Proc. of the IFIP WG11.3 Workshop on Database Security,
Halifax, U.K., September 18-21, 1990.

SEPARATION OF DUTIES IN COMPUTERIZED INFORMATION SYSTEMS

Ravi Sandhu*

Department of Information Systems and Systems Engineering
George Mason University
Fairfax, Virginia 22030-4444, USA

Abstract
We describe a novel general-purpose mechanism for enforcing separation of duties in computerized information systems. This mechanism of transaction control expressions has close similarities to traditional controls in manual paper-based systems. It has the great benefit of intuitive simplicity, in both concept and implementation.

1 INTRODUCTION

Prevention of fraud and errors is by far the dominant security issue in commercial organizations (see [3, 9, 14, 18] for instance). Separation of duties is a time honored and universally practised principle for this purpose. That is, no individual is given sufficient authority within the system to perpetrate fraud on his own. This is achieved by breaking larger actions into smaller steps executed by distinct individuals. For example, a check is issued by preparing a voucher, having it approved and only then writing the check.

Separation of duties has certainly been applied in computerized information systems (see [22] for instance), perhaps even routinely so. Yet current database management systems and operating systems provide very little to support it. For every application the requisite facilities must therefore be invented and built from scratch. This is unproductive to say the least.

Our main objective in this paper is to describe a novel general-purpose mechanism called *transaction control expressions* for the purpose of enforcing separation of duties in computerized information systems. The mechanism is very natural and intuitive, being close in spirit and letter to controls typically encountered in paper-based systems. We truly seek to reflect the paper world of forms and books in the electronic media of databases. Transaction control expressions were originally proposed by this author in [16]. Here we refine and elaborate on the original description as well as identify further research

*The author is also affiliated with the Center for Secure Information Systems at George Mason University.

directions. As explained in [16], the conception of transaction control expressions was influenced by the ideas in [8, 10, 12] but they go significantly beyond this previous work to enable specification of separation of duties in an intuitive and realistic way.

We explain the mechanism by example, by presenting a paper-based implementation followed by a computerized implementation. Our insight is that in the paper world a form carries within itself its history, which is used to enforce separation of duties. On the other hand, in the electronic world a database record is typically separated from its history as recorded in the audit trail. The key to our proposal is to keep the "relevant" history within each database record. We show this is feasible because the relevant history for each record can be kept quite small and therefore stored in a fixed length space. Finally we consider how this mechanism would apply in a totally automated environment where decision-making is embedded in programs rather than requiring human intervention.

In terms of the IFIP WG 11.3 question list, the most pertinent questions are 2 and 13. Other related questions are 8, 11 and 16.

2 THE CHECK VOUCHER EXAMPLE

Our example considers a situation in which payment in the form of a check is prepared and issued by the following sequence of events.

1. A clerk prepares a voucher and assigns an account.
2. The voucher and account are approved by a superviser.
3. The check is issued by a clerk who must be different from the clerk in step 1. Issuing the check also debits the assigned account.[†]

The policy expressed by this sequence of events embodies the idea of separation of duties since the three steps must be executed by different people. So it will take the collusion of two clerks and a superviser to perpetrate fraud.

This example incorporates the following three requirements on the basis of practical business considerations.

- *Dynamic Separation of Duties.* A particular clerk is prohibited from performing steps 1 and 3 for the same voucher. However he can perform step 1 on some vouchers while doing step 3 on some other ones. In other words a particular clerk can prepare vouchers as well as, on different occasions, issue checks. However he cannot issue a check for a voucher prepared by himself.

  This requirement has been often missed in proposals to implement separation of duties. This is true both of proposals [11, 20, for instance] based on Biba style mandatory controls [1] or those based on Boebert and Kain's assured pipelines [2].

[†]Strictly speaking we should debit one account and credit another in equal amounts. The important point for our purpose is that issuing a check modifies account balances.

Note that Shockley [19] introduces additional mechanisms, beyond Biba labels, to address this need.

Surprisingly Clark and Wilson [4] believe that dynamic separation of duties is a very difficult requirement to achieve and should therefore at most be an optional feature in any integrity criteria that might be developed along the lines of the Orange Book [5]. In our opinion this requirement is not terribly difficult to achieve but does require new, elegant and intuitive mechanisms such as described here.

