Approximation Algorithms for Determining Placement of Intrusion Detectors

Gaspar Modelo-Howard
Purdue University
gmodeloh@purdue.edu

Saurabh Bagchi
Purdue University
sbagchi@purdue.edu

Guy Lebanon
Georgia Institute of Technology
lebanon@cc.gatech.edu

Abstract

To secure today's computer systems, it is critical to have different intrusion detection sensors embedded in them. The complexity of distributed computer systems makes it difficult to determine the appropriate choice and placement of these detectors because there are many possible sensors that can be chosen and each sensor can be placed in several possible places in the distributed system. In this paper, we describe a method to evaluate the effect a detector configuration has on the accuracy and precision of determining the system's security goals. The method is based on a Bayesian network model, obtained from an attack graph representation of multi-stage attacks. We use Bayesian inference to solve the problem of determining the likelihood that an attack goal has been achieved, given a certain set of detector alerts. We quantify the overall performance as a function of the placement, quality, and uncertainty in knowledge of the detectors. Based on these observations, we implement a dynamic programming algorithm for determining the optimal detector settings in a large-scale distributed system and compare it against a previous Greedy algorithm. Finally, we report the results of five experiments, measuring the Bayesian networks behavior in the context of two real-world distributed systems undergoing attacks. Results show the dynamic programming algorithm outperforms the Greedy algorithm in terms of the benefit provided by the set of detectors picked. The dynamic programming solution also has the desirable property that we can trade off the running time with how close the solution is to the optimal.

1 Introduction

It is critical to provide intrusion detection to secure today's distributed computer systems. The overall intrusion detection strategy involves placing multiple detectors at different points of the system. Examples of specific locations are network ingress or combination points, specific hosts executing parts of the distributed system, or embedded in specific applications that form part of the distributed system. At the current time, the placement of the detectors and the choice of the detectors are more an art than a science, relying on expert knowledge of the system administrator.

The choice of the detector configuration has substantial impact on the accuracy and precision of the overall detection process. There are many choices to consider, including the placement of detectors, their false positive (FP) and false negative (FN) rates, and other detector properties. This results in a large exploration space which is currently explored using ad-hoc techniques. Our paper presents an important step in constructing a principled framework to investigate this exploration space.

At first glance it may seem that increasing the number of detectors is always a good strategy. However, this is not always the case and an extreme design choice of a detector at every possible network point, host, and application may not be ideal. First, there is the economic cost of acquiring, configuring, and maintaining the detectors. Detectors need tuning to achieve their best performance and to meet the targeted needs of the application (specifically in terms of the balance between false positive and false negative rates). Second, a large number of imperfect detectors means a large number of alert streams in benign conditions that could overwhelm the manual or automated response process. Third, detectors impose a performance penalty on the distributed system as they typically share bandwidth and computational cycles with the application. Fourth, system owners may have varying security goals such as requiring high sensitivity or ensuring less tolerance for false positive alerts.

In this paper we address the problem of determining where (and how many) to place detectors in a distributed system, based on situation-specific security and performance goals. We also show that this is an intractable problem. The security

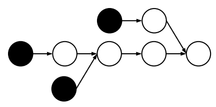


Figure 1. Attack graph model for a sample web server. There are three starting vertices, representing three vulnerabilities found in different services of the server from where the attacker can elevate the privileges in order to reach the final goal of compromising the password file.

goals are determined by requiring a certain trade-off between the true positive (TP) – true negative (TN) detection rates.

Our proposed solution starts with attack graphs, as shown in Figure 1, which are a popular representation for multi-stage attacks [8]. Attack graphs are a graphical representation of the different ways multi-stage attacks can be launched against a specific system. The nodes depict successful intermediate attack goals with the end nodes representing the ultimate attack goal. The edges represent the notion that some attack goals serve as stepping stones to other attack goals and therefore have to be achieved first. The nodes can be represented at different levels of abstraction; thus the attack graph representation can bypass the criticism that detailed attack methods and steps need to be known a priori to be represented (which is almost never the case for reasonably complex systems). Research in the area of attack graphs has included automation techniques to generate these graphs [28], [11], to analyze them [13], [22], and to reason about the completeness of these graphs [13].

We model the probabilistic relations between attack steps and detectors using the statistical formalism of Bayesian networks. Bayesian networks are particularly appealing in this setting since they enable computationally efficient inference for unobserved nodes (such as attack goals) based on observed nodes (detector alerts.) The important question that Bayesian inference can answer for us is, given a set of detector alerts, what is the likelihood or probability that an attack goal has been achieved. A particularly important advantage is that Bayesian network can be relatively easily created from an attack graph structure which is often assumed to be provided.

We formulate two Bayesian inference algorithms, implementing a greedy approach for one and dynamic programming for the other, to systematically determine the accuracy and precision a specific detector configuration has. We then proceed to choose the detector placement that gives the highest value of a situation-specific utility function. From prior work, we show the Greedy algorithm has an approximation ratio of $\frac{1}{2}$. The dynamic programming solution falls in the algorithm category of the *fully polynomial time approximation scheme* (FPTAS) and has the desirable property that we can trade off the running time with how close the solution is to the optimal.

We demonstrate our proposed framework in the context of two specific systems, a distributed E-commerce system and a Voice-over-IP (VoIP) system, and compared both algorithms. We experiment with varying the quality of the detectors, the level of knowledge of attack paths, and different thresholds set by the system administrator for determining whether an attack goal was reached. Our experiments indicate that the value of a detector in terms of determining an attack step degrades exponentially with its distance from the attack site.

The rest of this document is organized as follows. Section 2 introduces the attack graph model and provides a brief presentation of inference in Bayesian networks. Section 3 describes the model and algorithms used to determine an appropriate location for detectors. Section 4 provides a description of the distributed systems used in our experiments. Section 5 presents a complete description of the experiments along with their motivations to help determine the location of the intrusion detectors. Section 6 presents related work and section 7 concludes the paper and discusses future work.

2 Background

2.1 Attack Graph

An attack graph is a representation of the different methods by which a distributed system can be compromised. It represents the intermediate attack goals for a hypothetical adversary leading up to some high-level attack goals such as violating the confidentiality, integrity, or availability of a component in the system. It is particularly suitable for representing multi-stage attacks, in which one or more successful attack steps is used to achieve success in a subsequent attack step. The graph nodes represent attack stages, while edges connect the antecedent (or precondition) stages to the consequent (or post-condition) stages. To be accurate, we consider the exploit-dependency attack graph [11], [28], [13], which is by far the most common attack graph type. Recent research on attack graph generation has resulted in the creation of large graphs for systems scaling up to hundreds and thousands of hosts [11], [28].

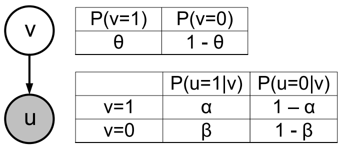


Figure 2. Simple Bayesian network with two types of nodes: an observed node u and unobserved node v. The observed node corresponds to the detector alert in our framework and its conditional probability table includes the true positive (α) and false positive (β) rates.

We construct a Bayesian network that includes the attack graph as a sub-graph. Additional nodes represent detectors, connected by edges to the attack stages they are designed to detect.

2.2 Inference in Bayesian Networks

Bayesian networks [12] provide a convenient framework for modeling the relationship between attack steps and detector alerts. Using Bayesian networks, we can infer which unobserved attack steps have been achieved based on the observed detector alerts.

Formally, a Bayesian network is a joint probabilistic model for n random variables $(x_1, \ldots, x_n)$ based on a directed acyclic graph $G = (V, E)$, where V is a set of nodes corresponding to the variables $V = (x_1, \ldots, x_n)$, and $E \subseteq VV$ contains directed edges connecting some of these nodes in an acyclic manner. Instead of weights, the graph edges are described by conditional probabilities of nodes given their parents that are used to construct a joint distribution via the product $P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i | \{x_j : x_j \text{ is parent of } x_i\})$. Figure 2 shows an example of a Bayesian network with two nodes $v, u \in V$ and the corresponding probability values associated to each node. Node v is the parent of node u.

As previously mentioned, in our context the nodes correspond to either attack stages or detectors. Formally, the nodes are binary random variables with 1 indicating that the attack stage was achieved or the detector issued an alert, and 0 otherwise. Edges correspond to attack graph edges (in the case of edges connecting attack stage nodes) or detector edges connecting detectors to the corresponding attack stage nodes. Consider for example, V_a as the set of attack stages and V_b as the set of detectors. Then, a specific pair of nodes $v \in V_a, u \in V_b$ represents an attack step and a corresponding detector alert. The conditional probability $P(v|u)$ determines the values $P(v = 1|u = 0), P(v = 0|u = 1), P(v = 0|u = 0), P(v = 1|u = 1)$. These probabilities represent FN, FP, and correct behavior (last two), measure the detector effectiveness and can be obtained by evaluating the quality of the detector.

There are three main tasks associated with Bayesian networks. The first is inferring values of variables corresponding to nodes that are unobserved given the values of the observed nodes. In our context this corresponds to predicting whether an attack step has been achieved based on detector alerts $P(x_a = 1|\{x_d : x_d \text{ is a detector connected to attack stage } x_a\})$. The second task is learning the conditional probabilities $P(x_i|\{x_j : x_j \text{ is parent of } x_i\})$ based on available data which in our context corresponds to estimating the reliability of the detectors and the probabilistic relations between different attack steps. The third task is learning the structure of the network based on available data. All three tasks have been extensively studied and despite their difficulty in the general case, may be accomplished relatively easily for Bayesian networks.

We focus in this paper mainly on the first task. The second task of estimating the conditional probabilities can be achieved by characterizing the quality of the detectors [21] and the perceived difficulty of achieving an attack step through risk assessment.

We consider the fact that the estimate is unlikely to be perfectly accurate and provide experiments to characterize the loss in performance due to these imperfections. The third task of obtaining the network structure may be achieved based on the attack graph and detector placement. In our Bayesian network, the network nodes $V = V_a \bigcup V_b$ are partitioned to attack stage nodes V_a, indicating whether an attack stage was achieved, and detector nodes V_b, indicating whether a specific detector issued an alert. The first set of nodes representing attack steps is typically unobserved while the second set of nodes corresponding to alerts are observed and constitute the evidence.

