Small Business Innovation Research Program Report
DARPA SB971-006 Contract DAAH01-97-C-R135
19 January, 1998

System and Security Management Tools

Neil R. Fraser, et. al.
Curriculum Corporation

Final Report

Prepared for:
U.S. Department of Defense
Small Business Innovation Research (SBIR) Program

CURRICULUM CORP.

Curriculum Corporation
P.O. Box 116
Kings Mills, OH 45034-0116

This page intentionally blank.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	01/19/98	Final 06/25/97 - 01/19/98

4. TITLE AND SUBTITLE
System and Security Management Tools
A DCE Based Intrusion Detection System

5. FUNDING NUMBERS
C DAAH01-97-C-R135

6. AUTHOR(S)
Neil Fraser

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Curriculum Corporation
P.O. Box 116, Kings Mills,
OH 45034

8. PERFORMING ORGANIZATION REPORT NUMBER
DAAH01-97-C-R135-2

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
Ms. Lunt
3701 North Fairfax Drive
Arlington, VA 22203-1714

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Public Availability

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report documents work performed under contract DAAH01-97-C-R135. This work explores the use of Distributed Computing Environment facilities to aid in developing a commercial Intrusion Detection System that runs on most modern operating systems. The conclusion of the report is that our prototypes were an unguarded success and that we hope to continue this work in SBIR phases II and III.

14. SUBJECT TERMS
Intrusion Detection
Distributed Computing Environment (DCE)

15. NUMBER OF PAGES
102

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

PAN RTW C8-97 PHASE I

DARPA/SBIR Program

Report Distribution List

Quarterly Status Report (2 Copies)

Commander 1 Copy
U.S. Army Missile Command
ATTN: AMSMI-RD-WS-DP-SB (Mr. Neal, Tech Monitor)
Bldg 7804, Room 213
Redstone Arsenal, AL 35898-5248

Director 1 Copy
Defense Advanced Research Projects Agency
ATTN: ITO (Ms. Lunt)
3701 North Fairfax Drive
Arlington, VA 22203-1714

Final Technical Report (9 Copies)

Commander 2 Copies
U.S. Army Missile Command
ATTN: AMSMI-RD-WS-DP-SB (Mr. Neal, Tech Monitor)
Bldg 7804, Room 213
Redstone Arsenal, AL 35898-5248

Commander 1 Copy
U.S. Army Missile Command
ATTN: AMSMI-RD-CS-R
Redstone Arsenal, AL 35898-5240

Director 1 Copy
U.S. Army Missile Command
ATTN: AMSMI-RD-WS
Redstone Arsenal, AL 35898-5248

PAN RTW C8-97

PHASE I

DARPA/SBIR Program

Report Distribution List (cont'd)

Director
Defense Advanced Research Projects Agency
ATTN: ITO (Ms. Lunt)
3701 North Fairfax Drive
Arlington, VA 22203-1714

1 Copy

Director
Defense Advanced Research Projects Agency
ATTN: OASB/ARPA Library
3701 North Fairfax Drive
Arlington, VA 22203-1714

1 Copy

Defense Technical Information Center
ATTN: Acquisitions/OCP
8725 John J. Kingman Rd., STE 0944
Ft. Belvoir, VA 22060-6218

2 Copies

Director
Defense Advanced Research Projects Agency
ATTN: OASB/SBIR
3701 North Fairfax Drive
Arlington, VA 22203-1714

1 Copy

Abstract

Intrusion detection consists of determining patterns of behavior that could compromise the integrity of a computer system by allowing an undesirable intrusion. This work explores the use of Distributed Computing Environment facilities to aid in developing a commercial grade distributed intrusion detection system. After studying previous work already done in this field, prototype development was conducted to determine how best to use the DCE Audit and Security Services to perform intrusion detection functions. Methods were developed to ensure secure, efficient, scalable, and fault tolerant communications would be made manifest in the overall architecture of the prototype. Tests were conducted independent of and in conjunction with previously developed intrusion detection systems from outside sources. These tests strongly indicate that a DCE-based intrusion detection system is not only possible, but would be welcomed, filling a void in a market place hungry for security solutions. Independent investors have committed their financial support to see this work continues.

This page intentionally left blank

Table of Contents

List of Figures

1 Summary

Beginning on June 25, 1997 Curriculum Corporation embarked upon a "proof of Concept" exercise in support of the project DARPA SB971-006 "System and Security Management Tools". Work has progressed even better than expected, and continues to support the intent of the original proposal. Endorsed by outside investors, we are confident we can produce a commercial grade Intrusion Detection System that will be unique in its marketplace.

The primary objective of the project is to determine if it is technically feasible and commercially viable to design an Intrusion Detection System (IDS) that uses the Open Group's Distributed Computing Environment (DCE) in two main ways:

1) As a source of information that might lead the IDS to detect an intrusion.
2) As a security infrastructure that can be utilized to respond to an intrusion.

The DCE Audit Service was found to be a valuable source of clues to possible intrusions. In much the same way that traditional Intrusion Detection Systems obtain clues from an operating system's audit trail, the proposed DCE-based solution was able to "observe" anomalies through the DCE Audit Service. Since the DCE Audit Service audits security-related events on behalf of each system in its cell, it was discovered that it is possible to detect "concerted attacks" across a wide range of systems (within a DCE cell).

Prototype code produced during our Phase I effort shows that we are able to, on the fly, deny an intruder access to resources across an entire DCE cell upon detection of an anomaly. This is accomplished by deleting authentication tickets at the site of the intrusion. Further, it was clear that the granularity of the DCE Access Control List (ACL) based authorization scheme meant that it would be possible to provide a range of response scenarios. Our approach avoids the "denial of service" problems associated with marking a login account as invalid by ticket deletion.

Some other current research projects in Intrusion Detection focus on Intrusion Detection and Response as a Distributed Systems problem, i.e. many systems must "look for" possible intrusions and patterns of attack across many systems. Our perception of one of the shortcomings in these projects is that they build their own (secure) client-server infrastructure within which distributed components can communicate. Our belief is that use of an existing industry standard distributed computing infrastructure (the Distributed Computing Environment, DCE) allows us to focus on the other parts of the Intrusion Detection problem that are difficult to solve.

With this data in place, Curriculum Corporation is confident that it will be able to develop a commercially successful Intrusion Detection product within the eighteen-month window provided by SBIR Phase II.

This page intentionally left blank

2 Introduction

Within the discipline of Computer System Security, there are tools, techniques, and tasks associated with preventing, detecting, and responding to undesirable events within a computer system. While the discipline is very focused, it is quite diverse in the range of topics discussed. Some topics are fairly straight forward and simple to use such as Access Control Lists (ACLs) to grant access to specific resources (e.g. user A is allowed to update file B). Other topics are more esoteric requiring sophisticated algorithms (e.g. keystroke signatures to detect a forged identity).

This research effort focuses on the non-trivial task of detecting and responding to intrusive behaviors in a dynamic system. While this is not a new area of study, it continues to be of great interest since it has serious potential in effectively supporting complex inter-connected computer networks. The frequency of "attachments" between previously isolated networks has escalated with the advent of the Internet. Often the user's demands for Internet access place the computer support personnel in the position of granting the access through the private network to achieve economy of scale. Once connected, the support personnel often do not expend the effort to keep constant vigil on potential or actual attacks upon their networks.

What is needed is an active monitoring system that can detect and respond to attacks as they happen. This system must adapt to the changing nature of the overall system without being readily "spoofed" by the would-be attacker. Generically, these systems are referred to as Intrusion Detection Systems. They take various forms of implementation based on the premise of their design. Early efforts focused on a single system while recent efforts focus on the collection of systems connected via some network technology. Generically, this latter focus is referred to as "distributed systems" and commonly involves many systems of diverse types (e.g. IBM OS/390 and AIX, DEC VMS and OSF/1, Microsoft Window 95 and Windows NT, Sun Microsystems Solaris, Hewlett-Packard HP-UX).

Performing intrusion detection in a distributed system is far more complex since vulnerabilities increase with the number of and differences between the systems involved. Often the distributed intrusion detection systems have focused on one class of platform (e.g. UNIX) to reduce the complexity of the task involved. The objective of this research uses a multi-vendor distributed infrastructure technology (Open Software Foundation (OSF) Distributed Computing Environment (DCE)) to create a distributed intrusion detection system. OSF/DCE is widely available and provides security features which can enable and support intrusion detection efforts.

2.1 Purpose of the Research

Curriculum Corporation proposed to research and develop an innovative platform-independent intrusion detection and response tool. The tool would use Open Software Foundation (OSF) Distributed Computing Environment (DCE) technology in order to support a diverse mixture of mainstream computer platforms. This tool would use the features inherent in the DCE Security Service to detect and respond to intrusions on-the-fly.

2.2 Scope of the Research

Intrusion detection has been an area of study since the early 1980's and yielded products which examine various aspects of a system's vulnerabilities. Since DCE is a newer standard that provides a centralized security model, work needed to be done to develop and integrate DCE features into previous intrusion detection efforts. The primary focus of the initial research proposal discussed herein related to the use of DCE features toward intrusion detection. Microsoft has now announced that the centralized security service in NT5.0 Domains will be an implementation of kerberos V (just as DCE is) using DCE Remote Procedure Calls (RPCs). It is now clear, therefore, that our research applies equally to networks of systems using NT Domain Controllers. This implies a wider marketplace than initially envisaged.

2.3 Report Structure

This report documents the research efforts as they unfolded, highlighting significant DCE "value added" functionality. Ultimately an initial architecture and elementary prototype from which a commercial grade product could be built is the end result of this research effort.

3 Methods, Assumptions, and Procedures

3.1 Identification of the Problem Being Addressed

Traditionally, much Intrusion Detection work has focused on examination of an audit trail from a single UNIX system and making predictions about a possible intrusion based on this local data. Curriculum Corporation's research focuses on developing a platform independent, heterogeneous intrusion detection and response tool based upon OSF/DCE technology. The technical problems being addressed are:

1) Today's Intrusion Detection Systems (IDSs) are mainly applicable to UNIX environments (few support Windows/NT or IBM/MVS) and none are deployable across all operating systems.
2) Automated responses to intrusions are often not implemented. Response needs to be fast and thus automation is a vital component.
3) Most IDSs have high "false alarm" rates. This has inhibited automation efforts.
4) IDSs are being developed for distributed environments but usually these IDSs are developed using proprietary and specially developed unique communication infrastructures for them. This makes "porting" to divergent platform expensive.
5) It is currently a non-trivial task to interface one IDS with another since, until recently, no common API existed.
6) Intrusion Detection problems also exist in HUGE computing environments (thousands of systems in a single site). Existing IDSs do not effectively scale to support environments this size.

Our Phase I proof of concept work has indicated that we can advance the state of the art in Intrusion Detection by:

1) Providing an enterprise-wide solution detecting anomalies on at least the following operating systems: Solaris, HP-UX, AIX, NT, MVS
2) Providing an immediate response facility affecting each of those operating systems listed above. The response shall be implemented across one thousand systems within one second.
3) Focusing much effort in the area of elimination of "false positive" indications of intrusion (through use of multiple independently developed Intrusion Detection engines and refinement of those engines, rather than invention another ID engine).
4) Utilizing a secure industry standard security and communications infrastructure (DCE RPC), therefore saving time that otherwise would be needed for creation of a security and communications infrastructure.
5) Implementing the forthcoming Common Intrusion Detection Framework (CIDF) to facilitate re-usability of components and interoperability with other Intrusion Detection products.
6) Leveraging the scalable DCE infrastructure (which supports tens of thousands of clients in a single cell) by interfacing with the DCE Security and Audit Service. The DCE Security Service only audits security related events and can therefore support a cell encompassing very large numbers of clients).

The tool set that we have prototyped makes up an enterprise-wide (scalable) detection and response system that is not tied to a specific operating system type. The proposed system is able to identify possible security anomalies and, in real time, transform the way that every system in a DCE cell behaves, in order to respond to the anomaly. A design goal is to be able to alert many systems in a very short interval. The proposed tool will allow very large numbers of desktops, midrange systems, and mainframes to react in concert to a possible security attack. The distributed nature of the design allows for a very agile response as well as a speedy return to normal work within a heterogeneous, secure environment such as DCE where a concerted response to an attack can be easily implemented.

One of the major technical challenges with any Intrusion Detection System (IDS) is to eliminate virtually all false alarms. After our six months of research, Curriculum Corporation believes that it would be infeasible to build an entirely "false-alarm-free" system from the ground up. Our research indicates that much other work has been done in building good detection engines, but that not as much has been done on tuning these engines for reliable use in a large distributed environment. We intend to license components of existing IDSs (including independently developed rules etc) and compare the results of these IDSs with those of our detection component (which monitors an entire network) in order to reach consensus on likelihood of attack. By licensing previous R&D works, we believe we can build upon a huge accumulated knowledge base and apply a more pragmatic approach to reduction of false positives. Other research in Intrusion Detection also validates the claim that the Detection Engine itself is no longer the only key element in a distributed IDS, but just one part of a larger application suite. Our research also shows that our centralized approach to detection facilitates identification of trends across an enterprise.

3.2 Phase I Objectives

Our Phase I technical objectives were:

1) Determination of a wide range of intrusion scenarios that can be identified by the combination of DCE Audit and our proposed IDS.
2) Determination of a range of responses to the intrusion scenarios identified above.
3) Design and implementation of prototype intrusion detection and response tool.

All of these three objectives were addressed and we are confident that there are no major obstacles to completion of a pragmatic, distributed Intrusion Detection System. This system extends the state of the art in a number of ways. With this in mind, ***Curriculum has sought and obtained enthusiastic outside reputable investors to support a Phase II fast track application.*** This application reflects the company's strong belief that an excellent state of the art commercial grade IDS can be created in Phase II.

We believe that two other major factors make our goals attainable in the Phase II timeframe:

1) Unlike other distributed IDS development projects, we shall make use of an existing communications and security infrastructure i.e. DCE security and RPC. This should save tremendous amount of "development the infrastructure" time. DCE already has a client/server environment that scales to very large environments and has many years of proven experience in large production environments.
2) We intend to license (rather than re-invent) some subset of our Intrusion Detection engine components. This will allow our company to build upon work already done, and focus on other areas of technical challenge (such as high false alarm rates).

3.3 Phase I Tasks

The following tasks were outlined in the Phase I proposal:

3.3.1 Task 1. Configure a Heterogeneous Test Environment

A test environment comprising the following operating systems was successfully configured:
SunSoft Solaris, Windows NT, IBM AIX, OS/2, HP-UX

The IBM AIX system and the HP-UX system were configured to replace the underlying authentication scheme of the native operating system with that provided by a centralized DCE Security Service. This forces every user authentication request to use the DCE Security Service (and therefore be audited in a central location).

The Solaris and AIX systems were configured to use a file system based on DCE (the DCE Distributed File Service – DFS). This has the effect of forcing (new) user file requests (for files

stored in DFS) to generate a DCE ticket request to the DCE Security Server. These requests can be detected through the DCE Audit Service (and therefore our IDS).

3.3.2 Task 2. Design an Interface to the Audit Service

Both the AIX and Solaris systems in our test environment were configured to run the DCE Audit Service. A prototype application was written that queried the Audit Service using the DCE Audit API to look for possible security anomalies. The prototype was a success. This code, see Appendix A, may be used as the basis for a production module if we are chosen to implement Phase II. The prototype interface to the DCE Audit Service was able to retrieve filtered events that were considered appropriate for intrusion detection.

3.3.3 Task 3. Configure the DCE Audit Service

The DCE Audit Service was re-configured so that all filtering was disabled. This allows our IDS to perform filtering at it's own discretion based on the maximum information possible from the DCE Audit Service.

