AudES—An Expert System for Security Auditing

Gene Tsudik and Rita Summers

Computer security auditing constitutes an important part of any organization's security procedures. Because of the many inadequacies of the currently used manual methods, thorough and timely auditing is often difficult to attain. The recent literature suggests that expert system techniques can offer significant benefits when applied to security procedures such as risk analysis, security auditing, and intrusion detection. This chapter presents an example of a novel expert system application, an expert system for security auditing (AudES). Issues in the development and use of the expert system that are unique to the application domain are discussed.

The importance of effective computer security measures has become increasingly evident with the advent of recently publicized intrusion attempts and virus attacks. Any organization implementing computer security policies is faced with a wide range of potential threats. Although some types of threats can be effectively countered using real-time methods (Anderson 1980; Denning 1987; Lunt and Jagannathan 1988), detection of others remains too time or resource intensive to address in real time. After-the-fact security auditing is frequently used to detect anomalous events that fall out of the scope of real-time security measures. A landmark study by Anderson (1980) suggests that exter-

nal intrusion attempts can be detected by auditing log-in records and that some internal intruders can be detected by analyzing resource access attempts.

Some recent literature indicates that AI techniques (expert system methods in particular) might have much to offer computer security practitioners (Summers and Kurzban 1988; Lunt 1988). The AudES expert system is an experiment in investigating potential expert system applications in the area of computer security auditing. It is designed to automate manual security auditing procedures and alleviate the burden on human auditors. AudES is interposed between a human auditor and the resource access control facility (RACF) (IBM Corporation 1990), a popular security mechanism for IBM mainframe systems.

Motivation

Currently, most RACF security auditing procedures are performed manually by scrutinizing system logs and identifying potential security violations. This arrangement is hardly surprising because security auditing has always been considered a sensitive task that inherently requires human expertise. Although manual auditing methods are sometimes adequate, when the volume of the audit data is high, and the data itself are diverse in nature, consistency and timeliness are frequently sacrificed. Therefore, effective security auditing is often beyond the limits of available auditor time. Moreover, it sometimes demands too much from human memory and problem-solving abilities.

RACF provides a report-generation facility, Report Writer, which is used to obtain listings of users' activity records. Because RACF records each potentially suspicious event and performs no filtering of violation patterns, huge volumes of audit data can be generated. An average mainframe system produces on the order of several thousand records every day. As was observed at one site with a dozen systems, the entire auditing process (generating reports, printing them, auditing and following up) consumed about eight hours each day for a team of two auditors.

Another problem plaguing manual auditing is the amount of paper generated. In addition to violation forms that are filled out (by hand) for each suspected violation, RACF reports are kept in printed form for extended periods of time to maintain audit trails. Altogether, the amount of paper generated can be enormous.

There is also a more subtle incentive for automation that is rooted in the social aspects of security auditing, namely, its relatively low prestige. In addition to being repetitive and rather unexciting, security auditing

is frequently viewed as a low-level job function. For this reason, despite widespread awareness of the need for routine security audits, the job is frequently assigned to employees on a part-time or temporary basis. The result is a contradictory situation where a sensitive task is being performed by insufficiently trained or motivated personnel. One of the consequences is a high turnover rate among security auditors, which, in turn, results in a lack of experts. (Even though most organizations have personnel who are knowledgeable in auditing and security issues, when it comes to fine details, experts are scarce.)

Existing automated tools built with conventional automation methods handle only part of the task. Because conventional programming does not afford flexibility, these tools tend to be difficult to transfer among sites and difficult to modify. Because of the inadequacies of the existing approaches (both manual and automated), security auditing has become a prime candidate for an expert system application.

Design Issues

In assessing the scope of the applicability of an expert system, it is helpful to examine the intended application domain and identify the activities that can potentially be incorporated into the system. Because the primary goal of a human auditor is to identify security violations, security auditing can be viewed as centered around the sequence of events that represent the violation life cycle: (1) occurrence, (2) detection, (3) follow-up, and (4) archiving. Items 2 and 3 are of particular interest and are discussed in the following subsections.

