A SERVICE ORIENTED SYSTEM BASED INFORMATION FLOW MODEL FOR DAMAGE ASSESSMENT

Yanjun Zuo and Brajendra Panda
Department of Computer Science and Computer Engineering
University of Arkansas, Fayetteville, AR 72701
Email: {yzuo,bpanda}@uark.edu
Phone: (479) 575-2067, Fax: (478) 575-5339

Abstract: The process of damage assessment and recovery in case of an information attack is time consuming. Any delay during recovery process may lead to system unavailability and is unacceptable in many time-critical applications. In this research we have developed a model to reduce this delay and aid in prompt assessment of damage, which is essential for faster recovery. In case of an attack notification, the goal of this model is to identify the affected data items quickly without having to evaluate too many data items and then make the unaffected data items available as soon as possible. This model can be used along with other intrusion detection mechanisms.

Key words: Information flow, damage assessment

1. INTRODUCTION

Damage assessment and recovery are time consuming processes. They could lead to denial-of-service in some situations and that is unacceptable. Any uncertainty in intrusion response can make the situation worse [3]. Some of the recent developments on damage assessment and recovery from information attacks are ([6], [8], [7], [10], [11], [13]). This paper focuses on achieving faster damage assessment by analyzing the patterns of information flow within an organization (or closely related organizations), which we call a domain. A very common situation in a domain is the time lag between the moment when an attack took place and the time when the attack is actually detected. During this period, any other legitimate programs or transactions may still read damaged data items and use their values to update other data items, thus, affecting latter. In database systems, damage may also spread via other means as discussed in [1] and [4]. Before the recovery process it is necessary to make accurate evaluations to determine the sets of data items that have been damaged or the sets that are clean so that clean sets of data items can be made available to user while feeding the damaged sets to recovery module. This can effectively reduce system down time after an attack. The approach presented in this paper employs a simple method to make preliminary damage assessment and can be used with other intrusion response systems.

1.1 The Data Model

A domain is an abstract entity, which is made up of multiple units. A unit, denoted by U, is a multi-service entity involved in multiple types of services in a domain. Each type of service is associated with and supported by a set of relevant data items distributed in multiple units. A unit could be a business department, a functional office, or a service center within an organization. Each of these units communicates with other units in the same domain and share information with them. We consider each unit to consist of multiple abstract entities, called service groups (SGs). For the service group in unit U, which is involved in the type of service S, we denote it as $SG_U(S)$. Each SG owns and manipulates a group of relevant data items, which are involved only in that type of service. There may be a public group corresponding to a unit in the domain, which is kept in a public group area and is available for SGs of that unit. Each public group owns and manipulates public data items for the corresponding unit. That public group area in the domain keeps all public groups for all units. Figure 1 shows the structure of a domain, which consists of units U_1, U_2, ..., U_n, and a public group area containing public group 1 (PG_1) for unit 1, public group 2 (PG_2) for unit 2, and so on. Figure 2 illustrates the typical structure of a unit U, which has multiple service groups involved in service S_1, S_2, ..., S_n.

Data items in public groups can only be updated by authorized administrators in controlled environments. Within a domain any data item in a unit is only owned and manipulated by one service group (SG) and each service group corresponds to only one service type. Information in each public group (PG) may be released to the corresponding SGs of the unit or available for other PGs of units under administrative control. However, no information flow is allowed between any two SGs of the same unit.

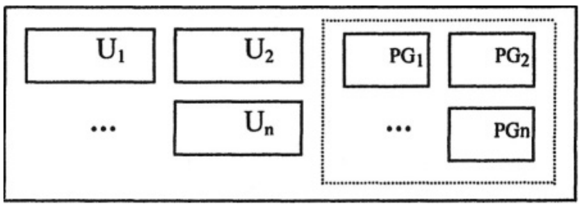

Figure 1: **The Structure of a Domain**

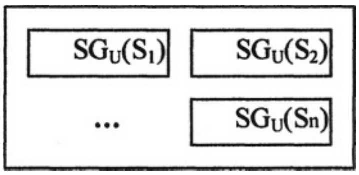

Figure 2: **The Structure of a Unit**

1.2 Separation of Service Data Items

The concepts of domain, unit and service group separate data items in a social network based on their services. Instead of grouping data items based on conflict of interests as done in the Chinese Wall Security Policy [2], our model separates data items based on relevant types of services provided in a domain. This restriction of possible information flows not only prevents manipulations of unnecessary data items but also helps perform quick damage assessment in case of an attack. The following examples would clarify the idea. Consider a domain such as a national administrative system, which consists of government bodies at federal, state, and city levels. Each government agency can be viewed as a unit. Each unit provides social services and manipulates data items related to military and civil sectors. There may be no (at least no need to allow) information flow from military sector to civil sector for each government unit. For those information needed by both sectors, a public group can be set up in the public group area within the domain. Information flow could happen from a civil service group in a state government unit to civil service groups in other state, federal or city government units, or vice versa. As another example consider a supply chain, which consists of wholesalers, retailers, customers, and vendors. Each component in this domain can be viewed as a unit. A wholesaler unit may have different service groups, each of which handles different product lines. For each product line, it has business connections with some vendors in that industry. There is no need to allow information flow between different service groups in this whole seller unit, say, from food group to baby cloth group. Common information such as financial data, or human resources data, can be retrieved from public group. The last example is about an academic body such as a university or a college, which consists of academic departments, administrative offices, and other agents. If we view each academic department as a unit, then it may have multiple abstract SGs, such as student academic group, faculty administrative group, computing facility group, and others. The department unit may keep administrative data such as faculty payroll information separated from student academic records and computer lab information from faculty research projects.

