2

# ANALYZING THE PERFORMANCE OF PROGRAM BEHAVIOR PROFILING FOR INTRUSION DETECTION

Anup K. Ghosh and Aaron Schwartzbard

**Abstract** This paper presents an analysis of a simple equality matching algorithm that detects intrusions against systems by profiling the behavior of programs. The premise for this work is that abnormally behaving programs are a primary indicator of computer intrusions. The analysis uses data collected by the Air Force Research Laboratory and provided by the MIT Lincoln Laboratory under the 1998 DARPA Intrusion Detection Evaluation program. Labeled attack sessions are embedded in normal background traffic so that the analysis can measure the probability of detection simultaneously with the probability of false alarm. The analysis uses Receiver Operator Characteristic (ROC) curves to show the performance of the system in terms of the probability of false alarm and probability of detection for different operating points.

**Keywords:** Anomalous noise, anomaly detection, equality matching, intrusion detection, N-gram, performance evaluation, ROC curve

## 1. INTRODUCTION

Intrusion detection generally can be defined as a method to detect any unauthorized use of a system. Intrusions are commonly considered to be attacks from unauthorized outside entities to gain access to a target system. Once inside, an intruder can illegally use resources, view confidential material, or

*This work is sponsored under Defense Advanced Research Projects Agency (DARPA) Contract DAAH01-98-C-R145. THE VIEWS AND CONCLUSIONS CONTAINED IN THIS DOCUMENT ARE THOSE OF THE AUTHORS AND SHOULD NOT BE INTERPRETED AS REPRESENTING THE OFFICIAL POLICIES, EITHER EXPRESSED OR IMPLIED, OF THE DEFENSE ADVANCED RESEARCH PROJECTS AGENCY OR THE U.S. GOVERNMENT.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35508-5_22

V. Atluri et al. (eds.), *Research Advances in Database and Information Systems Security*

even try to subvert or harm the system. The term is less commonly used, yet equally applicable, to describe attacks from within the target system by authorized insiders who attempt to abuse their privileges or acquire higher privileges in violation of security policies.

Desiring the ability to detect novel, unknown attacks against systems, an anomaly detection approach is employed in this research. Unlike traditional anomaly detection approaches, this research applies anomaly detection at the system process level. Rather than profiling the normal behavior of users, the approach here is to profile the normal behavior of executing processes. Monitoring at the process level abstracts away users' idiosyncrasies such that abnormal process behavior can be detected irrespective of individual users' behavior. Having baselined the normal behavior of a process, deviations from the normal behavior can be used to detect attempted exploitations of the program. A corollary of this approach is that attempted misusage of programs that does not alter normal program behavior is not flagged.

## 2. BUILDING PROGRAM BEHAVIOR PROFILES

The data collected for building program behavior profiles is the set of BSM events made by the programs under analysis and recorded by Sun Microsystem's Basic Security Module (BSM) system auditing facility for Solaris. The data was collected over a period of weeks in a controlled environment on an internal Air Force Research Laboratory (AFRL) network and provided to us by MIT's Lincoln Laboratory under the DARPA 1998 Intrusion Detection Evaluation program[1]. Attack sessions are embedded within normal background traffic sessions, which allows us to measure both correct detection and false alarm rates simultaneously.

The research described in [3] indicates that short sequences of system calls provide a compact and adequate "signature of self" that can be used to distinguish between normal ("self") behavior and abnormal (dangerous "non-self") behavior. The conclusions in [3] show that the data is still too inconclusive to determine if equality matching of system calls is a viable approach to intrusion detection. The detailed analyses performed in this paper in a controlled, scientific study coordinated with MIT's Lincoln Laboratory demonstrate the viability of this approach when extended to exploit temporal locality.