Detection

The *detection* of security violations entails isolating sequences of events that match certain patterns and identifying appropriate follow-up procedures. When performed manually, it can be an extremely time-consuming and tedious task because violations tend to make up only a small fraction of the audit data that a human auditor has to examine.

The biggest challenge in automating detection is the proper and robust representation of the *violation patterns*, that is, the sequences of events that constitute violations. Violation patterns can be determined by a number of variables: For example, depending on the violation type, thresholds and follow-up actions tend to differ. Another variable is the user type. Different classes of users are treated differently; for example, violation criteria for external users can be harsher than for internal users. The location of the user at the time of perpetration can also have bearing on the auditor's decisions. The time of day (for ex-

ample, business hours, after hours) and the date (workday, weekend, holiday, and so on) can be important in pinpointing suspicious events. Finally, each distinct combination of these variables typically determines a *threshold* (maximum number of attempts), exceeding which triggers a violation.

Another issue of concern is the large volume of data in the audit logs. At first glance, this concern seems to have little bearing on the system design, but in reality, the size of the audit logs is one of the most important characteristics that distinguish security auditing as an application domain.

Most available expert system packages target applications that use traditional expert system methodology, that is, a single consultation at a time. An example of a traditional system is a medical diagnostic system or a wine selection adviser. One common factor in all such systems is the two-step consultation process that begins with the expert system asking the user for some initial consultation data and then making a diagnosis or a recommendation.

In contrast, the violation detection process consists of a large number of iterations:

1. While there are more records,
   a. Acquire all records for a single user.
   b. Analyze records for all violations types.
2. Go to step 1.
3. Compute report statistics.

The iterative nature of the auditing process that, as previously mentioned, plagues human auditors does not lend itself to most expert system tools. The main reason is the amount of internal bookkeeping necessary to support multiple instances. In particular, the maintenance of long lists and records and complex memory management (efficient memory management is not a strong point of most expert system tools) because of iteration are just two of the factors contributing to the performance degradation. To combat this problem, AudES was made as lean as possible by implementing routine (not rule-based) tasks outside the knowledge base.

Follow-Up

Follow-up is the sequence of events immediately following detection. It requires timeliness and consistency. However, unlike detection, follow-up requires an active approach. In other words, an auditor might have to contact a suspected violator, contact a violator's manager, notify a resource owner(s), alert security, and so on. In summary, follow-up involves much communication and paperwork on the part of the audi-

tor—time that can be saved if the process is automated.

Although most organizations have guidelines for security auditing, they are rarely all encompassing and often leave some aspects of detection to the individual auditor's judgment. At times, even when a sequence of events meets a specific violation criterion, the auditor has to decide whether an investigation is warranted. For this reason, it might be beneficial to not completely automate follow-up procedures. Instead, an advisory approach can be taken, so that suspicious events meeting violation criteria are displayed or recorded, but the final decision and the appropriate actions are left to the human auditor.

Archiving and Verification

The verification of security audit procedures is frequently conducted by the management. The main purpose is to verify that prescribed auditing procedures are being performed, and corporate security guidelines are being followed. This process used to be a lengthy and manual one that involved a lot of paper shuffling. RACF reports were reviewed at random, and selected violations were traced to determine if appropriate actions had been taken. For this purpose, old RACF reports, individual violation reports, follow-up traces, and numerous other forms of paperwork had to be maintained.

With the detection process automated by the expert system, verification can easily be accomplished by validating the rules procedures implemented by the system. Accountability can be maintained by generating consultation traces that reflect the actions on the part of both the auditor and the expert system. Also, the amount of paperwork can greatly be reduced if instead of entire RACF reports, only suspected violation records detected by the expert system are archived.

Development Environment

Choosing the most appropriate tool for development is an issue of great importance in building an expert system. Ease of use, user interface, editing and run-time facilities, and speed of execution are just a few of the many attributes that must be considered in identifying the right tool. Not surprisingly, an optimal trade-off among all the desired properties is extremely difficult to find. Many expert system tools that target programmers emphasize speed but compromise user interface and run-time facilities, and other tools, those that target the end user, provide rich interfaces and easy syntax but sacrifice speed. In the case of AudES, the latter approach was seen as the preferred (though not the optimal) one. This choice might seem contrary to the goal of

speeding the auditing process, but the compensating factors (ease of use and modification, flexibility, and extensive user interface) outweighed the performance consideration at least for the initial implementation.

