# Fraud Detection in ERP Systems using Scenario Matching

Asadul K. Islam[1], Malcom Corney[1], George Mohay[1], Andrew Clark[1], Shane Bracher[2], Tobias Raub[2] and Ulrich Flegel[2]

[1]Queensland University of Technology, Australia
{a.islam, m.corney, g.mohay, a.clark}@qut.edu.au
[2]SAP Research
{shane.bracher, t.raub, ulrich.flegel}@sap.com

**Abstract.** ERP systems generally implement controls to prevent certain common kinds of fraud. In addition however, there is an imperative need for detection of more sophisticated patterns of fraudulent activity as evidenced by the legal requirement for company audits and the common incidence of fraud. This paper describes the design and implementation of a framework for detecting patterns of fraudulent activity in ERP systems. We include the description of six fraud scenarios and the process of specifying and detecting the occurrence of those scenarios in ERP user log data using the prototype software which we have developed. The test results for detecting these scenarios in log data have been verified and confirm the success of our approach which can be generalized to ERP systems in general.

**Keywords.** Fraud Detection, Enterprise Resource Planning, ERP Systems, Signature Matching

## 1 Introduction

Enterprise resource planning (ERP) systems provide complete automation of the business processes of an organization. Users within the organization operate the system to carry out day-to-day transactions such as financial, HR, and inventory transactions. Although ERP systems invariably provide controls, such as Segregation of Duties (SoD) to prevent different kinds of fraud, these are insufficient, and it is necessary also to deploy detection mechanisms. The two major reasons for this are, firstly, that organizations may not switch on the controls, and secondly constant monitoring is required for occurrences of fraudulent activities for which controls are not provided. Indeed, in the case of SMEs (Small and Medium Enterprises), this is in many cases a deliberate decision since the size of the enterprise makes it impossible to assign different conflicting duties to different people. While auditors examine transaction log files of ERP systems for the detection of any suspicious user activities, the fact is that ERP systems generate millions of transaction records making it impossible to guarantee the detection of all suspicious behavior, and certainly not in a timely manner.

A fraud scenario can be defined as a set of user activities that indicates the possible occurrence of fraud. The objective of the research work described in this paper is to design and develop a framework that enables us to define fraud scenarios and then to test for the existence of such scenarios in ERP system transaction logs. We describe the fraud scenario specification language and the detection framework and its evaluation. Our prototype system provides a user-friendly tool for defining fraud scenarios and searching for those scenarios. Although the design is applicable to transaction log files generated from any ERP systems, throughout our research we use transaction log files from SAP systems.

## 2. Related Work

In the field of Intrusion detection, one of the two fundamental strategies used to detect malicious computer activity is misuse detection, also known as signature matching. Misuse detection searches for patterns of computer activity ('signatures') in network and/or audit data which are known to signify security violations. The signatures in an intrusion detection system (IDS) are similar in concept to what we are structuring here - fraud scenario specifications. Our goal is to search for known fraud scenarios in the transaction logs of ERP systems. This is in contrast to statistical analysis techniques, techniques such as regression analysis, correlation analysis, dispersion analysis and use of Benford's Law to mention a few, which identify irregular or statistically inconsistent data ([1], [2]).

Some notable works for structuring signatures for IDS are: [3,4,5,6]. We note also the description of a semantic model for signatures or event patterns [7]. Most of the proposed signature specification languages try to make the language as comprehensive as possible. From careful examination, it can be observed that every language has been built with a specific problem domain in mind. Among the popular signature languages, STAT [3], USTAT [8], and NetSTAT [9] are based on a state transition model whereas EMERALD [10], CAML [11], and CARDS [12] are based on an event-based model. Both types of model have been commonly used for proposals to represent and detect multi-step attacks. Most have not been implemented in a form which makes them generally useful. Even when implemented, such languages are typically under-specified and difficult to use, and have been tested with test scripts and experimental programs for experimental research outcomes only.

The very few, including the improved STATL [13] and CAML [11], for which the authors claim language completeness, are complicated to work with and not generalizable so as to be applicable to a different domain e.g. fraud detection. Specifically, the enhanced version of STATL [13] adds flexibility to the original STATL design but, at the same time, adds more complexity for writing signatures. Signature writers must have an intimate understanding of system states and transitions, and carefully choose appropriate system transitions when specifying the components of a signature. A similar situation prevails for the few other languages which might at first sight be considered to be candidates for generalizing to the domain of fraud detection. In addition, language specific compilers are required in each case and these are again not readily available.

