NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

A PROTOTYPE OF MULTILEVEL DATA INTEGRATION IN THE MYSEA TESTBED

by

Andrew D. Portner

September 2007

Thesis Advisor:	Cynthia E. Irvine
Co-Advisor:	Thuy D. Nguyen

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** September 2007	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE A Prototype of Multilevel Data Integration in the MYSEA Testbed		**5. FUNDING NUMBERS**	
6. AUTHOR(S) Andrew Portner			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.		**12b. DISTRIBUTION CODE**	

13. ABSTRACT (maximum 200 words)

Much of the information managed by government agencies is stored in databases. Mission Assurance objectives often require the synthesis of data from separate databases. Data Integration is used to address this need for combining sets of data. However, because many government organizations store data in databases with different syntactic characteristics and at different classification levels, they will necessarily want to combine data from these divergent data sources. This requires a secure system to ensure that sensitive information is not disclosed to unauthorized parties.

The Monterey Security Architecture (MYSEA) is an experimental and extensible distributed Multilevel Secure (MLS) computing. This project set out to determine if a data integration application could be supported by the MYSEA environment. Through research on MLS database architectures, existing data integration technologies, and previous work to implement applications on MYSEA, this project was able to both define a high-level design for data fusion support in MYSEA and develop a proof-of-concept application to demonstrate that support.

14. SUBJECT TERMS Multilevel Security (MLS), Information Assurance (IA), Monterey Security Architecture (MYSEA), Data Integration			**15. NUMBER OF PAGES** 101
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited.

A PROTOTYPE OF MULTILEVEL DATA INTEGRATION IN THE MYSEA TESTBED

Andrew D. Portner
Civilian, Naval Postgraduate School
B.S., Catholic University of America, 2005

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2007

Author: Andrew D. Portner

Approved by: Cynthia E. Irvine, Ph.D.
Thesis Advisor

Thuy D. Nguyen
Co-Advisor

Peter J. Denning, Ph.D.
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Much of the information managed by government agencies is stored in databases. Mission Assurance objectives often require the synthesis of data from separate databases. *Data Integration* is used to address this need for combining sets of data. However, because many government organizations store data in databases with different syntactic characteristics and at different classification levels, they will necessarily want to combine data from these divergent data sources. This requires a secure system to ensure that sensitive information is not disclosed to unauthorized parties.

The Monterey Security Architecture (MYSEA) is an experimental and extensible distributed Multilevel Secure (MLS) computing. This project set out to determine if a data integration application could be supported by the MYSEA environment. Through research on MLS database architectures, existing data integration technologies, and previous work to implement applications on MYSEA, this project was able to both define a high-level design for data fusion support in MYSEA and develop a proof-of-concept application to demonstrate that support.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ABBREVIATIONS AND ACRONYMS

APS	Application Protocol Server
CGI	Common Gateway Interface
COTS	Commercial Off-the-Shelf
DBPS	Database Proxy Server
HTTP	Hypertext Transfer Protcol
ISA	Integration Stub Application
LAN	Local Area Network
MLS	Multilevel Secure
MYSEA	Monterey Security Architecture
NIC	Network Interface Card
RA	Remote Application
SAK	Secure Attention Key
SSD	Secure Session Daemon
SSS	Secure Session Server
STOP	Secure Trusted Operating Program
TCB	Trusted Computing Base
TPE	Trusted Path Extension
TPS	Trusted Path Server
TRSS	Trusted Remote Session Server

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank my advisors, Dr. Cynthia Irvine and Thuy Nguyen, for the clear guidance and input they provided over the course of this project. I would also like to thank Jean Khosalim for his technical assistance and Valerie Linhoff for bolstering my morale during the project.

This material is based upon work supported by the National Science Foundation, under grant No. DUE-0414102. Any opinions, findings, and conclusions or receommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. MOTIVATION

Databases have been used by business, government, and military organizations for years, as they provide an efficient way to organize, sort, manage, and retrieve large sets of data. As more and more data is stored, there arises a need for processing data from several different but related data sources at one time in order to reconstruct a complete view of the data.

One mechanism which has been developed to address this need is *Data Integration* [1], wherein data is retrieved from multiple data sources and automatically combined and condensed into a unified whole. This has a major advantage in that the data is presented to the user in a combined form without the need for altering the structure or management of the data sources where the data is stored. This is particularly attractive when one or more departments or agencies manage separate data store yet desire to share information. They are justifiably reluctant to cede authority or management rights to some other party, but if their information can be combined and presented to users in a useful format, the integrated data may provide users with better information for a wide variety of organizational processes.

Recently, there has been an increased call for government and military agencies to share information, as a way of increasing efficiency and enabling a more-complete picture of pertinent information [2]. Much of the information that such agencies deal with, however, resides at various classification levels, hence it is reasonable to conclude that they will necessarily want to synthesize data from several differing classification levels. In the context of Mission Assurance, there is great benefit to be had by moving to a unified cross-domain solution which will allow for such data processing.

The Monterey Security Architecture (MYSEA) is an experimental distributed multilevel secure computing environment. It integrates untrusted commercial off-the-shelf components with specialized high-assurance elements [3]. In the MYSEA

environment, multilevel servers provide isolation between security domains while allowing clients access to data within each domain as allowed by the security policy. It would be desirable to implement a data integration application in the MYSEA environment, in a way that does not compromise the security policy being enforced by the system.

B. PURPOSE OF STUDY

The objective of this thesis was to determine the feasibility of data integration in the context of the MYSEA environment. The thesis was to provide an answer to the following questions:

1. What modifications to the existing configuration are necessary to successfully implement data integration support on the MYSEA server?
2. What additional functionality is required to support the execution of data integration?

Through research on existing data integration technologies, MLS database architectures, and previous work regarding the development of user applications on the MYSEA server, it was determined that data integration can be implemented in the MYSEA environment. The project then set out to provide a proof-of-concept implementation to demonstrate the supported configuration. This involved the creation of a new trusted process, an integration stub application, and a Common Gateway Interface (CGI) for remote execution of the stub application to retrieve results for the user. Preliminary tests of the proof-of-concept implementation were also conducted.

C. ORGANIZATION OF THESIS

This thesis is organized as follows:

- Chapter I provides an introduction to the motivation and purpose of this thesis, including a brief presentation of the usefulness of data integration and its envisioned use in the MYSEA environment.

- Chapter II provides background information, including an in-depth explanation of data integration and a detailed discussion of the MYSEA environment and its pre-existing structures and processes.
- Chapter III describes the high-level requirements and design for data integration support in MYSEA. This chapter describes how a data integration application can be implemented in the MYSEA environment as a remote application, which was first described by Egan [4]. Data integration functionality and communication protocols to support the application are also described in this chapter.
- Chapter IV describes the details of the proof-of-concept implementation of the data integration application along with a description of the development environment used for the coding of that implementation.
- Chapter V describes the preliminary testing performed on the proof-of-concept data integration application.
- Chapter VI concludes with a summary of the project and suggestions for future work.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

A. INTRODUCTION

This chapter contains background information on databases, data integration, the Monterey Security Architecture (MYSEA), the Hyper Text Transfer Protocol, and the Apache-like web server used in MYSEA.

B. DATABASES

This section describes databases as data storage structures along with their usage in client-server environments and the security concerns they present in a multilevel environment.

1. Overview

A database is a data storage structure used to collect and organize information on similar items for easy retrieval. Each item is represented by a record with a number of fields containing the data of interest. The names and descriptions of these fields define the schema, or layout, of the database. This schema can then be used to form queries for retrieving a set of records from the database. There have been several languages developed for forming database queries [5][6][7][8], however even within the same language queries made for one database will not be understood by another database with a different schema. That is to say, queries made for one database may be used on another database with the same field names and descriptions, but it is more often the case that such field names and descriptions vary from one database to the next. In order to query multiple databases, queries in a canonical format must be translated into the proper schema for each data source.

There are several models of databases which describe how the records in the database are structured and used. The most prevalent models are the *hierarchical database*, where data is modeled as a tree of parent-child relationships, and the *relational*

database, which uses sets of tuples and predicate logic to organize and operate on the data [9]. This thesis will only address data integration architectural issues in the context of relational databases, the more-common of the two, though the same principles apply to all database types.

2. Client-Server Environment

Databases often store data which is useful to different persons or is accessed by several applications. In many cases, the database resides on a server which accepts queries from remote clients and returns the results. It is often convenient to have access to one or more databases mediated by a single database proxy server. The database proxy presents an interface to clients and allows them a simple way to access the databases it connects to. The database proxy acts as a server for database clients and as a client for the databases themselves. The proxy recieves queries from the database clients which it then relays to the databases on their behalf. When results are returned, the proxy presents them to the database client as though the client had interacted directly with the database. This has the advantage of avoiding a direct network connection to the database. By removing a direct connection to the network, the database is better protected against malicious users, who must now access the database through the approved, more-constrained interfaces.

3. Security Concerns

As databases are repositories of information that are accessed and used by multiple entities, there are generally two basic types of security concerns. The first concern is that an unauthorized entity could add to or change the information stored in the database. The second is that some entity or group of entities might read or otherwise learn information they are not authorized for.

These concerns hold for all databases in general and access controls are employed to ensure discretionary or role-based access rules are complied with; for example, an administrator may be able to add and delete records or tables whereas an ordinary user is only allowed to update existing records or present certain queries to the database. When

the environment includes information at multiple classification levels, access controls are complicated by the enforcement of a mandatory access control policy, which is not present in other systems. The concerns expand to include concepts of data leakage and signaling that are not pertinent in single-level environments. For example, a process must write a query to the database in order to read data from a database. This act of writing, even though its sole purpose is to read data, can create the means for a signaling channel between a high process and a low process. For a database system to be secure such channels must be absent, thus requiring that no process can read-down or that every process which queries a database is a trusted process.

There are other broader issues associated with data retrieval in MLS systems and in databases more generally, such as aggregation and inference which leads to the user having information to which authorization to access may not have been provided [10]. Because of the complex nature of these problems and the difficulties of solving them with any kind of architectural mechanism, aggregation and inference lie outside the scope of this thesis.

C. DATA INTEGRATION

1. Problem Space

Databases can be useful for storing and organizing related bits of information in a way that makes the information easy to retrieve. However, this is complicated when a given task or problem requires access to multiple sets of data housed in separate databases, with different schema, that are managed by different entities and that may have over-lapping or contradictory information. It is necessary to retrieve the data from the various sources and compile it in such a way as to provide a single set of data without overlap or internal contradictions. *Data integration* is a method for addressing this complication.

It is important to note that the databases to be used by the data integration application are not necessarily under the control of a single department or agency nor are they necessarily within the same organization as the integration application. The

information is shared between departments and agencies, but control over the content and maintenance of the databases is not ceded. This disqualifies possible solutions of reorganizing or restructuring data in the databases and requires that the data integration application adjust to the databases it relies upon.

2. Conceptual Operation

The intention behind data integration is to take a single query from the user and retrieve data from multiple sources based on that query. This data is stored in different forms and under different schemata, and managed by separate departments or agencies. The data integration application will itself have its own schema through which queries must be translated to the schemata of the different databases from which data is retrieved. The data integration application acts much like a regular database proxy in that it receives queries from a user and then acts as a client in order to retrieve information from several data sources; however there are a few differences.

The most apparent difference is that the data integration application must be able to handle data stored under multiple schemata; although there are database proxy servers which manage collections of physical databases, these all share the same schema. Additionally, unlike a regular proxy the data integration application must handle data that may overlap or be contradictory; the application will have to have some set of rules to account for this and produce unified results. Finally, it is notable that while a regular proxy might be a general application and allow for adding and modifying data, the data integration application is only designed to retrieve and coalesce data. The operation of the integration application is to accept a query, retrieve data from each data source, process the data to resolve conflicts or overlaps, and finally return the results to the client.

3. Security Concerns

Since the data integration application acts much like a database proxy the concerns are much the same. However as the data integration application does not update or add data to the database the concerns of the integration application are only a subset of the concerns of database proxies in general. The major concern is that of data leakage,

where entities may learn or read data for which they do not have authorization. This concern covers such threats as covert channels and weak configurations or software exploits that would allow privileged information or queries to be processed and returned. There is some additional concern over the integrity of the data returned by the application. Since part of the integration application's function is to alter the data it receives in order to deal with overlaps and contradictions in the data sources, the data returned from the integration application can never be of higher integrity than the application itself [11]. That is, although the data integration application retrieves information from outside sources which may have high integrity, the function of the application is to manipulate that data and so everything that the client receives effectively originates with the integration application. If the integration application can only be relied on to produce data at a certain integrity level, all of its results must be regarded no higher than that level, regardless of the integrity level of the source. Necessarily, if the data integration application retrieves data from a source of lower integrity, the results of the integration must be regarded as no higher than that lower level.

D. THE MONTEREY SECURITY ARCHITECTURE

1. Overview

The Monterey Security Architecture (MYSEA) is a distributed multilevel secure (MLS) networking environment designed to provide enforcement of mandatory security policies. The system consists of a small number of high-assurance components which enforce the mandatory policy combined with COTS products which provide the productivity applications required by the users. In addition to the MYSEA clients, TPE, and MLS Server, described below, the architecture can also support and service legacy single-level networks through specific ports from the Network Interface Card (NIC) through the use of a Trusted Communication Module, one of the high-assurance components of the architecture. Figure 1 shows the physical layout of the MYSEA environment.

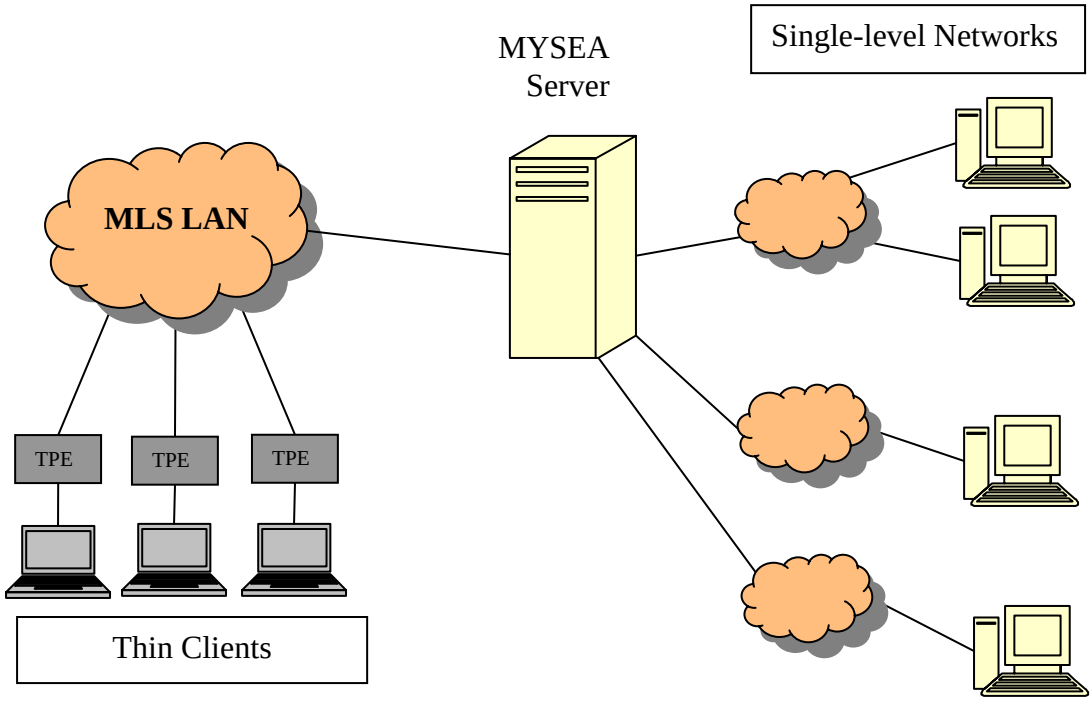

Figure 1. The MYSEA Testbed

2. MYSEA Server

MYSEA server is a high-assurance system, currently running on the BAE XTS-400 Trusted Computing System using the BAE Secure Trusted Operating System (STOP). STOP provides the enforcment of mandatory access controls that comply with the rules of the Bell-LaPadula model [12] for secrecy and the rules for the Biba model [13] for information integrity. This high-assurance server works in conjunction with untrusted commercial application protocol servers to provide services to the MYSEA clients.

