Misuse Detection Research Study

Reader's Digest Condensed Version

RFP # 030-96FS

October 20, 1996

Principal Investigators: Karl Levitt, Matt Bishop

Department of Computer Science

University of California, Davis

Davis, California 95616

(916) 752-7004, (916) 752-4767 FAX

misuse@cs.ucdavis.edu

1. Introduction

The Computer Science Department of the University of California, Davis is pleased to submit this proposal in response to RFP No. 030-96FS. The overall goal of the proposed study is to develop a system that will detect misuse on the part of individuals with authority to access sensitive information on information processing systems. This problem, called the “insider” problem, has not been considered in depth as compared with the “outsider” problem, which is concerned with the detection of unauthorized penetration into a system. An effective intrusion detection system should address both the outsider and insider problem, in part because outsiders can often gain access to a system by masquerading as insiders; defenses against only outsiders would be useless for masquerade attacks. In addition, many systems do not provide the granularity of protection necessary to prevent insider attacks; for such systems the kind of misuse system we propose here is the only way to detect such attacks.

In addition, most intrusion detection systems have focused on Unix hosts or networks of Unix systems, and the data source for these systems is primarily system-level audit reports. This proposal addresses misuse detection for non-Unix systems, primarily Windows NT, and also emphasizes the need to generate audit logs other than for system-level activity. Hence, we will investigate techniques to generate audit logs from applications such as database systems, and approaches to auditing object-based systems (such as CORBA) as a general framework for auditing objects and applications.

We also propose to consider an organization’s policy in order to specialize an intrusion detection system to meet the organization’s security needs. Previous work on intrusion detection has not been motivated by policy, but instead on generic attacks that can be launched against the system. Consequently, a major part of the proposed work is the development of a policy language that will assist an organization in deciding on what data to collect, and it will serve as the oracle against which the data is analyzed.

Five tasks are proposed to cover various aspects of the insider problem.

- Task 1 will produce a characterization of misuse with emphasis on the insider.
- Task 2 will determine measures for characterization of sessions and will identify algorithms for carrying out the analysis of the data with respect to a site’s policy.
- Task 3 addresses the detection of compromised software.
- Task 4 considers data visualization methods for the human-analysis of audit events and for the formulation of policy.

- Task 5 considers fundamental issues related to misuse detection, such as the development of a policy language, techniques to generate and protect audit data collected from the execution of applications, and limits to the effectiveness of intrusion detection.

The main deliverables will be algorithms, reports, and prototypes of misuse detection systems. The goal is to demonstrate the feasibility of misuse detection in an Intelligence setting. The paucity of unclassified data and applications for the Intelligence problem, leads us to use the medical problem as our running application. In particular, we will monitor accesses to medical databases and conduct the analysis with respect to policies for the release and modification of medical data.

The work on misuse techniques will draw heavily from previous and current U.S. Government sponsored work at UC Davis on intrusion detection. The work on the medical application will benefit from a close collaboration with the UC Davis Medical Center which has an active program in Medical Informatics.

The remainder of this proposal includes sections on the status of intrusion detection technology, a description of the tasks to be undertaken (including our technical approach), the qualifications of UC Davis for the proposed work, proposed deliverables, biographies of the personnel (including key personnel) who will contribute to the work, and the budget.

2. Background

In the face of a rapidly growing Internet that contains many weaknesses that allow outsiders to a domain the opportunity to “break-in,” much attention has been given to technology to improve the security of systems connected to the Internet. There is increasing interest in the development of trusted operating systems which offer security assurance not available in current commercial systems, but the availability of such trusted systems is not imminent. Cryptographic-based systems offer solutions to some security problems but in the absent of trusted hosts, security weaknesses will remain. A promising approach is intrusion detection, wherein suspicious activity is detected and forms the basis for a suitable response. Intrusion detection can, in principle, be applied to all of the components of a network (hosts, routers, servers of all kinds), as all components are likely (if not known) to contain weaknesses that leave open to attack the components and other components dependent on them.

