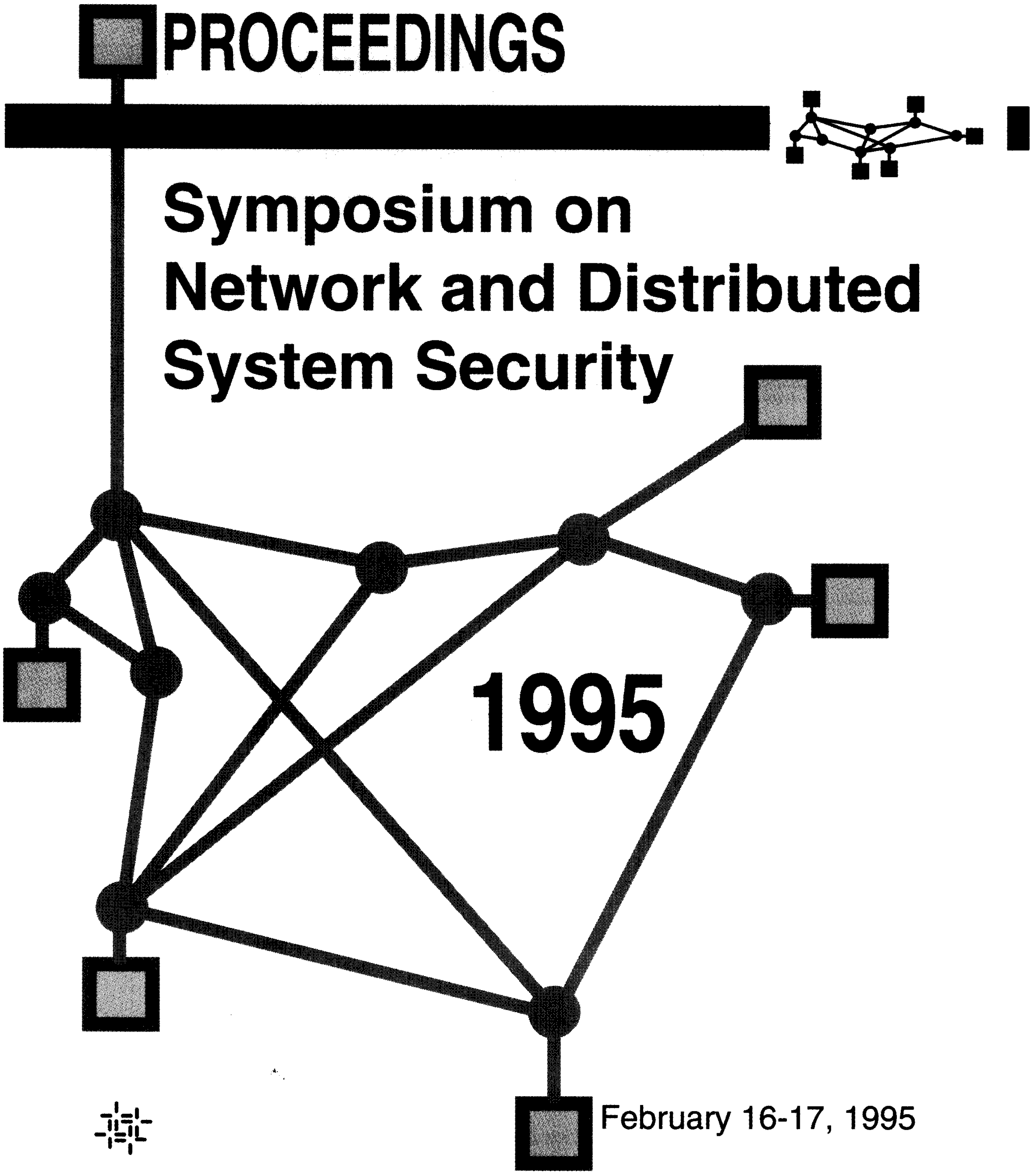
PROCEEDINGS

Symposium on Network and Distributed System Security



February 16-17, 1995

San Diego, California

IEEE Computer Society Press   The Institute of Electrical and Electronics Engineers, Inc.

