Scalable Optimal Countermeasure Selection using Implicit Enumeration on Attack Countermeasure Trees

Arpan Roy[‡], Dong Seong Kim[§] and Kishor S. Trivedi[‡]
[‡]*Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708, USA*
[§]*Department of Computer Science and Software Engineering, University of Canterbury, Christchurch 8140, New Zealand*
Email: arpan.roy@duke.edu, dongseong.kim@canterbury.ac.nz, kst@ee.duke.edu

***Abstract*—Constraints such as limited security investment cost precludes a security decision maker from implementing all possible countermeasures in a system. Existing analytical model-based security optimization strategies do not prevail for the following reasons: (i) none of these model-based methods offer a way to find optimal security solution in the absence of probability assignments to the model, (ii) methods scale badly as size of the system to model increases and (iii) some methods suffer as they use attack trees (AT) whose structure does not allow for the inclusion of countermeasures while others translate the non-state-space model (e.g., attack response tree) into a state-space model hence causing state-space explosion.**

In this paper, we use a novel AT paradigm called attack countermeasure tree (ACT) whose structure takes into account attacks as well as countermeasures (in the form of detection and mitigation events). We use greedy and branch and bound techniques to study several objective functions with goals such as minimizing the number of countermeasures, security investment cost in the ACT and maximizing the benefit from implementing a certain countermeasure set in the ACT under different constraints. We cast each optimization problem into an integer programming problem which also allows us to find optimal solution even in the absence of probability assignments to the model. Our method scales well for large ACTs and we compare its efficiency with other approaches.

***Keywords*-attack countermeasure tree, branch and bound, integer programming, optimization, security investment cost.**

I. Introduction

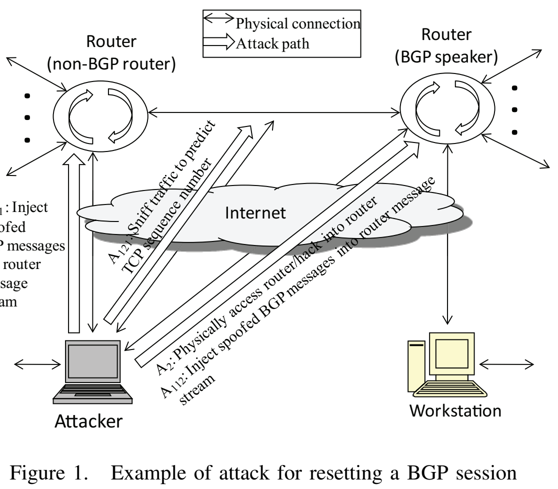
Security modeling is useful to assess security of a system. The first step towards security modeling is to design and construct a scalable model [1], [2] that helps quantify security [3] in terms of key attributes such as the loss caused by an attack or the gain accrued from enforcing a certain set of countermeasures [4]. This will aid not only in probabilistic risk analysis [5] but also in the development of a scheme as to where security investment should be prioritized in the system. The simplest model type in this context is attack tree (AT) [1]. Dewri *et. al* [6] utilized genetic algorithms to find optimal countermeasure sets for a system from their AT models. However, the basic formalism of AT does not take into account defense mechanisms. In earlier work [7], we developed a novel attack tree model called attack countermeasure tree (ACT). In ACT, (i) defense mechanisms can be placed at any node of the tree, not just at the leaf nodes, (ii) generation and analysis of attack scenarios and attack-countermeasure scenarios are automated using mincuts and (iii) security analysis (using measures such as attack and security investment cost, system risk, impact of an attack, return on attack (ROA) and return on investment (ROI)) is performed in an integrated manner. Zonouz *et. al* [8] proposed attack response trees (ARTs) in which both attacks and response events can be placed at any node of the tree but ARTs suffer from the state-space explosion problem due to the use of partially observable Markov decision process as a solution technique. In this paper, we show how to find optimal countermeasures using single objective optimization directly on ACT. More specifically, we show

- some analysis and optimization can be done without making probability assignments since many people believe it is hard or impossible to obtain probability values for attack, detection and mitigation events.
- how an optimal security countermeasure set can be selected from the pool of defense mechanisms using a non-state-space approach which is much less expensive than the state-space approach [8].
- greedy strategies and implicit enumeration techniques (branch and bound) are used to compute the optimal countermeasure set for various objective functions under different constraints and our work is compared with the approaches using genetic algorithms in [6] and ART-based state-space approach in [8].
- we present our method for automated generation of ACTs and discuss how our proposed optimization techniques scale for large ACTs.

The remainder of this paper is organized as follows. A brief overview of ACT is presented in Section II. In Section III, optimal countermeasure set selection on ACT without making any probability assignments is presented. In Section IV, optimal countermeasure set selection with probability assignments is described. We show results and their implications in Section V. Related work is presented in Section VI. Some simulation results and the impact of optimal countermeasure selection using ACT on security analysis are discussed in Section VII. Finally, we conclude the paper in Section VIII.

II. An Overview of Attack Countermeasure Trees (ACT)

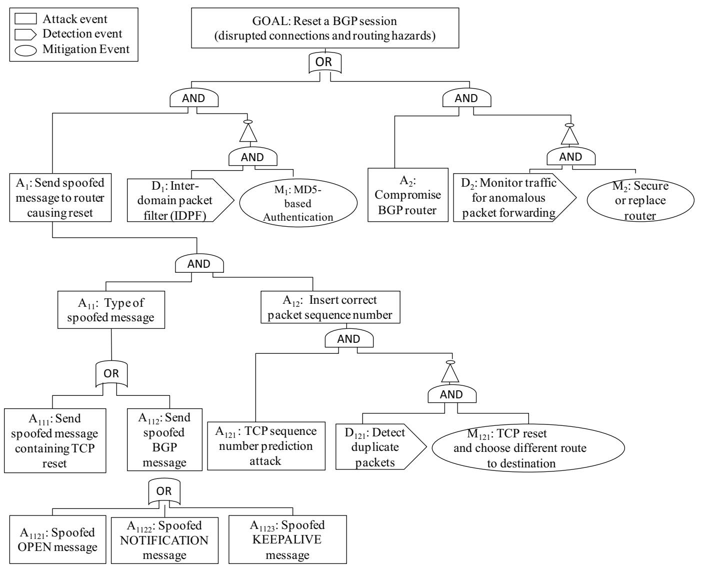
In ACT, there are three distinct classes of events: atomic attack events (e.g., install a keystroke logger), detection events (e.g., detect the keystroke logger) and mitigation events (e.g., remove the keystroke logger). An example scenario for a BGP (Border Gateway Protocol) attack is shown in Figure 1. An attacker prevents two peers from exchanging routing information by repeatedly causing a BGP session in Established state to reset. The BGP session can be reset by injecting a spoofed TCP (Transmission Control Protocol) or BGP message into the router message stream. Such spoofed packets can often be detected by methods such as the Inter-domain packet filter (IDPF) [9] and mitigated by adding an MD5 (Message-Digest algorithm) based authentication for packets from the source host of the spoofed packet. Building a valid TCP/BGP packet requires a valid TCP sequence number (obtained by TCP sequence number prediction). During the initial stages of a TCP sequence number attack, a spoofed packet from an attacker is usually followed by the original packet from the authentic source. Detecting such duplicate packets can be a giveaway for on-going TCP sequence number attacks. Dropping compromised connections and initiating a new connection to destination with a different route will mitigate such attacks. Spoofed TCP message with RST flag set will cause a connection to reset. Spoofed BGP messages (OPEN, NOTIFICATION or KEEPALIVE messages) received by the BGP speaker in the Connect or Active states will cause the router to reset resulting in a denial of service. The BGP speaker can also be compromised by gaining physical or logical (hijacking a router management session) access to the router. Usually router hijacking is characterized by anomalous packet forwarding [10] which can be detected by traffic monitoring at the router and mitigated by securing or replacing the router.



Physical connection
Attack path
Router (non-BGP router)
Router (BGP speaker)
Internet
A111: Inject spoofed TCP messages into router message stream
A121: Sniff traffic to predict TCP sequence number
A2: Physically access router/hack into router
A112: Inject spoofed BGP messages into router message stream
Attacker
Workstation

Figure 1. Example of attack for resetting a BGP session

We show an ACT for BGP attack by resetting a BGP session [11] and its countermeasures [12] in Figure 2. Among others, countermeasures include traceroute as one of the detection mechanisms for spoofed TCP reset messages and sequence number randomization as the corresponding mitigation techniques.



