Bayesian Event Classification for Intrusion Detection

Christopher Kruegel Darren Mutz William Robertson Fredrik Valeur

Reliable Software Group
University of California, Santa Barbara
{chris, dhm, wkr, fredrik}@cs.ucsb.edu

Abstract

Intrusion detection systems (IDSs) attempt to identify attacks by comparing collected data to predefined signatures known to be malicious (misuse-based IDSs) or to a model of legal behavior (anomaly-based IDSs). Anomaly-based approaches have the advantage of being able to detect previously unknown attacks, but they suffer from the difficulty of building robust models of acceptable behavior which may result in a large number of false alarms. Almost all current anomaly-based intrusion detection systems classify an input event as normal or anomalous by analyzing its features, utilizing a number of different models. A decision for an input event is made by aggregating the results of all employed models.

We have identified two reasons for the large number of false alarms, caused by incorrect classification of events in current systems. One is the simplistic aggregation of model outputs in the decision phase. Often, only the sum of the model results is calculated and compared to a threshold. The other reason is the lack of integration of additional information into the decision process. This additional information can be related to the models, such as the confidence in a model's output, or can be extracted from external sources. To mitigate these shortcomings, we propose an event classification scheme that is based on Bayesian networks. Bayesian networks improve the aggregation of different model outputs and allow one to seamlessly incorporate additional information. Experimental results show that the accuracy of the event classification process is significantly improved using our proposed approach.

1 Introduction

Intrusion detection can be defined as the process of identifying malicious behavior that targets a network and its resources. Intrusion detection systems have traditionally been classified as either *misuse-based* or *anomaly-based*. Systems that use misuse-based techniques contain a number of attack descriptions, or 'signatures', that are matched against a stream of audit data looking for evidence of the modeled attacks. The audit data can be gathered from the network [18, 25], from the operating system [7, 17], or from application [23] log files. Signature-based systems have the advantage that they usually generate few false positives (i.e., incorrectly flagging an event as malicious when it is legitimate). Unfortunately, they can only detect those attacks that have been previously specified. That is, they cannot detect intrusions for which they do not have a defined signature.

Anomaly-based techniques follow an approach that is complementary with respect to misuse detection. These approaches rely on models, or profiles, of the normal behavior of users [4, 8], applications [5, 26] and network traffic [10, 14, 15]. Deviations from the established models are interpreted as attacks. Anomaly detection systems have the advantage that they are able to identify previously unknown attacks. By defining an expected, normal state, any abnormal behavior can be detected, whether it is part of the threat model or not. This capability should make anomaly-based systems a preferred choice. However, the advantage of being able to detect previously unknown attacks is usually paid for in terms of a large number of false positives. This can make the system unusable by flooding and eventually desensitizing the system administrator with large numbers of incorrect alerts.

We have identified two main problems that contribute to the large number of false positives. First, the decision whether an event should be classified as anomalous or as normal is made in a simplistic way. Anomaly detection systems usually contain a collection of models that evaluate different features of an event. These models return an anomaly score or a probability value that reflects the 'normality' of this event according to their current profiles. However, the system is faced with the task of *aggregating* the different model outputs into a single, final result. The difficulty is the fact that this aggregation is not easy to perform, especially when the individual model outputs

differ significantly. In most current systems, the problem is solved by calculating the sum of the outputs and comparing it to a static threshold. The disadvantage of this approach is the fact that this threshold has to be small enough to detect malicious events that only manifest themselves in a single anomalous feature (i.e., only one model outputs a high value indicating malicious behavior). This can lead to false positives, because events with many features that deviate slightly from the profile might receive aggregated scores that exceed the threshold.

The second problem of anomaly-based systems is that they cannot distinguish between anomalous behavior caused by unusual but legitimate actions and activity that is the manifestation of an attack. This leads to the situation where any deviation from normal behavior is reported as suspicious, ignoring potential additional information that might suggest otherwise. Such additional information can be external to the system, received from system health monitors (e.g., CPU utilization, memory usage, process status) or other intrusion detection sensors. Consider the example of an IDS that monitors a web server by analyzing the system calls that the server process invokes. A sudden jump in CPU utilization and a continuous increase of the memory allocated by the server process can corroborate the belief that a certain system call contains traces of a denial-of-service attack. Additional information can also be directly related to the models, such as the confidence in a model output. Depending on the site-specific structure of input events, certain features might not be suitable to distinguish between legitimate and malicious activity. In such a case, the confidence in the output of the model based on these features should be reduced.

We propose to mitigate the two problems described above by replacing the simple, threshold-based decision process with a *Bayesian network*. Instead of calculating the sum of individual model outputs and comparing the result to a threshold, we utilize a Bayesian decision process to classify input events. This process allows us to seamlessly incorporate available additional information into the detection decision and to aggregate different model outputs in a more meaningful way. The contribution of this paper is the description of this decision process, a novel method of event classification in anomaly-based intrusion detection systems. Experimental results confirm that our approach is capable of significantly reducing the number of false alarms.

