portion of the database. We are going to use this statistical approach in our work as it does not require expert knowledge about the content of the data and is applicable to all database systems. We will use the metrics of view count, time of access, and influence.

Our approach is different from classical intrusion detection systems in that we will enable the system administrator to automatically identify which data items are critical. As listed in [6], establishing critical assets is one of the key strategies in order to minimize the impacts of insider threats. In a database, a critical data item is one on which many other data items in the system depend on. That is, if such a data item gets maliciously updated or deleted, many other items will consequently be affected, resulting in incorrect queries that will require repair procedures. Therefore, it is necessary to enforce tighter access control and also monitor access to these items more thoroughly. Next, we briefly define our technique for identifying these critical sets.

We detect these critical data items by considering data dependency relationships. The database log is scanned, and one or more disjoint and directed graphs are formed that show the can-influence relationships and the amount of times that the data items were accessed over this period. The can-influence graph shows the sets of relationships that occurred where two nodes connected by a directed edge indicates that the data item pointed by the arrow was influenced by the other data item. That is, if the latter is modified, there is a possibility that the former will be affected. Furthermore, by analyzing the graph for sinks (nodes to which many arrows point), sets of critical data items are identified by way of an algorithm that we have designed, tested, and evaluated.

The advantage of our approach is that it is a flexible and resource efficient technique. It solves the problem of identifying critical data items, and it also avoids the pitfalls of using static lists of critical data items. Also, as data systems change constantly, the proposed algorithm can be invoked to process this dynamically changing environment. By identifying the critical data items, system administrators can deploy tighter and more focused access policies, more detailed monitoring systems, and other focused insider detection tools, such as honeypots and honeynets.

The rest of the paper is organized as follows. In section 2 we briefly discuss why insiders are a threat to critical data items and what work has been done previously in detecting data items that can influence other data items, which is the starting point of our work. Section 3 details our proposed approach, defines the necessary definitions of our work, and shows an example of how our approach operates. Section 4 concludes the paper with an evaluation of the results and future areas of improvements.

2 Background and Related Work

In the following section, we will identify why we are trying to mitigate the effects of insider threats by identifying the data in the system that is critical. We define what a malicious insider is, what their motives are, the opportunities that are unique to

insiders to cause damage using critical data, and the methods that insiders use that make them hard to defend against. We also summarize past work on using graphs to track certain data items in a database.

2.1 Insider Threats in Relation to Critical Data

CERT defines a malicious insider as a current or former employee, contractor, or business partner who has or had authorized access to an organization's network, system, or data and intentionally exceeded or misused that access in a manner that negatively affected the confidentiality, integrity, or availability of the organization's information or information systems [4]. This definition of a malicious insider has a wide reaching scope and is a good starting point for defining why insider threats are so hard to mitigate against. Insider threats are an ever increasing problem, with the recent E-Crime Watch Survey conducted by the US Secret Service, CERT, Microsoft, and others reporting that in cases where respondents could identify the perpetrator of an electronic crime, in 31% of those cases the crimes were committed by insiders. Also, 49% of the respondents reported experiencing at least one malicious, deliberate attack by an insider within the last calendar year. The impact of insider crime can truly be devastating; in one recent case an employee stole blueprints on a new and classified process worth an estimated $100 million and sold them to a Taiwanese competitor with the hope of obtaining future employment with that organization [4].

The motive for a malicious attack can be grouped into three main areas: IT sabotage, theft/modification for financial gain, and theft/modification for a business advantage [4]. An example of IT sabotage occurs when a disgruntled employee who has recently been fired causes intentional damage to a database they are charged to maintain, knowing that this will cause a lack of availability for the organization in the time following their leaving the company. IT saboteurs target systems that are critical to the business; otherwise their malicious actions would be easily repairable or ignored and the sense of revenge over the precipitating event would not be fulfilled. Insiders who steal data for financial gain would include individuals who secretly sell proprietary data to an outsider with the hope of a monetary reward. Again, malicious insiders whose motive is financial gain will not target non crucial data as it would not be as valuable to an outsider. The final motive for insiders involves individuals who steal/modify data in order to bring it with them to a new job or to start up their own business. An example would be a salesman who copies the customer list from the database and brings it with them to a new job. Again, typically only important data will be taken [9]. So, the motivation for an insider to commit a crime involving critical data items is quite high [12], [13], [14].

