Open access • Proceedings Article • DOI:10.1109/SECPRI.1989.36302

Detection of anomalous computer session activity — Source link

H.S. Vaccaro, G.E. Liepins

Institutions: Los Alamos National Laboratory, Oak Ridge National Laboratory

Published on: 01 May 1989 - IEEE Symposium on Security and Privacy

LEGIBILITY NOTICE

A major purpose of the Technical Information Center is to provide the broadest dissemination possible of information contained in DOE's Research and Development Reports to business, industry, the academic community, and federal, state and local governments.

Although a small portion of this report is not reproducible, it is being made available to expedite the availability of information on the research discussed herein.

LA-UR--88-3656

DE89 003607

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36

TITLE DETECTION OF ANOMALOUS COMPUTER SESSION ACTIVITY

AUTHOR(S) Henry S. Vaccaro, N-4

SUBMITTED TO 1989 IEEE Symposium on Research in Security and Privacy
Oakland, CA
May 1-3, 1989

DISCLAIMER

Los Alamos National Laboratory
Los Alamos, New Mexico 87545

DETECTION OF ANOMALOUS COMPUTER SESSION ACTIVITY

H. S. Vaccaro
Safeguards Systems Group
Los Alamos National Laboratory

DETECTION OF ANOMALOUS COMPUTER SESSION ACTIVITY

I. INTRODUCTION

This paper describes recent Los Alamos National Laboratory (LANL) applications of research into automated anomaly detection ~~at the~~. In the context of computer security, anomaly detection seeks to identify events shown in audit records that are inconsistent with routine operation and therefore may be indicative of an intrusion into the computer, serious human errors, or malicious behavior by a legitimate user. Access by an intruder, execution of "Trojan horses" and "viruses, as well as malicious, destructive behavior are all assumed to produce anomalous events that are recorded in a computer audit trail. This trail, perhaps with augmented data collection capabilities, is processed, in real-time, to detect such events, alert a knowledgeable computer security officer to the threat, and help resolve the situation.

II. BACKGROUND

Despite recent major improvements to operating system security, available computer security features still are not good enough to detect many anomalous behaviors by computer "users" in time to prevent or minimize any damaging activity. The risks from stolen passwords and privileges, for instance, are still of great concern. Current computer security systems do not in general protect against:

- An imposter who gains access to a legitimate account and environment;
- A legitimate user taking advantage of mistakes in the configuration of system security measures;
- A highly privileged user behaving destructively;
- An executable program that has been tampered, through other means, to perform some new, improper function.

In practice, existing computer security systems are not necessarily configured effectively, so additional weaknesses typically exist.

Ironically, these same improper activities would generally be detected by an experienced human security officer using information on what was done and what resulted. In doing so, we believe that the security officer goes well beyond the inflexible application of "rules" describing intrusion scenarios.

- The experienced human recognizes the difference between "normal" behavior and "abnormal" behavior.
- When abnormal behavior is spotted, the security officer searches for "rules" of common sense to determine whether the abnormalities are important.
- If no rules fit exactly, new rules are devised on the spot by analogy with related situations.
- Then, if the officer concludes that the anomalies pose a risk, some sort of investigation begins.
- Depending on the outcome of this investigation, the new "rules" might be rejected or adopted. Similarly, his/her perception of normal behavior may be updated.

This is fundamentally the same approach we are trying to implement in software.

III. SOLUTION APPROACH

If behavior (e.g., of a user or an executable program) differs from normal patterns, and if data indicating the difference is collected, it should be possible to compare the new "different" data with the normal patterns and detect the anomaly. The problem can be solved by creating, in effect, specialized profiles of computer users, ports, executable software, privileges, time slots, etc., and determining whether the new data violates these profiles

We accomplish this profiling task via heuristics. Selected historical data is used to generate a tree-structured, instantiated rule base describing historical behavior patterns that were significant. The

rules define what was normal for the values in particular fields of the historical audit transactions, conditioned on combinations of the data in other fields of the transactions. Such data values may be computed based upon the contents of a series of related transactions (Fig. 1).

Chief among the factors that led us to this approach, as opposed to more classical pattern matching or statistical approaches, are

- Most of the audit data are non-metric, categorical information such as user names, privileges, action descriptions, access points, etc.
- Computer activity data has a significant random component; that is, the data is very noisy.
- Computer user activity exhibits transient as well as cyclical behaviors.
- Usage patterns drift as users become more experienced or gradually change the focus of their computer efforts.
- Parts of usage patterns abruptly change as users begin new projects, change offices, and restructure their computing environments.

These characteristics led us away from numerical approaches such as n-dimensional clustering toward new heuristics that were fundamentally categorical and which placed restrictions on what is considered "normal" only when there was sufficient applicable history. These factors also dictated that our approach be tolerant of conflicting historical behavior patterns.

The heuristics attempt to mimic in software part of the human learning and decision making process, and allow for expert opinion to supplement or modify the machine-learned information. We make provision for display of the factors contributing to a computerized decision so that an expert can verify the decision. Finally, we use the learned patterns as a predictive tool to help the expert resolve anomalies by presenting non-anomalous alternatives to the observed activity.