A key component of the high-assurance MYSEA server is the Secure Session Server (SSS). When a client requests a service from the MYSEA server, if that service is one of the allowed services supplied by the MYSEA server, the SSS determines the session level of the client and uses that information to launch instances of untrusted

application server processes running at the same security level as the requesting client. These untrusted processes then handle the requests of the user.

3. MYSEA Client

A MYSEA client is composed of a COTS personal computer and a Trusted Path Extension (TPE). The latter is juxtaposed between the client machine and the LAN and facilitates a secure, unforgable authentication with the MYSEA servers via a trusted path. This allows the user to login to the MYSEA servers and set the security level of the current session. The client machine runs commercial operating systems and application software familiar to users as the motivation is that "unless a secure system offers users comfortable and familiar interfaces for handling routine information, it will fail due to lack of user acceptability" [3].

The client machines are thin, stateless machines in order to address the issue of object reuse, i.e., the possibility of residual data remaining in memory between sessions. The client machines load their operating system and various applications from read-only disks or from the MYSEA server into volatile memory; this memory is purged when a user logs out or changes session level. Such a configuration allows for the use of COTS applications, providing familiar tools to the user and simplicity in purchacing end-user applications.

4. Trusted Path Extension

The Trusted Path Extension (TPE) is a trusted physical device that is juxtaposed between the MYSEA client machine and the MYSEA network. It is physically and logically associated with the client and is responsible for providing a secure, unforgeable connection with the Trusted Computing Base (TCB) of the MYSEA server. The TPE facilitates user login and authentication and supports setting the security level of the session. Without this authentication, a client can not connect to the MYSEA network.

5. Trusted Remote Session Server

Because MYSEA clients are thin, stateless machines, it becomes necessary to have a mechanism for providing server-resident client applications (RAs). The Trusted Remote Session Server is the trusted component of MYSEA that manages requests for RAs and manages communications between the RA and the client. The MYSEA server creates a new TRSS process for each established session and the TRSS manages all RAs for the TPE associated with that session.

There are two major motivations for the TRSS. First, the XTS-400 on which the MYSEA server is built only allows trusted applications or applications running as the network user to access network ports. Because the remote applications requested by the client must run at the security level associated with the client it is necessary to have a trusted process to provide access to the network ports. Additionally, the TRSS manages communications on a message-by-message basis to ensure that the RA only makes connections to peers at the same session level [14].

When such an untrusted remote application requires network access, it signals to the TRSS parent. The TRSS parent then spawns a child to handle network communications on behalf of the RA. When the RA requests a connection to some peer, the TRSS child looks the peer up in the Peer Level Database, a static database which indicates the level of different peer machines. A connection is then allowed or denied based on the level of the RA and the level of the peer. When all of the RA's sockets have been closed, or a terminate signal is received from the TRSS parent, the TRSS child terminates the RA.

E. HYPERTEXT TRANSFER PROTOCOL

The Hyper Text Transfer Protocol (HTTP) is a client/server application protocol which runs on top of the standard TCP/IP connection-oriented network protocol [15]. HTTP has been widely used over the Internet in support of the World Wide Web and has been one of the most used protocols on the Internet. The HTTP protocol defines messages which can be sent between a client and a server in order to request and transfer

data; in most cases the client is a web browser and the data requested can be text, images, audio files, or any other data which might be sent over the Internet.

F. APACHE WEB SERVER

This section describes the Apache server followed by a description of the Apache-like *httpd* which is in use on the MYSEA server. The Apache Project an open-source project which has developed and maintains the popular Apache Web Server. The Apache-like *httpd* was adapted from a standard Apache Web Server for operation on the MYSEA server by Bersack [16].

1. Standard Apache

The Apache HTTP server project is a "collaborative software development effort aimed at creating a robust, commercial-grade, featureful, and freely-available source code implementation of an HTTP (Web) server" [17]. The project is managed by volunteers around the world, and accepts contributions of code and documentation from its user community. In 2007, a survey found that over 50% of web pages were served by the Apache Web Server, making it the most widely-used web server by a large margin [18].

The main appeal of the Apache server is its flexibility and ease of adaptation. As an open-source project, the source code is readily available to developers allowing for in-house modifications of the compiled system [19]. In addition to this, the server is designed to be extensible through custom-made modules, many of which are also freely available along with the Apache server code. To use a module, all that is required is that the module be compiled and the Apache configuration file modified to include it when the server is started.

In a typical environment, the Apache server runs as a daemon process. It accepts incoming HTTP requests and spawns child processes to handle each connection. This allows the parent process to continue listening for other incoming connections while the child processes service the requests that have already been received.

2. Apache Adaptation for MYSEA

An Apache-like web server was chosen for use in MYSEA because of the flexibility of use and modification noted above. Because of the security constraints on the MYSEA server, it was necessary to have a web server that could be modified to fit certain needs. In particular, the server would have to be invoked by the trusted SSS child process when a client made a request for HTTP services. This is because the MYSEA server does not allow untrusted processes access to network ports, and the web server would have to run as the user ID and session level of the requesting client to maintain access control policies on the different web documents. Communications between the client and the Apache-like server flows through the SSS child via a "pseudo socket" mechanism implemented for the purpose of supporting network communications from untrusted processes. Because of this, the web server needed to be a stand-alone application that handled requests directly rather than spawn child processes to do so, it needed to be activated by another application, and it needed to communicate using pseudo-sockets rather than directly to network sockets. All of these requirements were met through modification of the configuration file and altering the Apache source code [16]. The most important aspect of this design is that the Apache-like server runs at the session level of the client and is allowed read-down permissions, in accordance with the mandatory security policy, allowing a user logged in at a higher security level to view less-sensitive documents without having to change their session level.

III. REQUIREMENTS AND DESIGN

A. INTRODUCTION

To implement data integration where the data to be integrated has several classification levels, it is necessary to ensure that communications and functions can be performed without compromising the mandatory security policy. That is, the application will be invoked by the user, and thus it should run at the user's current session level. However, the application will need access to data sources dominated by the user's session level, which presents concern regarding write-down operations by the application. The application must also work within the constraints of the secure operating system, i.e., STOP, that the MYSEA server runs on. Hence there are security issues associated with the communications necessary for the data integration application to perform it's function.

This chapter describes a high-level design that utilizes the existing MYSEA Remote Application support to demonstrate support for data integration in the MYSEA architecture. Because of the concerns and constraints noted above, a decomposition of data integration functionalities was identified that allowed for MLS data access while minimizing the amount of trusted code introduced to the system. This decomposition results in two new processes: the trusted Database Proxy Server (DBPS) daemon and the untrusted Integration Stub Application (ISA). This chapter presents a design for both the 'end goal' data integration application and for a proof of concept application which was implemented for this project. This work builds upon existing MYSEA support for Remote Applications, which was discussed by Cooper [14] and Egan [4].

B. TOP LEVEL REQUIREMENTS

The design for the prototype application involved decomposing the conceptual operation of data integration in order to separate out the functions which require special privileges from those which can be untrusted. It was determined that separating the

functions of a database proxy from those required for data integration would result in the trusted code required to access data sources at different security levels, whereas the rest of the application may run untrusted as a remote application at the session level of the user.

The trusted database proxy has the following requirements:

- The proxy shall accept data requests from a remote application.
- The proxy shall verify the security level of the remote application prior to granting access to the data sources.
- The proxy shall be able to interface with a heterogeneous set of backend data sources.
- The proxy shall query appropriate data sources based on the security level of the remote application.
- The proxy shall have read and write access to data sources of differing security levels.
- The proxy shall mark each of the results with labels appropriate to the security level of the data source queried.
- The proxy shall keep a log of all requests.

The untrusted application code has the following requirements:

- The application shall run with the user ID and current session level of the authenticated user.
- The application shall be launched by the client.
- The application shall process results received from the trusted proxy and integrate the data to return to the user.

C. HIGH LEVEL DESIGN

1. Overview

MYSEA support for Remote Applications has been discussed previously [14] [4] and the same concepts and existing processes can serve many of the needs of data integration support. However, as the data integration application must access data sources at multiple sensitivity levels and standard remote applications run with the user ID and at the current session level of the user, data integration support requires the

addition of a new trusted process family, the Database Proxy Server (DBPS), to process requests to the data sources and to return the results to the untrusted application code. Figure 1 presents an overview of this architecture. The steps in a data integration request will now be described.

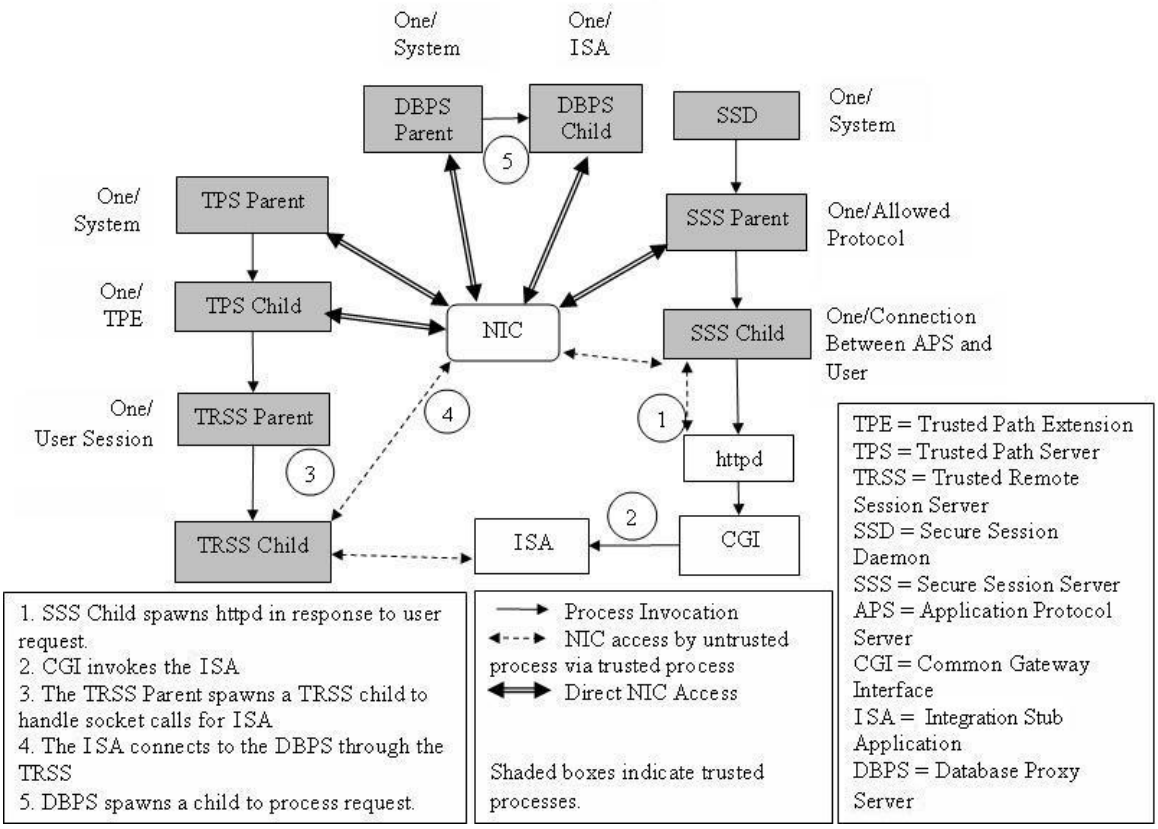

Figure 2. Process Overview (after [4])

At start up, both the Trusted Path Server (TPS) Parent and the Secure System Daemon (SSD) are launched as system daemons. The SSD launches a Secure Session Server (SSS) Parent process for each of the allowed protocols supported by the MYSEA server. The TPS Parent accepts login requests from remote clients and spawns a TPS Child to handle trusted path communications for each client. When a user successfully authenticates, the TPS Child launches a Trusted Remote Session Server (TRSS) Parent process to manage remote application requests for that user. The SSD, SSS, TPS, and TRSS processes are all trusted processes with access to the MLS network interface.

An authenticated user may open a web browser and request a page which will serve as an interface to the remote application. The SSS Child process that manages the HTTP protocol for the user receives the web page and spawns an untrusted *httpd* process with the user ID and at the current session level of the requesting user. This is the modified Apache-like application discussed in the previous chapter. Because the *httpd* process is untrusted, all network communications are mediated by the SSS Child process that spawned it. The user will select the link on the requested webpage for the remote application service and the *httpd* process will launch a Common Gateway Interface (CGI) script which will launch the requested remote application. This is the standard operation of a Remote Application as described in [4]. For data integration, the Integration Stub Application (ISA) will be launched as the remote application.

The design of the data integration system required the addition of another trusted process family, the Database Proxy Server (DBPS). The DBPS is run as a system daemon, i.e., is started during system start up, and is permitted to access the MLS network interface. After the ISA is launched, it first requests a MYSEA socket from the TRSS Parent and uses that socket to communicate with the DBPS Parent. The TRSS Parent spawns a TRSS Child to mediate communications for the ISA through the MYSEA socket mechanism. The ISA then makes a connection to the DBPS Parent which will verify the session level of the ISA and spawn a DBPS Child to query the appropriate data sources. The DBPS Child consolidates the results, returns them as a unit to the ISA, and closes the connection, at which point the ISA completes its data retrieval task. The ISA "integrates" the data and returns the integrated results via the *httpd* process. Through this request process, the *httpd*, CGI, and ISA are single-level processes with the user ID and current session level of the requesting user; the TRSS and DBPS processes are trusted processes.

The processes involved in the data integration communications are described in the following sections. The processes involved in general remote application operation have been described in [4] but are presented here in order to fully describe the context in which the data integration application operates. The processes which were developed for this project are the DBPS Parent and Child and the ISA.

2. Processes

a. Trusted Path Server (TPS) Parent

The Trusted Path Server Parent is a system daemon that runs at the startup of the MYSEA server. The TPS Parent is responsible for accepting trusted path session requests from TPEs, validating that the TPE is allowed to connect to the MYSEA server, and if the TPE is validated spawning a TPS Child process to handle the request. When a trusted path session request is received, the TPS Parent checks for the requesting TPE's identifier in the Allowed TPE Database. If the identifier is found, then a child process is spawned to handle the trusted path communications with that TPE; otherwise, the connection is dropped. The TPS Parent then continues to accept and validate session requests from other TPEs [4].

The TPS Parent accesses the MLS LAN and so runs at the level of the MLS LAN. Since it must write to system-low databases it is designated as a trusted process [4].

b. Trusted Path Server (TPS) Child

The TPS Child process is spawned by the TPS Parent process once a TPE has been validated, and the child process handles all of the trusted path session communications for that TPE. This includes requests to log in, set the session level, run the session, and log out. The TPS Child is also responsible for launching the TRSS Parent process, which will manage remote application requests for the client. When the TPS Child receives data from the client, which begins with a Secure Attention Request Packet (SARP), it reads and processes the client’s command and returns the output before waiting to receive the next command [4].

