Using Multiple-Valued Logic Decision Diagrams to Model System Threat Probabilities

Theodore W. Manikas, Mitchell A. Thornton
Department of Computer Science and Engineering
Southern Methodist University
Dallas, Texas, USA
{manikas, mitch}@lyle.smu.edu

David Y. Feinstein
Innoventions, Inc.
1045 Bissonnet Street
Houston, TX , USA
david@innoventions.com

***Abstract*—System security continues to be of increasing importance. To effectively address both natural and intentional threats to large systems, the threats must be cataloged and analyzed. Extremely large and complex systems can have an accordingly large number of threat scenarios. Simply listing the threats and devising countermeasures for each is ineffective and not efficient. We describe a threat cataloging methodology whereby a large number of threats can be efficiently cataloged and analyzed for common features. This allows countermeasures to be formulated that address a large number of threats that share common features. The methodology utilizes Multiple-Valued Logic for describing the state of a large system and a multiple-valued decision diagram (MDD) for the threat catalog and analysis.**

Large System Security, Threat cataloging, MDD

I. Introduction

In recent years, more emphasis has been placed on design for security, security assessment, disaster recovery, disaster tolerance, in addition to the ongoing efforts in the established areas in fault tolerance. All of these areas require the identification and analysis of faults or threats. Most fault models and analysis methods assume that a fault causes a system to either function or fail and are thus modeled with binary conditions.

Recent events have demonstrated our vulnerability to disasters, both natural and man-made. This motivates the need to incorporate disaster tolerance into large system designs. Disaster tolerance is a superset of the more established approaches commonly referred to as fault tolerance. Models for disaster tolerance differ from those for fault tolerance since they assume that failures can occur due to single or massive numbers of individual faults occurring simultaneously or in a rapidly cascading manner. Traditional fault tolerance models single points of failure. Therefore, a disaster-tolerant system can withstand a catastrophic failure and still function with some degree of normality [8, 15].

Threats on a system can include both accidental and intentional effects. Examples of accidental events include random component failures, natural disasters, and fires. Examples of intentional effects include sabotage and cyber threats. A variety of tree structures have been developed to represent possible system threats. These tree structures are commonly called fault trees or attack trees. Fault trees were originally developed to identify the effects of component failures on a system [16] while attack trees traditionally focused on the effects of cyber security breaches [14]. There have been recent variants of these trees, and sometimes these trees have been used interchangeably [2, 5, 6, 7, 11]. However, these trees all have one thing in common: they are binary-valued trees, connected with AND and OR operators. This limits the effectiveness of how they can represent possible system-wide attacks. Modeling different operational modes other than just the binary case of failure or normal operation are critical in analyzing large systems in the presence of threats. As an example, in the 2003 blackout of the US power grid, many complex interactions caused a blackout to occur in a large portion of the northeastern US; however, it would be incorrect to state that the entire US power grid failed. In order to effectively catalog system threats, it is necessary to determine the probabilities of these threats based on various system stimuli, including input conditions and their probabilities.

As part of our preliminary research on large-scale system threat assessment, we developed threat tree models [12]. Threat trees are a superset of fault and attack trees since they are based on multiple-valued (MV) or radix-p valued algebras over a finite and discrete set of values [10]. When the radix p=2, the threat tree reduces to a fault or attack tree depending on the nature of the disruptive events. Generally, threat trees have p>2: these additional logic states allow for more complicated interactions to be modeled. In particular, these additional states can represent partial failures or degraded performance in a system, which are critical in analyzing large systems in the presence of threats.

The structure of this paper is the following: First, background information on decision diagram models is provided. Next, an approach is presented using decision diagram models to determine system threat probabilities. Finally, a system example is used to illustrate these concepts.

II. Decision Diagrams

Fig. 1 shows an example of a simple binary fault tree where the circular nodes represent the event of a single component failure and the logic operators (AND/OR gates) show how the events combine to result in a subsystem failure. Referring to

Fig. 1, if one or more of events 1, 2, or 3 occur, subsystem A will fail. Alternatively, both events 4 and 5 must occur for subsystem A to fail. For practical systems, it is likely that a large number of components will need to be analyzed. As the system size grows (in terms of number of components and/or number of logic states), the system analysis process quickly becomes unwieldy. Therefore, an alternate system representation is required to make the analysis process more efficient.