The paper is structured as follows. Section 2 provides background information on Bayesian networks to help the reader in understanding the rest of the paper. Section 3 describes related work and discusses previous efforts to utilize Bayesian techniques for intrusion detection. Section 4 introduces our approach of Bayesian event classification. Section 5 describes the system implementation and provides details of the underlying anomaly-based models. Section 6 shows experimental results that confirm that our solution is more accurate (i.e., reports fewer false alerts) than previous approaches. Finally, Section 7 briefly concludes.

2 Bayesian Networks

A Bayesian network is used to model a domain containing uncertainty [9, 13]. It is a directed acyclic graph (DAG) where each node represents a discrete random variable of interest. Each node contains the states of the random variable that it represents and a conditional probability table (CPT). The CPT of a node contains probabilities of the node being in a specific state given the states of its parents. The parent-child relationship between nodes in a Bayesian network indicates the direction of causality between the corresponding variables. That is, the variable represented by the child node is causally dependent on the ones represented by its parents.

Consider the following example where a farmer has a bottle of milk that can be either infected or clean. She also has a test that can determine with a high probability whether the milk is infected or not (i.e., the outcome of the test is either positive or negative). This situation can be represented with two random boolean variables, `infected` and `positive`. The variable `infected` is true when the milk is actually infected and false otherwise. The variable `positive` is true when the test claims that the milk is infected and false when the outcome of the test is negative. Note that it is possible that the milk is clean when the test has a positive outcome and vice versa.

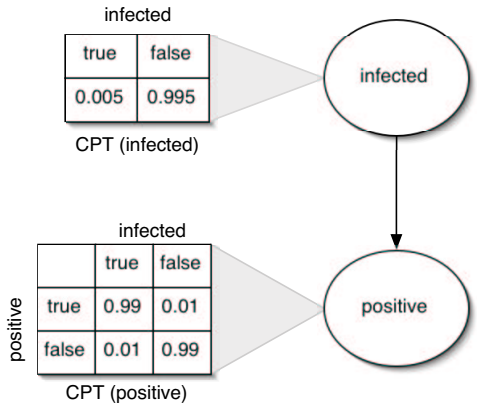

Figure 1. Bayesian Network and CPTs

A possible Bayesian network that models this situation is shown in Figure 1. The two random variables are represented as two nodes in the network. It is assumed that the farmer knows the CPT for the variable `positive`, that is, the probability of a positive result given that the milk is

infected and the probability of a positive test result given that the milk is clean. She also knows the CPT for the variable `infected`, which states the probability that a bottle contains infected milk. The arrow from the `infected` to the `positive` node indicates a causal relationship between the respective variables. In this case, we expect that the outcome of the test is dependent on the true state of the milk (infected or clean). Variables without parents are not influenced directly by other variables.

The node that represents the outcome of the test in Figure 1 is often called an *information* variable. The state of these variables are usually given or can be measured in a straightforward manner. The node that represents the actual state of the milk is called a hypothesis variable. The states of such variables cannot be obtained immediately. The purpose of a Bayesian network is to allow one to calculate the probability of the hypothesis variable(s) given the evidence gathered from information variables. In our example, the farmer might want to calculate the probability that the milk is infected given a positive test result. By entering the evidence (e.g., `positive` is true) into the Bayesian network, the probability that `infected` is true can be derived. The numerical value for this probability, called the a-posteriori probability given the support of the observed evidence, is 0.33. Intuitively, one would expect a higher value, especially when considering that the test is very accurate. However, the low initial probability of the milk being infected, called the a-priori probability before any observations are made, explains this result.

In the domain of intrusion detection, information nodes are associated with measurable properties of input events or corresponding model outputs. The hypothesis node is a classification that determines the state of the event – whether it is anomalous or not.

3 Related Work

Axelsson [1] wrote a well-known paper that uses the Bayesian rule of conditional probability[1] to point out the implications of the *base-rate fallacy* for intrusion detection. Similar to our example with the infected milk in the previous section, he observed that the positive result of a very accurate test (such as the test for infection) does not necessarily imply a high probability of the hypothesis variable to be true (i.e., the milk to be actually infected). For the domain of intrusion detection, this finding means that even tests or models that identify malicious events very accurately may raise many false alarms because the a-priori probability of an attack in the input data stream is usually very low. Although Axelsson's paper is only remotely related to our work through the application of Bayesian probability theory, it clearly demonstrates the difficulty and necessity of dealing with false alarms.

Several researchers have adapted ideas from Bayesian statistics to create models for anomaly-based IDSs. In [16], a behavior model is introduced that uses Bayesian techniques to obtain model parameters with maximal a-posteriori probabilities. In [6], a model is presented that simulates an intelligent attacker using Bayesian techniques to create a plan of goal-directed actions. Their work is similar to ours to the extent that Bayesian statistics is applied. Their work differs from our approach because it uses Bayes' rule to optimize or create models, while we utilize Bayesian networks to classify events based on model outputs and additional information from the environment.

