An Introduction to Intrusion Detection

by Aurobindo Sundaram

Introduction

In the last three years, the networking revolution has finally come of age. More than ever before, we see that the Internet is changing computing as we know it. The possibilities and opportunities are limitless; unfortunately, so too are the risks and chances of malicious intrusions.

It is very important that the security mechanisms of a system are designed so as to *prevent* unauthorized access to system resources and data. However, completely preventing breaches of security appear, at present, unrealistic. We can, however, try to detect these intrusion attempts so that action may be taken to repair the damage later. This field of research is called **Intrusion Detection**.

Anderson, while introducing the concept of intrusion detection in 1980 [1], defined an **intrusion attempt** or a **threat** to be the potential possibility of a deliberate unauthorized attempt to

- access information,
- manipulate information, or
- render a system unreliable or unusable.

Since then, several techniques for detecting intrusions have been studied. This paper discusses why intrusion detection systems are needed, the main techniques, present research in the field, and possible future directions of research.

The need for Intrusion Detection Systems

A computer system should provide *confidentiality*, *integrity* and *assurance* against denial of service. However, due to increased connectivity (especially on the Internet), and the vast spectrum of financial possibilities that are opening up, more and more systems are subject to attack by intruders. These subversion attempts try to exploit flaws in the operating system as well as in application programs and have resulted in spectacular incidents like the Internet Worm incident of 1988 [12].

There are two ways to handle subversion attempts. One way is to prevent subversion itself by building a completely secure system. We could, for example, *require* all users to identify and authenticate themselves; we could protect data by various cryptographic methods and very tight access control mechanisms. However this is not really feasible because:

1. In practice, it is not possible to build a completely secure system. Miller [10] gives a compelling report on bugs in popular programs and operating systems that seems to indicate that (a) bug free software is still a dream and (b) no-one seems to want to make the effort to try to develop such software. Apart from the fact that we do not seem to be getting our money's worth when we buy software, there are also security implications when our E-mail software, for example, can be attacked. Designing and implementing a totally secure system is thus an extremely difficult task.
2. The vast installed base of systems worldwide guarantees that any transition to a secure system, (if

it is ever developed) will be long in coming.

3. Cryptographic methods have their own problems. Passwords can be cracked, users can lose their passwords, and entire crypto-systems can be broken.
4. Even a truly secure system is vulnerable to abuse by insiders who abuse their privileges.
5. It has been seen that that the relationship between the level of access control and user efficiency is an inverse one, which means that the stricter the mechanisms, the lower the efficiency becomes.

We thus see that we are stuck with systems that have vulnerabilities for a while to come. If there are attacks on a system, we would like to detect them as soon as possible (preferably in real-time) and take appropriate action. This is essentially what an Intrusion Detection System (IDS) does. An IDS does not usually take preventive measures when an attack is detected; it is a reactive rather than pro-active agent. It plays the role of an informant rather than a police officer.

The most popular way to detect intrusions has been by using the *audit data* generated by the operating system. An *audit trail* is a record of activities on a system that are logged to a file in chronologically sorted order. Since almost all activities are logged on a system, it is possible that a manual inspection of these logs would allow intrusions to be detected. However, the incredibly large sizes of audit data generated (on the order of 100 Megabytes a day) make manual analysis impossible. IDSs automate the drudgery of wading through the audit data jungle. Audit trails are particularly useful because they can be used to establish guilt of attackers, and they are often the only way to detect unauthorized but subversive user activity.

Many times, even after an attack has occurred, it is important to analyze the audit data so that the extent of damage can be determined, the tracking down of the attackers is facilitated, and steps may be taken to prevent such attacks in future. An IDS can also be used to analyze audit data for such insights. This makes IDSs valuable as real-time as well as post-mortem analysis tools.

Spafford [13] reports:

- Information theft is up over 250% in the last 5 years.
- 99% of all major companies report at least one major incident.
- Telecom and computer fraud totaled $10 billion in the US alone.

It is thus more important than ever before that since it seems obvious that we cannot prevent subversion, we should at least try to detect it and prevent similar attacks in future.

In the following sections, we use definitions from the pioneering work in intrusion detection[1]

- **Risk** : Accidental or unpredictable exposure of information, or violation of operations integrity due to the malfunction of hardware or incomplete or incorrect software design.
- **Vulnerability** : A known or suspected flaw in the hardware or software or operation of a system that exposes the system to penetration or its information to accidental disclosure.
- **Attack** : A specific formulation or execution of a plan to carry out a threat.
- **Penetration** : A successful attack -- the ability to obtain unauthorized (undetected) access to files and programs or the control state of a computer system.