The Bayesian network defines a joint distribution $P(V) = P(V_a, V_b)$ which can be used to compute the marginal probability of the unobserved values $P(V_a)$ and the conditional probability $P(V_a|V_b) = \frac{P(V_a,V_b)}{P(V_b)}$ of the unobserved values given the observed values. The conditional probability $P(V_a|V_b)$ can be used to infer the likely values of the unobserved attack steps given the evidence from the detectors. Comparing the value of the conditional $P(V_a|V_b)$ with the marginal $P(V_a)$

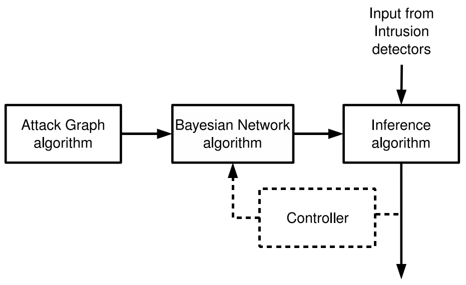


Figure 3. A block diagram of the framework to determine placement of intrusion detectors. The dotted lines indicate a future component, a controller, not included currently in the framework. The controller would provide for a feedback mechanism to adjust location of detectors.

reflects the gain in information for estimating successful attack steps given the current set of detectors. Alternatively, we may estimate the suitability of the detectors by computing the classification error rate, precision, recall, and the Receiver Operating Characteristic (ROC) curve associated with the prediction of V_a based on V_b.

Note that the analysis above is based on emulation done prior to deployment with attacks injected through vulnerability analysis tools, a plethora of which exist in the commercial and research domains, including integrated infrastructures for combining multiple tools.

3 System Design

3.1 Framework Description

Our framework uses a Bayesian network to represent statistical relationships between different attack steps, and between attack steps and detectors. The structure of the Bayesian network[1] is based on the attack graph whose nodes represent attack stages, with additional nodes added to represent the presence of detectors. Each node in the Bayesian network can be in one of two states: attack stage nodes can be 0 or 1, representing whether the stage was achieved or not, and detector nodes can be 0 or 1, representing whether an alarm was generated. The initial nodes correspond to the starting stages of the attack, which do not need any precondition, and the end nodes correspond to the adversary's end goals. Typically, there are multiple leaf nodes and multiple end nodes. Figure 3 shows the complete framework.

The Bayesian network requires that the sets of vertices and directed edges form a directed acyclic graph (DAG). This property is also found in attack graphs. The idea is that the attacker follows a monotonic path, in which an attack step does not have to be revisited after moving to a subsequent attack step. This assumption can be considered reasonable in many scenarios according to experiences from real systems.

A Bayesian network quantifies the statistical relation implied by the edges connecting parent nodes to children nodes. When a node has a parent, a conditional probability table (CPT) is attached to the child node and this is determined by the existence of an incoming edge. As such, the probability values for the state of the child are conditioned on the states of the parents. As an example, the different values of node u in Figure 2 are conditioned on the possible states of its parent, node v.

The values of the conditional probabilities in the CPTs may be obtained from data or by testing specific elements of the system, for example by using IPTables [25] or Snort [32], or by using the subjective judgment of a system administrator. From the perspective of the expert (administrator), the probability values reflect the difficulty of reaching a higher-level attack goal, having achieved some lower-level attack goal.

A potential problem when building the Bayesian network is that the CPT values cannot be reliably determined. In this case, we consider the performance of the Bayesian network under the assumption that the CPT values are inaccurate estimates that may somewhat degrade our ability to diagnose the system. We compute the conditional probabilities of unobserved attack stages given detector alerts using the *junction tree* inference algorithm [19]. The junction tree engine is one of the most

[1]Henceforth, when we refer to a node, we mean a node in the Bayesian network, as opposed to a node in the attack graph. The clarifying phrase is thus implied.

popular inference engines for Bayesian networks and generally provides accurate probability values relatively fast for small or middle-sized networks.

3.2 Cost–Benefit Analysis

In this paper, we address the problem of determining the number and placement of detectors as a cost-benefit exercise. The system benefit is calculated by the BENEFIT function shown below. This specific design considers only the end nodes in the BN, corresponding to the ultimate attack goals. Other nodes that are of value to the system owner may also be considered in alternate designs.

Algorithm 1 BENEFIT (d_i, a_j)

```
1: //This is to calculate the benefit from attaching detector $d_i$ to attack vertex $a_j$
2: //$F$ is the set of end attack vertices $f_k$
3: $F \leftarrow \bigcup_{k=1}^{M} f_k$
4: for all $f_k \in F$ do
5:     perform Bayesian inference with $d_i$ as the only detector in the network and connected to attack vertex $a_j$
6:     calculate $Precision(f_k, d_i, a_j)$
7:     calculate $Recall(f_k, d_i, a_j)$
8:     $systemBenefit \leftarrow \sum_{i=1}^{m} \frac{(1+\beta_{d_i}^2)\Big(Precision(f_k,d_i,a_j)\times(Recall(f_k,d_i,a_j))\Big)}{\Big(\beta_{d_i}^2 \times Precision(f_k,d_i,a_j)+Recall(f_k,d_i,a_j)\Big)}$
9: end for
10: return $systemBenefit$
```

The BENEFIT function is used to calculate the benefit from attaching a detector to an attack vertex in the Bayesian network. To evaluate the performance of a detector, the algorithm uses two popular measures from statistical classification, *precision* and *recall*. Precision is the fraction of true positives (TP) determined among all attacks flagged by the detection system. Recall is the fraction of TP determined among all real positives in the system. Then, the BENEFIT function combines both measures into a single measure, $F_\beta - measure$ [4], which is the weighted harmonic mean of precision and recall and a popular method to evaluate predictors. β is the ratio of recall over precision, defining the relative importance of one to the other. The resulting $F_\beta - measure$ constitutes the output of the BENEFIT function and is called the *systemBenefit*, provided from attaching the detector to the Bayesian network.

The cost model for the system under analysis is defined by the following formula, corresponding to the expectation (in the probabilistic sense) of the cost:

$$COST(d_i, a_j) = \sum_{k=1}^{M} \Big(Prob_{f_k}(TP)\times(cost_{respond})+Prob_{f_k}(FP)\times(cost_{respond})+Prob_{f_k}(FN)\times(cost_{notrespond}\Big) \quad (1)$$

We calculate the cumulative cost associated by selecting a detector, based on its different outcomes with respect to the end nodes: true positive (TP), false positive (FP), and false negative (FN). True negatives (TN) are not considered to compute the detector cost as we believe there should not be any penalty for correct classification of non-malicious traffic. The cost of positive (FP and TP) outcome is related to the response made by the detection system, whereas the FN cost depends on the damage produced by not detecting the attack.

In our design, all probability values (TP, FP, and FN) are first computed by performing sampling on the Bayesian network, since there are no real data (logs) when the system starts and placement of detectors is calculated for the first time. After the initial configuration is done and the system has been monitored for some time, the detection system can be reconfigured by using the log files collected to compute new probability values.

3.3 Greedy Algorithm

We present here an algorithm to determine the choice and placement of detectors in a distributed system, using a greedy approach, and first published in [23]. It takes as input (i) a Bayesian network with all attack vertices, their corresponding CPTs, and the hosts impacted by the attack vertices; (ii) a set of detectors, the possible attack vertices with which each

Algorithm 2 DETECTOR-PLACEMENT (BN, D)

Input: (i) Bayesian network $BN = (V, CPT(V), H(V))$ where V is the set of attack vertices, $CPT(V)$ is the set of conditional probability tables associated with the attack vertices, and $H(V)$ is the set of hosts affected if the attack vertices are achieved; (ii) Set of detectors $D = (d_i, V(d_i), CPT(i, j))$ where d_i is the ith detector, $V(d_i)$ is the set of attack vertices that the detector d_i can be attached to (i.e., the detector can possibly detect those attack goals being achieved), and $CPT(i, j) \forall v_j \in V(d_i)$ is the CPT table associated with detector i and attack vertex j.

Output: Set of tuples $\Theta = (d_i, \pi_i)$ where d_i is the ith detector selected and π_i is the set of attack vertices to which it is attached.

```
1:  systemCost = 0
2:  sort all $(d_i, a_j)$, where $a_j \in V(d_i), \forall i$ by BENEFIT$(d_i, a_j)$. Sorted list = $L$
3:  $n \leftarrow$ length[$L$]
4:  for $i \leftarrow 1$ to $n$ do
5:      systemCost $\leftarrow$ systemCost + COST$(d_i, a_j)$
6:      // COST$(d_i, a_j)$ can be in terms of economic cost, cost due to false alarms and missed alarms, etc.
7:      if (systemCost $> \tau$) then
8:          //$\tau$ is maximum cost system owners can afford (threshold)
9:          break
10:     end if
11:     if $(d_i \in \theta)$ then
12:         add $a_j$ to $\pi_i \in \theta$
13:     else
14:         add $(d_i, \pi_i = a_j)$ to $\theta$
15:     end if
16: end for
17: return $\Theta$
```

detector can be associated, and the CPTs for each detector with respect to all applicable attack vertices. In symbolical terms, the input and output are:

The algorithm DETECTOR-PLACEMENT starts by sorting all combinations of detectors and their associated attack vertices according to their benefit to the overall system (line 2). A greedy decision is made since the detector with highest benefit from the unpicked set is considered singly. From the sorted list, (detector, attack vertex) combinations are added in order, until the overall system cost due to detection is exceeded (line 7).

The worst-case complexity of the DETECTOR-PLACEMENT algorithm is $O(dvB(v, CPT(v)) + dv \log(dv) + dv)$ where d is the number of detectors and v is the number of attack vertices.

$B(v, CPT(v))$ is the cost of Bayesian inference on a BN with v nodes and $CPT(v)$ defining the edges. The first term is due to calling Bayesian inference with up to d times v terms. The second term is the sorting cost and the third term is the cost of going through the for loop (line 4) dv times. In practice, each detector will be applicable to only a small, constant number of attack vertices and therefore the dv terms can be replaced by the constant times d, which will be only d considering order statistics.

A slight but important variation to the Greedy algorithm presented above, found in [15], is also considered in our experiments. We define the modified Greedy algorithm m-DETECTOR-PLACEMENT, which consists of running the original Greedy algorithm first and then comparing the solution with the highest benefit value of any single item. The larger of the two is picked as the output to the algorithm.