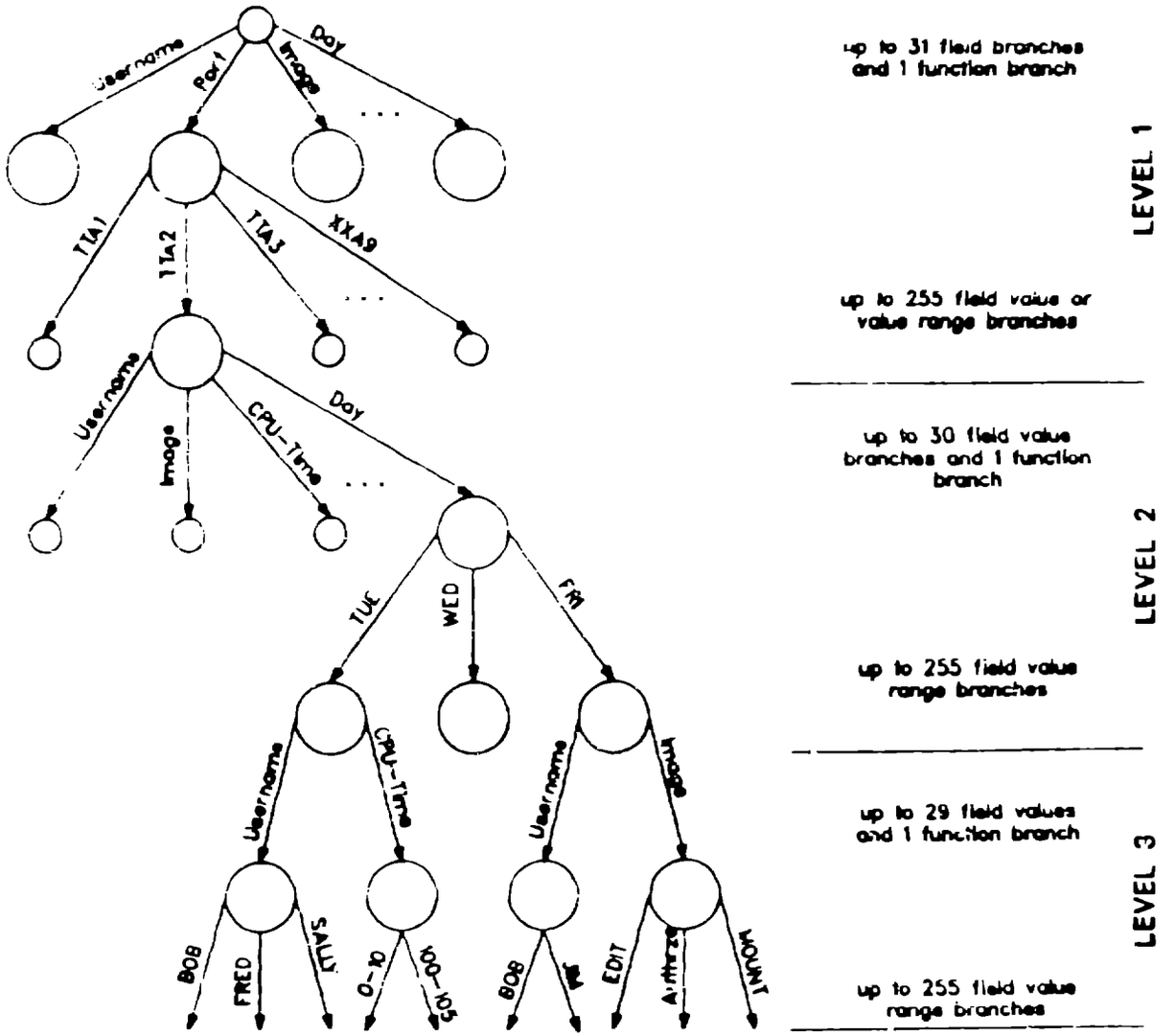

Fig. 1. W&S rule base structure.

IV. EXPLANATION

In our solution, a learning program examines a history of discrete system activity (individual commands, file accesses, etc.) to generate a production-like rule base by induction. The rule base it generates may be edited or supplemented by an expert to broaden its coverage or improve its decisiveness. The rule base is then applied to system activity transactions, either in batch mode or real-time, to determine which activity, or series of activities, is abnormal when viewed against the applicable patterns instantiated in the rule base. Naturally, the data used to build the rule base contains some anomalies, so these are heuristically filtered out of generated rules.

The rule base also reflects the quality of the behavior patterns it has learned. Patterns that occurred more often or with less noise have stronger grades.

A rule's strength is stored along with the conditions under which it applies (called the left-hand-side or LHS) and the implied conclusion (right-hand-side or RHS). A LHS in our approach is a series of field values or value ranges, or computer values based upon data in a series of related transactions (for example, mean time between some event type), or subroutines returning a boolean value. A given transaction satisfies the LHS and "fires" the rule if its values match those for the fields in the LHS and any subroutines in the LHS return true. Then we determine whether the transaction satisfies the rule's RHS conclusions.

We refer to the RHS as the rule's restriction, because it restricts what is considered "normal" for a transaction. Thus, our approach generates rules about the appropriate contents of transaction fields based on the contents of other fields in the same transaction or data derived from a sequence of related transactions. The latter is accomplished through a mechanism we call threads. Threads can access data for several related transactions to compute data such as a moving average of the time between login failures on a particular port.

For any collection of historical audit transactions, the derived rule base must have at least one thread. Basically, each thread has a figure of merit (FOM) which is the sum of time-decayed FOMs for transactions on the thread. One obvious thread that we find very important is a user-port

thread. Each time a specific user logs into a particular port, the thread continues. Thus, slightly anomalous transactions for the same user and port across several logins can lead to a user-port thread anomaly. Program-user threads and privilege threads also are of high interest.

The RHSs are limited to three basic forms:

- A list of acceptable categorical, non-metric values for a particular transaction field (e.g., the normal work days of the week).
- A list of acceptable ranges for a continuous, metric transaction field (e.g., the normal amount of disk I/O activity).
- A list of user-defined functions to be executed until one returns a true value, meaning the RHS is satisfied, or the end of the list is encountered, meaning the RHS is not satisfied.