- *Hierarchical Roles.* Suppose that a clerk is not available for step 1 or 3. In such cases it makes perfect business sense to allow a superviser to assume the role of a clerk. This can happen during quite routine circumstances such as lunch breaks or peak vacation periods as well as in unusual circumstances such as a strike. It is therefore all the more important that a superviser be able to assume a clerk's role without the intervention of some all powerful security czar. It should be a transparent and automatic act for a superviser. On the other hand we would not want a clerk to assume the role of superviser. So the superviser and clerk roles are hierarchical[‡] in the sense that a superviser should be able to assume a clerk's role on demand but not vice versa. It appears that the ntree hierarchy recently proposed by this author is an ideal candidate in this context [15, 17].

  Note that hierarchical roles mesh in very nicely with dynamic separation of duties. In our example it is quite proper for a superviser to play out the role of a clerk in step 1, but then he must be prevented from going ahead and approving that same voucher himself in his supervisery capacity! Indeed in absence of support for dynamic separation of duties hierarchical roles would be extremely dangerous. As a result papers based on static separation of duties, including the original Clark-Wilson paper [3], have simply ignored this issue.

- *Substitution of Attribution.* Our final requirement follows as a corollary to the first two. Suppose a superviser by virtue of hierarchical roles executes step 1 for a voucher and then discovers there are no other supervisers available to approve it. At the same time he finds there are now two clerks available since say one of them has just returned from lunch break. It would be convenient if the superviser could ask one of the clerks to assume the attribution for step 1 so that the superviser is then permitted to execute step 2.

  There are several different ways by which this effect can be achieved. We could void the voucher under progress and restart the entire sequence on a fresh voucher. In the current example this alternative would work reasonably well at the cost of reentering the required information on a fresh voucher. In a more complex situation redoing the entire sequence from step 1 might be infeasible since some of the earlier steps are say done in a different department or are very time consuming. In such cases it would be nice if we could back up one step and rerun that step alone under the authority of a different user. This is suggested by Nash and Poland [13] who draw

[‡]We use the term hierarchical in this paper with its usual computer science connotation of a partial order. This is different from the Orange Book [5] usage of this term which has the connotation of a linear order.

an analogy to crossing out a signature on a document and resigning by a different person, by which act the second signatory assumes the responsibilities implied by the signature. The analogy however is not perfect since redoing the previous step does involve some effort. To make the analogy perfect all we need to do is allow the effect of the most recent step to persist and only change the attribution of who actually executed that step. In the context of our example this means that the data entered on the voucher by a superviser in step 1 can remain unchanged, without requiring reentry, while the attribution of that step can be changed from that superviser to a clerk. This will permit the superviser to then approve the voucher. All three modes identified above—voiding the voucher, redoing the most recent step, and substituting the attribution for the most recent step—are of practical value and each one should be supported.

To summarize the three requirements itemized above are extremely important from a business standpoint. They emerge quite naturally from the operational requirements of our simple check-voucher example. Unless our mechanisms can accommodate this kind of flexibility, security will continue to be viewed as an unnecessary hindrance to normal business procedures and users will seek to circumvent it in order to get their job done.

3 PAPER-BASED CONTROLS

Implementation of this policy in a paper-based system follows quite directly from its statement.

- The voucher is realized as a form with blank entries for the amount and account, as well as for signatures of the people involved. As the three step sequence of our example gets executed these blanks are filled in. On completion of the sequence copies of the voucher are filed in various archives for audit purposes.
- The account is represented by say a ledger card, where debit and credit entries are posted along with references to the forms which authorized these entries.

In a paper-based system separation of duties—static or dynamic—can only be enforced by employee vigilance (in the sense that employees refuse to process improper vouchers). Violations can be detected by internal or external audits. Hierarchical roles present no problem since they merely lay out the policy regarding who can sign where on what forms. The issue of substituting attribution is easily handled in all three modes identified above. That is, a voucher may be voided or selected data or signature entries on it may be altered with appropriate signatures to attest that this is properly done.

By their very nature paper-based controls rely on employee vigilance and internal and external audits for their effectiveness. Computerization brings with it the scope for enforcing the required controls by means of an infallible, ever vigilant and omniscient automaton, viz., the computer itself. The crucial question is how do we specify and implement mechanisms similar to well-established paper-based controls in a computerized environment?

4 COMPUTERIZED CONTROLS

We begin by noting a fundamental distinction between a voucher and an account first observed by this author in [16].

- The voucher is *transient* in that it comes into existence, has a relatively small sequence of steps applied to it and then disappears from the system (possibly leaving a record in some archive). The history of a voucher can be prescribed as a finite sequence of steps with an a priori maximum length.
- The account on the other hand is *persistent* in the sense it has a long-lived, and essentially unbounded, existence in the system. During its life there may be a very large number of credit and debit entries for it. Of course, at some point the account may be closed and archived. The key point is that we can only prescribe its history as a variable-length sequence of steps with no a priori maximum length.