The TPS Child runs at the level of the MLS LAN and is designated at a trusted process because it requires access to system-low databases as well as the system's user identification and authentication information [4].

c. Secure Session Daemon (SSD)

The Secure Session Daemon is a system daemon that runs at the startup of the MYSEA server. It is responsible for spawning SSS Parent processes for each of the protocols the server supports, as listed in the Allowed Protocols Database. For each of these protocols (e.g. HTTP, SMTP, IMAP, etc.), it spawns an SSS Parent process to handle requests for service using that protocol [4].

The SSD process terminates either when all of its spawned processes terminate or when it is interrupted by a local Secure Attention Key (SAK). In this second case, the SSD is responsible for signaling each of its SSS Parent processes to terminate as well. The SSS Parent processes require privileged ports requiring them to run as the *network* user, and so the SSD must also run as the *network* user in order to spawn and terminate them. This requires that the SSD be designated as a trusted process [4].

d. Secure Session Server (SSS) Parent

The Secure Session Server Parent processes are spawned by the SSD, and each is responsible for accepting and validating requests for a specific allowed protocol. When a service request is received by the SSS Parent, it first checks to see if the requesting client has an established session by searching for a matching entry in the User Database. If no such entry is found, the SSS Parent checks for a valid remote application connection in the Remote Connections Database. If the request is associated with either an established trusted session or a remote application connection, then an SSS Child process is spawned to handle the request; if not, the connection is dropped. The SSS Parent then continues to accept and validate other requests [4].

The SSS Parent runs at the level of the MLS LAN, but must also switch to the *network* user in order to access privileged ports. Additionally, it must spawn children with MAC and DAC exemptions and the ability to change user IDs, which requires the Parent to have such privileges. Therefore, it is designated as a trusted process [4].

e. Secure Session Server (SSS) Child

The Secure Session Server Child handles all of the service requests for a certain protocol from a specific user or remote application. It is responsible for launching the appropriate Application Protocol Server (APS), which runs at the session level of the given TPE or remote application. Since the APS is untrusted it does not have access to the MLS interface and must rely on the SSS Child that spawned it to make socket calls on its behalf. Communications between the APS and the SSS Child are handles through a MYSEA socket (MSKT) allocated by the TPS Child process. When the APS requires a socket call, it enters its request into the MSKT database and signals the SSS Child; the SSS Child retrieves the request from the database, makes the requested call, and enters the return code and any retrieved data into the database. The SSS Child then signals the APS, which retrieves the results from the database [4]. The MYSEA socket interface is such that these functions appear to the APS to be ordinary socket calls.

Each time the APS requests a connection to a peer (using the *connect*, *sendto*, *accept*, or *receivefrom* socket calls), the SSS Child must verify that the connection is allowed by comparing the session level of the peer with that of the APS. The current session level of the peer is retrieved by consulting the Peer Level and User Databases. If the connection is allowed, the SSS Child consults the Source Address Binding Database to determine the source address that is configured for use with the supplied destination before making the connection. Each time a new connection is established on behalf of an APS, the SSS Child updates the Remote Connections Database with the security level of the new connection; when the connection is closed, the SSS Child removes the entry from the database [4].

The SSS Child must run at the level of the MLS LAN, but it is also required to communicate with the APS, which runs at the session level of the remote user, and it must change it's user ID in order to launch the APS process and initialize the MSKT database with the proper user ID. Therefore, the SSS Child is designated as a trusted process. [4]

f. Application Protocol Server (APS)

The Application Protocol Server is the untrusted process which handles the server side of client/server communications. For the purposes of this project this is the Apache-like *httpd* which was described in [16] and discussed in the previous chapter. The *httpd* has been modified in order to use the MSKT interface so that it can access the MLS LAN through the SSS Child [4].

The *httpd* (as is each APS process) is an untrusted process that runs with the user ID and session level of the requesting user [4].

g. CGI Remote Application Invocation

The CGI remote application invocation process is initiated by the *httpd* in response to the user's request to launch an RA from the web interface. The process is spawned by the *httpd* and is responsible for invoking the RA and packaging the results to be returned to the user by the *httpd*. This is the process that will invoke the Integration Stub Application. It is an untrusted process which runs with the user ID and session level of the *httpd*, which is itself running as the user and session level of the requesting user [4].

h. Trusted Remote Session Server (TRSS) Parent

The TRSS Parent process is launched by the TPS Child process once a user has been authenticated. The TRSS parent is responsible for managing requests from remote applications associated with the authenticated user. Specifically, the TRSS Parent spawns a TRSS Child process for each RA that requests the use of MYSEA sockets [4].

The TRSS Parent initializes by attaching to the databases it requires for its open operation as well as the databases which will be required by its children. When the TRSS Parents process receives a signal from an untrusted process, it searches the MSKT Database for an entry with the call type of *new socket call* and spawns a TRSS Child process to handle the request for a new socket and all future socket requests from that remote application. The TRSS Parent then sets the *trusted PID* field in the MSKT

Database entry to the process ID of the newly spawned child and then returns to waiting to be signaled by other remote applications [4].

The TRSS Parent itself does not access the MLS LAN but the children it spawns do, therefore the TRSS Parent must run at the level of the MLS LAN. However, the TRSS Parent must switch to the level and ID of the user requesting remote application services before it initializes the MSKT Database so that the RAs, which run at the user's session and ID level, will be able to read and write to the database. Additionally, the TRSS Parent must receive signals from the RAs which are running at the user's level. Because of these requirements, the TRSS Parent is designated as a trusted process [4].

i. Trusted Remote Session Server (TRSS) Child

The TRSS Child is spawned by the TRSS Parent to handle socket calls on behalf of a particular remote application. Communications by untrusted processes which require socket calls are done through a MYSEA socket (MSKT) allocated by the TPS Child process. After the initial call has been received by the TRSS Parent and the TRSS Child has been initiated, whenever the RA requires access to the MLS interface it enters its request into the MSKT Database and signals the TRSS Child. The TRSS Child retrieves the requested call from the database and makes the appropriate socket call, entering the return value and any retrieved data into the database, then signals the RA. The RA then retrieves the results from the database [4].

Each time the RA requests a connection to a peer (using the *connect*, *sendto*, *accept*, or *receivefrom* socket calls), the TRSS Child must verify that the connection is allowed by comparing the session level of the peer with that of the RA. The current session level of the peer is retrieved by consulting the Peer Level and User Databases. If the connection is allowed, the SSS Child consults the Source Address Binding Database to determine the source address that is configured for use with the supplied destination before making the connection. Each time a new connection is established on behalf of an RA, the TRSS Child updates the Remote Connections

Database with the security level of the new connection; when the connection is closed, the TRSS Child removes the entry from the database [4].

The TRSS Child runs at the level of the MLS LAN. It also requires access to privileged ports, and must communicate with the RA process which runs at the session level of the user which invoked it. Therefore, the TRSS Child is designated as a trusted process [4].

j. Integration Stub Application (ISA)

The Integration Stub Application (ISA) is one of the new processes implemented for this project. The ISA is the component of the data integration application which can run as untrusted code. This includes functions regarding accepting user input, performing the actual integration of the data, and returning results to the user. Because data integration requires access to multiple data sources and which may have different security levels, it is necessary to have a trusted process for such functions. Therefore, the ISA relies upon the trusted Database Proxy Server (DBPS) for access to the data sources.

The ISA itself behaves in much the same way as other remote applications already supported by MYSEA. It is invoked by a CGI script which is launched by the *httpd* in response to a user's request. The ISA is untrusted and runs at the session level of the requesting user. Communications with the DBPS are done through the MYSEA socket interface, mediated by the TRSS processes. This is a useful design in that it would allow the ISA and DBPS to reside on separate servers in a federated environment. The communications between the ISA and the DBPS are described in Section 4 below.

When the ISA is launched, it makes a request to the TRSS Parent for a new socket, and then requests a connection with the DBPS. The ISA sends a request for data based on parameters provided by the user to the DBPS, processes the returned data based on its integration algorithm, and returns the results to the user.

k. Database Proxy Server (DBPS) Parent

The Database Proxy Server (DBPS) Parent is a new process implemented for this project. To support multilevel data integration, it was necessary for an application to have access to data sources which would have different security levels associated with them. Although data integration only requires data to be retrieved from the database, rather than adding or manipulating data in the data source itself, the act of requesting data constitutes a write and could be used to covertly transmit information from high to low security levels. Therefore, processes that retrieve data from the data sources must be trusted. In order to minimize the amount of trusted code introduced into the system, the only function of the DBPS is to retrieve data from the data sources on behalf of the ISAs.

The DBPS Parent is responsible for accepting and validating connection requests from ISAs. When a connection request is received, the DBPS Parent searches the Remote Connection Database for an entry associated with the requesting ISA. If there is no such entry, the connection is dropped. If an entry is found, the DBPS spawns a DBPS Child to handle the requests from that ISA, passing that child the session level information of the ISA, and then returns to listening for other connection requests. These communications are described in Section 4 below.

The DBPS Parents runs at the level of the MLS LAN and needs no special permissions to read data from the Remote Connection Database. However the children it spawns must have access to data sources at multiple levels as well as communicate with an ISA running at the session level of the user requesting service. Therefore the DBPS Parent is designated as a trusted process.

l. Database Proxy Server (DBPS) Child

The DBPS Child is spawned by the DBPS Parent to handle requests from a particular ISA. Once the connection between the DBPS Child and the ISA is established, the DBPS Child receives query information from the ISA. The DBPS Child then makes queries to the data sources it is connected to with security levels equal to or

less than that of the requesting ISA. The DBPS Child informs the ISA of the amount of data that was retrieved and then returns the results.

The DBPS Child acts as both a database server for the ISA and as a database client for the external data sources; this is precisely the definition of a database proxy. The communications between the DBPS Child and the ISA, and between the DBPS Child and the external data sources, are described in Section 4 below.

The DBPS Child runs at the level of the MLS LAN, but is required to send queries to data sources of different security levels, as well as communicate to an ISA running at the session level of the requesting user. Therefore the DBPS Child is designated as a trusted process.

Table 1 presents a summary of the processes involved in executing the data integration application.

Process	Trusted?	Session Level	Launched By	Instances
TPS Parent	Yes	MLS LAN	(System Daemon)	One per system
TPS Child	Yes	MLS LAN	TPS Parent	One per TPE
SSD	Yes	MLS LAN	(System Daemon)	One per system
SSS Parent	Yes	MLS LAN	SSD	One per Allowed Protocol
SSS Child	Yes	MLS LAN	SSS Parent	One per APS-Client connection
APS (*httpd*)	No	Remote User	SSS Child	One per SSS Child
CGI Invocation	No	Remote User	APS (*httpd*)	One per user RA request

Process	Trusted?	Session Level	Launched By	Instances
TRSS Parent	Yes	MLS LAN	TPS Child	One per user per session level
TRSS Child	Yes	MLS LAN	TRSS Parent	One per remote application
DBPS Parent	Yes	MLS LAN	(System Daemon)	One per system
DBPS Child	Yes	MLS LAN	DBPS Parent	One per ISA
ISA	No	Remote User	CGI Invocation	One per user request

Table 1. Process Attributes (after [4])

Table 2 summarizes the privileges required by each trusted process.

Trusted Process	Privileges Required
TPS Parent	• Ability to access the MLS LAN as well as system-low databases.
TPS Child	• Ability to access the MLS LAN as well as system-low databases. • Ability to access user identification and authentication information.
SSD	• Ability to create and terminate child processes that run as the *network* user.

Trusted Process	Privileges Required
SSS Parent	• Ability to run as the *network* user in order to access privileged ports. • Ability to create child processes that can access the MLS LAN and also communicate with user-level RA processes. • Ability to create child processes that can switch user IDs.
SSS Child	• Ability to access the MLS LAN and also communicate with user-level APS processes. • Ability to switch to the user ID of the remote user before initializing the MSKT Database and launching the APS process.
TRSS Parent	• Ability to switch to the user ID of the remote user before initializing the MSKT Database. • Ability to create child processes with access to the MLS LAN and also communicate with user-level RA processes. • Ability to create and terminate child processes that run as the *network* user. • Ability to be launched by low-integrity processes (refer to [4] for details).
TRSS Child	• Ability to access the MLS LAN and also communicate with user-level RA processes. • Ability to run as the *network* user in order to access privileged ports.

Trusted Process	Privileges Required
DBPS Parent	• Ability to create child processes that can access the MLS LAN and communicate with system-low databases. • Ability to create child processes that can access databases of different security levels.
DBPS Child	• Ability to access the MLS LAN and communicate with system-low databases. • Ability to access databases of different security levels.

Table 2. Privilege Requirements of Trusted Processes (after [4])

3. Databases

a. Remote Connection Database

The Remote Connection Database is a database that is used to bind connections from an RA or APS to a specific user and session level. There is one Remote Connection Database per system; it is initialized by the TPS Parent and is updated by an individual TRSS Child or SSS Child each time their associated RA or APS establishes or terminates a connection. Each entry consists of a source port, source IP address, destination port, and destination IP address, along with session information [4].

The Remote Connection Database is used by the DBPS Parent to verify that a connection request is coming from an ISA associated with a valid user session. Entries in the Remote Connection Database include information on the source port, source IP address, destination port, and destination IP address; these values uniquely identify a socket. If an entry is found matching the socket being used by the ISA, session data is retrieved and passed to a DBPS Child which will handle the connection; otherwise, the connection is dropped [4].

b. Peer Level Database

The Peer Level Database is a database used to indicate whether or not a peer is multilevel and, if not, to associate the peer IP address with a specific security level. There is a single, static Peer Level Database system [4].

The database is used by the TRSS Child and SSS Child to determine if a given connection request is allowable. When an RA or APS requests a connection to a peer, the TRSS Child or SSS Child looks up the peer in the Peer Level Database and retrieves the security information for that peer. If the peer is multilevel or at the same security level as the requesting process then the connection is allowed, otherwise it is dropped. Since the ISA functions as an RA, there must be an entry in the database to indicate that the DBPS is a multilevel peer [4].

c. Source Address Binding Database

The Source Address Binding Database is a static, per-system database used to associate one or more destination IP addresses with a source IP address. Because of the way the SSS Parent recognizes and handles incoming traffic, the connection must be registered by the TRSS process with a known source port, source IP address, destination port, and destination IP address. This becomes a problem for certain socket calls that do not bind to a source IP address before attempting to establish a remote connection. A TRSS Child may account for this problem by consulting the Source Address Binding Database to find an appropriate source IP address for a given destination IP address [4].

Since the ISA relied on the TRSS Child to process its socket calls, and since the ISA must make a connection to the DBPS, there must be an entry in this table to indicate the appropriate address pair [4].

4. Communication Protocols

a. Overview

To initiate a data integration request the user fills out and submits an HTML form on the web page for the data integration remote application. Submitting the form launches the CGI Invocation process which in turn launches the ISA process, passing the form inputs as key-value pairs in command-line arguments. These command –line arguments are accepted by the ISA as string values. The ISA makes a connection with the DBPS Parent which, if the ISA is associated with a valid user session, spawns a DBPS Child to handle the ISA's request. The ISA sends the key-value pairs to the DBPS Child and the DBPS Child uses these pairs and the session-level information it received from its parent to retrieve data that is dominated by the ISA's session level. Once the DBPS Child has made all of its queries, it packages the results into a single data structure to be returned to the ISA. The ISA parses the returned structure, integrates the results it contains, and returns these integrated results to the user via the CGI script.