In the absence of rules restricting, say, normal computer ports, any port is considered normal. This may seem risky, but it is the same way that humans work.

Still, there are in general tens to hundreds of thousands of rule instantiations on a major subject such as behavior on a computer system. The rules vary from very general (e.g., the valid ports are P1, P2, ... Pn) to very detailed (e.g., On Tuesday between 6:00 am and 7:00 am, when the user has system operator privileges and is using port P3 only commands that cause very little direct disk activity are used) (Fig. 2). Also as with human experts, very specific rules carry more weight in making a decision, provided that they are still based on clear behavior patterns.

An expert usually takes many paths to arrive at a conclusion (e.g., the normal disk activity might be inferred from the user and time of day, or from the account number, or from the program being executed). A rule base is built the same way--it is highly redundant. Thus, the inferencing process reaches its conclusion about the normality of an audit record along many different reasoning paths, resolving conflicts through a weighting and scaling process.

An audit record field that violates many conclusions (RHSs) as to its normal content is considered anomalous. If the record contains several anomalous fields or a highly anomalous field, the record is anomalous. If a series of related records, say those for a particular user's session, are anomalous, the entire session is considered anomalous.

Left Hand Side	Grade	Right Hand Side
Username AMY Priv1 10148001	=7>	Image AUTHORIZE SET SHOW
Priv1 10148001	=6>	Image AUTHORIZE SET SHOW
Priv1 10148001 Priv2 00000008 DIR_IO 0:378	=7>	Image AUTHORIZE SET
Priv2 00000008 Priv1 10148001 CPU_time 0:370	=7>	Image AUTHORIZE SET
Priv2 00000008 Priv1 10148001	=7>	Image AUTHORIZE SET SHOW
Day "Wed" Priv1 10148001	=6>	Image AUTHORIZE DELETE SET SHOW SYSGEN
Terminal TXC3 Priv1 10148001	=7>	Image AUTHORIZE SET SHOW
Username OREN	=6>	Image AUTHORIZE COPY DELETE DIRECTORY LOGINOUT MAIL QUEMAN SET SHOW SUBMIT VMSHELP

Fig. 2. Examples of image "AUTHORIZE" rules generated by W&S.

V. SOFTWARE IMPLEMENTATION

These concepts are now implemented in three main software sections: a data preprocessor, a rule base generator, and a transaction analyzer. Collectively, we refer to the software as Wisdom and Sense or simply W&S. Our implementation of these concepts has enabled the rule base to be stored in memory as a highly compressed tree structure using 6-7 bytes per rule, and the inferencing process to be real-time, firing roughly 20,000 rules per second on a $10,000 computer workstation.* Typical rule bases require 0.5-1.0 Mbyte of memory and can process about 20 transactions per second on the same workstation.

A. Design Criteria

We designed the anomaly detection software to embody the following capabilities:

- Reduce raw audit data to more usable forms;
- Build its own rule base without human guidance;
- Store and use very large, instantiated rule bases efficiently;
- Tolerate conflicting rules;
- Deal with uncertain and erroneous knowledge;
- Continue to learn from experience, and adapt to transient conditions;
- Accept human modifications to its rule base, but not be overly dependent on scarce human expertise;
- Make real-time, graded decisions regarding anomalous behavior;
- Provide human-readable feedback on anomalies to aid in anomaly resolution;
- Create minimal interference with the real functions of its host system;
- Be portable to different applications, operating systems, and hardware.

Most of these design criteria have been attained in our software. However, there remain many gaps in our ability to detect anomalous computer activity, and determine whether the anomalies are significant.

*All performance figures are for an IBM RT Model 6151-125 with an Advanced Floating Point Accelerator. The operating system is IBM's AIX Version 2.1.

We need more experience in operating environments, and with simulated intrusions, before we can design additional analysis tools for this purpose and properly tune W&S.

A difficult problem is that computer operating systems do not generally capture the right data for analysis. Furthermore, the amount of data potentially available can easily overwhelm any anomaly detection scheme, so we will have to choose data of the greatest value.

B. Data Preprocessing

For any given application, Wisdom is configured to read a specified fixed record format sequential file. VMS ALAP W&S is one such application that has been heavily tested so far. The VMS ACCOUNTING.DAT file used by VMS ALAP W&S is not in this commercial form, so VMS ALAP includes a special filter to perform VMS file I/O and data conversions resulting in a fixed record format. The filter is run as either a batch job to convert a large ACCOUNTING.DAT file or as a VMS ALAP file I/O subroutine to convert new additions to the accounting file in real-time. In either mode, the converted records are placed into a correctly formatted W&S file.

The historical transactions file, used by Wisdom in building a dictionary, condensed file, and rule base, generally contains 10,000 to 100,000 historical transactions (Fig. 3). The current activity file is of the same form, but it contains transactions to be processed through the inference engine, Sense.

The kernel is given a description of the history and activity files via a format definition file. Users can create this format file interactively from within W&S.

VMS ALAP uses "image termination" records from the VMS accounting log as its transaction source. These very useful auditing records are readily available without undue systems overhead for collecting them. (The actual typed command line would be useful as well, but auditing the command line poses special problems under VMS.)

We extract 16 fields from the standard image termination records, 12 of which are used for rule base generation, and the rest for display only (Table I).