The session data provided by Lincoln Laboratory was labeled for training purposes such that intrusive sessions can be distinguished from normal sessions. Using these session labels, the intrusion sessions are separated from normal sessions. Because our approach is based on anomaly detection, we use

[1]See `www.ll.mit.edu/IST/ideval/index.html` for a summary of the program.

only the normal sessions for training. A database is constructed for each program. The database is simply a table that consists of every unique $N$-gram of BSM events that occurred during any execution, as well as a frequency count for each $N$-gram. An $N$-gram is a sequence of $N$ events. For instance, given the following snippet from a pre-processed file of BSM events (where single letters represent events),

```
[pid 1] A;[pid 1] B;[pid 2] C;[pid 1] D;
[pid 1] B;[pid 2] A;[pid 1] D;[pid 1] B;
[pid 2] E;[pid 2] B;[pid 2] D;,
```

and given an $N$-gram size of two, a table of all observed $N$-grams could be constructed. First, the two processes represented in the file must be de-interleaved, producing the following execution traces:

```
pid 1: A B D B D B
pid 2: C A E B D.
```

A sliding window with a size of two (as specified above) would be passed over the execution trace of the first process, to collect all $N$-grams with a size of two. As a result, the following table would be created. The same process is

*Table 2.1* Table of normal behavior profile for the first process for an $N$-gram size of 2 (total number of unique $N$-grams is 3, total number of $N$-grams is 5).

| *BSM N-grams* | *Frequency* |
|---|---|
| A B | 1 |
| B D | 2 |
| D B | 2 |

applied to the execution trace of the second process, and the results are added to the same table to produce table 2.2.

The table shows the unique strings that are created from a program's BSM profile with their associated frequency. Each table characterizes the normal behavior of the program under non-intrusive usage. Tables such as this one are built programmatically for all programs under analysis. In this study, tables for 155 different UNIX programs were constructed. The tables are used by the intrusion detection algorithm to detect anomalous behavior.

*Table 2.2* Table of normal behavior profile for a program for an $N$-gram size of 2 (total number of unique $N$-grams is 6, total number of $N$-grams is 9).

| *BSM N-grams* | *Frequency* |
|---|---|
| A B | 1 |
| B D | 3 |
| D B | 2 |
| C A | 1 |
| A E | 1 |
| E B | 1 |

## 3. USING EQUALITY MATCHING FOR INTRUSION DETECTION

The algorithm we use for detecting intrusions is simple but effective. With an approach similar to the that of [3], we employ equality matching of sequences of system calls captured during online usage against those stored in the database built from the normal program behavior profile. If the sequence of BSM events captured during online usage is not found in the database, then an anomaly counter is incremented. This technique is predicated on the ability to capture the normal behavior of a program in a database. If the normal behavior of a program is not adequately captured, then the false alarm rate is likely to be high. On the other hand, if the normal behavior profile built for a program includes intrusive behavior, then future instances of the intrusive behavior are likely to go undetected. Finally, it is important that the range of possible behavior patterns is much larger than the range of normal behavior patterns, because detection of intrusive events is only made possible when the intrusive behavior patterns are different from the normal behavior patterns. The more compact the representation of normal behavior and the larger the range of possible behavior, the better detection capacity this equality matching technique will have. The ratio of normal behavior patterns to possible behavior patterns provides an indication of the detection capacity of the technique. The smaller this ratio, the better the detection capacity.

Capturing system calls via the BSM provides a broad range of possible behavior. There are approximately 200 different BSM events that can be recorded. The $N$-grams described in Section 2. represent a sequence of $N$ consecutive BSM events. Therefore a single $N$-gram can have $200^N$ possi-

*Table 2.3* Number of unique BSM events for programs run in a sample `telnet` session.

| *Program* | *# of Unique BSM Events* |
|---|---|
| `cat` | 7 |
| `in.telnetd` | 14 |
| `login` | 17 |
| `mail` | 13 |
| `pt_chmod` | 10 |
| `quota` | 7 |
| `sendmail` | 18 |
| `sh` | 10 |
| `tcsh` | 13 |

ble combinations. In practice, however, a program will normally make only about 10 to 20 different system calls. Table 2.3 shows the number of different BSM events for programs in a sample `telnet` session under non-intrusive conditions.

Using 20 BSM events as an example, this gives rise to $20^N$ possible $N$-grams during normal operation. The ratio between possible BSM events during normal behavior to all possible system calls is $1/10^N$. Thus, as $N$ increases, the ratio of normal BSM events to possible BSM events decreases as a power of 10. In practice, the number of $N$-grams produced by a program is much smaller than $20^N$, mainly because either certain orderings never occur *e.g.*, {exit execve}, or because certain orderings do not correspond to what a particular program would ever do. Our results find that for $N = 6$, each program makes approximately 100 to 200 different $N$-grams, rather than $20^6$. Thus, the ratio of normal sequences of BSM events to possible sequences of BSM events is much smaller in practice than our pessimistic scenario.