3.3.4 Task 4. Test that Intrusions are Detected

A number of intrusion scenarios have been obtained, including: 1) a set of exploits from personnel at Rome Laboratories and 2) a collection of exploits obtained from numerous "hacker" pages on the internet. [16,17,18,19,20,21,22,23,24]

We intend to obtain many more scenarios for Phase II and to continue to test that these potential intrusions generate events in the DCE Audit Service that can be "seen" by an Intrusion Detection System. It is clear from the work that we have already done here that simple intrusions such as password guessing attacks can be detected centrally via the combination of our prototype tools and the DCE Audit Service.

It is unlikely that centralized audit (such as the DCE Audit Service or that of Kerberos V based NT 5.0 Domain controller) will be able to detect every intrusion scenario. For example, use of UNIX "setuid" files is invisible from the DCE Audit Service. The issue of setuid file accesses is primarily a UNIX-only issue (and DCE Audit is more heterogeneous), but there were other exploits that did not generate auditable events through our audit interface. It was therefore necessary to modify our initial architecture (as specified in the Phase I application). We have added a new component in our architecture, a local (lightweight) monitor. The local monitor will take the more traditional approach of reading local UNIX audit trails looking only for only a limited subset of patterns that are not registered in our centralized audit monitor (e.g. setuid/setgid file accesses and potentially other operating system centric areas of interest). This lightweight monitor is much simpler than a traditional IDS client component (to help in portability and performance) since its area of interest is limited only to events not detectable centrally. This new client component feeds events of interest back to our centralized Audit Monitors via secure DCE Remote Procedure Calls.

3.3.5 Task 5. Investigate Appropriate Response Scenarios

As a result of our Phase I work, the following response scenarios have been shown to be implementable in phase II:

1) Regeneration of a machine key
2) Account disabling
3) Re-encryption of the DCE Security Database
4) Regeneration of login keys used by authenticated application servers
5) Modification of the behavior of the DCE Password Strength Subsystem
6) Imposition of stricter auditing requirements
7) Shortening of the life span of existing and future tickets
8) Raise an appropriate alarm to security administration staff
9) Temporarily disable each DCE administrator account
10) Shut down applications

3.3.6 Task 6. Prototype Responses

The simplest response to an intrusion is to temporarily invalidate the login account being used by an intruder and to kill the processes used by that intruder. In the context of a DCE-based security infrastructure, these tasks would involve marking a login account as invalid (in the centralized DCE Security Service), and setting the "good since" (date) attribute on the tickets associated with this account to a date/time in the future. It was anticipated that these two operations would immediately revoke authorization for all tasks that require authentication. It turns out from our tests that there is currently a serious bug in the DCE Security Service that allows an identity to continue to perform privileged tasks despite the steps previously described. These steps are the appropriate steps as documented by the OSF. The problem has been reported to responsible vendors and we anticipate a fix across all DCE platforms. As a fallback position, we have tested that we can implement a local ticket revocation that would render an intruder harmless. This solution however is less elegant than the steps described previously since these only operate on the centralized security server and do not require our IDS to make any local operations on client systems.

We learned that more granular modifications to the DCE infrastructure can form a response without potentially jeopardizing day-to-day operations. For example, if an intruder were to obtain the login identity of (say) a business-critical DCE-based database server (all DCE processes have their own identities) then the effect of invalidating the login account for the database server would be to create a denial of service to the rest of the company. It is anticipated that, in Phase II, we can make use of the granular Access Control List mechanisms protecting the DCE security infrastructure to implement a range of responses to attacks of this kind.

3.3.7 Task 7. Performance Tests for Secure Broadcast DCE RPCs

As described below, the original "secure broadcast" approach was proven infeasible. An alternative technology was developed and proven to be viable in a large heterogeneous environment. The nature of the alternative proved to be more flexible yet providing the "quick response" needed for the rapid transmission of the alarm/response notifications. Using low-end 1990-vintage workstation hardware, we were able to transmit and receive (with no data loss) over 1,000 typical RPC notifications to multiple clients in less than seven seconds on a sustained basis. This was done without tuning the systems or 10Mbs ethernet components. We are confident that in phase II, the nature of this notification can be improved to achieve the "1,000 clients in less than 1 second" performance characteristics expected.

3.3.8 Task 8. Prototype the Entire Tool Set

As described in section 3, we have developed the tools to receive notifications from the DCE Audit Service, and trigger alarm notifications within an existing Intrusion Detection System. We have developed the architecture and fundamental technology to consolidate events across a diverse structure of centralized security servers to quickly identify "environment-wide" attacks, and a secure notification mechanism that can alert HUGE numbers of clients to appropriately respond to the attack. The prototyping we conducted actually created interfaces with both NIDES and IDIOT (pre-existing IDSs from outside sources) successfully. We now believe that it would be infeasible to build an entirely "false-alarm-free" system from the ground up. We intend to license components of existing IDSs (including independently developed rules etc) and compare the results of these IDSs with those of our detection component (which monitors an entire network) in order to reach consensus on likelihood of attack. By licensing previous R&D works, we believe we can build upon a huge accumulated knowledge base and apply a more pragmatic approach to reduction of false positives.

3.3.9 Task 9. Reporting

Three reports have been submitted, this being the final report.

3.4 Additional Tasks Completed in Phase I

Other than the tasks listed in our original Phase I proposal we have also performed two extra tasks:

1) Verification/development of a secure broadcast RPC mechanism to enable alarm notification between alarm_agents and alarm_clients.
2) Collection of source code of other intrusion detection systems and integration of these with OSF/DCE intrusion detection efforts.
3) Collaboration with an external consultant.
4) Obtained outside capital funding in support of a Phase II "Fast Track" application.
5) Market and Competitive Analysis

3.4.1 Verification & Development of a Secure Broadcast RPC Mechanism

We proved that secure broadcast RPC was not a viable alarm channel. We investigated an alternative and developed a secure direct (non-broadcast) RPCs using "maybe" execution semantics which produced an effective and secure multicast mechanism.

3.4.2 Integration of Our DCE-based Prototype with Prior Intrusion Detection Systems

We obtained source code from SRI International for their NIDES product, from Purdue University for their IDIOT application, and from U.C. Davis for ExMon work. We have had initial success in integrating with both NIDES and IDIOT. It appears likely that the concepts applied in NIDES and IDIOT can also be applied to DCE (with the additional advantage that a DCE-based environment will better support detection of concerted attacks and faster response). The ExMon application was a postgraduate thesis and was not particularly well documented and was therefore not considered as valuable as the other two products.

3.4.3 Collaboration with an External Intrusion Detection Consultant

We consulted with Dr. Stuart Staniford-Chen of U.C. Davis. The Curriculum management team believes that, in order to bring an excellent commercial product to fruition in Phase II, the use of a well-respected consultant in the Intrusion Detection field is essential. Dr. Staniford-Chen was chosen for his experience of the Common Intrusion Detection Framework (CIDF). It became apparent from discussions with Dr. Staniford-Chen that CIDF APIs could be applied to our Phase II effort. Dr. Staniford-Chen has been retained for extensive consultation in Phase II.

3.4.4 Obtained Outside Capital Funding to Support Phase II "Fast Track" Application.

The company was so pleased with the potential for our proposed Phase II effort that a number of venture Capital firms and private investors were approached with the objective of obtaining matching capital in support of a Phase II fast Track application to DARPA. This "SBIR Fast Track" has, historically, funded over 90% of all applications (compared to 35% or regular Phase II applications). The Phase I results were so compelling that company management was prepared to sell a proportion of the company in order to increase the chances of DARPA continuing this program!

In preparing our phase II proposal, much consideration was given to achieving the significant technical advancements specified in Section 1 within our timeframe and budget. Our Phase II cost proposal number of $500,198 is based upon our "fast track" application for which we obtained $125,049.50 in matching investment equity in support of phase II. The $500,198 of Phase II DARPA funding would provide 3 person-years of development effort. We intend to spend a large proportion of the additional investor equity also in support of our technical goals (thereby adding another person-year of development effort). If we find that additional resources are required, the company is prepared to re-invest up to $200,000 of our earnings (from

commercial operations) over the 22 month period to ensure success for a project that we believe has a huge commercial market.

3.4.5 Market and Competitive Analysis

There were three main factors in determining whether Curriculum Corporation should apply for continuance of this work into Phases II and III. These factors were:

1) Is the idea conceived of in our Phase I application achievable?
2) Is there a market for this product?
3) Is this product unique enough to command a significant market share?

The results of these analyses follow.

3.4.5.1 Is this idea achievable?

The technical challenges to a robust Intrusion Detection System that generates low false alarm rates are significant. Our research in Phase I indicates that this problem, though significant is not insoluble. We believe that by making use of an existing framework (DCE) and some existing IDS components we can develop the product envisaged in our original Phase I application.

3.4.5.2 Is there a market for this product?

Our almost one million dollars of DCE-related sales in 1997 indicate the company's ability to market our existing products.

The widespread implementation of Microsoft's COM and ActiveX technologies (as bundled with Windows 95 and NT) provides a huge market where DCE-compatible systems are already in place. The upcoming release of Active Directory with Windows NT version 5 implements a DCE-like centralized security registry and directory service. These Microsoft technologies use DCE-compatible Remote Procedure Calls (RPCs) already. It is a simple matter for security-conscious sites running NT and Win95 to implement a DCE cell and centralize all password and security requests at the central security service. In this kind of environment, the proposed product would be a natural fit. DCE today, however, is more commonly seen in enterprise-wide applications linking mainframes, midrange systems, and desktops together. Here the true benefits of the proposed product will lie. Every intrusion into an enterprise has the potential to affect every system connected by a network. It therefore only makes sense to implement an Intrusion Detection and Response tool that executes on EVERY platform in an enterprise, from the desktop to the mainframe. Our research involved discussion to a number of our security-conscious fortune 500 customers who today do not make use of an Intrusion Detection System. It became clear that our customers see far less value in an ID solution that is tied to only one operating system. Our heterogeneous solution had far more appeal (especially to very large sites).

A brief list of only a selection of Curriculum Corporation's current customers who are prime candidates for initial site-license sales of the proposed product includes: Bellcore, Boeing, Computer Sciences Corporation, Citicorp, DEC, Environmental Protection Agency, Honda America, IBM, Intel, Lexis-Nexis Corporation, Pacific Bell, T. Rowe Price.

From the list above it should be clear that our established customer base is made up almost exclusively of Fortune 500 customers who care deeply about enterprise-wide computer security. We believe that we can leverage our existing relationships with each of these companies to build up initial sales momentum. With successful implementations installed at these premiere sites, sales to a wider customer base should be easier. One of the hardest tasks faced by Curriculum Corporation in its first years of business was gaining credibility in the computer security marketplace. With this behind us, we feel that we now are well placed to exploit a huge underdeveloped market for a quality commercial (enterprise-wide) Intrusion Detection System.

The DCE (enterprise security) marketplace in which Curriculum Corporation operates is, in 1997, a 200 million dollar market. The largest player in this market, Transarc Corporation (a subsidiary

of IBM), holds around one third of this market (with estimated 1997 sales of 72 million dollars). The market appears to have grown by 60% from 1996 (based on Transarc's 1996 sales of 45 million dollars and constant market share – as reported by the Gartner Group). A conservative measure for the 1999 DCE market size would therefore be 512 million dollars (based upon a constant growth rate of 60% over two years). Given that our proposed tool is applicable to DCE based or NT Domain Controller based environments, it is reasonable to assume that the market for security tools in either of these environments will be at least double that of the DCE security marketplace, or in other words, around one billion dollars in 1999. Since commercial Intrusion Detection Systems have been few and relatively unused to date, it is hard to predict what percentage of the computer security marketplace they may hold by 1999. A very conservative figure would be three percent of a one billion dollar market or about 30 million dollars.

During the last six months of a Phase II development effort, Curriculum intends to hire a marketing and sales manager solely dedicated to the proposed product (funds provided by our company).

3.4.5.3 Is this product unique enough to command market share?

Other companies selling Intrusion Detection Systems include: Intrusion Detection Inc (Kane Security Monitor), Haystack Labs (Stalker), Bellcore/SAIC (CMDS), Digital Equipment Corporation (Polycenter Intrusion Detector), INTRINsec Corp, Trusted Systems Inc, Oddysey Research Associates, Wheel Group (NetRanger).

We shall differentiate the proposed Phase II product through the fact that it operates across the entire enterprise (across many operating systems). This, we believe, is a crucial differentiation that our competitors will not be able to match. In order to achieve this heterogeneity, our product must operate in an environment supporting a centralized security service. We believe that the use of either DCE or NT5.0 Domain Controllers as that security service gives us access to almost all computers in an enterprise (from NT workstation & Windows 95 clients to MVS at the upper end).

3.5 Background Research and Preparation

In order to develop a suitable prototype intrusion detection system, a study of previous research and works was conducted as well as basic platform preparation to conduct development testing and prototyping upon.

3.5.1 Literary Research

In the original proposal, a list of references was identified. As the research effort evolved, this list grew and a representative list of works are illustrated in the Bibliography at the end of this report.

3.5.1.1 An Introduction to Intrusion Detection

Intrusions can be generically divided into six categories [1,2]:

- Attempted break-ins, which are detected by atypical behavior profiles or violations of security constraints.
- Masquerade attacks, which are detected by atypical behavior profiles or violations of security constraints.
- Penetration of the security control system, which are detected by monitoring for specific patterns of activity.
- Leakage, which is detected by atypical use of system resources.
- Denial of service, which is detected by atypical use of system resources.
- Malicious use, which is detected by atypical behavior profiles, violations of security constraints, or use of special privileges.

The basic classes of intrusion detection systems include anomaly detection systems and misuse detection systems. Anomaly detection systems often employ one or more methods to identify

anomalies. Methods such as statistical methods, predictive pattern methods, and possibly neural network methods are common. Misuse detection systems often approach the problem by using technologies such as expert system methods, keystroke monitoring methods, model-based methods, state transition methods, and possibly pattern matching methods. There are still other intrusion detection systems that use techniques like "generic model" methods, network traffic monitoring methods, and possibly various distributed methods using "autonomous agents".

Several issues continue to plague developing solutions to the problem:

- Monitoring/response system overhead - can cost more than it is worth
- False identifications (non-intrusive activity identified as intrusive, or intrusive activity not identified as such) which is sometimes referred to as the signal-to-noise ratio
- Data sources - are the necessary audit events being provided by the subjects and are they in a form usable by an intrusion detection system
- Response techniques - what is appropriate and how do you effect the action

3.5.1.2 Next-Generation Intrusion Detection Expert System (NIDES)

A mature intrusion detection system has been developed since the late 1980s at the Computer Sciences Laboratory of SRI International [3,4,5]. Originally developed as the Intrusion Detection Expert Systems (IDES) it evolved into NIDES in the mid-1990s. This system contains well developed rule-based expert system and statistical components. These components combine together to determine if an event is intrusive or not. This system also includes effective graphical user interfaces, testing of potential rulesets, and other facilities which make the system usable in a "real world".

3.5.1.3 Intrusion Detection In Our Time (IDIOT)

As a graduate student project at Purdue University, Sandeep Kumar and other COAST students developed an intrusion detection system with the acronym IDIOT [6]. Sandeep's design made use of a new classification of intrusion methods based on complexity of matching and temporal characteristics. He also designed a generic matching engine based on colored Petri nets.

3.5.2 Product Research

A study of the current research efforts and market place was conducted to determine the solutions available in the summer of 1997. This was done to learn what benefits and deficiencies each product has.

Several commercial products were investigated but since this effort was a research effort, a focus was placed on freely available software that existed already. We obtained source versions of the NIDES and IDIOT systems. These two systems were studied and an interface developed such that DCE based audit events could trigger intrusion alerts in each system. This effectively proves that we could interface DCE related events with a good portion of existing systems without significant rewrite of the existing systems.