The greedy choice of detectors is not guaranteed to provide an optimal solution. For example, consider when three detectors $(d_1, d_2, and d_3)$ are evaluated and a maximum cost of $3M$ is allowed. Detector d_1 has cost $w_1 = 1$ and benefit $b_1 = 2$ and detectors d_2 and d_3 have the same cost and benefit with $w_2 = b_2 = w_3 = b_3 = 1.5M$. The Greedy algorithm would pick d_1 and d_2 for a total benefit of $2 + 1.5M$ while the optimal solution would include d_2 and d_3 and would have a benefit value of $3M$. The non-optimality of DETECTOR-PLACEMENT can be seen theoretically from the observation that the problem of optimal detector choice and placement can be mapped to the 0-1 Knapsack problem which is known to be NP-hard. The mapping is straightforward: consider the threshold cost in our problem to be the weight capacity W for the Knapsack; and the set of possible detectors are the items considered to be included in the Knapsack, with their corresponding benefit and cost as the profit and weight, respectively. An approximation ratio for the modified Greedy algorithm of the

Knapsack problem is found in [34] to be $\frac{1}{2}$. We have included a summary of the calculation in Appendix 9.

3.4 FPTAS Algorithm

The mapping of our DETECTOR-PLACEMENT problem to the 0-1 Knapsack problem allows us to utilize the existing algorithms for the popular NP-hard optimization problem. In particular, the Knapsack problem allows approximation to any required degree of the optimal solution by, as previously mentioned, using an algorithm classified as (FPTAS) since the algorithm is polynomial in the size of the instance n and the reciprocal of the error parameter ϵ. An FPTAS is the best possible solution for an NP-hard optimization problem, assuming of course that $P \neq NP$. The original FPTAS for the 0-1 Knapsack problem was given in [10].

A description of the FPTAS implemented for our experiments follows and is adapted from [15], [34]. The scheme is composed of two steps: first, the scaling of the benefit space to reduce the number of different benefit values to consider and second, running a pseudo polynomial time algorithm based on the dynamic programming technique on the scaled benefit space.

Step 1 - Scaling Step

To obtain the FPTAS, the benefit space is scaled to reduce the number of different profit values and effectively bound the profits in n, the input size. By scaling with respect to the error parameter ϵ, the algorithm produces a solution that is at least $(1-\epsilon)$ times the optimal value, in polynomial time with respect to both n and ϵ. The algorithm is as follows:

Algorithm 3 BENEFIT SPACE SCALING

```
1: Let $B \leftarrow$ benefit of the most profitable object
2: Given $\epsilon > 0$, let $E = \frac{\epsilon B}{n}$
3: $n \leftarrow$ length$[L]$
```

Step 2 - Dynamic Programming Step

Let W be the maximum capacity of the knapsack. All n items under consideration are labeled $i \in 1, \ldots, k, \ldots, n$ and each item has some weight w_i and a scaled benefit value b'_i. Then the Knapsack problem can be divided into sub-problems to find an optimal solution for S_k; that is the solution for when items labeled from 1 to k have been considered, but not necessarily included, in the solution. Then, let $B[k,w]$ be the maximum profit of S_k that has total weight $w \leq W$. Then, the following recurrence allows to calculate all values for $B[k,w]$:

$$B[k,w] = \begin{cases} B[k-1,w] & if w_k > w \\ max B[k-1,w], B[k-1,w-w_k] + b'_k & else \end{cases}$$

The first case of the recurrence is when an item k is excluded from the solution since if it were, the total weight would be greater than w, which is unacceptable. In the second case, item k can be in the solution since its weight (w_k) is less than the maximum allowable weight(w). We choose to include item k if it gives a higher benefit than if we exclude it. In the formula, if the the second term is the maximum value, then we include item k, and we exclude it if the first term is the maximum. The final solution $B[n,W]$ then corresponds to the set $S_{n,W}$ for which the benefit is maximized and the total cost is less or equal to W.

The running time of FPTAS is given by $O(\frac{n^2 B}{\epsilon})$, and its design is based on the idea of trading accuracy for running time. The original benefit space of the 0-1 Knapsack problem is mapped to a coarser one, by ignoring a certain number of least-significant bits of benefit values, which depend on the error parameter ϵ. The mapped coarser instance is solved optimally through an exhaustive search by using a dynamic programming-based algorithm. The intuition, then, is to allow the algorithm to run in polynomial time by properly scaling down the benefit space. This thus provides a trade-off between the accuracy and the running time.

4 Experimental Systems

We created three Bayesian networks for our experiments modeling two real systems and one synthetic network. These are a distributed electronic commerce (e-commerce) system, a Voice-over-IP (VoIP) network, and a synthetic generic Bayesian network that is larger than the other two. The Bayesian networks were manually created from attack graphs that include several multi-step attacks for the vulnerabilities found in the software used for each system. These vulnerabilities are associated

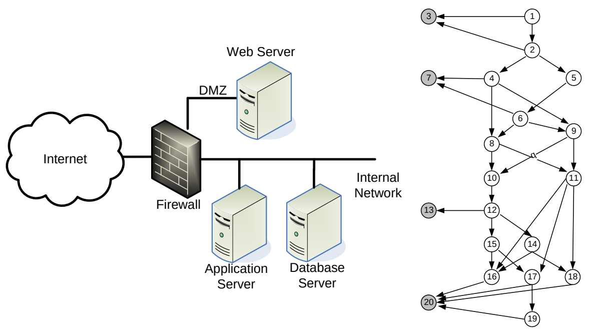


Figure 4. Network diagram for the e-commerce system and its corresponding Bayesian network. The white nodes are the attack steps and the blue nodes are the detectors.

with specific versions of the particular software, and are taken from popular databases [31], [27]. An explanation for each Bayesian network follows.

4.1 E-Commerce System

The distributed e-commerce system used to build the first Bayesian network is a three tier architecture connected to the Internet and composed of an Apache web server, the Tomcat application server, and the MySQL database backend. All servers are running a Unix-based operating system. The web server sits in a de-militarized zone (DMZ) separated by a firewall from the other two servers, which are connected to a network not accessible from the Internet. All connections from the Internet and through servers are controlled by the firewall. Rules state that the web and application servers can communicate, and that the web server can be reached from the Internet. The attack scenarios are designed with the assumption that the attacker is an external one and thus its starting point is the Internet. The goal for the attacker is to have access to the MySQL database (specifically, access to customer confidential data such as credit card information – node 19 in the Bayesian network of Figure 4).

As an example, an attack step would be a portscan on the application server (node 10). This node has a child node, which represents a buffer overflow vulnerability present in the rpc.statd service running on the application server (node 12). The other attack steps in the network follow a similar logic and represent other phases of an attack to the distributed system. The system includes four detectors: IPtables, Snort, Libsafe, and a database IDS. As shown in Figure 4, each detector has a causal relationship to at least one attack step.

Further details on the e-commerce testbed used for the experiments, including all the probability values used for the corresponding Bayesian network, is available in Appendix 8.

4.2 Voice-over-IP (VoIP) System

The VoIP system used to build the second network has a few more components, making the resulting Bayesian network more complex. The system is divided into three zones: a DMZ for the servers accessible from the Internet, an internal network for local resources such as desktop computers, mail server and DNS server, and an internal network only for VoIP components. This separation of the internal network into two units follows the security guidelines for deploying a secure VoIP system [18].

The VoIP network includes a PBX/Proxy, voicemail server and software-based and hardware-based phones. A firewall provides all the rules to control the traffic between zones. The DNS and mail servers in the DMZ are the only accessible hosts from the Internet. The PBX server can route calls to the Internet or to a public-switched telephone network (PSTN). The ultimate goal of this multi-stage attack is to eavesdrop on VoIP communication. There are four detectors – IPtables, and three network IDSs on the different subnets.

A third synthetic Bayesian network was built to test our framework for experiments where a larger network, than the other two, was required. This network is shown in Figure 7(a).

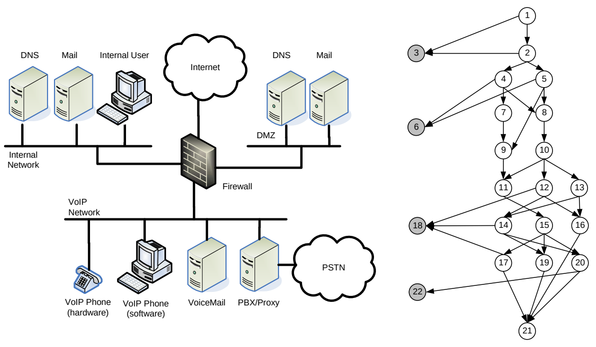


Figure 5. VoIP system and its corresponding Bayesian network.

5 Experiments

The correct number, accuracy, and location of the detectors can provide an advantage to the system's owner when deploying an intrusion detection system. Several metrics have been developed for evaluation of intrusion detection systems. In our work, as first presented in section 3.3, we concentrate on precision and recall. The notions of TP, FP, etc., are shown in Figure 6. We also plot the ROC curve which is a traditional method for characterizing detector performance – it is a plot of the true positive against the false positive.

For the experiments, we create a dataset of 50,000 samples or attacks, based on the respective Bayesian network. We use the Bayesian network toolbox for Matlab [24] for our Bayesian inference and sample generation. Each sample consists of a set of binary values, for each attack vertex and each detector vertex. A one (zero) value for an attack vertex indicates that attack step was achieved (not achieved), and a one (zero) value for a detector vertex indicates the detector generated (did not generate) an alert. Separately, we perform inference on the Bayesian network to determine the conditional probability of different attack vertices. The probability is then converted to a binary determination – whether or not the detection system flagged that particular attack step, using a threshold. This determination is then compared with reality, as given by the attack samples which leads to a determination of the system's accuracy. There are several experimental parameters – which specific attack vertex is to be considered, the threshold, CPT values, etc. – and their values (or variations) are mentioned in the appropriate experiment. The CPTs of each node in the network are manually configured according to the authors' experience administering security for distributed systems and frequency of occurrences of attacks from references such as vulnerability databases, as mentioned earlier.

5.1 Experiment 1: Distance from Detectors

The objective of experiment 1 was to quantify for a system designer the gain in placing a detector close to a service where a security event may occur. Here we used the synthetic network since it provided a larger range of distances between attack steps and detector alerts.

The CPTs were fixed to manually determined values on each attack step. Detectors were used as evidence, one at a time, on the Bayesian network and the respective conditional probability for each attack node was determined. The effect of the single detector on different attack vertices was studied, thereby varying the distance between the node and the detector. The output metric is the difference of two terms. The first term is the conditional probability that the attack step is achieved, conditioned on a specific detector firing. The second term is the probability that the attack step is achieved, without use of any detector evidence. The larger the difference is, the greater is the value of the information provided by the detector. In Figure 7(b), we show the effect due to detector corresponding to node 24, and in Figure 7(c), we consider all the detectors (again, one at a time). The effect of all the detectors shows that the conclusions from node 24 are general.