More importantly, while there are similarities between fraud scenario detection and intrusion signature detection, there are also important differences. Fraud activity scenarios are concerned with the high-level interactions of a user with financial data rather than with the low-level events or states of the computer system itself. There is a corresponding greater degree of system independence in the case of fraud detection, and this can in turn be exploited by more easily separating out abstract or semantic aspects of fraud signatures from configuration aspects such as, for example, event timeout thresholds. For instance, stateful network IDS signatures may take into account system aspects such as network bandwidth or hardware sensor type, but such basic parameters have no part to play in the case of fraud detection.

One of our objectives is to give the auditor a simple, user-friendly interface and tool for writing new fraud scenarios as well as changing existing ones. Auditors can select the scenario to search for in the log, and if not satisfied with the result they can change the scenario specification and criteria to examine more specific results. This highlights another difference from signature detection in IDS: our fraud detection system is able to exploit the benefits of post-hoc (non-realtime) investigation.

As a result of the above, we have developed a fraud scenario specification language specific to our requirements for fraud detection which provides the necessary features.

## 3 Signature Language, Scenarios and Scenario Detection

In this section we describe how our signature specification language relates to work in intrusion detection, provide examples of some of the fraud scenarios used to test and evaluate both the language and the scenario detection mechanism we have developed, and describe some of the detailed semantics of our language.

Our language generally follows the criteria for attack signature languages described in [7] and in [14, 15, 16], but it is specialized for fraud detection. The following section provides details of the language and compares the structure of the proposed scenario specification language with the signature requirements described by Meier [7]. The requirements are described in [17]. Note that in terms of the analogy between IDS misuse languages and fraud scenario languages, the term event in IDS corresponds to the term activity, transaction or component (we use these terms interchangeably) in fraud scenarios.

In Meier's paper, the author describes a list of semantic requirements for modeling misuse scenarios. Our requirements to model fraud scenarios are not exactly the same as needed for misuse signatures, but similar in concept. Meier defines the requirements for ad-hoc investigation. However, in our case we consider the post-hoc investigation on transaction log data. We compare the semantics of our fraud scenario language with Meier's requirements for modeling misuse scenarios. Meier's work [7] adopts Zimmer's model [18]. By comparing with that model, the requirements of "sequence" and "disjunction" have been fulfilled by the "Ordered" attribute of our proposed signature specification. Simultaneous activities imply components with identical occurrence time. The proposed language supports "simultaneous" activities. When two activities happen at the same time, the process considers them in random order. The process examines one transaction log data file, but it is possible to generate

the log data from multiple sources. In that case, time stamp is used to maintain the "simultaneous" feature.

The "continuity" semantics of a misuse model define, whether between three consecutive steps of the event types *A*, *B*, and *C* an event *c* is allowed to occur between events *a* and *b*, and if event *a* may occur between *b* and *c*. This semantics is supported in the proposed scenario language and specified using the "Ordered" attribute. Also "repetition", "context conditions", "step instance selection" and "step instance consumption" are well supported in our scenario language.

### 3.1 Composition and Fraud Scenarios – An Example

In our model, "composition" is considered as a basic building block and theoretically an unlimited level of composition is allowed. Composition is a fundamental requirement, of the language, a static feature which is needed in order to easily represent the hierarchical nature of activity. For example, the scenario 'Redirected Payment' or 'S01' described in Table 1 and Fig. 1 uses two scenarios 'Change_Vendor_Bank' and 'Pay_Vendor' with a few additional parameters to specify the 'sequence' requirement and 'interval' requirements. In this scenario, 'Change_Vendor_Bank' and 'Pay_Vendor', are the components of the scenario 'S01'.