3.5.3 Organizational Research

Since computer system security is a focused discipline and intrusion detection even more narrow within that discipline, there naturally occurs to be a relatively small group of organizations and individuals who constitute the "state-of-the-art" knowledge base. A study of these organizations was conducted to determine who would be of benefit to this research effort.

We found several groups with significant histories of intrusion detection activity. Most notable groups include:

- Defense Advanced Research Projects Agency (DARPA) Information Technology Office (ITO), Arlington, VA
- Computer Science Laboratory (CSL) of SRI International in Menlo Park, CA

- Computer Operations, Audit, and Security Technology (COAST) Project of the Computer Science Department of Purdue University, West Lafayette, IN
- Computer Security Research Laboratory, Department of Computer Science University of California, Davis, CA

We made contacts with each of these organizations and attended conferences as necessary to understand the existing groups. Relationships were developed with selected parties to aid in the interim and subsequent phases of the overall project.

3.5.4 Preparation

As discussed in the original proposal, Curriculum Corporation has resources ready to be used in the conduct of this research. However specific resources were required to actually build and test prototype architectures. These were systems running

Sun Microsystems (SUN): Solaris 2.5.1 and SunOS 4.1.3
International Business Machines (IBM): AIX 4.1.4 and AIX 4.2
Microsoft (MS): Windows NT Server 4.0, Windows 95
Hewlett-Packard (HP): HP-UX 10.10

Any potential Intrusion Detection System solution was expected to audit all of these environments as well as IBM/MVS.

3.6 Principal Areas of Investigation

In order to develop a commercial grade intrusion detection system using the Open Software Foundation (OSF) Distributed Computing Environment (DCE), several technological strengths were investigated.

3.6.1 Use of the DCE Audit Service as a source of Data for Intrusion Detection

With OSF/DCE Version 1.1, a new audit service was introduced. This audit service provided an Application Programming Interface (API) for DCE based applications to use such that events could be recorded by the audit service. The audit service adds routine pieces of information and maintains the control of the storage of the events.

This service as presently constituted in DCE 1.1, which vendors (e.g. IBM, HP) have been shipping since late 1995, is the predecessor technology to the **X/Open Distributed Audit System (XDAS)** which is currently evolving [7]. We fully expect the OSF/DCE Audit Service to keep pace with the XDAS specification and thus our use of this current service enables us to maintain currency without significant rework/change in the intrusion detection system.

The DCE Audit service is capable of being used in either of two main ways:

1) As a means of detecting user-defined events in a user-written application.
2) As a means of detecting any security interaction between a client computer and the Kerberos V security server that serves it. Distributed security systems implementing a centralized Kerberos server are DCE and NT 5.0 Domain Controllers

The latter of these approaches is used by Curriculum Corporation in this project,

The system structure of the DCE Audit Service is based on a client/server approach within the confines of a single operating system image. An administrative interface is provided as a subset of the control program for DCE. The audit service makes full use of the DCE Security Service to protect itself against unauthorized access.

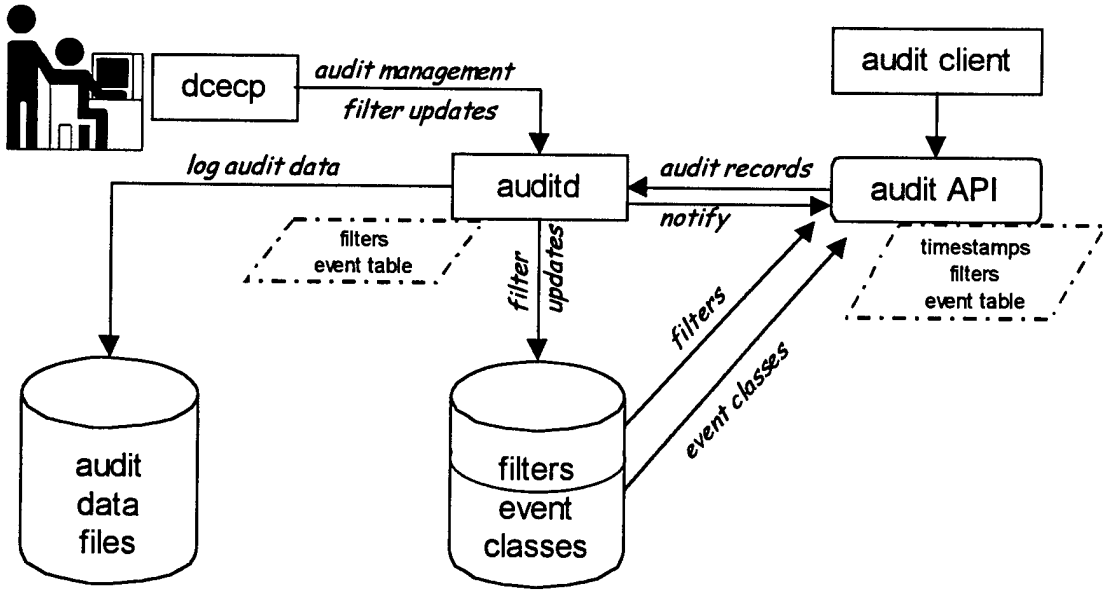

FIGURE 1 THE DCE AUDIT SERVICE

On each system where the DCE Audit Service is to be used, a long running process called auditd will be started as part of the DCE startup. This auditd process manages the recording of data and filters used to limit the data being recorded. The *filters* are used in conjunction with *event classes* which group specific *audit events*.

Each audit client defines what events are to audited by making calls to the audit application programming interface (API) at the desired code points. Audit events are identified by an event number which is a 32 bit integer. Event numbers are a tuple made up of the set-id and event-id thus segmenting the address space for effective management across multiple development groups. These event numbers are then equated to symbolic names within the client program to document the purpose for that event number. A generic example might be:

```
/*  Audit event numbers 0xc0000000 through 0xc00000ff are employee events. */
#define audit_event_query_employee          0xc0000000
#define audit_event_add_employee            0xc0000001
#define audit_event_modify_employee         0xc0000002
#define audit_event_delete_employee         0xc0000003
```

These event numbers are then grouped into event classes for effective filtering of events. The event classes, which are numbered also, can be supersets of the specific events. Example:

```
/*  Audit event numbers 0xc0000000 through 0xc00000ff are employee events. */
/*  Audit event numbers 0xc0000100 through 0xc00001ff are payroll  events. */
/*  Audit event numbers 0xc0000200 through 0xc00002ff are calendar events. */

query_events
        0xc0000000          #audit_event_query_employee
        0xc0000100          #audit_event_query_payroll
        0xc0000200          #audit_event_query_calendar

add_events
        0xc0000001          #audit_event_add_employee
        0xc0000101          #audit_event_add_payroll
        0xc0000201          #audit_event_add_calendar

update_events
        0xc0000001          #audit_event_add_employee
        0xc0000101          #audit_event_add_payroll
        0xc0000201          #audit_event_add_calendar
```

```
        0xc0000002          #audit_event_modify_employee
        0xc0000102          #audit_event_modify_payroll
        0xc0000202          #audit_event_modify_calendar
        0xc0000003          #audit_event_delete_employee
        0xc0000103          #audit_event_delete_payroll
        0xc0000203          #audit_event_delete_calendar

delete_events
        0xc0000003          #audit_event_delete_employee
        0xc0000103          #audit_event_delete_payroll
        0xc0000203          #audit_event_delete_calendar
```

Once event classes are created and accessible by the auditd process, the audit administrator can then invoke filtering policies based on the event classes. Example:

```
audfilter create {group employees} -attribute {add_events failure log}
audfilter create {group employees} -attribute {delete_events success log}
audfilter create {group employees} -attribute {modify_events denial {log alarm}}
audfilter create {principal guest} -attrib {{query_events modify_events} all log}
```

When audit filter updates are communicated to the auditd process, auditd notifies the audit clients that a change has occurred so the API runtime can refresh the in-memory filter table. In this way, filtering is done at the lowest possible level, is efficient, and can be dynamically updated with a consistent user interface at a very granular level.

3.6.1.1 Reading of the Audit Logs

As applications call the audit service API to record events, the audit service saves these events in the form of records within an audit log maintained on a disk resident filesystem. These audit logs are maintained on the system local to the application generating the event. The audit service additionally provides API calls to read the audit logs.

In order for an intrusion detection system to operate in a near real-time mode, the system must receive event information as it happens. As systems get larger and activity increases, the number of events escalates rapidly. It is not uncommon for commercial mainframe operating systems in use today to generate many millions of events in a 24 hour period - on a single system. As computer use grows, these single systems combine into an interwoven network of distributed systems each with strengths and weaknesses. Thus intrusion detection systems must be able to:

1) Examine audit events as they occur,
2) Consolidate audit events across multiple systems separated by a network layer,
3) Be able to handle very large volumes of events without being overwhelmed, and
4) Not be such a burden to the system that the primary purpose of the system is significantly hindered.

Our architecture calls for two layers of Intrusion Detection engines:

1) A detection engine reading the audit trail of a centralized DCE or NT security server, and
2) A lightweight detection engine reading the audit trail of clients.

The first detection engine is intended to detect a varied array of events and is therefore not lightweight. It runs on a specialized server with hardware designed to deal with a significant load.

The second detection engine detects only those patterns that are invisible to the centralized security server (e.g. illicit use of Unix setuid files). The limited scope of this client detection engine allows a lightweight implementation. Meaningful performance tests are not possible until a more production-oriented client is built.

3.6.1.1.1 Using the Audit Service API to Monitor Incoming Events

An early part of our research involved determining how to effectively read the audit events as they happen in an efficient manner. Programs were written using the OSF/DCE Audit Service API to read the various audit logs used by the Audit Service.

To perform this function, we developed a test program called auditListen which runs in the background and does the following:

1) Opens the audit trail using the `dce_aud_open()` API
2) Reads all records from the trail until receiving a NULL using the `dce_aud_next()` API and writes them to an external log file
3) Waits until new record(s) is/are written to the audit trail
4) When a new record is sensed, it reads the record(s) from the end of the audit trail using the `dce_aud_next()` API and appends them to the external log
5) goto (3)

We discovered that after auditListen opened the file using `dce_aud_open()` and read all the records using `dce_aud_next()`, it could never "see" any more updates to the file. It turns out that when you use `dce_aud_next()` and read until you get until the NULL, you never are able to see anything else added to the end of the file. ie. the next record from a NULL is, well, NULL.

We developed a new function within auditListen called `dce_aud_backup()` which decrements the current audit record position pointer a specified amount, in this case, 1 record. Therefore, the new version functions as follows:

1) Opens the audit trail using the `dce_aud_open()` API
2) Reads all records from the trail until receiving a NULL using the `dce_aud_next()` API and writes them to an external log file
3) Call `dce_aud_backup()` to backup to the last record and NOT the NULL pointer
4) Waits until new record(s) is/are written to the audit trail
5) When a new record is sensed, it reads the record(s) from the end of the audit trail using the `dce_aud_next()` API and appends them to the external log
6) goto (3)

This seemed to work under normal circumstances, but still two problems existed:

1) What happens when audit trail reached max size and rolls over?
2) What happens when someone rewinds the point at which events are logged?

The DCE Audit Service allows the log file to be managed in several ways. The default condition is to write to the log file until the value specified in the DCEAUTDITTRAILSIZE environment variable is attained, then logging is stopped. Alternatively, if the "wrap" option is used, once the maximum size has been achieved, auditd repositions the "next record" pointer to the beginning of the file thus rewriting over the file in a wrapping manner. Additionally, at any time, an audit administrator can "rewind" the log file via a command to auditd thus repositioning the "next record" pointer to be the beginning of the log file in effect manually evoking the wrapping function.

In each case, auditListen needed to sense this, close the audit trail file using the `dce_aud_close()` API and reopen it. Without this, auditListen's "next record position" in the file would not coincide with auditd's current record position. Since these conditions do not happen with great frequency, closing and opening the file does not add that much overhead and is a reasonable solution. auditListen was modified to do this so now it performs the desired function using the standard DCE Audit Service APIs.
[See Appendix A for related source code]

3.6.1.1.2 Consolidating Audit Service Events Across Multiple Systems

Since the audit service records events only on the system upon which it runs, filtering and forwarding these events into a central repository was needed to get a "overall view" of the entire distributed system.

Once auditListen was written, the next step was to develop a multi-system audit log consolidation methodology such that a "single system image" could be represented. While DCE offers great benefits in that the DCE Security Service is a centralized function, there is still need to consolidate audit data when the DCE Security Service is replicated on multiple systems. Further, should an activity wish to develop their own audit events for inclusion in the intrusion detection system, an architecture was needed to "see what was going on" across multiple systems.

Using the DCE Directory Services, in conjunction with the Remote Procedure Call (RPC) and Security Services, an architecture was developed that allows granular control of the consolidation and failure/backup recovery mechanisms. A new process, auditCentral, serves as a collector of audit log data forwarded from one or more auditListen processes. The auditCentral server publishes its interface in the DCE Directory Service using a Name Service Interface (NSI) Profile Entry and a NSI Server Entry. This allows the auditListen clients to locate multiple collection servers in a prioritized manner. The use of the Directory Service also allows servers to be added and deleted to the configuration without system disruption. Use of the prioritization inherent with the NSI Profile Entry, allows a controlled distribution of data collection as necessary.

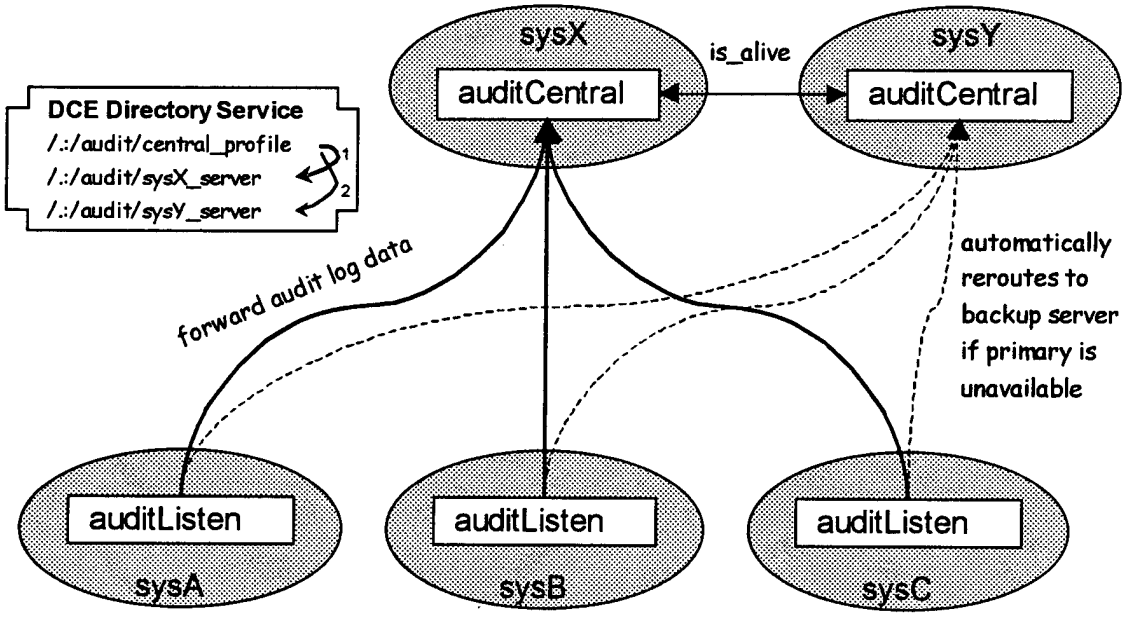

FIGURE 2 THE AUDITCENTRAL ARCHITECTURE

Each auditListen client uses secure RPC APIs to forward audit log data to the auditCentral server. If for some reason the primary server fails to respond, the auditListen clients automatically start forwarding the data to the backup server. When the primary is available again, the backup notifies the clients to relocate the primary.