Two anomaly-based IDS have been proposed that use naïve Bayesian networks to classify input events based on the output of several models. A naïve Bayesian network is a restricted network that has only two layers and assumes complete independence between the information nodes (i.e., the random variables that can be observed and measured). These limitations result in a tree-shaped network with a single hypothesis node (root node) that has arrows pointing to a number of information nodes (child nodes). All child nodes have exactly one parent node, that is, the root node, and no other causal relationship between nodes are permitted. In [24], a naïve Bayesian network (shown in Figure 2) is employed to perform intrusion detection on network events. In [19], a system is described that detects malicious proxylets (executable code) in active networks.

Unfortunately, the classification capability of a naïve Bayesian network is identical to a threshold-based system that computes the sum of the outputs obtained from the child nodes. This is due to the fact that all models (i.e., the child nodes) operate independently and only influence the probability of the root node. This single probability value at the root node can be represented by a threshold in traditional approaches. In addition, the restriction of having a single parent node complicates the incorporation of additional information. This is because variables that represent such information cannot be linked directly to the nodes representing the model outputs.

Alternatively, we propose an event classification that makes full use of Bayesian networks. This allows us to model inter-model dependencies (i.e., dropping the assumptions of independent child nodes) and to integrate additional data such as model confidence (i.e., dropping the restriction of at most a single parent node). Our experiments show that these extensions improve the quality of the decision process and significantly reduce the number of false alarms.

[1] The Bayesian rule of conditional probability is given as $p(B|A) = \frac{p(A|B)p(B)}{p(A)}$.

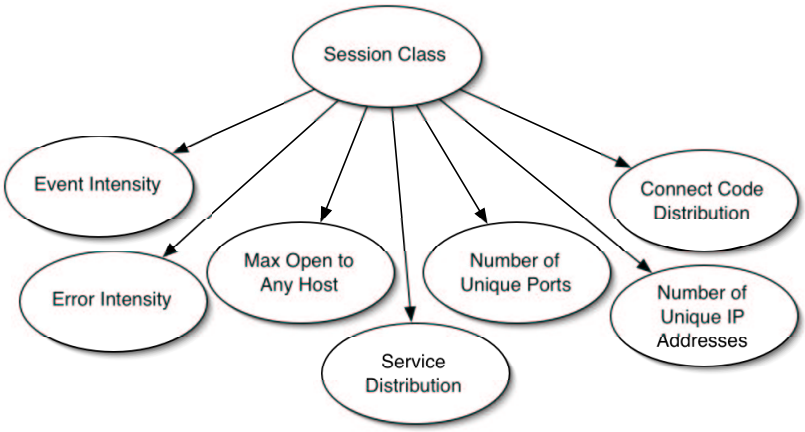

Figure 2. Naïve Bayesian Network

4 System Design

Given an ordered stream of input events $S = \{e_1, e_2, \dots\}$, the task of the event classification mechanism is to decide for each $e_i \in S$ whether it is normal or anomalous. This decision is based on the outputs $\{o_i | i = 1 \dots k\}$ of k models $M = \{m_1, \dots, m_k\}$ and possibly additional information I. Each model $m_i \in M$ analyzes one or more features (or properties) of a given input event and compares the event's feature(s) to the model's previously established profile (i.e., the description that specifies the normal features or properties). The result of this comparison is the output value o_i that characterizes the deviation of the event's features from the expected 'normal' values, one for each of the k models. The restriction of a single return value o_i per model does not result in a loss of generality. Every model that returns more that one result can be easily represented by multiple logical models, each returning a single output.

Given these definitions, the event classification can be defined more formally as a function EC that, for a certain input event e, accepts as parameters the corresponding model outputs $\{o_i | i = 1 \dots k\}$ and additional information I. The result of the event classification function is a binary value that identifies the input event e as normal or anomalous. That is, for a certain event e, the event classification function EC is defined as follows.

$$EC(o_1, o_2, \dots, o_k, I) = \{\text{normal}, \text{anomalous}\} \quad (1)$$

In most current anomaly-based intrusion detection systems, EC is a simple function that calculates the sum of the o_i values (often referred to as anomaly scores) and compares the result to a threshold, represented by I. That is, EC is defined as follows.

$$EC(o_1, o_2, \dots, o_k, I) = \begin{cases} e \text{ is normal :} & \sum_{i=1}^{k} o_i \leq I \\ e \text{ is anomalous :} & \sum_{i=1}^{k} o_i > I \end{cases} \quad (2)$$

We propose to replace this simple summation scheme by a Bayesian network. Our network consists of a root node (i.e., hypothesis node) that represents a variable with two states, namely normal and anomalous. In addition, we introduce one child node for each model to capture the model's respective outputs $\{o_i | i = 1 \dots k\}$. The root node is connected to each child node, reflecting the fact that the model outputs depend on the input event – that is, the outputs are expected to be different when the input event is anomalous and when it is normal.