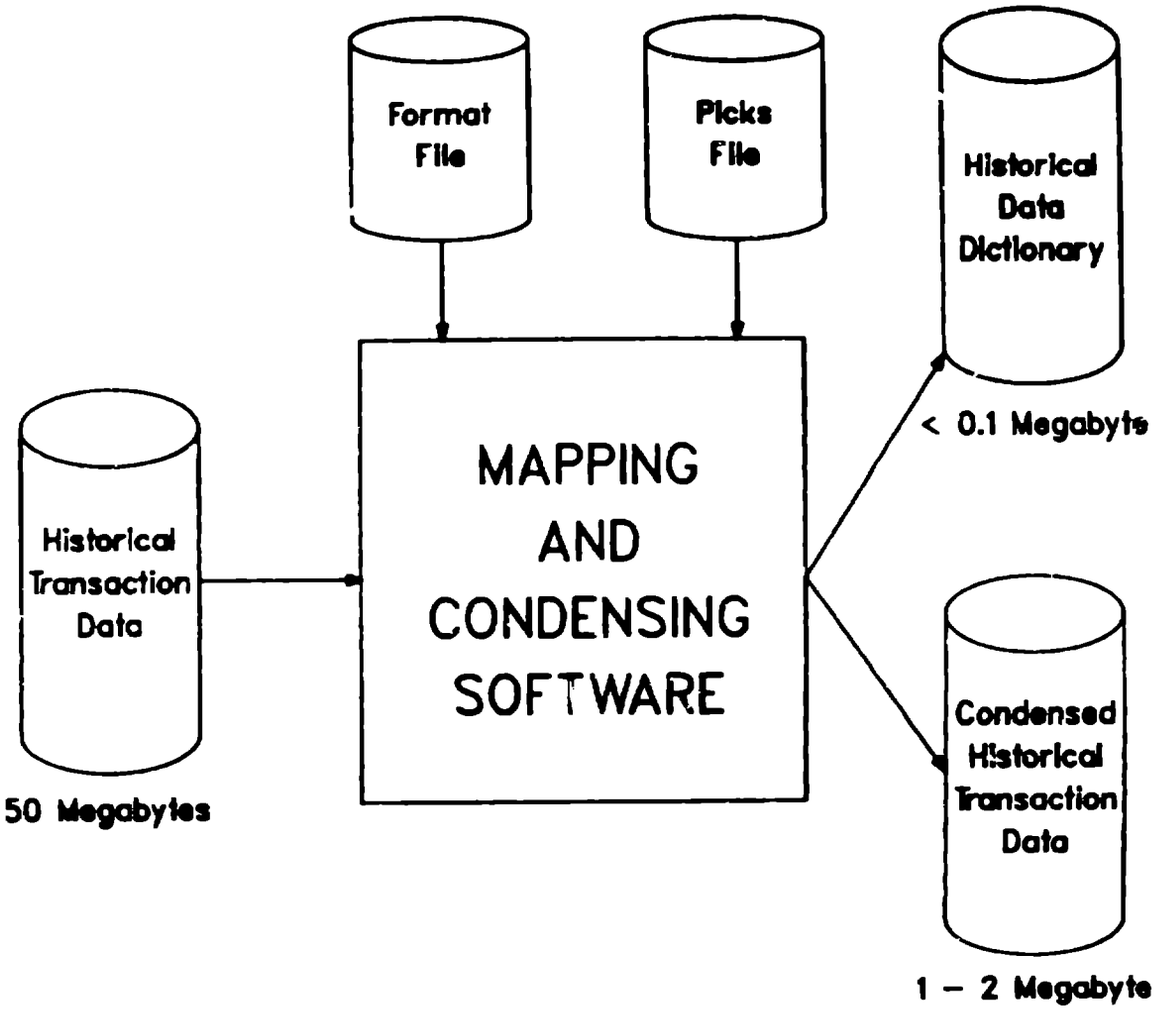

Fig. 3. Condensing an historical transaction file (typical file sizes on a 100,000 record VMS historical transaction file as input).

TABLE I

IMAGE TERMINATION RECORD FIELDS
USED IN VMS ALAP W&S

Field Name	Metric	Comment
Privl	no	first 32-bits of the privilege mask
Priv2	no	last 32-bits of the privilege mask
Status	no	32-bit program return code
Dir_IO	yes	Direct I/O - 512 byte blocks
Buf_IO	yes	Buffered I/O - 512 byte blocks
CPU time	yes	CPU milliseconds used
Username	no	User's login name
Image	no	Full name of the executed program
Day	no	Day of the week
Hr	no	Hour of the day
Terminal	no	I/O port name
Node name	no	Network node name, if any
Node ID	no	Network user ID, if any

The fields used for rule base generation are identified by a "picks" file, which is interactively created or modified from within W&S. The user specifies a format file, then picks up to 31 named fields in the format definition.

For each picked field, an optional mapping function can be designated. Mapping covers operations such as ASCII to integer conversions, integer to ASCII, numeric scaling, shifting character strings to upper case, string truncation, floating point to integer, etc. Proper mapping and data typing is important to obtaining a good rule base. For example, the day of the month in the time stamp of a transaction could be treated as a metric integer. We find it better to map it to a day of the week and treat it as a categorical value (Fig. 4,.