Anderson also classified intruders into two types, the *external* intruders who are unauthorized users of the machines they attack, and *internal* intruders, who have permission to access the system, but not some portions of it. He further divided internal intruders into intruders who *masquerade* as another user, those with *legitimate* access to sensitive data, and the most dangerous type, the *clandestine* intruders who have the power to turn off audit control for themselves.

Classification of Intrusion Detection Systems

Intrusions can be divided into 6 main types [11]

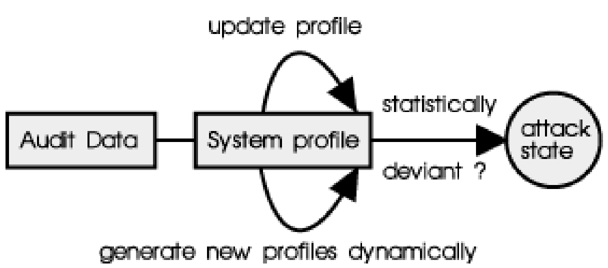
1. Attempted break-ins, which are detected by atypical behavior profiles or violations of security constraints.
2. Masquerade attacks, which are detected by atypical behavior profiles or violations of security constraints.
3. Penetration of the security control system, which are detected by monitoring for specific patterns of activity.
4. Leakage, which is detected by atypical use of system resources.
5. Denial of service, which is detected by atypical use of system resources.
6. Malicious use, which is detected by atypical behavior profiles, violations of security constraints, or use of special privileges.

However, we can divide the techniques of intrusion detection into two main types.

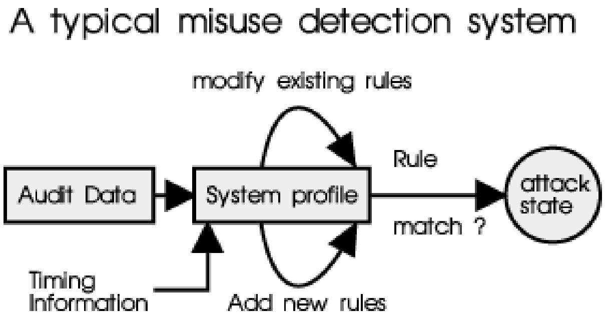
Anomaly Detection : Anomaly detection techniques assume that all intrusive activities are necessarily anomalous. This means that if we could establish a "normal activity profile" for a system, we could, in theory, flag all system states varying from the established profile by statistically significant amounts as intrusion attempts. However, if we consider that the set of intrusive activities only intersects the set of anomalous activities instead of being exactly the same, we find a couple of interesting possibilities: (1) Anomalous activities that are not intrusive are flagged as intrusive. (2) Intrusive activities that are not anomalous result in false negatives (events are not flagged intrusive, though they actually are). This is a dangerous problem, and is far more serious than the problem of false positives.

The main issues in anomaly detection systems thus become the selection of threshold levels so that neither of the above 2 problems is unreasonably magnified, and the selection of features to monitor. Anomaly detection systems are also computationally expensive because of the overhead of keeping track of, and possibly updating several system profile metrics. Some systems based on this technique are discussed in Section 4 while a block diagram of a typical anomaly detection system is shown in Figure 1.

A typical anomaly detection system



Misuse Detection: The concept behind misuse detection schemes is that there are ways to represent attacks in the form of a pattern or a signature so that even variations of the same attack can be detected. This means that these systems are not unlike virus detection systems -- they can detect many or all *known* attack patterns, but they are of little use for as yet unknown attack methods. An interesting point to note is that anomaly detection systems try to detect the complement of "bad" behavior. Misuse detection systems try to recognize known "bad" behavior. The main issues in misuse detection systems are how to write a signature that encompasses *all* possible variations of the pertinent attack, and how to write signatures that do not also match non-intrusive activity. Several methods of misuse detection, including a new pattern matching model are discussed later. A block diagram of a typical misuse detection system is shown in Figure 2 below.



Anomaly Detection Systems

There have been a few major approaches to anomaly intrusion detection systems, some of which are described below.