Attack event
Detection event
Mitigation Event
GOAL: Reset a BGP session (disrupted connections and routing hazards)
OR
AND
AND
AND
AND
A1: Send spoofed message to router causing reset
D1: Inter-domain packet filter (IDPF)
M1: MD5-based Authentication
A2: Compromise BGP router
D2: Monitor traffic for anomalous packet forwarding
M2: Secure or replace router
AND
A11: Type of spoofed message
A12: Insert correct packet sequence number
OR
AND
AND
A111: Send spoofed message containing TCP reset
A112: Send spoofed BGP message
A121: TCP sequence number prediction attack
D121: Detect duplicate packets
M121: TCP reset and choose different route to destination
OR
A1121: Spoofed OPEN message
A1122: Spoofed NOTIFICATION message
A1123: Spoofed KEEPALIVE message

Figure 2. An ACT for attacks involving resetting a BGP session and their countermeasures

An ACT for compromising a SCADA (supervisory control and data acquisition) system is shown in Figure 3 [8]. The ACT may have one or more repeated attack, detection and mitigation events (which means an event appears two or more times in the tree). For instance, in Figure 3 the attack event 'Unavailable LAN' occurs twice in the tree and contributes to both events: 'Power loads not provided' and 'Incorrect estimates to customers'.

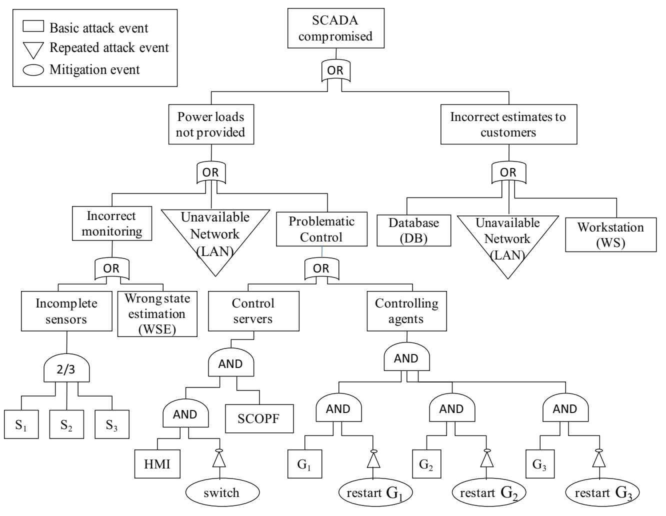


Figure 3. An ACT for SCADA system

Some notations relevant to ACT are listed below.

A_k	an atomic or leaf attack event
D_k	a detection event
M_k	a mitigation event
CM_k	a countermeasure event (which combines a detection and a mitigation event)
p_{A_k}	probability of occurrence of an atomic attack A_k
p_{D_k}	probability of success of a detection event D_k
p_{M_k}	probability of success of a mitigation event M_k
P_{goal}	probability of attack success at the ACT goal
i_{A_k}	impact of an atomic attack event A_k
I_{goal}	impact at the goal node of ACT
c_{A_k}	cost of an atomic attack event A_k
$C_{attacker}$	attack cost at the goal node of ACT
c_{CM_k}	security investment cost of a countermeasure CM_k

III. Optimal Countermeasure Selection without probability assignments

Many security researchers believe that it is hard to obtain the probability of an attack, detection and mitigation event. In this section, we discuss selection of optimal countermeasure set without making any probability assignments for the events in an ACT. We start to carry out analysis using the minimal cut-sets (in short, mincut from now on) of an ACT. A cut-set in an ACT is a set of leaf events whose occurrence ensures that the top event occurs. A cut-set in an AT represents an attack scenario whereas that in an ACT represent an attack-countermeasure scenario. A cut-set is said to be *minimal* if it cannot be reduced further without losing its status as a cut-set. There are well known algorithms for finding all the mincuts and they have been included in software packages such as SHARPE [13]. In ACT, the top event is associated with the set of all mincuts. The mincuts (i.e., attack countermeasure scenarios) of the ACT in Figure 2 are $\{(A_{111}, \overline{CM_1}, A_{121}, \overline{CM_{121}}), (A_{1121}, \overline{CM_1}, A_{121}, \overline{CM_{121}}), (A_{1122}, \overline{CM_1}, A_{121}, \overline{CM_{121}}), (A_{1123}, \overline{CM_1}, A_{121}, \overline{CM_{121}}), (A_2, \overline{CM_2})\}$ (where $\overline{CM_1}$=$(\overline{D_1 M_1})$, $\overline{CM_{121}}$=$(\overline{D_{121} M_{121}})$, $\overline{CM_2}$=$(\overline{D_2 M_2})$). Each of the 5 mincuts represents a combination of events to achieve attack success at the goal. For instance the mincut $(A_{1122}, \overline{CM_1}, A_{121}, \overline{CM_{121}})$ indicates that if both the atomic attack events A_{1122} and A_{121} occur and if both the countermeasures CM_1 and CM_{121} fail, attack will succeed. From the mincut $(A_{1122}, \overline{CM_1}, A_{121}, \overline{CM_{121}})$, we observe that the pair of atomic attack events (A_{1122}, A_{121}) is covered by either the countermeasure CM_1 or CM_{121}. SHARPE is used to generate the mincuts of the ACT. We show the optimal countermeasure set(s) selection to cover all atomic attack events by using minimum number of countermeasures as an objective function in an ACT in Section III-A. We discuss cost and impact analysis using ACT in Section III-B and we show optimal countermeasure set selection using minimization of the security investment cost as an objective function in Section III-C.

A. Selection of minimum number of countermeasures in ACT

A system administrator (or security decision maker) has to work within a given budget constraint which may preclude one from implementing all possible countermeasures. So we first present an optimization procedure that minimizes the number of countermeasures used in an ACT. In the following subsections, we present different ways of formulating this optimization problem using the BGP ACT [11] in Figure 2 and the SCADA ACT [8] in Figure 3 as examples. We show mainly two cases: full cover and partial cover of atomic attack events.

1) Full cover of atomic attack events: The full cover of ACT is achieved if the set of countermeasures in the optimal set ($\mathcal{OPT}$) covers all the atomic attack events. As discussed earlier, a countermeasure in a mincut covers every atomic attack event in that mincut. A matrix (T) is generated from the ACT mincuts where the columns are the countermeasures (CM_j) and the rows are the atomic attack events (A_i). Instances of matrix T are shown in Figure 4. If CM_j covers A_i, t_{ij} ((i,j)th entry in T) = 1 else t_{ij} = 0. To find the optimal countermeasure set, we minimize the number of columns to be selected from the matrix T, subject to the constraint that each row (atomic attack event) is covered by at least one column (countermeasure). The optimization problem is considered as a linear zero-one integer programming problem [14]. Let $x_j = 1$ if the countermeasure CM_j is selected, otherwise $x_j = 0$. Then the objective is to minimize

$$F_1 = \sum_{j=1}^{n} x_j \tag{1}$$

by selecting $x_j \in \{0, 1\} \forall j$ subject to:

$$\forall A_i \in \mathcal{A}, \sum_{j=1}^{n} t_{ij} \times x_j \geq 1 \tag{2}$$

i.e., covered set = $\mathcal{A}$, where $\mathcal{A} = \{A_1, A_2, A_3, ..., A_m\}$ is the set of all atomic attack events, $\mathcal{CM} = \{CM_1, CM_2, CM_3, ..., CM_n\}$ is the set of all countermeasures and n=$|\mathcal{CM}|$.

This optimization problem is a special case of the set cover problem [15] or the 'unate covering problem' [16]. To compute the cover, we improve on the greedy algorithm presented in [17]. Reduction techniques are first applied to the T matrix. Some terminologies relevant to the reduction are as follows:

- *Essential Columns*: Columns that contain the only non-zero entry of a certain row.
- *Row Dominance*: Row i is said to dominate row j if row i contains at least all the 1-entries in row j. Row i is the dominating row and row j is the dominated row.
- *Column Dominance*: Column i is said to dominate column j if column i contains at least all the 1-entries

in column j. Column i is the dominating column and column j is the dominated column.

Instances of essential columns and column dominance are shown in Figure 4. The procedure for reduction of the matrix is described in Algorithm 1.

Algorithm 1 Reduction of matrix T

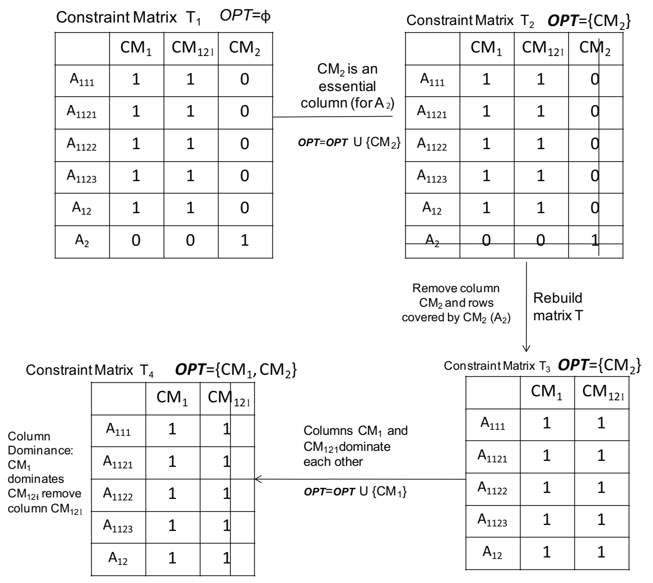
```
 REDUCE($\mathcal{OPT}$, T) {
1. do {
2.   find *essential columns* {$CM_1$,...,$CM_p$} in T
     remove essential columns {$CM_1$,...,$CM_p$} from T
     remove rows {$A_1$,...,$A_q$} covered by the essential
     columns {$CM_1$,...,$CM_p$} from matrix T
     $\mathcal{OPT}$=$\mathcal{OPT} \bigcup$ {$CM_1$,...,$CM_p$}
     rebuild matrix T
3.   check for *row dominance*$^*$
     remove dominating rows {$A_1$,...,$A_r$} from matrix T
     rebuild matrix T
4.   check for *column dominance*$^*$
     remove dominated columns {$CM_1$,...,$CM_s$}
     rebuild matrix T
     } while(reduction occurred in T)
5. return ($\mathcal{OPT}$, T)
}
```