Depending on the domain, causal dependencies between models are identified and appropriate links are introduced into the network. Under certain circumstances, it is possible that the outputs of two models are correlated. This can be as simple as a positive or a negative correlation (i.e., one anomalous feature makes it more or less likely that another one is also anomalous), but could also be more sophisticated such as the situation where the value of a certain feature indicates that the quality of a test performed by another model is reduced. Section 5 shows examples of model dependencies that we have identified for our intrusion detection system and Section 6 presents experimental results that demonstrate that incorporating dependencies reduces the number of incorrect classifications.

Additional information sources might indicate that anomalous behavior is in fact legitimate or might support the decision that the host is under attack. This could be information from other intrusion detection systems or system

health monitors (e.g., CPU utilization, memory usage, process status). An important piece of additional information is the confidence value associated with each model. Depending on the input events that are utilized for establishing the profile, a certain feature might not be very suitable to distinguish between attacks and regular behavior. It might be the case that the same values of a feature appear in both regular behavior and attacks or that the variance of a feature is very high. In these situations, it is useful to reduce the influence of the model output on the final decision. The confidence in the output of a model is an indication of the expected accuracy of this model. In our Bayesian network, each model confidence is represented by a node that is connected to its corresponding model node. Note that these additional nodes require a non-naïve network because each model node has at least two parent nodes (the root node and the corresponding confidence node). Section 5 discusses the models that we utilize for our intrusion detection system and provides details about their confidence levels.

Another possibility is to model dependencies between events in the input stream. Attacks tend to manifest themselves in bursts of suspicious events. Therefore, it might be useful to include a node in the Bayesian network that keeps track of recent anomalies. However, this extension has not been implemented and is left for future work.

5 System Implementation

We have implemented an intrusion detection system that analyzes operating system calls to detect attacks against daemon applications and setuid programs on machines running Linux or Solaris. In contrast to the work by Forrest [5, 26], we do not perform detection on a sequence of system calls but on individual system calls and their arguments. Each system call invocation performed by a monitored application is translated into an input event, represented by a feature vector. A feature vector captures information specific to each system call such as the system call number, its return code, and its arguments (such as file system paths, mode bit-fields, and user/process credentials).

The feature vector serves as input to the analysis process of the anomaly detection models. Each model evaluates one or more features of the input event and outputs a value that reflects the deviation of this event's features from its profile. We have developed four different models, described below in more detail, that analyze individual system call arguments (also called system call parameters). Three models are particularly designed to characterize features of string-type parameters, while one can be used for arbitrary argument types. For every monitored system call, we bind a number of models to each of its arguments.

The task of the event classification process is to determine whether a certain system call is anomalous, given the outputs of the individual models for all arguments. A simple event classifier was implemented that aggregates the model outputs and compares the result to a threshold. We also implemented our proposed Bayesian event classification scheme and observed a significant decrease in the number of false alarms.

In order to provide a suitable input event stream on multiple platforms, a modular event provider architecture was created to abstract away the platform-specific details of system call logging. We implemented a Linux auditing facility that converts Snare [21] audit data into feature vectors and a tool that offers a similar functionality for Solaris' Basic Security Module (BSM) [3].

5.1 Models

This section briefly describes our underlying models with their detection mechanisms and motivates why our chosen characterization is useful. In the following sections, we discuss the model confidence and the dependencies between models introduced in our system.

String Length

In many cases, the length of a string can be used to detect anomalous input. System call argument strings are usually relatively short and human-readable. However, the situation might look different when malicious input is present. For example, to overflow a buffer, it is often necessary to ship the shell code and additional padding, depending on the length of the target. As a consequence, a string can contain up to several hundred bytes. The goal of this model is to approximate the actual but unknown distribution of the lengths of a string argument and detect instances that significantly deviate from the observed normal behavior. Clearly, we cannot expect that the probability density function of the underlying real distribution follows a smooth curve. We also have to assume that it has a large variance. Nevertheless, the model is able to identify significant deviations.

Character Distribution

The character distribution model captures the concept of a 'normal' system call parameter string by looking at its character distribution. It is based on the observation that regular strings contain mostly printable, human-readable characters. A large percentage of characters in these strings are drawn from a small subset of the 256 possible 8-bit values (mainly from letters, numbers, and a few special characters). Like in English text, the characters are not uniformly distributed, but occur with different frequencies. The analysis is based only on the frequency values themselves and does not rely on the distributions of individual characters. That is, it does not matter whether the character with the

most occurrences is an 'a' or a '@'. For a regular parameter, one can expect that the *sorted*, relative frequencies slowly decrease in value. In case of manifestations of attacks, however, these frequencies can drop extremely fast (because of a peak caused by a very high frequency of a single character) or barely (in case of a nearly uniform character distribution). The 'normal' character distribution is determined as the average of the character distributions of the strings encountered during the training phase. The model output for a new string instance is calculated using the Pearson χ^2-test statistical test [2] that estimates the similarity of the new character distribution to the one derived as the average of the training set.