This architecture has yet to be developed, but since this is a natural DCE based client/server structure that we have developed previously, we have total confidence the structure will work with good success and performance.

3.6.1.1.3 Making Sense of the Audit Records

Since DCE 1.1 leaves it up to the application writer to determine what events should be audited, a study of what events the DCE Security Service and Audit Service record was needed. The code points chosen by the OSF/DCE development staff are in effect static as they are compiled into

the DCE core programs - none can be added or modified. With the use of audit filters, the audit administrator can effectively ignore any of the existing code points.

The below list shows the discrete events coded in the DCE Security and Audit services. Included in the listing are: 1) the symbolic name for the event, 2) event number (hexadecimal and decimal), and 3) one or more default event classes within which that event is defined and the corresponding event class number for that event class.

```
DCE SECURITY SERVICES EVENTS
EVENT SYMBOLIC NAME        EVENT NUMB {EVENT_CLASS_NAME EVENT_CLASS_NUMBER}
AS_Request                 0x0101 257 {dce_sec_authent 10}
TGS_TicketReq              0x0102 258 {dce_sec_authent 10}
TGS_RenewReq               0x0103 259 {dce_sec_authent 10}
TGS_ValidateReq            0x0104 260 {dce_sec_authent 10}
ACL_Lookup                 0x0105 261 {dce_sec_control 12} {dce_sec_query 13}
ACL_Replace                0x0106 262 {dce_sec_control 12} {dce_sec_modify 11}
ACL_GetAccess              0x0107 263 {dce_sec_control 12} {dce_sec_query 13}
ACL_TestAccess             0x0108 264 {dce_sec_control 12} {dce_sec_query 13}
ACL_TestOnBehalf           0x0109 265 {dce_sec_control 12} {dce_sec_query 13}
ACL_GetMgrTypes            0x010A 266 {dce_sec_control 12} {dce_sec_query 13}
ACL_GetReferral            0x010B 267 {dce_sec_control 12} {dce_sec_query 13}
PRIV_GetPtgt               0x010C 268 {dce_sec_authent 10} {dce_sec_control 12}
ACCT_Add                   0x010D 269 {dce_sec_control 12} {dce_sec_modify 11}
ACCT_Delete                0x010E 270 {dce_sec_control 12} {dce_sec_modify 11}
ACCT_Rename                0x010F 271 {dce_sec_control 12} {dce_sec_modify 11}
ACCT_Lookup                0x0110 272 {dce_sec_control 12} {dce_sec_query 13}
ACCT_Replace               0x0111 273 {dce_sec_control 12} {dce_sec_modify 11}
ACCT_GetProjlist           0x0112 274 {dce_sec_control 12} {dce_sec_query 13}
LOGIN_GetInfo              0x0113 275 {dce_sec_control 12} {dce_sec_query 13}
PGO_Add                    0x0114 276 {dce_sec_control 12} {dce_sec_modify 11}
PGO_Delete                 0x0115 277 {dce_sec_control 12} {dce_sec_modify 11}
PGO_Replace                0x0116 278 {dce_sec_control 12} {dce_sec_modify 11}
PGO_Rename                 0x0117 279 {dce_sec_control 12} {dce_sec_modify 11}
PGO_Get                    0x0118 280 {dce_sec_control 12} {dce_sec_query 13}
PGO_KeyTransfer            0x0119 281 {dce_sec_control 12}
PGO_AddMember              0x011A 282 {dce_sec_control 12} {dce_sec_modify 11}
PGO_DeleteMember           0x011B 283 {dce_sec_control 12} {dce_sec_modify 11}
PGO_IsMember               0x011C 284 {dce_sec_control 12} {dce_sec_query 13}
PGO_GetMembers             0x011D 285 {dce_sec_control 12} {dce_sec_query 13}
PROP_GetInfo               0x011E 286 {dce_sec_control 12} {dce_sec_query 13}
PROP_SetInfo               0x011F 287 {dce_sec_control 12} {dce_sec_modify 11}
POLICY_GetInfo             0x0120 288 {dce_sec_control 12} {dce_sec_query 13}
POLICY_SetInfo             0x0121 289 {dce_sec_control 12} {dce_sec_modify 11}
AUTHPOLICY_GetInfo         0x0122 290 {dce_sec_control 12} {dce_sec_query 13}
AUTHPOLICY_SetInfo         0x0123 291 {dce_sec_control 12} {dce_sec_modify 11}
REPADMIN_Stop              0x0124 292 {dce_sec_control 12} {dce_sec_server 14}
REPADMIN_Maint             0x0125 293 {dce_sec_authent 10} {dce_sec_control 12}
{dce_sec_server 14}
REPADMIN_Mkey              0x0126 294 {dce_sec_control 12} {dce_sec_server 14}
REPADMIN_Destroy           0x0127 295 {dce_sec_control 12} {dce_sec_server 14}
REPADMIN_Init              0x0128 296 {dce_sec_control 12} {dce_sec_server 14}
ERA_Update                 0x012B 299 {dce_sec_control 12} {dce_sec_modify 11}
ERA_Delete                 0x012C 300 {dce_sec_control 12} {dce_sec_modify 11}
ERA_TestUpdate             0x012D 301 {dce_sec_control 12} {dce_sec_query 13}
ERA_LookupById             0x012E 302 {dce_sec_control 12} {dce_sec_query 13}
ERA_LookupNoExpand         0x012F 303 {dce_sec_control 12} {dce_sec_query 13}
ERA_LookupByName           0x0130 304 {dce_sec_control 12} {dce_sec_query 13}
ERA_SchemaCreate           0x0131 305 {dce_sec_control 12} {dce_sec_modify 11}
ERA_SchemaDelete           0x0132 306 {dce_sec_control 12} {dce_sec_modify 11}
ERA_SchemaUpdate           0x0133 307 {dce_sec_control 12} {dce_sec_modify 11}
ERA_SchemaLookupId         0x0134 308 {dce_sec_control 12} {dce_sec_query 13}
ERA_SchemaLookupName       0x0135 309 {dce_sec_control 12} {dce_sec_query 13}
PRIV_GetEptgt              0x0136 310 {dce_sec_authent 10} {dce_sec_control 12}
PRIV_RdlgTokInfo           0x0137 311 {dce_sec_authent 10}
PRIV_BecomeDelegate        0x0138 312 {dce_sec_authent 10} {dce_sec_control 12}
PRIV_BecomeImpersonator    0x0139 313 {dce_sec_authent 10} {dce_sec_control 12}

AUDIT SERVICE EVENTS
delete filter              0x300  768 {dce_audit_filter_modify 0x30}
```

```
list filter              0x302  770 {dce_audit_filter_query 0x31}
add filter               0x303  771 {dce_audit_filter_modify 0x30}
remove filter            0x304  772 {dce_audit_filter_modify 0x30}
modify sstrategy         0x305  773 {dce_audit_admin_modify 0x32}
modify state             0x306  774 {dce_audit_admin_modify 0x32}
rewind                   0x307  775 {dce_audit_admin_modify 0x32}
stop                     0x308  776 {dce_audit_admin_modify 0x32}
show sstrategy           0x309  777 {dce_audit_admin_query 0x33}
show state               0x30a  778 {dce_audit_admin_query 0x33}
```

Each audit event may add additional event specific information to the tail of the audit record. For example, the ACL_Replace (0x106) event adds:

```
char              *component_name
uuid_t            manager_type
sec_aud_type_t    acl_type
sec_acl_list_t    old_acl_list
sec_acl_list_t    new_acl_list
```

In this way, the audit record can contain as much data as the code point would offer. In this example, you would know who changed what when.

Our investigations have shown that not every well-known UNIX or NT intrusion will generate an event at a centralized security server. Those intrusions that are "invisible" to this centralized location can be programmed as rules to our local (lightweight) detection engine.

3.6.1.1.4 Filtering Out the Noise

The Phase I tests conducted showed that audit events can be easily filtered such that only the "interesting" events are actually recorded in the audit logs. This is done through the use of event classes which can be dynamically added/changed/deleted within an running system. This "upstream" filtering reduces the CPU overhead associated with our detection engines.

Filters allow discrete events to be handled in specific ways for specific resources. Audit filters are applied with a *filter name* which incorporates the DCE Registry identity as follows:

principal	keyed with local DCE cell principal (user) name
foreign_principal	keyed with fully qualified principal (user) name
group	keyed with local DCE cell group name
foreign_group	keyed with fully qualified group name
cell	keyed with fully qualified cell name
cell_overridable	keyed with fully qualified cell name
world	unkeyed - applies to all entities
world_overridable	unkeyed - applies to all not otherwise specified

It should be clear that, in environments where client performance is a major concern, it is possible to filter out events that are attributed to certain classes of users. For example, the audit system events associated with obtaining administrator privileges only could be examined by our ID engine (i.e. throw away other events upstream of the ID engine). This would reduce the usefulness of the tool but may make it viable in certain sites.

Next filters are applied based on audit conditions which can be:

success	the requested event being audited was successful
denial	the requested event being audited was disallowed
failure	the requested event being audited was unsuccessful
pending	the status of the event being audited was not yet determined

Finally, the filters specify what actions to take when a match occurs. Possible actions are:

alarm	write a log event to the system console
log	write a log event to the log file
none	take no action - send event to bit bucket

all write a log event to BOTH the system console and log file

Along with the event class name, these three components are needed to complete the specification of the audit filter. Through the use of world_overridable and the more specific filters, one can ensure nothing "slips thorough the cracks" as time goes on.

As a practical example of how to use DCE audit filters, suppose we have the payroll application which has the following four code points where audit API calls are made:

```
#define audit_event_query_employee          0xc0000000
#define audit_event_add_employee            0xc0000001
#define audit_event_modify_employee         0xc0000002
#define audit_event_delete_employee         0xc0000003
```

We have further defined two event classes which cover these discrete events:

```
employee_query
        0xc0000000          #audit_event_query_employee

employee_modify
        0xc0000001          #audit_event_add_employee
        0xc0000002          #audit_event_modify_employee
        0xc0000003          #audit_event_delete_employee
```

Suppose also, that we have specified (as a separate activity) that all employees of the Payroll Department are members of the "payroll" security group within the DCE Security Service. We can then specify the following filters via the `dcecp` command interface:

```
audfilter create {world_overridable} -attribute {employee_query success log}
audfilter create {world_overridable} -attribute {employee_modify all all}
audfilter create {group payroll} -attribute {employee_query all none}
audfilter create {group payroll} -attribute {employee_modify success log}
```

These filters combine together to allow the Payroll Department employees to be treated differently than all other employees. Payroll employees may query records without any logging, and update records with logging, while all other employees with successful queries have the activity logged and any attempts to modify are both logged and written to the system console.

Of course, the specification and modification of audit filters is controlled via ACLs within the audit service. Activities within the audit service engage audit service APIs to complete the control and logging needs of such a service - it uses itself to record activities within itself.

3.6.2 Secure Communications for Network Transmissions between IDS Components

As mentioned above, DCE Audit Service events need to be consolidated across multiple systems within the network. Additionally, for certain intrusions, local agents on each system in the network could take actions based on notification from some central authority. Hence the need for reliable and secure communications across the network.

3.6.2.1 DCE Remote Procedure Call (RPC)

DCE uses the Remote Procedure Call (RPC) mechanism for application to application communication. The use of RPC as a programming paradigm is well understood and widely used within the industry. DCE allows RPCs to flow from one application to another with several security options and various execution semantics.

3.6.2.1.1 RPC Security Options

The security levels available to the application programmer via the DCE RPC API are as follows:

rpc_c_protect_level_default

Default protection level of the authentication service

rpc_c_protect_level_none

No protection.

rpc_c_protect_level_connect

Protection provided only when the client establishes the connection with the server. An initial encrypted "handshake" indicates the client and server are who they purport to be. No further checking or encryption is done after the initial call.

rpc_c_protect_level_call

Protection provided on each RPC call to the server to ensure the two parties have not changed. This effectively performs the connect level of protection for each RPC call. Multiple network packets are not protected and no encryption is done. (Connection-based protocol sequences such as *ncacn_ip_tcp* use *rpc_c_protect_level_pkt* instead.)

rpc_c_protect_level_pkt

Protection provided on each RPC call and every message to ensure the two parties have not changed. This effectively performs the call level of protection for every packet sent between the two parties. Other than the encrypted "handshaking" of each packet, the data within the data packets is not encrypted.

rpc_c_protect_level_pkt_integrity

Protection provided on each RPC call and every message (like pkt above) with the additional feature that a cryptographic checksum is computed and added to each message to ensure the data within the packet has not been changed. This is the highest level of security that can be guaranteed to be available in the DCE runtime base (see pkt_privacy below).

rpc_c_protect_level_pkt_privacy

Protection provided as with pkt_integrity except the data within the packet is encrypted. This is the highest level of privacy available - every message is encrypted before it crosses the wire. Since the packet data is normally encrypted using the Data Encryption Standard (DES), which has export restrictions by the United States Government, it may not be present in the runtime that performs the encryption. Applications that request this level of protection without support in the runtime, will receive an error. As with any cryptography, there are significant performance costs associated with using cryptographic checksums or data encryption as in these highest two protection levels.

Our Phase I applications were designed to implement *rpc_c_protect_level_pkt_privacy* as the default security level. This provides the highest level of security but at a cost in terms of performance. It appears that RPC's using this security level are around twice as slow as unencrypted RPC's. The communications infrastructure appears not to be the bottleneck that we should be concerned with however. The client detection engine is expected to consume far greater CPU resources. If required, in Phase II, we can re-evaluate whether some communications between our components can implement *rpc_c_protect_level_pkt_integrity* in order to improve performance slightly.

3.6.2.1.2 RPC Execution Semantics

DCE supports several execution semantics for RPCs such that the sender can be assured how the intended receiver will receive the RPC packets. The execution semantics available to the application programmer are as follows:

at-most-once

The operation must execute once, partially, or not at all. This is the default and most commonly used form of execution semantics.

idempotent - maybe

The operation may execute once, partially, not at all, or many times without causing negative behavior. The maybe form allows the caller to neither require

nor receive any negative indication even though the operation might not have executed successfully even once. No return parameters are permitted.

idempotent - broadcast

The operation may execute once, partially, not at all, or many times without causing negative behavior. The broadcast form allows the caller to send to one server on each system on the local network with once reply being received. This method does not require a client to connect to a particular server, but is limited by the boundaries of broadcast propagation through the network to which the caller is attached.

As will be explained below, there were components of our architecture proposed in the Phase I application that used RPCs using broadcast semantics. Broadcast RPC was attractive as an alarm mechanism that might be both very secure and very fast. The discussion that follows addresses the limitations we discovered in authenticated idempotent broadcast RPC.

3.6.2.1.3 Secure Broadcast RPC

As discussed above, secure broadcast RPC was initially considered an ideal form of communication between our intrusion detection components in the case of an alarm which necessitated a widespread response. The idea was to combine the *rpc_c_protect_level_pkt_privacy* security level with idempotent broadcast semantics. Unfortunately, we discovered that such a program cannot be written. The concept of a "secret key" based security model (such as Kerberos Version 5 upon which DCE was originally based) does NOT fit with the generalized "network broadcast" semantics initially desired. Each party in an authenticated communication requires a secret key (distributed by a Kerberos Key Distribution Center in this case). In a broadcast RPC, there is no knowledge of all the potential receivers of the broadcast and therefore Kerberos cannot provide secret keys to the broadcast receivers. However through successive efforts, a reasonable alternative solution was found.