The approach employed here exploits temporal locality of anomalous sequences for detecting intrusions. That is, rather than averaging the number of anomalous sequences out of the total number of sequences, we employ thresholding functions in fixed-size intervals, where the fixed-size interval is much smaller than the total execution trace. A session is considered to be intrusive if it contains an anomalous execution of a program. A program is considered anomalous if a portion of its windows is anomalous. A window is considered

*Table 2.4* Using intervals of fixed sizes and thresholds to exploit temporal locality of attacks.

| *Interval* | *Unit Contained* | *Threshold* |
|---|---|---|
| session | file | $ST$ |
| file | window | $FT$ |
| window | $N$-gram | $WT$ |
| $N$-gram | BSM event | — |

anomalous if a portion of its $N$-grams is anomalous. An $N$-gram is considered anomalous if it cannot be found in the database corresponding to the program currently being checked. Reliance on temporal locality is necessary due to *anomalous noise*. Anomalous noise is the behavior that might not appear to be normal, but is not subversive. Such behavior should be expected throughout a program execution. Truly subversive activity is normally accompanied by a much higher level of abnormal activity than one should expect from anomalous noise. Although anomalous noise might result in more abnormal behavior during a program execution than some subversive activity, within some specific degree of locality, the subversive behavior should produce much more anomalous behavior. Three example scenarios illustrate this point in Figure 2.1. [2]

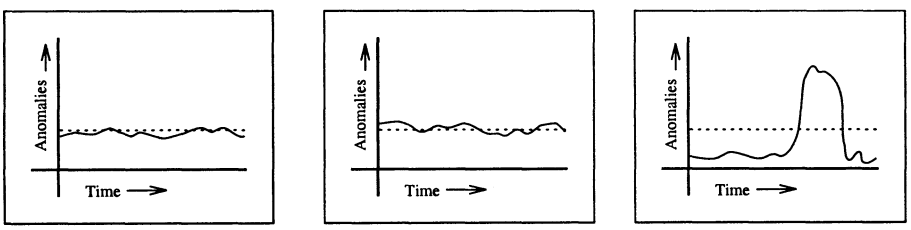



*Figure 2.1* Possible usages of a program in terms of anomalous behavior.

Table 2.4 shows the fixed-size intervals, the units they contain, and their thresholds that are used to detect anomalous behavior. When the threshold for a fixed interval — such as a window of $N$-grams — is exceeded, the whole

[2] If some threshold $T$ (displayed as the dotted line) is applied to the entire execution of a program, false positives and false negatives may occur frequently. A) During a normal execution, the anomalous noise is, on average, less than $T$. B) Due to normal statistical variance, the anomalous noise will, on average, slightly exceed $T$, resulting in a false positive. C) A knowledgeable intruder could minimize the anomalous noise of a program (by performing very mundane actions) so that, on average, the subversive behavior does not exceed $T$, resulting in a false negative.

interval is marked as anomalous. Then a thresholding function is applied at the next level up in abstraction using the marked anomalous interval to determine if an intrusion has occurred. For instance, if a window is marked as anomalous because its threshold, $WT$, for anomalous $N$-grams is exceeded, then this window is used as part of the count for file intervals to determine if the file threshold $FT$ is exceeded. Ultimately, the session threshold, $ST$, must be exceeded for the session to be labeled as an intrusion. The thresholds are all configurable and the tuning of the thresholds will impact the performance of the system as shown in the following sections.

## 3.1 TUNING PARAMETERS OF THE SYSTEM

The performance of the system is highly dependent on several independent parameters. Adjusting any one of several parameters can result a large change in the performance of the system, and there is some trade-off involved with each parameter. The parameters are:

- **Extent of Training** The amount of data used for training is constrained both by processing ability (as training the system is computationally very expensive), and the amount of training data available. The ideal amount of training results in a set of $N$-grams being collected which characterizes the behavior of a program under normal operation. Too little training results in very high levels of anomalous noise; too much training results in a decreased ability to recognize truly anomalous events as anomalous.