A sample run of the procedure REDUCE($\mathcal{OPT}$,T) on the matrix of the BGP ACT in Figure 2 is shown in Figure 4 with the optimal solution $\{CM_1$,$CM_2\}$. The same reduction technique is used by the Quine-McCluskey method for finding the smallest set of prime implicants that cover a given Boolean sum of minterms [16]. Sometimes the set covering problem is easy to solve, but this is not always true (for instance, when matrix T is reducible to a cyclic matrix). We will deal with such cases later. Further, we may not be always able to cover all atomic attack events due to various reasons such as budgetary constraints, inadequate number of countermeasures and so on. This case (partial cover of atomic attack events) is discussed in the next subsection.

2) Partial cover of atomic attack events: A system administrator is interested in finding the best possible way of defending only the critical sectors of their system. This type of optimization problem is termed as 'partial cover'. Based on the nature of motivation, partial cover may be of two types: partial cover with/without intent.

(a) *Partial cover with intent*: A system administrator has to work within a given budget constraint which may preclude him from implementing all the countermeasures required for full cover. If given the cost constraint, the countermeasures can cover only a subset of all the attack events in the ACT (we call this subset the 'critical vulnerability set' or in short, $\mathcal{CVS}$) then this case is said to be partial cover *with intent*.

(b) *Partial cover without intent*: When the countermeasures provided are insufficient in covering all the attack events (e.g., as in case of the SCADA ACT, in some cases we can not find all the countermeasures against attacks even though we have enough security investment cost), the system administrator is said to be performing a partial cover *without intent.*



Constraint Matrix T_1 *OPT*=ϕ
CM_2 is an essential column (for A_2)
OPT=*OPT* U {CM_2}
Constraint Matrix T_2 ***OPT***={CM_2}
Remove column CM_2 and rows covered by CM_2 (A_2)
Rebuild matrix T
Constraint Matrix T_3 ***OPT***={CM_2}
Columns CM_1 and CM_{121} dominate each other
OPT=*OPT* U {CM_1}
Constraint Matrix T_4 ***OPT***={CM_1, CM_2}
Column Dominance: CM_1 dominates CM_{121} remove column CM_{121}

	CM_1	CM_{121}	CM_2
A_{111}	1	1	0
A_{1121}	1	1	0
A_{1122}	1	1	0
A_{1123}	1	1	0
A_{12}	1	1	0
A_2	0	0	1

	CM_1	CM_{121}	CM_2
A_{111}	1	1	0
A_{1121}	1	1	0
A_{1122}	1	1	0
A_{1123}	1	1	0
A_{12}	1	1	0
~~A_2~~	~~0~~	~~0~~	~~1~~

	CM_1	CM_{121}
A_{111}	1	1
A_{1121}	1	1
A_{1122}	1	1
A_{1123}	1	1
A_{12}	1	1

	CM_1	CM_{121}
A_{111}	1	1
A_{1121}	1	1
A_{1122}	1	1
A_{1123}	1	1
A_{12}	1	1

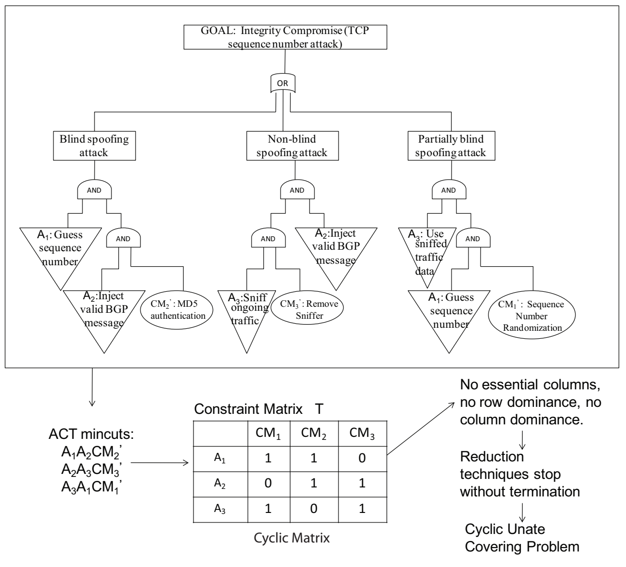
Figure 4. A sample run of the procedure REDUCE (OPT,T) (in Algorithm 1) on the matrix T for the BGP ACT in Figure 2

The partial cover problem can be considered as a special case of the full cover problem. The objective function is the same as in Eq. (1) but the constraint is different from the full cover case:

$$\forall A_i \in \mathcal{CVS}, \sum_{i=1}^{n} t_{ij} \times x_j \geq 1 \qquad (3)$$

Optimal solution of BGP ACT in Figure 4 is an instance of the partial cover with intent case. One optimal solution $\{CM_1$,$CM_2\}$ requires a total security investment cost of 90\$ (based on values in Table A.4). The other possible optimal solution $\{CM_{121}$,$CM_2\}$ requires a total security investment cost of 80\$. If the system administrator has a budget constraint of less than 80\$ (let us say 50\$), only one countermeasure can be implemented in the ACT (CM_1 or CM_{121} or CM_2, each with its own CVS). For example, CVS of CM_1 is $\{A_1\}$ or $\{A_{111}$,A_{1121},A_{1122},A_{1123},$A_{121}\}$ whereas CVS of CM_2 is $\{A_2\}$. The system administrator then decides which countermeasure to be enforced based on its corresponding CVS. The SCADA ACT in Figure 3 is an instance of a partial cover without intent. The system administrator is provided with countermeasures for dealing with corruption in the power load system but none of them can deal with any corrupt estimates provided by the customers. Optimal countermeasure set is computed using Algorithm 2 but with covered set $\mathcal{CVS}$ = {HMI, SCOPF, G_1, G_2, G_3}. The possible optimal countermeasure sets are OPT_1={(switch HMI),(restart G_1)},OPT_2={(switch HMI),(restart G_2)} and OPT_3 = {(switch HMI),(restart G_3)}.

3) Special cases on full/partial cover of attack events: For the BGP ACT in Figure 2, greedy solution works and returns an optimal countermeasure set of $\mathcal{OPT}$={CM_1,CM_2} (see Figure 4). However, only greedy strategy does not work for all ACTs. For the ACT for TCP sequence number attack in Figure 5, we see that reduction techniques do not work on the matrix as the matrix is devoid of essential columns, row or column dominance (this form of a matrix is called a cyclic matrix). This problem is a special case of the 'unate covering problem' called the 'cyclic unate covering problem'. Noel *et. al.* in [18] discussed a similar drawback while analyzing attack graphs with their Topological Vulnerability Analysis (TVA) tool. If an ACT has a cyclic matrix or has a matrix that is reducible to a cyclic matrix, multiple optimal solutions may exist. To reduce computational overhead, we perform an implicit enumeration of the possible solutions. We use a branch and bound algorithm [19]. The optimal solution using objective function F_1 can be found using Algorithm 2 where we separate out all four cases (i) full cover, (ii) partial cover with intent, (iii) partial cover without intent, and (iv) cyclic constraint matrix. Before we present a branch and bound


GOAL: Integrity Compromise (TCP sequence number attack)
OR
Blind spoofing attack
Non-blind spoofing attack
Partially blind spoofing attack
AND
AND
AND
AND
AND
AND
A_1: Guess sequence number
A_2:Inject valid BGP message
CM_2': MD5 authentication
A_3:Sniff ongoing traffic
CM_3': Remove Sniffer
A_2:Inject valid BGP message
A_3: Use sniffed traffic data
A_1: Guess sequence number
CM_1': Sequence Number Randomization
ACT mincuts:
$A_1A_2CM_2$'
$A_2A_3CM_3$'
$A_3A_1CM_1$'
Constraint Matrix T
Cyclic Matrix
No essential columns, no row dominance, no column dominance.
Reduction techniques stop without termination
Cyclic Unate Covering Problem

	CM_1	CM_2	CM_3
A_1	1	1	0
A_2	0	1	1
A_3	1	0	1

Figure 5. Instance of a cyclic matrix

algorithm for the special cases, we introduce cost analysis using ACT.

B. Quantitative Analysis without probability assignment

Cost Computation. In ACT, cost may be of two types: cost of attack and security investment cost. Cost of attack in an ACT ($C_{attacker}$) with no repeated events is computed using the expressions in Table I [20]. For an ACT containing one or more repeated events, attack cost of the mincut with lowest cost is selected to be the cost of attack for the ACT in general. Security investment cost for ACT is computed by summing the security investment cost of countermeasures present in the ACT.