2. DAMAGE ASSESSMENT MODEL

The following definitions would be useful in explanation of the damage assessment model. The first definition has been taken from [9] with slight modification.

Definition 1: Data item *a* "can-influence" data item *b* if *a* is allowed to be used to update *b*. This permission to update is determined by the SG of the unit, which owns *a*, based on the SG's functional roles and policies or mutual agreements of relevant units. This relationship is denoted as $a \longrightarrow b$. If *b* also can-influence *a*, we denote the relationship as $a \longleftrightarrow b$.

Definition 2: $SG_{U1}(S)$ "can-influence" $SG_{U2}(S)$ if a data item of $SG_{U1}(S)$ can-influence a data item of $SG_{U2}(S)$, where U_1 and U_2 are two units within one domain and S is a type of service. This relationship is denoted as $SG_{U1}(S) \longrightarrow SG_{U2}(S)$. It must be ensured that this type of relationship only exists between SGs involved in the same type of service.

Property 1: It is obvious that $SG_{U1}(S_1)$ does not influence $SG_{U2}(S_2)$ and $SG_U(S_1)$ does not influence $SG_U(S_2)$. We denote these as $SG_{U1}(S_1) \not\longrightarrow SG_{U2}(S2)$ and $SG_U(S_1) \not\longrightarrow SG_U(S_2)$ respectively.

Definition 3: We use the representation $SG_{U1}(S) \blacklozenge\!\!\longrightarrow SG_{U2}(S)$ for actual information flow from $SG_{U1}(S)$ to $SG_{U2}(S)$. Moreover, when there is two way information flow between $SG_{U1}(S)$ and $SG_{U2}(S)$, we use the symbol $SG_{U1}(S) \longleftarrow\!\blacklozenge\!\longrightarrow SG_{U2}(S)$.

Information flow has the following characteristics:

1) Reflexive: $X \blacklozenge\!\!\longrightarrow X$;

2) Non-commutative: $X \blacklozenge\!\!\longrightarrow Y \,!\!\Rightarrow Y \blacklozenge\!\!\longrightarrow X$;

3) Transitive: $X \blacklozenge\!\!\longrightarrow Y$ and $Y \blacklozenge\!\!\longrightarrow Z \Rightarrow X \blacklozenge\!\!\longrightarrow Z$.

While the “can-influence” relationship is a direct reflection of units’ functional polices, the “information flow” relationship is the actual functional activity between two SGs. For example, if unit U_1 provides certain type of service S in a domain and allows unit U_2 to use its data items related to that service, then a “can-influence” relationship exists between the corresponding two SGs of U_1 and U_2 respectively. When unit U_2 actually updates its data items using information from unit U_1, then the information flow takes place (via SGs).

2.1 Probability Graph Analysis

In this model, we assume the attack has been detected by using some intrusion detection methods. The separation of service data items facilitates the speedup of the damage assessment process. If we can determine the service group, which a damaged data item belongs to, then any other data items belonging to different type of services can be made free and available to users.

2.1.1 Probability Graph

For those data items involved in the same type of service as the damaged one, a probability graph can be used to make quick damage assessments. We recognize the dynamic nature of a domain in a time span. Given a type of service S, each unit (or SG) may change its policies dynamically affecting the “can-influence” relationships with SGs of other units. A static period *t* for S refers to the time duration when no unit (or SG) changes its “can-influence” policies and the domain is in a static period. A time period refers to a static time period in the following sections unless stated otherwise. An evaluation time period for S is the time range from the time point when the attack started (as detected) until its effects were stopped (usually the time when the damage assessment process begins). During this period, legitimate transactions may still read dirty data.

Definition 4: A probability graph for a given type of service S in a time period t_1 is a directed graph representing can-influence relationships among SGs of different units in a domain involved in S. It is denoted as $Pt_1(S)$. A node in a probability graph denotes an SG of a unit and an edge represents can-influence relationship between the two connected nodes. An edge can be unidirectional or bi-directional. An unit has no more than one SG as a node in a probability graph for a given service type S and static time period t_1. Figures 3(a) and (b) show two probability graphs for the static time period t_1 and t_2 respectively.