The nature of the communications required between the ISA and the DBPS lend themselves toward a connection-oriented strategy. For this reason it was decided that the communications should be made on top of TCP/IP, rather than through a UDP protocol or some other method. The use of TCP/IP allows the communications to be discussed as a series of messages. Additionally, the management of the connection, the set-up and tear-down before and after communications is offloaded to standard socket function calls.

Figure 3 shows the message flow of the protocol between the DBPS and the ISA, after a TCP connection has been established.

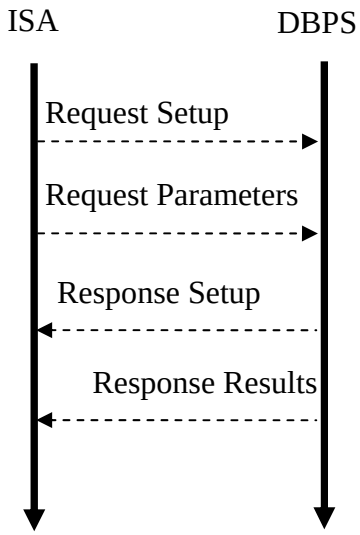

Figure 3. Message Flow Between DBPS and ISA

Header information is required for the messages sent between the DBPS and the ISA. This header consists of the protocol version being used, the message type, and the size of the payload. This general information is necessary to coordinate communications between the DBPS and the ISA. The version field is 8 bits long, as is the message type field which denotes either "Request Setup," "Request Parameters," "Response Setup" or "Response Results." The last header field is a 16-bit field which holds the size of the payload. Following this header information is the message payload. Figure 4 shows this generic packet format. For the Request Setup and Request Setup the payload is a simple 32-bit integer indicating the total size in words (including header information) of the following message, either a Request Parameters or Response Results. This size information is then used by the recipient to dynamically allocate and appropriate-sized buffer to hold the incoming data. In this way both the DBPS and ISA expect a message of known size (64 bits total for the header and payload) which will establish the size of the variable-length message that is to follow and thereby avoid buffer overflows. Figure 5 shows the packet format used for the Request Setup message and Figure 6 shows the format for the Response Setup message. In a Request Parameters or Response Results message, the message payload is a variable-sized block of data

containing either null-terminated strings or a results unit composed of security labels and database result sets. The form of these parameter and result payloads is described in the ISA and DBPS Child sections below.

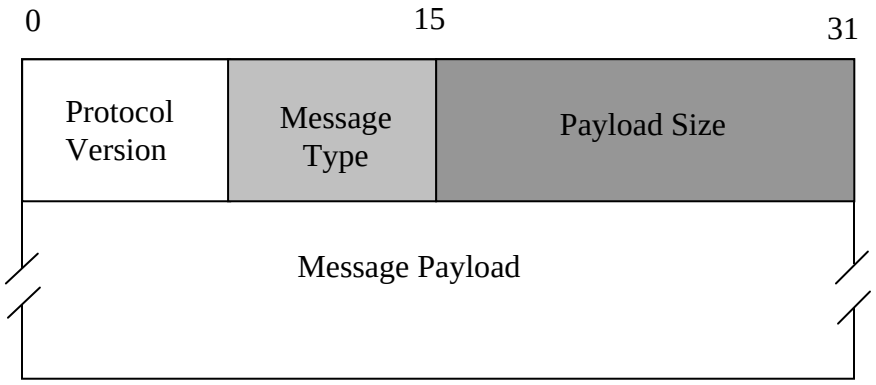

Figure 4. Generic Message Format for DBPS-ISA Protocol

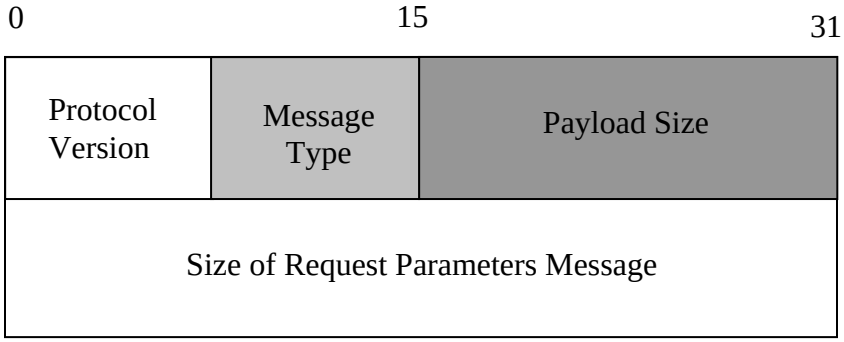

Figure 5. Message Format for Request Setup

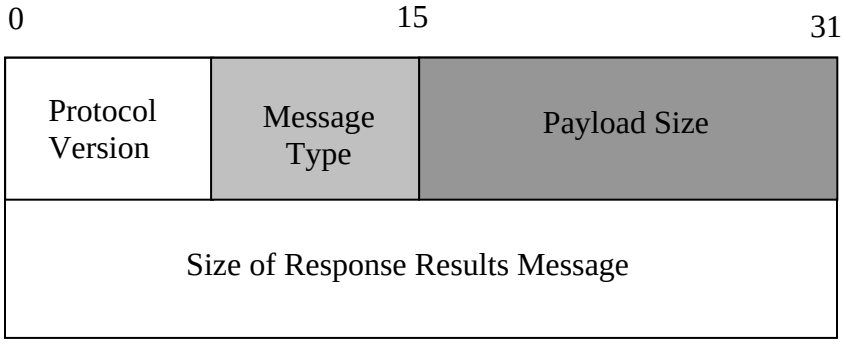

Figure 6. Message Format for Response Setup

b. CGI Invocation

The CGI Invocation script is launched by the *httpd* process when the user submits a request to invoke the ISA. The script uses the Perl 'backtick' operation to launch the ISA as a command-line application. The key-value pairs that were entered into the HTML form are passed to the CGI script when it is called by the *httpd*, and are used as command-line arguments to the ISA. The backtick operation also allows the Perl script to capture the standard output of the invoked application; this is how the CGI script receives results from the ISA, and those results are then returned to the user as part of a web page via the *httpd* process.

c. ISA

The ISA is invoked by the CGI script with the IP address of the DBPS and the key-value pairs that were input by the user as command-line arguments. These arguments are then stored in internal variables as string values. The ISA initiates a connection with the DBPS, through the mediation of the TRSS processes and the MYSEA socket interface. If the ISA is associated with a valid user session then a DBPS Child is spawned and the connection is completed; otherwise the connection is dropped. Once the connection is completed, the ISA sends a Request Setup message followed by a

Request Parameters message that includes the number of pairs and the key-value pairs, which are presented as a series of null-terminated strings, to the DBPS Child. The number of pairs is a 32-bit numerical value. The strings are arranged alternating key with value so that the value of a given key is the string following it. The Request Parameters message is a variable-length message and takes advantage of the connection-based nature of TCP/IP to ensure that the entire message is transferred and that the DBPS Child knows when the message has been received. Figure 7 shows this message format.

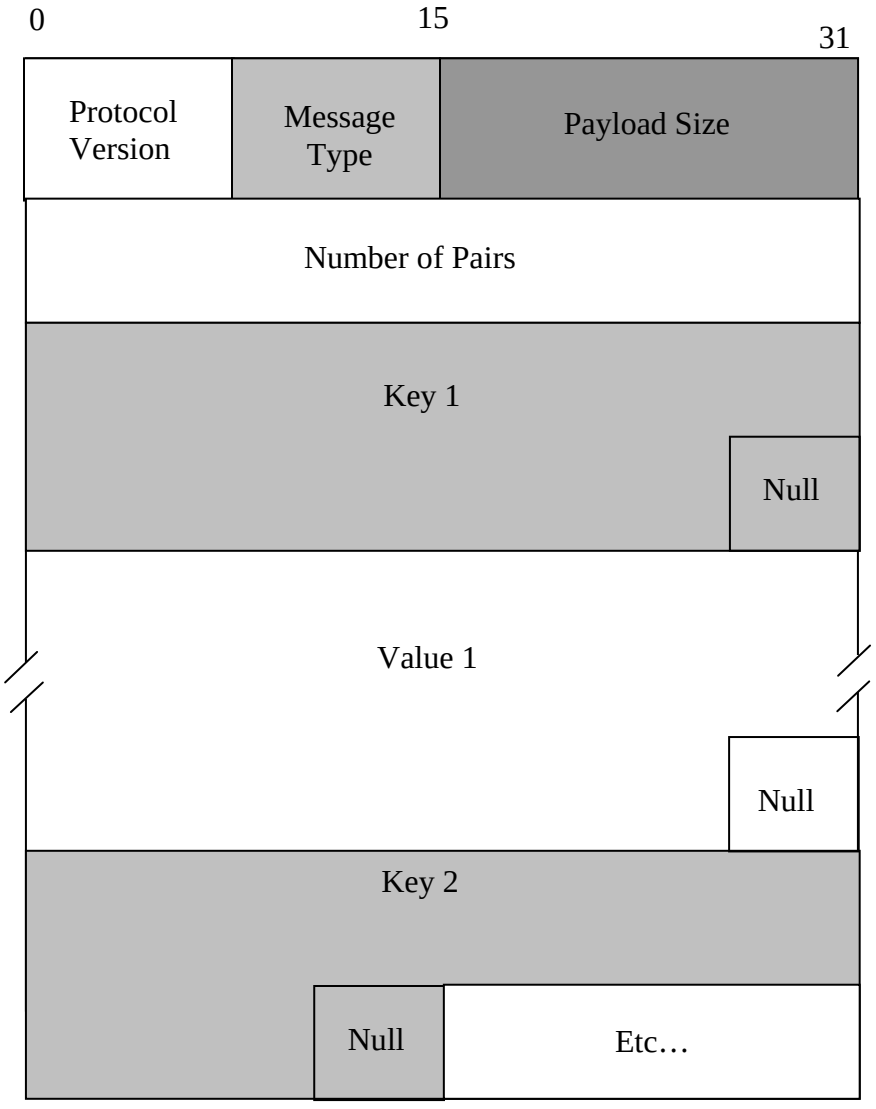

Figure 7. Request Parameters Format

The ISA then waits to receive the Response Setup and Response Results messages from the DBPS Child and, once they have been received, the ISA closes the connection. Included in the Response Results message are the size and security label for each result set returned by the DBPS. The form of this unit is described in the DBPS Child section below. The ISA uses the size information to allocate the appropriate amount of space to store each result set.

The ISA parses the data structure into its individual result sets in order to perform the integration function. The results of the integration are then printed to standard out, to be captured by the CGI script and returned to the user.

d. DBPS Parent

The DBPS receives a connection request from the ISA and immediately looks up the socket associated with the connection in the Remote Connection Database. The socket is identified by a four-value set of source IP, source port, destination IP, and destination port [4]. If the socket is found in the table then the ISA is associated with a valid user session and session-level information is retrieved for that ISA; otherwise the connection is dropped. Once session level information has been found for the connection it is stored in an internal variable and the DBPS creates a child process through the use of *fork( )*; as a result of the *fork( )*, the child inherits the connection to the ISA and the session-level information and continues execution from that line of code. The DBPS Parent then returns to listening for requests from other ISAs.

e. DBPS Child

The DBPS Child inherits the connection and session-level information from the DBPS Parent as internal variable when it is created by the use of *fork( )*. The first thing the DBPS Child does upon creation is wait to receive the Request Setup and Request Parameters messages from the ISA. The Request Parameters message contains the key-value pairs used for the database queries the DBPS Child is to make. These parameters are sent as a TCP/IP message in the form of a comma-delimited string of key-value pairs (see Figure 5 above). Once the pairs have been received, the DBPS Child

makes its queries. The session-level information is checked against a series of string comparisons to determine which set of data bases are dominated by the ISA's session level; once this is determined, the DBPS Child constructs SQL queries from pre-established statements and the addition of the key-value pairs received from the ISA.

The DBPS Child queries each of the data sources individually and attaches a binary security label to each of the returned result sets based on the classification level of the data source from which they were retrieved. All of the result sets and their associated labels are packaged together into a single unit. The unit begins with a 32-bit field indicating the number of result sets returned. Following this is the binary security label of the first set, the 32-bit size of the set, and the set itself. The result sets are kept in the form they were received by the DBPS from the external data source so that once they have been unpacked, they can be manipulated by the ISA as though the ISA had received them directly. Additional sets follow immediately, security label first then the size of the result set, and the result set itself. Figure 8 shows the format for the Response Results message.

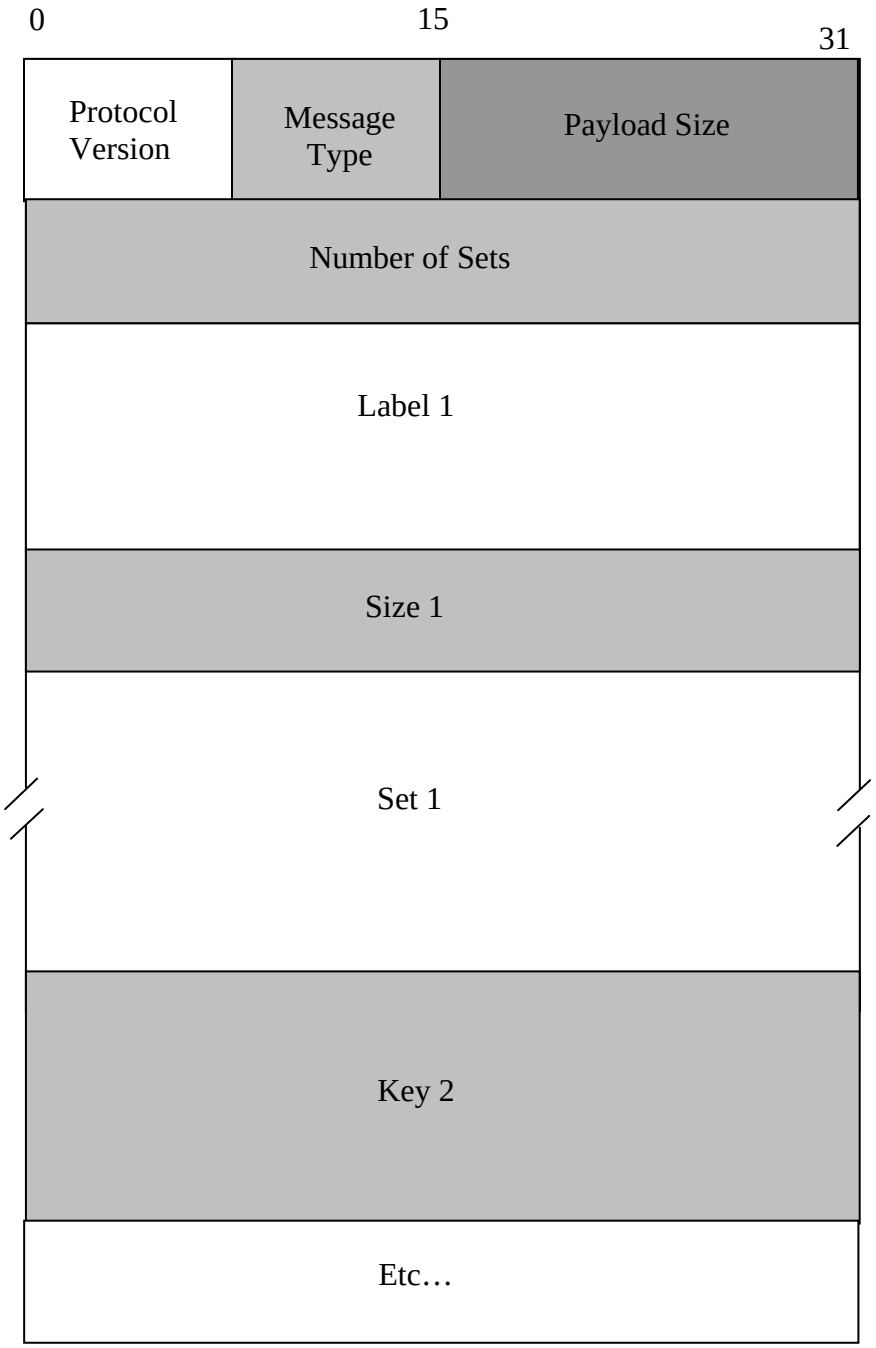

Figure 8. Response Results Format

The security label is a 96-bit binary value indicating the security level, integrity level, and any compartmental information associated with the result sets. Along with the potentially-large amount of data returned by each data source, the results unit

can quickly become a very large amount of data. Development of an efficient method of managing large amounts of networked data is presented as a topic for future work in Chapter VI.

