AI Lessons Learned from Experiments in Insider Threat Detection

Alexander Liu and **Cheryl Martin** and **Tom Hetherington** and **Sara Matzner**
Cyber Information Assurance & Decision Support Branch
Signal and Information Sciences Laboratory
Applied Research Laboratories
The University of Texas at Austin
{aliu, cmartin, tomh, matzner}@arlut.utexas.edu

Abstract

Although AI has been successfully applied to many different domains with different characteristics, the task of applying a solution that is successful in one problem domain to a different domain remains far from automatic. Even the simpler task of applying a solution to a related but different domain is problematic. In this paper, we discuss various problems that can occur when trying to solve a classification problem in a new problem domain (insider threat) by trying previously successful approaches in a related problem domain (intrusion detection). We examine in depth why our results in the new problem domain did not reflect the successes from the previous domain. We conclude with various lessons learned that can be used when approaching a new problem domain.

Introduction

Daily reminders of successful applications of AI have become ubiquitous and pervasive in the form of successful search engines, spam filters, and recommender systems. AI and its branches have been used successfully in varied realms such as facial recognition, robotic soccer, and bioinformatics. One attractive promise of AI is that the same algorithm has the potential to be deployed successfully in different domains. The ability for the same algorithm to be used for marketing purposes or to detect cancerous tumors seems wonderful and fantastic.

Unfortunately, applying AI techniques to a specific domain often comes with its own unique pitfalls and challenges, often stemming from sources beyond the actual algorithms used. There are many things that can go wrong when trying to apply AI to a new problem domain, making transferal of solutions from one domain to another far from automatic. These pitfalls could range from simple problems such as having to retune various thresholds and parameters to much larger problems such as using a technique that is simply not suitable for a particular domain. The sources of challenges extend beyond simply selecting an appropriate algorithm, and they include such problems as dataset selection and/or creation, feature selection and/or extraction, and learning about the problem domain.

In this paper, we first discuss a basic framework for solving new problems in a new domain by describing basic steps one can take when applying machine learning techniques to a new problem area. After briefly describing some of our past work on insider threat detection, we use our insider threat detection experiments as a running example of things that can go wrong at various steps of our basic framework, focusing on various problems encountered and lessons learned. Finally, we end with a summary of lessons learned and general conclusions.

A Framework for Solving New Problems

In this section, we discuss a basic framework for applying AI techniques in a new problem domain. The goal of this section is not to claim that our approach is necessarily novel or unique, but simply to lay out a framework in order to facilitate discussion. Our framework is particularly suited to the application of supervised and unsupervised machine learning techniques. Other AI techniques which fall outside this realm may have additional considerations.

The five basic steps when applying machine learning to a new problem are as follows:

1. Problem definition
2. Dataset creation
3. Selecting a feature representation
4. Application of machine learning algorithm
5. Analysis of results

Throughout, there is the overarching task of learning enough about the problem domain to successfully apply the steps in our framework. Let us briefly discuss each of the five steps listed above.

Problem definition

The first step in our framework is to define the problem that needs to be solved. It is often important to define the problem clearly and concretely such that a meaningful and feasible solution exists. That is, if one simply begins with a nebulous or ill-posed question, it may be difficult to decide when or if the goal has actually been accomplished. Realistically, if one's work is funded by a particular client or customer, the problem definition will also largely be influenced by their needs and concerns.

Dataset creation

Once a problem is defined, one needs to find an appropriate source of data. This data could come from actual observations from the problem domain, simulations of the problem at hand, or some other dataset appropriately modified to represent the problem domain. Issues such as the total amount of data, the amount of data available from each class, or the amount of available labels will also affect subsequent stages.

Selecting features

Intimately tied with dataset creation is the task of feature representation, selection and extraction. This step is quite often difficult in a new problem domain since the possible features can be extremely numerous. Feature selection and extraction are both dependent on the type of data available and also on the type of machine learning algorithm being applied. Other issues such as missing values, whether features are categorical or numerical, and the dimensionality of the datasets created need to be considered as well.