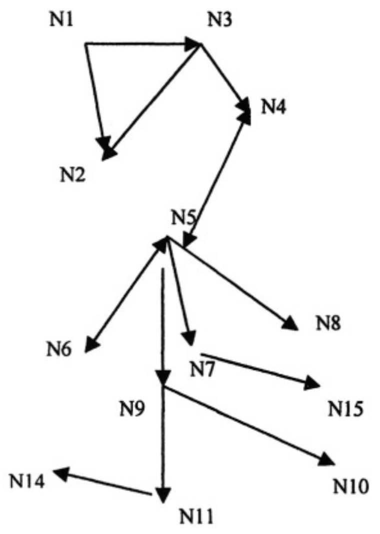

Figure 3(a): **A Probability Graph with Service S for Time Period t_1**

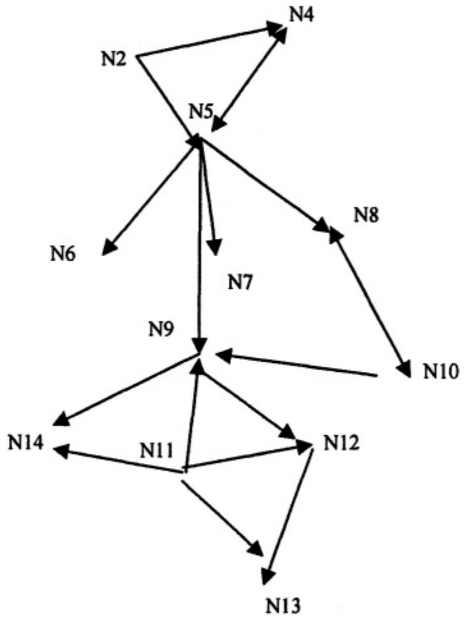

Figure 3(b): **A Probability Graph with Service S for Time Period t_2**

Definition 5: Pt(S) is a cumulative probability graph for an evaluation period t for a given type of service S and $Pt(S) = Pt_1(S) + Pt_2(S) + \ldots + Pt_k(S)$ based on the following rules:

1) Period $t_1, t_2, \ldots, t_k$ are continuous, $t = t_1 + t_2 + \ldots + t_k$, and t has the same beginning time as t_1 and the same ending time as t_k;
2) If a node N∈ Pm(S)'s node set, where $m = t_1, t_2, \ldots,$ or t_k, then N∈ Pt(S)'s node set;
3) If an edge E ∈ Pm(S)'s edge set, where $m = t_1, t_2, \ldots,$ or t_k, then E∈ Pt(S)'s edge set.

A cumulative probability graph reflects any possible "can-influence" relationships among units during a cumulative time period t for a given service type. It is calculated based on the probability graphs in each time period (these time periods are continuous). Figure 4 represents such a cumulative probability graph for the evaluation time period t, where t_1 and t_2 are continuous and t has the same starting time as t_1 and the same ending time as t_2, which is the end time of spreading of damage.

If we can effectively reduce the evaluation time period (e.g., detect the intrusion earlier), then the cumulative probability graph may be simpler. On the other hand, if a domain is less dynamic in term of changes of "can-influence" relationships, then the needs to accumulate probability graphs could also be reduced (or eliminated).

Definition 6: If G' is a sub graph of G (where G and G' are built based on the same time period), then the graph(s) G-G' contains only the nodes in G but not in G'. Further, all edges in G except those with a vertex in G' are still in G-G'. If a node N is in G, and {N} denotes a single-node graph, G-{N} will be the result if we delete N from G.

Property 2: If we consider graph accumulation as a special operation of graph "addition", as denoted as $G = G_1 + G_2$, then $G = G_1 + G_2 \ !\Rightarrow G_1 = G - G_2$.

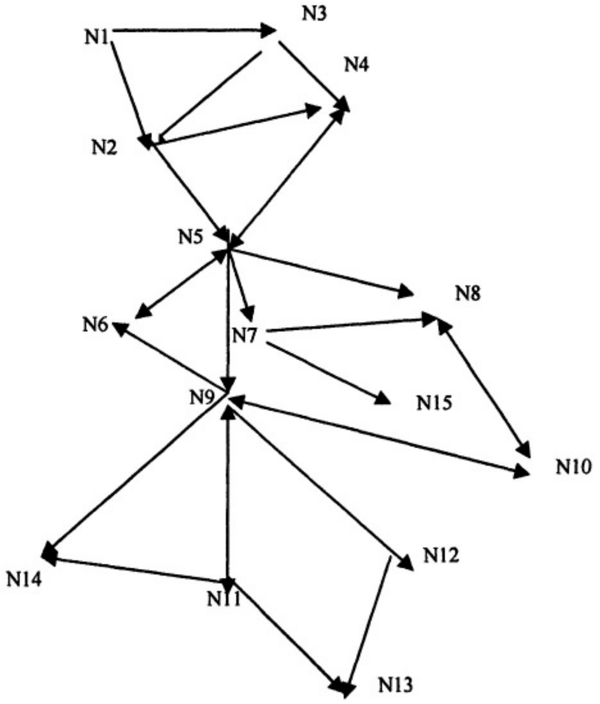

Figure 4: **A Cumulative Probability Graph with Service S for an Evaluation Time Period** $t = t_1 + t_2$**)**

Definition 7: A critical node in a probability graph or an information flow graph (see section 2.2) is a specific connecting point, removal of which may divide the graph into multiple connected graphs (the original graph is no longer connected). In Figure 4, nodes 5, 7, and 9 are critical nodes. A critical node in this paper is similar to an articulation point in a graph as described in [12]. Critical nodes in a graph can be found using the corresponding algorithm presented in [5]. But not every probability graph or an information flow graph has critical nodes. If there is a cycle in a graph visiting every node, then this graph has no critical node.