Both kinds of objects are essential to the logic and correct operation of an information system. Transient objects embody a logically complete history of transactions corresponding to a unit of service provided to the external world by the organization. Persistent objects embody the internal records required to keep the organization functioning with an accurate correspondence to its interactions with the external world.

Our fundamental thesis is that separation of duties can be achieved by enforcing controls on transient objects, for the most part. The crucial idea, which makes this possible, is that transactions should be executed on persistent objects only as a side effect of executing transactions on transient objects. This thesis is actually simply borrowed from the paper-based world where it has been routinely applied ever since bookkeeping became an integral part of business operations.

Our proposal is summarized in figure 1. The idea is that the first level of defense is the limitation of updates to transactions. That is users cannot write into the database directly but may do so only by executing transactions which have been certified to be correctness preserving. In other words a transaction maps a correct state into another correct state. This notion of a transaction has been an integral part of database management systems [6] for some time now and is precisely equivalent to the transformation procedure of Clark-Wilson [3]. Our contribution is to identify the second line of defense required to effectively enforce separation of duties. This is based on partitioning the objects in a database into transient objects such as vouchers and persistent objects such as accounts. The principle here is that persistent objects cannot be modified directly by a transaction. Modification of persistent objects is possible only indirectly by the side-effect of operations on transient objects. Each transient object carries with it its entire history. Persistent objects carry an abbreviated history.

4.1 THE EXAMPLE REVISITED

The abstract framework of figure 1 is now made concrete in terms of our check-voucher example. We propose to represent the potential history of an information object by a transaction control expression. First consider transient objects. The history of the check in our example is described as follows.

prepare	•	clerk;
approve	•	superviser;
issue	•	clerk;

Each term in this expression has two parts. The first names a transaction. The transaction can be executed only by a user with role specified in the second part. For simplicity in discussion assume each user has only one role. So 'prepare • clerk' specifies that the prepare transaction can be executed on a check object only by a clerk. The semi-colon signifies sequential application of the terms. That is a superviser can execute the approve transaction on a check only after a clerk has executed the preceding prepare transaction. Finally, separation of duties is further enforced by requiring that the users who execute different transactions in the transaction control expression all be distinct. As individual transactions are executed the expression gets incrementally converted to a history, for instance as follows.

prepare	•	Tom;	prepare	•	Tom;	prepare	•	Tom;
approve	•	superviser;	approve	•	Dick;	approve	•	Dick;
issue	•	clerk;	issue	•	clerk;	issue	•	Harry;
	(a)			(b)			(c)	

The identity of the user who executes each transaction is recorded to enforce the requirement that these users be distinct. So if Tom attempts to issue that check at point (b) in this sequence the system rejects the attempt. Dynamic separation of duties is therefore an intrinsic aspect of this mechanism.

A transaction control expression thus contains a history of transactions executed on the object in the past and a potential history which authorizes transactions which can be executed in future. The expression begins as a constraint and ends as a complete history of the object. Transient objects will generally have a history of the order of a dozen steps at most, so this approach is viable. Moreover the information which is filled out as the history gets executed is an essential part of the object and should anyway be represented as part of the object state. The reader is referred to [16] for additional notation and examples of transaction control expressions.

As pointed out by Nash and Poland [13] the voting notation of [16] can account for hierarchical roles. In our example this would require the following modification to the transaction control expression for a voucher.

1:	prepare	•	clerk=1, superviser=1;
	approve	•	superviser;
1:	issue	•	clerk=1, superviser=1;

The notation 1: indicates the number of weighted votes required in order to proceed to the next step in the sequence. Each vote is registered by an execution of the indicated transaction—prepare in step 1 and issue in step 3. The ‘clerk=1’ and ‘superviser=1’ terms indicate that both supervisers and clerks have a weighted vote of 1 for this purpose. The voting notation in [16] was developed to allow for situations where multiple approvals are needed to proceed to the next step. It is interesting that the notation also accounts for hierarchical roles. However it does so on a expression by expression basis. This is clearly a useful facility to have. At the same time it would much more convenient to specify hierarchical roles globally, independent of what individual transaction control expressions are specifying. So we would like to say that a superviser can always do what a clerk can do without requiring that this be encoded into every occurrence of clerk in a transaction control expression. To do so efficiently we need to quickly recognize whether one role dominates another in a given hierarchy. The ntree hierarchy proposed by this author [15, 17] is ideally suited for this purpose.