The goal of feature selection is to choose features such that the various different classes or concepts one is trying to represent are as separable in the feature space as possible. Whether or not a chosen feature space separates the classes one is trying to learn often cannot be determined until later stages of this framework (namely choosing a machine learning algorithm and analysis of results), leading to a possible need for refinement of the selected features.

Application of machine learning algorithm

The actual machine learning algorithm is an important choice when approaching a new problem domain, but it is only one of several important steps. In this stage, one must make such choices as to whether there is enough labeled data to support a supervised, semi-supervised, or unsupervised approach. One must also make the necessary decision of what type of classifier or clustering algorithm to use, as well as the values for any necessary parameters. The problem domain once again needs to be taken into consideration since factors such as the inherent dimensionality of the problem space affect one's choice of algorithm and choice of parameters.

Analysis of results

Finally, one needs to analyze the results to determine how well the current approach is performing. We include this step in our framework instead of setting it aside as an impartial, external process since any implications or discoveries in this stage of the framework may influence reformulations or improvements in the previous steps. In particular, in initial forays into a new problem domain, the steps in the framework may be repeated iteratively, with information from this stage directly feeding into future improvements of the previous stages.

The Insider Threat Detection Problem

Throughout the rest of this paper, we use some of our initial experiments in the realm of insider threat detection as an example to illustrate what can go wrong in the context of the framework described above. We will try to limit application-specific discussion only to the parts of our experiments that are necessary for this paper. Interested readers should refer to (Liu *et al.* 2005) for additional details, including related work, results, and further discussion.

Insider threat occurs when a user who is authorized to access a system abuses those privileges to accomplish or attempt to accomplish malicious activity that is not allowed under normal rules and regulations. For example, a user might be authorized to access the company payroll, but is clearly not authorized to e-mail the payroll to a competing company.

In terms of our five step framework, our insider threat experiments were set up as follows.

Problem definition

Although some researchers have previously investigated automatically detecting insider threat (e.g., (Nguyen, Reiher, & Kuenning 2003) and (Maxion 2003)), very little work has been published on the domain of insider threat detection. A domain related to insider threat detection is intrusion detection, which refers to the task of automatically detecting when a computer system has been breached or compromised by an external attacker. In contrast to insider threat detection, intrusion detection has been well studied in the academic community.

Our initial goal was to leverage past work on intrusion detection and attempt to transfer solutions from one related domain to another. In both intrusion detection and insider threat detection, one typically assumes that the amount of normal activity far outnumbers the amount of malicious activity. In the case of intrusion detection, this means that most activity is performed by legitimate users, while in the case of insider threat detection, this means that most activity (all of which is by legitimate users) is itself legitimate. In addition to being rare, one also assumes that unauthorized activity is unusual and therefore separable from legal activity in the selected feature space. Anomaly detection is therefore well suited for this problem since an anomaly detection algorithm is particularly adept at finding rare, unusual activity. Past work such as (Forrest *et al.* 1996) and (Hofmeyr, Forrest, & Somayaji 1998) have successfully applied anomaly detection techniques to the problem of intrusion detection.

We attempted to perform anomaly detection on Unix system call traces. Unix system calls are useful for both intrusion detection and insider threat detection for two reasons. First, they provide the most detailed, comprehensive view of system activity. Second, system calls can be made the most resistant to malicious tampering.

Thus, our problem was to analyze the efficacy of anomaly detection on Unix system calls for the problem of insider threat detection. For our initial experiments, we restricted ourselves to a Unix environment, mainly to make the next stage of dataset creation easier to manage.

Dataset creation

In the realm of insider threat detection, there are no publicly available datasets available for experimentation. Thus, we needed to simulate the activities of a malicious insider on

a Unix system and capture their activity. We modified the Snare [1] package in order to monitor, record, and label the system calls created by a user's activity. One of the authors then proceeded to execute both normal activity and malicious activity in order to create appropriate datasets. Each captured system call was labeled based on whether it was part of normal or malicious activity and also labeled based on the general type of activity being executed (e.g., e-mail, database management).