Then the DBPS sends the packaged results themselves as a Response Results message, utilizing the connection-based nature of the TCP/IP connection to ensure the message is transmitted completely. Once the results have been sent, the DBPS Child closes the connection and terminates.

D. PRE-IMPLEMENTATION MODIFICATIONS TO DESIGN

Most of the processes and interfaces necessary to support data integration in MYSEA were developed as part of the design and implementation of remote applications in [14] and [4]. The Integration Stub Application operates identically to other remote applications and connects to the DBPS as a regular multilevel peer. Because of this, there is little in the existing design that needs to be modified to incorporate data integration. All that is required is to ensure that entries exist in the Peer Level Database and the Source Address Binding Database to indicate the necessary information required for the ISA to connect to the DBPS. Since this project developed both the ISA and the DBPS to run on the same server, the current MYSEA server configuration already includes entries in the databases with the information required.

E. OVERVIEW OF IMPLEMENTATION REQUIREMENTS

1. Newly Implemented Components

The following processes were implemented specifically for demonstrating the support for data integration in MYSEA:

- CGI Invocation Script for data integration

The CGI Invocation Script is launched by the *httpd* process in response to a user request for the ISA remote application. The script uses the Perl ‘backtick’ operator to launch the ISA and capture its output. The script passes the key-value pairs input by the user to the ISA as command-line arguments.

- Integration Stub Application (ISA)

The ISA runs as a remote application on the MYSEA server. It is invoked through a CGI script from a web page and runs with the ID and at the current session level of the requesting user. The ISA is an untrusted process.

- Database Proxy Server (DBPS) Processes

The DBPS Parent and Child processes are new trusted processes developed for this project. The DBPS Parent runs as a system daemon and validates connection requests from ISAs. If a requesting ISA is associated with a valid user session, a DBPS child is spawned to handle the request. Otherwise the connection is dropped.

The DBPS Child is spawned by the DBPS Parent to handle data requests from an ISA. The DBPS Child is trusted to make queries to data sources of different levels and return the results to the requesting ISA.

2. Modification to Existing Components

No existing components required modification for this project.

F. SUMMARY

The design specifications summarized in this chapter provide a complete framework for data integration support as a remote application in MYSEA. A prototype implementation based upon these requirements was completed for this project. The implementation process is described in Chapter IV.

IV. IMPLEMENTATION

A. OVERVIEW

This chapter describes the implementation of a proof-of-concept prototype that was developed as part of the demonstration of MYSEA support for data integration applications. The prototype is composed of a CGI invocation script, a simplified ISA and a simplified DBPS. While some of the specific functionality of a fully-developed data integration system are absent, none of the missing functions would require additional process architecture of inter-process communication mechanisms. The process organization and communication architecture required for data integration support has been completed with this project.

Each of the following sections are presented with a diagram of the system processes involved in the data integration application to show the context in which the components are designed to function. These figures are taken from the system-wide view of Figure 2 but are restricted to only the processes directly involved in data integration. Boxes with light shading indicate those which were developed for this project, and the darker-shaded box indicates the component currently being discussed.

Appendix A provides a listing of MYSEA source code files that were created for this project.

B. DEVELOPMENT ENVIRONMENT

All of the code involved in this project was developed on an XTS-400 system running the STOP 6 operating system. The ISA and DBPS processes were implemented in the C language and compiled using gcc 3.2. The CGI script to invoke the ISA was written in Perl and executed using version 5.8.0 of the Perl interpreter. Both the C compiler and Perl interpreter were the standard versions shipped with the XTS-400 system. This proof-of-concept data integration application was integrated into version 1.1x of the MYSEA server software.

C. IMPLEMENTATION DETAILS

1. DBPS Process Functional Detail

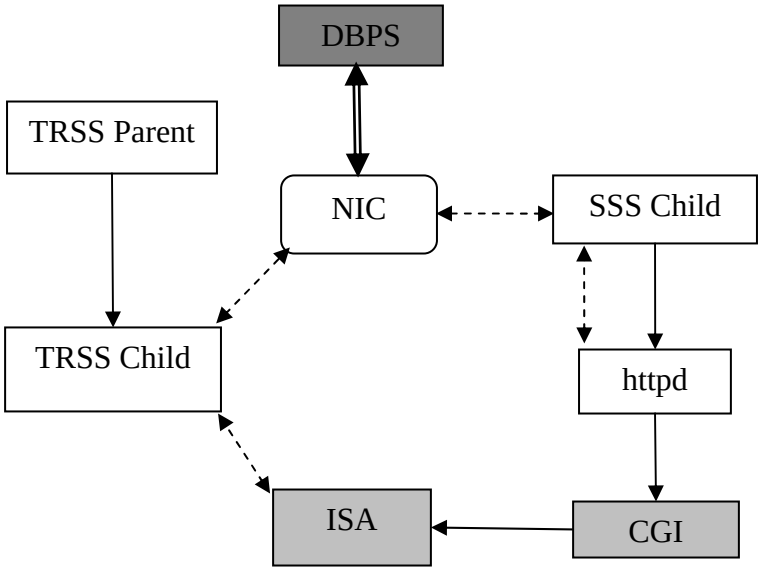

Figure 9. DBPS Implementation Context

To demonstrate a proof of concept for the DBPS processes it was sufficient to implement a single process which would perform the actions of both the DBPS Parent and DBPS Child processes. The main consequence of such an implementation is that each request from an ISA must be completely processed and results returned before another request can be handled. This is obviously inefficient for a production-level application, but is adequate for this demonstration. In the case of the DBPS, the creation of child processes is simply a mechanism which allows for multiple requests to be handled in parallel and it is not a mechanism which requires additional privileges on the MYSEA server.

To perform the functions of the Parent process, the DBPS prototype needed to take the following actions:

1. The DBPS must initialize a “server socket” and enable it to listen for incoming connections. This simply requires the use of the standard socket calls *socket*, *bind*, and *listen*. The DBPS then enters a *while* loop and waits for incoming connections using the *accept* socket call.

2. Once a connection has been received, the DBPS must verify that it is associated with a valid user session. This involves a lookup to the Remote Connections Database. First the DBPS initializes access to the database using the *rem_conn_access* function call. Once access is established, the DBPS requests the session level information associated with the socket the requester is using by making a *rem_con_get_user_session* call; if the socket is associated with a valid user session then session level information is returned along with the result REM_CONN_FOUND. If the REM_CONN_FOUND result is not found, the socket is not associated with a valid user session and the connection is dropped.

At this point the DBPS Parent would spawn a child to handle the connection and return to listening for other requests. The prototype continues with the DBPS Child functions.

3. To make comparisons of the session level, the results from the Remote Connection Database must be converted into a string format using the HRL Database. The DBPS initializes access to the HRL Database using the *init_hrl_db_for_access* call. The DBPS then makes a call to *get_hrl_db_label*, which returns the string equivalent of the session level information.

4. At this point the DBPS should receive the parameters from the ISA which will be used to construct the database queries. As noted below, however, use of integration of data sets was beyond the scope of this project, and static strings were used instead.

5. The DBPS determines which results to use through string comparisons on the HRL label that was retrieved. The results are copied into a buffer.

The DBPS then sends the size of the buffer to the ISA, followed by the contents of the buffer. Having fulfilled the request, the DBPS returns to the top of the while loop and continues to listen for requests.

2. ISA Process

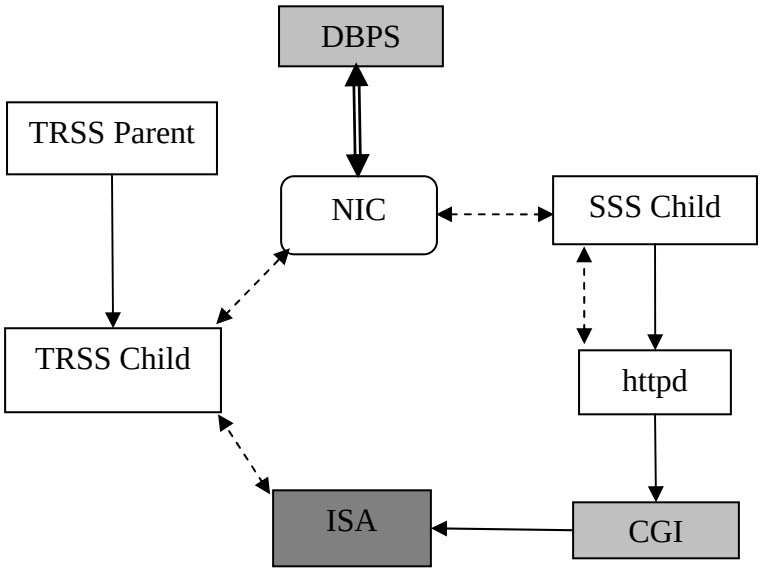

Figure 10. ISA Implementation Context

The Integration Stub Application (ISA) is the portion of the data integration system which requests data from the DBPS, integrates the results, and returned the integrated results to the user. The development of a data integration function was out of the scope of this project and, at the time of this project, there was no readily-available open-source integration application to incorporate. As such the ISA developed for this project is a simplified application which demonstrates the support for the necessary communications with the trusted DBPS from the user-level ISA. The act of integrating the data requires no additional privileges on the MYSEA system and so its absence here is not significant.

Because of the use of static strings used for results from the DBPS, the ISA prototype does not use command-line parameters nor does it transmit key-value pairs.

The MYSEA support of communications t between the DBPS and the ISA are demonstrated sufficiently, and such additions are simply extensions of functionality.

The ISA must perform the following actions:

6. The ISA must use the MYSEA socket interface to receive access to the network interface card. The ISA must first retrieve the handle to the MSKT database by making a *usm_get_id* function call. Then access to the database is made with the *mskt_access* call. This establishes access to use the necessary MYSEA socket calls. The ISA functions require the use of *mskt_socket*, *mskt_connect*, *mskt_read*, and *mskt_close*.
7. The ISA requests a connection to the DBPS with the *mskt_connect* call. This establishes a TCP/IP connection with the DBPS.
8. Since the simplified ISA does not transmit parameters, it simply waits to receive the size of the data to be sent using the *mskt_read* call. Once the size is received, the ISA uses a call to *malloc* to dynamically allocate the appropriate amount of memory to hold the results. The ISA then receives the results from the DBPS and stores them in the allocated space.
9. At this point the ISA should integrate the results. Our simplified ISA prints the contents of the buffer to standard out, to be captured by the CGI invocation as described in the previous chapter.
10. The ISA closes the socket with *mskt_close*, frees the allocated space, and terminates.

3. CGI Invocation Process

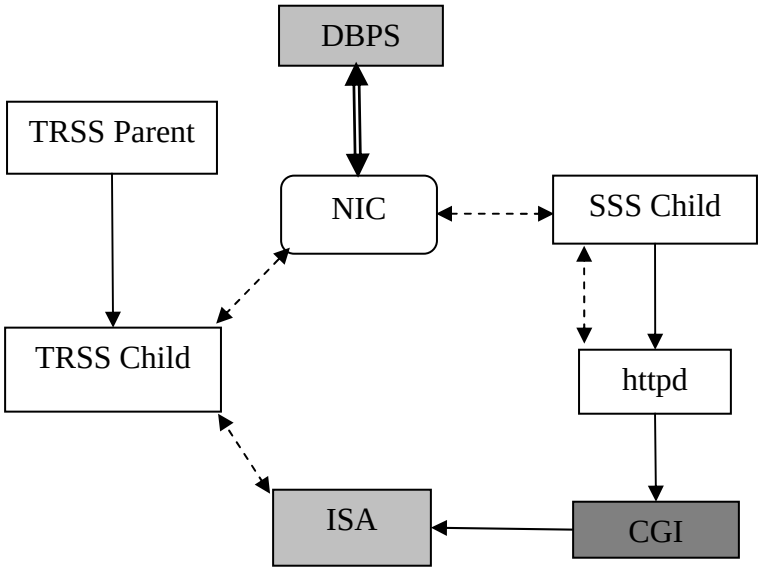

Figure 11. CGI Implementation Context

The CGI Invocation script was developed to present the uses with access to the ISA remote application. It is platform-neutral by design and can be accessed by any general web browser. The invocation script presents the user with apropriate form fields for input of parameters for use by the ISA. As parameters have not be utilized by the simplified ISA developed for this project, the CGI interface launches the ISA and displays the results automatically. Examples of the output may be found in Appendix C.

The ISA is launched by the backtick mechanism in Perl, which spawns the application as though it had been run from the command line and captures any output that the ISA writes to standard out. Future implemention of parameter passing to the ISA requires only that fields be added to the form and that their values be appended to the backtick operation.

D. SUMMARY

The implementation of the components discussed in this chapter demonstrate MYSEA support of a database application as a remote application. Although the proof-

of-concept application lacks some of the internal functionality of a full data integration application, it conforms to the design presented in Chapter III for secure communications in the MYSEA environment. The next chapter describes preliminary testing performed on this proof-of-concept application.