It remains to consider substitution of attribution. It is a simple matter to arrange for substitution without redoing any of the previous steps. We simply need to change the recorded identity in the portion of the expression which has been converted to a history. The previous record can be stored in an audit trail and does not need to be available to us online as part of the object’s transaction control expression. The ability to redo the last step is also easy to achieve. The necessary information is available in a recovery log or some similar mechanism so long as the previous step was not committed. We can continue to maintain this information until the following step (if any) has been committed.

In a paper-based system a transient object is represented by a form. Different transactions executed on the object are recorded by appropriate entries on the form. Identification of the user executing each transaction is achieved by signatures. In automated systems user identities must be recorded with guaranteed correctness. This is a critical facility lacking in current commercial access-control systems such as IBM’s RACF [7].

Turning to persistent objects, we propose the following transaction control expression for representing the potential history of an account.

$$\begin{array}{clcll} & \text{create} & \bullet & \text{superviser;} & \\ \{ & \text{debit} & \bullet & \text{clerk} + & \\ & \text{credit} & \bullet & \text{clerk} & \}; \\ & \text{close} & \bullet & \text{superviser;} & \end{array}$$

The curly parenthesis denote repetition while ‘+’ gives a choice on each repetition. The idea is that a account gets created and is thereafter debited or credited. At some point it may be closed. Any object whose transaction control expression contains repetition is by definition a persistent object. Similarly any object whose transaction control expression does not contain repetition is by definition transient.

The history of a persistent object is likely to be lengthy. It is clearly impractical to convert the transaction control expression incrementally into an history, as done for transient objects. We can realistically have only some abbreviated history for persistent objects available to the access control system. Fortunately it is improper to require that all transactions executed on a persistent object be performed by distinct users. After all an account may have hundreds of debit and credit operations while the organization employs only a few dozen clerks. Separation of duties carried to this extreme will paralyze the organization.

Our fundamental principle is that transactions are executed on persistent objects only as the side effect of executing them on transient objects. Then separation of duties can be enforced by keeping the following history information.

1. The entire history of transient objects.
2. A partial fixed length history of persistent objects for non-repetitive portions of the transaction control expression.

For the account example assume that Dick is the superviser who creates the account (as a side effect of executing a transaction on some transient object). The transaction control expression of the account is modified to record this fact as follows.

$$\begin{array}{lllll} & \text{create} & \bullet & \text{Dick;} & \\ \{ & \text{debit} & \bullet & \text{clerk} + & \\ & \text{credit} & \bullet & \text{clerk} & \}; \\ & \text{close} & \bullet & \text{superviser;} & \end{array}$$

Thereafter as debit and credit transactions are executed on the account the expression remains unmodified. Finally when the account is closed by some superviser other than Dick, say Jerry, this fact is recorded to give us the following.

$$\begin{array}{lllll} & \text{create} & \bullet & \text{Dick;} & \\ \{ & \text{debit} & \bullet & \text{clerk} + & \\ & \text{credit} & \bullet & \text{clerk} & \}; \\ & \text{close} & \bullet & \text{Jerry;} & \end{array}$$

So there is a separation of duty involved in creating and closing the account. But separation of duty in debiting and crediting it is enforced only to the extent specified in the transaction control expressions on the transient objects related to this account.

Finally we have a more subtle issue concerning the interaction between the actions of a superviser in approving a check and in creating a account. That is Dick should be prevented from approving checks for accounts that he created. Note that the transaction control expression of an account contains the information necessary for this purpose, i.e., whether or not Dick created the account. This gives us the necessary basis for enforcing separation of duties across different but related objects.

4.2 AUTOMATED DECISIONS

Thus far we are assuming that the individual transactions in a transaction control expression will be explicitly invoked by some user. We now consider situation where the decisions are themselves programmed into transactions. In terms of our check-voucher example the approval transaction is actually now responsible for making the decision that it appropriate to issue a check. This may for instance involve verifying that a vendor has delivered on a purchase order.

It has been argued that with such automated decision making there is no benefit to breaking up the sequence of prepare, approve and issue into three separate transactions. Instead it suffices to have a single program accomplish all three steps, with the stipulation that the programmer should not be the one to use it in production [21].

We believe on the contrary that the same separation used in a manual or partially automated system should carry over to the fully automated case. The only difference being that the separation of duties is applied to the coding, maintenance and installation of transactions rather than to their actual execution. That is we need three separate prepare, approve and issue transactions with the stipulation that the same programmer cannot modify all three!