Decision diagrams are well-suited for compact representation of a large number of threats and due to their canonical structure, efficient algorithms are formulated that analyze threats and identify those that pose the greatest threat to the system. Decision diagrams are rooted directed acyclic graphs (DAG) that can be used to represent large switching functions in an efficient manner. For binary-valued logic, the binary decision diagram (BDD) is a well-known structure [3] that has been applied to many areas including the representation of fault trees [13]. Furthermore, efficient software is readily available to manipulate BDDs.

In the case of *Multiple-Valued Logic* (MVL), an extension to the BDD construct has been developed and implemented called the *Multiple-Valued Decision Diagram* (MDD). Consider a totally-specified *p*-valued function with *n* inputs, $f(x_0, x_1, \ldots, x_n)$ where each dependent variable $x_i \in \{0, 1, \cdots p-1\}$. $f(x)$ can be efficiently represented by an MDD. As is similar to BDD, the MDD is also a DAG and it contains a maximum of *p* terminal nodes, where each terminal node is labeled by a distinct logic value in the range [0, *p*-1]. Every non-terminal node is labeled by an input variable and has *p* outgoing edges, where each edge corresponds to each logic value. MDD can be minimized using various techniques that were developed for BDD, thus allowing the representation of exceptionally large number of such functions [9].

III. Determining System Output Probabilities

For binary tree structures (fault and attack trees), BDD's have been applied to simplify the modeling threats on complex systems [1, 4, 5, 17, 18]. The common approach to determine probabilities on binary tree structures is to apply probability equations, such as those described in [6]. Assume we have n inputs, each with probability $P_{in}(x_i)\{1\}$, $i \in [1, n]$, where $P_{in}(x_i)\{1\}$= probability that input x_i is 1. If the inputs are mutually independent, then the output probability (probability that the output is 1) of an *n*-input AND gate is the sum of all the edges pointing to the terminal-1 vertex.

As an example, assume we have a 2-input AND gate with the following input probabilities: $P_{in}(x_0)= 0.5$, $P_{in}(x_1) = 0.75$. Then, $P_{out}\{1\}= (0.5)(0.75) = 0.375$.

While the separate application of probability equations has been commonly used for previous fault tree assessment methods, this approach can become inefficient for systems with large number of nodes. Since the decision diagram contains

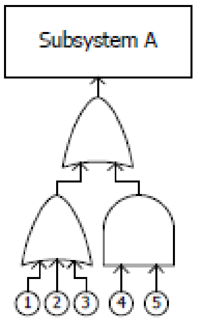

Figure 1. Fault tree example.

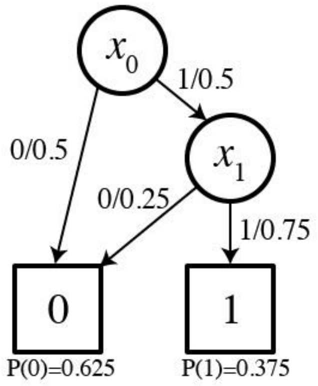

Figure 2. Binary AND BDD with probabilities.

edges with logic value weights, a more efficient approach would be to incorporate probabilities as edge weights, then calculate output probabilities by traversing the graph edges.

Using this approach, we can expand on the previous example as follows:

Input x_0: $Px_0[x_0=0] = 0.5$, $Px_0[x_0=1] = 0.5$

Input x_1: $Px_1[x_1=0] = 0.25$, $Px_1[x_1=1] = 0.75$

For the 2-input AND gate, $P_{out}\{1\} = Px_0[x_0=1]\, Px_1[x_1=1] = (0.5)(0.75) = 0.375$. Since this is a binary system, $P_{out}\{0\} = 1 - P_{out}\{1\} = 1 - 0.375 = 0.625$.