In our final experiments, we tested our approach on six datasets. Each dataset typically consisted of both malicious and normal versions of a certain class of actions. For example, the "e-mail" dataset consisted of both normal e-mail activity as well as malicious e-mail activity. The only exception is a dataset called "misc", which consisted of various Unix command line activity (e.g. "ls" or "chmod"). While all of these actions are executed at the Unix command line, they differ in their purpose and the programs called during execution. In comparison, the other datasets would use the same programs to accomplish both normal and malicious activity.

Selecting features

Since the main goal set forth in our problem definition was to examine the efficacy of intrusion detection techniques in insider threat detection, we selected three ways of representing data that have previously been used in anomaly detection of Unix system call data for the purpose of intrusion detection.

The first feature representation is an n-gram representation (Forrest *et al.* 1996),(Hofmeyr, Forrest, & Somayaji 1998). In this representation, each data point consists of a single n-gram of system call names. While simple, this feature representation and related variants have proven to be both effective and popular in the realm of intrusion detection.

In our second choice of feature representation, each data point is a count or histogram of the types of system calls that have been executed in some window of S system calls. This representation is very similar in nature to the popular "bag-of-words" model used to represent text. While much less popular than the n-gram representation, this second feature representation can be found in works such as (Liao & Vemuri 2002) and (Kang, Fuller, & Honavar 2005).

Our third feature representation has been used in work such as (Kruegel *et al.* 2003) and (Tandon & Chan 2003). In this representation, each data point consists of an individual system call and the various parameters and attributes associated with that system call. For example, a data point could consist of a system call of type "open" and associated attributes such as its associated return code and process name. In this representation, only system calls of the same type can be directly compared since different types of system calls may have a different number of associated parameters. In addition, the same parameter used by one system call might have a completely different meaning when used by a different system call type.

[1] available at http://www.intersectalliance.com/projects/Snare/

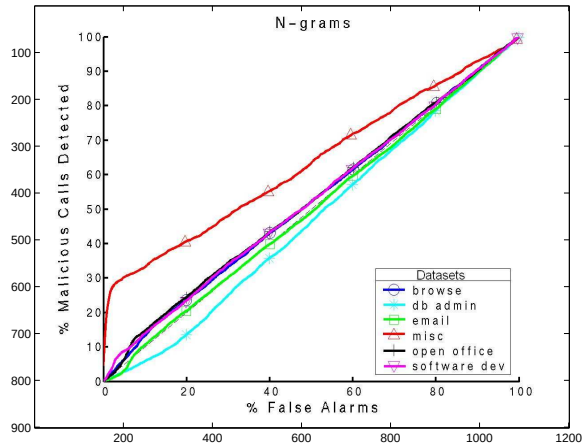

Figure 1: ROC results for n-gram feature representation

Application of machine learning algorithm

Like in the previous stage, our choice of machine learning algorithm to apply to the data was restricted by our problem definition. In this step of the framework, we applied an anomaly detection algorithm to the three data representations described above. Specifically, we used Bay's Orca outlier detection package (Bay & Schwabacher 2003).

We experimented with anomaly detection in both a supervised and unsupervised scenario. In intrusion detection, supervised anomaly detection refers to the case where a model is created on purely normal or purely malicious data. Since normal data is much easier to obtain, typically the model is created on purely normal data. Unsupervised anomaly detection refers to the case where both normal and malicious activity is present in an unlabeled dataset, and the task is to identify the malicious activity. We adopted these conventions in our supervised and unsupervised experiments as well.

Analysis of results

As mentioned, analysis of results can be used to improve previous steps in our problem solving framework. We will discuss some of these iterative improvements throughout the next section, but first, let us examine some of our final results as well as the implications of these results.