THIS PAGE INTENTIONALLY LEFT BLANK

V. TESTING

A. OVERVIEW

This chapter describes the plans for the preliminary tests which were performed on the proof-of-concept data integration application support. Two systems were used during the testing of data integration support: a MYSEA server with proof-of-concept data integration components and a Windows XP client machine running a TPE implemented as a Java application and a web browser. The MYSEA server was configured to have a single network interface which connected it to the MLS LAN on which the client was located. The network interface of the MYSEA server was configured as maximum secrecy and integrity level 0 (denoted max:il0) for reasons discussed in [4]. The physical testing environment is depicted in Figure 12 below, with the logical configuration shown in Figure 13

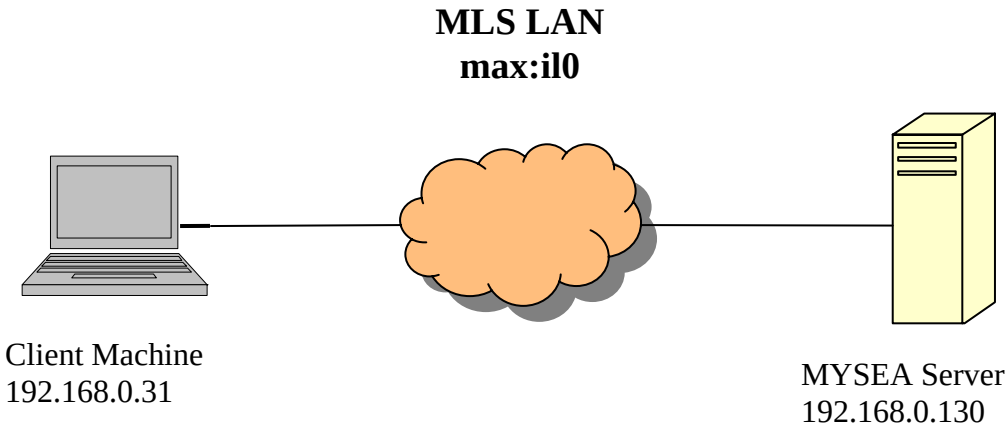

Figure 12. Physical Test Network Topology

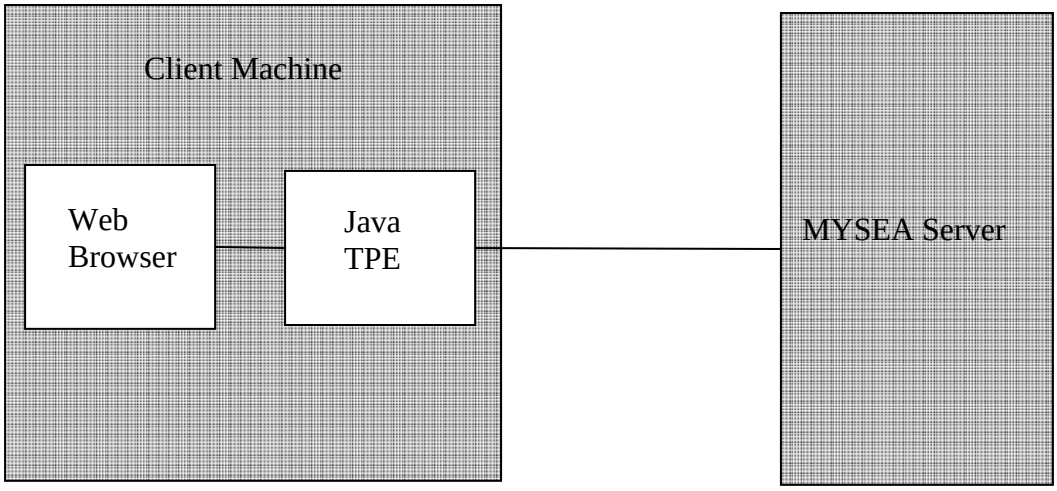

Figure 13. Logical Test Environment

The tests performed and the results of those tests are discussed at a general level in this chapter; specific test procedures and expected results are documented in Appendix C.

B. TEST PLAN

1. Overview

As discussed in the previous chapter, the proof-of-concept application developed as a part of this project lacks much of the functionality that would be expected from a production level product. The purpose of this application is to demonstrate that data integration can be successfully implemented in the MYSEA environment. Inclusion of the missing functionality would not invalidate the design of these applications or their support in the current MYSEA configuration.

Testing of an application such as that developed in this project falls into two categories: functional testing, which demonstrates that the components perform the expected functionality, and exception testing which, demonstrates that errors are handled correctly and the components do not exhibit unintended behavior. The limited amount of

functionality and the lack of user input limited the range of testing that could be performed on the proof-of-concept application. The following two test cases were determined to be adequate given these limits.

2. Test Suite 1

Test Suite 1 verifies that a user receives the data appropriate to his current session level. When an ISA establishes a connection with the DBPS on behalf of the user, the DBPS looks up the data which is at or below the user's current session level and returns these results to the ISA. The DBPS compares the user's session level to the different possible session levels, and when a match is found a particular string denoting the consolidation of data at different levels is copied into the buffer which is returned to the ISA. Additionally, the DBPS is designed to allow a record of each request, and the steps taken, to be entered into a log file.

This process requires the DBPS to properly perform several functions internally. First it must retrieve the session level of the connection from the Remote Connection Database, then it must convert the level from binary form into a human-readable format using the HRL module, and finally it must use the converted label string to look up the hard-coded, unconsolidated data. Error checking is not a part of this proof-of-concept application.

The following table describes the test cases and the expected results of each test:

Test ID	User Session	Expected Results
a1	SIM_UNCLASSIFIED	"No results" message
a2	SIM_CONFIDENTIAL	Confidential-level string
a3	SIM_SECRET	Secret-level string
a4	SIM_PACIFIC_SECRET	Coalition-Secret string
a5	SIM_NATO_SECRET	NATO-Secret string
a6	COALITION_COMMAND	NATO, Coalition, and Secret string

Test ID	User Session	Expected Results
a7	SIM_TOP_SECRET	Top Secret-level string

Table 3. Functional Testing

3. Test Suite 2

Test Suite 2 verifies that a rogue ISA executable will not be given access to the DBPS. The lack of user input limits the possible exception cases which can be tested. It was determined that since the DBPS listens on a port which might potentially be known or learned by a malicious entity, it is possible for a "rogue ISA" to attempt to make a connection to the DBPS. For this test, a "rogue ISA" is any application which may attempt to connect to the DBPS on its listening port, and which may or may not attempt to communicate to the DBPS using the appropriate MYSEA

The worst case scenario would be that such an attempt would result in a rogue ISA receiving a connection and some amount of data. This should not be possible in any situation, as the DBPS has been designed with several levels of security verification. A more likely case would result in the DBPS failing the lookup in the Remote Connection Database and then dropping the connection, which would at least present the opportunity for a denial of service attack, preventing legitimate requests from being serviced.

For this test a modified ISA was compiled using the same gcc 3.2 compiler as the legitimate ISA, except using sockets instead of MYSEA sockets. This rogue ISA was then run on the Windows client machine, demonstrating that a malicious user with a connection to the MLS LAN can not directly connect to the DBPS in order to bypass the enforcement of the mandatory security policy.

Test ID	Action	Expected result
b1	Connection from rogue ISA	Dropped connection after a failed database lookup

Table 4. Rogue ISA Exception Test

C. TEST RESULTS

The testing presented in Test Suite 1 produced the expected results, demonstrating that the DBPS was performing as expected with the proper strings being returned and the proper entries being made in the log file. There was no unexpected behavior.

The results for the second test suite were also as expected. A rogue ISA was able to establish a connection with the DBPS but was then dropped without receiving any data after a failed lookup in the Remote Connections Database. While this does not result in a security violation, the process of querying the Remote Connections Database ties up resources and may potentially lead to a denial of service for legitimate requests. The log file produced by the DBPS during testing confirms that the connection was accepted and a query made to the Remote Connections Database before the connection was determined to be invalid and dropped.

D. SUMMARY

This chapter presents the testing performed to demonstrate the data integration support of the proof-of-concept application developed for this project. All tests completed successfully.

Chapter VI concludes this work with a project summary and potential future work based on the findings of this project.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS

A. SUMMARY

This project has demonstrated the support of data integration in the MYSEA environment as a remote application. To accomplish this demonstration, the project established an application design involving the use of an untrusted Integration Application contacting a trusted Database Proxy Server on behalf of the user. This design involves the implementation of:

1. The Common Gateway Interface (CGI) which is used to accept input from the user and invoke the ISA.
2. The Integration Stub Application (ISA) which connects to the DBPS on behalf of the user, submits user-supplied parameters for the data queries made by the DBPS, and returns the integrated results of those queries to the user via the CGI script and the *httpd*.
3. The Database Proxy Server (DBPS) Parent and Child processes, for accepting and servicing requests for data from ISAs by making queries to databases of differing security levels.

Having determined a suitable design for data integration in the MYSEA environment, this project also developed a proof-of-concept application for these components and demonstrated their functionality through preliminary testing. The application and tests demonstrate the support available in MYSEA for data integration applications.

B. ANALYSIS OF THESIS QUESTIONS

This project was prompted by two main questions:

1. What modifications to the existing configuration are necessary to successfully implement data integration support on the MYSEA server?

2. What additional functionality is required to support the execution of data integration?

In answer to the first question, it was determined that a new trusted process would need to be implemented in order to complete the tasks required for data integration and maintain the security policy of the MYSEA system. Specifically, the need to send queries to data sources at different security levels necessitated the implementation of the Database Proxy Server. This is because the act of querying a data source involves writing data to that data source in the form of a SQL query, which could be used to establish a covert data channel between the high DBPS and some low data source. The DBPS must be trusted to ensure than information is not leaked through such a channel. With the addition of this trusted component, the rest of the MYSEA configuration remains unchanged.

Regarding the second question concerning additional functionality that would be needed to implement data integration, it was determined that, with the DBPS in place, nothing else needed to be added to support data integration applications. Building on the work in [4], this project was able to design a data integration application as a remote application, allowing the untrusted component of the data integration to run with the user ID and session level of the invoking user and request data from the trusted DBPS through the use of MYSEA sockets. No complications were encountered during the development of this design or the proof of concept implementation.

C. FUTURE WORK

1. DBPS Child

The current proof-of-concept implementation of the DBPS is missing functionality which would enhance the efficiency of the system and would be expected to be present in a production-level application. One of these functionalities is the spawning of child processes to handle requests from ISAs associated with valid user sessions. This functionality was determined to add extra complexity to the system without adding significantly to the demonstration of MYSEA's support for the application and so was left

out of the proof-of-concept implementation. The communications between the parent and child processes, as well as the potential clean-up that these children may require, presents an interesting addition to the basic application and would provide useful work for future study.

2. Data Base Connections

Also missing from the current implementation of the DBPS is the ability to connect to back-end data sources which would be queried for data. The current implementation of the DBPS does access pre-existing MYSEA databases through library interfaces; additional connections to other databases would not be prohibited by the MYSEA configuration. The addition of external data sources would also require the implementation that accepted parameters from the user and passed them between the ISA and the DBPS.

These additions add another layer of complexity and offer a range of acceptance and exception testing which was not covered in the current project. In particular, user input would have to be checked to guard against attacks at several levels within the application, such as interactions with the backtick mechanism in the CGI, the communications between the ISA and DBPS, and the potential of SQL-injection attacks on the backend databases.

3. Data Integration Functionality

It was determined early that the complexity of handling actual data integration was too large to be handled as a part of this project. Currently, there is no readily-available open source data integration project that could be incorporated into the MYSEA environment, and building such functionality from the ground up would require additional study on the mechanisms and algorithms used in data integration. There is added complexity when the discussion moves into the realm of multilevel data integration, including how to maintain security labels on data during and after the integration process, as well as concerns over the integrity of the data integration system itself.

4. Large Data Sets

The purpose of data integration is to take similar data from multiple sources and use them to synthesize a more-complete view than any of the individual sources themselves would present. The proof-of-concept application developed for this project handled small amounts of data, but it is given that a production-level application would deal with multiple data sets from multiple external sources, and that each of these data sets could be arbitrarily large quantities of data. Additionally, as noted in Chapter IV, there is an amount of overhead required by the protocol between the DBPS and the ISA for each request and for each data set returned. This presents the need for an method of efficiently handling large amounts of data to avoid bottlenecking and degradation of performance.

5. Federated Server Environment

The design of the DBPS and ISA is such that it readily lends itself to a federated environment, where the DBPS, ISA, and the backend data sources could all potentially reside on separate machines connected through the MLS LAN or over single-level networks. This was intentional in that the true utility of data integration itself comes when the data sources in question are not under the direct control of the department or agency which runs that data integration application. In such a situation the data integration application and data sources would almost surely reside on different machines and communicate over the network.

As presented in [4], however, this sort of environment presents challenges particular to the current configuration of the MYSEA server. Specifically, the SSS Parent process is designed to only accept connections from a user logged into the server via a TPE or from remote applications executing on the server which have been invoked by such a user. The SSS checks the User and Remote Connection databases when it receives a request and drops the connection if no matching entry is found. Currently these databases are local and are updated by other local trusted applications on the

MYSEA server. As was shown in the testing of the data integration application, a connection from some outside source not in the local database will never reach the DBPS.

One solution has already been addressed in previous work by Bui [20]. The specification presented in that work describes a “Single Sign-on” solution, utilizing an authentication server and an “application-mapping” server to handle user credentials and session levels and presents a protocol for allowing applications on separate servers to recognize all valid users. Additional design work is needed for the system as a whole and to verify support for remote ISAs to contact a DBPS residing on another MYSEA server A re-evaluation of the problem of "rogue" ISAs is advisable for any such solution.

6. Audit

The current implementation of the DBPS enters a record of each request it receives and the steps it carries out. The information provided in these logs has not been analyzed for usefulness or clarity and a study of what information should be logged and which can be left out would provide additional future work. There is a need for the MYSEA server as a whole to be provided with a unified auditing functionality, and the addition of this functionality may provide the basis for an entire project itself.

7. Further Testing

Several of the functional additions listed above present specific needs for testing. Implementing user input and external data source connections present well-known vectors for malicious activity, such as SQL injection and exploitation of CGI mechanisms, as noted. Implementing standard data integration functionality adds a level of complexity which requires its own testing to catch unintended behaviors and ensure that exceptional conditions are handled securely. In addition to these concerns, a production-level implementation of data integration would require stress testing to determine whether and to what extent the processing encounters bottlenecks, and to determine ways of mitigating such occurrences.

D. CONCLUSION

As more and more data is stored and manipulated through heterogeneous data sources there is a growing need for tools which aid in sharing and compiling related data which may be inconsistent or contradictory in order to get a full picture. Data integration is a powerful tool to handle this problem, but its usability for government and military purposes is complicated and confounded by the necessity to integrate data at potentially different classification levels. MYSEA is a project aimed at creating a distributed multilevel secure computing environment which lends itself well to the incorporation of data integration applications. The demonstration of MYSEA support for data integration applications is another step towards a fully functional multilevel secure system.

APPENDIX A: SOURCE CODE LISTING

This appendix provides a listing of the internal MYSEA source code files that were created for this project. All the files listed are new to the MYSEA distribution and were created from scratch.

/usr/local/mysea/dbps/dbps.c

/usr/local/mysea/dbps/Makefile

/usr/local/mysea/isa/isa.c

/usr/local/mysea/isa/Makefile

/home/http/cgi-bin/data_ integration.cgi

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B: INSTALLATION PROCEDURES

A. OVERVIEW

The purpose of this appendix is to explain the installation of data integration support into the MYSEA environment. Specifically, this entails compiling binaries for the DBPS and ISA, installing the CGI script used to invoke the ISA, and enabling the DBPS as a daemon and as a trusted process.

The following instructions refer to the Secure Attention Key (SAK). On the XTS-400 console, this is enabled as the “alt” and “print screen” keys pressed simultaneously. In order to change security levels the user should issue a SAK followed by the *sl* command, and enter the desired security and integrity levels as prompted.

The user should be logged in to the MYSEA server with the user ID of admin and the group ID of mysea for all of the steps presented.

B. BUILD MYSEA BINARIES

All of the object code and executables are pre-built and contained in the mysea.tar file. Issuing the make_all.sh command during the normal MYSEA installation process will remove and regenerate all of the executables for the MYSEA environment, including those implemented during this project for data integration support. The following instructions explain how to remove and regenerate these new components individually.

