Payoff

Microcomputers (clients), local area networks, and mainframes (servers) should be combined into a cooperative processing structure, because such a structure can leverage the advantages of each technology. The success of this type of system depends on the effective distribution of data on the client and server components. This article discusses a client/server system that connects clients through advanced program-to-program networks to an MVS mainframe server.

Introduction

Because of recent developments in both data communications and data base management, organizations can implement many different types of client/server applications. Clients can be Windows, LAN, or OS/2 applications, while the server can manage data bases residing on the mainframe. System developers expect such distributed processing approaches to provide many long term corporate benefits.

Such implementations can make use of data base solutions on a mix of mainframe and nonmainframe platforms, for example RS/6000 and AS/400, and link those data bases with data bases on the mainframe. Given this likelihood, this article addresses reviewing the infrastructure for client/server and distributed data bases. A secure mainframe environment provides a well-designed control model that nonmainframe client/server implementations can follow. It is assumed that the security practitioner has a familiarity with MVS mainframe security issues (e.g., the use of SAF) and relational data base security concerns, as well as issues specific to DB2 and MVS implementations.

Everyone has a different definition of what it is that precisely constitutes a client/server system. In addition, security professionals have opinions about what role client/server systems have in their organizations. It is a fact that many applications referred to as client/server applications involve the mainframe.

The security practitioner should be aware that this article does not attempt to provide a technical blueprint for securing all possible client/server systems that use the MVS mainframe. The permutations of mainframe-based client/server processing are limited by the amount of time and space available. Readers are encouraged to consult other sources in addition to this article.

The MVS mainframe server holds a relational data base running on DB2. Use of DB2 affords the implementation the opportunity to use the DB2/Client Server suite of products, which use distributed relational data base architecture to manage cross-platform DB2 communications.

Planning For Security

The first step in client/server systems design is to determine what information can be distributed. To make the best decision, system designers consider such issues as the need for remote access and backup and recovery requirements. Security practitioners can contribute to this analysis by determining the level of security required to protect certain data. The analysis should assess the sensitivity of data to exposure and determine the controls needed. This sensitivity to exposure is heightened by the fact that many distributed system implementations do not provide the same level of protection as mainframe security services. Therefore, although performance benefits may arise from distributing the data,
such benefits need to be weighed against the lack of security services in a distributed environment.

A general guideline for security practitioners to follow in analyzing the security required is to pursue a consistent level of security across platforms and networks. In particular, this consistent level of security should be applied to protecting the system at the level necessary to secure the most sensitive information that is managed by the client/server network. This means analyzing from one end of the data communications flow to the other the security facilities that need to be implemented. For example, if the client/server system stores credit card information on the host, and client machines process it on a Novell network, security measures applied on the Novell network should be as close as possible to those implemented on the host.

When a system performs distributed data base processing for a client application, it must be able to satisfy the security requirements of the application client, the application server, and the network connecting them. Defining security requirements means making important security decisions regarding the following:

- Selection of end-user and network node names.
- Definition of network security parameters.
- Implementation of data base manager security.

It may seem redundant to emphasize the selection of end-user and network node names as a key control, however, many exposures can result from a lack of enterprisewide, cross-platform naming standards. For instance, a network operates based on the assignment of node addresses to define all nodes on a network. When two independent networks are interconnected, problems can develop if nodes on one network follow the same addressing scheme as nodes on another network.

Network security parameters define how nodes authenticate each other's activities and manage the data communications flow between them. All client/server transactions involve some aspect of one computer requesting services from another computer. Defining how these machines communicate, or indeed whether they are permitted to communicate at all, is a cornerstone of client/server network security. Data base manager security specifies how the users are to be permitted to process specific items of data. These issues are largely beyond the scope of this article.

Controlling Sessions With Vtam

The first step in developing a control structure is identifying the source and identity of the client/server request. Client/server applications connect to mainframe services through virtual telecommunications access method (VTAM) definitions, otherwise the connection with the mainframe-based server simply does not function. VTAM performs the node addressing and communications path control functions in an IBM System Network Architecture network. In doing this, VTAM is what allows terminals and applications to communicate with applications on the mainframe.

VTAM identifies network nodes and the method they are using to connect to MVS. These methods are called sessions, and each session is identified to VTAM by an application identifier(APPLID). For instance, a CICS session may have an APPLID of CICSNY. To initiate the session, the user types CICSNY at the VTAM screen. Without strong controls over these session connections, whoever knows the related APPLID can breach controls over the VTAM screen.