Table I
FORMULAE FOR ATTACK COST AND IMPACT COMPUTATION

Gate type	attack cost	impact
AND gate	$\sum_{i=1}^{n} c_{A_i}$	$\sum_{i=1}^{n} i_{A_i}$
OR gate	$\min_{i=1}^{n} c_{A_i}$	$\max_{i=1}^{n} i_{A_i}$
k-of-n gate	$\sum_{i=1}^{k} c_{A_i}$	$\sum_{i=1}^{k} i_{A_i}$

Impact Computation. Impact computation for different gates in ACT with no repeat events is summarized in Table I. If one or more repeated events are present in the ACT, we follow a procedure similar to that used in cost computation. We first find the mincuts of the ACT. Impact of a mincut is the sum of the impact values of the atomic attack events in the mincut.

C. Minimization of Security Investment Cost

We use objective function F_2 shown in Eq. (4) to find a countermeasure set to minimize the security investment cost and maximize the security level of a system. We can see that F_2 is a general version of the objective function F_1 where $\forall CM_j$, c_{CM_j} =1. Then the objective is to select $x_j \in \{0, 1\} \forall j$ so as to minimize

$$F_2 = \sum_{j=1}^{n} x_j \times c_{CM_j} \tag{4}$$

subject to the same constraint as in Eq. (2) (covered set = $\mathcal{A}$). With the objective function F_2 and the constraint given above, the optimization problem is still a linear zero-one integer programming problem. We build a recursion tree of all possible solutions while computing cover under F_2 in search of the optimal solution. After the current best solution is found, the tree is pruned whenever we come across a solution that is worse than the current best solution. Some terminology relevant to the branch and bound algorithm are summarized as follows:

- *Partial Solution*: A partial solution consists of a subset of the entire solution. Consider the complete solution as a binary n-vector {x_1, x_2,..., x_{m-1}, x_m,... x_{n-1}, x_n}. At any stage, we may have a partial solution of the form P = {x_1, x_2, ..., x_{m-1}, x_m}= {1,0,..., 0,1}, with unassigned values for {x_{m+1},...,x_n}.
- *Upper Bound* (Z_U): This is the value of the objective function for the best feasible solution found so far. If no solution has yet been found, upper bound is assumed to be $\sum_{j=1}^{n} c_{CM_j}$.
- *Lower Bound* (Z_L): For each partial solution, a lower bound for the value of the objective function of the solutions within that subset can be computed.

Algorithm 2 Algorithm for solution with obj. function F_1

```
OPTCMACT (ACT mincuts from SHARPE) {
1. Initialize $\mathcal{OPT}$ (optimal set of countermeasures) = $\phi$,
   m = $|\mathcal{A}|$ and n = $|\mathcal{CM}|$.
   where, $\mathcal{A}$ = set of atomic attack events in ACT
          $\mathcal{CM}$ = set of all countermeasures
2. Initialize the $m \times n$ matrix (T)
   for every $A_i \in \mathcal{A}$
      for every $CM_j \in \mathcal{CM}$
         if (atomic attack event $A_i$ is covered by countermeasure
         $CM_j$)
            set $t_{ij}$= 1
         else set $t_{ij}$= 0
         end if
      end for
   end for
3. If $\mathcal{A} \neq \mathcal{CVS}$, (partial cover with intent)
         (where $\mathcal{CVS}$=set of atomic attack events to be
         covered)
         eliminate all rows in T $\notin \mathcal{CVS}$
   end if
4. Eliminate any row in T with all zeros (partial cover w/o
   intent i.e. this atomic attack event cannot be covered)
5. ($\mathcal{OPT}$,T) = REDUCE ($\mathcal{OPT}$,T) /*matrix reduction*/
6. if (T is a column matrix ($CM_k$))
         $\mathcal{OPT}$=$\mathcal{OPT} \bigcup \{CM_k\}$ /*greedy solution works*/
   else /*cyclic unate covering problem*/
   /*call branch and bound*/
         matrix T is cyclic $^a$
         $\mathcal{OPT}$=$\mathcal{OPT} \bigcup$ B&BACT ($\mathcal{P}$=$\phi$,T,$\mathcal{OPT}_1$=$\phi$,Z$_L$=0,
         Z$_U$=no. of columns in T,$c_{\mathcal{CM}}$={1,1,...,1})$^b$
   end if
7. return $\mathcal{OPT}$
}
```

- *Fathoming Tests*: A partial solution is said to be fathomed if the corresponding subtree in the recursion tree can be excluded from further consideration. A partial solution can be fathomed if:
  (1) Lower bound of the partial solution exceeds the current upper bound of the objective function i.e., $Z_L \geq Z_U$.
  (2) The subset of solutions under this partial solution contains no feasible solutions. In terms of the matrix T and partial solution $\{x_1, x_2, ..., x_{m-1}, x_m\}$, the test is:

$$\exists A_i \in \mathcal{A}, \sum_{j=1}^{m} t_{ij} \times x_j + \sum_{j=m+1}^{n} t_{ij} < 1 \qquad (5)$$

  (3) A feasible solution (with respect to the current upper bound) under that partial solution has already been found. In terms of the matrix T and partial solution $\{ x_1, x_2,..., x_{m-1}, x_m \}$, the test here is:

$$\forall A_i \in \mathcal{A}, \sum_{j=1}^{m} t_{ij} \times x_j > 1 \qquad (6)$$

The recursive branch and bound algorithm [19] for computing the optimal countermeasure set using the objective function F_2 is depicted in Algorithm 3. For objective function F_2, the T matrix for the ACT is generated. The essential columns

Algorithm 3 Recursive Branch and Bound algorithm

```
B&BACT ($\mathcal{P}$, T, $\mathcal{OPT}$, $Z_L$, $Z_U$, c$_{CM}$) {
1. /*compute lower bound*/
   $Z_L$=LOWER BOUND($\mathcal{P}$, $Z_L$, c$_{CM}$)
2. /*is current solution bounded ?*/
   if ($Z_L$ $\geq$Z$_U$) /*start Fathoming Test 1*/
         return ($\mathcal{OPT}$,$Z_U$) /*return current optimal*/
   else /*end Fathoming Test 1*/
3.    /*are there any feasible solutions in this subtree ?*/
      for every $A_i \in \mathcal{A}$ /*start Fathoming Test 2*/
         sum$_i$=0
         for every $x_j \in \mathcal{P}$
            sum$_i$=sum$_i$+$t_{ij} * x_j$
         end for
         for every $x_j \notin \mathcal{P}$ & $CM_j \in \mathcal{CM}$
            sum$_i$=sum$_i$+$t_{ij}$
         end for
         if (sum$_i$ = 0) /*current solution is bounded*/
            return ($\mathcal{OPT}$,$Z_U$) /*return current optimal*/
         end if
      end for /*end Fathoming Test 2*/
4.    /*is current $\mathcal{P}$ a feasible solution for this subtree?*/
      flag=0 /*start Fathoming Test 3*/
      for every $A_i \in \mathcal{A}$ /*for every row*/
         sum$_i$=0
         for every $x_j \in \mathcal{P}$
            sum$_i$=sum$_i$+$t_{ij} * x_j$
         end for
         if (sum$_i$ = 0)
            flag=1
         end if
      end for
      if (flag = 0) /*found new optimal*/
         k=$|\mathcal{P}|$
         set all $x_{k+1}$ to $x_n$ to 0
         $\mathcal{OPT}$=P $\bigcup$ {$x_{k+1}$,$x_{k+2}$,....,$x_n$} /*set new optimal*/
         $Z_U$ = $Z_L$ /*set new upper bound = current $Z_L$*/
         return ($\mathcal{OPT}$,$Z_U$) /*return new optimal*/
      else /*Fathoming Test 3 failed*/
5.       $x_k$=1 /*Branch*/
         ($\mathcal{OPT}$,$Z_U$)= B&BACT($\mathcal{P} \bigcup x_k$,T,OPT,$Z_L$,$Z_U$,$c_{CM}$)
         if ($\mathcal{OPT}$ = P$\bigcup x_k$) /*skip the right subtree*/
            return ($\mathcal{OPT}$,$Z_U$) /*return current optimal*/
         else /*search the right subtree*/
            $x_k$=0
            ($\mathcal{OPT}$,$Z_U$)= B&BACT ($\mathcal{P} \bigcup x_k$, T, $\mathcal{OPT}$,$Z_L$,
            $Z_U$,$c_{CM}$)
            return ($\mathcal{OPT}$,$Z_U$) /*return optimal*/
      end if
   end if
}
```

in the T matrix are removed and placed in the set $\mathcal{ESS}$ and the sum of the costs of the countermeasures in $\mathcal{ESS}$ is stored in $c_{\mathcal{ESS}} = \sum_{\forall CM_j \in \mathcal{ESS}} c_{CM_j}$. The call B&BACT ($\mathcal{P}$=$\phi$, T, $\mathcal{OPT}$=ϕ, Z_L=0, $Z_U = \sum_{j=1}^{n} c_{CM_j}$, c_{CM}) to Algorithm 3 returns an optimal countermeasure set ($\mathcal{OPT}$). Set union of $\mathcal{OPT}$ and $\mathcal{ESS}$ gives us the optimal solution corresponding to the objective function F_2 (where c_{CM}={c_{CM_1},c_{CM_2} ,..., c_{CM_n}}) and the corresponding optimal value of the objective function F_2 is obtained by adding $c_{\mathcal{ESS}}$ to Z_U.