One pressing need is for an intrusion detection system that can respond successfully to an attack of the severity of the Internet Worm of 1988, which caused the Internet to got off the air for about 5 days. 7 years later, there are no systems to detect a problem on an Internet-wide scale. We can conceive of other wide-scale coordinated attacks on systems similar to a worm which might occur in an information-war scenario. The lesson of the Internet Worm was that such an attack defies the ability of humans to stop it due to lack of coordination among various computer organizations and inability to detect the problem (this section needs work).

Most of the current work on intrusion detection is directed towards the detection of outsiders attempting (or succeeding) to gain access to a system. In turn, very little effort is being directed to the detection of suspicious behavior by an insider. The early intrusion detection systems (IDES, NIDES, Wisdom and Sense) had the capability of detecting suspicious behavior by insiders through statistical-based anomaly detection. However the data they used was largely system-level audit logs, which do not contain sufficient information to detect many attacks. For example, the system-level logs associated with a database application will not be useful to detect activity that is dependent on the semantics of the database system; application level auditing is essential for this task, and there has been little effort directed to application-level auditing. Another weakness of the anomaly-based systems is the absence of a security policy; any behavior that is inconsistent with a profile is judged to be suspicious. Our proposal will investigate techniques to drive the detection of insider misuse from a policy specialized for the site being monitored.

3. Statement of Work and Technical Approach

3.1 Characterizing Misuse

An effective way to detect misuses is to characterize the misuses as a set of identifiable use patterns, and to monitor the system for misuses. If the specification of misuses is incomplete or imprecise, some classes of misuses will not be identified and/or some non-misuse classes will be detected. A goal of this proposed research project is to develop techniques to characterize these misuse patterns.

There are two approaches misuse detection. One is to characterize the expected user behavior on the system, and report a misuse when the uses of the system deviate from the expected behavior. Another approach is to characterize the known misuse patterns, and report misuse when a match to a pattern is discovered. For either approach, we need to develop a set of parameters to characterize the system usage. Some possible system characteristics include:

Commands invoked. It may be possible to describe user activities on a system by a set of user commands. For example, a computer programmer should mainly use code development commands, such as running the compilers. If he/she attempts to issue system administrative commands, he/she might be misusing the system.

Information accessed. It may be possible to identify the set of information that users are expected to access. For example, a programmer who accesses files in others' personal directories might be misusing the system.

The way the information is accessed. The amount of information being accessed, the rate of accessing the information, or the time the information is accessed may also serves as misuse indicators. For example, it is normal to see a physician access a few patient records a day. However, when a physician retrieves hundreds of patient records at a time, he/she might be misusing the system.

User roles. In a role based access control system, a user may take on several roles. A user may bypass the separation of duties control by using one role to perform the misuse activities while taking on the control role to cover up the misuses. Therefore, it is necessary to characterize the proper uses of the roles. The access pattern also depends on the user's role. For example, when doing research, a physician may need to access a huge amount of patient records, but during normal medical practice, he/she should access his/her patient information only.

A policy language will be developed to express the misuse characterization. The policy written in this language can be parsed by the monitoring system which reports misuse when the policy is violated.

We will also review existing misuse detection methodologies and determine where new techniques are needed. Some methodologies to be investigated include,

1. statistical approach: generates system usage profiles, or categorizes sessions into different categories;

2. specification-based approach: specify the expected or anomalous user behavior; and

3. artificial intelligence approach: using feature selection to identify the essential features that characterize computer usage. The work here would extend our preliminary investigation on the use of search techniques to determine a near-optimal feature set. Our work to date is promising for a small feature size (about 20) and is completely automatic. Work is needed to handle larger feature size (about 1000 features) and to handle statistically dependent features.

Experiments will be carried out to evaluate the various detecting misuse methodologies with respect to our chosen application domain. The evaluation criteria include, but not limited to,

- the rate of false positive and false negative;

- the ability of detecting misuses occur in a distributed system with a massive amount of data

- the ability of detecting misuses that spread over a prolonged period of time;

- the ability of adapting to changes of system usage, or how long it takes for the detecting system to pick up the new system usage pattern?

- the capability of incorporating application-dependent usage pattern, e.g., allowing the specification of the job requirements;
- percentage of performance degrade (computing power as well as storage space) brought to the system.