The expert system environment (ESE) (IBM Corporation 1989) was the tool adopted for AudES development. It is a commercially available product that has been successfully applied in a number of tasks, including prediction, diagnosis, planning, and decision making, in a variety of application domains.

ESE rules and commands use a simple English-like syntax that is easily understood by nonprogrammers (see examples later in the chapter). Automatic error checking and compilation are provided at edit time. Furthermore, ESE provides an extensive interface to external procedures, which is especially important for AudES, where some of the more routine tasks are implemented outside the knowledge base. An important run-time feature is the *explanation facility*, which allows the user to ask questions about the system's actions. Consultation traces are automatically stored, so prerecorded consultations can easily be reviewed or rerun.

However, as frequently happens in environments that facilitate AI methods, the emphasis on generality and ease of use greatly affects the performance. Because of ESE's performance limitations and the typical volume of audit data, much tuning was necessary to meet the performance criteria. Whenever possible and appropriate, routine tasks not requiring rule-based techniques were implemented in Pascal subroutines.

Knowledge Acquisition

As an inherently sensitive task, security auditing is relatively well specified, unlike most other application domains. In most organizations, security auditing procedures are scrupulously documented. Furthermore, every attempt is made to cover all possible scenarios to leave as little as possible to the auditor's judgment. All this documentation should make the process of knowledge acquisition rather uncomplicated. Experience with AudES, however, has shown that it is not always so.

For the most part, the guidelines and other documentation used by security auditors are written by nonprogrammers, which is neither surprising nor unusual. People responsible for writing this documentation are, in general, skilled security professionals who are well qualified for this kind of work. However, written documents are not as conducive to covering all possible aspects of the task as is programming (even in a

high-level expert system shell). This difference leads to a curious phenomenon where violation conditions previously ignored or missed are recognized as such at the knowledge-acquisition stage.

An example of this phenomenon is the so-called partial log-in violation. A *mainstream log-in violation* is defined as a sequence of *M* or more failed log-in attempts for the same user identification (ID) (*M* is also known as the log-in violation threshold). A typical sequence of failed log-in attempts consists of the following steps: (1) Three attempts are made to log in, which constitutes a grace period. (2) The fourth failed attempt causes the user ID to be revoked. (3) Subsequent log-in attempts are recorded but ignored; that is, the user is not logged in even if the correct password is entered.

Human auditors are instructed to spot sequences of *M* or greater log-in attempts. If an auditor sees, say, *M-3* attempts, he/she ignores them. However, if these *M-3* attempts begin with a revoke record, then at least three attempts (grace period) must have previously occurred. Thus, even though *M-3* attempts are recorded, M must have taken place. This situation (and other similar ones) occurs when a sequence of records is split between two RACF reports (for example, when attempts are made on two consecutive days). Of course, events of this kind are not frequent and, thus, are not generally considered by those who write the security auditing guidelines.

Another discovery made during the knowledge-acquisition stage is the multisystem violation. Many users at the experimental site have accounts on several systems under the site's administrative control. As described in Motivation, RACF reports are individually produced and audited for each system. Because no correlation between systems' activity is performed, potential violators can slip through by distributing their attempts among several systems. Although this type of violation is very difficult to identify with manual methods, an expert system makes the task feasible.

Current Version

In its present state, the AudES system incorporates all the functions pertaining to violation detection. All RACF auditing rules and procedures have been fully represented in the knowledge base.