3.6.2.1.4 Secure Broadcast RPC Alternative

Our objective was to develop a scalable secure way for many clients to receive alert notifications from an agent as well as possibly forward information back to the agent for propagation and management notification. Since the *broadcast* mechanism was proven to be insecure, it was abandoned as a useable execution semantic. Instead, the *maybe* idempotent semantic was used. An RPC using *maybe* semantics does not require an acknowledgement from a receiver. This leads to a very fast RPC from the sender's point of view.

In the context of our prototypes, a client at startup, builds a list of servers that should "hear" its alert (if an intrusion is detected locally). Upon detection of an intrusion, the client effectively performs a fast multicast to all of the servers in its list. A secure session key has already been obtained by the client for conversations with each of these servers. The *maybe* semantics of the client RPC allows it to RPC alarm to the next server as soon as it has sent an alarm to the first server on its list. This means the client RPC does not block waiting on an acknowledgement of receipt from any server.

Using the classical DCE client/server paradigm, we created an alarm_agent program that registers itself in the DCE Directory Service so clients can locate the agent of choice. This can be illustrated as follows:

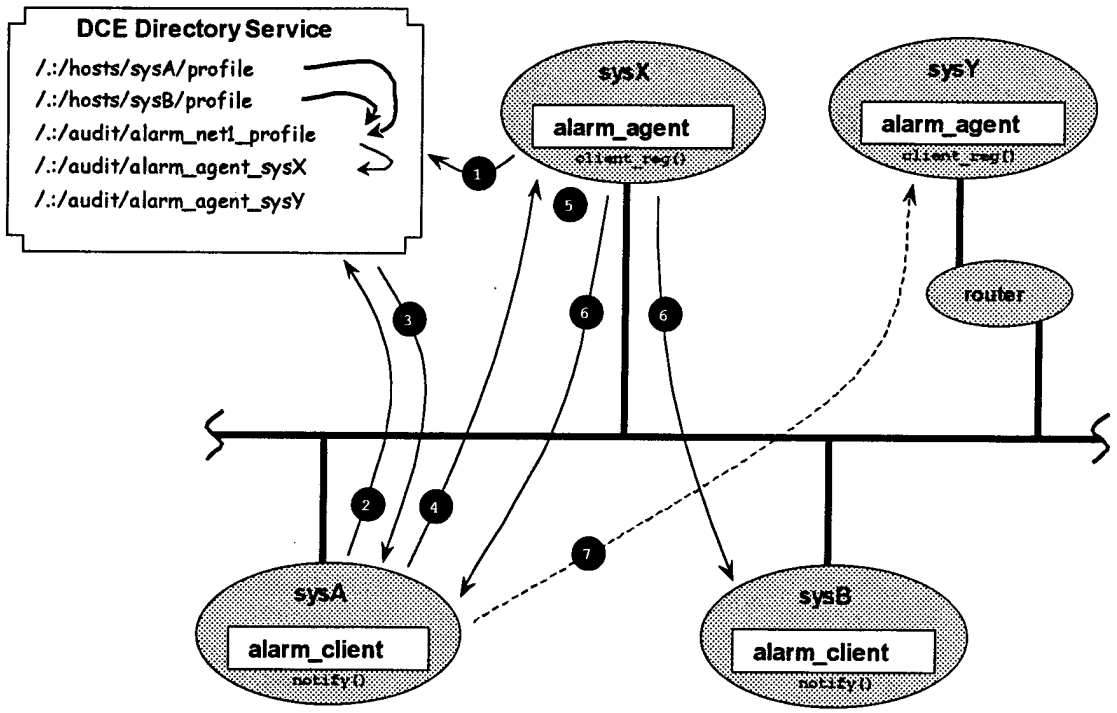

FIGURE 3 THE SECURE BROADCAST RPC ALTERNATIVE

The entire process takes the following steps:

1) When started, the alarm_agent server registers its location in the DCE Directory Service and local endpoint map as a server should to "advertise" its availability.
2) When started, the alarm_client contacts the DCE Directory Service to locate the server it should contact.
3) The DCE Directory Service stores a series of Profiles, Groups, and Server Entries which the client searches through to locate the alarm_agent it should contact. In the illustration above, sysA will contact sysX since it is first in the prioritized Profile entry.
4) The alarm_client registers with the alarm_agent using the alarm_agent's `client_reg()` interface. This is a secure point-to-point RPC call, so both parties know the valid identity of each other. It is a short lived request whose only purpose is to let the alarm_agent know that the alarm_client is out there and wishes to be contacted by that alarm_agent whenever the alarm_agent issues notifications.
5) The alarm_agent uses the DCE RPC API `rpc_binding_server_from_client()` to convert the RPC binding handle obtained from step 4 into a server binding handle. This dynamically *reverses* the client/server relationship, so the alarm_agent can be a client of the alarm_client server. Once the binding handle is converted, it is stored (in memory) by the alarm_agent so it will know what clients to contact when a notification is to be sent.
6) When a notification needs to be sent, the alarm_agent loops through its list of known alarm_clients to retrieve the server style binding handles for the alarm_clients. The alarm_agent then uses a secure RPC with the ***maybe*** idempotent execution semantic to send the notification to each alarm_client via the `notify()` interface that the alarm_client supports. Since the maybe RPC allows no return parameters, the alarm_agent proceeds without knowledge/concern that the alarm_client has

received the notification. This achieves the same effect as the broadcast execution semantic yet provides the security desired.

7) Since the DCE Directory Service is being used in this architecture, greater flexibility and control can be achieved. In the original design, a router that did not pass broadcast requests would cause the audit administrator to create new instances of alarm_agents for each segment. In this new design, clients/agents can locate each other even across routers and thus achieve a greater degree of reliability in an organization where the audit administrators and network administrators may not always synchronize their efforts. It can also have the effect of providing a failover/backup mechanism in the event of a primary alarm_agent being down or unavailable for some reason.

The prototype was developed and proven using the above methods with good success. We believe this is a better long term solution than the "secure broadcast RPC" mechanism proposed originally. [See Appendix A for related source code]

3.6.3 Detecting Intrusions using Centralized Security Services

Paramount to our design is the desire to limit the CPU bandwidth used by our client Intrusion Detection component. To that end, the more intrusion data we can direct to our centralized detection component, the lower the load will be on the clients. It became apparent in our Phase I proof of concept work that there would likely be few applications making direct calls for authentication to a centralized security component unless that component is integrated into the base operating system from the ground up. With both the DCE security server and the NT 5.0 domain controller (as advertised) options exist to integrate what would otherwise be local authentication and authorization requests into a centralized server. Much effort was expended, therefore, on ensuring that:

1) We can integrate UNIX and NT security into a central domain, and
2) The effect of this integration is that local authorization/authentication requests are auditable centrally.

3.6.3.1 Integrated Login and Single Sign-On

The vendors providing DCE to the market place have responded to customer demands for single sign-on capabilities by leveraging the DCE identity credentials across the environment. These credentials are used by the various "security aware" programs to allow access in place of prompting for a userid/password. The credentials are specially encrypted such that neither the sender nor recipient can decrypt the information, but only the DCE Privilege Service can encrypt and decrypt it. This ensures that the "trusted third party" protocol is maintained across the environment. Whenever an authentication function is required, the encrypted Extended Privilege Attribute Certificate (user credential) is presented in lieu of a userid/password. Thus no passwords are sent across the network, thereby strengthening the enterprise's security posture.

The term "integrated login" often refers to an implementation where two security paradigms are used at the same time (local operating system security and DCE security), but the authentication process has been combined into one event. When the user conducts a login, the same userid and password are used to authenticate to both security paradigms at once. This integrated approach has the desirable event of a single user action, and yet allows legacy security methods to be supported without significant change.

The term "single sign-on" often refers to an implementation where there are may be two security paradigms but the DCE security provides the actual authentication process on behalf of both paradigms. This has the significant advantage that the administrative burden of maintaining two security paradigms (and synchronizing the multiple databases) is reduced to half.

When DCE was first designed by the OSF, the single sign-on paradigm was the intended design goal. However, the implementation was left to the vendor's discretion. Though this area has

been languishing for some time, it has gained renewed emphasis and development effort from several vendors - IBM most notably. Without these implementations, the user must first authenticate to the local operating system, then perform a separate dce_login to authenticate to the DCE security. This method can have advantages when a significant portion of the users will not be using DCE.

DCE is often employed to provide enhanced security and file services throughout an enterprise, thus single sign-on is of great interest to serious enterprises. IBM suggests an annual savings for an enterprise of about 10,000 users to be over $8,000,000 dollars in end-user productivity gains - administrative and security enhancement savings would be additive to that number. [9]

3.6.3.2 Integrated Login under Microsoft's Windows 95 and Windows NT

The Microsoft platforms have been supported by several vendors who provide the DCE product. Most notable in the field is Gradient Technologies [10] who has focused on Windows from the outset. They offer a wide range of DCE based products. In their implementation, when the DCE software is installed, you are provided the option to use "integrated login". This allows the user to have a userid in both the Windows and DCE security databases, with changing of passwords being done in both places at once. The user integrated login is limited to the local system, meaning that while Windows security token are passed from system to system, the DCE credentials are not, thus forcing a DCE re-authentication at each system boundary. With the use of DCE/DFS and truly distributed services, the user need not cross the system boundary so the impact is limited.

International Business Machines (IBM) and Digital Equipment Corporation (DEC) have provided Windows based DCE products for some time as well. IBM's version is a serious competitor to the Gradient product and likewise has an enabled integrated login facility. [11]

Since NT 5.0 Domain Controllers (running under Active Directory) were not released at the time the Phase I work was performed, it was impossible to test the equivalent integrated network login through the Microsoft-centric solution. We expect to confirm Microsoft's claims of seamless integration of NT security into a Kerberos V based environment as soon as NT 5.0 becomes commercially available.

3.6.3.3 Single Sign On under IBM's AIX

IBM has made great commitment to DCE development and has thus changed their AIX operating system to embrace the DCE security paradigm. [11] This means that the Systems Administrator adds users to the DCE Registry and has no need to add the user to local operating system files. An extra process on the local operating platform performs the gateway operation between local authentication and DCE authentication.

Of all vendors, IBM appears most committed to developing the most sophisticated single sign on product base. On January 13th, 1998, IBM announced a significant enhancement and extension to their Global Sign-On (GSO) product line, which takes the single sign-on to new heights. [12]

3.6.3.4 Integrated Login under HP's HP-UX

Hewlett-Packard (HP) [13] has also made great commitment to DCE development and has likewise changed their HP-UX operating system utilities to embrace the DCE security paradigm while allowing a fallback to the traditional HP-UX paradigm if DCE security is unavailable. While not as robust as IBM's approach, it is a significant capability which aids in supporting a centralized security model, and is manageable for an enterprise to implement.

3.6.3.5 Integrated Login under SunSoft's Solaris using Transarc's DCE

Hindered in that they do not own/control the operating system, Transarc [14] has been slow to implement integrated DCE solutions with Solaris. SunSoft has not chosen to really embrace the

DCE technology leaving Transarc as the primary DCE vendor in for this operating platform. Transarc is the DCE/DFS technology provider on almost all platforms.

3.6.3.6 Remote Access Security using SnareWorks

Provided by IntelliSoft, SnareWorks [15] is a product that can insert DCE security into an otherwise uncooperative environment. SnareWorks is able to insert itself in the socket layer and intercept packets as they flow from remote systems to applications on the local system. Employing "Protocol Support Modules" and predefined entry points, SnareWorks can enforce security rules with DCE ACLs. Using this technology on the sender and receiver side, credentials can be provided for authentication. This effectively provides a full single sign-on solution not only for the traditional system utilities (login, telnet, ftp, etc.), but any socket based inter-process communication desired by an enterprise. While not as easy to implement for an enterprise, it is the most flexible and extensible of all.

3.6.4 Response Techniques

Our prototype response tools have three key elements for response to intrusion:

1) Access to centralized security registry,
2) Authentication credentials (from our use of authenticated RPC),
3) Access Control Lists (ACLs).

The DCE Security infrastructure, around which our Phase II production response tools are based, is based to two contributing technologies: 1) MIT Project Athena's Kerberos Network Authentication Service, Version 5 and 2) HP/Apollo's centralized password registry (database).

3.6.4.1 The Security Model applied to Response Capability

The DCE Registry is the centralized and replicated security database. It contains information about all the security entities, called *principals*, in the DCE cell (a logical collection of systems). The association of a principal with a password, a primary *group*, *organization* and other information comprises an *account*. A user/server must have an account in order to authenticate to the DCE Security Service.

An important guard against intrusion is to ensure the strength of passwords. DCE allows the definition of the following password policies in the normal Registry schema:

- lifespan
- expiration date
- minimum length
- consist of all spaces
- consist of all alphanumeric characters

DCE 1.1 provides an integrated password strength subsystem, which may be used to extend the management of password strength and password generation. DCE provides an example password strength server which may be customized. This password strength subsystem relies on the use of ERAs in the Registry to require a principal's password be validated by the subsystem. One of the response techniques listed below relies on the ability to customize password strength checking to prevent use of passwords that have been compromised.

Access Control Lists (ACLs) are used to control access to objects (resources) in DCE. These POSIX 1003.6/Draft 3 based ACLs contain a list of users and groups that are allowed access to a resource and what type of access is allowed. Applications may grant access to resources with a high degree of granularity (e.g. unauthenticated users, specific users from a foreign cell). The use of ACLs requires the application responsible for the resource to provide one or more *ACL Manager* routines. These routines are used to compare the authorization information sent from the client against the ACL entries from the ACL database residing within the application. Access is granted or denied based on the outcome of the comparison.

Authentication credentials are obtained when a user authenticates with the DCE Security Service (i.e. logs into the DCE cell). They expire after a specific period of time. Credentials are used to prove the user's identity in the cell without having to provide a password again. The length of time that credentials are valid is determined by authentication policy within the Registry. Credentials contain of three types of information: TGT, PAC, and service tickets. The *Ticket Granting Ticket (TGT)* is used to prove the identity of the user and can only be decrypted by the Security Service. The *Privilege Attribute Certificate (PAC)* contains the project list, simply a list of authorization groups in the Registry to which the principal belongs. In DCE 1.1, the PAC was enhanced and is now properly referred to as the *Extended Privilege Attribute Certificate (EPAC)*. A *service ticket* is obtained by the client in order to request service from a server. The service ticket proves to the application server the client's identity. The ACL manager compares the PAC to the ACL protecting the resource to determine whether the client is authorized to perform the operation requested.

Non-human users, servers, use *keytab* (key table) files to store their account passwords. A keytab contains one or more principal/password pairs. It is normal file, which is read by an application when authenticating to DCE. Thus all principals "login" to DCE.

The below diagram illustrates the interaction between the various parts of DCE security.