3.2 Session Characterization

Given definitions of "misuse" and "intrusion," one must determine the characteristics of a session relevant to each. A specific sequence of system calls designed to delete a system-critical file indicates misuse, possibly preceded by an intrusion. The characteristics of misuse and intrusion differ; intrusions are characterized by an attempt to exceed or acquire authority, whereas misuse is characterized by repeated and unusual exercise of existing authority. Doorknob rattling (intrusion) would show repeated network connections and disconnections, whereas browsing mail files looking for passwords (misuse) would show repeated accesses to a set of distinguished files (mailboxes) which the user is authorized to access.

The precise characterizations may be single commands, sequences of commands, sequences of keystrokes, intra-keystroke timings, sequences of system calls, arguments to those calls, or other qualities related to the security state of the system. Extracting these quantities will require the entire session, plus any ancillary information in the characteristics. Note that secondary characteristics are also relevant. A misfeasor may attempt to write to several configuration files in order to gain privileges; here, the relevant characteristic is that privileged programs read all the files accessed. Then these must be reduced (normalized), analyzed, and correlated. As this project will require extensive experimentation to determine the precise characteristics and measures to associate with misuse and intrusion, and the measures to distinguish the two, this means the entire session must be recorded and saved. Once the characteristics are determined, then one need only reduce the session by extracting and storing the relevant sets of characteristics.

Compression techniques will depend upon the nature of the characteristics. One can represent commands and system calls as tokens, and collapse other characteristics into less space, as most parts of a session are at least as redundant as the English language. Then a standard compression technique (such as LZW) can compress this even more. As analysis techniques can decompress the data on the fly, we omit further discussion of this point.

An analyst may use a number of techniques to analyze the resulting session extract. Neural nets have shown some promise. Pattern matching is more complex, as one must define a pattern matching language to capture all the characteristics, and the analysis engine must know the patterns (signatures) of intrusion and misuse exactly. A better approach is to use approximate pattern matching, in which a metric determines if the string is "close enough" to the pattern to be of interest. Given the expected complexity of many attack signatures, and in particular any dependence on time, approximate pattern matching may be able to collapse some of the possible values of the signature of interest.

An interesting idea is to use some of the N-gram work of the CIA, the NSA, and Lawrence Livermore National Laboratories. These studies attempted to characterize documents based on statistics related to the occurrence of specific N-grams. If one views the characterization of an intrusion or misuse event as a "document" in the "language" of the characteristics and their possible values, one can expand the notion of an N-gram search to this document. In some sense, it is similar to pattern matching, but much simpler and quicker to perform. Success is directly related to the efficacy of the signatures, and the ability to express them in a form amenable to N-gram analysis.

Thus, the sub-tasks for this task are:

1. Identify the characteristics of intrusion. This will be performed by analyzing known intrusions and attacks and extracting features sufficient to distinguish them from normal use and from misuse.
2. Identify the characteristics of misuse. This will be performed by analyzing known instances of misuse and attacks and extracting features sufficient to distinguish them from normal use and from intrusions.
3. Determine how, if at all, one could compress the data contained in a session beyond what is possible with current standard compression techniques. The analyst must be able to reconstruct the relevant characteristics of the original session from the compressed version.
4. Compare the effectiveness of approximate pattern matching and N-gram analysis with more standard analysis techniques (such as neural nets or pattern matching).

3.3 Compromised Software

The goal of this task is to investigate techniques to discover compromised software, that is programs that are "doctored" in a manner to benefit the person who compromised the software or cause pain to the user of the doctored programs. One can imagine an employee attempting to doctor a program in order to acquire access to files or other data he is not authorized to access. Also, through the use of Java, an unsuspecting user might execute programs that have access to his workspace; the detection of malicious Java programs is being investigated, but no general techniques are emerging.

In the task we will address the following issues:

- A specification-based approach to the detection of compromised programs, where suspicious behavior is detected at run-time.

- A statistical-based approach, again where detection takes place at run-time.
- The limits of these run-time approaches, but enhanced with static analysis of the system configuration after suspicious behavior is detected.
- Static analysis of possibly suspicious programs based on an analysis of programs with respect to tell-tale signs of compromise.
- Applications of the above techniques to known compromised programs.
- Applications of these techniques to detect attempts to compromise an intrusion detection system.