**Table 1.** Sample Scenario 'S01': Redirected Payment

| |
|---|
| **Description.** The intention of this fraudulent behavior is to make a payment in such a way so that the payment goes not to the vendor's actual account but to a re-directed account. The scenario involves making payments to a vendor account after changing the bank account details of the vendor to a different account, and then after payment changing the bank details back.<br>**Details.** For 'changing of vendor's bank details' we found three transaction codes ('FK02, 'FI01', 'FI02') are used for this activity. The scenario, which we call 'Change_Vendor_Bank', will be the existence of any of these transaction codes in the log. Similarly, another scenario, 'make payments to vendor' is the existence of any of the four transaction codes ('F-40', 'F-44', 'F-48', 'F-53') found in the transaction log, and we call this 'Pay_Vendor'. |

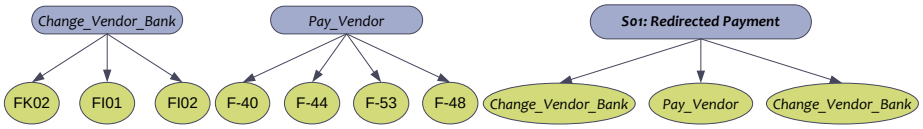


**Fig. 1.** Structure of the fraud scenario 'S01'

### 3.2 Scenario Specification and Semantics

We use XML to specify and store scenario definitions in a file referred to as the "*Scenario Definition File*" (SDF). XML is used because it is a simple text format file which supports hierarchical structure for storing complex data and there are built-in engines/libraries for XML in most of the popular development tools.

Our fraud scenario specification language takes many ideas from [10]. Broadly, the structure of a fraud scenario consists of: Name and Description, list of Components, Attributes and Scenario rules. Name is unique for each scenario. A scenario used as a component in another scenario is referred to by its name.

Each scenario consists of a collection of components. A component is a transaction (extracted from a transaction log) or any already defined scenario. Each component has component level attributes such as “Order” to maintain sequence of components and “Name” to specify the name of the scenario. There are other optional attributes such as “Timeout” to specify the time the component will be active to take part in the scenario, “Required” to make the component mandatory or optional, and “TimeMin” to specify the minimum time required for the component to be active.

Scenario attributes hold the values required to specify other aspects of the scenario which defines the behavior and characteristics of the scenario. The minimum or maximum intervals allowed between components are defined in three levels: default, scenario and component level. Component level attribute values override default and scenario level attribute values. Default values are used when there is no value specified in the definition. The scenario level interval value applies between all components and the component level value applies between two specific components. “Duration” applies as a condition for maximum time for the whole scenario to complete. Intervals and durations have an optional attribute “Tolerance” which can be used for flexibility. “Required_Components” is the minimum number of components required from the component collection to match the activity pattern of the fraud scenario. “Ordered” is used to specify whether the components should be present sequentially or not. “Maximum Repetition” is used to specify the maximum number of repetitions allowed and “Minimum Repetition” is to specify the minimum number of repetitions required within the time specified in “Duration”. The repetition is not for individual components rather it is for the pattern.

Additional conditions may be necessary for defining a scenario. For example, a scenario may need to be defined as the event sequences performed by the same user. In that case, a “same user” condition can be added to the definition. These types of condition are defined using “Match” in the scenario structure. If there are multiple scenario rules present, optional match operators are used to define which operator, “AND” or “OR”, will be used between them.

### 3.3 Fraud Scenarios

We have described scenario ‘S01’ in Section 3.1 and we now describe other fraud scenarios. This section describes scenarios ‘S02’ and ‘S03’ in the same format used for scenario ‘S01’. We do not describe scenarios ‘S04’, ‘S05’ and ‘S06’ in the same detail because scenarios ‘S01’, ‘S02’ and ‘S03’ cover all the features of the model. It should be noted that occurrences of these scenarios do not always confirm fraud; rather they identify the possible occurrence of fraud and raise a red flag.

**Sample Scenario ‘S02’.** *False Invoice Payment* (Invoices created, approved and payment; any two of these activities performed by the same user).

*Description.* The intention of this fraud is to make a false payment for one's own benefit. This can happen when there is lack of segregation of duties. In this scenario, we specify three activities; create an invoice, approve that invoice and make the payment. If any two activities have been done by the same user on a specific invoice, we consider that a possible fraud.

*Details.* We build the 'Create_Invoice' scenario by defining the presence of either of two transaction codes, 'FB60' or ''MIRO', in the transaction log which are indicative of the 'create invoice' activity. The second activity 'approve invoice', is indicated by transaction code 'MRBR' and we will use this transaction code without building a separate scenario for it. For the third step 'make payment', we have already defined the scenario 'Pay_Vendor' as part of scenario 'S01'.

**Sample Scenario 'S03'.** *Misappropriation* (Purchase order and purchase approval by the same user)

*Description.* This fraud scenario also arises because the failure to implement segregation of duties correctly. The fraud is the result of misappropriation of company capital. When a user is allowed to make purchase orders and also has the privileges to approve them, there is a possibility of making the purchase of private possessions.