Statistical approaches: In this method, initially, behavior profiles for subjects are generated. As the system continues running, the anomaly detector constantly generates the variance of the present profile from the original one. We note that, in this case, there may be several measures that affect the behavior profile, like activity measures, CPU time used, number of network connections in a time period, etc. In some systems, the current profile and the previous profile are merged at intervals, but in some other systems profile generation is a one time activity. The main advantage to statistical systems is that they adaptively learn the behavior of users; they are thus potentially more sensitivte than human experts. However there are a few problems with statistical approaches: they can gradually be trained by intruders so that eventually, intrusive events are considered normal, false positives and false negatives are generated depending on whether the threshold is set too low or too high, and relationships between events are missed because of the insensitivity of statistical measures to the order of events.

An open issue with statistical approaches in particular, and anomaly detection systems in general, is the selection of measures to monitor. It is not known exactly what the subset of all possible measures that accurately predicts intrusive activities is. Static methods of determining these measures are sometimes misleading because of the unique features of a particular system. Thus, it seems that a combination of static and dynamic determination of the set of measures should be done. Some problems associated with this technique have been remedied by other methods, including the method involving *Predictive Pattern Generation*, which takes past events into account when analyzing the data.

Predictive pattern generation: This method of intrusion detection tries to predict future events based on the events that have already occurred [14]. Therefore, we could have a rule

```
E1 - E2 --> (E3 = 80%, E4 = 15%, E5 = 5%)
```

This would mean that given that events E1 and E2 have occurred, with E2 occurring after E1, there is an 80% probability that event E3 will follow, a 15% chance that event E4 will follow and a 5% probability that event E5 will follow. The problem with this is that some intrusion scenarios that are not described by the rules will not be flagged intrusive. Thus, if an event sequence A - B - C exists that is intrusive, but not listed in the rulebase, it will be classified as unrecognized. This problem can be partially solved by flagging any unknown events as intrusions (increasing the probability of false positives), or by flagging them as non-intrusive (thus increasing the probability of false negatives). In the normal case, however, an event is flagged intrusive if the left hand side of a rule is matched, but the right hand side is statistically very deviant from the prediction.

There are several advantages to this approach. First, rule based sequential patterns can detect anomalous activities that were difficult with traditional methods. Second, systems built using this model are highly adaptive to changes. This is because low quality patterns are continuously eliminated, finally leaving the higher quality patterns behind. Third, it is easier to detect users who try to train the system during its learning period. And fourth, anomalous activities can be detected and reported within seconds of receiving audit events.

Another approach taken in intrusion detection systems is the use of **neural networks**. The idea here is to train the neural network to predict a user’s next action or command, given the window of n previous actions or commands. The network is trained on a set of representative user commands. After the training period, the network tries to match actual commands with the actual user profile already present in the net. Any incorrectly predicted events (events and commands are used interchangeably in this discussion) actually measure the deviation of the user from the established profile. Some advantages of using neural networks are: [8] they cope well with noisy data, their success does not depend on any statistical assumption about the nature of the underlying data, and they are easier to modify for new user communities. However, they have some problems. First, a small window will result in false positives while a large window will result in irrelevant data as well as increase the chance of false negatives. Second, the net topology is only determined after considerable trial and error. And third, the intruder can train the net during its learning phase.

Misuse Detection Systems

There has been significant research in misuse detection systems in the recent past, including attempts at SRI, Purdue University and the University of California-Davis. Some of these systems are explained in depth in this section.

Expert systems are modeled in such a way as to separate the rule matching phase from the action phase. The matching is done according to audit trail events. The Next Generation Intrusion Detection Expert System (NIDES) developed by SRI is an interesting case study for the expert system approach. NIDES follows a hybrid intrusion detection technique consisting of a misuse detection component as well as an anomaly detection component. The anomaly detector is based on the statistical approach, and it flags events as intrusive if they are largely deviant from the expected behavior. To do this, it builds user profiles based on many different criteria (more than 30 criteria, including CPU and I/O usage, commands used, local network activity, system errors etc.) [8]. These profiles are updated at periodic intervals. The expert system misuse detection component encodes known intrusion scenarios and attack patterns (bugs in old versions of sendmail could be one vulnerability). The rule database can be changed for different systems. One advantage of the NIDES approach is that it has a statistical component as well as an expert system component. This means that the chances of one system catching intrusions missed by the other increase. Another advantage is the problem's control reasoning is cleanly separated from the formulation of the solution.