C. Rule Base Generation

We obtain high performance in rule base generation by condensing the historical file, so that it can reside in random access memory during the entire process. This condensed historical data is processed through the Wisdom rule base generator module, which builds a forest of rule trees. The structure uses nodes of two alternating types. One type designates fields and can have up to 32 branches. The other designates "normal" field values, and has at most 255 branches (Fig. 1.). Together, the two

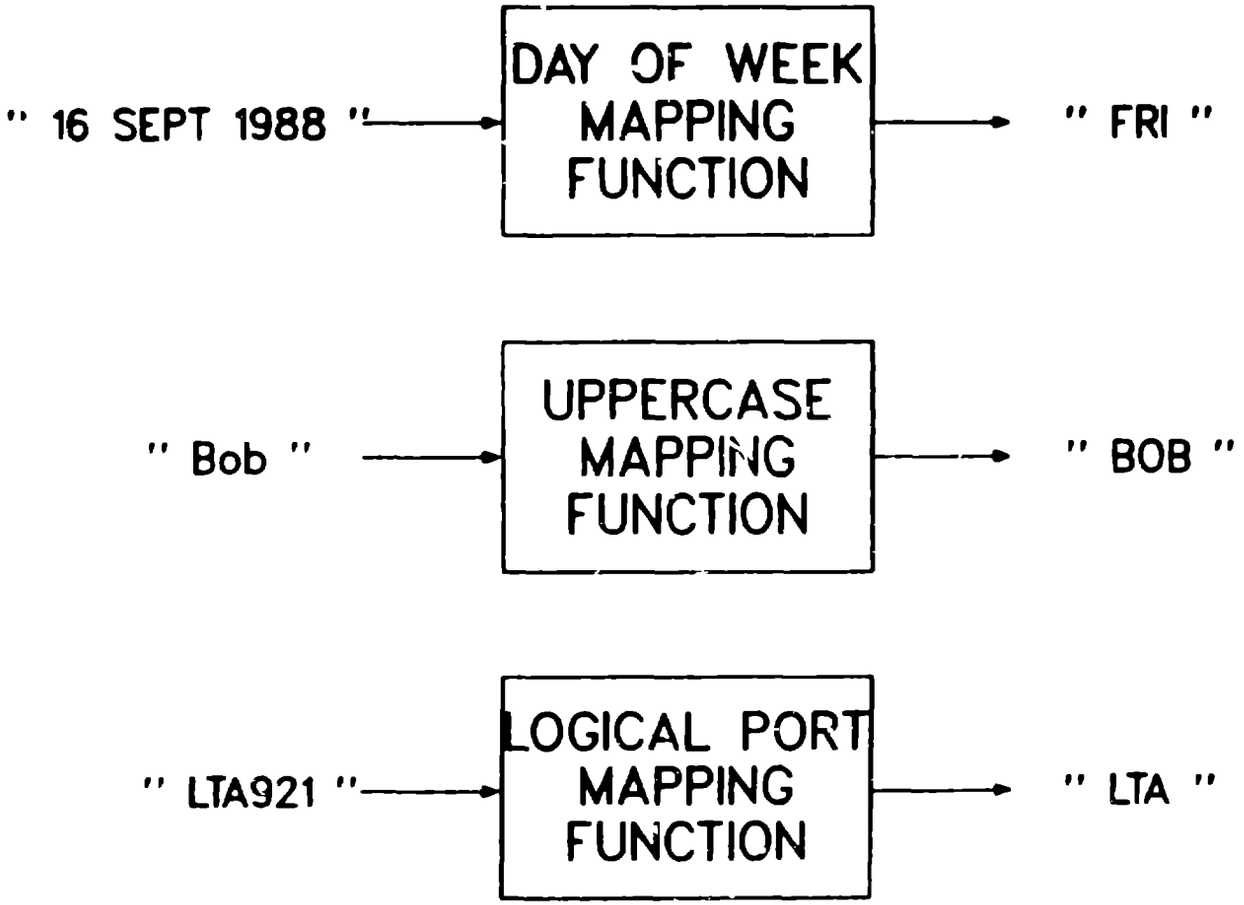

Fig. 4. Example mapping functions.

node types compose a rule base "level"; thus the topmost level can have up to 8160 (32 x 255) branches. We typically limit depth to 4 or 5 levels, so the tree forest is very broad (with an upper limit of 2.6×10^{19} leaves at level 5). Pruning algorithms ensure that the actual generated rule base is of acceptable size.

Rules stored in this forest require an average of 6-8 bytes each. This is achieved by massive sharing of rule base data by rules with similar structure, and through the use of the data value dictionary created when the historical data is condensed.

The rules themselves are generated by repeatedly sorting the historical data and examining the frequency of field values within sorted subsets of the records. Because we represent our condensed historical data via 1- or 2-byte fields, linear-time radix sorts can be applied,[1] and the in-memory sorts are very fast

1. **Metric Data Clustering.** Rule-building on metric fields uses a simple, ad-hoc clustering algorithm. The sorted data are viewed as a histogram, with the histogram bucket widths being variable, but with each bucket containing roughly the same number of points. The target number of points per bucket is given by an empirically developed function of the total number of points and an estimated anomaly fraction. The widest buckets represent the portions of the number line with the lowest data point density (i.e., the unlikely values) and are tagged as "anomalous" ranges until a target for total anomalies is reached approximately. Adjacent non-anomalous ranges are combined to yield contiguous "accepted" ranges (Fig. 5). The rule, if kept, then defines these ranges as normal values for the element given the LHS.

The metric element algorithms were originally tested on simulated normal and multi-normal distributed data with good results. Importantly for this application, the algorithms work well with multi-modal data and do not require normally distributed data. Furthermore, the computational time is small.