Insiders have significant unique opportunities over others (including the system administrators and security engineers) when it comes to committing an electronic crime [10]. Insiders, by design, can bypass physical and technical security measures designed to prevent unauthorized access; the data must be made available to them in order for their business function to work properly. In terms of methods, insiders can often use the same methods that they use everyday to access the data. As mentioned before, most systems are tuned to detect threats from outside the system, not necessarily internal threats. Insiders are also aware of the policies, procedures,

technologies, and the associated vulnerabilities that are linked with them. They are able to exploit flaws in the system because of their expert knowledge; they work with these critical data items everyday. This unique opportunity that is afforded to malicious insiders of familiarity and their knowledge of the methods required in order to access the important data is what makes identifying what is and is not critical in the data system such an important task in order to mitigate this risk [16].

2.2 Previous Work

The idea of using graphs in insider threat detection has been used before, though not in relation to automatic identification of critical data items. In [1] and [11], attack tree graphs were used to identify malicious attacks from users who were performing seemingly innocuous actions in the system. In [3], a graph was developed that showed what data items could potentially influence other data items were an attack to occur. Then, in the recovery process, only the items that were influenced by the malicious action would need to go through the recovery process; those that were not in the scope of influence could be ignored, saving time and resources. In [2], a scalable graph based network vulnerability analysis was performed, identifying potential network targets. Graphs and attack trees have been proven to be a good tool at identifying and stopping potential insider threats. We have chosen to use some of these basic ideas in identifying critical data items, and some of the theorems that were proven in the previous work will be applied to our graph based system.

Previous insider threat works have called for more effort on the identification of critical data items [15]. In [5], it was pointed out that clear definitions are needed regarding what constitutes critical assets on a system in order to protect it from insider misuse. Without such guidance, most protection measures are doomed to failure. In [6] it was stated that it is important to focus R&D and operational measures on those items within an information system that are critical. It posed the question about whether procedures could be developed by which critical information within a system could be automatically identified by using expert systems and/or rule-based approaches in order to expedite this process.

3 Automatic Identification System

Our automatic identification of critical data item system works briefly as follows. We begin by developing a directed graph architectural model by scanning the database log. The log is scanned for items which are typically read before some other data item is written to or used for some business process. These data items are considered to have “influenced” the latter item because information flow occurred; based on the value of the previous item, the latter item was changed accordingly. The frequency that this influence occurs is maintained, as well as the total number of instances that the particular data item is used. Also, the average time between influences is calculated during this process. This results in several potentially disjoint graphs that will then be traversed, starting at the leaf nodes, until no more internal nodes remain

for each tree. A function will be called for each item in order to determine this level of criticality, taking into account the number of items that are influenced, the number of times this item was used, and the rate at which this influence occurs. Ultimately, each item in the database will be associated with a criticality value, and those items with a relatively large critical value will be afforded greater protection. As we are focusing on those items that are critical from a usage point of view as opposed to a content based viewpoint, our algorithm may not find the items that are the most important based on the confidentiality level of the data. Rather, our approach is content independent, and will find those data items, which if damaged or altered, would cause the most damage due to their highly influential status in the system.

In the following sections we describe each component of our system in details. We begin by describing how our process scans the database logs in order to form the criticality graph.