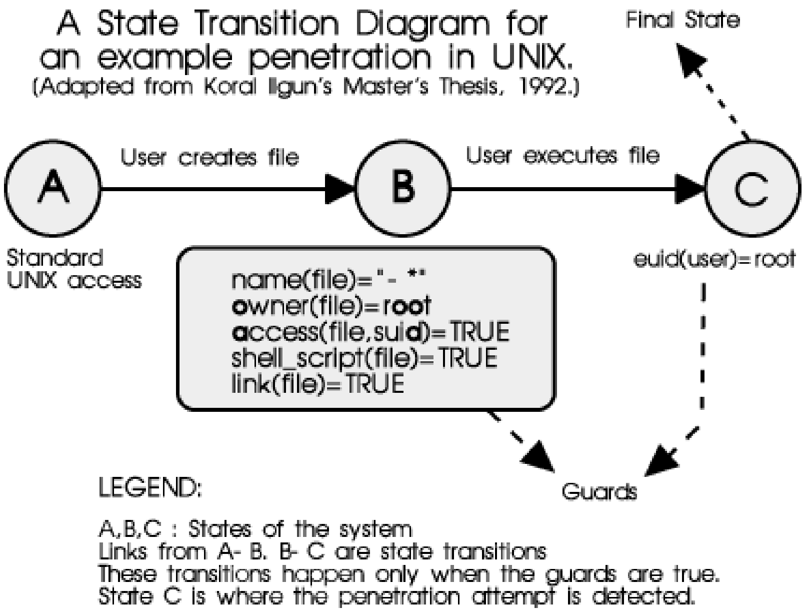
There are some draw backs to the expert system approach too. For example, the expert system has to be formulated by a security professional and thus the system is only as strong as the security personnel who programs it [7]. This means that there is a real chance that expert systems can fail to flag intrusions. It is for this reason that NIDES has an anomaly as well as a misuse detection component. These two components are loosely coupled in the sense that they perform their operations independently for the most part. The NIDES system runs on a machine different from the machine(s) to be monitored, which could be unreasonable overhead. Furthermore, additions and deletions of rules from the rule-base must take into account the inter-dependencies between different rules in the rule-base. And there is no recognition of the sequential ordering of data, because the various conditions that make up a rule are not recognized to be ordered.

Keystroke monitoring is a very simple technique that monitors keystrokes for attack patterns. Unfortunately the system has several defects -- features of shells like *bash*, *ksh*, and *tcsh* in which user definable aliases are present defeat the technique unless alias expansion and semantic analysis of the commands is taken up. The method also does not analyze the running of a program, only the keystrokes. This means that a malicious program cannot be flagged for intrusive activities. Operating systems do not offer much support for keystroke capturing, so the keystroke monitor should have a hook that analyses keystrokes before sending them on to their intended receiver. An improvement to this would be to monitor system calls by application programs as well, so that an analysis of the program's execution is possible.

Model Based Intrusion Detection states that certain scenarios are inferred by certain other observable activities. If these activities are monitored, it is possible to find intrusion attempts by looking at activities that infer a certain intrusion scenario. The model based scheme consists of three important modules[4]. The *anticipator* uses the active models and the scenario models to try to predict the next step in the scenario that is expected to occur. A scenario model is a knowledge base with specifications of intrusion scenarios. The *planner* then translates this hypothesis into a format that shows the behavior as it would occur in the audit trail. It uses the predicted information to plan what to search for next. The *interpreter* then searches for this data in the audit trail. The system proceeds this way, accumulating more and more evidence for an intrusion attempt until a threshold is crossed; at this point, it signals an intrusion attempt.

This is a very clean approach. Because the planner and the interpreter know what they are searching for at each step, the large amounts of noise present in audit data can be filtered, leading to excellent performance improvements. In addition, the system can predict the attacker's next move based on the intrusion model. These predictions can be used to verify an intrusion hypothesis, to take preventive measures, or to determine what data to look for next.

However, there are some critical issues related to this system. First, patterns for intrusion scenarios must be easily recognized. Second, patterns must always occur in the behavior being looked for. And finally, patterns must be *distinguishing*; they must *not* be associated with any other normal behavior.



In the **State Transition Analysis** technique, the monitored system is represented as a state transition diagram. As data is analyzed, the system makes transitions from one state to another. A transition takes place on some Boolean condition being true (for example, the user opening a file). The approach followed in USTAT [5] is to have state transitions from safe to unsafe states based on known attack patterns. To make this model clearer, we illustrate with an example based almost entirely on an example in Ilgun's thesis.