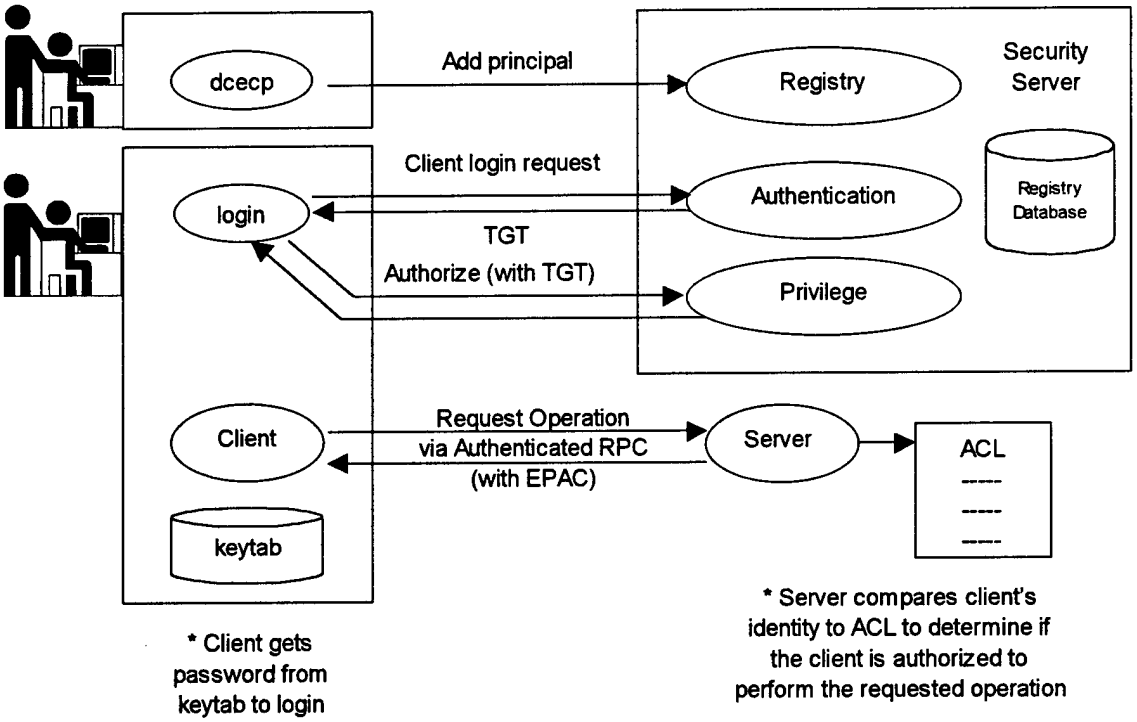

FIGURE 4 THE DCE AUTHENTICATION PROCESS

3.6.4.2 Using DCE Security Services as a Response Technique

Based upon our Phase I proof of concept work we are confident that, once an intrusion has been detected, one or more of the following actions may be used to respond to the threat.

3.6.4.2.1 Prevent ticket renewal

We are able to set the *maxtktrenew* attribute on the compromised account to zero to prevent existing credentials from being renewed. The renewable ticket lifetime determines the maximum length of time tickets may be refreshed before new login is required.

3.6.4.2.2 Invalidate existing credentials

Our response prototype is able to set the *goodsince* attribute on the account to the current date/time and generate a new password for the account. Any tickets granted before the date and time indicated in the *goodsince* attribute will not be valid. To obtain new credentials, the new uncompromised (hopefully) password must be used.

3.6.4.2.3 Remove access to threatened object

If permission is granted to the singular principal whose account has been compromised, then we may change the ACL protecting the object to give no permission to the compromised principal. In the case that the compromised account gains access from being a member of an authorization group having permissions to the object, then either add a more restrictive ACL entry reducing the privileges for that principal or remove the principal from that security group.

3.6.4.2.4 Disable compromised account

We may set the *acctvalid* attribute to *no*. This will prevent any future login to the account. Notify security administrators who will decide the next action, perhaps removal of the account. This response will likely be used with other responses depending on the risk assessment of the intrusion.

3.6.4.2.5 Disable administrative accounts (cell_admin and all accounts belonging to administrative groups)

Our prototypes are able to set the *acctvalid* flag on the cell_admin account to *no*, preventing future logins as that privileged user. We can also set the *inprojlist* attribute on the administrative accounts to *no*, preventing it from inclusion in any future Privilege Attribute Certificates (PAC) issued by the Security Service

3.6.4.2.6 Regenerate login key for authenticated application

Generate a new password for the application. It will be stored in both the keytab and the Registry. Remove the previous password in the keytab. Re-start the application to use the new password. This may be handled securely from any system participating in the cell and thus does not require logging into the actual system where the application runs.

3.6.4.2.7 Regenerate login keys for machine principal

Generate a new password for the system (machine). It will be stored in both the keytab and the Registry. Remove the previous password in the keytab. Re-start the security client process (dced) to use the new password. This must be done from the console of the machine to ensure the password is securely changed.

3.6.4.2.8 Shutdown breached application

The application may be shutdown from any machine that participates in the cell by using the DCE application management facilities included in DCE 1.1 core services. This presumes the application has already been configured to be managed through DCE - a one-time configuration task performed by a systems administrator.

3.6.4.2.9 Reinitialize authenticated application

The application may be re-started from any machine that participates in the cell by using DCE application management facilities. This presumes the application has already been configured to be managed through DCE.

3.6.4.2.10 Restore original application binary

If the application binary is stored in DCE's DFS (Distributed File Service) in an unmounted read-write fileset, it could easily be mounted and copied over the compromised binary with the original ACLs intact.

3.6.4.2.11 Notify password strength subsystem (compromised password now considered weak)

If a password is compromised, perhaps by an enhanced dictionary attack, notify the password strength subsystem to prevent users from choosing that password in the future.

3.6.4.2.12 Re-encrypt security database with newly generated key

Force generation of a new master key and re-encryption of the entire security database. This copy of the database would be unavailable during the re-encryption task.

3.6.4.3 Choosing an Appropriate Response Technique

A risk assessment would be performed to determine which actions from the list above is the most appropriate response to the intrusion. DCE's extended registry attributes (ERAs) could be used to store the potential threat level on a per account basis. In this way, when an account is compromised the risk associated with the use of the account could be automatically be assessed and the appropriate response taken without human intervention. Below is one possible assignment of risk to level. Further, ERAs could be used to maintain a list of appropriate responses on an account by account basis, based on a specific threat level.

0 standard access user account
1 standard access server application account
2 confidential access user account
3 confidential access server application account
4 administrative access account

3.6.5 Events Detected by our Central Audit Monitor

Crucial to our entire architecture is the need to detect some proportion of intrusions through our central Audit Monitor (attached to the audit trail of the DCE security service). Using the audit events provided by the core DCE servers combined with the vendor integrated login components, intrusion scenarios were developed.

3.6.5.1 Security Anomalies that can be detected by the proposed Audit Monitor.

The Audit Monitor interfaces with the DCE Audit Service. The DCE Audit Service is able to audit all of the DCE servers that physically reside on that physical computer. One of the most significant server processes that is audited by the Audit Server is the DCE Security Server. The events that the Audit Monitor can locate via the DCE Audit Server are as follows:

Attempts to modify (or delete) parts of the DCE security database (the registry)

This can include just about any change to user account information, group information or a change in a policy of the security service itself.
Changes to Access Control Lists (ACLs) can also be detected in this manner.

Attempts to initialize or delete a DCE security server

It is important to avoid the possibility of an attack that is aimed purely at "denial of service". An attempt to shut down one or more copies of the security server would fit this category.

Any login to the DCE security server computer

Much of DCE security relies on the fact that the systems that hold the security database (the registry) are physically and logically secure. The only time a login request should be entered to a security server is during system maintenance. If a login attempt is detected at any other time, there is the potential that someone is attempting to break in.

Attempts to regenerate the key used to encrypt the security database

The whole of the DCE security database is encrypted on disk. A master encryption key is used to encrypt. An attacker may attempt, through whatever means, to somehow re-encrypt the database with a known key.

Failed client responses to a server's challenge

DCE version 1.1 includes a login protocol that is much more secure than the kerberos protocol from which it was derived. In this newer DCE version, an improved mutual authentication check is initiated between client and server i.e. both clients and servers must prove who they are. If a client or server fails this mutual challenge this is a possible sign of attack.

Detected replays of previous security operations

DCE security relies on ticket transfers across a network. An attacker might attempt to break down security by re-playing an old ticket request.

Invalid ticket requests

Any ticket request that is either an invalid type or a prohibited type should raise an alarm.

The usage of cryptographic keys in the DCE RPC runtime

It is possible to detect any time that any client tries to use some form of DCE-based encryption.

Attempts to change the system time (to re-use expired tickets)

In order to attempt to replay an old (expired) ticket, the system time of a server would have to be set backwards. This condition should always raise an alarm.

Attempts to turn off or modify the DCE Audit Service

These events should clearly be audited !

Attempts to connect to the security server through any means other than authenticated DCE RPC

The DCE security server should only be accessed by authenticated DCE RPC. Any other form of attempted access e.g. ftp, telnet, mail, etc is a likely sign of attack.

The correlation of these esoteric events to known intrusions is the heart of our detection philosophy. This correlation makes up the rules of our rule based detection tools. A brief six month proof of concept effort does not provide the time to tie large numbers of well-known intrusions to specific patterns of events above, however, based on our research, we are confident that some proportion of intrusions can be detected in this manner. The remaining well-known intrusions shall be detected by rules applied at the local client monitors.

3.6.5.2 Responses

Based on our research, in response to an alert, the following actions can be implemented across the entire DCE cell:

Regeneration of the machine key for each affected system.
DCE uses a machine key (like a password) as proof of a client computer's identity. This key is also used as the basis for the encryption keys that are used to transmit information from a client to the security server). After a suspected security breach, regeneration of this key will ensure that an attacker can no longer use a key he or she has obtained.

Modification to the behavior of the DCE password strength subsystem
DCE implements a password strength server to control the quality of user passwords in a security-sensitive environment. It is possible to raise the limitations on the passwords that users are allowed to create and therefore impose stricter password strength policies.

Disabling accounts for which there have been attacks
This is one of the most common actions currently taken as a result of a security alert. In a DCE-based environment, with its centralized security service, it is possible to disable a particular account across thousands of systems instantly.

Re-encryption of the DCE security database with a newly generated key
This action can even be initiated without shutting down security entirely.

Imposition of stricter auditing requirements across all DCE applications
Obviously this is an after-the-fact improvement, but it may be very valuable nevertheless

Re-generation of the login keys used by authenticated application servers
DCE application servers store their login keys (passwords) in files on disk. After a security breach, there is the potential that these files could have been read by an intruder & the keys could have been learned. It therefore makes sense to re-generate these keys after a suspected breach.

Shorten the life span of existing and future DCE tickets
After a successful attack, an intruder will have obtained one or more tickets from the DCE security server. In order to prevent the attacker from doing further damage, the lifespan of his or her tickets may be set to zero.

Raise an appropriate alarm to inform security administration staff
Regardless of the degree of automation of this proposed tool, it is important to inform security operations of a breach so that other measures may be taken against the attacker.

Temporarily disable each DCE administrator account
It may make sense to disable all administrative accounts so that no privileged operations can be performed until maintenance is completed by a person with physical access to

the console of the security server host.

Shut Down All Applications

In some environments denial of service is preferred to unauthorized data access. In these environments, shutdown of applications is possible via the Distributed Agent running on each system.

3.7 Overall System Architecture

The end result of the research effort was to start with an architectural concept, develop and modify it until a valid architecture was prototyped from which a commercial grade product could be built.

The key elements of the original proposal were:

1) an Administrative Client,
2) a set of Centralized Audit Monitors,
3) a (much larger) set of Distributed Agents, and
4) a (one-per-network-segment) set of Alert Agents.

These components are documented in the pages that follow.

The previous pages in this report document a phase II prototype that differs from the original Phase I proposal in two main ways:

1) The use of a multicast RPC using "maybe" semantics was required as a replacement for broadcast RPC, and
2) The original Phase I application architecture was based on the naive assumption that all intrusions would manifest themselves in auditable events at a centralized security server. The modified architecture we have explained in this report calls for a lightweight client Intrusion Detection component at each client system.

While these differences are significant, they did not drastically alter the overall architecture. This reinforced our confidence in our initial design and assumptions. Integration with pre-existing IDS technologies is expected to alter the design to some degree, but the overall architecture appears to be sound based on our research and integration efforts thus far. The pages that follow document the originally proposed architecture.

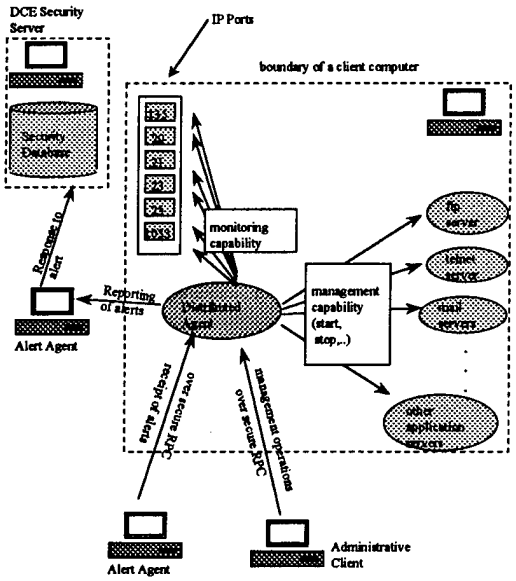

Figure 5: The Distributed Agent

FIGURE 5 SYSTEM ARCHITECTURE - OPERATION OF THE DISTRIBUTED AGENT

The figure above shows the operation of a Distributed Agent. The primary role of the Distributed Agent is to implement any local security policy decisions based on information provided by an Administrative Client or an alert from an alert agent. A secondary role is to raise an alert based on local monitoring of IP ports. The Distributed Agent is a privileged process and is able to stop and start processes that may be at risk in the event of an attack.

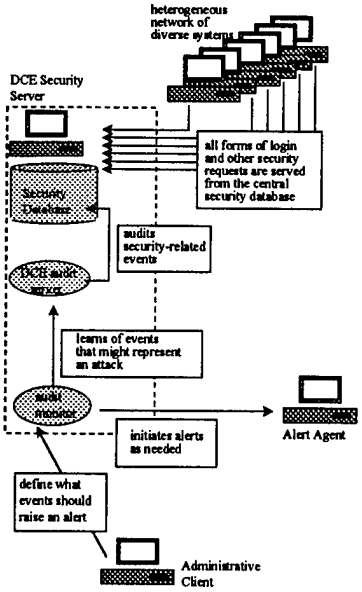

Figure 6: The Audit Monitor

FIGURE 6 SYSTEM ARCHITECTURE - THE AUDIT MONITOR

The above figure shows the operation of an Audit Monitor. The DCE Audit Service is able to monitor security-related events across very large heterogeneous networks. The instance of the DCE audit server attached to the DCE security server host is especially significant. This DCE security server will receive authentication requests, ticket requests and requests to change security policies etc from DCE and non-DCE applications across the entire DCE cell. The DCE audit server attached to the security server can be configured to log all security related events in a cell (all the way down to every POSIX getuid() call in the entire distributed cell). The proposed Audit Monitor would therefore be able to identify events of concern via a secure interface with the DCE Audit Service and raise alarms as required to the Alert Agent in the local network segment. These alarms could take the form of secure (encrypted) multicast DCE Remote Procedure Calls (RPCs).

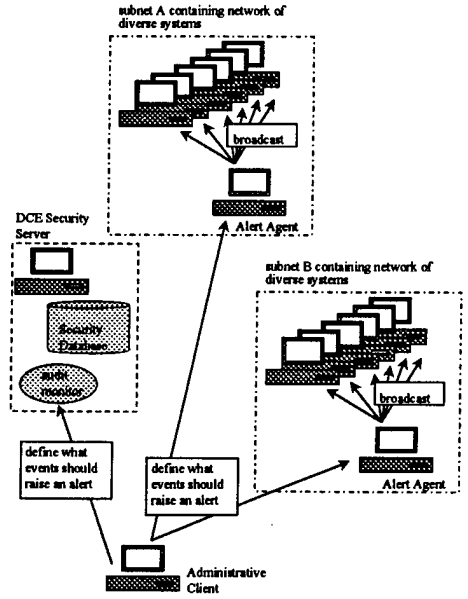

Figure 7: The Administrative Client

FIGURE 7 SYSTEM ARCHITECTURE - THE ADMINISTRATIVE CLIENT

The figure above shows the operation of an Administrative Client. The administrative client has the facility to define what events should raise an alarm. The Administrative Client defines the events that should raise an alarm and the actions expected as a result of each alarm. This information needs to be disseminated to every Distributed Agent (of which there could be many thousands) and to each Audit Monitor (there may be two for a replicated DCE security database). It is impractical for the Administrative Client to access each Distributed Agent directly, therefore a two-tiered architecture is employed. An Alert Agent resides in each network segment and can be located via the DCE directory service (CDS). The Audit Monitors can be located in a similar manner. The Administrative Client locates each Alert Agent and Audit Monitor & sends policy updates via secure (encrypted) DCE RPC (non-broadcast). The Alert Agents located in each network segment then issue a secure (encrypted) broadcast RPC to each of the clients (Distributed Agents) in their network segment.