Definition 8: A distance of a critical node in a probability graph or an information flow graph is the number of nodes along the shortest path from the node (SG), which contains the initially detected damaged data item(s), to that critical node.

All the critical nodes for a given graph form a critical node set, denoted as CNS. We record CNS in increasing order. In Figure 4, the distance of node 5 from node1 (source of the damage) is 2, the distance of node 7 is 3, and the distance of node 9 is also 3. So CNS = {5, 7,9} or {5,9,7}.

Definition 9: A sector is a connected graph formed by nodes as obtained after the removal of a critical node from a probability graph. Several sectors may be resulted after removal of one critical node. The sector, which contains the initially known damaged data item, is called dirty sector. Figure 5 depicts two sectors produced after removal of critical node N9 from the graph shown in Figure 4. Sector 1 will be marked as a dirty sector if we identify the initially detected damaged data in N1, for example.

For any critical node, if we can determine it is free of damage, then all of the nodes it "can-influence" can be ensured not damaged. If the considered critical node cannot be eliminated, we then pick up the next critical node with smallest distance value and perform the same evaluation.

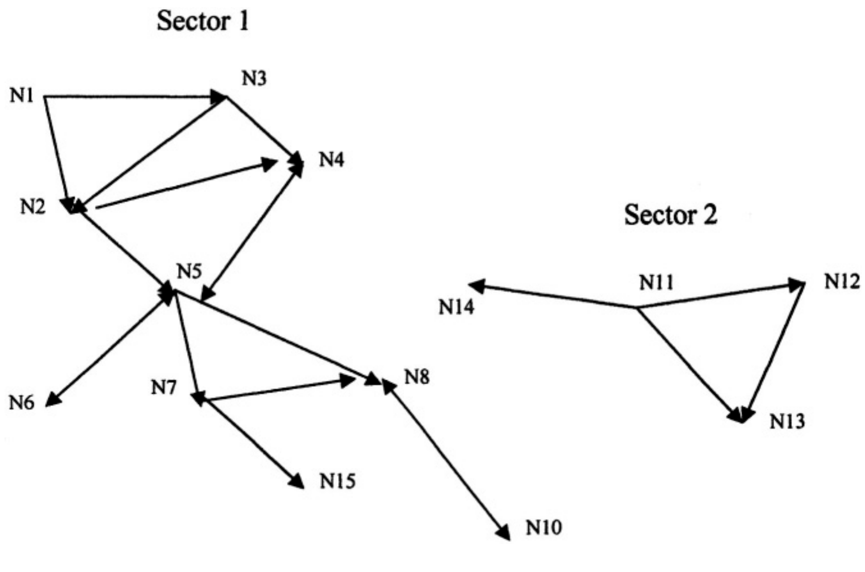

Figure 5: Two Sectors after Removal of N9 from Figure 4

2.1.2 Evaluation of a Critical Node

It is possible for a critical node to be quickly assessed to determine whether it contains damaged data items without reading all other nodes in a probability graph. This could be done by several means, such as by carefully studying the legitimate transactions and calculating the "should-be" values for data items in that critical node. Then comparisons can be made to decide the cleanness of that critical node. This paper introduces a method to determine whether any data item in a critical node has been damaged without analyzing other nodes.

2.1.2.1 Release Log and Update Log

For every critical node two logs are maintained: release log and update log. An entry in each log for a critical node has two fields, namely, SG_id and timestamp. However, these fields mean differently for the two logs. We use an example of a service group, *sg,* to clarify these concepts.

For *sg*'s update log, the first field, "SG_id", represents the ID of an SG, from which a data item was read earliest or latest by *sg*. We consider the possible earliest time since a base point when the domain has no damaged data item for the given type of service (e.g., organization startup time, or the last time when the intrusion was detected and cleaned). Before an entry is added, if there has been no more than one entry for the SG in *sg*'s update log, then the new entry is inserted. Otherwise, overwrite the most recent entry for the SG.

For *sg's* release log, the "SG_id" field represents the ID of an SG, into which a data item in *sg* flows most recently. Before an entry is added, if there is already an entry for the SG, then overwrite that entry using the most recent timestamp.

The second field, "timestamp" records the moment when the information flow takes place for each recorded entry in both release and update logs. The complete format for this field should be *year:month:date:hour:minute:second.* This paper uses the format *hour:minute* for simplicity.