Figures 1 through 4 contain results for our supervised experiments on six different datasets. Figure 1 contains results using the n-gram representation, figure 2 contains results for the histogram representation, and figures 3 and 4 are examples of results obtained for the parameter based representation on two different types of system calls ("open" and "fork"). The curves shown are ROC curves which indicate how many false alarms/false positives one must generate in order to capture a certain percentage of the true positives/malicious activity. Ideally, one should capture 100% of the malicious activity before capturing any false alarms. That is, we want a curve to approach the upper left corner of the graph. Since the axes of our graph indicate percentages of true/false positives captured, the expected curve of

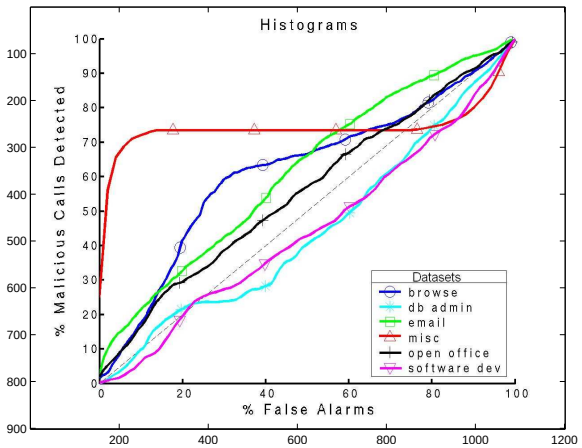

Figure 2: ROC results for histogram feature representation

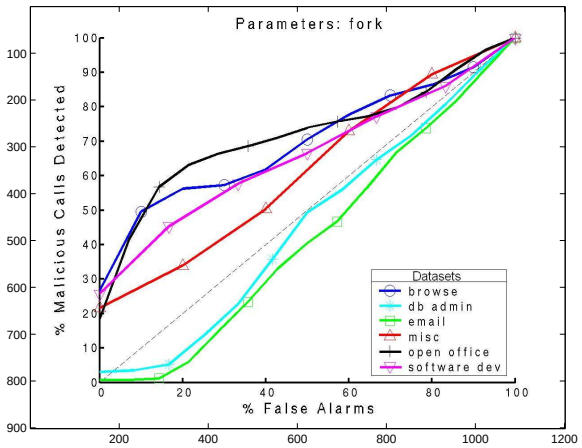

Figure 3: ROC results for parameter feature representation("fork" system calls)

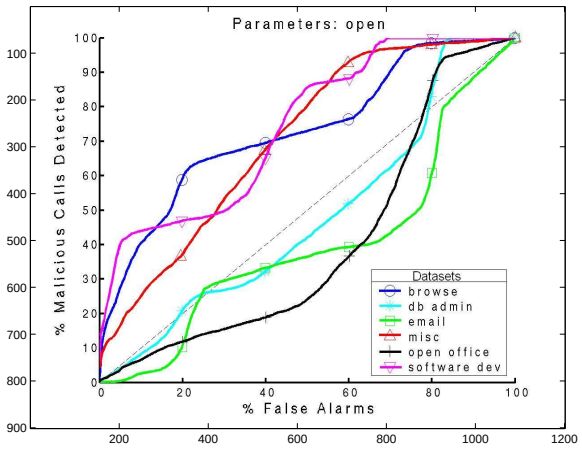

Figure 4: ROC results for parameter feature representation("open" system calls)

a purely random approach would be a diagonal line from the bottom left to the top right in our graphs. This expected curve is represented in the graphs as a dotted line.

For the n-gram representation in particular, we see that on five of our datasets, we are basically performing randomly. For the histogram representation, we perform than random better on some datasets, but poorer than random on two others. In both cases, we perform quite well on the "misc" dataset. It turns out that the "misc" dataset is very similar to datasets used in intrusion detection, while the other five datasets are not. Using the parameter based feature representation, we are able to obtain results that are promising but still far from ideal. However, these results are promising on more than just the "misc" dataset.

In short, the implications of these results is that, despite similarities in the problem domains, intrusion detection and insider threat detection require different approaches. In particular, our conjecture is that sequential information is important in the realm of intrusion detection, while the information provided by the parameters (i.e. the context in which a system call is executed) is more important in the realm of insider threat detection. That is, n-grams and histograms, which capture sequential data, cannot detect insider threat, while the parameter based representations show greater sensitivity in detecting malicious insider activity. Thus, in moving from intrusion detection to insider threat detection, we find that the feature representation stage is the stage of the framework that needs the most modification.