- **$N$-gram Size** When $N$ is very small, less temporal information is available. In the case where $N$ is one, all temporal information is lost. A very large value for $N$ results in $N$-grams that carry too much information. Each $N$-gram should be a general sequence of events that does not necessarily reflect the context in which it occurs. As $N$-grams carry more contextual information, anomalous noise rises.

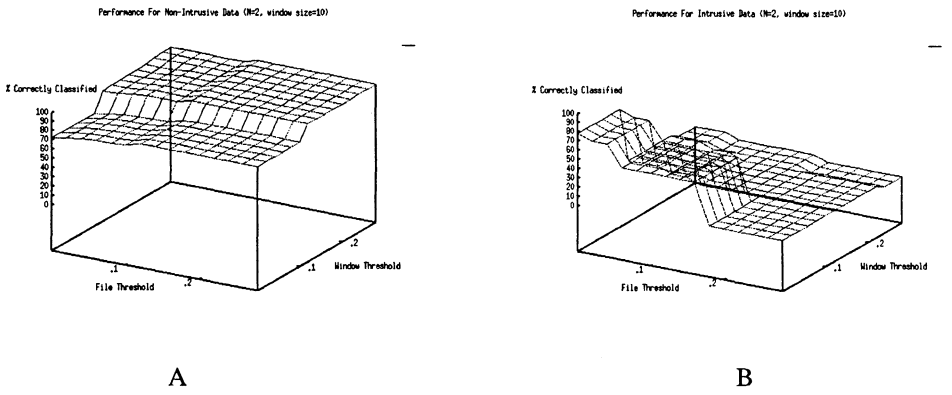
- **Window Size** The window size is the number of $N$-grams in a window. A window is meant to contain a relatively high-level user event. If the window is too small, it might not be able to contain an entire event, and short bursts of anomalous $N$-grams, which should be averaged into the anomalous noise, extend through the entire window, resulting in the window being falsely classified as anomalous. If the window size is very large, it might encompass many $N$-grams not related to a subversive action occurring during the window, and thus the subversive action might blend into the anomalous noise.

- **Window Threshold** This threshold is the value that the ratio of anomalous $N$-grams in a window to total $N$-grams in a window must exceed before the window is flagged as anomalous.

- **File Threshold** This threshold is the value that the ratio of anomalous windows in the execution of a program to total windows in a program must exceed before the execution of the program is considered to be anomalous.

- **Session Threshold** This threshold is the value above which the score for a session must exceed to be flagged as intrusive. This score corresponds to the anomalous measure of the most anomalous file.

The extent of training is typically limited by the amount of data and computational resources on hand.Surprisingly, even with only two weeks of training data, the equality matching system was able to detect intrusions with fairly high accuracy.



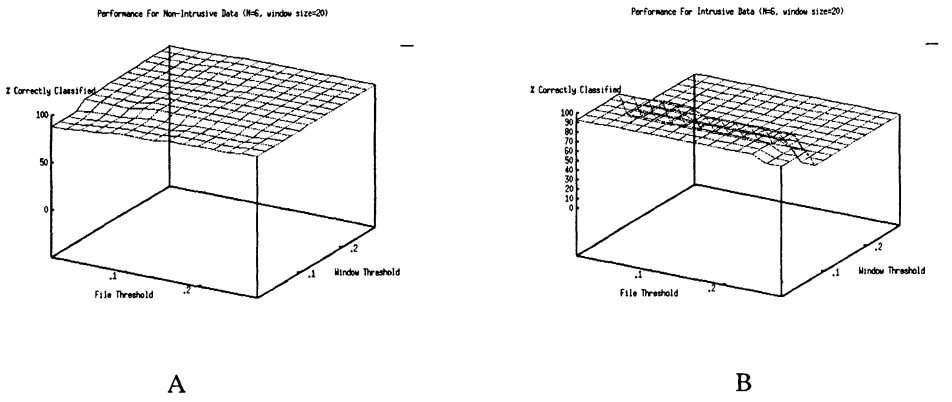

A B

*Figure 2.2* Performance of equality matching intrusion detection system (window size of 10, $N$-gram size of 2).