Property 3: A node N*'s update log always keeps the earliest and the most recent time for incoming information flows from each other SGs (if there is only one entry for the SG, the earliest and the most recent time are the same). N*'s release log only keeps the most recent outgoing information flow from N* for each other node.

For each SG, we don't keep information flow for each individual data item. Rather we keep information for communications among SGs. An example is given in Table 1 to show the update log for node N9 based on Figure 4 and 5 as well as actual information flow records. In addition, we assume that an attack was detected as taking place at 11:00.

Table 1: Update Log for N9 in Figure 4

SG id	Timestamps
N5	7:00
N10	10:10
N10	9:00
N11	15:00
N11	6:00

2.1.2.2 Latest In-flow Timestamp

As mentioned previously, if a critical node N* in a probability graph or an information flow graph, G, is removed, several sectors are formed. In G, some nodes of a sector have incoming and outgoing edges from and/or to N*. We are interested in finding one parameter related to a sector *s* for N* as follow. Latest in-flow timestamp from *s* to N* is the largest timestamps in N*'s update log entries, which are associated with any node in *s*. This value represents the time for the most recent read operation by N* from a node in *s*. Based on Table 1 and Figures 4 and 5, L(in, N9, Sector 1) = max{7:00, 10:10, 9:00} = 10:10. Since this time is earlier than the time when the attack took place, which is 11:00, we can determine for sure that N9 is damage free. This way, preliminary assessment of a critical node is done.

2.1.2.3 Algorithm to Evaluate a Probability Graph

Based on the analysis for a critical node, there are two cases as discussed below in analyzing the probability graph (we use node N5 in Figure 4 as an example).

Case1: If we cannot evaluate any data item in N5 as damage free or we can determine at least one data item in N5 has been damaged, we will then move to the next critical node, such as N7.

Case 2: If we can determine that no data item in N5 was damaged during the evaluation time *t,* this will lead us to remove the critical node N5 (along with all its edges) resulting in two sectors as shown in Figure 5. Since sector 2 is not the dirty sector, we can determine all data items in sector 2 are clean. For sector 1, which contains the source of damaged data item, if it still contains critical nodes and one of these critical nodes has not been evaluated, then the same procedure is applied to that critical node. If there is no unevaluated critical node in the dirty sector, we stop. Next we present an algorithm that evaluates a probability graph to identify the damage.

Algorithm 1

```
CNS: a set of all ordered critical nodes in G based on their distance values
iNode = CNS[0];   //assigns the first node in CNS to an initial critical node to be analyzed
ESet={};  //ESet holds only evaluated critical nodes.  Initially it is empty
FreeSet={};  //FreeSet holds all of the nodes which are determined free of damage
G: the cumulative probability graph for an evaluation period t
(All the above variables have global scopes for the algorithm)

Eval_Prob_Graph(G, iNode)
{
 1 If iNode is Null then exit     //no critical node
 2 If iNode is damage free, then
     2.1  Add iNode to ESet
     2.2  Obtain sectors $s_1, s_2, \ldots s_k$ if iNode is removed from G
     2.3  For each $s_m$ in $\{s_1, s_2, \ldots, s_k\}$
          2.3.1  if $s_m$ is not the dirty sector, then
              2.3.1.1  Put all nodes of $s_m$ into FreeSet
              2.3.1.2  Put all critical nodes of $s_m$ into ESet
          2.3.2  else //$s_m$ is the dirty sector
          2.3.2.1   If any node N in $s_m$ has only an incoming edge from
                    iNode, put N into FreeSet
             2.3.2.2 $s_m = s_m - \{N\}$
             2.3.2.3 $G = s_m$
             2.3.2.4 Let Nod is the critical node in $s_m$ with smallest
                     distance value
             2.3.2.5 Call Eval_Prob_Graph(G, Nod)
 3 else  //iNode can not be determined damage free
    3.1  Add iNode to Eset
    3.2  Find next critical node Node in CNS but not in Eset (if no such
         node, set
            Node = Null)
    3.3  INode = Node
    3.4  Call Eval_Prob_Graph(G, iNode)
 } //end of Eval_Prob_Graph()
```

If any nodes are left with at least one critical node after the analysis of probability graph, then an information flow graph should be built and analyzed. Although probability graphs for each time period can be built well ahead of time, a cumulative probability graph for an evaluation period should be calculated whenever an attack is detected. This cumulative probability graph reflects possible information flows in this domain during the period between attacks took place and the attacking effects are stopped. Furthermore, we are only interested in part of this cumulative probability graph. When an attack took place and some data items in an SG are detected as damaged, this cumulative probability graph can eliminate any node, which has no direct or indirect information flow from the damaged SG. For example, if node A has only an outgoing edge, which is towards the damaged SG node, it can be eliminated from this cumulative probability graph.

2.2 Information Flow Graph Analysis

In this section, we analyze an information flow graph, which could be built after a probability graph.