Problems and Challenges

Below, we discuss various problems that can go wrong at steps two through four in our basic framework given a problem defined in step one. In particular, we use our experiments in the domain of insider threat detection as an example of various problems encountered and lessons learned during experimentation. We divide our discussion into problems that can be completely or partially avoided and problems that cannot be avoided.

Dataset creation

Avoidable problems When entering a new domain, one must select an appropriate source of data in order to study problems in that domain. Thus, one must either find and record an existing source of data or possibly create a data source. It may be necessary to create several datasets as one iteratively refines the type of data being captured. For example, one may have to recapture an entire dataset if one later learns that a new type of information is needed in order to create a particular feature. In our own experiments, we had to both create our own datasets as well as iterate over several versions of datasets as we refined both the type of data captured and the way that it was captured.

In any domain, there are several issues that must be addressed during dataset creation. We concentrated on two issues we felt to be most important: 1) getting an accurate and unbiased data source and 2) collecting the data once an appropriate source had been identified. Since we did not have access to any real system call logs containing insider activ-

ity, we needed to simulate both normal activity and malicious insider activity. We spent most of our initial efforts on creating the software necessary to log and capture user behavior. We also concentrated on populating the types of malicious activity in our datasets in order to more fully represent the types of activities viable for a malicious insider. We created and discarded several initial datasets as the project progressed because 1) we either added abilities to our system (e.g., a new insider exploit or the capability to monitor a new system call type) or because 2) we found bugs in how the system had captured data (e.g., the commands used to label individual system calls were also being captured by the system instead of remaining transparent to the system).

Unfortunately, we later discovered that by focusing on the problems of identifying and capturing an appropriate data source, we had allowed some less obvious problems to creep into the datasets. Two issues we had not addressed were the issue of class balance and the issue of appropriately grouping normal and malicious activity.

Many of our initial datasets did not correctly balance the ratio between normal and malicious activity that one might expect in the real world. In some of our initial datasets, almost 55% of the entire dataset consisted of malicious activity. While it is unknown how much time malicious insiders spend on malicious activity in the real world, one usually assumes that malicious activity makes up only a small amount of their total behavior. From case studies, we estimated that malicious insiders spend far less than 10% of their time at work on unauthorized activity. Therefore we adjusted the ratio of normal and malicious activity in the latest datasets.

A second problem that required dataset revision was that we originally failed to include positive and negative examples of the same activity in the same dataset. For example, in one dataset, our simulated insider would e-mail a protected document, an unauthorized activity. However, the dataset contained no examples of normal e-mail behavior. Our initial results on these datasets were much better than they should have been because e-mail itself was rare behavior. That is, the anomaly detection stage was catching the malicious e-mail not because malicious and normal e-mail appear different from each other, but because e-mail appears different compared to all other behavior. In comparison, our final set of six datasets consisted of both positive and negative examples of similar behavior.

Unavoidable problems If there is no way to get the data needed to perform experiments in a new domain, then the problem of finding an appropriate dataset cannot be avoided. This has been especially true in the insider threat detection domain, where it is difficult to obtain actual data either because of proprietary information, privacy issues, or simply because organizations do not wish to admit they have had problems due to insiders. This is one of the biggest obstacles for researchers wanting to study insider threat detection. The only recourse is to simulate this data by either creating data as we did or modifying other datasets to approximate this domain. For example, Maxion modified the usage logs of several users in order to approximate a masquerade detection dataset (Maxion 2003). In general, without publicly available datasets, academic research into a particular domain is stifled simply because there is no data on which to test new hypotheses and methods.

Selecting a feature representation

Avoidable problems In order to guide feature selection in a new domain, one must be aware of past approaches in the new domain and related domains. In our case, the most closely related domain was that of intrusion detection. Thus, we selected three feature representations that had been successful in the realm of intrusion detection as initial feature representations in the realm of insider threat detection.