*Details.* For the first activity we found five transaction codes ('ME21N', 'ME25', 'ME58', 'ME59N', 'ME22N') all referring to the same activity. We build a scenario 'Create_PO' which is the existence of any one of these transaction codes in the log. For the second activity we found two transaction codes ('ME29N', 'ME28') and we build a scenario 'PO_Approval' for this activity. The fraud scenario 'S03' is the sequence of these two scenarios on the same purchase order by the same user.

**Sample Scenario 'S04'.** *Non-purchase Payment* (Purchase order or good received; any one of these activities and creating invoice by the same user)

**Sample Scenario 'S05'.** *Anonymous Vendor Payment* (Create or modify vendor master and create invoices to that vendor by the same user)

**Sample Scenario 'S06'.** *Anonymous Customer Payment* (Create or modify customer records and grant credit to that customer by the same user)

### 3.4 Scenario Detection

In summary, the scenario detection process finds the definition of known fraud scenarios from the Scenario Definition File and searches for the existence of this scenario in the transaction log of the ERP system. To make the detection process independent and applicable to the different transaction log structures of different ERP systems, before feeding the log into the process a tool named "Anonymizer tool" is used to convert the data into a pre-defined format and to anonymize sensitive information with a consistent one-way hash function to maintain privacy requirements. The process imports the formatted data to a database against which it can run SQL queries. We use the MySQL database for this purpose. Fig. 2 shows the workflow of the detection process.

The Scenario Detection process has five distinctive tools. "*Scenario tools*" extracts the scenario definition from the SD file, takes the default values and other configuration information from "*Configuration tools*", and generates output as a "*Scenario Definition Object*". "*SQL generator*" generates the SQL query from the "*Scenario Definition Object*". "*Query resultset*" tool sends the SQL query to the database and gets the result. "*Configuration tools*" is for communicating with the configuration files which store application related information, database connection parameters and scenario default values.

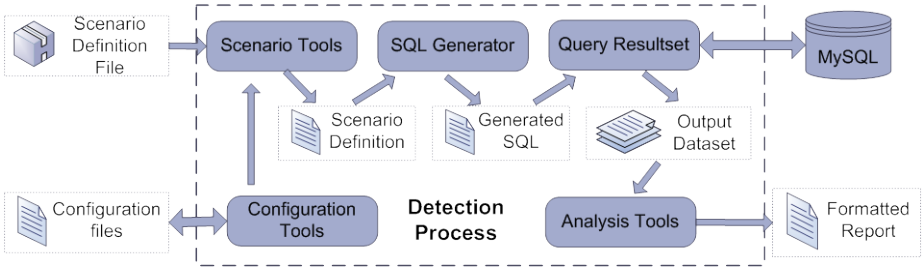


**Fig. 2.** Workflow of the detection process

The result-set contains a set of matched scenario patterns from the transaction log. Each set or row contains the identification of each component in the search pattern. From that identification, it is possible to locate them in the actual transaction log. The "Analysis Tool" acts as a reporting service and is used to generate summary reports from the result-set.

## 4 Implementation

The fraud detection process includes generation of fraud scenario definitions and the detection of fraudulent activities by matching the transactions recorded in transaction logs. Our prototype software allows users to generate and edit scenarios, and to parse, anonymize and import transaction log data into a database. The software generates SQL queries from scenario definitions and then sends the query to the database. In this section, we describe the process of extracting transaction log data from an SAP system and the process of scenario detection using the prototype software on that data.

### 4.1 Data Preparation

We extract application log data from SAP system logs. There are some common fields extracted for every activity, e.g. date and time, user etc. as well as some fields extracted only for specific activities, e.g., for the 'change vendor bank details' activity we extract the vendor identification number or VendorID. From the various log data,

we build the combined log table, described in Table 2, which holds data for all activities. The fieldset of the table described is the combination of fields required for all scenarios and the RowIdentity field, which holds a serial number to uniquely identify each record. We cannot use the timestamp values for this purpose as multiple transaction records may exist in the log for one user activity meaning it is possible to have multiple records with the same timestamp value.