3.1 Requisite Terminologies

Several intrusion detection systems scan the historical database logs in order to establish the normal operation of the data system. Most anomaly detection systems work by comparing known good behavior to present behavior, and when deviations occur, further security procedures are activated. We scan the database log, taking note of the following, in order to identify potential critical data items: 1. Items in the read set of the transactional sequence; 2. Items in the write set of the transactional sequence; 3. Time stamp of the transaction.

For instance, suppose the following transaction is found in the database logs: *r(x) w(a,b) timestamp: 11:32 AM, July 14, 2009*. We would then conclude that a and b were influenced by x at that particular time. This scanning of the logs is performed for each transaction that occurred over a representative time period. Also, if a transaction views a data item and does not write to another item, this is also maintained as a field. For example, this type of operation would occur when a student executes a query to view the classes that are offered for the next semester but does not register because a class is currently closed. The following definitions help in understanding these concepts.

Definition 1: A transactional sequence is an ordered list of read and/or write operations with an associated timestamp value. We denote a transactional sequence *s* by $< O_1(d_a), O_2(d_b), \ldots O_n(d_n) \mid T_s >$ where $O_i \in \{r,w\}$, d_x is a unique data item with $1 \leq x \leq n$, and T_s being the time *s* was committed to the database.

Definition 2: The read set of a transactional sequence *s* for a data item *x* is the ordered list with the format $< r(d_a), r(d_b),\ldots r(d_n), w(x) \mid T_s >$ which represents that the transaction *s* reads all data items $d_a, d_b,\ldots d_n$ *before* the transaction updates data item *x*. It must be noted that each data item may have several read sequences each having different length. All these sequences together are called the transactional read set of this data item.

The notation $r_s(x)$ is used to represent the transactional read set for transaction *s* on data item *x*. For example, consider the following update statement in a transaction: *Update table1 set x = a+b+c where d > 100* .

In this statement, the values of a, b, c, and d must be read before the potential updating of x. So, $< r(a), r(b), r(c), r(d)\ w(x) \mid T_s > \in r_s(x)$ and a, b, c, and d all potentially influenced x. It should be noted that the database log only contains before and after images of *x* instead of the exact mathematical operation used for calculating x. The above example is only used for illustrating the concept of a read sequence. The database log entry containing the above transaction may in fact look like:

$< r(m), r(n), w(y), r(u), r(v), r(a), r(b), r(c), r(d), w(x), r(d), w(c), w(v) \mid T_s >$

Table 1. Hypothetical Data obtained by log scanning.

Item	***Infl.***	***Freq.***	$\Delta t_{avg.}$
A	B	12	30
A	C	13	45
A	-	9	180
B	D	12	480
B	-	20	10
C	G	19	10
C	J	11	15
D	E	50	16
D	F	37	7
D	J	25	10
E	G	3	60
E	-	2	120
F	H	17	15
F	E	25	23
G	-	14	10
H	I	10	15
H	G	7	14
H	-	3	5
I	-	12	47
J	-	17	5
W	Y	5	15
W	-	3	19
X	Y	3	2
X	Z	2	34
Y	-	11	57
Z	-	0	-
Z	Z	67	18

Definition 3: The write set of a transactional sequence *s* is the list with the format < $r(d_1)$, $r(d_1)$,… $r(d_n)$, $w(d_a)$, $w(d_b)$, … $w(d_{k)} \mid T_s >$ which represents that the transaction *s* reads all data items d_1, d_2,…d_n *before* the transaction updates data items d_a, d_b, … d_k. Therefore, data items d_a, d_b, … d_k. were each influenced by data items d_1, d_2,…d_n.

Definition 4: The maximum time between any two transactional sequences is called T_{max}. T_{max} is calculated as $T_{end} - T_{start}$, where T_{end} and $T_{start} \in < T_{x,} T_y, \ldots T_z >$ and $\min(< T_{x,} T_y, \ldots T_z >) = T_{end}$ and $\max(< T_{x,} T_y, \ldots T_z >) = T_{start}$.