The corresponding procedure for computation of lower bound is shown in Algorithm 4. We obtain the optimal countermeasure set as $\{CM_1$,$CM_2\}$ using Algorithm 2 on the ACT for TCP Sequence Number attack (shown in Figure 5) with the objective function F_1. Here one of the three optimal solutions is selected by the branch and bound algorithm and the other solutions are bounded under fathoming test 1 and hence discarded. Using F_2 on the BGP ACT, an optimal countermeasure set of $\{CM_{121},CM_2\}$ with minimum security investment cost = 80\$ is returned. The necessary input parameter values are in Table A.3 and Table A.4.

Algorithm 4 Computing lower bound for obj. function F_2

```
LOWER BOUND($\mathcal{P}$, $Z_L$, $c_{CM}$) {
1. k=$|\mathcal{P}|$
2. if $x_k$ = 1
        $Z_L$=$Z_L$ + $c_{CM_k}$
        return $Z_L$ /*return new lower bound*/
  else
        return $Z_L$ /*return old lower bound*/
  end if
}
```

IV. Optimal Countermeasure Selection with probability assignments

In this section, we discuss the selection of optimal countermeasure set from ACT with probability assignments. Probability of success of an attack, detection or mitigation event can be measured using simulation, emulation (like, DETER testbed [21]), a testbed (in which real security attack, detection and mitigation events happen) [22], a honeypot [23] or IDS log information as in [8]. Incorporating these probability values into our optimization problem, we can provide more efficient optimal solutions.

A. *Quantitative Analysis using probability values*

Quantitative analysis using ACT also includes metrics such as probability of attack, risk and return on investment.

Probability of Attack. Figure 6(a) shows an ACT with one atomic attack event, one detection event and one mitigation event. Eq. (7) is the corresponding expression for the probability that attack is successful, i.e., either attack is undetected or the attack is detected but unmitigated (D represents a detection event and M represents a mitigation event).

$$\begin{aligned} P_{goal} &= p_A(1 - p_D + p_D(1 - p_M)) \\ &= p_A(1 - p_D \times p_M)) \end{aligned} \tag{7}$$

Figure 6(b) shows an ACT with one atomic attack event, m detection events and n mitigation events. The corresponding probability of successful attack is:

$$P_{goal} = p_A(1 - (1 - \prod_{i=1}^{m}(1 - p_{D_i})) \times (1 - \prod_{i=1}^{n}(1 - p_{M_i}))) \tag{8}$$

Figure 6(c) shows an ACT with one atomic attack event and n pairs of detection and mitigation events. The nature of mitigation triggered depends on the nature of intrusion

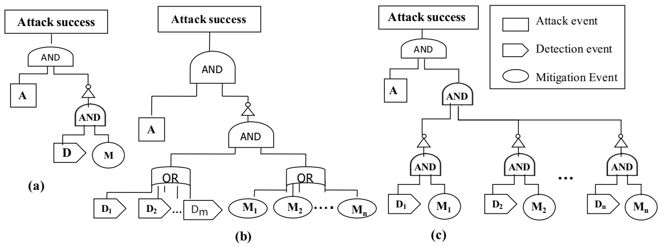


Figure 6. (a) ACT with one attack, one detection and one mitigation event, (b) one attack, m detection and n mitigation events and (c) one attack and multiple pairs of detection and mitigation events

detected. The corresponding expression for P_{goal} is:

$$P_{goal} = p_A \prod_{i=1}^{n}(1 - p_{D_i} \times p_{M_i})) \tag{9}$$

Table II enumerates formulae for output probability for AND, OR gates and k-of-n gates (with identical inputs for k-of-n gates) in an ACT without repeated events.

Risk Computation. Risk to the system is defined as the expected value of the impact. Hence Risk$_{sys}$ is the product of the amount of damage an attack scenario can render to the system (I_{goal}) and probability of attack success (P_{goal}):

$$\text{Risk}_{sys} = P_{goal} \times I_{goal} \tag{10}$$

ROI Computation. The basic definition of ROI_{CM_j} [4] is the profit obtained by the implementation of CM_j (thereby signifying the efficacy of that countermeasure). ROI for countermeasure CM_j is a function of the impact of attack, the decrease in the probability of attack at the ACT goal ($P_{\text{goal without } CM_j} - P_{\text{goal with } CM_j}$) due to CM_j and the security investment cost for CM_j (c_{CM_j}). Adapting Sonnenreich's definition of Return on Security Investment [4], we have:

$$\begin{aligned} \text{ROI}_{CM_j} &= \frac{\text{profit from CM}_j - \text{Cost of implementing CM}_j}{\text{Cost of implementing CM}_j} \\ \text{ROI}_{CM_j} &= \frac{I_{goal} \times (P_{\text{goal without } CM_j} - P_{\text{goal with } CM_j}) - c_{CM_j}}{c_{CM_j}} \end{aligned} \tag{11}$$

Table II
Formulae for probability of attack success

Gate type	Prob. of attack success
AND gate	$\prod_{i=1}^{n} p(i)$
OR gate	$1 - \prod_{i=1}^{n}(1 - p(i))$
k/n gate	$\sum_{j=k}^{n} \binom{n}{j} p^j * (1 - p)^{n-j}$

ROI for a set of countermeasures (S) can be defined similarly. ROI$_S$ is as in Eq. (12) where $|S|$=m.

$$\text{ROI}_S = \frac{I_{goal} \times (P_{\text{goal without S}} - P_{\text{goal with S}}) - \sum_{j=1}^{n} x_j \times c_{CM_j}}{\sum_{j=1}^{n} x_j \times c_{CM_j}} \tag{12}$$

Note that, for any $0 < p_{CM_j} \leq 1$, ROI$_{CM_j}$ $>$ -1.

B. Maximize the profit from implementing a set of countermeasures

Profit from implementing a set of countermeasures (S) is the value of the numerator of ROI$_S$. The objective function in this case is to select $x_j \in \{0,1\} \forall j$ so as to,

$$\begin{aligned} \max_{\forall S \in 2^{\mathcal{CM}}} F_3 &= ROI_S \times \sum_{j=1}^{n} x_j \times c_{CM_j} \\ &= I_{goal} \times (P_{\text{goal without S}} - P_{\text{goal with S}}) \\ &\quad - \sum_{j=1}^{n} x_j \times c_{CM_j} \end{aligned} \tag{13}$$

where S={CM$_1$,CM$_2$,...,CM$_m$} and the constraint (covered set = $\mathcal{A}$) is as in Eq. (2). Since P$_{goal}$ is a non-linear function of $\{x_1, x_2, ..., x_n\}$, the optimization problem is a non-linear binary integer programming problem. We use Watters' [14] transformation to convert this non-linear integer programming problem into a linear one. The non-linearity in P$_{goal}$ is owing to the product terms of the form $x_{i1}x_{i2}x_{i3}...x_{iq}$ (this being the ith product term). It can be easily seen that the value of this product term will be one only when $x_{i1}=x_{i2}=x_{i3}=...=x_{iq}=1$. In all other cases, it is zero. Hence we can replace the product term by a new binary integer variable x_Q (where $x_Q \in \{0,1\}$) such that x_Q is one only when $x_{i1}=x_{i2}=x_{i3}=...=x_{iq}=1$ else zero. This is taken into account by adding two additional constraints.

$$\begin{aligned} \sum_{m=1}^{q} x_{im} - x_Q &\leq q-1 \\ -\sum_{m=1}^{q} x_{im} + q x_Q &\leq 0 \end{aligned} \tag{14}$$

In this way every term in P$_{goal}$ that is a product of binary integer variables is replaced by a new binary integer variable with addition of new constraints (as shown in Eq. (14)). Thus we now have a linear binary integer programming problem which is solvable by the branch and bound Algorithm 3 with the additional constraints being added to the fathoming tests.