The results show that a detector can affect nodes inside a radius of up to three edges from the detector. The change in probability for a node within this radius, compared to one outside the radius, can be two times greater when the detector is used as evidence. For all Bayesian networks tested, the results were consistent with the three edges radius observation.

	Attack = True	*Attack = False*
Detection = True	*TP*	*FP*
Detection = False	*FN*	*TN*

$$Recall = \frac{TP}{TP + FN} \qquad Precision = \frac{TP}{TP + FP}$$

Figure 6. Parameters used for our experiments: True Positive (TP), False Positive (FP), True Negative (TN), False Negative (FN), precision, and recall.

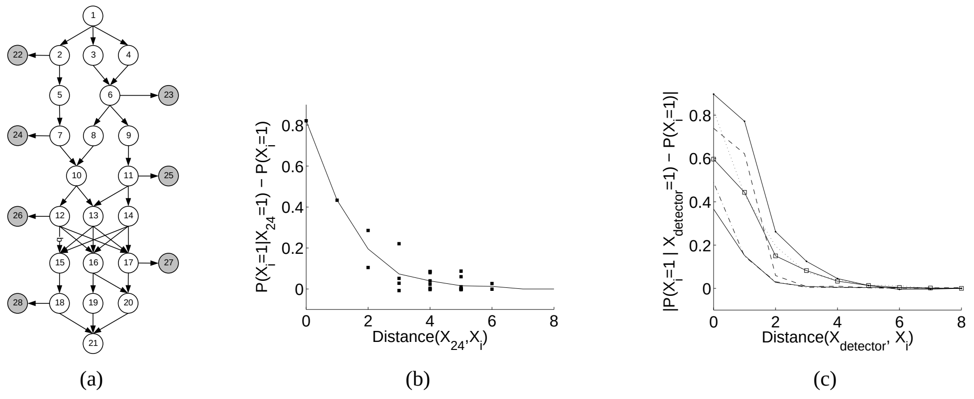


Figure 7. Results of experiment 1: Impact of distance on a set of attack steps. (a) Generic Bayesian network used. (b) Using node 24 as the detector (evidence), the line shows mean values for rate of change. (c) Comparison between different detectors as evidence, showing the mean rate of change for case.

5.2 Experiment 2: Impact of Imperfect Knowledge

The objective of experiment 2 was to determine the performance of the detection system in the face of attacks. In the first part of the experiment (experiment 2a), the effect of the threshold that is used in converting the conditional probability of an attack step into a binary determination is studied. This corresponds to the practical situation in which a system administrator has to make a binary decision based on the result of a probabilistic framework, and there is no oracle at hand to help. For the second part of the experiment (experiment 2b), the CPT values in the Bayesian network are perturbed by introducing variances of different magnitudes. This corresponds to the practical situation that the system administrator cannot accurately gauge the level of difficulty for the adversary to achieve attack goals. The impact of the imperfect knowledge is studied through a ROC curve.

For experiment 2a, precision and recall were plotted as a function of the threshold value. This was done for all the attack nodes in the Bayesian network, and the results for a representative sample of six nodes are shown in Figure 8. We used threshold values from 0.5 to 0.95, since anything below 0.5 would imply the Bayesian network is useless in its predictive ability.

As expected, when the threshold is increased, there are fewer FP and the precision of the detection system improves. The opposite is true for the recall of the system since there are more false negatives. However, an illuminating observation is that the precision is relatively insensitive to the threshold variation while the recall has a sharp cut-off. Clearly, the desired threshold is to the left of the cut-off point. Therefore, this provides a scientific basis for an administrator to set the threshold for drawing conclusions from a Bayesian network representing the system.

In experiment 2b we introduced variance to the CPT values of all the attack nodes, mimicking different levels of imperfect knowledge an administrator may have about the adversary's attack strategies. When generating the samples corresponding to the attacks, we used three variance values: 0.05, 0.15, and 0.25. Each value could be associated with a different level of knowledge from an administrator: expert, intermediate, and nave, respectively. For each variance value, ten batches of 1,000 samples were generated and the detection results were averaged over all batches.

In Figure 9, we show the ROC curves for nodes 1 and 6 of the e-commerce system, with all four detectors in place. As the

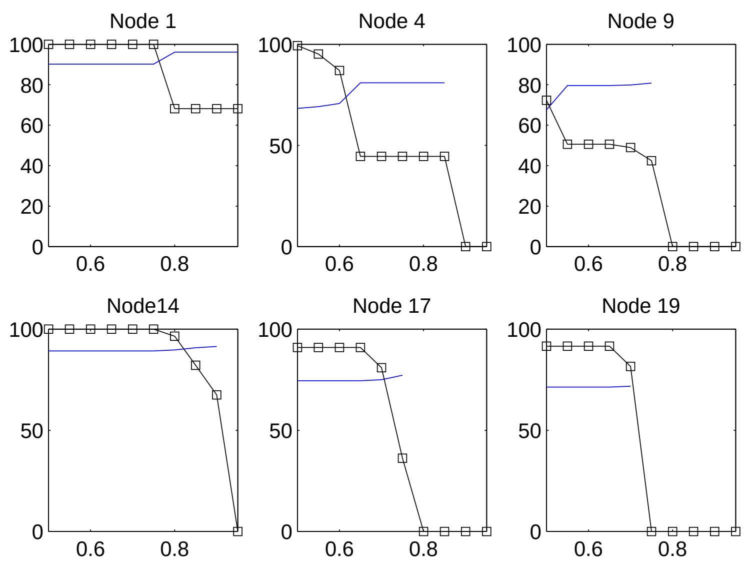


Figure 8. Precision and recall as a function of detection threshold, for the e-commerce Bayesian network. The line with square markers is recall and other line is for precision.

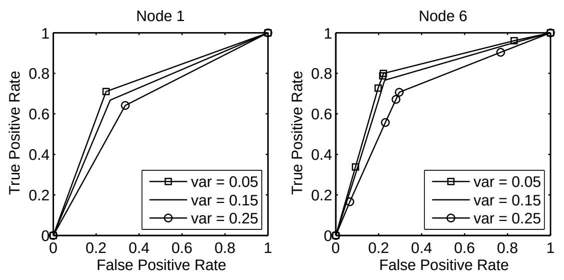


Figure 9. ROC curves for two attack steps in e-commerce Bayesian network. Each curve corresponds to a different variance added to the CTP values.

variance increases, the performance suffers. However, the process of Bayesian inference shows an inherent resilience since the performance does not degrade significantly with the increase in variance. For node 1, several points are placed so close together that only one marker shows up. On the contrary, for node 6, multiple well spread out TP-FP value pairs are observed. We hypothesize that since node 1 is directly connected to the detector node 3, its influence over node 1 dominates over all other detectors. Hence fewer numbers of sharp transitions are seen compared to node 6, which is more centrally placed with respect to multiple detectors.

Experiment 2c also looked at the impact of imperfect knowledge when defining the CPT values in the Bayesian network. Here we progressively changed the CPT values for several attack steps in order to determine how much we would deviate from the correct value. We used two values, 0.6 and 0.8, for each CPT cell (only two are independent), giving rise to four possible CPT tables for each node. We plot the minimum and maximum conditional probabilities for a representative attack node for a given detector flagging. We change the number of CPTs that we perturb from the ideal values. As assumed, when the number of CPTs changed increases, the difference between the minimum and the maximum increases, but the range is within 0.03. Note that the point at the left end of the curve for zero CPTs changed gives the correct value.

Both experiments indicate that the BN formalism is relatively robust to imperfect assumptions concerning the CPT values. This is an important fact since it is likely that the values determined by an experienced system administrator would still be somewhat imperfect. Overall, as long as the deviation of the assumed CPTs from the truth is not overwhelming, the network

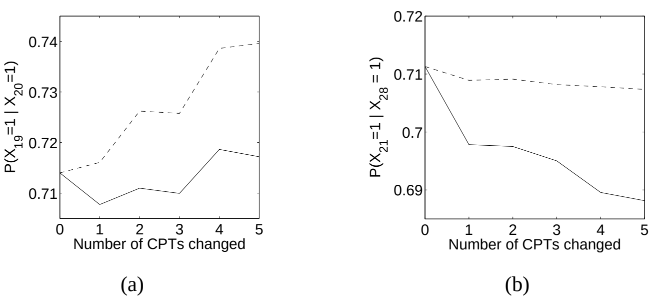


Figure 10. Impact of deviation from correct CPT values, for the (a) e-commerce and (b) generic Bayesian networks.

	Capacity (W=0.51)	Capacity (W=0.60)	Capacity (W=1.20)	Capacity (W=1.50)
Selections made by greedy algorithm	(d_3,a_1)	(d_3,a_1) (d_3,a_2)	(d_3,a_1) (d_3,a_2) (d_{20},a_{19})	(d_3,a_1) (d_3,a_2) (d_{20},a_{19}) (d_7,a_4)
Benefit	0.64	0.61	1.46	2.05
Cost	0.12	0.42	0.88	1.27
Selections made by FPTAS algorithm (ε=0.01)	(d_{20},a_{19})	(d_{20},a_{19}) (d_3,a_1)	(d_{20},a_{19}) (d_{20}, a_{17}) (d_3,a_2)	(d_{20},a_{19}) (d_{20}, a_{17}) (d_3,a_1) (d_7,a_4)
Benefit	0.91	1.57	1.49	2.21
Cost	0.46	0.58	1.21	1.41

Figure 11. Comparison between Greedy algorithm and FPTAS for different cost values.

performance degrades gracefully.

5.3 Experiment 3: Comparison between Greedy algorithm and FPTAS

The objective of experiment 3 was to determine the performance of the FPTAS and compare it to the Greedy algorithm, using the Bayesian network for the e-commerce distributed system. The experiment was repeated for different capacity thresholds, representing cases for different numbers of detectors $(1, 2, 3, \text{or } 4)$. For FPTAS, the algorithm used $\epsilon = 0.01$ since varying the parameter for different values, from 0.01 to 0.30, produced no relevant change on the resulting set of detectors and the running time was similar to the one from Greedy. More information on the running times is provided at the end of this section.