All of the APPLIDs managed by VTAM are listed in the ATCCONxx (where xxrepresents any two alphanumeric characters) member of the SYS1.VTAMLST data set.SYS1.VTAMLST
contains the information necessary for network definition and communications routing, and it is one of the most important data sets referred to in this discussion. All APPLIDs are identified in the other members in SYS1.VTAMLST. For example, CICSNY would be a member of SYS1.VTAMLST, and it would define the parameters used to manage the connections.

The security professional should keep in mind that although appropriate management of VTAM can improve controls, VTAM is not designed as a security-management product but as a communications door to the mainframe. Therefore, to protect VTAM from unauthorized changes, the external security manager product (e.g., RACF) should be used to limit access to SYS1.VTAMLST and similar data sets (e.g., SYS1.VTAMLIB). Update access to these data sets should be limited to those personnel responsible for managing distributed systems. Otherwise, there is a chance that other mainframe users could add unauthorized VTAM session definitions or make changes to existing ones.

After controls are established to prevent unauthorized VTAM session definitions from being added, a control that addresses how connections with existing, approved sessions occur should be established. The system authorization facility (SAF) interface of VTAMAPPL can be used to control the connection of sessions to VTAM and prevent unauthorized sessions from being initiated. For example, if VTAMAPPL is not used, there is no mechanism by which VTAM can make sure that the inbound client/server request is permitted to issue an OPEN call.

SAF's VTAMAPPL checking is necessary to make sure that the inbound request is authorized to open particular VTAM APPLIDs. It is important that the SAF interface controls the connection of sessions with VTAMAPPL, because this allows CA-ACF2, CA-Top Secret, and RACF to manage the VTAMAPPL class. One overall objective is to use as many of the facilities in external security managers as is feasible to protect mainframe client/server interactions. Consequently, there should be no APPLIDs that are not managed by the organization's external security manager product.

Next, the security practitioner should verify whether all of the SYS1.VTAMLST-defined nodes are authorized to connect with the client/server sessions controlled by VTAM. All of the client/server network nodes are connected in a system network architecture (SNA), which actually defines nodes as logical units (LUs). A LU can be another MVS host machine, a terminal, a printer or an external (i.e., non-MVS) machine. This is important because some of the authorization services in a SNA network are based on identification of LUs.

In addition, SNA Network Interconnect (SNI) connects independent SNA networks by mapping one name space to another name space. This mapping allows a logical unit on one SNA network to connect with a logical unit on another SNA network. These connections are defined to VTAM through definition of SNI links. If a session APPLID's dynamic cross domain parameter (CDRYN) is set to YES, then cross domain log-ons for the APPLID will be allowed. This allows other environments to log in to VTAM through the SNI links. SNI links are defined in SYS1.VTAMLST (ATCCONxx). Forcing the VTAM APPL definition to go through a secured session controls SNI link access.

Controlling VTAM, then, is the necessary first step in controlling client/server interactions, and it consists of limiting access to key VTAM libraries and managing changes to VTAM through strong quality-assurance processes. The first major control point in managing the security of connections with VTAM is to control session establishment. The second is to control the authorization of sessions to users and network nodes. These control issues recur throughout this discussion of client/server security.

The control of VTAM alone, however, does not address the full scope of control over client/server network interfaces with MVS. Rather, it simply manages how data is controlled by MVS after it gets there. Controls over data before it gets to the VTAM session must be addressed within the client/server network itself. Client/server communications methods can be defined to the MVS mainframe to facilitate secured communications between client and server.
Managing Communications With Application Program to Program Communication

Application program-to-program communication (APPC) is an IBM Corporation product that provides programs the ability to communicate through conversations across a SNA network. The programs involved in a conversation are logical units (LU), and each LU is a communications socket that allows a program to access SNA services. The LU is responsible for routing data packets to the appropriate partner LUs. In effect, the LU definition is what facilitates data communications between machines. In a client/server network, the client machine could be viewed as the partner LU to the host in an APPC conversation. For APPC conversations to take place, an APPC session must be established at each LU to allow data packets to flow between partners.