Varying the $N$-gram size and the window size will impact the performance of the system. Various $N$-gram sizes tested ranged from two to 50. The anomaly detection capabilities drop sharply for values of $N$ larger than 12. Therefore, we focused on smaller values for $N$. The smallest window size tested consisted of 10 $N$-grams.

To optimize the file and window thresholds, the performance of the system was measured against these two variables while fixing the $N$-gram size and the window size. Figure 2.2A shows the performance of the system with respect to

non-intrusive, normal data. [3] As expected, as we increase the window and file thresholds, the performance of the system in classifying non-intrusive data improves. In other words, increasing these thresholds increases the tolerance for anomalous noise in a session. Figure 2.2B shows that increasing these thresholds for intrusion data decreases the performance of the system. That is, as the tolerance for anomalous noise is increased, the rate of correct classifications will decrease due to missed intrusions (false negatives). Notice in both these plots that the (0.1, 0.1) threshold appears to be a turning point where performance sharply changes. This is the optimum threshold point given the fixed parameters. Plots such as these are used to determine optimal thresholds for tuning the equality matching intrusion system.




A B

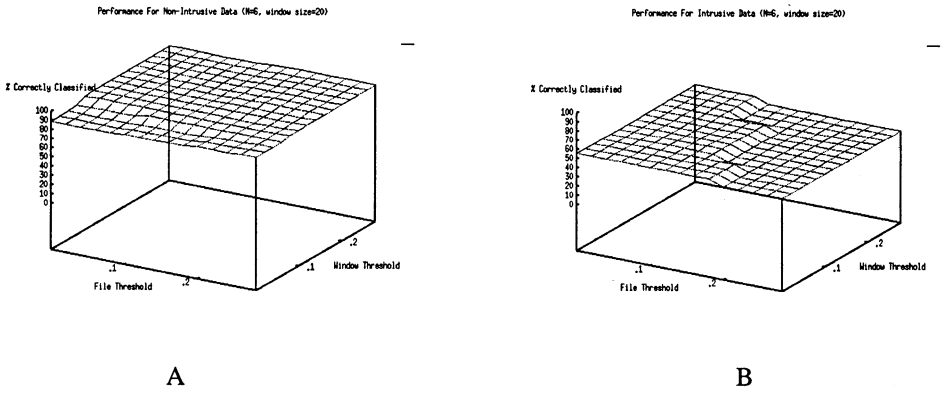
*Figure 2.3* Performance of equality matching intrusion detection system (optimal window size of 20, optimal $N$-gram size of 6).

To determine optimal $N$-gram size and optimal window size, plots similar to Figure 2.2 were constructed except one parameter was fixed, while the other was varied. For instance, to determine optimal $N$-gram size, the window size was fixed at 20, and numerous plots similar to Figure 2.2 were constructed for values of $N$ ranging from two to 12 in increments of two. Using this process for both $N$-gram sizes and window sizes, the optimal performance was found for $N = 6$ and window size of 20. Figure 2.3 shows the performance of the equality matching system with these optimal parameters under varying file and window thresholds. Figure 2.3A shows the large plateau in the center of the plane, which indicates that there is a large range of window and file

[3] These graphs display the percentage of data that is correctly classified when the data is A) non-intrusive, and B) intrusive. As the file and window threshold ratios rise, the system becomes more tolerant of anomalous behavior. Thus, lower thresholds result in an increased ability to detect intrusions, but also a greater likelihood of misclassification of non-intrusions.

thresholds which produce optimal performance. [4] Figure 2.3B also shows a well-defined and broad region of good performance for intrusion data. Having a large and well-defined range of performance for these thresholds is important because these thresholds may be varied for different programs and in different environments.

In order to insure that the optimal parameters found were generally applicable, and not simply specific to the data being used, this process was repeated for different training sets and intrusive and non-intrusive sessions. The results from performing this analysis on other data sets showed consistently good performance in correctly classifying non-intrusive data, but varying performance in classifying intrusion data. This result is expected because the ratio of intrusive data to non-intrusive data in the test sets is very small. Thus, one would expect the performance of the system to vary more widely when presented the rare intrusion data. Nonetheless, the results showed that the $N$-gram size of six and window size of 20 were in fact optimal for the different data sets. Results from testing the system on new data sets are shown in Figure 2.4. [5]




A B

*Figure 2.4* Performance of the intrusion detection system on new data (window size of 20, $N$-gram size of 6).