2. **Non-Metric Data Clustering.** For non metric fields, the least frequent values are tagged as "anomalies" up to approximately a target percentage. This target is based on expert judgment as to the likely

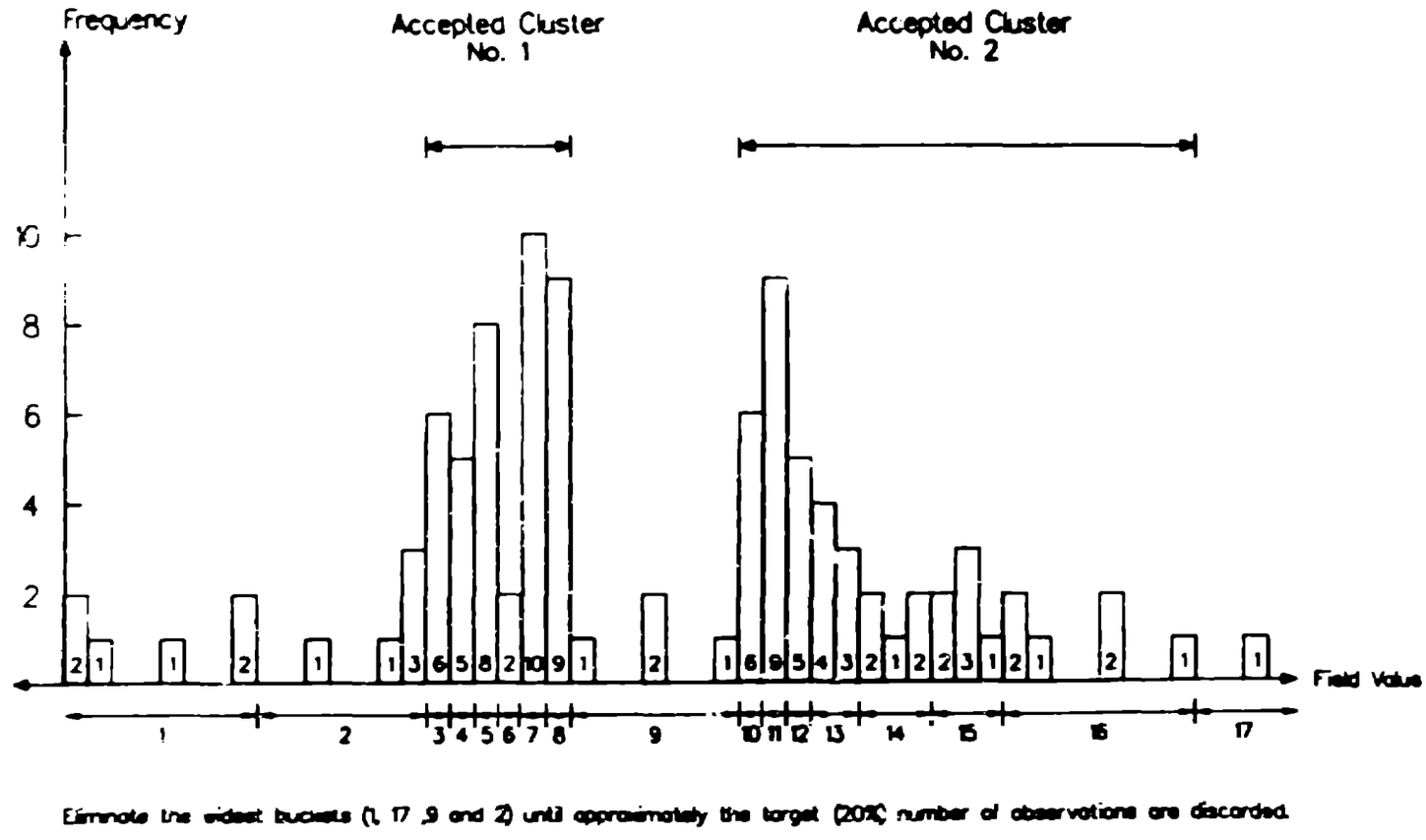

Fig. 5. Metric data clustering heuristic (100 observations, bucket size 6).

number of anomalies in each field. (Choosing 0% for each field would mean that the expert believes there are no anomalies in the historical data for that field.) The values not tagged are considered acceptable, and make up the RHS of a new rule, if the rule is kept.

3. **Rule Grades.** Each rule has a grade which is a measure of the historical accuracy of the rule. This grade, G, is calculated as:

$$G = (\text{int})\log_2 \left[\frac{(T + 2) \times D}{A + 1}\right] ,$$

where T is the total number of observations, A is the number tagged as anomalous, and D is the current rule base depth. The +2 in the numerator and the +1 in the denominator result from an assumed uniform a priori distribution for the grade. See Ref. 2 for computational details.

The grade can be thought of as approximately

$$\log_2[\text{historical odds that the rule was violated}] + \log_2[\text{depth}] .$$

The second term biases the grades in favor of more specific rules, i.e., those with longer LHS's.

4. **Pruning.** Tree pruning ensures that the rule base contains predominantly "worthwhile" rules, and that it avoids exponential growth. Ideally, each rule should add at least some specified minimum amount of new information to the rule base. Here is what we have found practical for small (10,000-40,000 records) VMS accounting log files:

(1) Rules with a grade below a threshold (typically 3) are discarded, pruning the tree at that point.

(2) Rules with too many different acceptable values are discarded. The thresholds depend on the current tree depth. We currently use:

(# unique values) x (depth - 1) < 13

The first, unconditional rules generated from the root of the rule base are at depth 1, and therefore are permitted to have the maximum number of branches (acceptable values)--255. Rules at depth 4 could have only 4 acceptable values.

(3) A rule whose LHS is a permutation of another rule's LHS is discarded.

(4) A rule whose RHS matches the RHS of another rule and whose LHS is a superset of that rule's LHS is discarded unless it has a higher grade.

(5) The rule is kept, but further growth is pruned if there is only 1 value allowed by the rule;

(6) Any rule's value with fewer than 30 + 50/(depth+1) observations is not permitted to grow new branches;

(7) No rules below level 6 are generated.

D. Transaction Analysis

The Sense modules (Fig. 6) provide an interactive, windowed interface to:

- the kernel's inference engine,
- transaction analysis tools,
- configuration settings, and
- rule base maintenance routines.