Structure

Often the manifestation of an exploit is immediately visible as unusually long strings, or as strings that contain repetitions of non-printable characters. Such anomalies are easily identifiable by the two mechanisms explained above. There are situations, however, when an attacker is able to craft her attack in a manner that makes the manifestation appear more regular. For example, to exploit a vulnerability, it might not be necessary to inject long chunks of exploit code. As another example, repetitions of non-printable characters, often found in the sled of a buffer overflow, can be replaced by constructs that behave similarly but contain only printable characters.

In such situations, it is necessary to use a more detailed model of the string that shows the trace of the attack. This model can be acquired by analyzing the string's structure. For our purposes, the structure of a parameter means the regular grammar that describes all its normal, legitimate values. When structural inference is applied to a set of strings, the result has to be a grammar that can derive at least all training examples. Unfortunately, there is no single grammar that can be uniquely defined for a set of sample inputs. When no negative examples are given (i.e., elements that should not be derivable by the grammar), it is always possible to create either a grammar that contains exactly the training data or a grammar that allows one to derive arbitrary strings. The first case is called over-simplification, as the resulting grammar is only able to derive the learned input without providing any level of abstraction. This means that no new information is deduced. The second case is a form of over-generalization; although the grammar is capable of producing all possible strings, there is no structural information left.

The basic approach used for our structural inference is to generalize the grammar as long as it seems to be 'reasonable' and stop before too much structural information is lost. The notion of reasonable generalization is formalized using hidden Markov models and Bayesian probability [22]. The output value of this model depends on whether a new input string can be derived from the grammar or not.

Token Finder

The purpose of the token finder model is to determine whether the values of a system call parameter are drawn from a limited set of possible alternatives (i.e., they are tokens or elements of an enumeration). An application often passes identical values via APIs, such as flags or handles. When an attack changes the normal flow of execution and branches into maliciously injected code, such constraints are often violated. When no such enumeration can be identified in the training data, it is assumed that the values are randomly drawn from the argument type's value domain (i.e., random values for every system call). The token finder technique can be applied to any parameter type, but it is mostly used for numerical values. In case that the monitored values are tokens drawn from an enumeration, every new value is expected to appear in the set of known identifiers. Otherwise, the token finder cannot provide any useful information.

5.2 Model Confidence

The confidence that the system has in the output of a model should be an important factor in the event classification process. When a model claims a high confidence in its output, this model's anomaly score should clearly have a higher impact on the final decision than the score of a model that can only provide low-confidence information. In traditional systems, the confidence is often neglected or approximated with static weights. When a model is expected to produce more accurate results, it receives a higher a-priori weight. However, this is not sufficient, as the confidence in a model can vary depending on the training data used to create the corresponding profile. Consider, for example, the token finder model. When this model detects an enumeration during the learning phase, its anomaly scores are considered highly accurate. When random identifiers are assumed, the anomaly score is not meaningful. With statically assigned weights, this distinction cannot be made. Although it is possible to choose between two static weights in the case of the token finder, the situation becomes more complicated with other models. Therefore, a seamless integration of dynamic weights that are calculated after the training phase is desirable.

We take the model confidences into account by including a confidence node for every model. Each confidence node in the Bayesian network has a link to the node which represents its corresponding model. The conditional probability tables are adjusted so that the model output has a significant influence on the decision when the confidence is highest and no influence on the final result when the confidence

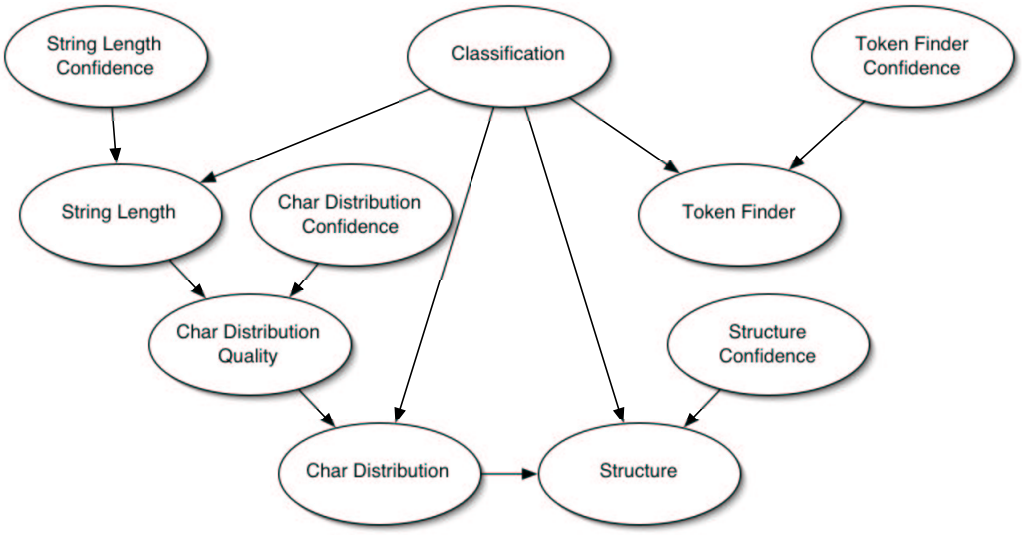

Figure 3. Bayesian Network for `open` and `execve` System Calls

is lowest. The model confidence is represented as one of five discrete levels: very high, high, medium, low and none. When models create their profiles of normal behavior, the variance of the input training data is evaluated. When the variance of the analyzed feature is high, a low confidence value is assumed. When a small, coherent set of feature values is observed during the training, the confidence in the correctness of the model output is high.