The APPC product supports Logical Unit 6.2 (LU6.2) network protocols. LU6.2 allows programs on different platforms to communicate with each other so that client/server communications between an AS/400 and MVS, for example, are possible. LU6.2 manages sessions, and thus provides the capability of managing connections by verifying all sessions. The security of end-user connections in LU6.2 are managed in two modes: already verified mode and conversation mode. These settings are determined by the APPLID definition in SYS1.VTAMLST's ATCConxx member. Conversation security mode is applied to allAPPLIDs where the SECACPT parameter is set to CONV. When conversation security is used, a variety of security settings are available (these are to be discussed in subsequent sections). IfSECACPT is set to ALREADYV, this means the APPLID is managed in a mode in which security is already verified.

To understand how and when security controls can be applied in APPC conversations, the practitioner must understand how APPC conversations take place. They must then identify where controls can be applied.

First, the definition of the host LU must take place. The client/server application will make use of a particular session on the mainframe (as already described in the section on VTAM). The control issue is to assess whether the security parameters defined in VTAM for this host session are appropriate.

Second, definition of the partner LUs must take place, so that they are available for client/server processing. The security issue is whether the partner LUs are authorized to connect with the VTAM session associated with the client/server process.

Third, an APPC session is established. Session-level security in APPC verifies partner LUs at the establishment of sessions. Session-level security is based on the use of an encryption key, which is never transmitted between nodes but is used to generate random strings that are used to validate the identity of partner LUs.

When the session is established, VTAM and APPC capabilities can be used to further tighten client/server communications controls. Extensions can make VTAM APPC-aware and increase controls by checking the source of the inbound client/server request. This is done inSYS1.VTAMLST's listing of authorized VTAMAPPLIDs. If the APPL statement forCICSNY, for example, specifiesAPPCC=YES, then APPC manages communications with that APPLID. The VERIFY keyword determines whether the source of the inbound request will be checked. For instance, VERIFY=NONE means that no such checking will take place. VERIFY=OPTIONAL means that this checking will occur, but only when there is a session key available. VERIFY=REQUIRED forces inbound request source checking to occur. When source checking occurs, theAPPLCLU class (in SAF) is referenced. This is the SAF class used for LU-LU encryption, and this approach should be used in managing client/server network security. LUs in distributed DB2 can be established and protected in RACF, for example, through definition of APPLCLU resource profiles.

Fourth, the APPC conversation is allocated. This is done by APPC issuing an ALLOCATE request to initiate a conversation between LUs. It is at this stage when the previously discussed conversation mode becomes critical to security. Depending on this
mode setting, the APPC conversation can take place with or without the host security package's authorization.

Application security dictates the parameters under which conversation level security will be enforced. For instance, if the APPC conversation mode is set to SECURITY=PGM, then user ID and password validation must take place. This validation takes place when the ALLOCATE conversation request is issued. If the VTAM session is linked to the external security manager, then the ID and password will be validated by the external security manager.

When the mode is SECURITY=SAME, the user ID is assumed to have already been verified at the partner LU, and the password is not sent. In short the partner LU is assumed to be a trusted machine. This user ID endorsement acts as a substitute for a password, and is carried on SNA ATTACH requests. As a result, SECURITY=SAME should only be used on communications originating from machines that actually are trusted hosts. The transaction flow is such that ALLOCATE(SEC=SAME) occurs. The server checks the LU partner support made available for the client. The client has already verified the user ID, so the next step in the communication flow is to ATTACH the session between the two LUs. The server checks with the external security manager to confirm the already verified user ID is authorized to use its services.

Problems can occur when SECURITY=SAME is being used. A user could stuff the client's Set_Conversation_Security_User_ID call (a standard common programming interface call) to set the conversation user ID to something with powerful authorities on the MVS host. The session would be attached between the MVS host and the client on the basis of an unauthorized use of a user ID.

The first point where security checks can be applied in APPC conversations occurs when the session is about to be bound. Properly designed security can prevent unauthorized clients from establishing conversations with the server. APPC can be configured with the external security manager and VTAM security options to limit the logical units from which APPC conversation requests may enter MVS. These limits should be put into effect wherever possible to ensure that inbound APPC requests contain user IDs and passwords and to enforce the SECURITY=PGM conversation mode. Otherwise, the mainframe will be exposed to the possibility of being breached by inbound requests that have not authenticated the user.

The described controls can confirm that the user ID is authenticated at the server and is connected with the right session. But the activities of users within the mainframe's data bases still need to be controlled after access is granted. Standard DB2 security measures can be applied to ensure that basic controls are in force, but some specific steps should be taken to address client/server issues.