Its most important function, the detection and display of transaction anomalies, makes use of the rule base and dictionary generated by Wisdom and the inference engine described below.

Sense, the anomaly detection module, looks at a new transaction, finds the rules that apply, and synthesizes a transaction score. Normally, data that are newer than that used for rule base generation would be processed. However, the historical data must still be representative of the new data; if not, a new rule base needs to be generated from more representative (typically newer) data.

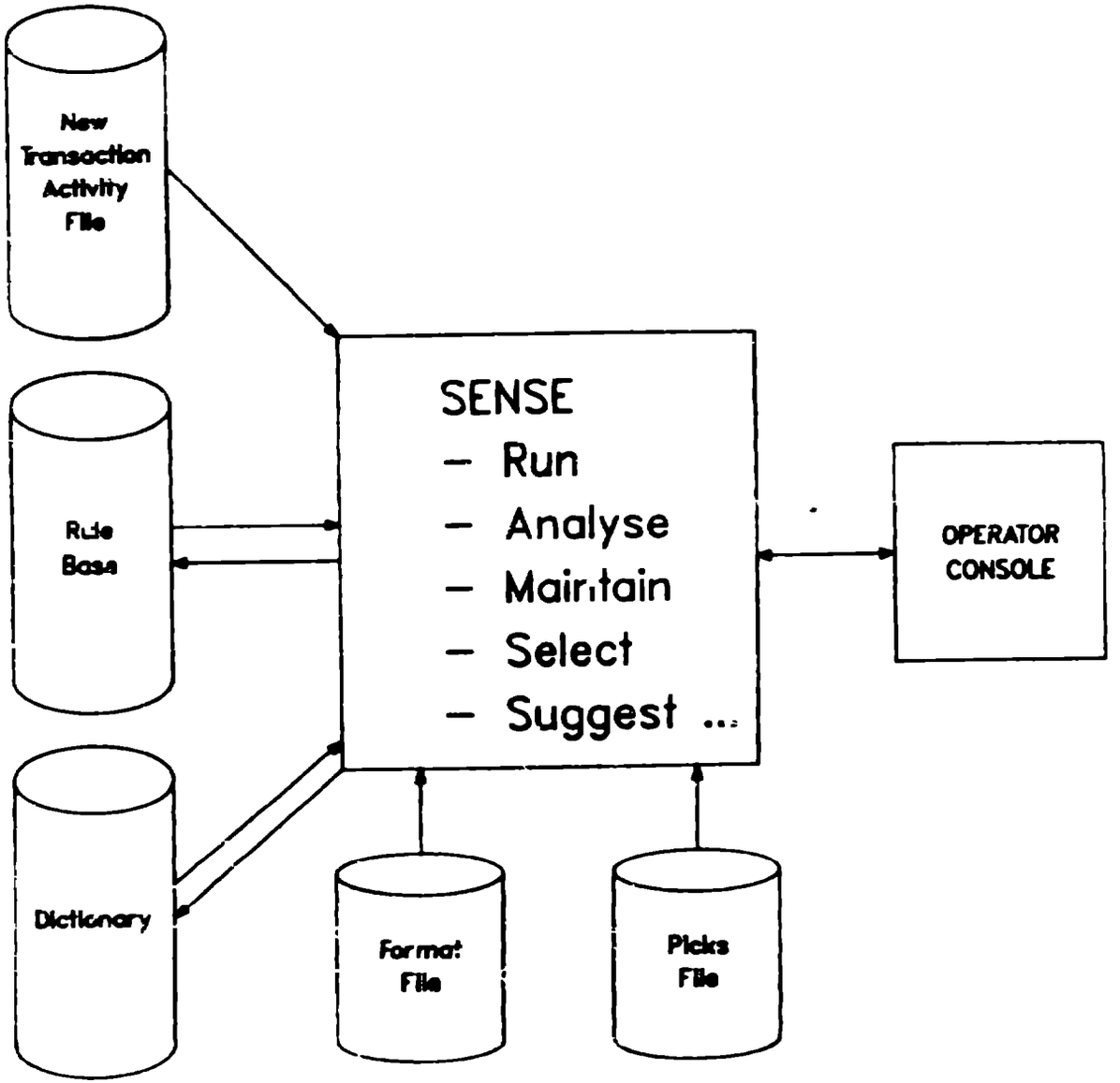

Fig. 6. Inferencing, analysis, and maintenance modules.

The frequency with which rule bases need to be replaced depends on the stability of the system being monitored. On multi-user computer systems, processing new transactions against rule bases more than a few weeks old tends to produce unacceptably high anomaly rates.

1. Scoring Transactions. As each transaction is processed by Sense, it computes a scoring function result for each field, for the transaction as a whole, and for any thread to which the transaction belongs. The score for each picked field is a function of the grades of each rule violated and each rule obeyed. A transaction score, or FOM, of 0 describes a transaction that is "perfectly normal" in every field.

A transaction fires a rule if the rule's LHS is satisfied by the data in the transaction fields. Thus, a rule of the form:

User Bob and Day Tue and Terminal OPA1 implies with grade 7
either Image BACKUP or INIT or AUTHORIZE

would fire only if the transaction was for Bob using the console OPA1 on Tuesday. If Bob executed an image other than BACKUP, INIT or AUTHORIZE, the rule would be failed.

2. Transaction FOM. The transaction FOM is the weighted sum of the FOMs for each picked field (negative FOMs are set to 0 before summing). The FOM for each element is approximately the sum of scores for failed rules minus the expected sum normalized by the square root of the variance of this sum. In calculating these moments, we ignore the depth adjustment to a rule's grade, and we assume that all rules are independent.