We can also determine the output probabilities using a BDD with probabilities incorporated into the edge weights. The BDD for this gate is shown in Fig. 2. The edge weights have the notation "V/P", where V = input value and P = probability of that input value. The output state probabilities $P_{out}[f=1]$ and $P_{out}[f=0]$ are calculated by traversing the edges of the BDD. This is done along all the paths from the root node to the output state nodes (0 and 1). The probabilities along each path are multiplied, and paths leading to the same output state node are added:

$P_{out}[f=0] = 0.5 + (0.25)(0.5) = 0.625$

$P_{out}[f=1] = (0.5)(0.75) = 0.375$

Since an MDD is an extension of a BDD, it should also hold that we can determine threat probabilities by applying the specified state probabilities as weights to the MDD edges and traversing the tree. Table I shows a system with three operational states and two components with given input probabilities. The system has the following truth table (Table II), which becomes the MDD shown in Fig. 3. Note that branch (1,2) from A to output state (2) adds the probabilities of the two events: 0.25 + 0.5 = 0.75. We get the following output probabilities by traversing the MDD. This is done along all the paths from the root node to the output state nodes (0, 1, and 2). The probabilities along each path are multiplied, and paths leading to the same output state node are added together:

$$P_{out}[f=0] = (0.2)(0.25) = 0.05$$

$$P_{out}[f=1] = (0.2)(0.5) + (0.3)(0.25) = 0.175$$

$$P_{out}[f=2] = (0.2)(0.25) + (0.3)(0.75) + 0.5 = 0.775$$

TABLE I. RADIX-3 SYSTEM EXAMPLE.

State	Component A	Component B
	Probability	Probability
(2) Operational	0.25	0.5
(1) Degraded	0.5	0.3
(0) Offline	0.25	0.2

TABLE II. TRUTH TABLE.FOR RADIX-3 EXAMPLE

A	B	f
0	0	0
0	1	1
0	2	2
1	0	1
1	1	2
1	2	2
2	X	2

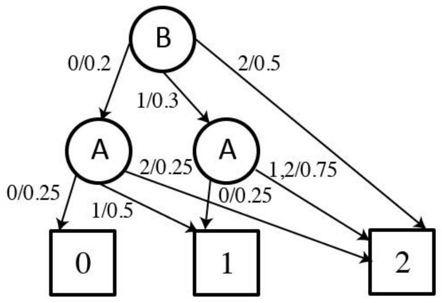

Figure 3. MDD for radix-3 example.

IV. ALGORITHM FOR OUTPUT PROBABILITY CALCULATIONS

Using the concepts described in the previous sections, we developed the algorithm shown in Fig. 4 to calculate system output probabilities for a given MDD. Initially, the output probabilities are set to zero. The algorithm starts at the root vertex of the MDD, then traverses the MDD using a depth-first search. A standard depth-first search on a graph with V vertices and E edges has a complexity of O(V+E) [19]. Each edge is traversed, and the corresponding probabilities are multiplied to form a working probability value until a terminal node is reached. At this point, the output probability corresponding to the terminal node is updated by adding the working probability value. When all vertices have been traversed, the output probabilities have been updated to their final values.

```
Given MDD as graph G(V,E) with vertices V and edges E

Begin Calc_Prob(G)
          Let p = 1.0                         // current probability value
          For j = 0 to (radix-1) do
                    Pout[j] = 0.0             // output probability values
          End for
          let u = root vertex of G
          DFS (u,p)                           // depth-first search on MDD
End Calc_Prob


Begin DFS(u,p)
          If u is a terminal node then
                    k = value of terminal node (0,1…,radix-1)
                    Pout[k] = Pout[k] + p     // update output probability value
          Else
                    For each child vertex v of u do
                              x = probability value on edge(u,v)

                              y = x*p         // multiply edge probability by current probability
                              DFS(v,y)        // do depth-first search starting at child v
                    End for
          End if
End DFS
```

Figure 4. Algorithm for output probability calculations.

V. EXAMPLE

Using the previously mentioned 2003 US power grid failure example, assume that we can represent a simple power grid by three operational states: fully operational, partially operational, and non-operational. This is a radix-3 system, so it cannot be represented by the traditional attack tree structure. However, it can be represented by our threat tree structure: state 2 = fully operational, state 1 = partially operational (degraded) and state 0 = non-operational. Also assume that the simple power grid system has three generation plants: coal, hydro, and wind. The total power available on the power grid is the sum of the power output produced by the coal, hydro and wind plants. Table III shows the threshold power values for fully operational, degraded, and non-operational states for the generation plants and total power grid. Using this information, an MVL truth table for this system is created (Table IV), where *f* represents the total power grid operational state. Note that "X" indicates a "don't-care" state, where the value can be either 0, 1, or 2. For example, row 4 in Table IV identifies the system state where the coal plant is non-operational (state 0: output = 0), the hydro plant is partially operational (state 1: output = 1500 MW) and the wind plant is fully operational (state 2: output = 200 MV). The total power grid output for this system state is 1700 MW, so the power grid is partially operational (f = 1).