Distributed Audit Trail Analysis

Abdelaziz Mounji Baudouin Le Charlier Denis Zampuniéris
Naji Habra,
Institut d'Informatique,
FUNDP,
rue Grangagnage 21,
5000 Namur
E-mail: {amo, ble, dza, nha}@info.fundp.ac.be

Abstract

An implemented system for on-line analysis of multiple distributed data streams is presented. The system is conceptually universal since it does not rely on any particular platform feature and uses format adaptors to translate data streams into its own standard format. The system is as powerful as possible (from a theoretical standpoint) but still efficient enough for on-line analysis thanks to its novel rule-based language (RUSSEL) which is specifically designed for efficient processing of sequential unstructured data streams.
In this paper, the generic concepts are applied to security audit trail analysis. The resulting system provides powerful network security monitoring and sophisticated tools for intrusion/anomaly detection. The rule-based and command languages are described as well as the distributed architecture and the implementation. Performance measurements are reported, showing the effectiveness of the approach.

1 Introduction

Auditing distributed environments is useful to understand the behavior of the software components. For instance this is useful for testing new applications: one execution trace can be analyzed to check the correctness *wrt* the requirements. In the area of real-time process control, critical hardware or software components are supervised by generating log data describing their behavior. The collection and analysis of these log files has often to be done real-time, in parallel with the audited process. This analysis can be conducted for various purposes such as investigation, recovery and prevention, production optimization, alarm and statistics reporting. In addition, correlation of results obtained at different nodes can be useful to achieve a more comprehensive view of the whole system.

Computer and network security is currently an active research area. The rising complexity of today networks leads to more elaborate patterns of attacks. Previous works for stand-alone computer security have established basic concepts and models [3, 4, 5, 7, 8] and described a few operational systems [1, 6, 9, 12, 18]. However, distributed analysis of audit trails for network security is needed because of the two following facts. First, the correlation of user actions taking place at different hosts could reveal a malicious behavior while the same actions may seem legitimate if considered at a single host level. Second, the monitoring of network security can potentially provide a more coherent and flexible enforcement of a given security policy. For instance, the security officer can set up a common security policy for all monitored hosts but choose to tighten the security measures for critical hosts such as firewalls [2] or for suspicious users.

A software architecture and a rule-based language for universal audit trail analysis were developed in the first phase of the ASAX project [10, 11, 12]. The distributed system presented here uses this rule-based language to filter audit data at each monitored host and to analyze filtered data gathered at a central host. The analysis language is exactly the same at both local and central levels. This provides a tool for a flexible and a gradual granularity control at different levels: users, hosts, subnets, domains, etc.

The rest of this paper is organized as follows. Section 2 briefly describes the system for single audit trail analysis and its rule-based language. Section 3 details the functionalities offered by the distributed system. Section 4 presents the distributed architecture. Section 5 describes the command interface of the security officer. In section 6, the implementation of the main components is outlined. Performance measurements are reported in section 7. Finally, section 8 contains the conclusion and indicates possible improvements of this work.

2 Single Audit Trail Analysis

In this section, the main features of the stand alone version of ASAX for single audit trail analysis are explained. However, we only emphasize interesting functionalities. The reader is referred to [12] for a more detailed description of these functionalities[1]. A comprehensive description of ASAX is presented in [10, 11].

[1]Notice however that [12] is a preliminary description of a system under implementation. The examples in the present paper have been actually run on the implemented system

0-8186-7027-4/95 $4.00 © 1995 IEEE

2.1 A motivating example

The use of the RUSSEL language for single audit trail analysis is better introduced by a typical example: detecting repeated failed login attempts from a single user during a specified time period. This example uses the SunOS 4.1 auditing mechanism. Native audit trails are translated into a standard format (called *NADF*). The translation can be applied on-line or off-line. Hence, the description below is based on the NADF format of the audit trail records.