1. The attacker creates a link starting with "-" (say -x) to root's setuid shell script containing the *#!/bin/sh* mechanism.
2. The attacker executes -x.

The point of this attack is that whenever a hard link to a file is created, a new inode with the target's original permissions is created. Since invoking a script with the *#!/bin/sh* mechanism ianvokes a subshell, and further, if the name of the subshell begins with a dash an interactive shell is created, we see that the attacker has obtained an interactive shell with root privileges. The state diagram for this is shown in Figure 3. We see that for the final *compromised* state to be reached, some conditions have to be fulfilled. If these *guard* conditions are true, then there is almost certainly an intrusion attempt going on. However, if any of these conditions do not hold, the probability of an intrusive action is considerably decreased. We see that the guard conditions exist to filter the intrusive activities from the non-intrusive ones. Hence, this can serve as a data pruning mechanism as observed in the model based scheme above. Some advantages of this approach are: it can detect co-operative attacks, it can detect attacks that span across multiple user sessions, and it can foresee impending compromise situations based on the present system state and take pre-emptive measures.

However there are also a few problems with state transition systems. First, attack patterns can specify only a sequence of events, rather than more complex forms. Second, there are no general purpose methods to prune the search except through the assertion primitives described above. And finally, they cannot detect denial of service attacks, failed logins, variations from normal usage, and passive listening -- this is because these items are either not recorded by the audit trail mechanism, or they cannot be represented by state transition diagrams.

A small point to be noted is that USTAT was never meant to be a stand-alone intrusion detection system; indeed, it is meant to be used with an anomaly detector so that more intrusion attempts may be detected by their combination. Some of the weaknesses of state transition systems are remedied by the *Pattern Matching Model*, discussed next.

Kumar [6] proposed a new misuse detection system based on **Pattern Matching**. This model encodes known intrusion signatures as patterns that are then matched against the audit data. Like the state transition analysis model, this model attempts to match incoming events to the patterns representing intrusion scenarios. The implementation makes transitions on certain events, called *labels*, and Boolean variables called *guards* can be placed at each transition. The difference between this and the state transition model is that the state transition model associates these guards with states, rather than transitions. The important advantages of this model are:

1. Declarative Specification : It only needs to be specified what patterns need to be matched, not how to match them.
2. Multiple event streams can be used together to match against patterns for each stream without the need to combine streams. This means that streams can be processed independently, and their results can be analyzed together to give evidence of intrusive activity.
3. Portability : Since intrusion signatures are written in a system independent script, they need not be rewritten for different audit trails. The patterns' declarative specifications enable them to be exchanged across different Operating Systems and different audit trails.
4. It has excellent real-time capabilities. Kumar reports a CPU overhead of 5-6% when scanning for 100 different patterns, which is excellent.
5. It can detect some attack signatures like the failed logins signature that the state transition model cannot do.

One problem with this model it it can only detect attacks based on known vulnerabilities (a problem with misuse detection systems in general) In addition, pattern matching is not very useful for representing ill-defined patterns and it is not an easy task to translate known attack scenarios into patterns that can be used by the model. Also, it cannot detect passive wire-tapping intrusions, nor can in detect spoofing attacks where a machine pretends to be another machine by using its IP address.

An interesting fact about Kumar's IDS is that it is called IDIOT (Intrusion Detection In Our Time), and we leave it to the reader to ponder the appropriateness of the name for the state of the art in intrusion detection.

6 Other Models and Directions in Research

Dorothy Denning [3] introduced a **Generic Intrusion Detection Model** that was independent of any particular system, application environment, system vulnerability, or type of intrusion. The basic idea of the model is to maintain a set of profiles for *subjects* (usually, but not necessarily users of a system). When an audit record is generated, the model matches it with the appropriate profile and then makes decisions on updating the profile, checking for abnormal behavior and reporting anomalies detected. To do this, it monitors system services such as file accesses, executable programs, and logins. It has no specific knowledge of the target system's vulnerabilities, although this knowledge would be extremely useful in making the model more valuable. In fact, the Intrusion Detection Expert System (IDES) developed at SRI was based on this model. The basic ideas in this model appear with little modification in many systems built. However, there are some systems that do not fit easily into this model.