### Distributed Relational Data Base Architecture Security

IBM Corporation's Distributed Relational Data Base Architecture (DRDA) is a method for distributing client/server DB2 data base systems across platforms. These platforms include RS/6000, OS/2, and AS/400. Particular security measures must be enforced, otherwise the security of the entire client/server system will be suspect.

End-to-end distributed data base security can be achieved using a combination of the above software products. With clients, the various DB2 products (e.g., DB2/2, DB2/6000) can implement distributed relational data base request access control at a local level. A combination of VTAM, APPC, external security manager, and DB2 can implement DRDA remote access request access control at the host mainframe. The important point is that DRDA and its communications components do not provide security, but instead rely on controls in these other software products to achieve the desired level of control.

If APPC security has been established with SECURITY=PGM, the application server validates the user ID before any data base processing occurs. In some DRDA
implementations, the communication subsystem validates security information before passing the request to the database management system. The client's authorization mechanisms can control access between the application requester and application server.

When DB2 receives the user name from the client, the server can restrict the user names received from a given client. DB2's Distributed Data Base Facility (DDF) has an inbound checking facility that accomplishes this task. This type of checking can specify that a given user ID can only use particular partners. The server can restrict the user ID DARRENJ to come from client WATERLOO. Any other clients that attempt to pass the user ID DARRENJ to the server will not have their client/server requests authenticated.

A major function of any transaction processing (TP) monitor is to provide ways of ensuring that only authorized users and network nodes can access the resources controlled by the TP monitor. DDF functions fulfill this requirement, because DDF provides many security mechanisms for controlling access to DB2 servers:

- Confirming remote client access.
- Controlling end-user access with DB2 catalog entries.
- Translating end-user connections to unique identifiers.

Even though user IDs must be unique within a specific system, user IDs may not be unique throughout the entire SNA network. For example, suppose there is a user named DARRENJ on the NEWYORK system, and another user named DARRENJ on the WATERLOO system. If the DARRENJ in WATERLOO is not the same person as the DARRENJ in NEW YORK, the SNA network (and consequently the distributed data base systems within that network) cannot distinguish between them. If this situation is not corrected, the DARRENJ in WATERLOO can use the data base privileges granted to the DARRENJ at the NEW YORK system.

To eliminate naming conflicts and the potential security hazards that result, names can be translated on the different nodes within a client/server system. Outbound name translation allows the application requester to translate the end user's ID before sending the ID to the destination in the SNA network. Inbound name translation allows the application server to translate the end user's name it receives from its partner LU.

Under such a scheme, the DARRENJ in NEW YORK receives a different name (NYDMJ) when DARRENJ makes distributed data base requests. In outbound translation, whenever DARRENJ issues a SQL request to WATERLOO, the NEW YORK system can change the name to NYDMJ before transmitting the user ID. The WATERLOO system administrator must register this new name (NYDMJ) so the WATERLOO system can accept the name. With inbound translation, the approach is to allow WATERLOO to translate names it receives from NEW YORK. In this case, WATERLOO changes NEW YORK's DARRENJ to NYDMJ.

DB2 facilitates inbound name translation by using an extension to the DB2 catalog called the Communications Data Base (CDB). Just as the DB2 catalog is the brains of DB2's internal operations, the CDB is the brains of DB2's remote operations using DDF. DB2 activates inbound name translation by having the USERNAME column of the CDB's SYSIBM.SYSLUNAMES table set to I (to allow inbound translation) or B (to allow both inbound and outbound translation).

The inbound name translation process receives the user ID and LU name, and then searches for a corresponding row in the DB2 catalog's SYSIBM.SYSUSERNAMES table. Remote access is denied when no corresponding row exists in this DB2 catalog table. Entries in the table's AUTHID and LUNAME columns indicate a specific user from a particular client is being converted. When only AUTHID is specified, the specific user ID is accepted and converted regardless of the client it comes from. Where only LUNAME is specified, any user ID from a particular client is accepted and converted. The inbound user ID is then changed to the user ID indicated in the NEW AUTHID column of the SYSIBM.SYSUSERNAMES
Any resource authorization checks internal to DB2 (e.g., SQL table privileges) are performed with this new translated user ID.

It is possible to translate user IDs so that a group of similar users will be represented by a new name. In addition, the partner LUs allowed to connect to the server can be restricted through the DB2 catalog's SYSIBM.SYSLUNAMES table. It is also possible to use the CDB to manage special privileges by granting them to inbound users. This is a potential productivity tool as well as a potential exposure. If JOHNU has SYSADM authorities on the NEWYORK system, and JOHNU should be able to connect to WATERLOO with SYSADM authorities, the inbound access request can be translated to the NEWAUTHID = "SYSADM". Such techniques for granting privileged access should be discouraged or, if used, used with extreme care.