(Set security and integrity levels – sl0:il3)

SAK

Enter command? `run`

To rebuild the Database Proxy Server (DBPS) executable:

```
cd /usr/local/mysea/dbps
```

```
make clean
make dbps
```

To rebuild the Integration Stub Application (ISA) executable:

```
cd /usr/local/mysea/isa
make clean
make isa
```

C. CONFIGURATION OF DBPS

In order for the DBPS to be initialized to accept incoming requests from and ISA applications launched by users it must be enabled as a system daemon, to be launched as part of the system startup process. This is accomplished using the *daemon_edit* program provided by the STOP 6 operating system. The steps are as follows:

(Set security and integrity levels – min:max)

SAK

Enter command? `daemon_edit`

`add`

`dbps`	For Daemon Name
`dbps`	For Command Line
`<CR>`	For Arguments
`<CR>`	For Daemon Environment Setting
`Y`	For Start Daemon at Startup
`Y`	For High Integrity Program File
`N`	For Control Device

`max` For Security Level

`il0` For Integrity Level

`admin` For Username

`mysea` For Groupname

`N` For Display Current Order

`<CR>` For Daemon Start Index

`2` For Start Delay

`4` For Stop Delay

`Database Proxy Server` For Description

`exit` To leave `daemon_edit`

The DBPS must also be designated as a trusted process in order to perform it's functionality. This is accomplished using the `tp_edit` program provided by the STOP 6 operating system using the steps that follow:

(Set security and integrity levels - min:max)

SAK

Enter command? `tp_edit`

cd make sure in /system directory

(The XTS 400 only allows high-integrity daemons to execute from the /system directory.)

`add`

`dbps` For Program Name

`/usr/local/mysea/bin/dbps` For Path Name

`<CR>` For Max Integrity

`<CR>` For Min Integrity

`N`	For Assign Privileges
`Y`	For Modify Discretionary Access
`<CR>`	For New Owner
`<CR>`	For New Group
`rwx`	For Owner
`<CR>`	For Username for Specific Permission
`rwx`	For Group
`<CR>`	For Groupname for Specific Permission
`none`	For Other

`exit` To Exit `tp_edit`

D. SET UP CGI SCRIPT

The CGI invocation script for the ISA must be placed in the web server's cgi-bin directory. The following steps should only be performed after the /home/http directory has been created and populated through the course of the normal MYSEA installation:

(Set security and integrity levels – sl0:il3)

SAK

Enter command? `run`

```
cp /usr/local/mysea/cgi-bin/data_integration.cgi
/home/http/cgi-bin/
```

```
chmod 555 /home/http/cgi-bin/data_integration.cgi
```

E. VERIFY INSTALLATION

To verify that the data integration components have been properly installed and are running correctly, perform the tests outlined in Appendix C, Section B.

APPENDIX C: TESTING PROCEDURES

A. PRECONDITIONS

The purpose of this appendix is to describe the test procedure used in the test plan presented in Chapter V. The following precondition must be met before testing can begin:

- The test network connecting the client machine to the MYSEA server is set up as illustrated in Chapter V.
- The mdemo1 account exists on the MYSEA server with the default session level of sl1:il0 and a maximum session level of at least sl7:il0. (Following the standard MYSEA installation instructions will ensure this.)
- The ccdemo account exists on the MYSEA server with the default and maximum session level of sl5:il0 with the sc1, sc2, and sc3 security compartments. (Following the standard MYSEA installation instructions will ensure this.)
- The client machine has an installation of Cygwin, including the `gcc` and `make` packages. A copy of the isa.c and associated Makefile should be placed in Cygwin's `/tmp` directory. This directory will be found in `C:\cygwin\tmp` for a default installation of Cygwin.

With these preconditions met testing can begin.

B. TEST SUITE 1

Log in to the MYSEA server from the client machine.

1. Launch the Java TPE application by double-clicking the "tcbe" icon on the desktop.

2. Click the "SAR" button.
3. At the login prompt, type "mdemo1" and press Enter.
4. At the password prompt, type the password for mdemo1 and press Enter.

The session level should be SIM_UNCLASSIFIED. To ensure this:

5. Click the "SAR" button
6. Type "sl" and press Enter.
7. Type "SIM_UNCLASSIFIED" and press enter.

To access the web interfaces, a session must be established.

8. Click the "SAR" button.
9. Type "run" and press Enter.

Navigate to the data integration remote application invocation page

10. Open a web browser by double-clicking on the "Internet Explorer" icon on the desktop.
11. Navigate to the application invocation page by typing `http://192.168.0.130/cgi-bin/data_integration.cgi` in the "Address" field of the web browser.

    The page should display the following:

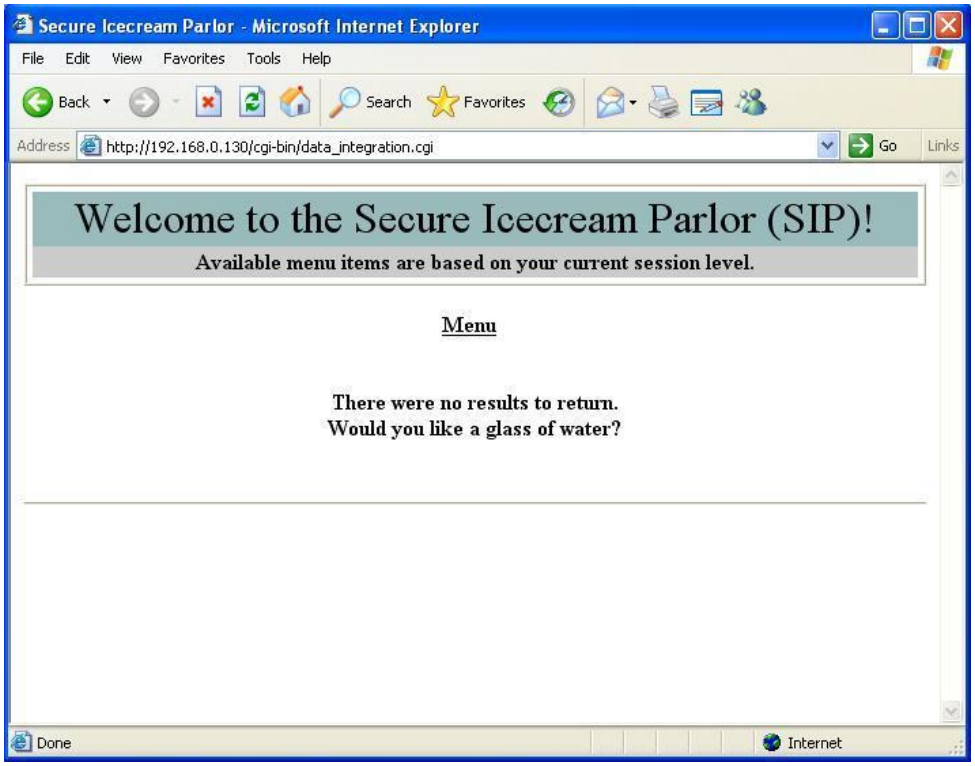

Figure 14. SIM_UNCLASSIFIED Test Results

Repeat the test for the SIM_CONFIDENTIAL security level.

12. Click the "SAR" button
13. Type "sl" and press Enter.
14. Type "SIM_CONFIDENTIAL" and press enter.
15. Click the "SAR" button.
16. Type "run" and press Enter.
17. Navigate to the application invocation page by typing `http://192.168.0.130/cgi-bin/data_integration.cgi` in the "Address" field of the web browser.

    The page should display the following:

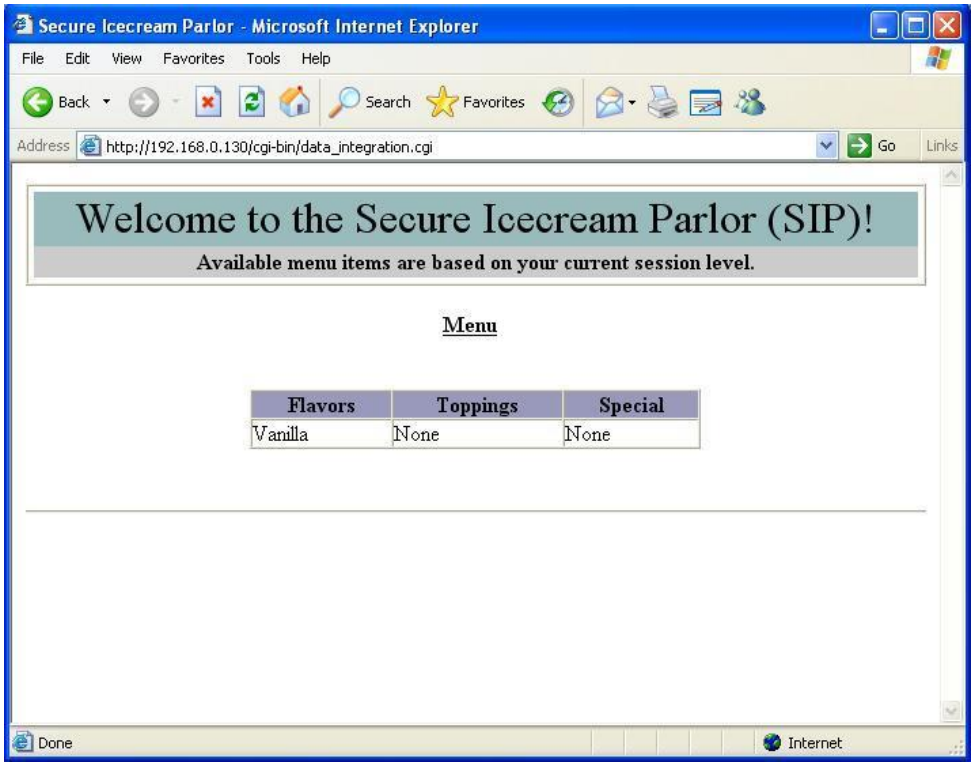

Figure 15. SIM_CONFIDENTIAL Test Results

Repeat the test for the SIM_SECRET security level.

18. Click the "SAR" button
19. Type "sl" and press Enter.
20. Type "SIM_SECRET" and press enter.
21. Click the "SAR" button.
22. Type "run" and press Enter.
23. Navigate to the application invocation page by typing `http://192.168.0.130/cgi-bin/data_integration.cgi` in the "Address" field of the web browser.

    The page should display the following:

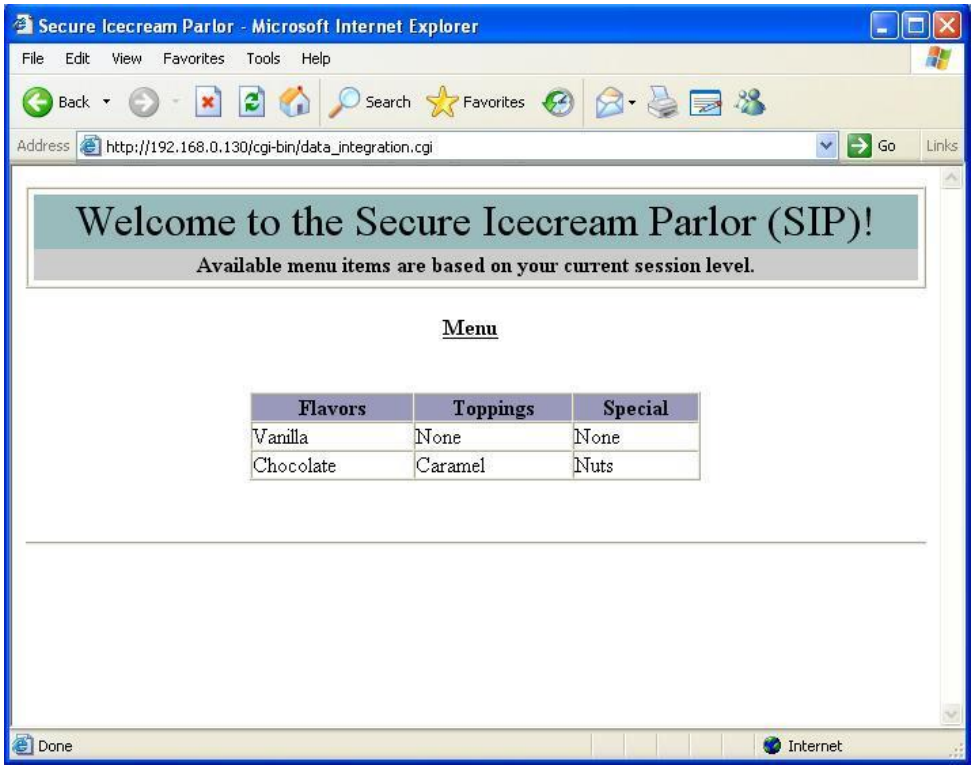

Figure 16. SIM_SECRET Test Results

Repeat the test for the SIM_TOP_SECRET security level.

24. Click the "SAR" button
25. Type "sl" and press Enter.
26. Type "SIM_TOP_SECRET" and press enter.
27. Click the "SAR" button.
28. Type "run" and press Enter.
29. Navigate to the application invocation page by typing `http://192.168.0.130/cgi-bin/data_integration.cgi` in the "Address" field of the web browser.

    The page should display the following:

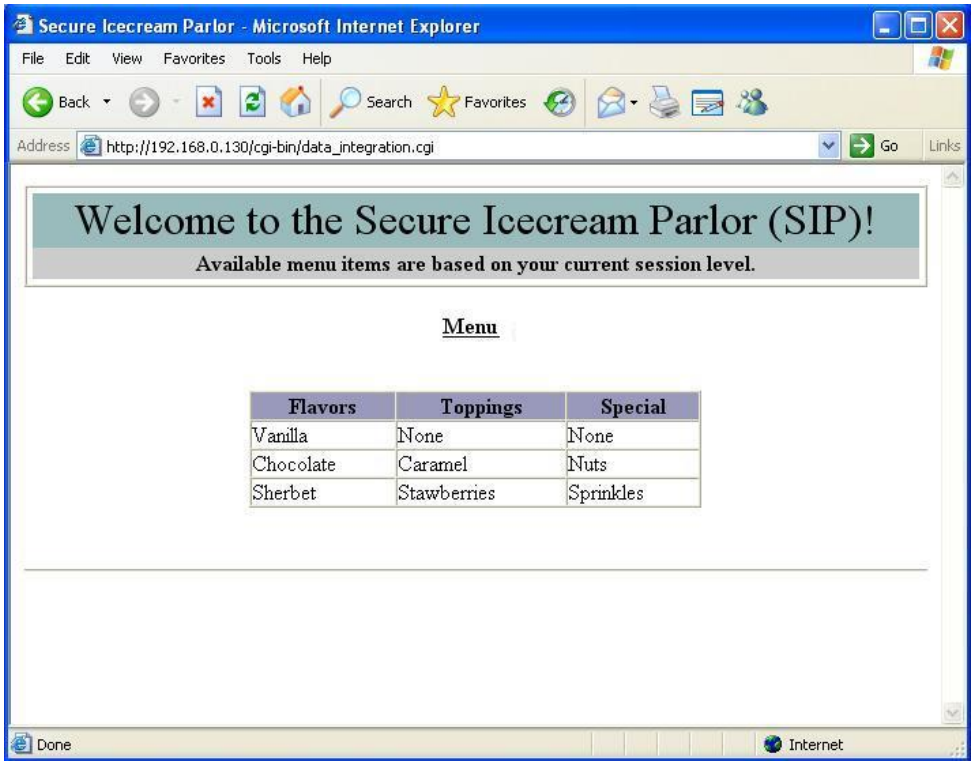

Figure 17. SIM_TOP_SECRET Test Results

In order to test for SIM_NATO_SECRET, SIM_PACIFIC SECRET, and COALITION_COMMAND security levels, we must switch to the ccdemo user.

30. Click the "SAR" button.

31. Type "logout" and press Enter

32. Click the "SAR" button.

33. At the login prompt, type "ccdemo" and press Enter.

34. At the password prompt, type the password for ccdemo and press Enter.

Repeat the test for the SIM_NATO_SECRET security level.