C. Multiple Objective Functions

This is an approach most likely to be taken by a security decision maker. Utility theory can be used to incorporate multiple objective functions in a single objective function as in the objective function F_4 (Eq. (15)), where S={CM$_1$,CM$_2$,...,CM$_m$} (constraint being full cover of $\mathcal{A}$). In F$_4$, we simultaneously minimize P$_{goal}$ and the security investment cost with weights attached to each objective function (the sum of the weights being one) and the objective is to select $x_j \in \{0,1\} \forall j$ so as to minimize

$$F_4 = \alpha \times P_{goal} + (1-\alpha) \times \sum_{j=1}^{n} x_j \times c_{CM_j} \tag{15}$$

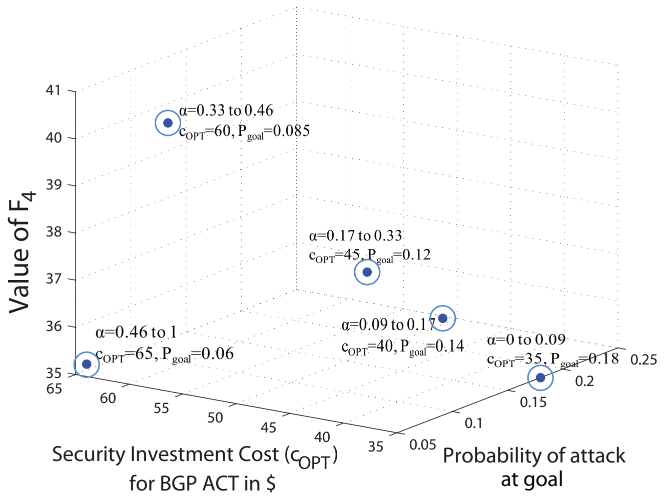


Figure 7. 3D scatter plot of optimal solution points for BGP ACT using objective function F$_4$ for different values of α

The quality of the solution obtained will be quite sensitive to the selected weights. The constraint (covered set = A) is as in Eq. (2). This is a non-linear binary integer programming problem which is solvable in the same way as objective function F$_3$, i.e., the non-linear binary integer programming problem is first reduced to a linear one and then the branch and bound algorithm in Algorithm 3 is used to solve it. A 3D scatter plot of the optimal solution points for BGP ACT using objective function F$_4$ for different values of α plotted against the values of P$_{goal}$, security investment cost and F$_4$ value is shown in Figure 7. Minimum value of F$_4$ (=35) is noted for the range α=(0.46,1).

V. Performance Evaluation

We implemented a module for automatic description and evaluation of ACTs in SHARPE [13]. For the computation of probability of attack and mincuts of ACT we simply call the already existing algorithms for solving fault trees in SHARPE. We have added the relevant algorithms (described in [7]) for computing cost, impact and risk in ACT with and without repeated events in SHARPE. ROI computation is done by defining functions in the SHARPE textual input file. We have implemented all the relevant optimization algorithms in SHARPE.

In the absence of a cyclic T matrix, Algorithm 2 is solvable in polynomial time [15]. Runtime = (total number of atomic attack events in the ACT)*(total number of

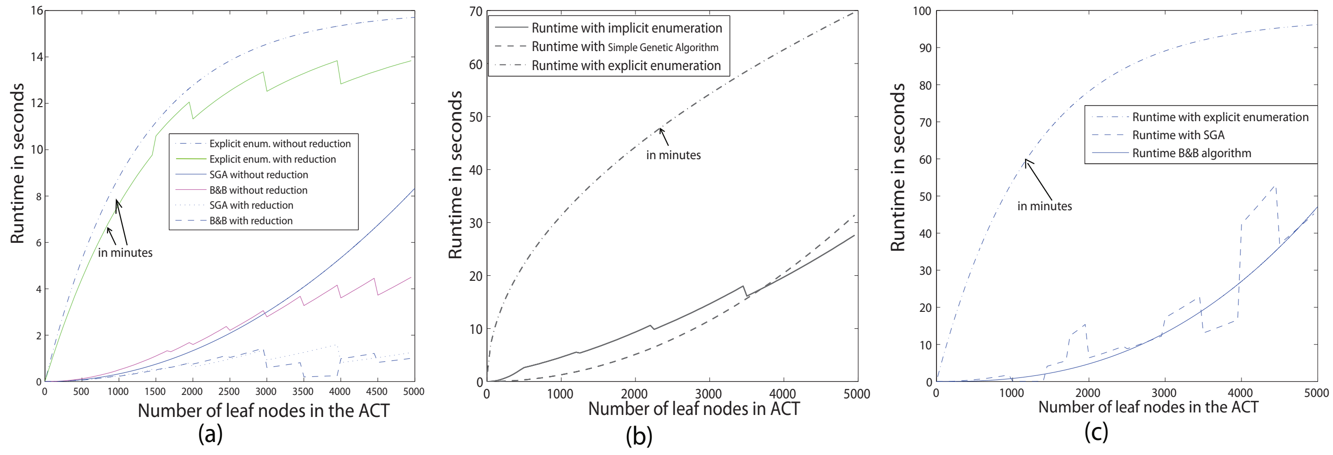


Figure 9. Runtime plot for optimization (a) with obj. function F_1 with increasing tree size using explicit enumeration, branch and bound (Algorithm 3) and an SGA [6] with and without reduction (b) with obj. function F_2 using explicit enumeration, branch and bound (Algorithm 3) and an SGA with increasing tree size (c) with obj. function F_3 using explicit enumeration, branch and bound (Algorithm 3) and an SGA with increasing tree size

defense mechanisms in the ACT)*min(|no of atomic attack events|,|no of defense mechanisms|) = $O(mn * min(m, n))$ which indicates polynomial runtime. Recursion tree for an example cyclic T matrix using Algorithm 3 with objective function F_2 on an ACT that forms a 4×4 cyclic T matrix is shown in Figure 8. A binary string represents the partial solution (P) at any node of the tree. Each bit of the string (represented by indicator function $1_P(\mathrm{CM}_i)$ in Figure 8) represents a certain countermeasure. Bit 0 indicates the corresponding countermeasure is not considered in the partial solution whereas bit 1 indicates it is considered. A string of length 1 indicates a partial solution that takes into account only CM_1 whereas the other countermeasures are redundant. A string of length 2 indicates a partial solution that considers only CM_1 (most significant bit) and CM_2 (least significant bit) whereas the other countermeasures are redundant and so on. Thus, a partial solution of 0101 represents a partial solution where CM_2 and CM_4 are present and CM_1 and CM_3 are absent. The recursion tree in Figure 8 explores every possible solution and shows cases where the subtree is pruned by fathoming rules of branch and bound. The tree shown has 13 nodes as opposed to 31 (2^{4+1}-1) nodes in its explicit enumeration search tree. An analogous pruning technique was proposed in [24] using BDMP (Boolean Logic Driven Markov Processes) where the state-space is trimmed by eliminating attack paths with low success probability to alleviate the state space explosion problem. In the average case, the size of the recursion tree generated by the branch and bound in Algorithm 3 is polynomial in the number of control variables in the optimization (here no. of countermeasures). Since the search space is exponential, the worst case runtime is also exponential. However, this worst case occurs only when the fathoming or bounding rules are not satisfied at any node of the search tree. Such cases are negligible.

In Figure 9, we compare the efficiency of our approach

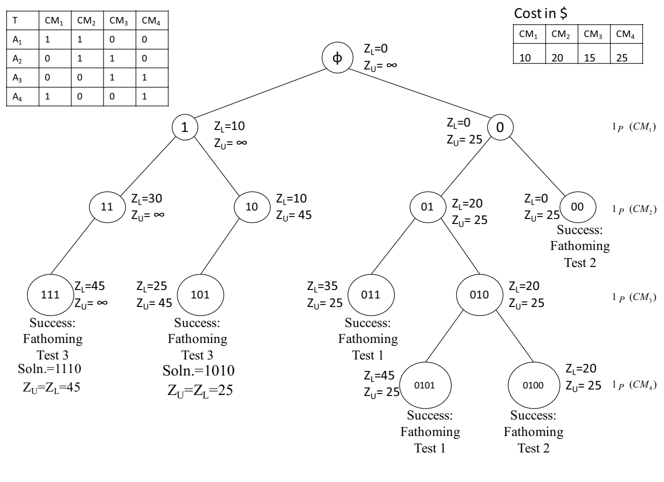

T | CM1 | CM2 | CM3 | CM4
A1 | 1 | 1 | 0 | 0
A2 | 0 | 1 | 1 | 0
A3 | 0 | 0 | 1 | 1
A4 | 1 | 0 | 0 | 1
Cost in $
CM1 | CM2 | CM3 | CM4
10 | 20 | 15 | 25
ϕ Z_L=0 Z_U=∞
1 Z_L=10 Z_U=∞
0 Z_L=0 Z_U=25
11 Z_L=30 Z_U=∞
10 Z_L=10 Z_U=45
01 Z_L=20 Z_U=25
00 Z_L=0 Z_U=25 Success: Fathoming Test 2
111 Z_L=45 Z_U=∞ Success: Fathoming Test 3 Soln.=1110 Z_U=Z_L=45
101 Z_L=25 Z_U=45 Success: Fathoming Test 3 Soln.=1010 Z_U=Z_L=25
011 Z_L=35 Z_U=25 Success: Fathoming Test 1
010 Z_L=20 Z_U=25
0101 Z_L=45 Z_U=25 Success: Fathoming Test 1
0100 Z_L=20 Z_U=25 Success: Fathoming Test 2
1_P (CM1)
1_P (CM2)
1_P (CM3)
1_P (CM4)