2.2.1 Information Flow Graph

Definition 10: An information flow graph for a given type of service S in an evaluation period *t* is a directed graph representing actual information flow related to S in a domain. It is denoted as **IFt(S)**. A node in an information flow graph represents an SG and an edge represents information flow relationship between the two connected nodes. An edge can be unidirectional or bi-directional. An information flow graph is created based on actual information flow records kept by the domain. A critical node can be defined and found for an information flow graph in the same way as that for a probability graph.

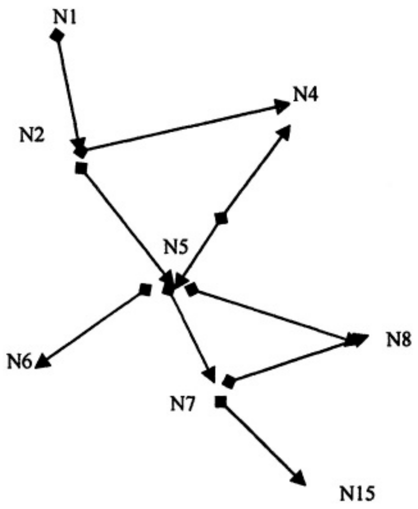

Figure 6: An Information Flow Graph

Consider the information flow graph in Figure 6 obtained by assuming that the critical node N9 in Figure 4 is determined damage free. Figure 7 shows three sectors if the critical node, N5, is removed from the graph in Figure 6.

Property 4: An information flow graph is a sub graph of the corresponding probability graph.

Property 5: Any critical node that remains in an information flow graph cannot be evaluated to be damage free.

Property 6: For a critical node N* in an information flow graph G', L(in, N*, DS) > *st,* where DS is the dirty sector after removal of N* from G', and *st* is the time when an attack was made. If this is not hold, then N* is damage free as shown in section 2.1.

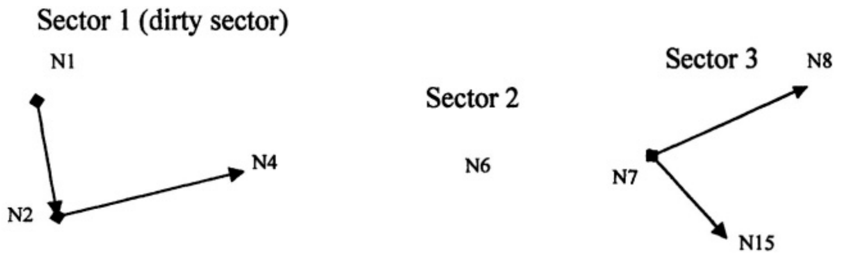

Figure 7: Three Sectors after Removal of Node N5

2.2.2 Analysis of an Information Flow Graph

Elimination of nodes based on critical nodes in an information flow graph is a complex process. An efficient analysis of the information flow and timestamp at each step is required to identify any clean nodes, if there are any.

Table 2: Release Log for N5 in Figure 6

SG id	Timestamp
N4	9:10
N7	11:15
N8	12:30
N6	10:10

Table 3: Update Log for N5 in Figure 6

SG id	Timestamps
N2	12:00
N2	13:15
N4	14:00
N4	15:30

An example of release log and update log for node N5 related to Figure 6 is given in Table 2 and Table 3 respectively.

In section 2.1, we discussed the concept "latest in-flow timestamp from *s* to N*", where *s* is a sector, which consists of multiple nodes, and N* is a critical node. We discuss other two concepts, "earliest in-flow timestamp from *s* to N* since the attack" and "latest out-flow timestamp from N* to *s*".

"Earliest in-flow timestamp from s to N* since the attack" is the smallest timestamp value (earliest time) in N*'s update log entries associated with an SG in *s*, which is later than the time when the attack occurred. It is denoted as E(in, N*, *s*). This parameter represents the earliest time when N* read a data item from an SG in *s* after the attack occurred. Related to Figures 6 and 7, E(in, N5, Sector 1) = min{12:00, 13:15, 14:00, 15:30} = 12:00, where {12:00, 13:15, 14:00, 15:30} is the set of all timestamps related to nodes in Sector 1 in Table 3 and the timestamp is later than the time of occurrence of attack (which is 11:00 as mentioned in 2.1).

Latest out-flow timestamp from N* to *s* is the largest timestamp value in N*'s release log entries, which are associated with an SG in *s*. It is denoted as L(out, N*, *s*). This parameter represents the latest time when N* wrote a data item to one SG in *s*. Related to Figures 6 and 7, L(out, N5, Sector 2) = max{10:10} = 10:10, where {10:10} is the set of all timestamps related to nodes in Sector 2 in Table 2. Similarly, L(out, N5, Sector 3) = max{1 1:15, 12:30) = 12:30, where {11:15, 12:30} is the set of all timestamps related to nodes in Sector 3 as shown in Table 2. Similar concepts can be applied to a node N relative to the critical node N*, such as E(in, N*, N) and L(out, N*, N).