NSM (**Network Security Monitor**) is an intrusion detection system developed at the University of California-Davis. NSM is a network-based IDS that differs from all of the IDSs discussed earlier because it does not use or analyze the host machine(s) audit trails. Rather, it monitors network traffic in order to detect intrusions [9]. Since network based attacks are expected to be prevalent in the future due to the mushrooming of the Internet, NSM could prove to be a valuable tool to detect intrusive activity.

NSM has several perceived advantages. First, the IDS gets instantaneous access to network data. Second, the IDS is hidden from the intruder because it is passively listening to network traffic. Therefore, it cannot be shut off or its data compromised. Finally, the IDS can be used with any system, because it is monitoring network traffic, protocols for which (TCP, UDP etc.) are standardized. There is no problem with different audit files, for example.

Researchers at Purdue University are working on several issues in intrusion detection. Crosbie and Spafford [2] propose to build an IDS using **Autonomous Agents**. Instead of a single large IDS defending the system, they propose an approach where several independent, small processes operate while co-operating in maintaining the system. The advantages claimed for this approach are efficiency, fault tolerance, resilience to degradation, extensibility and scalability. The foreseen drawbacks include the overhead of so many processes, long training times, and the fact that if the system is subverted, it becomes a security liability. An interesting possibility they open up is that of an *active* defense, that can respond to intrusions actively instead of passively reporting them (it could kill suspicious connections, for example).

Conclusion

Intrusion Detection is still a fledgling field of research. However, it is beginning to assume enormous importance in today's computing environment. The combination of facts such as the unbridled growth of the Internet, the vast financial possibilities opening up in electronic trade, and the lack of truly secure systems make it an important and pertinent field of research. Future research trends seem to be converging towards a model that is a hybrid of the anomaly and misuse detection models; it is slowly acknowledged that neither of the models can detect all intrusion attempts on their own. This approach has been successfully adopted in NIDES, and we can expect more such attempts in the future. Some schools doing research in this field include The COAST group at Purdue University, The University of California-Davis, and The University of California-Santa Barbara. The interested reader is encouraged to browse the provided links for more information.

References

1. J.P Anderson. Computer Security Threat Monitoring and Surveillance. Technical report, James P Anderson Co., Fort Washington, Pennsylvania, April 1980.
2. Mark Crosbie and Eugene Spafford. Defending a Computer System Using Autonomous Agents. Technical Report CSD-TR-95-022, Department of Computer Sciences, Purdue University, 1995.
3. Dorothy E Denning. An Intrusion Detection Model. In *IEEE Transactions on Software Engineering*, Number 2, page 222, February 1987.
4. T D Garvey and Teresa F Lunt. Model based intrusion detection. In *Proceedings of the 14th National Computer Security Conference*, pages 372-385, October 1991.

5. Koral Ilgun. USTAT - A Real-time Intrusion Detection System for UNIX. Master's Thesis, University of California at Santa Barbara, November 1992.
6. Sandeep Kumar. Classification and Detection of Computer Intrusions. Ph.D. Dissertation, August 1995.
7. Teresa F Lunt. Detecting Intruders in Computer Systems. Conference on Auditing and Computer Technology, 1993.
8. Teresa F Lunt. A survey of intrusion detection techniques. In *Computers and Security*, 12(1993), pages 405-418.
9. Biswanath Mukherjee, L Todd Heberlein and Karl N Levitt. Network Intrusion Detection, IEEE Network, May/June 1994, pages 26-41.
10. Barton P Miller, David Koski, Cjin Pheow Lee, Vivekananda Maganty, Ravi Murthy, Ajitkumar Natarajan, Jeff Steidl. Fuzz Revisited: A Re-examination of the Reliability of UNIX Utilities and Services. Computer Sciences Department, University of Wisconsin, 1995.
11. Steven E Smaha. Haystack: An Intrusion Detection System. In *Fourth Aerospace Computer Security Applications Conference*, pages 37-44, Tracor Applied Science Inc., Austin, Texas, December 1988.
12. Eugene H Spafford. The Internet Worm Program: An Analysis. In *ACM Computer Communication Review*; 19(1), pages 17-57, Jan 1989.
13. Eugene H Spafford. Security Seminar, Department of Computer Sciences, Purdue University, Jan 1996.
14. Henry S Teng, Kaihu Chen and Stephen C Lu. Security Audit Trail Analysis Using Inductively Generated Predictive Rules. In *Proceedings of the 11th National Conference on Artificial Intelligence Applications*, pages 24-29, IEEE, IEEE Service Center, Piscataway, NJ, March 1990.