**Table 2.** Fieldset of final transaction log table

| | Field Name | Description |
|---|---|---|
| 1 | RowIdentity | Row identification number. |
| 2 | DateTime | The timestamp value when the activity occurred. |
| 3 | Transaction Code | This is code identifies user activity. There could be multiple codes for a single activity. |
| 4 | User | User name or identification who performed the activity. |
| 5 | Terminal | Terminal identification from which the user performed the activity. |
| 6 | VendorID | Vendor identification number. When activity related to Vendor. |
| 7 | InvoiceNo | Invoice number when the activity is related to any payment. |
| 8 | PRNumber | Purchase requisition number for purchasing activity. |
| 9 | PONumber | Purchase order number for purchasing activity. |
| 10 | CustomerID | Customer identification for customer related activity. |

### 4.2 Data Sources

The information in Table 2 is readily obtained and we illustrate this below with examples.

In SAP systems payment information is stored in the table BKPF: Accounting Document Header and BSEG: Accounting Document Segment. BSEG has the corresponding Accounting document number as in BKPF, and Account number of the vendor. From these two tables we can extract the transaction data necessary for the activity described above as ‘payment to vendor’. The Security Audit log in SAP systems logs each user’s activity. By matching user name, transaction code and time with activities in the BKPF table and Security Audit log it is possible to extract the terminal information. Fig. 3 describes the transaction data extracted from these two tables for vendor payment activity.

In SAP systems, all vendor master record information is stored in three tables: LFA1, LFB1 and LFBK. The change logs for the master records are stored in two tables – CDHDR: Change Document Headers and CDPOS: Change Document Items. These tables record the changed values and the user name of the person responsible for changing the document, along with the date and time of the change and the transaction code used to make the change. The table LFBK holds banking details for vendors so using CDPOS and CDHDR we can extract transactions made when vendor bank details are changed. Fig. 3 explains the process of extracting the activity ‘change vendor bank details’ from these two tables.

# 5 Testing

The fraud detection system - architecture and software - that we have designed and developed comprises several different components as illustrated in Fig. 2. To verify that our software framework generated the correct SQL queries, we hand coded SQL queries for all the scenarios described previously and matched them against those generated by the software.

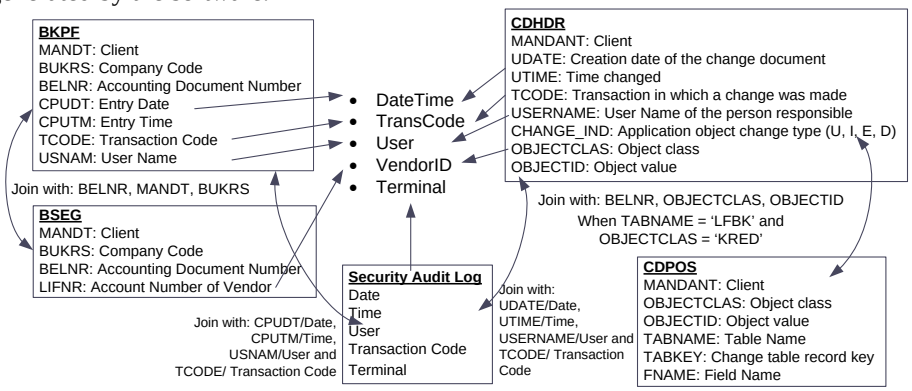


**Fig.3.** Data extraction for activities from SAP System

The prototype system runs the SQL statements generated by our software on the combined log table described earlier. Unfortunately it has been difficult to obtain real data for reasons to do with confidentiality and privacy, although we are currently discussing such possibilities with one large international company. For some of our testing we have used data from QUT's student SAP system which is implemented for teaching purposes but the data in this system while generated through real activities, is somewhat limited. As a result, we have adopted an approach utilizing synthetically generated data for our evaluation. This is described below.

## 5.1 Generating Test Data

One possible approach to the use of synthetically generated test data for evaluation is to generate clean data containing no fraudulent activity and then to insert fraudulent activity. We have adopted a slight variation of this approach which to some extent deflects the criticism that it is naturally easy to detect what has been deliberately inserted for the purpose of detection. Our approach is based on generating transactions at random with random timestamps, within some specified window of time, and storing the generated transactions in the combined log table. Lundin et al. [19] describe an interesting technique for generating synthetic data based on authentic background data merged with data representing fraudulent activity. . Generating synthetic data in this way is complex and we believe provides no advantage in our context.