In all cases of the experiment, FPTAS performed better than Greedy algorithm in terms of achieving a higher benefit. FPTAS always first picked the (detector, location) pair $(d_{20,}a_{19})$ closer to the attack goal of interest and with the highest benefit, given the capacity constraint. Nevertheless, the selection of (detector, location) pairs was not accumulative as the capacity threshold was increased. As an example, when the threshold was set to 0.60 (representing the case for two detectors picked), FPTAS selected pairs $(d_{20,}a_{19})$ and $(d_{3,}a_1)$ but when threshold was increased to 120 (three detectors), FPTAS selected $(d_{20,}a_{19})$, $(d_{20,}a_{17})$, and $(d_{3,}a_2)$, removing $(d_{3,}a_1)$ from the solution set. The reason for this situation is that as the capacity threshold is increased, it might include a detector with higher benefit and cost than one selected under the previous threshold considered.

The performance of the Greedy algorithm was interesting as it always started selecting the $(d_{3,}a_1)$ pair, regardless of the capacity threshold. This actually shows the drawback of the Greedy algorithm. It picks detectors that are accurate but are far from the ultimate attack goals that we are interested in. The case when the threshold was set to $W = 0.51$ (one detector) represented an example of a worst-case scenario since the ratio between the optimal selection and the greedy choice was $\frac{1}{2}$. Still, as the threshold was increased, the Greedy algorithm seemed to correct itself and provide a solution closer to the optimal set. For cases of $W = 1.20$ and $W = 1.50$, the Greedy algorithm had all but one of the (detector, location) pairs that are part of the solution set chosen by FPTAS.

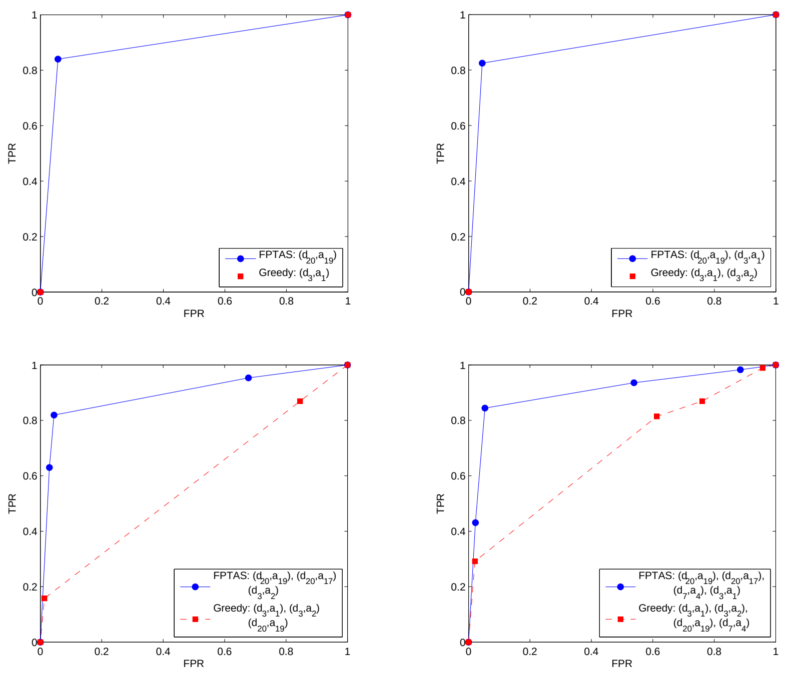


Figure 12. ROC curves for detectors picked by Greedy (dashed line) and FPTAS (solid line) for different capacity values: (a) $W = 0.51$, (b) $W = 0.60$, (c) $W = 1.20$ and (d) $W = 1.50$.

An interesting result is the cost associated to the detectors picked by the Greedy algorithm when compared to the choices made by FPTAS. In all our experiments, the selections made by Greedy had an overall lower cost and benefit than FPTAS. Although the cost value achieved by Greedy might be considered positive result, it is important to remember that our DETECTOR-LOCATION problem is an optimization problem where we try to maximize the benefit.

The ROC curves shown in Figure 12 also represent the results from the experiment. In (a) and (b), the Greedy algorithm starts picking detectors too far away from the attack goal such that it doesn't have any (TPR,FPR) points, except for (0,0) and (1,1). We decided not to plot such lines because the performance of the detector(s) selected is no better than flipping a (fair) coin to determine if the attack goal has been achieved.

FPTAS performs better than the Greedy algorithm by immediately picking the detector closest to the attack goal, in the case of cost = 0.51 (one detector). In this case, FPTAS picks detector d_{20}, which is directly connected to the attack goal. In comparison, the Greedy algorithm starts by picking the detector farthest away from the attack goal. The reason for this is that such a detector has the highest benefit-to-cost ratio among all detectors. The problem is that this ratio does not reflect the actual performance of the detector for the attack goal. Such performance is shown in the corresponding ROC curve (one detector).

In the case of cost = 0.60 (two detectors), The Greedy algorithm follows a similar pattern as the previous case, picking the remaining detector with highest benefit-to-cost ratio. This detector is also far from the attack goal. In contrast, FPTAS picks a detector connected to an attack step, which is connected to attack goal a_{19}, and increasing the overall True Positive Rate of the detection system. Nevertheless, the same addition also increases the false positive rate.

For cases of cost = 1.20 (three detectors) and 1.50 (four detectors), the Greedy algorithm starts picking detectors closer to the attack goal that, as is shown in the corresponding ROC curves, perform relatively similar to the set of detectors selected by FPTAS.

In conclusion, FPTAS starts by selecting the closest (best) detector for the attack goal, and as it adds more detectors improves (marginally) the TPR of the detection system but with a price (also increasing the FPR). The Greedy algorithm selects from a decreasingly sorted list of detectors, according to their benefit-to-cost ratio. These are two examples of the first experiment we performed with Bayesian networks, where we demonstrated that as distance increases between detector and attack goal, the detection capability decreases.

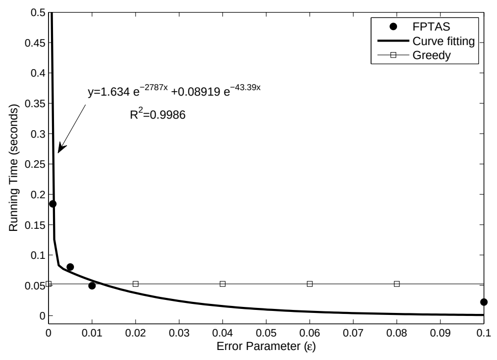
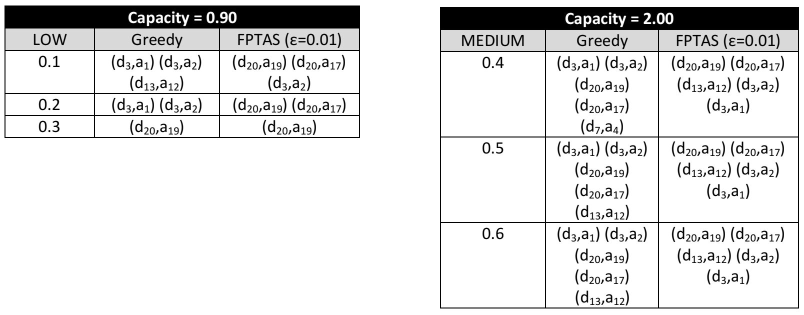


Figure 13. Execution time comparison between Greedy algorithm and FPTAS, for different values of the error parameter (ϵ). In our experiments, values of ϵ equal or larger than 0.01 allow FPTAS to run faster than Greedy.



Capacity = 0.90		
LOW	Greedy	FPTAS (ε=0.01)
0.1	(d_3,a_1) (d_3,a_2) (d_{13},a_{12})	(d_{20},a_{19}) (d_{20},a_{17}) (d_3,a_2)
0.2	(d_3,a_1) (d_3,a_2)	(d_{20},a_{19}) (d_{20},a_{17})
0.3	(d_{20},a_{19})	(d_{20},a_{19})

Capacity = 2.00		
MEDIUM	Greedy	FPTAS (ε=0.01)
0.4	(d_3,a_1) (d_3,a_2) (d_{20},a_{19}) (d_{20},a_{17}) (d_7,a_4)	(d_{20},a_{19}) (d_{20},a_{17}) (d_{13},a_{12}) (d_3,a_2) (d_3,a_1)
0.5	(d_3,a_1) (d_3,a_2) (d_{20},a_{19}) (d_{20},a_{17}) (d_{13},a_{12})	(d_{20},a_{19}) (d_{20},a_{17}) (d_{13},a_{12}) (d_3,a_2) (d_3,a_1)
0.6	(d_3,a_1) (d_3,a_2) (d_{20},a_{19}) (d_{20},a_{17}) (d_{13},a_{12})	(d_{20},a_{19}) (d_{20},a_{17}) (d_{13},a_{12}) (d_3,a_2) (d_3,a_1)

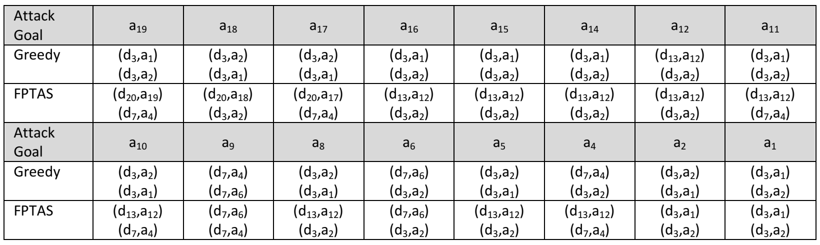
Figure 14. Sensitivity analysis to different low (0.1, 0.2, 0.3) and medium (0.4, 0.5, 0.6) cost values. The table on the left is for capacity of 0.90, which corresponds to two detectors selected by both algorithms. The table on the right is for capacity of 2.0, which corresponds to five detectors selected.

We evaluated the running time for both algorithms by performing 100 execution runs for the Greedy algorithm and for each FPTAS with an error parameter (ϵ) from 0.0001 to 1. Figure 13 summarizes the findings on the running time for both algorithms. It shows the results for FPTAS, for ϵ from 0.0001 to 0.1 along with an exponential regression curve to fit the series of data points collected. In the case of the Greedy algorithm, we report a single value (represented by a straight line at 0.0523 seconds) since it is unaffected by ϵ. From the results, the Greedy algorithm ran faster than FPTAS, when the error parameter was less than 0.01. In our experiments, around that error value (0.01) they both showed similar running times. For the Greedy algorithm it took on average 0.0523 seconds, while for FPTAS the average running time was 0.0499 seconds. We excluded from both algorithms the time taken to create the Bayesian network, the samples and to compute the probability values. All are necessary inputs for both algorithms and took 32.75 seconds on average to create and compute.