The T_{max} value will be used to define the maximum time period upon which the transactions were recorded. If a transactional read/write sequence only occurs once in the database logs, then it will have a length between influences of T_{max}.

The table on the previous page shows a hypothetical scanning of the logs, detecting the data items that influenced other data items, the rate at which this transactional influence occurred, and also the number of times the data item was used without influencing another item. All of this information is necessary for the algorithms that will be presented in the sections that follow. The $\Delta t_{avg.}$ column in Table 1 shows the average time between each event that occurred in the corresponding row. This is recalculated for each new access that occurs. For simplicity, in the following descriptions, the time values used are measured in minutes, though this unit of time measure is not required algorithmically.

3.2 Criticality Graphs

Then, from this tabular information, one or more directed graphs are formed to represent this information. From the above example, two disjoint graphs will be formed, one with data item A as the root, and another disjoint graph with no roots. These graphs are presented on the following page and will be used as examples to show several important definitions and properties about this graphing process.

3.3 Formal Definition of Criticality Graph

The following definitions and properties formally define what the criticality graph is and how it will be used to locate and identify critical data items in the database. The first definition has been taken from [2] with slight modification.

Definition 5: Data item a "influences" data item b if data item a is read in order to update or write to data item b in a single, atomic transaction *s* that is recorded in the logs of the database with a discrete time stamp, T_s. This relationship is denoted as $a \rightarrow b$ in the graph structure. The following properties hold: 1. Reflexive: $a \rightarrow a$; 2. Non-commutative: $a \rightarrow b \not\Rightarrow b \rightarrow a$; 3. Transitive: $a \rightarrow b \cap b \rightarrow c \Rightarrow a \rightarrow c$.