If the essential individual components of the scenario are not present in the generated data then no match will be found. Likewise if they are present but in the wrong temporal order or if they don't meet the timeout conditions. If however, the individual components are present in the correct order and their timestamps meet timeout conditions, then a match should be found. The generator tool uses unique user names and unique terminal names and selects from these lists at random. The value of several of the fields in a transaction record depends on the transaction code and this is accommodated by grouping related transaction codes and having a separate list of values for the special fields of each group. When a transaction code is selected from the list, values for the special fields are also generated depending on which group it is in. For example, if the transaction code is in Group 1 (creating vendor records), it generates a new value for the VendorID field and stores it in the VendorID list.

For implementation and testing purposes we generated 100,000 records using a list of 100 users and 100 terminals. Before running the synthetic data generator the process was provided with a list of VendorIDs, InvoiceNos, Purchase Order Numbers and CustomerIDs. This data was then uploaded to the MySQL database.

### 5.2 Scenario Testing

We have used the above synthetic data and three of the previously described scenarios to evaluate our system. These three scenarios contain all the features in our fraud scenario detection model. We have in each case examined the output result-set of the SQL queries and verified that the output is correct. We have in addition, as a secondary check in each case, added and deleted records, and then re-run the scenario detection. This enabled us to verify that adding relevant transactions has the expected effect of producing a match where none existed previously, and that deleting records has the effect of producing no match where one did exist previously.

We ran all user activities used in the described fraud scenarios. Table 3 contains the number of activities or scenarios found in the randomly generated data.

**Table 3.** Number of different activities in log

| Scenario Name | Match returns | Scenario Name | Match returns |
|---|---|---|---|
| Change_Vendor_Bank | 2867 | PO_Approval | 6561 |
| Pay_Vendor | 12347 | Good_Receipt | 19775 |
| Create_Invoice | 6721 | Create_Vendor | 20567 |
| Approve_Invoice | 3215 | Create_Customer | 6558 |
| Create_PO | 15507 | Credit_to_customer | 5882 |

The scenario 'Change_Vendor_Bank' returned 2867 matching records and the scenario 'Pay_Vendor' returned 12347 matching records. We ran the scenario 'S01' to determine if any sequence of 'Change_Vendor_Bank', 'Pay_Vendor' and 'Change_Vendor_Bank' happened for the same Vendor when the maximum interval between any two activities was 2 days and the overall duration was within 3 days. Extra conditions were users or terminal must be the same. This process returned 20

matching scenarios which took 119 seconds to run on a Pentium 4 machine with 2GB RAM running Microsoft Windows XP Professional. The first two matches are shown in Table 4 and Table 5. The process returned one record for each match, but for clarity we show one match displayed across three rows. To narrow the result further, we changed the maximum interval between activities to 1 day from 2 days. With this changed scenario specification, the process returned 6 matches and took 119 seconds.

**Table 4.** One matching result of Scenario 'S01' where Terminal is same

| RowIdentity | DateTime | TransCode | User | Terminal | VendorID |
|---|---|---|---|---|---|
| 910 | 2007-02-01 05:33:07 | FK02 | USR013 | TRM43 | VID00004 |
| 4197 | 2007-02-02 01:07:38 | F-40 | USR030 | TRM43 | VID00004 |
| 5664 | 2007-02-02 07:09:45 | FK02 | USR071 | TRM43 | VID00004 |

**Table 5.** One matching result of Scenario 'S01' where User is same

| RowIdentity | DateTime | TransCode | User | Terminal | VendorID |
|---|---|---|---|---|---|
| 2722 | 2007-02-01 05:3:07 | FK02 | USR013 | TRM43 | VID000017 |
| 9105 | 2007-02-03 03:38:32 | F-48 | USR013 | TRM23 | VID000017 |
| 12313 | 2007-02-03 04:46:23 | FK02 | USR013 | TRM18 | VID000017 |

We ran scenario 'S02' requiring matches for the same User, the same Terminal and the same Invoice. The process returned 702 matches in 91 seconds. To narrow the result we changed the overall duration of the scenario to 1 day. This gave us 80 matches.

We ran 'S03' with a requirement that the same User and same Purchase Order Number were detected. It returned 468 records in 26 seconds. We changed the scenario by adding a requirement for the Terminal to match and it returned 4 matches.

We also tested the other fraud scenarios, checked the results and verified the correctness of the detection process.