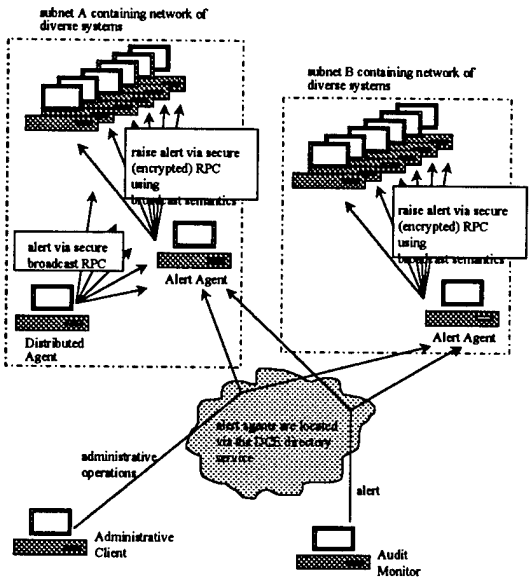

Figure 8: The Alert Agent

FIGURE 8 SYSTEM ARCHITECTURE - THE ALERT AGENT

The figure above shows the operation of an Alert Agent. At process startup, each Alert Agent registers its location in the DCE directory service (CDS). Each Alert Agent can "hear" alarms from Distributed Agents or Audit Monitors in the local network segment (since these alarms would be based on secure broadcast RPC). Systems in a network segment will not respond to the alert initiated by Distributed Agent as shown in Figure 8 middle left, but instead will wait to hear an alert from the Alert Agent. The advantage here is that the Alert Agent will run on an operating system with a greater level of security than most desktop systems. It is therefore possible for the Alert Agent to perform a mutual authentication check to the Distributed Agent raising the alarm in order to verify the true identity of the system raising the alarm and avoiding nuisance alerts from external sources.

In order to securely implement the suggested architecture, each system in the infrastructure must be configured into a DCE cell. At first it might therefore appear that only DCE applications could benefit from this intrusion detection system. This is by no means the case, in fact, as long as each system is configured with "Integrated DCE Login", every security interaction that previously was channeled to the local operating system is now passed to the DCE security server. In this fashion, systems (including non-DCE applications) can be monitored and served by a DCE-based intrusion detection and response tool with a low overhead.

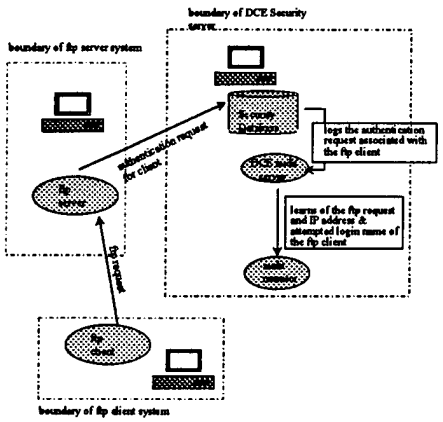

Figure 9 Using Integrated DCE Login to Enable Audit of Non-DCE Applications

FIGURE 9 SYSTEM ARCHITECTURE - DCE INTEGRATED LOGIN

The figure above illustrates the use of Integrated DCE Login to allow non-DCE applications to make use of DCE audit. DCE integrated login replaces the operating system library that holds POSIX security calls such as getuid() and routes those calls to a centralized security server. A DCE audit server at this location will therefore be able to log events from non-DCE processes throughout the DCE cell. The use of secure (encrypted) Remote Procedure Calls (RPCs) throughout the design facilitates secure communications. The centralized nature of the resultant security system implies that, when an attack is identified, modifications to security policies are required ONLY at the DCE security server and these modifications are seen instantly by clients (hence speeding up the response to an attack).

This page intentionally left blank

4 Results and Discussion

Over the last seven months, much pure research has been conducted. Much more, in fact, than was promised in our Phase I application. We have conducted extensive literary searches, reading and studying papers that cover the last two decades. We have identified the position in the market place where our commercial grade Intrusion Detection System (IDS) would fit. We have identified potential customers of such a system.

We have made contacts with key players in the field who have been engaged in this work for many years. We have attended conferences to further develop these contacts. Through these contacts, we have obtained source code for three existing systems. We studied these systems and have had success in integrating DCE Audit Services events with two of these systems.

We have developed a viable secure multicast RPC method that is scalable, efficient and can be made to be fault tolerant. We have developed an interface to the DCE Audit Service that uses the standard Audit Service APIs in an effective way for use by a real-time IDS. We have discovered that there are serious bugs in some vendor DCE Security Service implementations and are working with the vendor community to resolve this deficiencies.

From a response perspective, we have proven we can invalidate access and reduce access on a global and discrete level. We have shown that DCE Single Sign-on will, in fact, generate Security Service audit events that can be seen by an IDS.

We know we have the enthusiastic endorsement of outside sponsors who wish us to pursue full scale development of a commercial grade IDS and have offered their financial support for us to do so. We know that we will be able develop an audit data collection mechanism that will be efficient, secure and fault tolerant across many systems. This will enable a true "single system image" view desperately needed by a commercial grade IDS.

We can build a hierarchical notification alert mechanism that will be scalable, secure, and fault tolerant. This notification mechanism will aid in the overall effectiveness and management of a highly distributed IDS.

We believe that the DCE based Single Sign-on products available today are not all inclusive and will need to be augmented. We know that a "lightweight client monitor" will be required to search for intrusions invisible to our centralized IDS. We also know that there are DCE based products currently available (separate and distinct from Single Sign-on products) that could extend the DCE based capabilities to many different levels of detection and response.

We believe that the DCE Registry can hold information relevant to the operation of an IDS and would present a natural location to hold "resource sensitivity" information. Using Extended (user) attributes in the Registry, we believe we could set policies and effect responses that an IDS would take.

We believe another commercial grade IDS product is needed and would be well received in the market place where DCE is being used. Our key differentiators are:

1) Provision of an enterprise-wide solution detecting anomalies on most modern operating systems.
2) Provision of an immediate response facility affecting each of those operating systems listed above. The response shall be implemented across one thousand systems within one second.
3) Focusing much effort in the area of elimination of "false positive" indications of intrusion (through use of multiple independently-developed Intrusion Detection engines and refinement of those engines, rather than invention another ID engine).

4) Utilizing a secure industry-standard security and communications infrastructure (DCE RPC), therefore saving time that otherwise would be needed for creation of a security and communications infrastructure.
5) Implementing the forthcoming Common Intrusion Detection Framework (CIDF) to facilitate re-useability of components and interoperability with other Intrusion Detection products.
6) Leveraging the scalable DCE infrastructure (which supports tens of thousands of clients in a single cell) by interfacing with the DCE Audit Service. This audit service only audits security-related events and can therefore support a cell encompassing very large numbers of clients.

5 Conclusions

The effort of the Phase I SBIR has been more successful than anticipated. The proof of concept work undertaken has validated that our novel approach to creation of an Intrusion Detection solution is feasible within the timeframe of Phase II. We now know that we can use some proportion of existing technologies combined with new and original work to develop a viable commercial grade IDS product with the key differentiators indicated in section 4 above. We seek the opportunity to move into Phase II and engage in full scale development activities.

This page intentionally left blank

6 References

1. Aurobindo Sundaram. *An Introduction to Intrusion Detection* http://www.acm.org/crossroads/xrds2-4/intrus.html
2. Steven E. Smaha, *Haystack: An Intrusion Detection System*, Proceedings of the 4th Aerospace Computer Security Applications Conference, Dec. 1988, pp37-44.
3. Teresa F. Lunt *Automated Audit Trail Analysis and Intrusion Detection: A Survey.* 11th National Computer Security Conference, October 1988, Outstanding Paper Award.
4. Teresa F. Lunt.. *IDES: An intelligent system for detecting intruders.* In Proceedings of the Symposium: Computer Security, Treat and Countermeasures, Rome, Italy, November 1990.
5. Teresa F. Lunt and Debre Anderson. *Software Requirements Specification: Next-Generation Intrusions Detection Expert System (NIDES)* March 1993, Computer Sciences Laboratory, SRI International, Meno Park, CA 94025 produced under U.S. Government contract number N00039-92-C-0015 funded by the U.S. Navy SPAWAR.
6. Sandeep Kumar, *Intrusion Detection In Our Time*, paper at the 1994 National Computer Security Conference, see http://www.cs.purdue.edu/coast/coast-tools.html.
7. X/Open Distributed Audit Service (XDAS) Specification as described and maintained by the Open Group at: http://www.rdg.opengroup.org/public/tech/security/das/index.htm
8. X/Open Single Sign-On (SSO) Specification as described and maintained by the Open Group at: http://www.rdg.opengroup.org/public/tech/security/sso/index.htm
9. IBM Global Sign-On White Paper at: http://www.networking.ibm.com/gso/ssowpaper.html
10. PC-DCE from Gradient Technologies at: http://www.gradient.com/Products/Pc-dce/pcdce_front.htm
11. IBM DCE products at: http://www.networking.ibm.com/dce/dceprod.html
12. IBM Global Sign-On products at: http://www.networking.ibm.com/sso/ssohome.html
13. HP DCE products at: http://www.hp.com/gsy/dce/products.html
14. Transarc DCE products at: http://www.transarc.com/afs/transarc.com/public/www/Public/ProdServ/Product/index.html
15. SnareWorks from IntelliSoft at: http://www.isoft.com/snare/
16. Hacker Web Page at: http://www.sonic.net/~z/a-h.shtml
17. Hacker Web Page at: http://www.czimmermann.com
18. Hacker Web Page at: http://www.agate.net/~krees/philes.html
19. Hacker Web Page at: http://www.njh.com/latest/9706
20. Hacker Web Page at: http://oliver.efri.hr/~crv/security/bugs/list.html
21. Hacker Web Page at: http://www.parodius.com/~splice/rush/exploits.2/sploits.html
22. Hacker Web Page at: http://www.cyberstreet.com/users/rellik/diebarney
23. Hacker Web Page at: http://www.hardlink.com/~blackout/hacking
24. Hacker Web Page at: http://seclab.cs.ucdavis.edu/projects/vulnerabilities

This page intentionally left blank

Appendix A - Components of DCE

Since DCE provides security, directory, and RPC technologies, it aids in the actual development of a distributed intrusion detection system. Further is offers a unique vantage point from which to detect intrusions.

DCE Based Remote Procedure Calls (RPCs)

DCE's RPC mechanism is a powerful, secure, and interoperable technology which enables applications such as an intrusion detection system to be written and execute across a wide variety of platforms. While there are many RPC technology choices in the market place today, DCE has a unique heritage of being developed through a consortium of vendors in a fully "open" way with primary sponsorship/development from The Open Group. A little known fact is that Microsoft used DCE RPC as the basis for their Common Object Model (COM) and Distributed Common Object Model (DCOM) work. DCE RPC performance compares favorably with any other RPC technology, and offers far more richness in feature set thorough the addition of various security options as mentioned previously.

DCE Security Services

Second only to the RPC mechanism, the DCE Security Service is the most attractive feature of DCE. While initially based on MIT's Kerberos Version 5 and HP/Apollo's Centralized Password Registry, significant enhancements have been made over the last eight years. With DCE 1.1 the Registry Services was enhanced to support Extended Registry Attributes (ERAs) which make the Registry an extendable centralized resource. The Authentication Service performs the function necessary to validate the identity of resources within the cell. This service has been enhanced such that at no time is any authentication information sent across the network that is not fully encrypted with strong encryption keys. The Privilege Service has been enhanced to ensure that no one (not even the client) can compromise the integrity of the certificates of authenticity (EPAC). This makes the DCE Security Service an extremely robust and reliable technology. As such, it is often a primary factor in choosing DCE technology where high security is needed.

The DCE Security Service is implemented with technology choices in mind. Most commonly this is implemented as a centralized and replicated database. Some vendors (e.g. IBM) have improved the database structure to be stored in a common database type (e.g. DB2). The use of the DCE Security Services is tightly integrated with the DCE RPC mechanism.

Since the Security Service is centralized, we need only to audit the centralized server for security related events in order to see the "whole cell". For an intrusion detection system, this greatly simplifies the challenge in seeing widespread attacks throughout an environment. It also means that responses made within the centralized Registry are seen throughout the entire environment. This simple fact, makes a DCE based intrusion detection system a compelling choice.

DCE Directory Services

DCE offers a standards based naming and directory service that enables distributed applications. Servers are able to "advertise" their services and clients able to locate the servers in a dynamic way. This allows servers to "come and go", as would be expected in a dynamic system, without clients loosing the capability to locate a server that can service their request. As was illustrated in the "Consolidating Audit Service Events Across Multiple Systems" section above, this service allows an application to be designed such that locality of reference and priority of servers can be taken into account for a rich fault-tolerant application architecture.

The DCE Directory Service is implemented with technology choices in mind. Most commonly used is the DCE Cell Directory Service (CDS) which is implemented as a centralized, partitioned and replicated database. It can also be integrated with a more global directory service such as Domain Name Service (DNS) or X.500 Directory Services. The use of the Directory Service is tightly integrated with the DCE RPC mechanism.

DCE Time Services

It is critical in a distributed environment that system clocks are properly synchronized. Without proper synchronization, valuable data may be corrupted or misleading information given to unsuspecting users. DTS prevents this from happening by synchronizing the clocks across the cell. To this end, DTS provides administrators with a variety of methods to achieve clock synchronization. Options include obtaining a time value from an external source and an algorithm based on sampling the clocks in the cell. Most important is that time always moves forward (never backward) which is achieved with modification of the operating system's interaction with the hardware clock.

DCE/DFS Distributed File Services

DFS is the distributed file system that is offered with DCE. It is a single integrated file system that can coexist with a system's native file system. Chief among its strengths is its tight integration with DCE. This allows DFS to take advantage of authentication provided by the DCE Security Service. In addition to this, DFS provides a more granular set of file permissions than those provided with the UNIX file system. System administrators can be much more precise when defining permissions on sensitive files or directories.

Also of great value is the excellent performance offered by DFS. Caching is used on the client machine to reduce the number of calls made to the server. This significantly speeds file operations as the client only contacts the server when absolutely necessary.

DFS offers an excellent file replication facility which allows a certain degree of fault tolerance from a network failure. Should a particular mount point fail, DFS can automatically re-establish contact with a replica. This transfer remains invisible to the user and results in no lost productivity.

The systems administrator's duties are simplified by an excellent data movement facility which enables user data to be moved from file server to file server even which the file is open and actively being written to.