Assuming that login events are pre-selected for auditing, every time a user attempts to log in, an audit record describing this event is written into the audit trail. Audit record fields (or audit data) found in a failed login record include the time stamp (*au_time*), the user id (*au_text_3*) and a field indicating success or failure of the attempted login (*au_text_4*). Notice that audit records representing login events are not necessarily consecutive since other audit records can be inserted for other events generated by other users of the system.

In the example (see Figure 1), RUSSEL keywords are noted in bold face characters, words in italic style identify fields in the current audit record, while rule parameters are noted in roman style. Two rules are needed to detect a sequence of failed logins. The first one (*failed_login*) detects the first occurrence of a login failure. If this record is found, this rule triggers off the rule *count_rule* which remains active until it detects count_down failed logins among the subsequent records or until its expiration time arrives. The parameter *target_uid* of rule *count_rule* is needed to count only failed logins that are issued by the same user (*target_uid*). If the current audit record does not correspond to a login attempt from the same user, *count_rule* simply retriggers itself for the next record otherwise. If the user id in the current record is the same as its argument and the time stamp is lower than the expiration argument, it retriggers itself for the next record after decrementing the count down argument. If the latter drops to zero, *count_rule* writes an alarm message to the screen indicating that a given user has performed *maxtimes* unsuccessful logins within the period of time *duration* seconds. In addition, *count_rule* retriggers the *failed_login* rule in order to search for other similar patterns in the rest of the audit trail.

In order to initialize the analysis process, the special rule **init_action** makes the *failed_login* rule active for the first record and also makes the *print_results* rule active at completion of the analysis. The latter rule is used to print results accumulated during the analysis such as the total number of detected sequences.

2.2 Salient features of ASAX

2.2.1 Universality

This feature means that ASAX is theoretically able to analyze arbitrary sequential files. This is achieved by translating the native file into a format called NADF (*Normalized Audit Data Format*). According to this format, a native record is abstracted to a sequence of audit data fields. All data fields are consid-

```
global v:  integer;

rule failed_login(max_times, duration:  integer);

if        event = 'login_logout'
  and au_text_4 = 'incorrect password'
  --> trigger off for_next
        count_rule(au_text_3,
                   strToInt(au_time)+duration,
                   max_times-1)
fi;

rule count_rule(target_uid:  string;
                expiration,
                count_down:  integer);

if         auid = suspect_auid
  and    event = 'login_logout'
  and au_text_4 = 'incorrect password'
  and au_text_3 = target_uid
  and strToInt(au_time) < expiration
  --> if count_down > 1
        --> trigger off for_next
                count_rule(target_uid,
                           expiration,
                           count_down-1);
        count_down = 1
        --> begin
              v := v + 1;
              println(gettime(au_time),
                      ':  3 FAILED LOGINS ON ',
                      target_uid);
              trigger off for_next
                failed_login(3,120)
            end
      fi;
  strToInt(au_time) > expiration
  --> trigger off for_next failed_login(3,120);
  true
  --> trigger off for_next
        count_rule(target_uid,
                   expiration,
                   count_down)
fi;

rule print_results;
begin
  println(v, ' sequence(s) of bad logins found')
end;

init_action;
begin
  v := 0;
  trigger off for_next failed_login(3, 120);
  trigger off at_completion print_results
end.
```

Figure 1: *RUSSEL module for failed login detection on SunOS 4.1*

ered as untyped strings of bytes. Therefore, an audit data in the native record is converted to three fields[2]:

an identifier (a 2-bytes integer) identifies the data field among all possible data fields;

a length (a 2-bytes integer;)

a value i.e., a string of bytes.

A native record is encoded in NADF format as the sequence of encodings of each data field with a leading 4-bytes integer representing the length of the whole NADF record. Note that the NADF format is similar to the TLV (*Tag, Length, Value*) encoding used for the BER (*Basic Encoding Rules*) which is used as part of the *Abstract Syntax Notation* ASN.1 [14]. However, the TLV encoding is more complex since it supports typed primitive data values such as boolean, real, etc as well as constructor data types. Nevertheless, any data value can be represented as a string of bytes in principle. As a result, the flexibility of the NADF format allows a straightforward translation of native files and a fast processing of NADF records by the universal evaluator.