[4]These graphs display the percentage of data that is correctly classified when the data is A) non-intrusive, and B) intrusive. The large plateau in the center of these plots indicates good performance over a large range of file and window thresholds.

[5]The variation in performance in intrusive data results from the fact that testing is performed on much more normal data than intrusive data. Thus, normal statistical variation have a larger effect on the intrusive data.

# 4. COMPUTING THE PROBABILITY OF DETECTION VERSUS THE PROBABILITY OF FALSE ALARM

A measure of the overall effectiveness of a given intrusion detection system can be provided by the receiver operating characteristic (ROC) curve. A ROC curve is a parametric curve that is generated by varying the threshold of the *intrusive measure*, which is a tunable parameter, and computing the probability of detection and the probability of false alarm at each operating point. The curve is a plot of the likelihood that an intrusion is detected, against the likelihood that a non-intrusion is misclassified for a particular parameter, such as a threshold. The ROC curve can be used to determine the performance of the system for any possible operating point. Thus, it can be applied to different file and window thresholds to judge performance. ROC curves were generated for our system for the labeled training data and are illustrated in Figure 2.5. [6] The results are for data that the system had not seen before, *i.e.*, data on which it had not trained.

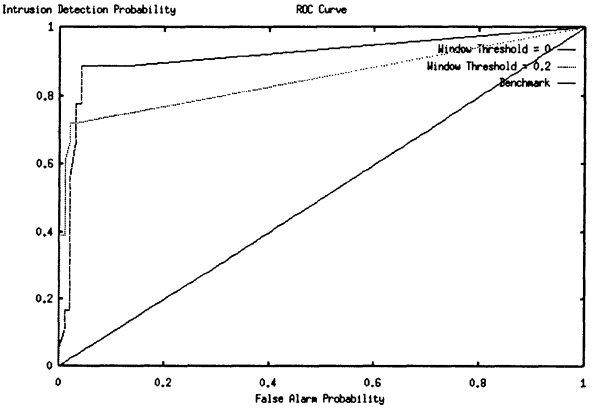



*Figure 2.5* Receiver operating characteristic (ROC) curve for optimal parameters after 2 weeks of training data.

To better understand this performance measure, consider an intrusion detection oracle that scores a session with a value of one if and only if it is an intrusion, and a value of zero otherwise. The resulting ROC curve would actually not be a curve, but rather, a single point at the location (0,1) since it is would detect intrusions with a likelihood of 1/1, and it would misclassify non-

[6]The ROC curve allows an intrusion detection system user to adjust the sensitivity of the intrusion detection system. This graph shows both the worst possible ROC curve (*i.e.*, $y = x$), as well as a ROC curve generated from actual data.

intrusions with a likelihood of 0/1. Further, as the threshold varied between zero and one (exclusive), there would be no change in the way sessions are classified, so the parametric value would remain at that one point. This can be called the *oracle point.* However, at the thresholds of 1 and 0 (inclusive), the (0,0) and (1,1) points remain fixed. Connecting these points and computing the area under the curve gives an area of 1, or a power of 100%.

At the other end of the spectrum, consider the curve that defines the worst possible intrusion detection system. The ROC curve for the worst case scenario is the $y = x$ line shown in Figure 2.5. Assume a system that randomly assigns a value between zero and one for every session. Starting from a threshold of zero, we derive the (1,1) point because all sessions would be classified as intrusions. As the session threshold increases, the likelihood of both correctly classifying an intrusion and incorrectly classifying a non-intrusion decrease at the same rate until the session threshold is 1 (corresponding to the point (0,0)). The power of this system is 50%, corresponding to the area under this curve of 0.5. If an intrusion detection system were to perform even worse than this curve, one would simply invert each classification to do better. Therefore, the $y = x$ plot represents the benchmark by which all intrusion detection system should do better.