Twenty-five events are monitored and recorded by RACF. They range from log-in records to modifications to RACF itself. In addition, each event type has several (three to eight) possible return codes or qualifiers. Each [event, qualifier] combination forms a distinct exception type. There are a total of 89 exception types, which AudES groups into

IF (resource-violation-counter > resource-threshold)
AND (user-type IS *CUSTOMER*)
AND (resource-type IS *INTERNAL*) **THEN**
recommended-action = "Contact the resource owner and report incident to Customer Assistance"

Figure 1. AudES Rule Example.

four categories: (1) log-in—failed log-in attempts, for example, invalid password supplied.; (2) resource—unauthorized attempts to manipulate resources, for example, access, rename, delete; (3) command—attempts to execute restricted-privileged commands, for example, various RACF commands; and (4) mixed—the previous violations not detected but the combined probability of all violation types exceeding a specified threshold.

RACF divides users into four classes: unprivileged, special, operations, and auditor. *Unprivileged users* constitute the majority. *Special users* are the ones with the highest security clearance, such as the site security administrators; they are virtually given carte blanche. *Operations users* are typically the system programmers who are given some limited authority to manipulate RACF. Finally, *auditors* are users who are permitted to access activity reports and use the RACF Report Writer.

AudES supports all RACF-supported user classes. Furthermore, it subdivides unprivileged users into internal and external (or outside) users. The internal users are (in our case) IBM employees, whereas external users are customers, vendors, contractors, and various other non-IBM system users.

AudES treats each combination of user class, exception type as a distinct violation type. Each violation type has a corresponding (configured) threshold, that is, a minimum number of records that cause AudES to suspect a violation. In addition, each violation type is associated with a prescribed sequence of follow-up actions that are presented to the auditor on violation discovery.

Figure 1 shows an example ESE rule that AudES uses to identify suspected violations. The particular example is concerned with identifying anomalous resource access attempts:

Consultation

A typical AudES consultation proceeds as follows: First, the auditor

runs a filter program on all raw RACF reports to produce AudES-readable input records. Next, ESE is used to load AudES. The auditor can preselect a number of reports to audit by entering their names in a profile or can select them interactively at run time. The consultation can be conducted either in the interactive or the unattended mode.

In the *interactive mode*, AudES displays suspected violations as they are discovered and waits for the auditor to pronounce judgment (that is, initiate or ignore). All suspected violations are tagged as processed when in this mode, and the auditor's decisions are duly recorded. In *unattended* (or batch) *mode*, the auditor runs a consultation in the background without requiring any input from the auditor. In this mode, suspected violations are recorded but tagged as suspended. This mode is particularly convenient when auditing must be done at odd hours of the day, or available auditor time is limited. Regardless of the mode, all suspected violations are recorded to disk to serve as the proof of audit. Suspected violations are recorded and displayed (interactive mode only) along with all the necessary user data as well as the information on commands; resources; affiliations; and, most importantly, appropriate actions that an auditor must take according to the local security auditing guidelines. When the consultation is completed, the auditor can obtain a hard copy of the recorded violations. In addition to consultation reports, the entire consultation (at the keystroke level) is recorded in a separate file that can subsequently be used to replay the entire consultation. This feature can be used for verifying past audits.

Portability

AudES can easily be adjusted to an individual site's requirements. The configuration is done by creating several consultation profiles for AudES execution. These profiles are created by the site security administrator (usually, the user with the RACF special attribute) with the help of a separate knowledge base, AudINIT. AudINIT takes the user through the configuration process step by step and checks the configuration parameters for possible anomalies and inconsistencies. The profiles and their contents are described in the following paragraphs:

The *site profile* contains site-specific information, such as the systems under the site's control, the users authorized to use AudES (auditors), the privileged users (special and operations), and the frequency of RACF report generation. The *event selection profile* determines how AudES detects suspected violations. Each violation type can be turned either on or off (monitored or ignored) and associated with a threshold. Follow-up actions for each violation type are also entered in this profile. The site and event selection profiles are usually modified infre-

quently. Finally, the *run-time profile* sets various AudES execution parameters. It specifies (among other things) the execution mode (interactive or unattended) and a list of reports to audit as well as whether to print the suspected violations. This profile can be modified as often as needed by the auditors to afford more flexibility.