2.2.2 The RUSSEL language

RUSSEL (RUle-baSed Sequence Evaluation Language) is a novel language specifically tailored to the problem of searching arbitrary patterns of records in sequential files. The built-in mechanism of rule triggering allows a single pass analysis of the sequential file *from left to right*.

The language provides common control structures such as conditional, repetitive, and compound actions. Primitive actions include assignment, external routine call and rule triggering. A RUSSEL program simply consists of a set of rule declarations which are made of a rule name, a list of formal parameters and local variables and an action part. RUSSEL also supports modules sharing global variables and exported rule declarations.
The operational semantics of RUSSEL can be sketched as follows:

- records are analyzed sequentially. The analysis of the current record consists in executing all active rules. The execution of an active rule may trigger off new rules, raise alarms, write report messages or alter global variables, etc;

- rule triggering is a special mechanism by which a rule is made active either for the current or the next record. In general, a rule is active for the current record because a prefix of a particular sequence of audit records has been detected. (The rest of this sequence has still to be possibly found in the rest of the file.) Actual parameters in the set of active rules represent knowledge about the already found subsequence and is useful for selecting further records in the sequence;

- when all the rules active for the current record have been executed, the next record is read and the rules triggered for it in the previous step are executed in turn;

- to initialize the process, a set of so-called *init* rules are made active for the first record.

User-defined and built-in C-routines can be called from a rule body. A simple and clearly specified interface with C allows to extend the RUSSEL language with any desirable feature. This includes simulation of complex data structures, sending an alarm message to the security officer, locking an account in case of outright security violation, etc.

2.2.3 Efficiency

Is a critical requirement for the analysis of large sequential files, especially when on-line monitoring is involved. RUSSEL is efficient thanks to its operational semantics which exhibits a bottom-up approach in constructing the searched record patterns. Furthermore, optimization issues are carefully addressed in the implementation of RUSSEL: for instance, the internal code generated by the compiler ensures a fast evaluation of boolean expressions and the current record is pre-processed before evaluation by all the current rules, in order to provide a direct access to its fields.

3 Administrator Minded Functionalities

3.1 Introduction

The previous sections showed that ASAX is a universal, powerful and efficient tool for analyzing sequential files, in general, and audit trails, in particular.
In this section, the functionalities of a distributed version of ASAX are presented in the context of distributed security monitoring of networked computers. The implemented system applies to a network of SUN workstations using the C2 security feature and uses PVM (*Parallel Virtual Machine*) [15] as message passing system. However, the architecture design makes no assumption about the communication protocol, the auditing mechanism or the operating system of the involved hosts.

3.2 Single point administration

In a network of computers and in the context of security auditing, it is desirable that the security officer has control of the whole system from a single machine. The distributed on-line system must be manageable from a central point where a global knowledge about the status of the monitoring system can be maintained and administered in a flexible fashion. Management of the monitoring system involves various tasks such as activation of distributed evaluators and auditing granularity control. Therefore, monitored nodes are, in a sense, considered as local objects on which administration tasks can be applied in a transparent way as if they were local to the central machine.

[2] In fact, native files can be translated to NADF format in many different ways depending on the problem at hand. The standard method proposed here was however sufficient for the applications we have encountered so far.

3.3 The local and global analyses

Local analysis requirement corresponds to the ability of analyzing any audit trail associated to a monitored host. This is achieved by applying an appropriate RUSSEL module to a given audit trail of a given host. The analysis is considered local in the sense that analyzed audit data represents events taking place at the involved host. No assumption is otherwise made about which host is actually performing the analysis. Local analysis is also called *filtering* since at the network level, it serves as a pre-selection of relevant events. In fact, pre-selected events may correspond to any complex patterns of subject behaviors.