It is important to secure the USERNAMES table in the CDB as one of the columns in the PASSWORD column. These passwords for outbound user IDs are stored in clear text, which means that anyone with SELECT access to this table can SELECT all passwords. Preventing access to this table is necessary, although the password risk will be a nonissue if SECURITY=PGM is used (as, in this case, the inbound encrypted password is forwarded directly to RACF).

Controls in DRDA implementations help to identify the source of client/server requests, and limit access depending on who the user is or where the access has come from. However, these controls have assumed that the user is issuing commands interactively, when in fact the client/server system is more likely to be processing transactions or programs sent to the mainframe from the client. The next step in our inquiry is to determine how to build on the controls suggested so far to secure remotely requested program processes.

**Remote Procedure Call Model**

Remote procedure calls (RPCs) enable a program to start a subroutine in a different computer. A remote procedure call is a procedure requested locally on a client that runs on a machine other than the client. This remote server machine may be the mainframe, another microcomputer, an AS/400, or any machine.

When the program issues a call, the RPC code takes the parameters passed with the call and sends them to a remote subroutine. Any information returned by the remote subroutine goes back to the program that made the call. In this way, RPCs simplify programming for distributed applications, because no communication specifications are required in the partner programs. The calls look like standard subroutine calls running on one system. The RPC code automatically correlates the requests and responses. In addition, most RPC implementations provide automatic data conversion functions.

RPC services are usually stored as a service layer available on client machines. In Windows systems, the RPC service layer places a series of dynamic link libraries in the machine that gives client/server applications access to many transactions on various host systems.

As a result, RPCs automate data translation services between platforms and applications. Client/server systems need only to define communications interfaces, which act as stubs that reformat data between client and server programs. To invoke RPC services, application program code simply calls the stub routine, which takes the data, locates the server, opens a session, and passes the message. The server stub receives the message, unpacks the data, and converts the data to be processed by the server computer.

RPCs are valuable because a server program could be updated or replaced, but as long as the interface stays the same no changes need to be made to the client. The same RPC would be used, but the server's stub would convert the data differently. What the RPC model does is impose a standard format for the input and output to remote functions. This allows the RPC to retrieve data, update data, and request calculations. There is great value
in being able to run RPCs in client/server systems, but some acute exposures exist in relation to RPCs.

It is necessary to ensure that all RPCs function in the manner in which they are intended to function. This does not seem like too significant an exposure, but if the client/server application makes use of internally developed RPCs that retrieve information from the DB2 Communication Data Base's USERNAMES table, it becomes evident that such information would be important in the operation of a client program.

As shown, this USERNAMES table contains user IDs, LU names, converted-to user IDs, and outbound user IDs passwords. If there is an internally developed RPC calledGetUserInfo() residing on a client machine that simply looks up user information from the CDB for testing purposes, testers want to be able to occasionally confirm the user IDs and LU names associated with program processes. So this RPC is an open RPC, meaning that it retrieves information based on user-defined input parameters. As such, even though the individuals doing the testing may be restricted from accessing certain tables at the user ID level, the RPC runs with enough authorization that it is able to retrieve information from any data base table.

The possibility is clearly there for someone to misuse the RPC and retrieve all sorts of information that normally should not be retrieved. The RPC can do this by receiving out-of-the-ordinary, user-defined input. The user can do this by inputting a different data base name into an end-user program that calls the RPC from any of a number of end-user applications. In manipulating this input, the user could request the following:

{NEW YORK} [— This goes against the NEW YORK system on the host.
[SYSIBM.SYSUSERNAMES] [— This is from the user names data group.
AUTHID=DARRENJ [— This is the user.
LUNAME=WATERLOO [— This is what DARRENJ accesses from NEW YORK.
PASSWORD=? [— What is DARRENJ's password on WATERLOO?