5.4 Experiment 4: Sensitivity to Cost Value

The objective of experiment 4 was to evaluate the impact of varying the quantitative value assigned to each cost category (low, medium, high). Three values were used for each category: low (0.10, 0.20, 0.30), medium (0.40, 0.50, 0.60) and high (0.70, 0.80, 0.90). The experiment was repeated on both algorithms, FPTAS and Greedy algorithm, using the Bayesian network for the e-commerce distributed system and for different knapsack capacities (0.51, 0.90, 1.20, 1.50, 2.00, 2.50, and 3.50). Such capacities correspond to the total resources available to the administrator to deploy and administer the detection system. For FPTAS, the algorithm used $\epsilon = 0.01$.

Varying the quantitative value of a cost level seems to only slightly affect the outcome from the algorithms. In all cases,



Attack Goal	a_{19}	a_{18}	a_{17}	a_{16}	a_{15}	a_{14}	a_{12}	a_{11}
Greedy	(d_3,a_1) (d_3,a_2)	(d_3,a_2) (d_3,a_1)	(d_3,a_2) (d_3,a_1)	(d_3,a_1) (d_3,a_2)	(d_3,a_1) (d_3,a_2)	(d_3,a_1) (d_3,a_2)	(d_{13},a_{12}) (d_3,a_2)	(d_3,a_1) (d_3,a_2)
FPTAS	(d_{20},a_{19}) (d_7,a_4)	(d_{20},a_{18}) (d_3,a_2)	(d_{20},a_{17}) (d_7,a_4)	(d_{13},a_{12}) (d_3,a_2)	(d_{13},a_{12}) (d_3,a_2)	(d_{13},a_{12}) (d_3,a_2)	(d_{13},a_{12}) (d_3,a_2)	(d_{13},a_{12}) (d_7,a_4)
Attack Goal	a_{10}	a_9	a_8	a_6	a_5	a_4	a_2	a_1
Greedy	(d_3,a_2) (d_3,a_1)	(d_7,a_4) (d_7,a_6)	(d_3,a_2) (d_3,a_1)	(d_7,a_6) (d_3,a_2)	(d_3,a_2) (d_3,a_1)	(d_7,a_4) (d_3,a_2)	(d_3,a_2) (d_3,a_1)	(d_3,a_1) (d_3,a_2)
FPTAS	(d_{13},a_{12}) (d_7,a_4)	(d_7,a_6) (d_7,a_4)	(d_{13},a_{12}) (d_3,a_2)	(d_7,a_6) (d_3,a_2)	(d_{13},a_{12}) (d_3,a_2)	(d_{13},a_{12}) (d_7,a_4)	(d_3,a_1) (d_3,a_2)	(d_3,a_1) (d_3,a_2)

Figure 15. Detectors selected by Greedy algorithm and FPTAS for different attack goals (a_i).

both algorithms are somehow consistent picking at least the first two or three (detector, attack node) pairs, while varying the quantitative value of the cost level. This is a positive result since the quantitative values are arbitrarily determined by the system administrator or person responsible for assessing the detection systems.

Comparing these results to the previous experiment, both the Greedy algorithm and FPTAS performed as they did in the previous experiment. In the case of Greedy algorithm, it keeps picking (d_3, a_1) before any other pair as this has the highest benefit-to-cost ratio. The FPTAS algorithm starts by picking (d_{20}, a_{19}) as it shows the highest benefit for the knapsack capacity constraint, regardless of the different values used for each cost level. In the case when a *(detector, attack node)* pair is changed because a level value has been changed, this can be explained from the impact the pair has on the individual cost assigned to each pair.

5.5 Experiment 5: ROC curves across Different Attack Graphs

The goal of this experiment is to show the performance of each algorithm, Greedy and FPTAS, by picking a pair of detectors for different attack goals. All attack goals in the e-commerce Bayesian network were used to evaluate the performance of the algorithms.

We decided to limit the size of the set of detectors picked to two, for each case and algorithm, since in our experience it is a reasonable number of detectors for a system administrator to use to defend a particular attack goal. Although such number would ultimately depend on several factors (for example: number of detectors available, the size of the network and its corresponding Bayesian network), we believe that the two-detector scenario allows us to show the behavior of each algorithm for the different attack goals considered.

To show the results, Figure 15 is used to present the detectors picked for each attack goal scenario and the corresponding algorithm used to picked the pair of detectors. Also, a ROC curve is created by averaging the FPR and TPR from the different attack goal scenarios.

The results from this experiment validate the observations from previous experiments. The Greedy algorithm starts by picking the detectors showing the highest benefits regardless of its distance from the attack goal. Still, since all attack nodes are considered as goals, there are several cases where the Greedy algorithm will pick detectors close to the goal and with high benefit values. Therefore, the ROC curve (dashed line) shown in Figure 16 performs just slightly worse than in the case of the FPTAS algorithm. Looking at the choices made by FPTAS, it consistently picks detectors with a high benefit and close to the attack goal considered.

All the experiments validate the intuition that the algorithms can provide good results when multiple detectors are considered together and over the entire Bayesian network. The Greedy algorithm performed well under the scenarios considered, which we believe a good representation of the cases found in real-world systems. Still, as it is shown in Appendix 9, there could be some scenarios for which the Greedy algorithm could produce results as low as half of the optimal solution. The FPTAS allows getting closer to the optimal solution as the algorithm is bounded by a polynomial in the size of the input and the reciprocal of the error parameter. In the experiments the FPTAS always selected a solution equal to or better than the Greedy algorithm, in terms of the benefit provided. As future work, we will test the algorithms under larger scenarios, which will determine the impact of the error parameter on the running time of the scheme.

6 Related Work

Bayesian networks have been used in intrusion detection to perform classification of events. [17] proposed the usage of Bayesian networks to reduce the number of false alarms. Bayesian networks are used to improve the aggregation of different

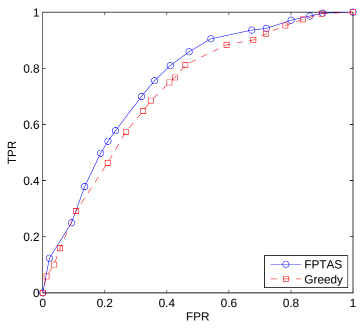


Figure 16. ROC curves for detectors picked by Greedy (dashed line) and FPTAS (solid line) across all different attack goals in E-Commerce Bayesian network.

model outputs and allow integration of additional information. The experimental results show an improvement in the accuracy of detections, compared to threshold-based schemes. [1] studied the use of nave Bayes in intrusion detection, which included a performance comparison with decision trees. Due to similar performance and simpler structure, naïve Bayes is an attractive alternative for intrusion detection. Other researchers have also used nave Bayesian inference for classifying intrusion events [33].

To the best of our knowledge, the problem of determining an appropriate location for detectors has not been systematically explored by the intrusion detection community. However, analogous problems have been studied to some extent in the physical security and the sensor network fields.

[14] developed a Markov Decision Process (MDP) model of how an intruder might try to penetrate the various barriers designed to protect a physical facility. The model output includes the probability of a successful intrusion and the most likely paths for success. These paths provide a basis to determine the location of new barriers to deter a future intrusion.

In the case of sensor networks, the placement problem has been studied to identify multiple phenomena such as determining location of an intrusion [2], contamination source [5], [30], and atmospheric conditions [16]. In these cases, the authors are attempting to detect events of interest, which propagate using some well-defined models, such as through the cluster head en route to a base node. Some of the work [16] is focused on detecting natural events that do not have a malicious motive in avoiding detection. In our case, we deal with malicious adversaries who have an active goal of trying to bypass the security of the system. The adversaries' methods of attacking the system do not follow a well-known model making our problem challenging. As an example of how our solution handles this, we use noise in our BN model to emulate the lack of an accurate attack model.

There are some similarities between the work done in alert correlation and ours, primarily the interest in reducing the number of alerts from an intrusion to be analyzed. Approaches such as [26] have proposed modeling attack scenarios to correlate alerts and identify causal relationships among the alerts. Our work aims to closely integrate the vulnerability analysis into the placement process, whereas the alert correlation proposals have not focused on such scenario.

The idea of using the Bayes' theorem for detector placement is suggested in [29]. No formal definition is given, but several metrics such as accuracy, sensitivity, and specificity are presented to help an administrator make informed choices about placing detectors in a distributed system. These metrics are associated with different areas or sub-networks of the system to help in the decision process.

Many studies have been done on developing performance metrics for the evaluation of intrusion detection systems (IDS), which have influenced our choice of metrics here. [3] showed the applicability of estimation theory in the intrusion detection field and presented the Bayesian detection rate as a metric for the performance of an IDS. His observation that the base rate, and not only the false alarm rate, is an important factor on the Bayesian detection rate, was included in our work by using low base rates as part of probability values in the Bayesian network. The MAFTIA Project [7] proposed precision and recall to effectively determine when vulnerability was exploited in the system. A difference from our approach is that they expand the metrics to consider a set of IDSes and not only a single detector. The idea of using ROC curves to measure performance of intrusion detectors has been explored many times, most recently in [6], [9].

Extensive work with attack graphs has been done for many years. Recent work has concentrated on the problems of generating attack graphs for large networks and automating the process to describe and analyze vulnerabilities and system components to create the graphs. The NetSPA system [11] and MulVal tool [28] are two examples.

7 Conclusions and Future Work

Bayesian networks have proven to be useful tools in representing complex probability distributions, such as in our case of determining the likelihood that an attack goal has been achieved, given evidence from a set of detectors. By using attack graphs and Bayesian inference, we can quantify the overall detection performance in the systems by looking at different choices and placements of detectors and the detection parameter settings. We also quantified the information gain due to a detector as a function of its distance from the attack step. Also, the effectiveness of the Bayesian networks can be affected by imperfect knowledge when defining the conditional probability values. Nevertheless, the Bayesian network exhibits considerable resiliency to these factors, as our experiments showed. Finally, we compared the performance of Greedy and FPTAS algorithms to determine a set of detectors given an attack goal. FPTAS consistently outperformed Greedy, although the latter could be used in scenarios where time constraints exist.

Future work will include looking at the scalability issues of Bayesian networks and its impact on determining the location for a set of detectors in a distributed system. The probability values acquisition problem can be handled by using techniques such as the recursive noisy-OR modeling [20] or using honeynets to monitor the behavior of attackers and compute the corresponding probability values. Experimentation is required to determine its benefits and limitations for our scenario.