Audit records filtered at various nodes are communicated to a central host where a global (network level) analysis takes place. In its most interesting use, global analysis aims at detecting patterns related to global network security status rather than host security status. In this regard, global analysis encompasses a higher level and a more elaborate notion of security event.

Concerted local and global analysis approach lends itself naturally to a hierarchical model of security events in which components of a pattern are detected at a lower level and a more aggregate pattern is derived at the second higher level and so on. Note that an aggregate pattern could exhibit a malicious security event while corresponding sub-patterns do not at all. For instance, a login failure by a user is not an outright security violation but the fact that this same user is trying to connect to an abnormally high number of hosts may indicate that a network attack is under course. Organizations often use networks of interconnected Lans corresponding to departments. The hierarchical model can be mapped on the organization hierarchy by applying a distributed analysis on each of the Lans and an organization-wide analysis carried out on audit data filtered at each Lan. Thus, concerted filtering and global analysis can lead to the detection of very complex patterns.

In the following, the node performing the global analysis is called the *central* or *master* machine while filtering takes place at *slave* machines. Correspondingly, we will also refer to master and slave evaluators. A *distributed evaluator* is a master evaluator together with its associated slave evaluators.

3.4 Availability

This requirement means that a distributed evaluator must survive any of its slave evaluators failure and must easily be recovered in case of a failure of the master evaluator. The availability of a distributed evaluator ensures that if for some reasons a given slave is lost (broken connection, fatal error in the slave code itself, node crash, etc), the distributed analysis can still be carried on the rest of monitored hosts. On the other hand, if the master evaluator fails, the distributed analysis can be resumed from an other available host. In all cases, and especially for on-line analysis, all generated audit records must remain available for analysis (no records are lost). Distributed analysis recovery must also be done in a flexible way and require a minimum effort.

3.5 Logging control

This functionality involves control of the granularity of security events at the network, host and user levels. Typically, the security officer must be able to set up a standard granularity for most audited hosts and to require a finer granularity for a particular user or all users of a particular host. According to the single point administration requirement, this also means that logging control is carried out from the central machine without need for multiple logging to remote hosts.

4 Architecture

The architecture of the distributed system is addressed at two different levels. At the host level, a number of processes cooperate to achieve logging control and filtering. The global architecture supports the network level analysis. This section aims at giving an intuitive view of the overall distributed system.

4.1 Host level

Processes in the local architecture are involved in the generation of audit data, control of its granularity level, conversion of audit data to NADF format, analysis of audit records and finally transmission of filtered sequences to the central evaluator. At the master host, a network level analysis subsequently takes place on the stream of records resulting from merging records incoming from slave machines. Both global and local analyses are performed by a slightly modified version of the analysis tool outlined in the previous section.

4.1.1 Audit trail generation

This mechanism is operating system dependent. It generates audit records representing events such as operations on files, administrative actions, etc. It is assumed that all monitored hosts provide auditing capabilities and mechanism for controlling granularity level. The process generating audit records is called the *audit daemon* (*auditd* for short).

4.1.2 Login controller

This process communicates with *auditd* in order to alter the granularity. It is able to change the set of preselected events. This can be done on a user, host and network basis. Furthermore, we distinguish between a temporary change which applies to the current login session and a permanent change affecting also all subsequent sessions.

4.1.3 Format adaptor

This process translates audit trails generated by *auditd* to the NADF format. Native files can be erased after being converted since they are semantically redundant with NADF files. Keeping converted files instead of native files has several advantages: the files are converted only once and can be reanalyzed several times without requiring a new conversion. Moreover, in the context of an heterogeneous network, they provide a standard and unique format.

4.1.4 Local evaluator

It analyzes the NADF files generated by the format adaptor. Note that several instances of the evaluator can be active at the same time to perform analyses on different NADF files or possibly on the same file. Off-line and on-line analyses are implemented in the same

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | sales@docketalarm.com | 1-866-77-FASTCASE

fastcase®
Smarter legal research.