UC Davis has been investigating a new approach to intrusion detection, which we call specification-based intrusion detection. This technique appears to be very useful to the detection of intrusions that exploit errors in privileged programs or servers which are trusted to behave properly. Briefly, for each program (or server) that must be trusted, we write a specification that captures its security-relevant behavior. We have developed a specification language, called the Parallel Environment Grammar (PEG), which represents the security-relevant behavior as a sequence of allowable events. As a simple example, privileged programs might be permitted access to certain files and ports, but would not be permitted to change permissions of critical files. As a more complicated example, an editor should have exclusive access to such files as the password file in order to prevent the (perhaps accidental) creation of inappropriate passwords; thus, PEG can be used to specify security-critical behavior for a concurrent program. Many of the security problems with privileged programs arise from race conditions that exploit “time-of-check” and “time-of-use” being critical events; sequences of activities that attempt to exploit such race conditions is prohibited behavior in a PEG specification. Being a grammar, a detection system based on PEG is implemented as a parser, where events are run against the parser which rejects any behavior outside its grammar. We have implemented a detection system that processes audit logs generated by the Solaris operating system.

For the proposed project, we would attempt to use PEG to detect compromises to programs in general. The creator of a program, assumed to be trusted, would write security-critical specifications for his program; or, the user of the program could write specifications that capture program behavior he is willing to accept. As we have implemented for operating system privileged programs, system-level audit logging would be a data source to detect compromised behavior. But, certain malicious behavior (see below) is not revealed through analysis of the audit records. To detect such behavior, it might be necessary to instrument the programs with run-time checks; we call this approach “application auditing”. We would investigate techniques to link in the instrumentation based on the user-provided specifications. The instrumented programs would have the appearance of a program being run in debugged mode; we would apply debugging techniques we have been investigating for sequential and concurrent programs, work that has led to a debugger called Dalek.

We will apply the specification-based method to known examples of compromised programs, for example viruses and Trojan horses deposited by hackers using *rootkit*. Also, we will evaluate the method on known compromised programs, for example causing unauthorized or excessive accesses to files.

Traditionally, intrusion detection has considered anomalous activity to be suspicious. Similarly, a compromised program will exhibit behavior different from its profile. Furthermore, anomaly detection should be a more sensitive measure for application programs than for system programs where the primary data source is logs of system calls. As with the specification-based method, the key is what data to collect and how to collect it. Some data can be collected by the system, but it appears that application auditing is essential.

As indicated above, there are limits to the effectiveness of any detection method dependent on system logs for its data. For example, the specification-based method can detect attempts to access a file a process has no authorization for, but cannot detect an attempt to write erroneous data to a file; system audit logs do not record the arguments to file writes. So, a trojan horse in the */etc/passwd* program that causes a null password to be written for a hacker would not be detected. However, the intrusion detection system can record that a particular critical file was written to and trigger a configuration analysis to check the contents of the file. The same technique can be applied to the detection of compromised programs. Again, it is essential to have a policy language that indicates program behavior that is acceptable. It appears to be feasible to develop a detection compiler that will generate code for both dynamic analysis checking and configuration analysis.

The detection of compromised programs is a generalization of the problem of virus detection. A “specification” for a program suspected of containing a virus would be “the virus does not spread to other files”. Run-time approaches have been considered for virus detection, but the predominant defense against viruses is static analysis involving virus scanners. Of course, virus scanners can detect only known viruses, and are likely to be ineffective against polymorphic viruses (viruses that mutate as they spread). For known kinds of compromises, pattern recognition with respect to the known compromises would work. But, it is unlikely that all known kinds of compromise will be available, as any method we would develop should be effective against unknown compromise. Our approach would provide a language in which a security analyst could state the “tell-tale” signs of compromise, such as unexpected file accesses, unauthorized probing of a database. Then, a static analysis of the program could be carried out with respect to this property in order to determine the presence of program code that could the program to behave according to this property. The analysis involves dataflow analysis of the program in order to generate a slice of the program with respect to this property. We have developed a slicer for C that would be used in this effort. The key research is in the development of a language in which compromises of interest can be stated.