TABLE III. OPERATIONAL STATES FOR POWER GRID EXAMPLE.

State	Total Power Grid	Coal Plant		Hydro Plant		Wind Plant	
	Total Power Available (MW)	Output	Probability	Output	Probability	Output	Probability
(2) Fully Operational	≥ 2400	1000	0.98	2000	0.99	200	0.6
(1) Degraded	1600 - 2400	600	0.018	1500	0.009	100	0.3
(0) Non-operational	<1600	0	0.002	0	0.001	0	0.1

The MDD for our example power grid system is shown in Fig. 5. The path along the MDD that corresponds to Row 4 in Table IV is indicated by the dashed arrows in Fig. 5.

TABLE IV. TRUTH TABLE FOR POWER GRID EXAMPLE

Coal	Hydro	Wind	f
0	0	X	0
0	1	0	0
0	1	1	1
0	1	2	1
0	2	X	1
1	0	X	0
1	1	X	1
1	2	X	2
2	0	X	0
2	1	X	2
2	2	X	2

Revisiting the earlier power grid example, we can now calculate the output probabilities for this radix-3 system. Note that the input probabilities from Table III are denoted in the MDD of Fig. 5. Applying the algorithm of Fig. 4, we obtain the following output probabilities:

$P_{out}[f{=}0]$ = (0.002)(0.009)(0.1) + (0.002)(0.001) + (0.018)(0.001) + (0.98)(0.001) = 0.001002

$P_{out}[f{=}1]$ = (0.002)(0.009)(0.3) + (0.002)(0.009)(0.6) + (0.018)(0.009) + (0.002)(0.99) = 0.002158

$P_{out}[f{=}2]$ = (0.98)(0.009) + (0.018)(0.99) + (0.98)(0.99) = 0.99684

The output probability results indicate that this system has a high probability of being fully functional in the presence of threat conditions. The specific output probabilities can also be used to classify the particular threats to the system.

Note that the MDD allows for good scaling when the problem sizes increase. Analysis may be further refined by extending the system radix above 3 and using the same MDD methodology.

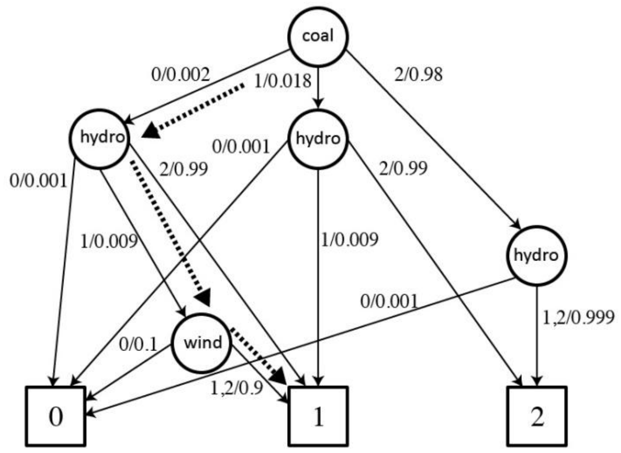

Figure 5. MDD for power grid example.

VI. CONCLUSION

The determination of system threat probabilities is an important component of system classification and threat cataloging. We have shown how to model system threat probabilities using edge weights on MDD's, which is a more efficient approach than the current method of developing separate probability equations for each possible output threat scenario. It allows easier determination of overall system state probabilities and it can accommodate complex systems with the efficient scalability of modern MDD packages. The framework discussed in this paper can be further applied to risk analysis which is useful in the determination of the initial system element probability values.

In terms of augmenting large systems to make them more disaster tolerant, we intend to develop further analysis methods based on threat trees. Because threat trees inherently combine common subtree structures, we note that a tree representing a collection of threats automatically yields common characteristics of subsets of threat by combining common subpaths. Such common subpaths correspond to similar characteristics among subsets of threats and they may be used to determine which parts of the system to augment in order to address a maximum number of threats. Finally, another area for further research is how to model threat probabilities that are not mutually exclusive, such as conditional probabilities, and determine how to incorporate these into the MDD structure.