5.3 Bayesian Network

Figure 3 shows the structure of the Bayesian networks for the `open` and `execve` system call. Both system calls have two parameters and are monitored by our intrusion detection system. The three string models (String Length, Character Distribution and Structure) are attached to the first string parameter (file path and name in the case of the `open` call, execution arguments in the case of the `execve` call). The token finder is attached to the numerical parameter in the case of the `open` call (mode flags) and to another string parameter in case of the `execve` call (program image executed). Similar but simpler networks are used for other monitored system calls that have only a single argument. A different Bayesian network instance is utilized for every system call; however, most of these networks have an identical structure.

In addition to the structure of the Bayesian networks, conditional probability tables (CPTs) were specified for each node. We used our domain-specific knowledge to estimate appropriate probability values for the various tables. For each node, one has to provide the probabilities for all states of the corresponding variable, conditionally dependent on the states of all parent nodes. When a suitable structure of the network is chosen, these probabilities are mostly intuitive and can be determined in a sufficiently accurate way by a domain expert. Note that we have not tuned the CPTs in any way for our experiments. The probabilities were selected before the evaluation was started and were not modified thereafter.

The output of the models is a real value in the interval [0,1] that describes the deviation of the input event feature(s) from the profiles. This value is mapped onto one of five possible states associated with each model node in the network. The mapping of a continuous function output onto a number of different states is called *discretization*. This process is required to keep the CPTs of the Bayesian network manageable and to allow efficient calculations of the probabilities at the hypothesis nodes. As shown in Table 1, model outputs close to zero indicate normal features while outputs close to one indicate anomalous ones.

Anomaly Score Range	Level
$[0.00, 0.50)$	Normal
$[0.50, 0.75)$	Uncommon
$[0.75, 0.90)$	Irregular
$[0.90, 0.95)$	Suspicious
$[0.95, 1.00]$	Very Suspicious

Table 1. Anomaly Score Intervals

The Bayesian network in Figure 3 shows the two model dependencies that we have introduced for our intrusion detection system. One dependency connects the node cor-

responding to the output of the string length model to the quality of the character distribution (which is also influenced by the confidence in the output of the character distribution). The mediating node '`Char Distribution Quality`' in our network expresses the idea that the quality of the anomaly score calculated by the character distribution is not only dependent on the 'a-priori' confidence of the model in the quality of its profile, but also on the length of the string that is currently analyzed. When this string is very short, the quality of the statistical test that assesses the character distribution is significantly reduced. This is reflected by the conditional probability tables of the '`Char Distribution Quality`' node.

The other dependency is introduced between the nodes representing the character distribution and the structure model. The reason is that an 'abnormal' character distribution is likely to be reflected in a structure that does not conform to the learned grammar. This is an example of a simple positive correlation of output values between models.

During the analysis phase, the output of the four models and their confidences are entered as evidence into the Bayesian network. Then, the probabilities of the two states (normal, anomalous) associated with the root node (`Classification`) are calculated. When the probability that an event is anomalous is high enough, an alarm is raised. Note that the requirement that the probability value being 'high enough' to raise an alarm could be interpreted as a threshold as well. However, the difference is that this probability value directly expresses the probability that a certain event is an attack, given the specific structure of the Bayesian network. The sum of model outputs in a threshold-based system, on the other hand, is not necessarily proportional to the probability of an event being an attack. It is possible, due to the assumption of independence of model outputs and the potential lack of confidence information in these systems, that the sum of the outputs is increasing while the probability of an attack is, in fact, decreasing.

Both the threshold in a traditional system and the notion of a sufficiently high probability for raising an alarm in the Bayesian approach can be utilized to tune the sensitivity of the intrusion detection system. However, the result of the Bayesian network directly reports the probability that an event is anomalous, given the model outputs and the structure of the network, while a simple summation of model outputs is only an approximation of this probability. The difference between the exact value and the approximation is important, and accounts for a significant number of false alarms, as shown in Section 6.

5.4 Bayesian Network Library - Smile

We implemented the models as part of a C++ library and utilized a Bayesian statistics library called Smile [20], developed by the Decision Systems Laboratory at the University of Pittsburgh, for our event classification module. Smile was the best choice among the available statistical software given the requirements that the package must implement actual Bayesian networks rather than performing Bayesian statistical analysis and must provide a usable API rather than solely a GUI.