There are two steps to analyze the sectors in Figure 7. The first step is to analyze a sector s as a whole. Relative to the critical node N*, if the latest read time for all nodes in *s* from N* is still earlier than the earliest write time from a node of the dirty sector to N*, we can guarantee that all nodes in *s* are clean. In Figures 6 and 7, since L(out, N5, Sector 2) = 10:10 and E(in, N5, Sector 1) = 12:00 as we calculated above, hence L(out, N5, Sector 2) < E(in, N5, Sector 1) and all nodes in Sector 2 are damage free. Hence, node N6 can be released (N6 is the only node in Sector 2). Because L(out, N5, Sector 3) > E(in, N5, Sector 1), however, we can't eliminate any node in Sector 3 at this time.

The second step follows if step 1 fails for a sector and it attempts to evaluate individual node in a sector. Any node, N, in a sector with a sole incoming edge from N* (in term of the original information flow graph), is analyzed first. If there is no such node, the analysis stops. If L(out, N*, N) < E(in, N*, DS), where DS is the dirty sector, then node N is determined free of damage. Then node N and all of its outgoing edges within that sector can be removed. Consider sector 3 in Figure 7. Since L(out, N5, N7) =11:15 found in Table 2 and E(in, N5, Sector 1) = 12:00 as calculated before, thus L(out, N5, N7) < E(in, N5, Sector1). So node N7 is damage free. Hence

N7 and its outgoing edges (from N7 to N8 and from N7 to N15) can be removed. For any other node in the considered sector, if it has only one incoming edge and that edge is from the removed node (in term of the original information flow graph), it can also be removed. Hence, node N15 in Figure 7 can be removed. However, we cannot remove N8 since N8 also has another incoming edge from N5 in addition to N7. Since L(out, N5, N8) = 12:30 found in Table 2 and E(in, N5, Sector 1) = 12:00 from Table 3, L(out, N5, N8) > E(in, N5, Sector 1). We cannot remove N8 during the information flow graph analysis. After this step, only node N8 is left in Sector 3. N8 (and its incoming edge from N5) should be forwarded for analysis by other means such as using information flow based on data items instead of SGs.

From the above example, we can deduce that for any "information flow chain",
$N1' \bullet\!\!\longrightarrow N2' \bullet\!\!\longrightarrow \ldots \bullet\!\!\longrightarrow Nk'$ in the information flow graph, if N1' can be removed, then N2', ..., Nk' can also be removed (such as N5, N7, N15 in Figure 7 as discussed above). The algorithm to analyze an information flow graph is given below.

<u>Algorithm 2</u>

```
CNS': a set of all ordered critical nodes in G' based on their distance values
iNode' = CNS'[0]; //assigns the first node in CNS' to the initial critical node to be analyzed
ESet'={}; //ESet holds the critical nodes already evaluated. Initially it is empty
FreeSet'={}; //FreeSet holds all of the nodes which are determined free of damage
G': the information flow graph for an evaluation period t
(All of the above variables have global scopes for the algorithm)

Eval_IF_Graph(G', iNode')
{
1  If iNode == Null then exit          //no critical node
   2  Add iNode into ESet'
   3  Obtain sector $s_1, s_2, \ldots, s_k$ if iNode is removed from G'
   4  Let $s_n$ = DS and DS is the dirty sector (1<= n <=k)
   5  For each $s_m$ in $\{s_1, s_2, \ldots, s_k\}$
      5.1  If (E(in, iNode', DS) > L (out, iNode', $s_m$))
           5.1.1 Add all nodes of $s_m$ to FreeSet'
           5.1.2 Add all critical nodes of $s_m$ to ESet'
      5.2  Else
           5.2.1  For any node $N \in s_m$, where N has only one incoming
                  edge in G', which is from iNode'
```

```
                5.2.1.1   If ( E(in, iNode', DS) > L(out, iNode', N))
                  5.2.1.1.1  Add N to FreeSet'
                 5.2.1.1.2   If there is a chain in $s_m$ with nodes N, N1,..., Nt
                             and information only flows from N to N1, N1 to
                             N2, ... and Nt-1 to Nt
                     5.2.1.1.2.1  Add N1, N2 ..., Nt to FreeSet'
                     5.2.1.1.2.2  $s_m = s_m - \{N, N1, N2 ..., Nt\}$
                     5.2.1.1.2.3  If any element N' in (N1, N2, ..., Nt} is a
                                  critical node
                       5.2.1.1.2.3.1   Add N'to ESet'
                  5.2.1.1.3  else $s_m = s_m - \{N\}$
            5.3  G' = G' – $s_m$
        6     For any critical node Node in CNS' but not in Eset' (if no such
              node, set Node = NULL)
          6.1  iNnode' = Node
          6.2 Call Eval_IF_Graph(G', iNode')
    }   //end of Eval_Graph(G', iNode')
```

3. USING THE MODEL

The goal of this model is to reduce the "denial of service" and free as many data items as possible in a timely manner. The general guideline to use this model is described below.