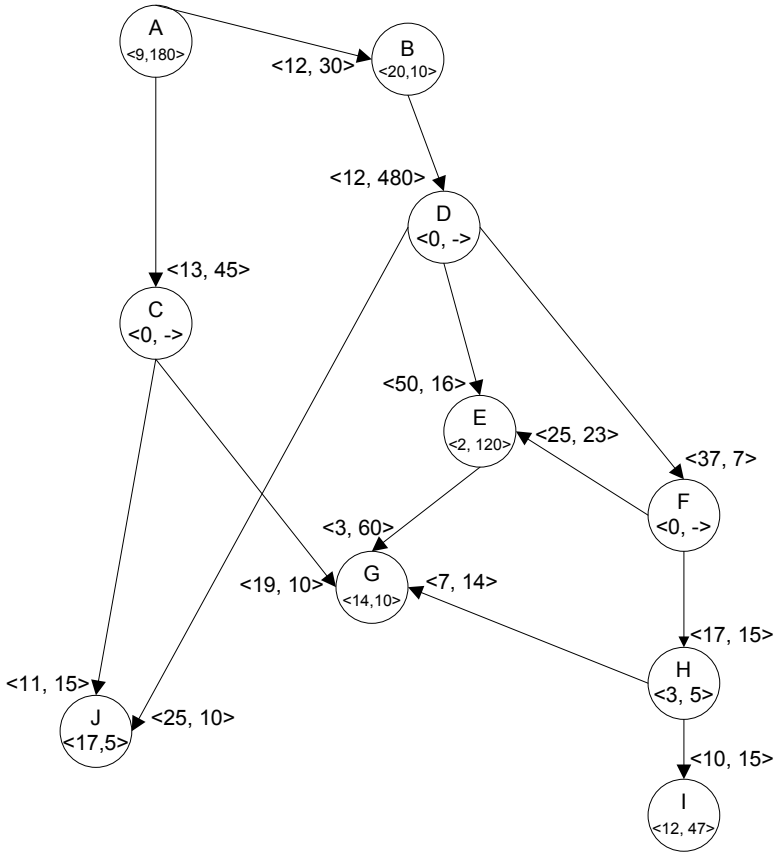

Fig. 1. Simple Criticality Graph Corresponding to Hypothetical Data

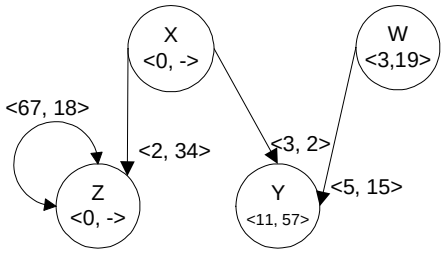

Fig. 2. Second, Disjoint Criticality Graph Corresponding to Hypothetical Data

Definition 6: A criticality graph is a rooted or rootless graph structure defined as CT = (A, E, C), where

1. A is the set of nodes in the graph corresponding to the different data items that are related by some sphere of influence. The set A can be partitioned into two subsets, leaf_nodes and internal_nodes such that
   a) leaf_nodes $\cup$ internal_nodes = A,
   b) leaf_nodes $\cap$ internal_nodes = ϕ.
2. E $\subseteq$ A x A constitutes the set of edges in the criticality graph. An edge (v_i, v_j) $\in$ E represents the "influences" relationship transition from a parent node $v_i \in$ A to a child node $v_j \in$ A in the graph. The edge (v_i, v_j) is said to be "emergent from" v_i and "incident to" v_j. Further if edges (v_i, v_j) and (v_i, v_k) exists in the set of edges, then v_j and v_k represent the same transition.
3. C is a set of criticality labels. A label l $\in$ C is associated with either a node or a transitional edge as previously defined. If S $\in$ A is a node then the criticality label l_S is given by the tuple <f, t> where f $\in$ integers and $\geq$0, and t $\in$ reals and $\phi \leq t \leq T_{max.}$ The item f is termed the frequency of influence and item t is termed the average time between influences.

For example, the criticality tree in Figure 1 consists of the nodes A,B,C,D,E,F,G,H,I, and J. In node A, the internal criticality label is <9, 180> signifying that it was read 9 times without being used to influence other items with an avg. time between these events being 180 min. The label <12, 30> from node A to node B represents the 12 times A was used to update B with an average time between these influences being 30 minutes.

Definition 7: Given a node $v_i \in$ A in a criticality tree, then $v_i \in$ internal_nodes iff. $\exists$ some (v_i, v_j) $\in$ E emergent from v_i that is not incident to v_i.

Definition 8: Given a node $v_k \in$ A in a criticality tree, then $v_k \in$ leaf_nodes iff. ! $\exists$ some (v_k, v_l) $\in$ E emergent from v_k that is not incident to $v_k$and $\exists$ some (v_k, v_m) $\in$ E incident to v_k.

In Figure 2, node Z is actually a leaf node. It does have an edge that leaves the node, but the edge returns to the same node. This models when a transaction reads a data item and then writes back to the same item. This would occur, for example, when a query reads the current salary, increases it by 5%, and then writes back to the salary location.

3.4 Calculating Criticality

We now use the criticality graph to determine the criticality of each data item element in the graph. The algorithm below shows all the steps in the process, but the process is briefly explained in the following. The graph is traversed, starting at the leaf nodes, and criticality is calculated for each node. As the leaf nodes influence no other items, the criticality calculation is relatively straightforward. For the parent nodes, the criticality algorithm takes into account the frequency/average time of use of that item

when it was not being used to influence other items to arrive at an internal criticality value. Then, this value is summed with each data item that is influenced, taking into account the child's criticality, and also the frequency and average time that this influence occurred. During the traversal, the leaf nodes are removed, and ultimately criticality values are calculated for each item. As we wish to find the items that influence several other items at a high rate of time and frequency, frequency of use and influencing several other influential items is considered to add to the criticality value. If the average time between each event is relatively high, this means that the influence takes a long time to spread, so this is a negative contributor to the criticality.