If this data store was entered into an RPC call to get the DB2Communications Data Base information, enough information would be received to compromise DARRENJ's user ID at the WATERLOO LU:

{NEW YORK}
[SYSIBM.SYSUSERNAMES]
AUTHID=DARRENJ LUNAME=WATERLOO
PASSWORD=MIBUNGLE

A need to control how all client/server transactions and RPCs interact with the host clearly exists. The security practitioner should be able to trust that the RPC originates from a secured machine, by ensuring no RPCs can originate from any machine that does not adequately authenticate its users. The practitioner also needs to ensure the operation of the RPC is strictly controlled at the host. RPCs that depend on an excessive amount of interactive user input should be discouraged. Otherwise, the practitioner should be prepared to accept the risk that the RPCs may be used for unauthorized purposes.

Managing Transactions With CICS

As described in the previous section, a need exists to control how the remote procedures interact with mainframe-controlled data. If the RPC leaves data open to user-defined manipulation or extraction, then the benefit of using RPCs to connect with mainframe data is only a performance benefit. It is possible to configure RPCs so that they tie directly to specific mainframe application processes.

The Customer Information Control System (CICS) is a transaction processing system, and it manages a great deal of access to information kept on corporate legacy systems. The most common way CICS manages information access is through CICS transactions. These
transactions are defined for specific purposes, and have access to information only within the limits of that function. CICS transactions cannot be manipulated by the person requesting the function to operate outside those limits. For this reason, CICS transactions are used to manage access to data base information (e.g., DB2). The most important concepts in the CICS architecture are session authorization and transaction authorization (i.e., controlling who can access what sessions and what program functions).

External security products interact with CICS to manage access to data base information (e.g., DB2). The most important concepts in the CICS architecture are session authorization and transaction authorization functions. External security interfaces with CICS (through defined SAF classes for CICS) for applications also need to be controlled. These interfaces define what each application can reference or change.

A cornerstone of managing CICS transactions is transaction security. In client/server systems that access mainframe data, CICS transactions can be used to receive client/server calls from outside of MVS (for example through RPCs) and perform the data access request on behalf of the client/server call. The benefit in this approach is that the CICS transaction has already been encoded to access specific data in a particular way. So long as only particular client/server calls can activate the CICS transaction, then existing CICS and SAF controls can be used for security.

SAF provides CICS transaction security by defining protected transactions as profiles within the 'TCICSTRN' class, for example. Security profiles in the external security product can then be granted access to specific CICS transactions through these transaction profiles. For CICS transactions that service client/server calls, it is recommended that the default security for each transaction be set to NONE. It should be noted that the type of access users and groups have to the data cannot be segregated within the CICS transaction profile. Users and groups can only be given access to execute the transaction or be prevented from executing it. Decisions about READ or UPDATE access to data are made within the transaction programming logic (i.e., whether a user has READ or UPDATE access to data depends on how the transaction used is defined). For instance, DMJREAD may be defined as an CICS transaction that gives its user READ access to certain information. If the security practitioner issues a client/server RPC to GetUserInfo() and this RPC uses DMJREAD to access information, the RPC or DMJREAD cannot be manipulated to do anything else but read that defined information. Even if the practitioner attempts to manipulate GetUserInfo(), he or she would still only end up being tied to DMJREAD, accessing only the specific data DMJREAD was constructed to access. Transaction managers such as CICS or IMS should be used whenever possible to control remotely issued RPCs.

In client/server systems, a common approach is to link RPCs from clients with CICS transactions on servers. This approach has the benefit of requiring application developers to simply call the RPC and by extension the CICS transaction without having to know how the transaction operates. This approach is also beneficial from a control perspective, as client requests can be packaged into well-defined CICS transactions controlled by the external security manager.

**Conclusion**

In general, security needs to be applied as an end-to-end function from the origination of the client/server request on the client machine to the management of the data on the host server. The goal in any information security construct is to ensure that only specified subjects (user IDs, machines) may use specified objects (data bases, files, transactions). Eventually, with the advent of developments like the Open Systems Foundation's Distributed Computing Environment standard, security will evolve to support a single, global user ID. RPC authentications may then be performed within network communications, with the Kerberos model applied to secure conversations.
Until such a time, however, mainframe security practitioners should be vigilant in ensuring the security of mainframe-based client/server applications. A need exists to establish consistent, enterprisewide naming standards for user identifiers and network nodes. The client/server implementation also must use external security manager software to authenticate user IDs and passwords and control the resources to which they have access. The goal is to use as many of the tried-and-true mainframe security techniques as possible to secure client/server systems that access mainframe data.

**Author Biographies**

**Darren Jones**

Darren Jones, BMath, CMA, is an information systems auditor with the Mutual Group in Waterloo, Ontario. He recently contributed to the development of a key client/server application at the Mutual Group.