3.1 Preconditions

(a) Probability graphs $G_1, G_2, \ldots, G_k$ for time periods $t_1, t_2, \ldots, t_k$, where $t_1, t_2, \ldots, t_k$ are continuous in time line, i.e., without any time gap between two time periods. Usually, t_k ends at the time when the spread of damage is stopped.

(b) An evaluation time period $t = t_1 + t_2 + \ldots + t_k$ and t has the same beginning time as t_1 and the same ending time as t_k.

(c) The damaged data items are identified in a node, such as N1 (SG).

(d) Each critical node (SG) keeps an update and a release log for a time period covering *t*.

3.2 The Procedure to Use this Model

(a) Identify the type of service that the damaged data item is involved in. All the data items in SGs that are involved in other types of services are put into set FreeSet1. If the damaged data item is in a PG, then multiple probability graphs might be needed to make accurate assessment.

(b) For the service type that the damaged data item is related to, build the cumulative probability graph G for the time period *t* based on $G_1, G_2, \ldots, G_k$.

$G = G_1 + G_2 + \ldots + G_k$

Call Eval_Prob_Graph() algorithm to generate FreeSet2.

(c) Nodes in G – {all nodes of FreeSet2} are used to generate an information flow graph G' based on actual information flow information.

Call Eval_IF_Graph() algorithm to generate FreeSet3. A set T = G' – {all nodes of FreeSet3} is left.

3.3 Post Conditions

Any data items in a node of FreeSet1, FreeSet2, or FreeSet3 are guaranteed to be free of damage and then can be available to users immediately. Evaluating the set T is beyond the scope of this model and other methods should be used to analyze it.

4. CONCLUSIONS

The effects of this model are largely dependent on the structure of a domain. If there are some "central points" in the probability (and/or information flow) graph and information flow tends to be in one direction, then the model has a good chance to eliminate large sets of data items, which are free of damages, at the earliest. In many cases, organizations have such structures with a central point, which acts as a critical node. Evaluating this central point often can release sets of nodes that are damage free.

This model requires that there is no information flow across the boundary of multiple service groups (SGs) of the same domain. If several SGs must share information, we can either fuse them together into a bigger SG, or put the shared data items in the public group kept in one area of that domain.

ACKNOWLEDGEMENTS

We are thankful to Dr. Robert L. Herklotz for his support, which made this work possible. This work was supported in part by US AFOSR under grant F49620-01-10346.

REFERENCES

[1] P. Amman, S. Jajodia, C. D. McCollum, and B. Blaustein, "Surviving Information Warfare Attacks on Databases", In Proceedings of the 1997 IEEE Symposium on Security and Privacy, May 1997.

[2] D. Brewer and M. Nash, "The Chinese Wall Security Policy", In Proceedings of the 1989 IEEE Symposium on Security and Privacy, pp. 206-214, May 1989.

[3] C.A, Carver, J. M. D. Hill, and U. W. Pooch, "Limiting Uncertainty in Intrusion Response", in Proceedings of the 2001 IEEE Workshop on Information Assurance and Security, United States Military Academy, West Point, June 2001.

[4] R. Graubart, L. Schlipper, and C. McCollum, "Defending Database Management Systems Against Information Warfare Attacks", Technical report, The MITRE Corporation, 1996.

[5] E. Horowitz, S. Sahni, S. Rajasekaran, "Computer Algorithms/C++", Computer Science Press, pp. 329-336, 1998.

[6] S. Jajodia, C. D. McCollum, and P. Amman, "Trusted Recovery", Communications of the ACM, 42(7), pp. 71-75, July 1999.

[7] P. Liu and X. Hao, "Efficient Damage Assessment and Repair in Resilient Distributed Database Systems", In Proceedings of the 15th Annual IFIP WG 11.3 Working Conference on Database and Application Security, July 2001.

[8] P. Liu, P. Ammann, and S. Jajodia, "Rewriting Histories: Recovering from Malicious Transactions", Distributed and Parallel Databases, 8(1), pp. 7-40, January 2000.

[9] B. Panda, S. Tripathy, "Data Dependency Based Logging for Defensive Information Warfare", Proceedings of the 2000 ACM Symposium on Applied Computing.

[10] B. Panda and J. Giordano, "Reconstructing the Database After Electronic Attacks", Database Security XII: Status and Prospects, S. Jajodia (editor), Kluwer Academic Publishers, 1999.

[11] P. Ragothaman, and B. Panda, "Modeling and Analyzing Transaction Logging Protocols for Effective Damage Assessment", In Proceedings of the 16th Annual IFIP WG 11.3 Working Conference on Data and Application Security, King's College, University of Cambridge, UK, July 2002.

[12] R. Sedgewick, "Algorithms", Addison Wesley, pp. 324-330.

[13] R. Sobhan and B. Panda, "Reorganization of Database Log for Information Warfare Data Recovery", In Proceedings of the 15th Annual IFIP WG 11.3 Working Conference on Database and Application Security, Niagara on the Lake, Ontario, Canada, July 15-18, 2001.