A problem with Smile is the fact that the source is not freely available, and that its licensing precludes one from using it in any open source software. Therefore, we wrote adapter classes to provide an abstraction layer between our modules and Smile. This allows for Smile's replacement should the issues become too great a liability.

The problem of belief propagation – that is, the calculation of probabilities at the hypothesis nodes when evidence is entered at information nodes – is, in general, NP-hard [9]. Despite this fact, Smile implements efficient algorithms that can solve almost all problems in a reasonable amount of time. Note also that the NP-hard calculations need to be done only once, given that the information and hypothesis nodes do not change. In addition, these calculations can be done off-line. The computational cost during run-time to evaluate particular values is linear in the number of nodes in the network. Our proposed solution takes advantage of this fact as the sets of information and hypothesis nodes remain static. This allows our system to analyze a stream of system calls in real-time without incurring noticeable computational or memory overhead.

6 Evaluation

For the purposes of evaluating our approach, we used the MIT Lincoln Labs 1999 data set [11]. This data set consists of a series of network packet dumps and BSM system call records which have been widely used for intrusion detection system development and evaluation. We used data recorded during two attack-free weeks to train our models and then ran the system on the complete test data that was recorded during the two following weeks. Although several aspects of the Lincoln Labs data have been criticized, it still remains the most used large-scale data set to evaluate intrusion detection systems [12].

The truth file provided by MIT Lincoln Labs lists all attacks carried out against the network installation during the two week test period. When analyzing the attacks, it turned out that many of these were reconnaissance attempts such as network scans or port sweeps, which are only visible in the network dumps, and do not not leave any traces at the system call level. Therefore, we could not detect them with

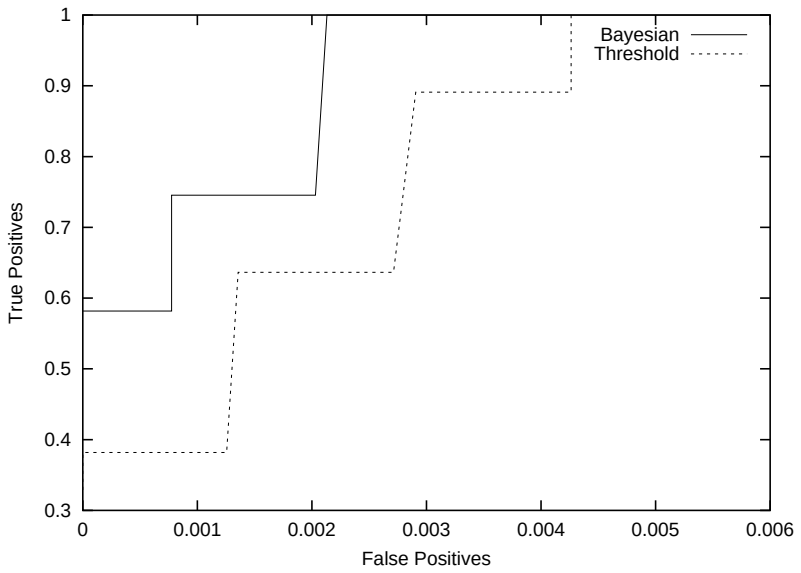

Figure 4. ROC Comparison between Bayesian Network and Threshold-based System

our approach, which is limited to the analysis of system call parameters.

Another class of attacks are policy violations. This class of attacks contains intrusions that do not exploit a weakness of the system itself, but rather exploit a mistake that an administrator made in setting up the access control mechanism. These attacks are not visible to our system either, as information is leaked by 'normal' but unintended use of the system.

The most interesting class of attacks are those that exploit a vulnerability in a remote or local service and allow an intruder to elevate her privileges. The MIT Lincoln Labs data contains several instances of attacks that try to exploit vulnerabilities in four different programs: `eject`, `fbconfig`, `fdformat`, and `ps`. Figure 4 shows the Receiver Operating Characteristic (ROC) curves for our system when monitoring these applications. The ROC of a classifier shows its performance as a trade off between selectivity and sensitivity; a curve of the false positive rate versus the true positive rate is plotted, while a sensitivity or threshold parameter is varied. Ideally, a classifier has a true positive rate of 1 and a false positive rate of 0. The ROC curve for the Bayesian event classifier is plotted by varying the 'anomalous' probability value that is required for an event to be reported as an attack. The ROC curve for the threshold-based classifier is determined by varying the threshold that is compared to the sum of outputs. The graphs show that both classifiers output some false alarms when all attacks are correctly detected. However, the Bayesian approach consistently performs better – when all attacks are correctly detected (i.e., the true positive rate is 1), it only reports half as many false positives. Note that the shapes of the curves are not a consequence of an insufficient number of data points. The horizontal and vertical lines contain intermediate points, reflecting changes in either the false positive or the true positive rate alone.