Figure 8. Recursion tree for a cyclic T matrix from Algorithm 3

with existing approaches for large ACTs (for instance, ACT for compromising a university LAN). For increasing tree size, runtime (y-axis) for finding optimal countermeasure set using objective function F_1, F_2 and F_3 is plotted against the number of leaf nodes (x-axis) in the ACT in Figure 9(a), Figure 9(b) and Figure 9(c) respectively. For the objective function F_1, we compare the runtimes of six different variants of our methods in Figure 9(a). We compare the runtimes of (i) explicit enumeration without reduction (ii) explicit enumeration with reduction, (iii) branch and bound without reduction (iv) branch and bound with reduction

(v) simple genetic algorithm (SGA) used by Dewri *et.al* without reduction and (vi) a simple genetic algorithm with reduction. Methods (iv) and (vi) are shown to have minimum runtime. For the objective function F_2, our method (branch and bound) returns an optimal solution for the ACT with upto 5000 leaf nodes in almost 25 seconds (as shown in Figure 9(b)) on a system with an Intel(R) Core(TM)2 CPU 1862.103 MHz processor with 2 MB of cache, 4 GB of memory and the CentOS Release 5.4 (Linux 2.6.34) operating system which is much better than the runtime of explicit enumeration technique (almost 70 minutes) and slightly better than the runtime of a simple genetic algorithm (nearly 35 seconds) used for the same objective function. Though the runtime difference between our approach and SGA approach is in seconds for ACT with 5000 leaf nodes, this difference will increase for ACT with higher number of leaf nodes. This demonstrates improved scalability of our approach. For the objective function F_3, our method (branch and bound) returns an optimal solution for the ACT with upto 5000 leaf nodes in 38 seconds on the same system (as shown in Figure 9(c)) which is much better than the runtime of explicit enumeration technique (almost 96 minutes) and slightly better than the runtime of a simple genetic algorithm (40 seconds) used for the same objective function.

VI. Related Work

Schneier developed the basic attack tree (AT) formalism [1]. Dewri *et.al* [6] used genetic algorithms to find optimal security action using single and multi-objective optimization on ATs. However these methods suffered because AT structure did not take countermeasures into account. Bistarelli *et.al* [25] proposed defense trees (DTs) to incorporate defense mechanisms in AT and applied game theory to find the most cost effective set of countermeasures. However in DTs, countermeasures could only be placed at the leaf nodes. Foo *et.al* [26] used intrusion graphs (I-GRAPHs) to model dynamic intrusion response but did not deal with model scalability issues. Zonouz *et.al* [8] proposed attack-response trees (ARTs) that incorporate both attacks and responses but they used a state-space model (partially observable stochastic game model) to find an optimal set of countermeasures thereby leading to state-space explosion problem. Attack countermeasure tree (ACT) [7] provides a simple yet compact approach for security analysis, harnessing the benefits of the aforementioned models and allowing us to perform optimal countermeasure selection for different attack scenarios under given constraints with a non-state-space approach using reduction technique and implicit enumeration (branch and bound). We compare the efficiency of our algorithms with the approach in [6] as well as the ART-based method [8].

VII. Discussion

In this section, we first describe the utility of our proposed approach in large ACTs. Then we discuss a comparative study of efficiency of ACT over other attack tree models via simulation results.

Optimal countermeasure selection in large ACTs. To design large ACTs, we use an approach similar to the one used in [27]. We build the ACT for a simple network with a router and firewall with one or more attacker hosts outside the firewall, two or more target hosts inside the firewall. A certain number of vulnerabilities are associated with each host. To design larger ACTs, we keep increasing the number of hosts and no. of vulnerabilities per host in this benchmark network. The method we use for ACT generation was discussed later. In ACTs with thousands of nodes, partial cover with intent is convenient for protecting the system against only the critical attack scenarios (i.e., the attack events in CVS). Using objective function F_1 in large ACTs, we find that the majority of countermeasures in the corresponding optimal countermeasure set are placed at higher levels of the ACT. This is because a countermeasure at a higher level of the ACT can cover a larger subtree than one at a lower level and F_1 aims to minimize the number of countermeasures used. Security investment cost for a countermeasure increases as we move up the ACT (as countermeasures tend to get more sophisticated at higher levels/advance stages of attack). Hence with objective function F_3, F_4 and in some special cases F_2 in large ACTs, the optimal countermeasure sets are dominated by countermeasures at lower levels of the ACT. The use of branch and bound algorithm is also a good fit for large ACTs. From Figure 8, we see that recursion tree size for explicit enumeration is 2^n where n is the number of countermeasures in ACT. However owing to our bounding strategies, we find the size of our branch and bound recursion tree to be polynomial (n^2) in the number of countermeasures (n) in the average case and linear in the best case. Attack actions that occur sequentially (for e.g., Sybil attack where masquerading attacker sends out fake response to route request messages that results in spreading of incorrect neighborhood information) are best represented by Markov chains. To incorporate such sequential attack actions in large ACTs, an ACT is transformed into a hierarchical model with lower level Markov chains that represent attack progression. These low-level Markov chains are characterized by safe states and unsafe states. Probability of being in one or more unsafe states is passed to the high-level ACT (as probability of attack success for that node). Countermeasure actions can also be incorporated into the state transition model of the Markov chain. Algorithm 3 for objective functions F_3 and F_4 can still be used to solve for optimal countermeasure sets of hierarchical ACT. The presence of the lower-level Markov chain is factored in only during the computation of the objective function value for a certain partial solution.

Comparison with ART-based optimization [8] via trace-driven simulation. We use a network model with 2 attacker hosts, 7 target hosts and a set of 12 vulnerabilities on each host. The resultant ACT has 3078 nodes whereas the resulting ART has 2859 nodes (detection techniques are not included in ART structure and hence ART has lesser no. of nodes). We preprocess a slice of the traffic trace from the MIT Lincoln Lab tcpdump dataset [28] to simulate our network data traffic. A simplified version of the MDP (Markov Decision Process) and ART-based RRE engine (presented in [8]) with objective function F_3 returns an optimal countermeasure set within 3 minutes $\pm$ 7 sec whereas our ACT-based branch-and-bound method with objective function F_3 does the same in 17 seconds $\pm$ 2 sec. The simulation is run on the same system described earlier in Section V. Simulation results return the same optimal solution for our approach and ART-based optimization using objective functions F_3 and F_4. Our comparison with RRE [8] here is limited to the efficiency of the optimal response action selection step. RRE with its local and global decision making engines performs online recovery of a system under attack. In future work, we will compare the efficiency of online recovery with RRE to our approach. Among other methods for selecting optimal security actions using ATs [6], attack response trees [8], intrusion graphs [26] and attack graphs [18], our approach is the first to cast the optimization problem into an integer programming problem (as opposed to linear programming problem in existing strategies). Existing strategies [6] use probability of attack, security cost or impact as control variables for their objective functions. In contrast, the control variables in our objective functions are boolean variables (x_j) that represent the inclusion or exclusion of a countermeasure (CM_j) in the optimal countermeasure set ($\mathcal{OPT}$), hence making the problem simpler to understand and easier to solve.

Automated generation of ACT. Early work on attack graph (AG) generation applied symbolic model checking [29] on the system's finite state machine (FSM) model with extensions and extended the approach for AT generation. However the worst case size of an FSM model being exponential in the number of bits in its state representation leads to state space explosion problem. Hence, we extend Ammann *et.al.*'s approach [27] by using a series-parallel graph model [30] called exploit graph (EG) (which is a non-state-space model) for our automated ACT generation. Any atomic attack is characterized by a set of preconditions and postconditions and the vulnerability exploit used to execute the attack step. Bypassing countermeasures specific to a certain attack also feature among preconditions of that attack. These preconditions and post-conditions can be viewed as nodes in the exploit graph structure. An edge (representing the exploit step) connects each precondition node to its corresponding postcondition node. If a set of preconditions all need to be true for the exploit to be successful, the resulting subgraph is of AND type. If the exploit succeeds when any one of the preconditions is true, then the resulting subgraph is of OR type. In this way, one can build an exploit graph for every system. Any system's exploit graph is a forest of ACTs. Given a set of preconditions and a specific attack goal, we generate the ACT for that attack goal by pruning the system's exploit graph. In future work, we look to improve our ACT generation by extending the attack graph generation method proposed by Ingols *et.al.* [31] that uses enhanced host-to-host reachability computation and modeling of zero-day exploits while taking into account multi-stage, multi-host attacks as in MulVal [32].

VIII. Conclusions

In this paper, we began with brief introduction to attack countermeasure trees (ACT), an analytic model that allows one to comprehensively perform qualitative and probabilistic analysis of the security of a system. ACT takes into account attacks as well as countermeasures (in the form of detection mechanisms and mitigation techniques). Detections and mitigations can be placed not just at leaf nodes but also at any intermediate nodes of ACT. We use ACT to perform fast and efficient computation of optimal set of countermeasures for systems based on a non-state space model thus avoiding the state-space explosion problem [8]. We study several objective functions (some linear, some non-linear) with goals such as minimizing security investment cost in the ACT and maximizing the benefit from implementing a certain countermeasure set in the ACT under different constraints. Our approach is to first cast the optimization problem into an integer programming problem as opposed to linear programming problem in existing strategies. Using a non-state-space model allows us to explore objective functions such as minimizing the number of countermeasures used in ACT that do not need probability assignments. We showed greedy strategies and implicit enumeration techniques (branch and bound) to compute optimal countermeasure sets and we discuss several case studies that illustrate the corresponding algorithms. We also compare the efficiency of our algorithms with explicit enumeration strategy, genetic algorithms [6] and MDP-based approach using attack response trees [8].