Knowledge about the problem domain is also needed to guide the feature selection process. In our case, we were recording Unix system calls. However, if one were to record all possible system calls that occur, the system would not be able to cope with the volume of system calls available. One member of our research team was a domain expert in the area, so we left the selection of appropriate system calls to record to him.

Unavoidable problems One unavoidable limitation that may be encountered when performing feature extraction is that the nature of the data itself may limit the number of viable feature representations. For example, one may encounter missing, categorical, or ordinal data that limits the types of features one can use.

In our experiments, we used Unix system call data along with associated parameters and attributes of the system calls. All of this data is essentially categorical. In contrast, many machine learning algorithms are more suited to handle numerical features. We considered many outlier detection algorithms which work well for numerical data but performed poorly when using the categorical data in our experiments. Techniques to transform categorical features into numerical features (e.g., replacing instances of categorical features with their frequency of occurrence or replacing categorical features with multiple binary indicators) also produced poor results when combined with outlier detection.

Application of machine learning algorithm

Avoidable problems As in the feature selection stage, one way to guide the selection of an appropriate machine learning algorithm is to be aware of past work in this domain. As stated throughout this paper, one can also select algorithms based on similarities between the problem at hand and other problem domains.

One issue with using any algorithm in a new domain is tuning any appropriate parameters and thresholds. One aspect of anomaly detection we experimented with was how best to find distances between individual data points. While the histogram representation had numerical features, the n-gram and parameter based representations did not. Thus, we needed to select a distance metric applicable to categorical data.

For the histogram representation, we were able to use standard Euclidean distance. For the n-gram and parameter based representation, we finally settled on Hamming distance. In our experiments, the Hamming distance between

two data points was simply the number of features that differed between the two data points. Experimentally, we found that the use of Hamming distance had some undesirable side effects. For example, in the parameter based representation, using Hamming distance to find outliers resulted in finding unusual combinations of system call parameters, particularly those system calls with parameters which occur only once or twice in the entire dataset. These unusual combinations of system call parameters were just as likely from many normal activities as well as malicious activities, meaning that both normal and malicious activities were flagged as outliers.

In another example using the parameter based representation, the same set of five features were different for almost all data points. Since we were using Hamming distance, many of the top outliers were therefore a distance of five from all other points. On closer investigation, we found that these five features tended to be the same for system calls created by the same process. The outlier detector was therefore finding system calls that only occurred once in a given process. To address this, we started to separate data points before comparison and performed outlier detection only on data points with the same process ID. This helped solve our problem when data points were similar because they came from the same process, but many data points were still very similar because Hamming distance was unable to differentiate between them.

Unavoidable problems Machine learning algorithms have certain biases and assumptions built into the algorithm. It is not always possible to choose a new algorithm that has a different bias. When using machine learning algorithms in real world solutions, one may be forced to use certain machine learning algorithms due to constraints on processing speed or ease of interpretability. For example, not all algorithms can be used in an online setting, while the need for easily interpretable decisions might cause one to use decision trees or rule learning algorithms.

Our problem definition restricted our choice of machine learning algorithm to an anomaly detection algorithm. In using anomaly detection to detect malicious activity, one makes two assumptions. The first is that malicious actions are outliers when compared to normal activity in the feature space. That is, normal and malicious data are separable in the selected feature space. The second assumption is that there are very few outliers in the entire dataset, and that most of these outliers correspond to malicious activity. If either of these assumptions is not true, then anomaly detection algorithms will perform poorly.

While not tested in our experiments, we propose a simple way for a malicious insider to take advantage of these two biases. Let us discuss how a malicious insider could potentially fool both an unsupervised anomaly detection algorithm and a supervised anomaly detection algorithm.

In an unsupervised anomaly detection setting, a determined malicious insider could simply violate the assumption that malicious activity occurs rarely. For example, by spending half of their time performing malicious activity and half of their time performing normal activity, it might be difficult to differentiate this user from a normal user who divides their time between two unrelated normal activities.