Starting with the leaf nodes of the graph, the criticality value for a generic leaf K, called C_K, is calculated as the number of uses (f_K) times a scaling constant minus a penalty value if the average time between uses (t_K) is over a certain acceptable threshold:

$$C_K = K_1 * f_K - P_K \tag{1}$$

where K_1 is a constant used to weight the criticality value towards or away from the number of uses and P_K is:

$$P_K = \begin{Bmatrix} 0 \mid t_K < t_{\min} \\ \frac{K_1 * f_k}{t_{\max} - t_{\min}} * (t_k - t_{\min}) \mid t_{\min} \le t_K \le t_{\max} \\ K_1 * f_K \mid t_K > t_{\max} \end{Bmatrix} \tag{2}$$

where t_{min} is the bound on the time between use that is considered to happen so often that no penalty is needed to be assessed to the criticality value because the event occurs so often that its influence will spread quickly and t_{max} is the upper bound on the time between usage that is considered to be so infrequent that it results in the data item not being critical because security procedures are adequate enough to repair any damage that might be caused by malicious uses of that item. When t_k is between t_{min} and $t_{max,}$ the penalty is a linear progression from 0 to the maximum penalty of $K_1 * f_k$. For the following examples, we will use a K_1 value of 1. Also, it is important to note that:

$$0 \le C_{LK} \le K_1 * f_K \tag{3}$$

so the criticality can not be negative. As a future work, we may consider how having negative criticality values would better enable us to rank low criticality value data items. Therefore, the criticality of the leaf nodes will consist of the sum of the instances when the item was read without influencing other data items and the instances when the data item was used to influence itself, as this is the definition of what a leaf node is.

Then, once the criticality is calculated for each leaf node, the leaf nodes are removed, and the critical values are stored. This results in a new set of leaf nodes that

were once parents. The links that they once had to the children are still maintained in order to calculate the next set of criticality values. The criticality for a node N with edges $v_i \ldots v_k$ each associated with a frequency $f_i \ldots f_k$, average time duration of $t_i \ldots t_k$ and criticality of $C_i \ldots C_k$, then the criticality for node N is:

$$C_N = \sum_{i=0}^{i=j} (C_i (K_2 * f_{N \to i} - P_{N \to i})) + K_1 * f_N - P_N \quad (4)$$

where K_2 is a constant to weigh the criticality value of a parent towards or away from the criticality value of the link to the child. For the examples that follow, we assume K_2 to be 1. In the above equation, the $f_{N \to I}$ and $P_{N \to I}$ values are the frequency and average time between events for a particular edge I which is summed for each link. The summation is also multiplied times the criticality of the child that the link points to. Finally, the reflexive criticality for the node is added once to give the final critical value for the node.

These operations are performed for each node until no more remain in the graph. While one may assume that the criticality would increase as the graph is traversed upwards, this is not always the case due to the penalty assessed by the average time between influence events. The ultimate output of this process is a list of every data item with an associated criticality value. This process is automatic and requires little input from the system administrator.

3.5 Criticality Example

We will now show, by way of example, how our proposed algorithm operates. We will use the criticality graph that was shown in Figure 1 along with all the frequency and time averages associated with it. We will also assume that the t_{min} and t_{max} are 10 minutes and 60 minutes respectively. Again the constants K_1, and K_2 in the above criticality equations are defined to be one.

First, the criticality for nodes G, I and J are calculated as they are leaf nodes:

C_G: 14 – 0 = 14 C_I: $12 - \frac{12}{60-10}(47-10) = 3.12$ C_J: 17 – 0 = 17

For the following nodes, the self-criticality is calculated first. Then the contributions from the child nodes are calculated in the order given. The criticality for the parents (C, H, and E) of nodes G, I, and J is calculated as follows:

C is the parent of J and G; C_C: $0+17\left(11 - \frac{11}{60-10}(15-10)\right) + 14(19-0) = 434.3$

H is the parent of I and G;