Acknowledgment

This research was supported by US National Science Foundation grant NSF-CNS-08-31325. The authors would like to thank Prof. Paulo Maciel and Prof. Dijiang Huang for their invaluable comments during structuring of the paper.

References

[1] B. Schneier, *Secrets and Lies: Digital Security in a Networked World.* John Wiley and Sons Inc., 2000.
[2] X. Ou, S. Govindavajhala, and A. Appel, "MulVAL: A Logic-based Network Security Analyzer," in *Proc. Usenix Security*, 2005.
[3] K. S. Trivedi, D. S. Kim, A. Roy, and D. Medhi, "Dependability and security models," in *Proc. DRCN*. IEEE, 2009, pp. 11–20.

[4] W. Sonnenreich, J. Albanese, and B. Stout, "Return On Security Investment (ROSI): A Practical Quantitative Model," *J. of Research and Practice in Information Technology*, vol. 38, no. 1, pp. 45–56, 2006.

[5] I. Ray and N. Poolsapassit, "Using Attack Trees to Identify Malicious Attacks from Authorized Insiders," in *Proc. ESORICS*, 2005.

[6] R. Dewri, N. Poolsappasit, I. Ray, and D. Whitley, "Optimal security hardening using multi-objective optimization on attack tree models of networks," in *Proc. CCS*. ACM, 2007, pp. 204–213.

[7] A. Roy, D. S. Kim, and K. S. Trivedi, "ACT: Towards unifying the constructs of attack and defense trees," *J. of Security and Communication Networks*, vol. SI: Defending against insider threats and data leakage, 2011.

[8] S. A. Zonouz, H. Khurana, W. H. Sanders, and T. M. Yardley, "RRE: A Game-Theoretic Intrusion Response and Recovery Engine," in *Proc. DSN*. IEEE, 2009, pp. 439–448.

[9] Z. Duan, X. Yuan, and J. Chandrasekhar, "Constructing Inter-Domain Packet Filters to Control IP Spoofing Based on BGP Updates ," in *Proc. INFOCOM*, 2006, pp. 1–12.

[10] A. Mizrak, S. Savage, and K. Marzullo, "Detecting compromised routers via packet forwarding behavior," *IEEE Networks*, vol. 22, no. 2, pp. 34–39, 2008.

[11] S. Convery, D. Cook, and M. Franz, "An Attack Tree for the Border Gateway Protocol," *Cisco Internet draft 2002*.

[12] R. Kuhn, K. Sriram, and D. Montgomery, "Border gateway protocol security: Recommendations of the national institute of standards and technology," *NIST Special Publication 800-54*, 2007.

[13] K. S. Trivedi and R. Sahner, "Sharpe at the age of twenty two," *ACM SIGMETRICS Perf. Eval. Review*, vol. 36, no. 4, pp. 52–57, 2009.

[14] L. Watters, "Reduction of integer polynomial programming problems to zero-one linear programming problems," *Operations Research*, vol. 15, no. 6, pp. 1171–1174, 1967.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT press, 2001.

[16] E. L. McCluskey, "Minimization of Boolean functions," *Bell Systems Technology Journal*, vol. 35, pp. 1417–1444, 1959.

[17] A. Roy, D. Kim, and K. Trivedi, "Cyber Security analysis using Attack Countermeasure Trees," in *Proc. CSIIRW*. ACM, 2010, pp. 28–31.

[18] S. Noel and S. Jajodia, "Optimal ids sensor placement and alert prioritization using attack graphs," *J. of Network and Systems Management*, vol. 16, no. 3, pp. 259–275, 2008.

[19] F. Hillier and G. Lieberman, *Introduction to operations research*. McGraw-Hill New York, 1990.

[20] D. M. Nicol, W. H. Sanders, and K. S. Trivedi, "Model-based evaluation: From dependability to security," *IEEE Trans. on Dependable and Secure Computing*, vol. 1, no. 1, pp. 48–65, 2004.

[21] http://www.isi.deterlab.net/.

[22] A. Sharma, Z. Kalbarczyk, J. Barlow, and R. Iyer, "Analysis of security data from a large computing organization," in *Proc. DSN*. IEEE, 2011, pp. 506–517.

[23] E. Alata, V. Nicomette, M. Kaâniche, M. Dacier, and M. Herrb, "Lessons learned from the deployment of a high-interaction honeypot," in *Proc. EDCC*. IEEE, 2006, pp. 39–46.

[24] L. Piètre-Cambacédès and M. Bouissou, "Attack and defense dynamic modeling with BDMP," in *Proc. MMM-ACNS*, 2010, pp. 86–101.

[25] S. Bistarelli, M. D. Aglio, and P. Peretti, "Strategic Games on Defense Trees," *LNCS*, vol. 4691, pp. 1–15, 2007.

[26] B. Foo, Y. S. Wu, Y. C. Mao, S. Bagchi, and E. Spafford, "ADEPTS: Adaptive Intrusion Response Using Attack Graphs in an E-Commerce Environment," in *Proc. DSN*. IEEE, 2005, pp. 508–517.

[27] P. Ammann, D. Wijesekara, and S. Kaushik, "Scalable Graph based Network Vulnerability Analysis," in *Proc. IEEE S & P*, 2002, pp. 217–224.

[28] http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/1999data.html.

[29] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. Wing, "Automated generation and analysis of attack graphs," in *Proc. IEEE S & P*, 2002, pp. 273–284.

[30] R. Sahner and K. S. Trivedi, "Performance and Reliability Analysis using Directed Acyclic Graphs," *IEEE Transactions on Software Engineering*, vol. 13, no. 10, pp. 1105–1114, 1987.

[31] K. Ingols, M. Chu, R. Lippmann, S. Webster, and S. Boyer, "Modeling modern network attacks and countermeasures using attack graphs," in *Proc. ACSAC*. IEEE, 2009, pp. 117–126.

[32] X. Ou, W. Boyer, and M. McQueen, "A scalable approach to attack graph generation," in *Proc. CCS*. ACM, 2006, pp. 336–345.

[33] G. H. Baker and A. Berg, "Supervisory Control and Data Acquisition (SCADA) Systems," *The Critical Infrastructure Protection Report 1.6*, 2002.

Appendix

Values for the input parameters for atomic attack events of BGP and SCADA ACTs are in Table A.3 and for the input parameters for countermeasure nodes of both ACTs are in Table A.4. Probability values are generated from traffic traces. Attack cost and impact values in Table A.3 and security investment cost values in Table A.4 for BGP and SCADA ACT are obtained from [11] and [33] respectively. However these values were estimated for a certain specific network (BGP) and industrial (SCADA) environment and we only use them to illustrate our examples.

Table A.3
Parameter values for attack nodes in ACT

ACT Node	Probability of attack	attack cost(in \$)	attack impact (in 10^3\$)
$A_{111}(BGP)$	0.08	50	200
$A_{1121}(BGP)$	0.1	60	130
$A_{1122}(BGP)$	0.15	70	100
$A_{1123}(BGP)$	0.2	100	300
$A_{121}(BGP)$	0.1	150	250
$A_2(BGP)$	0.4	190	275
$A_{S_1}(SCADA)$	0.1	100	300
$A_{S_2}(SCADA)$	0.1	110	150
$A_{S_3}(SCADA)$	0.1	90	225
$A_{WSE}(SCADA)$	0.25	250	250
$A_{ULAN}(SCADA)$	0.3	275	275
$A_{HMI}(SCADA)$	0.2	100	100
$A_{SCOPF}(SCADA)$	0.15	120	120
$A_{G_1}(SCADA)$	0.15	100	300
$A_{G_2}(SCADA)$	0.3	30	200
$A_{G_3}(SCADA)$	0.2	40	150
$A_{DB}(SCADA)$	0.5	170	50
$A_{UWAN}(SCADA)$	0.35	160	100
$A_{WS}(SCADA)$	0.4	150	150

Table A.4
Parameter values for countermeasure nodes in ACT

ACT Node	p_{D_k} or p_{M_k}	Security investment cost (in \$)
$D_1(BGP)$	0.5	10
$M_1(BGP)$	0.6	30
$D_{121}(BGP)$	0.8	10
$M_{121}(BGP)$	0.5	20
$D_2(BGP)$	0.7	15
$M_2(BGP)$	0.5	35
$M_{switch}(SCADA)$	0.25	15
$M_{restartG_1}(SCADA)$	0.4	25
$M_{restartG_2}(SCADA)$	0.5	20
$M_{restartG_3}(SCADA)$	0.6	30