In summary, we will conduct the following investigations in connection with this task:

1. Develop a specification-based approach to the detection of compromised programs, where suspicious behavior is detected at run-time.
2. Develop a statistical-based approach to the detection of compromised programs, where detection takes place at run-time.
3. Determine the limits of run-time approaches. Enhance the run-time approaches with configuration analysis triggered by the run-time detection of suspicious behavior.
4. Investigate the feasibility of static analysis of possibly suspicious programs based on an analysis of programs with respect to tell-tale signs of compromise.
5. Apply the methods developed to known compromised programs.

3.4 Data Visualization

This task is concerned with visualization of security related data such as audit logs, security policies and protection configuration. Security officers need data visualization tools to cope with the large amounts of audit data, and to bring the human pattern recognition abilities to bear upon the problem of assessing the extent and severity of detected instances of misuse. Such tools can also help determine if an existing system configuration is secure with respect to the system policy.

The Davis Audit Workbench project has developed tools (e.g., visual audit browser [VAB]) to graphically display events from the SunOS host audit logs and to browse host audit logs using an interactive graphical interface and to display the aggregation of tens to hundreds of audit events simultaneously in a single graph. Audit browsing is the direct inspection of audit data in search of clues that indicate malicious activity (intrusion detection), to recount the steps of an already detected attack (post-mortem), and to assess the damage and extent of an activity.

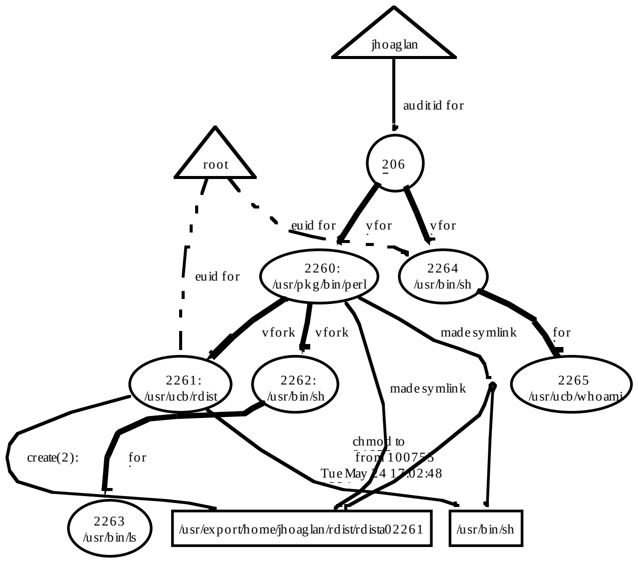

Figure 1. Graph of an rdist session attack

Characteristically, browsing does not have a precise methodology or well-defined algorithms as it is most useful when applied in situations where a systematic analysis is unavailable. The visual audit browser renders the events of an audit trail as a graph diagram, aggregating accesses to objects by users and their processes. The graph is an easily digested summary of the accesses by users occurring during a session. A further enhancement to the static graphical display is a set of graphs comprising a “movie” of audit log events. Each frame of the movie is annotated with the current access as well as aged versions of previous accesses. The aging function selects certain accesses to emphasize and de-emphasizes others through shading and color.

We propose to investigate graphical representations of audit log and policy data. We propose to study these technical issues in browsing including audit reduction (filtering), audit aggregation, audit log representation, techniques of visual presentation including graphs, animation and hypertext, the formation of associations (links) and annotation of audit logs.

We will extend our previous work on visualization SunOS audit trails to visualize audit trails from NT systems and IP/LAN Manager networks. Furthermore, we will investigate visual representations of policy and policy enforcement mechanisms. The initial experiments will be visual representations of a simple access control matrix. We want to develop techniques to display the histories of changes to the matrix (paths) as well as secure and insecure states. There are significant challenges in scaling such a simple approach to a realistic system composed of hundreds of users and tens of thousands of objects. Addressing these scalability issues are necessary for effective data visualization of misuse.