In a supervised anomaly detection setting, there is the issue of obtaining training data free of malicious activity. If we are collecting data for each user, it is difficult to easily ensure that the data is completely free of malicious activity. In particular, when collecting data for users who are actually malicious insiders, it may be possible that their malicious activity will be captured in the training data and mistakenly labeled as "normal". Even if the data is completely clean, a malicious insider could still potentially fool the system by flooding the system with false positives by running rare but purely normal activity. While these false positives are being investigated, the malicious insider could run a particular set of malicious activity multiple times. Thus, by making certain types of normal activity rare and by making malicious activity less rare, the malicious insider again potentially circumvents the anomaly detection algorithm by violating its inherent assumptions.

Conclusion

When applying AI techniques in new problem domains, two major themes became clear throughout our experiments. The first is that one must iterate through the steps described in our problem framework. That is, when entering a new problem domain, one may expect to iterate through these steps several times since unforeseen problems or solutions in one stage may affect the stages both before and after it. For example, the desire to use a new type of feature may trigger changes to both the data collection stage and also to the choice of an appropriate machine learning algorithm.

The second theme we encountered through our experiments is the importance of learning about the problem domain. In particular, learning about the problem domain is needed to guide appropriate dataset capture and feature selection. Not surprisingly, learning from problems encountered during dataset capture and feature selection also leads to a deeper understanding of the problem domain. While the simple mantra of "learn about the problem domain" is oft repeated in textbooks and courses on AI, it is indeed an idea that cannot be overemphasized when entering a new and unfamiliar problem domain.

Transferring AI solutions from one problem domain remains problematic. While there have been successes in knowledge transfer and knowledge reuse techniques, problems still arise due to inherent differences between many domains. In this paper, we have tried to list some of the problems that could go wrong when transferring solutions between the two related domains of intrusion detection and insider threat detection.

We illustrated potential problems by listing pitfalls we encountered in our own foray into a new and unfamiliar domain. One interesting question is how far one can get in new domains if all avoidable problems are avoided. Unfortunately, there are still few automated "black box" approaches in AI, and it is usually only through hindsight that avoidable problems are identified. We hope that by listing what went wrong in our own experiments that this paper will be informative to other researchers who try to adapt solutions

from one domain to another. In this way, practioners will be able to see a greater number of avoidable problems via foresight and collective experience instead of hindsight and serendipity.

Acknowledgements

This work was funded by the Advanced Research Development Activity (ARDA), under the Advanced IC Information Assurance Research Thrust. Alexander Liu was supported by a Thrust Fellowship from the University of Texas at Austin. Thanks to Karl Fisher and Joydeep Ghosh for useful comments.

References

Bay, S., and Schwabacher, M. 2003. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Forrest, S.; Hofmeyr, S.; Somayaji, A.; and Longstaff, T. A. 1996. A sense of self for unix processes. In *IEEE Symposium on Computer Security and Privacy*.

Hofmeyr, S.; Forrest, S.; and Somayaji, A. 1998. Intrusion detection using sequences of system calls. In *Journal of Computer Security*, 151–180.

Kang, D.-K.; Fuller, D.; and Honavar, V. 2005. Learning classifiers for misuse and anomaly detection using a bag of system calls representation. In *2005 IEEE Workshop on Information Assurance*.

Kruegel, C.; Mutz, D.; Valeur, F.; and Vigna, G. 2003. On the detection of anomalous system call arguments. In *8th European Symposium on Research in Computer Security*.

Liao, Y., and Vemuri, V. R. 2002. Using text categorization techniques for intrusion detection. In *USENIX Security Symposium*.

Liu, A.; Martin, C.; Hetherington, T.; and Matzner, S. 2005. A comparison of system call feature representations for insider threat detection. In *2005 IEEE Workshop on Information Assurance*.

Maxion, R. 2003. Masquerade detection using enriched command lines. In *International Conference on Dependable Systems & Networks*.

Nguyen, N.; Reiher, P.; and Kuenning, G. H. 2003. Detecting insider threats by monitoring system call activity. In *2003 IEEE Workshop on Information Assurance*.

Tandon, G., and Chan, P. 2003. Learning rules from system call arguments and sequences for anomaly detection. In *ICDM Workshop on Data Mining for Computer Security*.