The ROC curve allows the end user of an intrusion detection system to assess the trade-off between detection ability and false alarm rate in order to properly tune the system for acceptable tolerances. For example, Figure 2.5 shows ROC curves produced with a window size of 20 and an $N$-gram size of six. These parameters were found to be optimal from our previous analyses. In the ROC curves plotted in Figure 2.5, the window threshold was held constant for each curve, while the *session threshold* was varied. The session threshold is the value above which the score for a session must exceed to be flagged as possibly intrusive. Recall that the score reported for a session is the measure of the *most* anomalous file in that session.

Three ROC curves are shown in Figure 2.5. The $y = x$ curve is shown as the benchmark for the worst case scenario. The top curve corresponds to a window threshold fixed at zero, and the other corresponds to a window threshold fixed at 0.2. The ROC curve for the window threshold of 0.2 encloses an area of 0.85. The ROC curve for the window threshold of 0 encloses an area of 0.91. No value for the window threshold resulted in an enclosed area greater than 0.91. Thus, a window threshold of 0 is optimal.

## 5. CONCLUSION

The analyses presented here are for perhaps the simplest intrusion detection algorithm — equality matching — for detecting intrusions by profiling pro-

gram behavior. The real constraints of this approach are the computing and storage capacity necessary to process data, and the necessity for good training data in the first place. However, we were able to produce results that indicated that the simple equality matching technique for program behavior profiling provides sufficiently good performance for intrusion detection. A slightly more "intelligent" algorithm — one that could recognize $N$-grams *similar* to those it had previously encountered — might result in fewer false positives, thereby providing a reliable intrusion detection mechanism.

## Acknowledgment

We are pleased to acknowledge Richard Lippmann and Marc Zissman of MIT's Lincoln Laboratory for providing us the training and testing data, as well as introducing us to the ROC assessment approach.

## References

[1] Cohen, W. (1995). Fast effective rule induction. *Machine Learning: Proceedings of the Twelfth International Conference*, Morgan Kaufmann.

[2] D'haeseleer, P., Forrest, S., and Helman, P. (1996). An immunological approach to change detection: Algorithms, analysis and implications. *Proceedings of the IEEE Symposium on Security and Privacy.*

[3] Forrest, S., Hofmeyr, S., and Somayaji, A. (1997). Computer immunology. *Communications of the ACM*, **40(10)**, pp. 88–96.

[4] Forrest, S., Hofmeyr, S., Somayaji, A., and Longstaff, T. (1996). A sense of self for unix processes. *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 120–128.

[5] Garvey, T. and Lunt, T. (1991). Model-based intrusion detection. *Proceedings of the Fourteenth National Computer Security Conference.*

[6] Ilgun, K. (1992). Ustat: A real-time intrusion detection system for unix. Master's thesis, Computer Science Dept, UCSB.

[7] Kumar, S. and Spafford, E. (1996). A pattern matching model for misuse intrusion detection. The COAST Project, Purdue University.

[8] Lee, W., Stolfo, S., and Chan, P. (1997). Learning patterns from unix process execution traces for intrusion detection. *Proceedings of AAAI97 Workshop on AI Methods in Fraud and Risk Management.*

[9] Lunt, T. (1990). Ides: an intelligent system for detecting intruders. *Proceedings of the Symposium: Computer Security, Threat and Countermeasures.*

[10] Lunt, T. (1993). A survey of intrusion detection techniques. *Computers and Security*, **12**, pp. 405–418.

[11] Lunt, T. and Jagannathan, R. (1988). A prototype real-time intrusion-detection system. *Proceedings of the IEEE Symposium on Security and Privacy.*

[12] Lunt, T., Tamaru, A., Gilham, F., Jagannthan, R., Jalali, C., Javitz, H., Valdos, A., Neumann, P., and Garvey, T. (1992). A real-time intrusion-detection expert system (ides). Technical Report, Computer Science Laboratory, SRI Internationnal.

[13] Monrose, F. and Rubin, A. (1997). Authentication via keystroke dynamics. *Proceedings of the Fourth ACM Conference on Computer and Communications Security.*

[14] Porras, P. and Kemmerer, R. (1992). Penetration state transition analysis - a rule-based intrusion detection approach. *Proceedings of the Eighth Annual Computer Security Applications Conference*, pp. 220–229.

[15] Voas, J., Payne, J., and Cohen, F. (1992). A model for detecting the existence of software corruption in real time. *Computers and Security Journal*, **11(8)**, pp. 275–283.