More precisely, for each field i, let:

N_i = number of fired rules,

K_i = number of failed rules, and

G_{ij} = the grade of the j^{th} rule fired on RHS field i.

S_{ij} = the score ($2^{G_{ij}}$) for the j^{th} rule fired on RHS elements,

S_i = the sum of scores (S_{ij}) for all fired rules, and

F_i = the sum of scores for failed rules.

Then FOM_i, the score for element i, is computed as:

$$FOM_i = \frac{F_i - E[F_i]}{\sqrt{\text{variance}[F_i]}} \quad .$$

The expected value and variance can be calculated from:

$$E[F_i] = \sum_{j=1}^{N_i} \left(\frac{1}{S_{ij}} \times S_{ij} \right) = N_i \quad \text{assuming independence}$$

and

$$\text{variance}[F_i] = E[F_i^2] - E^2[F_i] = E[F_i^2] - N_i^2$$

$$E[F_i^2] = E\left[\sum_{j=1}^{N_i} \frac{1}{S_{ij}} \times S_{ij}^2 \right] = \sum_{j=1}^{N_i} S_{ij} + (N_i - 1) \times N_i$$

and

$$\text{variance}[F_i] = S_i - N_i \quad .$$

Therefore

$$FOM_i = \frac{F_i - N_i}{\sqrt{(S_i - N_i)}} \quad .$$

The transaction FOM for a record with M picked fields is then given by:

$$TFOM = \sum_{i=0}^{M} \max(FOM_i, 0) \times W_i \quad ,$$

where W_i is the importance factor for the field i.

The thread FOM takes into account the scoring history of each transaction in the thread. FOMs for previous transactions in the same thread are decayed and then added to the current transaction FOM. The result after T transactions in the thread is computed as:

$$FOM_T = \sum_{t=0}^{T} TFOM_t \times d^{T-t} = TFOM_0 + TFOM_1 \times d \quad ,$$

where $TFOM_T$ is the FOM for the most recently observed transaction in the thread and d is some suitable constant between 0.0 and 1.0. With d near 0.0, only the current transaction carries significant weight. With d near 1.0, the thread FOM approaches the sum of all transaction FOMs for the thread processed so far.

3. **Anomaly Detection.** Sense finds an anomaly whenever either the transaction or thread FOM exceeds an operator-set limit. Transaction evaluation times are roughly proportional to the log of the number of rules. Thus a very large rule base need not be slow. W&S handles rule bases of up to 500,000 rules, averaging 7.0-9.0 bytes per rule and 20,000-40,000 rule firings per second. Typical transactions on these large rule bases have fired approximately 1% of the rules, resulting in measured performance ranging from 20-40 transactions per second for more typical (for W&S) rule bases of 100,000 instantiated rules.

4. **Real-Time Issues.** A serious impediment to real time detection, discussed by Denning,[3] is the difficulty of assembling session and machine activity data into usable transaction records. Existing operating

system accounting software simply does not make the job straightforward. As a minimum, the anomaly detection software will probably have to wait until the accounting software has written its data to disk. Highly buffered data may be written too late for real-time analysis. (For example, we experience buffering delays up to 9 minutes on our VAX running VMS 4.5.) Or, if data from separate accounting files (e.g., disk activity, user image executions, and keyboard input) must be matched and assembled, real-time may not be possible, or may be feasible only with a lower detection sensitivity (by treating each accounting data stream independently).

5. **Anomaly Resolution.** Anomaly resolution is the task of explaining the meaning and likely cause of an anomalous transaction. W&S attempts to provide information useful in this task; nonetheless, it is primarily one that must be accomplished by a human.

W&S currently offers four significant aids to anomaly resolution:

- Identification of the data in a transaction that appear to have triggered the anomaly;
- Listing of the violated rules that triggered the anomaly determination;
- Providing a thread history;
- Suggesting what data specific fields would have avoided the anomaly determination.

Each of these aids builds upon the inferencing process just described.

VI. CONTINUING EFFORTS

W&S is now undergoing operational tests in two computer security environments and one process monitoring environment. Preliminary results have shown that the software does periodically detect anomalies of high interest even in data thought to be free of such events. Thus far, we have tested only trivial intrusion scenarios (with successful detection). We hope to test the effectiveness of W&S on a wide variety of planted

anomalous events during the current year. Futhermore, several enhancements, such as hybrid rule bases consisting of user-defined rules inserted into the generated rule base, will require extensive evaluation.

Nonetheless, it is already clear that the anomaly detection approach in W&S is effective for a wide range of applications where large volumes of repetitive data are generated by some chemical, mechanical, electrical, or biological system and where anomalous events are of interest. The heuristics employed in W&S make a reasonable comprimise between computational accuracy and full use of available information, especially categorical and threaded data.

ACKNOWLEDGMENTS

I would like to thank the intrusion detection team at SRI International, especially Teresa Lunt and Hal Javits, for their encouragement and constructive comments on W&S. Gunar Liepins at Oak Ridge National Laboratory has generated the ideas for several important enhancements now in the W&S software. At Los Alamos National Laboratory, I am especially indebted to James Tape and Jack Markin for urging me to pursue this work and for their continuing support of our R&D efforts.

REFERENCES

1. Donald E. Knuth, The Art of Computer Programming, Volume 2: Seminumerical Algorithms (Addison-Wesley Publishing Company, Reading, Massachusetts, 1969).

2. Ronald A. Howard, "Decision Analysis: Perspectives on Inference, Decision, and Experimentation," Proceedings of the IEEE, Vol. 58, No. 5, 632-643 (May 1970).

3. D. E. Denning, "An Intrusion Detection Model", IEEE Transactions on Software Engineering, Vol. SE-13, No. 2 (February 1987).