5 CONCLUSION

We have presented a mechanism and notation for specifying and enforcing separation of duties in computerized information systems. The mechanism is strongly motivated by the usual paper-based controls in terms of forms and books. A key idea is to distinguish transient objects (analogous to forms) with a fixed length history from persistent ones (analogous to books) with potentially unbounded histories. Transactions are executed on persistent objects only as the side effect of executing transactions on transient objects. This allows us to enforce separation of duties by maintaining a complete history for each transient object and an abbreviated history for persistent objects.

We have shown that our mechanism is capable of enforcing dynamic separation of duties with hierarchical roles and substitution of attribution. We would urge the community to regard these features as quite easy to implement and therefore required in any system that claims to support application integrity.

References

[1] Biba, K.J. "Integrity Considerations for Secure Computer Systems." Mitre TR-3153, Mitre Corporation, Bedford, Mass., April 1977.

[2] Boebert, W. and Kain, R. "A Practical Alternative to Hierarchical Integrity Policies." *NBS-NCSC National Computer Security Conference*, 18-27 (1985).

[3] Clark, D.D. and Wilson, D.R. "A Comparison of Commercial and Military Computer Security Policies." *IEEE Symp. on Security and Privacy*, 184-194 (1987).

[4] Clark, D.D. and Wilson, D.R. "Evolution of a Model for Computer Integrity." In [14].

[5] Department of Defense National Computer Security Center. *Department of Defense Trusted Computer Systems Evaluation Criteria.* DoD 5200.28-STD, (1985).

[6] Gray, J. "Notes on Data Base Operating Systems." In *Operating Systems—An Advanced Course*, Bayer, R. et al (editors), Springer-Verlag, pages 393-481 (1978).

[7] I.B.M. Corporation. *Resource Access Control Facility (RACF) General Information Manual.* GC28-0722 (1987, 12th edition).

[8] Karger, P.A. "Implementing Commercial Data Integrity with Secure Capabilities." *IEEE Symposium on Security and Privacy*, 130-139 (1988).

[9] Katzke, S.W. and Ruthberg, Z.G. (editors). *Report of the Invitational Workshop on Integrity Policy in Computer Information Systems.* National Institute of Standards and Technology, Gaithersberg, Maryland, U.S.A. (1987).

[10] Kieburtz, R.B. and Silberschatz, A. "Access-Right Expressions." *ACM Transactions on Programming Languages and Systems* 5(1):78-96 (1983).

[11] Lee, T.M.P. "Using Mandatory Integrity to Enforce "Commercial" Security." *IEEE Symposium on Security and Privacy*, 140-146 (1988).

[12] Minsky, N. "An Operation-Control Scheme for Authorization in Computer Systems." *International Journal of Computer and Information Sciences* 7(2):157-191 (1978).

[13] Nash, M.N. and Poland, K.R. "Some Conundrums Concerning Separation of Duty." *IEEE Symposium on Security and Privacy*, 201-207 (1982).

[14] Ruthberg, Z.D. and Polk, W.T. (editors). *Report of the Invitational Workshop on Data Integrity.* National Institute of Standards and Technology, Gaithersberg, Maryland, U.S.A. (1989).

[15] Sandhu, R.S. "The NTree: A Two Dimension Partial Order for Protection Groups." *ACM Transactions on Computer Systems* 6(2):197-222 (1988).

[16] Sandhu, R.S. "Transaction Control Expressions for Separation of Duties." *Fourth Computer Security Applications Conference*, 282-286 (1988).

[17] Sandhu, R.S. "Recognizing Immediacy in an NTree Hierarchy and its Application to Protection Groups." *IEEE Transactions on Software Engineering* 15(12):1518-1525 (1989).

[18] Sandhu, R.S. and Jajodia, S. "Integrity Mechanisms in Database Management Systems." *13th NIST-NCSC National Computer Security Conference*, 526-540. (1990).

[19] Schockley, W.R. "Implementing the Clark/Wilson Integrity Policy Using Current Technology," *NIST-NCSC National Computer Security Conference*, 29-37 (1988).

[20] Terry, P. and Wiseman, S. "A 'New' Security Policy Model." *IEEE Symposium on Security and Privacy*, 215-228 (1989).

[21] Weber, R. *EDP Auditing: Conceptual Foundations and Practice.* McGraw-Hill (1988).

[22] Wimbrow, J.H. "A Large-Scale Interactive Administrative System." *IBM Systems Journal* 10(4):260-282 (1971).