35. Click the "SAR" button

36. Type "sl" and press Enter.

37. Type "SIM_NATO_SECRET" and press enter.

38. Click the "SAR" button.

39. Type "run" and press Enter.

40. Navigate to the application invocation page by typing `http://192.168.0.130/cgi-bin/data_integration.cgi` in the "Address" field of the web browser.

    The page should display the following:

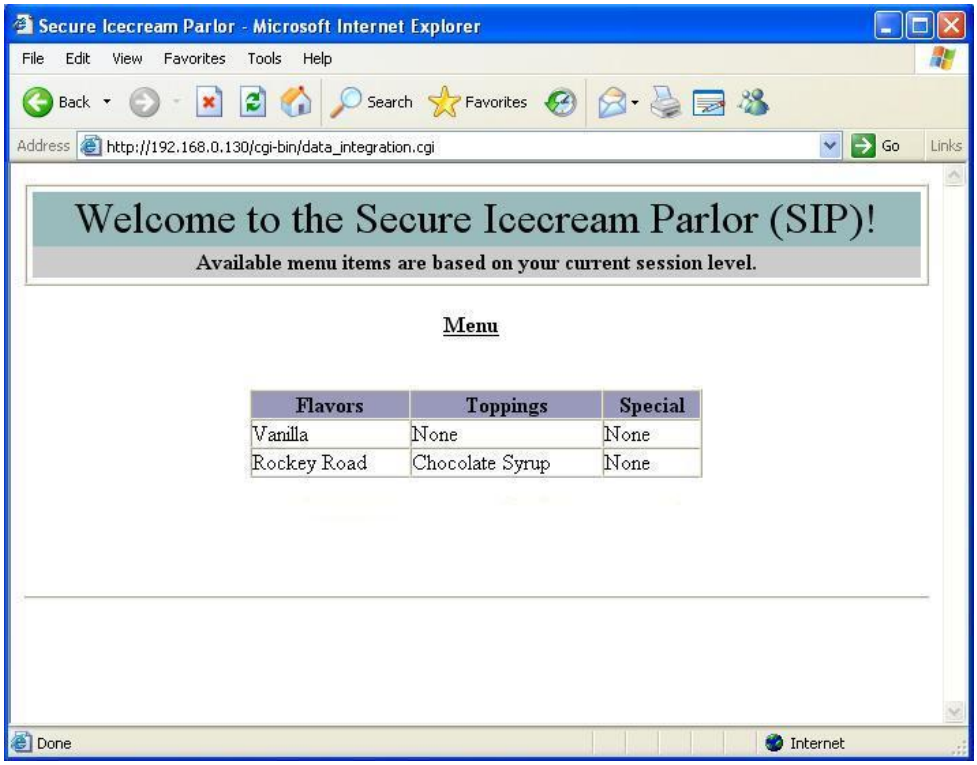

Figure 18. SIM_NATO_SECRET Test Results

Repeat the test for the SIM_PACIFIC_SECRET security level.

41. Click the "SAR" button

42. Type "sl" and press Enter.

43. Type "SIM_PACIFIC_SECRET" and press enter.

44. Click the "SAR" button.

45. Type "run" and press Enter.

46. Navigate to the application invocation page by typing `http://192.168.0.130/cgi-bin/data_integration.cgi` in the "Address" field of the web browser.

    The page should display the following:

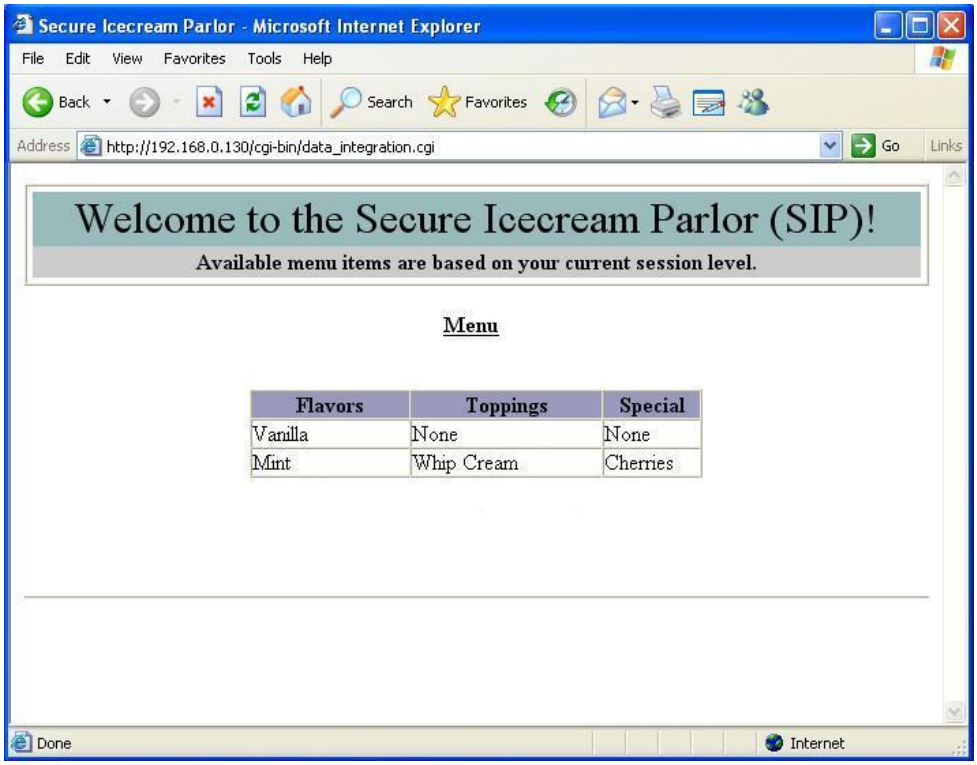

Figure 19. SIM_PACIFIC_SECRET Test Results

Repeat the test for the COALITION_COMMAND security level.

47. Click the "SAR" button
48. Type "sl" and press Enter.
49. Type "COALITION_COMMAND" and press enter.
50. Click the "SAR" button.
51. Type "run" and press Enter.
52. Navigate to the application invocation page by typing `http://192.168.0.130/cgi-bin/data_integration.cgi`

in the "Address" field of the web browser.

The page should display the following:

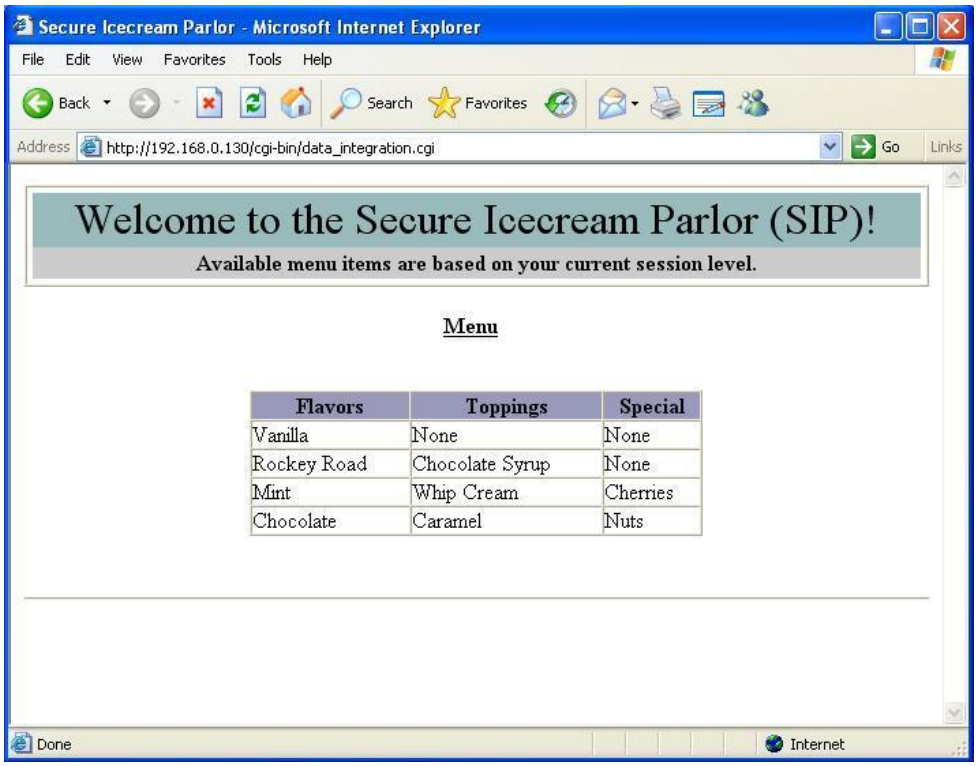

Figure 20. COALITION_COMMAND Test Results

C. TEST SUITE 2

Test Suite 2 is intended to demonstrate that a 'rogue' ISA process, as described in Chapter V, Section B.2, can not make a successful connection to the DBPS and receive data.

The client machine has a 'rogue' ISA application which uses standard socket calls instead of MYSEA socket calls to connect directly to the DBPS's listening port. A rogue ISA can be created with the isa.c and Makefile files with these steps:

1. Launch Cygwin my double-clicking the Cygwin icon on the desktop.
2. Type "`cd /tmp`" and press Enter.
3. Type "`make rogue`" and press Enter

Run the rogue ISA:

4. In the Cygwin window, type "`./rogue_isa 192.168.0.130`" and hit enter.

The command will run but should hang. Hit ctrl-c to end the process. The DBPS log should indicate that a connection was made, no entry was found in the Remote Connections database, and the connection was dropped. To view the DBPS log you must log into the MYSEA server console with the username admin, the groupname mysea, and the session level set to max:il0. After logging in:

5. Press the Secure Attention Key (SAR)
6. Type "run" and hit enter.
7. Type `cat /tmp/dbps.log` and hit enter.
8. The last lines of the log should look like the following (the process ID will probably be different):

```
27109:In while loop.
27109:Connection accepted.
27109:Checking security.
27109:Connected to Remote Connection DB.
27109:Disconnected from Remote Connection DB.
27109:Invalid connection.  Connection dropped.
27109:In while loop.
```

LIST OF REFERENCES

[1] J. Ullman, "Information Integration Using Logical Views" in ICDT '97: Proceedings of the 6th International Conference on Database Theory, Delphi, Greece, January 1997, pp 19 - 40.

[2] National Security Agency, "Overview of the End-to-End IA Component of the GIG Integrated Architecture" October 2004.

[3] C. Irvine, T. Levin, T. Nguyen, D. Shifflett, J. Khosalim, P. Clark, A. Wong, F. Afinidad, D. Bibighaus, and J. Sears, “Overview of a High Assurance Architecture for Distributed Multilevel Security” in Proceedings 5th IEEE Systems, Man and Cybernetics Information Assurance Workshop, United States Military Academy, West Point, NY, 10 - 11 June 2004, pp. 38 - 45.

[4] M. Egan, An Implementation of Remote Application Support in a Multilevel Environment, M.S. Thesis, Naval Postgraduate School, March 2006.

[5] J. Ullman, “Implementation of Logical Query Languages for Databases”, in ACM Trans. Database Syst., 10, 1985, pp. 289-321.

[6] D. Chamberlin, R. Boyce, SEQUEL: A Structured English Query Language, pp. 249 - 264.

[7] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, D. Sucio, “A Query Language for XML,” in Computer Networks vol. 31, Amsterdam, Netherlands, 1999, pp. 1155 - 1169.

[8] S. Abiteboul, D. Quass, J. McHugh, J. Widom, J. Wiener, “The Lorel Query Language for Semistructured Data,” in International Journal on Digital Libraries vol. 1, 1997, pp. 68 - 88.

[9] E. Codd, “A Relational Model for Large Shared Data Banks.” Communications of the ACM, June 1970, pp. 377 - 387.

[10] S. Castano, M. Fugini, G. Martella and P. Samarati, Database Security, ACM Press, Addison-Wesley Publishing Company, Wokingham, England, 1995.

[11] C. E. Irvine and T. E.Levin, "A Cautionary Note Regarding the Data Integrity Capacity of Certain Secure Systems", Integrity, Internal Control and Security in Information Systems, ed. M. Gertz, E. Guldentops, L. Strous, Kluwer Academic Publishers, Norwell, MA, 2002, pp 3-25.

[12] D. E. Bell and L. J. La Padula, "Secure Computer System: Unified Exposition and Multics Interpretation", ESD-TR-75-306, MITRE Corporation, Bedford, MA, 1976.

[13] K .J. Biba, "Integrity Considerations for Secure Computer Systems", ESD-TR-76-372, MITRE Corporation, Bedford, MA, 1977.

[14] R. Cooper, Remote Application Support in a Multilevel Environment, M.S. thesis, Naval Postgraduate School, March 2005.

[15] The World Wide Web Consortium, "Hypertext Transfer Protocol -- HTTP/1.1." [Online Document] June 1999, [Sept. 2007], Available at http://www.w3.org/Protocols/rfc2616/rfc2616.html.

[16] E. Bersack, Implementation of a HyperText Transfer Protocol Server on a High Assurance Multilevel Secure Platform, M.S. thesis, Naval Postgraduate School, December 2000.

[17] The Apache Software Foundation, "About the Apache HTTP Server Project", [Online Document] November 2006, [August 2007] Available at HTTP://httpd.apache.org/ABOUT_APACHE.html.

[18] Netcraft News, "Netcraft: Web Server Survey Archives", [Online Document], July 2007, [July 2007], Available at HTTP://news.netcraft.com/archives/web_server_survey.html.

[19] P. Wainwright, Professional Apache, Wrox Press Ltd., 1999.

[20] S. Bui, Single Sign-On Solution For MYSEA Services, M.S. Thesis, Naval Postgraduate School, September 2005.

[21] National Security Agency, “Information Assurance Technical Framework” [Online Document] Sept 2002, [Sept 2007], Available at http://www.iatf.net/framework_docs/version-3_1/file_serve.cfm?chapter=FntMatter.pdf.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, VA

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, CA

3. Hugo A. Badillo
   NSA
   Fort Meade, MD

4. George Bieber
   OSD
   Washington, DC

5. John Campbell
   National Security Agency
   Fort Meade, MD

6. Deborah Cooper
   DC Associates, LLC
   Roslyn, VA

7. Louise Davidson
   National Geospatial Agency
   Bethesda, MD

8. Steve Davis
   NRO
   Chantilly, VA

9. Vincent J. DiMaria
   National Security Agency
   Fort Meade, MD

10. Dr. Tim Fossum
    National Science Foundation
    Arlington, VA

11. Jennifer Guild
    SPAWAR
    Charleston, SC

12. Steve LaFountain
NSA
Fort Meade, MD

13. Dr. Greg Larson
IDA
Alexandria, VA

14. Dr. Karl Levitt
NSF
Arlington, VA

15. Dr. Vic Maconachy
NSA
Fort Meade, MD

16. Dr. John Monastra
Aerospace Corporation
Chantilly, VA

17. John Mildner
SPAWAR
Charleston, SC

18. Mark T. Powell
Federal Aviation Administration
Washington, DC

19. Jim Roberts
Central Intelligence Agency
Reston, VA

20. Keith Jarren
NSA
Fort Meade, MD

21. Ed Schneider
IDA
Alexandria, VA

22. Keith Schwalm
Good Harbor Consulting, LLC
Washington, DC

23. Ken Shotting
NSA
Fort Meade, MD

24. CDR Wayne Slocum
SPAWAR
San Diego, CA

25. Dr. Ralph Wachter
ONR
Arlington, VA

26. Dr. Cynthia E. Irvine
Naval Postgraduate School
Monterey, CA

27. Thuy D. Nguyen
Naval Postgraduate School
Monterey, CA

28. Andrew Portner
Civilian, Naval Postgraduate School
Monterey, CA