Acknowledgement

The authors would like to thanks the anonymous reviewers of a preliminary conference version of this article [23]. Gaspar Modelo-Howard was partly supported by an IFARHU-SENACYT Scholarship from the Republic of Panama. Saurabh Bagchi was partly supported in this work by an endowment grant from Purdue's Center for Education and Research in Information Assurance and Security (CERIAS).

References

[1] N. B. Amor, S. Benferhat, and Z. Elouedi. Naive bayes vs decision trees in intrusion detection systems. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 420–424, New York, NY, USA, 2004. ACM.

[2] F. Anjum, D. Subhadrabandhu, S. Sarkar, and R. Shetty. On optimal placement of intrusion detection modules in sensor networks. *Broadband Networks, International Conference on*, 0:690–699, 2004.

[3] S. Axelsson. The base-rate fallacy and the difficulty of intrusion detection. *ACM Trans. Inf. Syst. Secur.*, 3(3):186–205, 2000.

[4] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[5] T. Berger-Wolf, W. Hart, and J. Saia. Discrete sensor placement problems in distribution networks. *Mathematical and Computer Modelling*, 42(13):1385 – 1396, 2005.

[6] A. A. Cárdenas, J. S. Baras, and K. Seamon. A framework for the evaluation of intrusion detection systems. In *2006 IEEE Symposium on Security and Privacy (S&P 2006), 21-24 May 2006, Berkeley, California, USA*, pages 63–77. IEEE Computer Society, 2006.

[7] M. D. (Ed.). Design of an intrusion-tolerant intrusion detection system. Technical report, Maftia Project, 2002.

[8] B. Foo, Y.-S. Wu, Y.-C. Mao, S. Bagchi, and E. H. Spafford. Adepts: Adaptive intrusion response using attack graphs in an e-commerce environment. In *2005 International Conference on Dependable Systems and Networks (DSN 2005), 28 June - 1 July 2005, Yokohama, Japan, Proceedings*, pages 508–517. IEEE Computer Society, 2005.

[9] G. Gu, P. Fogla, D. Dagon, W. Lee, and B. Skorić. Measuring intrusion detection capability: an information-theoretic approach. In *ASIACCS '06: Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 90–101, New York, NY, USA, 2006. ACM.

[10] O. H. Ibarra and C. E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM*, 22(4):463–468, 1975.

[11] K. Ingols, R. Lippmann, and K. Piwowarski. Practical attack graph generation for network defense. In *ACSAC '06: Computer Security Applications Conference, 22nd Annual*, pages 121–130, December 2006.

[12] F. V. Jensen. *Bayesian networks and decision graphs*. Springer, 2001.

[13] S. Jha, O. Sheyner, and J. M. Wing. Two formal analys s of attack graphs. In *15th IEEE Computer Security Foundations Workshop (CSFW-15 2002), 24-26 June 2002, Cape Breton, Nova Scotia, Canada*, pages 49–63. IEEE Computer Society, 2002.

[14] D. Jones, C. Davis, M. Turnquist, and L. Nozick. Physical security and vulnerability modeling for infrastructure facilities. In *Sandia National Laboratories*, 2005.

[15] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 1 edition, October 2004.

[16] A. Krause, C. Guestrin, A. Gupta, and J. M. Kleinberg. Near-optimal sensor placements: maximizing information while minimizing communication cost. In *Proceedings of the Fifth International Conference on Information Processing in Sensor Networks, IPSN 2006, Nashville, Tennessee, USA, April 19-21, 2006*, pages 2–10. ACM, 2006.

[17] C. Krügel, D. Mutz, W. K. Robertson, and F. Valeur. Bayesian event classification for intrusion detection. In *19th Annual Computer Security Applications Conference (ACSAC 2003), 8-12 December 2003, Las Vegas, NV, USA*, pages 14–23. IEEE Computer Society, 2003.

[18] D. R. Kuhn, T. Walsh, and S. Fires. Nist special publication 800-58 security considerations for voice over ip systems, 2005.

[19] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, 50(2):157–224, 1988.

[20] J. F. Lemmer and D. E. Gossink. Recursive noisy or - a rule for estimating complex probabilistic interactions. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 34(6):2252–2261, 2004.

[21] R. Lippmann, D. Fried, I. Graf, J. Haines, K. Kendall, D. McClung, D. Weber, S. Webster, D. Wyschogrod, R. Cunningham, and M. Zissman. Evaluating intrusion detection systems: the 1998 darpa off-line intrusion detection evaluation. In *DARPA Information Survivability Conference and Exposition, 2000. DISCEX '00. Proceedings*, volume 2, pages 12 –26 vol.2, 2000.

[22] V. Mehta, C. Bartzis, H. Zhu, E. M. Clarke, and J. M. Wing. Ranking attack graphs. In *Recent Advances in Intrusion Detection, 9th International Symposium, RAID 2006, Hamburg, Germany, September 20-22, 2006, Proceedings*, volume 4219 of *Lecture Notes in Computer Science*, pages 127–144. Springer, 2006.

[23] G. Modelo-Howard, S. Bagchi, and G. Lebanon. Determining placement of intrusion detectors for a distributed application through bayesian network modeling. In *RAID '08: Proceedings of the 11th international symposium on Recent Advances in Intrusion Detection*, pages 271–290, Berlin, Heidelberg, 2008. Springer-Verlag.

[24] K. Murphy. Bayes net toolbox for matlab, March 2006.

[25] Netfilter/IPTables. Iptables firewall. http://www.netfilter.org/projects/iptables/, 2008.

[26] P. Ning, Y. Cui, and D. S. Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 245–254. ACM, 2002.

[27] NIST. National vulnerability database. http://nvd.nist.gov/nvd.cfm, 2008.

[28] X. Ou, W. F. Boyer, and M. A. McQueen. A scalable approach to attack graph generation. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 336–345, New York, NY, USA, 2006. ACM.

[29] C. Peikari and A. Chuvakin. *Security Warrior*. O'Reilly Media, 2004.

[30] S. Ray, D. Starobinski, A. Trachtenberg, and R. Ungrangsi. Robust location detection with sensor networks. *IEEE Journal on Selected Areas in Communications*, 22(6):1016–1025, 2004.

[31] SecurityFocus. Bugtraq vulnerability database. http://www.securityfocus.com/vulnerabilities, 2008.

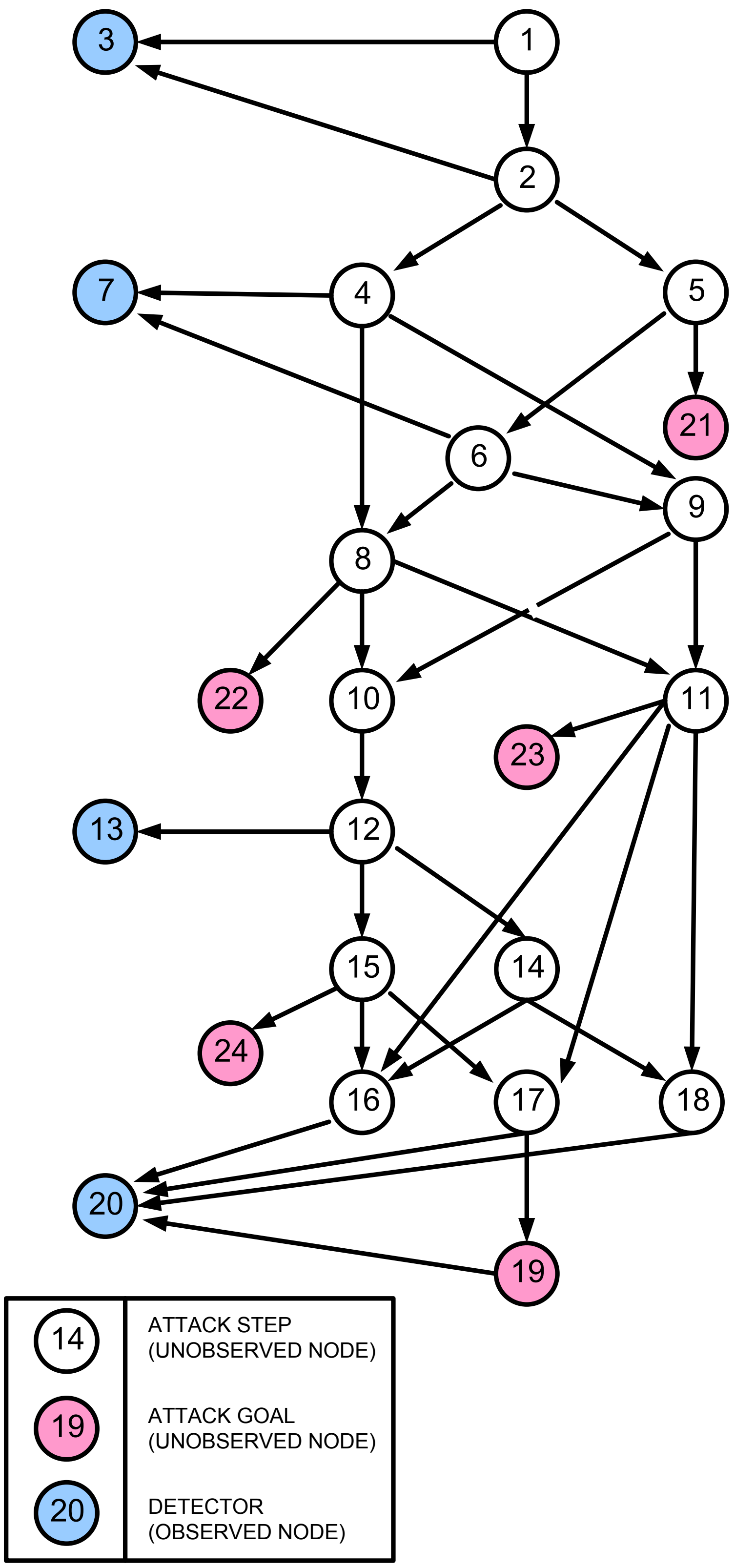
[32] Sourcefire. Snort ids. http://www.snort.org, 2008.

[33] A. Valdes and K. Skinner. Adaptive, model-based monitoring for cyber attack detection. In *Recent Advances in Intrusion Detection, Third International Workshop, RAID 2000, Toulouse, France, October 2-4, 2000, Proceedings*, volume 1907 of *Lecture Notes in Computer Science*, pages 80–92. Springer, 2000.