## 6 Conclusion

A fraud scenario definition structure has been created for the purpose of defining fraudulent activity in ERP systems. The semantics of the structure have been developed and tested for common fraud scenarios in ERP systems. The scenario definition specification considers only the required fields for the described scenarios. We have described the process of defining common fraud scenarios, the process of extracting transaction log data from SAP systems, and the process of detecting scenarios from the transaction logs. Using the developed prototype software we successfully tested the detection of all scenarios described in this paper on synthetically generated transaction log data. We are hopeful of getting the opportunity to test the fraud scenario detection prototype on real SAP system data.

**Acknowledgment.** We gratefully acknowledge the support of SAP Research. The research is supported in part by an ARC Linkage grant.

## References

1. G. Mohay, A. Anderson, B. Collie, O. De Vel, R. McKemmish. Computer and Intrusion Forensics, Artech House (2003)
2. Coderre, D. G., Fraud Detection: Using Data Analysis Techniques to Detect Fraud. Global Audit Publications, Canada (1999)
3. Porras, P.A., Kemmerer, R.A.: Penetration State Transition Analysis: A Rule-Based Intrusion Detection Approach. In: Computer Security Applications Conference (1992)
4. Michel, C., Mé, L.: ADeLe: an Attack Description Language for Knowledge-Based Intrusion Detection. In: ICIS, pp. 353-368. Kluwer (2001).
5. Cuppens, F., Ortalo, R.: LAMBDA: A Language to Model a Database for Detection of Attacks. In: RAID, LNCS vol. 1907, pp. 197-216. Springer, Berlin/Heidelberg (2000)
6. Pouzol, J., Ducasé, M.: From Declarative Signatures to Misuse IDS. In: RAID, LNCS vol. 2212, pp. 1-21. Springer, Berlin/Heidelberg (2001)
7. Meier, M.: A Model for the Semantics of Attack Signatures in Misuse Detection Systems. In: 7th Information Security Conference, LNCS vol. 3225, pp. 158-169. Springer, Berlin/Heidelberg (2004)
8. Ilgun, K.: USTAT: A Real-time Intrusion Detection System for UNIX. In: IEEE Symposium on Security and Privacy, p. 16. IEEE Computer Society, Washington (1993)
9. Vigna, G., Kemmerer, R.A.: NetSTAT: A Network-Based Intrusion Detection Approach. In: 14th ACSAC, p. 25. IEEE Computer Society, Washington (1998)
10. Porras, P.A., Neumann, P.G.: EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. In: National Information Systems Security Conference, pp. 353-365. NIST/National Computer Security Center (1997)
11. Cheung, S., Lindqvist, U., Fong, M.W.: Modeling Multistep Cyber Attacks for Scenario Recognition. In: DARPA Information Survivability Conference and Exposition (DISCEX III), pp. 284-292. (2003)
12. Yang, J., Ning, P., Wang, X.S., Jajodia, S.: CARDS: A Distributed System for Detecting Coordinated Attacks. In: IFIP TC11 16th Annual Working Conference on Information Security, pp. 171--180. (2000)
13. Eckmann, S.T., Vigna, G., Kemmerer, R.A.: STATL: An Attack language for State-based Intrusion Detection. In: ACM Workshop on Intrusion Detection Systems. (2000)
14. Meier, M., Schmerl S., Koenig, H.: Improving the Efficiency of Misuse Detection. In: DIMVA. LNCS vol. 3548, pp. 188-205. Springer, Berlin/Heidelberg (2005)
15. Schmerl, S., Koenig, H., Flegel, U., Meier, M.: Simplifying Signature Engineering by Reuse. In: Emerging Trends in Information and Communication Security, LNCS vol. 3995, pp. 436-450. Springer, Berlin/Heidelberg (2006)
16. Abbott, J., Bell, J., Clark, A., de Vel, O., Mohay, G.: Automated Recognition of Event Scenarios for Digital Forensics. In: ACM Symposium on Applied Computing, pp. 293-300. ACM, New York (2006)
17. Flegel, U.: Privacy-Respecting Intrusion Detection. In: Advances in Information Security, Vol. 35, p. 307, Springer, Berlin/Heidelberg (2007)
18. Zimmer, D.: A Meta-Model for the Definition of the Semantics of Complex Events in Active Database Management Systems. PhD Thesis, University of Paderborn. (1998)
19. E. Lundin, H. Kvarnstrom, and E. Jonsson. In: Lecture Notes in Computer Science, ICICS 2002, Laboratories for Information Technology, Singapore. Springer (2002)