C_H: $(3-0)+3.12\left(10 - \frac{10}{60-10}(15-10)\right)+14\left(7 - \frac{7}{60-10}(14-10)\right) = 121.24$

E is the parent of G; C_E: (2 – 2) + 14(3 – 3) = 0

Then C, H, and E are removed and the criticality for the next set of node(s) is:
F is the parent of E and H;

$$C_F: 0+0\left(25-\frac{25}{60-10}(23-10)\right)+121.24\left(17-\frac{17}{60-10}(15-10)\right) = 1854.97$$

Then the criticality for node D is calculated:
D is the parent of J, E, and F;

$$C_D: 0+17(25-0)+0\left(11-\frac{50}{60-10}(16-10)\right)+1854.97(37-0) = 69{,}058.9$$

Then node B:
B is the parent of D; C_B: (20 – 0) + 69,058.9(12 – 12) = 20

Finally, the criticality of node A is calculated:

A is the parent of B and C;

$$C_A: 20\left(12-\frac{12}{60-10}(30-10)\right)+434.3\left(13-\frac{13}{60-10}(45-10)\right) = 1837.77$$

So, node D is the most critical data item in the database based on its frequency of usage, time betweens uses, and the criticality and rate of the data items that it influences. This is not something that is immediately obvious by looking at the logs, and our algorithm was very capable of revealing the most critical data items automatically. The most critical data items, in order of importance, are D, F, A, C, H, B, J, G, I, and then E, in that order.

3.6 Bidirectional Criticality

There are cases where information flow or influence can go two ways. For example in Figure 3 below data item A influenced data item B 89 times and data item B influenced A 44 times. In the cases where the influence is bidirectional, the above criticality calculation requires some minor adjustments. The original criticality graph is broken down into two sub graphs, and the bidirectional links are separated, one for each direction. Then, the criticality of the affected data items are calculated separately, resulting in the total criticality of the affected nodes.

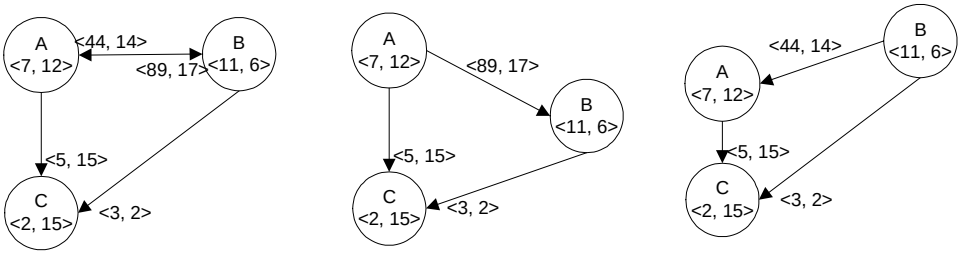

Fig 3: A criticality graph with a bidirectional edge is shown at left. It is decomposed into two sub criticality graphs, which are shown in the center and right.

4. Conclusions and Future Work

In this paper, we propose a new and novel method to detect critical data items in a database automatically. Our system is intended to complement existing intrusion detection systems to help fight the threat of malicious insiders abusing critical data items. We develop a quantitative framework that applies very well to determining which data items are critical in that they influence a significant amount of other data items that are significant themselves. Also, our algorithm takes into account the number of uses and the average time between uses in order to better model a realistic system. The advantage of our approach is that it is a flexible and resource efficient technique that can be applied to any system that maintains a log of the transactions that are operated on the database. While our approach is aimed to be used by system administrators in a defensive mode of operation, it is also applicable to individuals who wish to use it in an offensive mode in order to efficient target the areas of the data system where malicious actions will cause the most damage and disruption to the enemy.

We plan to further our work by extending the algorithm to the level of the user. A criticality graph will be developed for each user, and if the monitoring system can be tuned to be specific to particular users, better security can be achieved. As the criticality of certain documents is very different to different users, extending our work to that level will be a good and worthwhile improvement.