Size

AudES is a relatively small system. Its two major components are the ESE knowledge base and the system interface. The rules and procedures are embodied within the ESE knowledge base. There are on the order of 80 rules and 15 focus control blocks (FCBs) that are used to group rules, parameters, and external actions. The system interface consists of 10 Pascal subroutines used for external actions, for example, fetching user records from RACF reports, identifying the user, and composing violation records. All the interface services are directly accessible from the knowledge base. In addition, there are two standalone utility programs: AudSTRIP and AudINIT. AudSTRIP is used to reformat (strip) the raw RACF reports before running AudES, and AudINIT is an ESE knowledge base that is used to customize AudES.

Deployment

Before replacing manual auditing procedures, AudES had to be thoroughly field tested to assure its correctness and to validate its rules. For eight months, it was used in parallel with manual auditing. This interim period proved to be extremely valuable largely because of the auditors' feedback regarding the user interface, performance aspects, and violation reporting. The reactions to AudES on the part of the auditors were extremely favorable. The ease of use, the savings in time and paperwork, and the ubiquity of unattended interactive consultation modes were the features most appreciated by the users.

At the time of this writing, AudES is in full deployment at one experimental site, having completely replaced the previous auditing procedures. It is being used on a daily basis for analysis of RACF reports generated on some 20 to 25 mainframes. The initial readjustment period turned out to be short (less than 1 week). The auditors find AudES easy to use and customize. In summary, the task that used to consume an entire day for a team of two and, sometimes, three, auditors now requires about 1.5 hours each day for a single auditor. Of this time, only one-third is spent in preparing input for and running AudES. The balance is taken up by printing and filing of AudES-generated violation reports.

Also at this time, AudES is being field tested at two other sites with

radically different user populations and business needs. Both sites are large (about 80 mainframe systems at each). The size presents a number of new challenges to AudES, of which performance consideration is the most immediate.

In conclusion, the benefits from the use of AudES, as opposed to manual RACF security auditing procedures, are (1) greatly reduced routine work for auditors, (2) greater consistency of audit, (3) timelier violation detection and follow-up, (4) the ability to conduct more frequent audits, (5) cost savings from reduced paperwork, (6) clear verification of auditing rules and procedures used, (7) easy customization to meet individual site requirements, and (8) reduced training time for auditors and a tool for training. The last three items also represent the benefits of an expert system approach as opposed to more conventional non-AI techniques.

Summary

AudES demonstrates the feasibility and the benefits of applying expert system methodology to computer security auditing. As the system evolves, it can acquire more knowledge and gradually take over some additional, more judgmental tasks. The working system is a start toward a longer-range goal of expert systems for security audit and administration. It is but one component of security maximization, and its integrity depends on the soundness and completeness of the auditing procedures that it implements. At the same time, AudES serves as an example of the application of expert system technology in a field that to date has largely been untouched. It demonstrates the feasibility and scope of the potential automation of security auditing procedures and exhibits the underlying issues, limitations, and concerns.

Acknowledgments

The authors would like to thank Stan Kurzban, Ray Martin and the anonymous reviewers for their insightful suggestions and comments on the draft of this chapter.

References

Anderson, J. P. 1980. *Computer Security Threat Monitoring and Surveillance*. Fort Washington, Pa.: James P. Anderson.

Denning, D. E. 1987. An Intrusion-Detection Model *IEEE Transactions on Software Engineering* 13(2): 222–232.

IBM Corporation. 1990. Resource Access Control Facility, general in-

formation manual, GC28-0722- 10.

IBM Corporation. 1989. Expert System Environment, general information manual, GH20-9597.

Summers, R. C., and Kurzban, S. A. 1988. Potential Application of Knowledge-Based Methods to Computer Security. *Computers & Security Journal* 7(1): 373–385.

Lunt, T. F. 1988. Automated Audit Trail Analysis and Intrusion Detection: A Survey. Presented at the Eleventh National Computer Security Conference, October.

Lunt, T. F., and Jagannathan, R. 1988. A Prototype Real-Time Intrusion Detection Expert System. In Proceedings of the 1988 Symposium on Security and Privacy, 166–188. Los Alamitos, Calif.: IEEE Computer Society Press.