When analyzing the false positives raised by both classification approaches, we observed that the Bayesian scheme always reported a subset of the false alarms raised by the threshold-based mechanism. The false positives common to both mechanisms are caused by system call invocations that have arguments which significantly deviate from all examples encountered during the training phase. This is due to the fact that the training data for these particular system calls was very homogeneous, leading to profiles that were very sensitive to changes. During the detection phase, legitimate system calls with significantly different arguments were observed. This resulted in their incorrect classification.

The system calls that were reported as anomalous by the threshold-based approach but correctly classified as normal by the Bayesian scheme were instances with short string arguments. As explained in Section 5.3, short strings can significantly influence the quality of the character distribution model, causing it to report incorrect anomalies. This problem is addressed by the Bayesian network using the mediating '`Char Distribution Quality`' node (see Figure 3), correctly evaluating these system calls as normal.

7 Conclusions

In this paper, we presented a novel method for performing Bayesian classification of input events for intrusion detection. We have improved upon the naïve threshold-based schemes traditionally used to combine model outputs by employing Bayesian networks. This allows us to naturally incorporate model confidence and dependencies between models into the event classification process. The experimental results show that a significant reduction of false alerts was achieved. When all attacks in our test data set are detected, the Bayesian event classification reports only half as many false alerts as the traditional approach, based on the same model outputs.

Acknowledgments

This research was supported by the Army Research Office, under agreement DAAD19-01-1-0484. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Army Research Office, or the U.S. Government.

References

[1] S. Axelsson. The Base-Rate Fallacy and its Implications for the Difficulty of Intrusion Detection. In *6th ACM Conference on Computer and Communications Security*, 1999.

[2] P. Billingsley. *Probability and Measure*. Wiley-Interscience, 3 edition, April 1995.

[3] Basic Security Module Guide – SunSHIELD BSM. `http://docs.sun.com/db/doc/802-5757`.

[4] D. Denning. An Intrusion Detection Model. *IEEE Transactions on Software Engineering*, 13(2):222–232, Feb. 1987.

[5] S. Forrest. A Sense of Self for UNIX Processes. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 120–128, Oakland, CA, May 1996.

[6] R. Goldman. A Stochastic Model for Intrusions. In *Symposium on Recent Advances in Intrusion Detection (RAID)*, 2002.

[7] K. Ilgun. USTAT: A Real-time Intrusion Detection System for UNIX. In *Proceedings of the IEEE Symposium on Research on Security and Privacy*, Oakland, CA, May 1993.

[8] H. S. Javitz and A. Valdes. The SRI IDES Statistical Anomaly Detector. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 1991.

[9] F. Jensen. *Bayesian Networks and Decision Graphs*. Springer, New York, USA, 2001.

[10] C. Kruegel, T. Toth, and E. Kirda. Service Specific Anomaly Detection for Network Intrusion Detection. In *Symposium on Applied Computing (SAC)*. ACM Scientific Press, March 2002.

[11] M. Lincoln Labs. DARPA Intrusion Detection Evaluation. `http://www.ll.mit.edu/IST/ideval`, 1999.

[12] J. McHugh. Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evalautions as Performed by Lincoln Laboratory. *ACM Transaction on Information and System Security*, 3(4), November 2000.

[13] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1997.

[14] P. Porras and P. Neumann. EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. In *Proceedings of the 1997 National Information Systems Security Conference*, October 1997.

[15] P. A. Porras and A. Valdes. Live traffic analysis of TCP/IP gateways. In *Proceedings of the 1998 ISOC Symposium on Network and Distributed System Security (NDSS'98)*, San Diego, CA, 1998.

[16] R. Puttini, Z. Marrakchi, and L. Me. Bayesian Classification Model for Real-Time Intrusion Detection. In *22th International Workshop on Bayesian Inference and Maximum Entropy Methods in Science and Engineering*, 2002.

[17] RealSecure. `http://www.iss.net/products_services/enterprise_protection`.

[18] M. Roesch. Snort - Lightweight Intrusion Detection for Networks. In *USENIX Lisa 99*, 1999.

[19] A. A. Sebyala, T. Olukemi, and L. Sacks. Active Platform Security through Intrusion Detection Using Naive Bayesian Network for Anomaly Detection. In *London Communications Symposium*, 2002.

[20] Smile: Structural Modeling, Inference, and Learning Engine. `http://www.sis.pitt.edu/~genie/`.

[21] SNARE - System iNtrusion Analysis and Reporting Environment. `http://www.intersectalliance.com/projects/Snare`.

[22] A. Stolcke and S. Omohundro. Inducing probabilistic grammars by bayesian model merging. In *International Conference on Grammatical Inference*, 1994.

[23] Swatch: Simple Watchdog. `http://swatch.sourceforge.net`.

[24] A. Valdes and K. Skinner. Adaptive, Model-based Monitoring for Cyber Attack Detection. In *Proceedings of RAID 2000*, Tolouse, France, October 2000.

[25] G. Vigna and R. A. Kemmerer. NetSTAT: A Network-based Intrusion Detection System. *Journal of Computer Security*, 7(1):37–71, 1999.

[26] C. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *IEEE Symposium on Security and Privacy*, pages 133–145, 1999.