[34] V. V. Vazirani. *Approximation Algorithms*. Springer, March 2004.

8 Appendix: Description of E-Commerce Bayesian Network

We provide a description of the Bayesian network built for the e-commerce system used in the experiments. It includes a description of each node in the Bayesian network, for the observed and unobserved nodes, as well as the corresponding probability values (shown as tables) associated with each node.



DESCRIPTION OF NODES

NODE 1: Attack step - ping or traceroute to web servers

NODE 2: Attack step - run portscanner on web servers

NODE 3: Detector alert – IPTables

NODE 4: Attack step - exploit ssldump vuln. on web server

NODE 5: Attack step - access web server admin site

NODE 6: Attack step - Brute force admin pwd

NODE 7: Detector alert – Snort

NODE 8: Attack step - Copy hacker tool to web server by using tftp

NODE 9: Attack step - Install vuln scanner on web server

NODE 10: Attack step - Run portscanner on internal network

NODE 11: Attack step - Install sniffer to capture pwds

NODE 12: Attack step - Exploit rpc.statd service on app controller

NODE 13: Detector alert – Libsafe

NODE 14: Attack step - Exploit remote vuln. on MySQL server

NODE 15: Attack step - Brute force root pwd on app controller

NODE 16: Attack step - Run SQLplus to execute queries on tables

NODE 17: Attack step - Connect to MySQL server with admin account

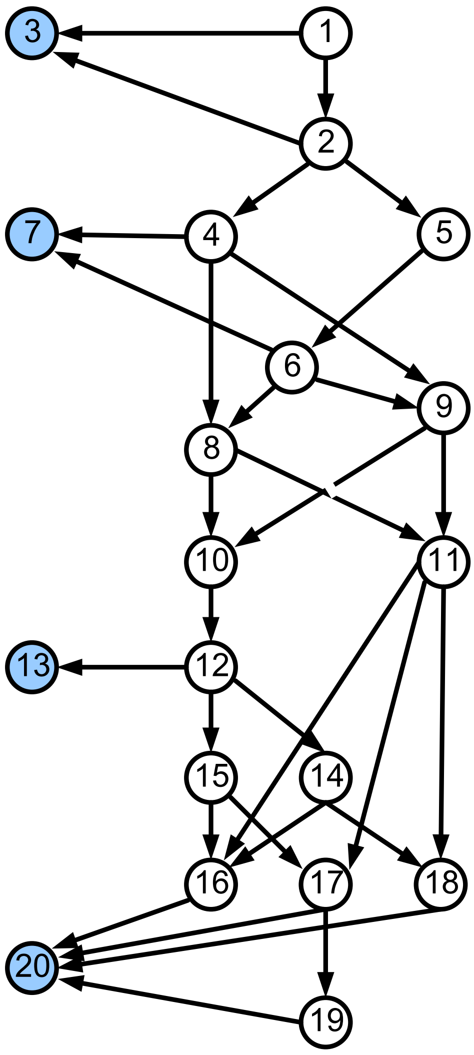
NODE 18: Attack step - Read customer data table

NODE 19: Attack step - Copy customer credit card list

a

NODE 20: Detector alert - Database IDS (Application Security DbProtect)

Figure 17. Bayesian network for the e-commerce system with corresponding description of the nodes. Each node is either an attack step or a detector.



	X1
F	0.99998
T	0.00002

	X2	
X1	F	T
F	0.7	0.3
T	0.3	0.7

		X3	
X2	X1	F	T
F	F	0.9	0.1
T	F	0.6	0.4
F	T	0.4	0.6
T	T	0.2	0.8

	X4	
X2	F	T
F	0.3	0.7
T	0.1	0.9

	X5	
X2	F	T
F	0.3	0.7
T	0.1	0.9

	X6	
X5	F	T
F	0.9	0.1
T	0.7	0.3

		X7	
X6	X4	F	T
F	F	0.95	0.05
T	F	0.7	0.3
F	T	0.3	0.7
T	T	0.15	0.85

		X8	
X6	X4	F	T
F	F	0.3	0.7
T	F	0.2	0.8
F	T	0.2	0.8
T	T	0.1	0.9

		X9	
X6	X4	F	T
F	F	0.99	0.01
T	F	0.3	0.7
F	T	0.3	0.7
T	T	0.01	0.99

		X10	
X9	X8	F	T
F	F	0.99	0.01
T	F	0.1	0.9
F	T	0.1	0.9
T	T	0.01	0.99

		X11	
X9	X8	F	T
F	F	0.99	0.01
T	F	0.1	0.9
F	T	0.1	0.9
T	T	0.01	0.99

	X12	
X10	F	T
F	0.2	0.8
T	0.1	0.9

	X13	
X12	F	T
F	0.9	0.1
T	0.1	0.9

	X14	
X12	F	T
F	0.2	0.8
T	0.1	0.9

	X15	
X12	F	T
F	0.2	0.8
T	0.1	0.9

				X20	
X19	X18	X17	X16	F	T
F	F	F	F	0.99	0.01
T	F	F	F	0.4	0.6
F	T	F	F	0.35	0.65
T	T	F	F	0.3	0.7
F	F	T	F	0.35	0.65
T	F	T	F	0.3	0.7
F	T	T	F	0.3	0.7
T	T	T	F	0.15	0.85
F	F	F	T	0.35	0.65
T	F	F	T	0.3	0.7
F	T	F	T	0.3	0.7
T	T	F	T	0.15	0.85
F	F	T	T	0.3	0.7
T	F	T	T	0.15	0.85
F	T	T	T	0.15	0.85
T	T	T	T	0.01	0.99

			X16	
X15	X14	X11	F	T
F	F	F	0.9	0.1
T	F	F	0.8	0.2
F	T	F	0.7	0.3
T	T	F	0.3	0.7
F	F	T	0.7	0.3
T	F	T	0.3	0.7
F	T	T	0.3	0.7
T	T	T	0.2	0.8

		X17	
X15	X11	F	T
F	F	0.95	0.05
T	F	0.7	0.3
F	T	0.8	0.2
T	T	0.2	0.8

		X18	
X14	X11	F	T
F	F	0.9	0.1
T	F	0.6	0.4
F	T	0.4	0.6
T	T	0.2	0.8

	X19	
X17	F	T
F	0.9	0.1
T	0.1	0.9

Figure 18. Bayesian network for the e-commerce system with the conditional probabilities values used for the experiments.

9 Appendix: Calculation of Approximatio Ratio for Greedy Algorithm

We provide the calculation of the approximation ratio for a *Greedy algorithm* of the 0-1 knapsack problem (KP), for the bounded case (when there is a limited number of items from which to pick and put in the knapsack). The proofs for the calculation of the approximation ratio are adapted from [15] and [34]. We include the proofs in this paper to make the previous proofs more accessible to a systems security audience and to show the thinking process that went on behind our search for FPTAS after having designed the Greedy solution.

KP can be formally defined as the following: given an *instance* with item set N, consisting of n items x_i, each with a profit p_i and weight w_i. The knapsack has a capacity value c. The objective is to select a subset of N such that the total profit of the selected items $(\Sigma_{i=1}^{n} p_i x_i)$ is maximized subject to the corresponding total weight not exceeding the knapsack capacity $(\Sigma_{i=1}^{n} w_i x_i \leq c)$. The optimal solution value is denoted by z^{OPT}.

The idea of the Greedy algorithm with a solution value z^G is to start with an empty knapsack, sort the items in decreasing order according to its profit to weight ratio $\frac{p_i}{w_i}$ and go through the sorted items, adding every item into the knapsack while its capacity is not overwhelmed. The final step is making a comparison between the given solution and the highest profit value of any item. The larger of the two is finally taken and is denoted by z^{mG}. This final step can be considered a modification of the original Greedy algorithm found in literature [33], but necessary to guarantee an approximation ratio of $\frac{1}{2}$ to the optimal solution.

A linear programming relaxation (LKP) is made to compute the approximation ratio, omitting the integer constraint of KP and optimizing instead over all nonnegative real values. Naturally, the optimal solution value z^{LKP} of the relaxed problem is at least as large as the original value z^{OPT} because the set of feasible solutions for the original KP is a subset of the feasible solutions for the relaxed problem. The Greedy algorithm for LKP packs the items in decreasing order of profit-to-weight ratio, similar to the original Greedy algorithm, but with one difference. When adding an item s to the knapsack would cause the capacity c to overflow for the first time, only an appropriate fractional part of the item is used. Item s is referred as the split item, its corresponding profit as p_s and weight as w_s. The split solution, not including the split item, is defined by a profit $\hat{p}$ and weight $\hat{w}$. Therefore, the optimal solution value of LKP is defined as

$$z^{LKP} = \Sigma_{i=1}^{s-1} p_i + (c - \Sigma_{i=1}^{s-1} w_i)\frac{p_s}{w_s}. \tag{2}$$

The value z^{LKP} is an upper bound on the optimal solution for KP. A tighter upper bound U^{LP} for z^{OPT} can be obtained by using the floor of z^{LKP}, i.e. $U_{LP} := \lfloor z^{LKP} \rfloor$, since all data are integers. Then we get the following bounds on z^{LP}:

$$\hat{p} \leq z^{OPT} \leq U_{LKP} \leq z^{LKP} \leq \Sigma_{i=1}^{s} p_i = \hat{p} + p_s = z^G + p_s. \tag{3}$$

Another consequence of these considerations is the following fact:

$$z^{OPT} - z^G \leq z^{OPT} - \hat{p} \leq p_{max}, \tag{4}$$

where p_{max} denotes the largest profit of any item in the set N.

The Greedy algorithm has an approximation ratio of $\frac{1}{2}$ and this bound is tight. As proof, we know from (3) that: $z^{OPT} \leq z^G + p_{max} \leq z^{mG} + z^{mG} = 2z^{mG}$

The tightness of the bound can be shown by the following example. Item 1 is given by $w_1 = 1$, $p_1 = 2$, and $b_1 = 1$ (number of item 1 available). Item 2 is given by $w_2 = p_2 = M$ and $b_2 = 2$. The knapsack capacity is $c = 2M$. The Greedy algorithm would pack item 1 first and then an item 2, reaching a solution value of $2 + M$ while the optimal solution would pack items 2 and would reach a value of $2M$. Choosing M large enough, the ration between the approximate and optimal solution value can be arbitrarily close to $\frac{1}{2}$.