VII. ACKNOWLEDGMENT

This research was partially supported by the U.S. Office of Naval Research (ONR) project N000140910784.

REFERENCES

[1] L.M. Bartlett and J.D. Andrews, "Choosing a heuristic for the "fault tree to binary decision diagram" conversion, using neural networks," *IEEE Trans. Reliability*, vol.51, pp. 344- 349, Sept. 2002.

[2] S. Bistarelli, P. Peretti, and I. Trubitsyna, "Analyzing Security Scenarios Using Defence Trees and Answer Set Programming", *Proc. 3rd Int. Workshop on Security and Trust Management (STM 2007)*, 22 February 2008, pp. 121-129.

[3] R.E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation", *IEEE Trans.Computers*, vol. C-35, pp. 677-691, Aug. 1986.

[4] Y.R. Chang, S.V. Amari, and S.Y. Kuo, "OBDD-based evaluation of reliability and importance measures for multistate systems subject to imperfect fault coverage," *IEEE Trans. Dependable and Secure Computing*, vol.2, pp. 336- 347, Oct.-Dec. 2005.

[5] S. Contini, G.G.M. Cojazzi, and G. Renda, "On the use of non-coherent fault trees in safety and security studies", *17th European Safety and Reliability Conf.*, Dec. 2008, pp. 1886-1895.

[6] I.N. Fovino, M. Masera, and A. De Cian, "Integrating cyber attacks within fault trees", *18th European Safety and Reliability Conf.*, Sept. 2009, pp. 1394-1402.

[7] L. Grunske and D. Joyce, "Quantitative risk-based security prediction for component-based systems with explicitly modeled attack profiles", *J. Systems and Software*, vol. 81, pp. 1327-1345, Aug. 2008.

[8] M. A. Harper, C. M. Lawler, and M. A. and Thornton, "IT Application Downtime, Executive Visibility and Disaster Tolerant Computing". *Proc. Int. Conf. on Cybernetics and Information Technologies, Systems and Applications (CITSA 2005), and Int. Conf. on Information Systems Analysis and Synthesis (ISAS*), 2005, pp. 165-170.

[9] D. M. Miller, and R Drechsler, "Implementing a multiple-valued decision diagram package," *Proc. 28th IEEE Int. Symp. on Multiple-valued Logic*, 1998, pp. 52-57.

[10] D. M. Miller. and M. A. Thornton, *Multiple Valued Logic: Concepts and Representations*, Morgan & Claypool Publishers, ISBN 10-1598291904, 2007.

[11] A. L. Opdahl and G. Sindre, "Experimental comparison of attack trees and misuse cases for security threat identification", *Information and Software Technology*, vol. 51, pp. 916-932, May 2009.

[12] P. Ongsakorn, K. Turney, M. A. Thornton, S. Nair, S. A. Szygenda, and T. W. Manikas, "Cyber Threat Trees for Large System Threat Cataloging and Analysis," *Proc. IEEE Int. Systems Conf.*, 2010, pp. 610-615.

[13] R. Remenyte and J. D. Andrews, "A simple component connection approach for fault tree conversion to binary decision diagram," *Proc. 1st Int. Conf. on Availability, Reliability and Security*, April 2006.

[14] B. Schneier., "Attack Trees: Modeling Security Threats," *Dr. Dobb's Journal*, Dec. 1999.

[15] S. A. Szygenda and M.A. Thornton, "Disaster Tolerant Computing and Communications", *Proc. Int. Conf. on Cybernetics and Information Technologies, Systems and Applications (CITSA 2005), and Int. Conf. on Information Systems Analysis and Synthesis (ISAS)*, 2005, pp. 171-173.

[16] W. E.Vesely, F. F. Goldberg, N. H. Roberts, and D. F. Haasl, "Fault tree handbook," NUREG-0492, U.S. Nuclear Regulatory Commission, Jan. 1981.

[17] L. Xing; and Y. Dai, "A New Decision-Diagram-Based Method for Efficient Analysis on Multistate Systems," *IEEE Trans. Dependable and Secure Computing*, vol.6, pp.161-174, July-Sept. 2009.

[18] O. Yevkin, "Truncation approach with the decomposition method for system reliability analysis," *Reliability and Maintainability Symp.*, 2009. pp.430-435.

[19] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*. 1990: MIT Press.

.