Thus, the tasks for this section are:

1. Port existing Audit Workbench visualization tools to use NT audit logs and NT registry.
2. Investigate methods to graphically represent the state as well as the history of an NT registry, or its generalized abstraction, an access control matrix.
3. Investigate visual representations of session data (as developed in Task 2). These representations enable the security officer to rapidly recall previous attempts to perpetrate similar misuse.
4. Develop interactive, three-dimensional visualization techniques for the rendering and analysis of statistical data generated by some intrusion detection systems. Furthermore, current techniques rely on the analysis of vector-valued data sets that capture the essence of sessions. Visual means to represent these vector valued data sets will allow the interactive analysis of large quantities of session protocols and will allow a viewer to instantly depict anomalous pattern in user behavior.

3.5 Misuse Detection Factors

Starting with the original Anderson report on audit analysis, there has been extensive work during the past 12 years on intrusion detection and related techniques. Some of this work is leading to prototypes and even commercial products. Despite this extensive work, there remains numerous open questions regarding the effectiveness of the technique for particular applications (such as the insider problem) and in general. The purpose of this task is to consider some of these questions as the basis for further work on intrusion detection.

The questions to be considered are the following:

What is the feasibility of policy-driven analysis? In principle, the policy of an organization should drive the entire intrusion detection system, identifying data to be collected and serving as an oracle against which the data collected is analyzed. Signature-based misuse systems employ a simple policy—attacks which are to be detected. But, an organization’s policy involves much more than just signatures and previously seen attacks. Among the policy issues that could be addressed by an intrusion detection system include: allowable connections, frequency of accesses, data to be accessed from a database system, temporal ordering of events. Except for the temporal properties, these are expressible in our Parallel Environment Grammar (PEG). We will would consider extensions to PEG to address temporal properties and to show its utility for general policy specification. We will formulate important policies in an extended PEG. To produce specifications that are more perspicuous, we will also use a graph-based language to write policy specifications.

What are the limits of intrusion detection? Intrusion detection is mostly untested for large problems, i.e., many users and hosts, complicated policies, large networks. The problem is exacerbated if real time detection is required. The work to be carried out here would develop preliminary models to determine the limits to intrusion detection. We would assume data sources such as audit logs, application logs, configuration analysis reports, and network sniffers. We would also assume that the detection engine carries out anomaly detection, misuse detection, and specification-based detection. Without loss of generality, we would assume that the analysis is based on rulesets, and we would develop models that determine the computation costs as dependent on the number of external events and the population size.

How can object auditing be implemented? Throughout this proposal we have mentioned the need to collect data from the execution of application programs. Work at UC Davis has led to LAFS, a system to audit user-level file system according to a user-specified policy; LAFS augments the Unix file system with special files to record the analysis of calls to a user's file system, and relies on the protection afforded by Unix to file systems; additional security measures could be included. In addition, we have developed preliminary designs for audit systems for database systems and for object-oriented systems in general. Given the attention being given to COBRA-based systems for distributed object management, we believe that it is timely to consider auditing for CORBA-related systems. We would develop a preliminary design for augmenting CORBA with auditing, considering such issues as what data is to be collected, where the data is to be reduced and logged, what kinds of analysis should be carried out, how policies for the use of objects can be specified.

3.6 References

Jim Hoagland, Christopher Wee, Karl Levitt, "Audit Log Analysis Using the Visual Audit Browser Toolkit," *U.C. Davis Computer Science Department Technical Report* CSE-95-11, 1995.

Allan Heydon & J.D. Tygar, "Specifying and Checking Unix Security Constraints," *USENIX Security Symposium*, 1992.

Kenneth C. Cox and Stephen G. Eick (1995), "3D displays of Internet traffic," *Proceedings. Information Visualization '95*, Nahum Gershon and Stephen G. Eick, eds., IEEE Computer Society Press, Los Alamitos, CA, pp. 129--131.

Stephen G. Eick and Graham J. Wills (1993), Navigating large networks with hierarchies, *Proceedings. Visualization '93*, Gregory M. Nielson and Daniel Bergeron, eds., IEEE Computer Society Press, Los Alamitos, CA, pp. 204--210.

Richard A. Becker, Stephen G. Eick, Eileen O. Miller and Allan R. Wilks (1990), "Dynamic graphics for network visualization," *Proceedings. Visualization '90*, Arie E. Kaufman, ed., IEEE Computer Society Press, Los Alamitos, CA, pp. 93--96.