Acknowledgement

This work has been supported in part by US AFOSR under grant FA9550-08-1-0255. We are thankful to Dr. Robert Herklotz for his support, which made this work possible.

References

1. Ray, I., and Poolsappasit, N. Using Attack Trees to Identify Malicious Attacks from Authorized Insiders. *In ESORICS 2005*, Milan, Italy, pp. 231-246 (2005)
2. Hu,Y. and Panda, B. Identification of Malicious Transactions in Database Systems. In *ideas*,pp.329. 7th Intl. Database Engineering and App. Symposium (IDEAS'03) (2003)
3. Zuo, Y. and Panda, B. A Service Oriented System Based Information Flow Model for Damage Assessment. In *6th IFIP WG 11.5 Working Conference on Integrity and Internal Control in Information Systems*, Lausanne, Switzerland, November 13-14 (2003)
4. Cappelli, D., Moore, A., Shimeall, T, Trzeciak, R. "Common Sense Guide to Prevention and Detection of Insider Threats", Carnegie Mellon University (2008)
5. Insider Threat Integrated Process Team, Department of Defense (DoD-IPT), 2000. "DoD Insider Threat Mitigation" U.S. Department of Defense (2000)
6. Anderson R, Bozek T, Logstaff T, Meitzler W, Skroch M, Wyk KV. Research on mitigating the insider threat to information sys., RAND Corporation Report CF-163 (2000)
7. Whitman, M. "Enemy at the Gate: Threats to Information Security". In *Communications of the ACM*, Vol. 46, No. 8. (2003)
8. Abbadi, I., Alawneh, M. "Preventing Insider Information Leakage for Enterprises", In *Proceedings of the 2008 Second International Conference on Emerging Security Information, Systems and Technologies*, p.99-106 (2008)
9. Anderson, R. and Brackney, R. "Understanding the Insider Threat", In *Proceedings of a March 2004 Workshop*, RAND National Defense Research Institute (2004)
10. Ha, D., Upadhyaya, S., Ngo, H., Pramanik, S., Chinchani, R., Mathew, S. "Insider Threat Analysis Using Information Centric Modeling" P. Craiger and S. Shenoi (Eds.), In *Advances in Digital Forensics III*, Springer, Boston (2007)
11. Sheyner, O., Haines, J., Jha, S., Lippmann, R., Wing, J. "Automated Generation and Analysis of Attack Graphs," In *Proc. IEEE Symposium on Sec. and Priv.*, Oakland, (2002)
12. Cathey, R., Ma, L., Goharian, N., Grossman, D. "Misuse detection for information retrieval systems" In *CIKM '03: Proceedings of the twelfth international conference on Information and knowledge management*, pages 183–190, New York, NY, USA (2003)
13. White, J., Panda, B. "Implementing PII Honeytokens to Mitigate Against the Threat of Malicious Insiders", in *Proc. of the IEEE International Conference on Intelligence and Security Informatics (ISI 2009)*, Dallas, Texas, pp. 233, (2009)
14. White, J., Panda, B., Yaseen, Q., Nguyen K., Li, W. "Detecting Malicious Insider Threats using a Null Affinity Temporal Three Dimensional Matrix Relation", in *Proc. of the 7th Inl. Workshop on Security in Info. Sys. (WOSIS 2009)*, Milan, pp. 93 – 102, (2009)
15. Meza, B., Burns, P., Eavenson, M., Palaniswami, D., Sheth, A. "An ontological approach to the document access problem of insider threat." in *ISI Lecture Notes in Computer Science*, P. B. Kantor, G. Muresan, F. Roberts, D. D. Zeng, F.-Y. Wang, H. Chen, and R. C. Merkle, Eds., vol. 3495. Springer, pp. 486–491 (2005)
16. Bradford, P., Brown, M., Perdue, J., Self, B. "Towards proactive computer-system forensics". In *Proceedings of ITCC*, pages 648–652 (2004)