Appendix B - Source Code to Programs

AuditListen

Makefile	Used to build the auditListen program
aud.c	The primary code to the auditListen program
aud.h	The C header to the auditListen program
aud_misc.c	Utility C code used by the auditListen program

Secure Broadcast RPC Alternative

Makefile	Used to build the client and agent pograms
myapp.c	General purpose code used by both client and agent programs
myapp.h	The C header file
alarm_a.c	The primary code to the agent program
alarm_a.h	The C header file
alarm_a.idl	The DCE Interface Definition Language (IDL) file
alarm_c.c	The primary code to the client program
alarm_c.h	The C header file
alarm_c.idl	The DCE Interface Definition Language (IDL) file

Appendix B - auditListen

Makefile

```
CC = gcc

DCE_INC = -I/opt/dcelocal/share -I/opt/dcelocal/share/include -
I/opt/dcelocal/share/include/dce
INC_DIR = -I. $(DCE_INC)
DEBUG = -g
LIBS = $(LIB_DIR) -ldce -laudit -lthread -lsocket -lm

OBJS = aud_misc.o aud_idiot.o aud.o

auditListen: $(OBJS)
   $(CC) -o auditListen $(OBJS) $(LIBS)

clean:
   rm -f *.o auditListen
```

aud.c

```
/*
 * MODULE: aud.c
 * DESCRIPTION:
 *   This module contains the implementation of the auditListen test.
 *
*/
#include "aud.h"

void main (int argc, char **argv)
{
    dce_aud_rec_t               ard;
    unsigned_char_t             *buff;
    int                         fd, i, listc, code;
    boolean32                   done;
    error_status_t              st, st2;
    pthread_t                   kill_tid, notfy_tid;
    unsigned_char_t             trail_name[STRING_LEN];
    FILE                        *data;
    kill_handler_t              kill_args;

    long                        c_trail_pos,c_index_pos;
    long                        p_trail_pos,p_index_pos;

    new_record      = FALSE;
    recycle         = FALSE;
    trail_size      = 0;
    c_trail_pos     = 0;
    p_trail_pos     = 0;
    c_index_pos     = 0;
    p_index_pos     = 0;

    /* Did the user not specify a trail file?
    */

    if (argc < 2) {
        fprintf(stderr,"ERROR: Audit trail file not specified.\n");
        exit(0);
    }
    strcpy(trail_name,argv[1]);

    /* Open data file to cut audit informtation to
```

```
    */

    if ((data=fopen("/tmp/audit_data.log","w")) == NULL) {
       fprintf(stderr,"ERROR: Cannot open the log file.\n");
    }

    /* Open audit trail file
    */

    dce_aud_open(aud_c_trl_open_read, trail_name, 0, 0, (dce_aud_trail_t
*)&trail, &st);
    CHECK_STATUS(st,"dce_aud_open() FAILED.\n",ABORT);
    cut_debug_info(data,"Opened audit trail file.");


    /* Set up kill_handler thread
    */

    code = sigemptyset(&mask);
    if (code != 0) {
       fprintf(stderr,"sigemptyset() FAILED.\n");
       fclose(data);
       exit(1);
    }
    code = sigaddset(&mask,SIGINT);
    if (code != 0) {
       fprintf(stderr,"sigaddset() FAILED.\n");
       fclose(data);
       exit(1);
    }
    code = sigaddset(&mask,SIGTERM);
    if (code != 0) {
       fprintf(stderr,"sigaddset() FAILED.\n");
       fclose(data);
       exit(1);
    }
    code = sigprocmask(SIG_BLOCK,&mask,NULL);
    if (code != 0) {
       fprintf(stderr,"sigprocmask() FAILED.\n");
       fclose(data);
       exit(1);
    }
    kill_args.trail = (dce_aud_trail_t)trail;
    kill_args.log = data;

    code = pthread_create(&kill_tid,pthread_attr_default,
            (pthread_startroutine_t)kill_handler,
            (pthread_addr_t)&kill_args);
    if (code != 0) {
       fprintf(stderr,"pthread_create() FAILED.\n");
       fclose(data);
       exit(1);
    }

    /* Notify of new records appended to trail file
    */

    code = pthread_create(&notfy_tid,pthread_attr_default,
            (pthread_startroutine_t)notify_of_new_records,
            (pthread_addr_t *)trail_name);
    if (code != 0) {
       fprintf(stderr,"pthread_create() FAILED.\n");
       fclose(data);
```

```
        exit(1);
    }

    /* Read records from the file
    */

    while (1) {
       if (recycle) {

          /* Close and reopen trail file because we detected some
operation
     ** that shrunk the size of the trail file such as:
     **     (1) aud trail file rollong over after hitting max size.
     **     (2) user executing dcecp -c aud rewind
     */

          dce_aud_close((dce_aud_trail_t)trail, &st);
          CHECK_STATUS(st,"dce_aud_close() FAILED.\n",ABORT);
     dce_aud_open(aud_c_trl_open_read, trail_name, 0, 0,
(dce_aud_trail_t *)&trail, &st);
          CHECK_STATUS(st,"dce_aud_open() FAILED.\n",ABORT);
          cut_debug_info(data,"Rewound the audit log.");

     recycle = FALSE;
     c_trail_pos = 0;
     c_index_pos = 0;
       }
       if (new_record) {
          done = FALSE;
     while (!done) {

         /* Get next record
         */

              dce_aud_next((dce_aud_trail_t)trail, NULL, 0, &ard, &st);
         CHECK_STATUS(st,"dce_aud_next() FAILED.\n",ABORT);

         p_trail_pos = c_trail_pos;
         p_index_pos = c_index_pos;

         c_trail_pos = ftell(trail->trail_fp);
         c_index_pos = ftell(trail->md_index_fp);

             if (ard == NULL) {
          dce_aud_backup(trail,p_trail_pos,p_index_pos,&st);
          CHECK_STATUS(st,"dce_aud_backup() FAILED.\n",ABORT);

                    done = TRUE;
          new_record = FALSE;
                    break;
             }

         /* Cut record information to data file
         */

         if (USE_IDIOT) {
            convert_to_idiot(ard);
         }
         cut_record_info(ard,data);


         /* Release memory for record
         */
```

```
                dce_aud_discard(ard, &st);
              CHECK_STATUS(st,"dce_aud_discard() FAILED.\n",ABORT);
            }
          }
        }
      }
```

aud.h

```
#ifndef _AUD_H
#define _AUD_H

/*
 * MODULE: aud.h
 *
 * DESCRIPTION:
 *   This module contains prototypes and definitions used in the
 *   testcase auditProbe, used to show the records in the audit trail.
 *
*/

/* INCLUDE FILES  */

#include <dce/dce.h>
#include <dce/dce_msg.h>
#include <dce/audit.h>
#include <dce/audit_control.h>
#include <dce/dce_error.h>
#include <dce/dceaudmsg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <pthread.h>
#include <signal.h>
#include <sys/stat.h>
#include <sys/types.h>

/* DECLARATIONS */

#define ABORT       1
#define RESUME      0
#define STRING_LEN         2000
#define USE_IDIOT          1

/* MACRO DEFINITIONS */

#define CHECK_STATUS(input_status, comment, action)                \
{                                                                  \
   if(input_status != aud_s_ok) {                                  \
      dce_error_inq_text(input_status, error_string, &error_stat);
   \
      fprintf(stderr, "%s %s\n", comment, error_string);           \
      if(action == ABORT) {                                        \
         exit(1);                                                  \
      }                                                            \
   }                                                               \
}

/* VARIABLE DECLARATIONS */

static int           error_stat;
static unsigned char error_string[dce_c_error_string_len];
```

```
sigset_t              mask;
off_t                 trail_size;
boolean32             new_record;
boolean32             recycle;

/* TYPEDEF DECLARATIONS */

typedef struct aud_trail_handle {
   int                    location;
   rpc_binding_handle_t   auditd_binding;
   char                   *trail_file;
   int                    trail_fd;
   FILE                   *trail_fp;
   char                   *index_file;
   int                    index_fd;
   FILE                   *index_fp;
   int                    md_index_fd;
   FILE                   *md_index_fp;
   unsigned32             index_cursor;
   unsigned32             trail_cursor;
   unsigned32             flags;
   boolean32              file_size_limit;
   unsigned32             file_size_limit_value;
   pthread_mutex_t        mutex;
} aud_trail_t;

typedef struct kill_handler_type {
   dce_aud_trail_t        trail;
   FILE                   *log;
} kill_handler_t;

/* PROTOTYPES */

void kill_handler (kill_handler_t *args);
void notify_of_new_records (unsigned_char_t *trail_name);

#endif
```

aud_misc.c

```
/*
 * MODULE: aud_misc.c
 *
 * DESCRIPTION:
 *   This module contains misc fuctions used by the
 *   auditListen test..
 *
*/


void cut_record_info(dce_aud_rec_t ard, FILE *log)
{
   dce_aud_hdr_t  *header;
   error_status_t st;
   struct tm      tmevnt;

   /* Get header from record
   */

   dce_aud_get_header(ard,&header,&st);
   CHECK_STATUS(st,"dce_aud_get_header() FAILED.\n",ABORT);

   /* Get time information
   */
```

```
    utc_anytime(&tmevnt,0,0,0,0,&(header->time));

    /* Cut record information
    */

    fprintf(log,"-> %d-%02d-%02d-%02d:%02d:%02d <-  event: %d\n",
        tmevnt.tm_year+1900,
        tmevnt.tm_mon+1,
        tmevnt.tm_mday,
        tmevnt.tm_hour,
        tmevnt.tm_min,
        tmevnt.tm_sec,
        header->event);
    fflush(log);
}

void cut_debug_info(FILE *log, char *string)
{
    utc_t     time;
    struct tm tmevnt;

    /* Get time information
    */

    utc_gettime(&time);
    utc_anytime(&tmevnt,0,0,0,0,&time);

    /* Cut record information
    */

    fprintf(log,"-> %d-%02d-%02d-%02d:%02d:%02d <-  %s\n",
        tmevnt.tm_year+1900,
        tmevnt.tm_mon+1,
        tmevnt.tm_mday,
        tmevnt.tm_hour,
        tmevnt.tm_min,
        tmevnt.tm_sec,
        string);
    fflush(log);
}

void kill_handler (kill_handler_t *args)
{
    int            sig;
    error_status_t st;

    sig = sigwait(&mask);
    if (sig == -1) {
       fprintf(stderr,"sigwait() FAILED.\n");
       exit(1);
    }
    fprintf(stderr,"\n*** Got Signal %d ***\n", sig);

    /* Close the aud trail file
    */

    fprintf(stderr,"Closing audit trail file.\n");
    dce_aud_close(args->trail, &st);
    CHECK_STATUS(st,"dce_aud_close() FAILED.\n",ABORT);
    cut_debug_info(args->log,"Closed audit trail file.");

    /* Close the log file
```

```
    */

    fprintf(stderr,"Closing the log file /tmp/audit_data.log\n");
    fclose (args->log);
    exit(1);
}

void notify_of_new_records (unsigned_char_t *trail_name)
{
    int         code;
    struct stat buf;

    while (1) {
        code = stat(trail_name,&buf);
        if (code != 0) {
            fprintf(stderr,"stat() FAILED.\n");
            exit(1);
        }
        if (buf.st_size > trail_size) {
            new_record = TRUE;
            trail_size = buf.st_size;
        } else if (buf.st_size < trail_size) {
            recycle = TRUE;
            trail_size  = 0;
        }
    }
}

void dce_aud_backup (aud_trail_t trail,
            long t_position,
            long i_position,
            error_status_t *status)
{
    int code;

    trail.trail_cursor = t_position;
    code = fseek(trail.trail_fp,t_position,SEEK_SET);
    if (code != 0) {
        fprintf(stderr,"dce_aud_backup() fseek failed.\n");
        *status = 1;
        return;
    }
    code = fseek(trail.md_index_fp,i_position,SEEK_SET);
    if (code != 0) {
        fprintf(stderr,"dce_aud_backup() fseek failed.\n");
        *status = 1;
        return;
    }
    *status = aud_s_ok;
}
```

Appendix B - Secure Broadcast RPC Alternative

Makefile

```
# make file for alarm agent and alarm client
CC = g++

GCC = gcc

COPTS = -g

DEFINES = -D_SUN

GCC_INC =

DCE_INC = -I/opt/dcelocal/share -I/opt/dcelocal/share/include -
I/opt/dcelocal/share/include/dce

INC = -I. ${DCE_INC} ${GCC_INC}

LDFLAGS = -ldce -lm -lthread -lsocket
CFLAGS = $(COPTS) $(DEFINES) $(INC)

IDLC = idl
IFLAGS = -cc_cmd $(CC) -cc_opt "-c $(COPTS) $(DEFINES)"

default:
   @echo "Please specify a target."

all: alarm
IF1= alarm_c
IF2= alarm_a

myapp.o: myapp.h
alarm: alarm_c alarm_a
alarm_c: alarm_c.o myapp.o $(IF2)_cstub.o $(IF1)_sstub.o
   $(CC) $(CFLAGS) -o alarm_c alarm_c.o myapp.o $(IF2)_cstub.o
$(IF1)_sstub.o $(LDFLAGS)
alarm_c.o: alarm_c.h alarm_a.h myapp.h
alarm_a: alarm_a.o myapp.o $(IF2)_sstub.o $(IF1)_cstub.o
   $(CC) $(CLFAGS) -o alarm_a alarm_a.o myapp.o $(IF2)_sstub.o
$(IF1)_cstub.o $(LDFLAGS)
alarm_a: alarm_a.h alarm_c.h myapp.h
   $(IF1)_cstub.o $(IF1)_sstub.o $(IF1).h: $(IF1).idl
   $(IDLC) $(IFLAGS) $(IF1).idl
   $(IF2)_cstub.o $(IF2)_sstub.o $(IF2).h: $(IF2).idl
   $(IDLC) $(IFLAGS) $(IF2).idl
```

myapp.c

```
extern "C" {
   #include <stdlib.h>
}
#include "myapp.h"
#define DEBUG 1

// definition of class DCE_server first
// DCE_server::DCE_server()
// select protocol sequence
// inquire server binding
// register with endpoint map
// export binding to name space
```

```
// setup management thread to refresh identity before expired and key
//    management
// listen to client calls
//
DCE_server::DCE_server(
    rpc_if_handle_t ih,
    unsigned_char_t * prin,
    uuid_t *mgr_type_uuid,
    rpc_mgr_epv_t mgr_epv,
    Myproto p,
    int authn_lgn,
    unsigned32 protection,
    unsigned32 authn,
    unsigned32 authz,
    unsigned_char_t * kt,
    int ep,
    int ns,
    int is_listen
)

:if_handle(ih), principal(prin), mgr_type_uuid_of_obj(mgr_type_uuid),
manager_epv(mgr_epv), myproto(p),
authn_login(authn_lgn), protection_level(protection),
authentication(authn), authorization(authz), keytab_file(kt),
endpoint(ep), name_space(ns), server_entry(0), group_entry(0),
profile_entry(0), refresh_id_thread(0),
key_mgmt_thread(0), sigcatch_thread(0)
{
    use_protocol();
    if (DEBUG) cout << "use_protocol complete\n";

    inq_server_bindings();
    if (DEBUG) cout << "inq_server_binding complete\n";

    if(authn_login) sec_login();
    if (DEBUG) cout << "authn_login complete\n";

    register_interface();
    if (DEBUG) cout << "register_interface complete\n";

    if (ep) register_endpoint();
    if (DEBUG) cout << "register_endpoint complete\n";

    if (ns) export_to_namespace();
    if (DEBUG) cout << "export_to_namespace complete\n";

    if(authn_login) start_mgmt_thread();

    start_signal_thread();

    if (is_listen) listen();
}

// DCE_server::listen()
// FUNCTIONALITY:
// start to listen incoming calls
//
void DCE_server::listen() {
    cout << "server " << principal << " is listening" << endl;
    rpc_server_listen(rpc_c_listen_max_calls_default, &status);
    dce_error(status, "server stop listening...");
}
```

```
// DCE_server::use_protocol()
// FUNCTIONALITY:
// select udp, tcp or both
//
void DCE_server::use_protocol() {
    switch (myproto) {
        case all: {rpc_server_use_all_protseqs(
            rpc_c_protseq_max_reqs_default, &status);
            dce_error(status, "rpc_server_use_all_protseqs failed");
            break;
        }
        case udp: {rpc_server_use_protseq(
            (unsigned_char_p_t) "ncadg_ip_udp",
            rpc_c_protseq_max_reqs_default, &status);
            dce_error(status, "rpc_server_use_proteseq udp failed");
            break;
        }
        case tcp: {rpc_server_use_protseq(
            (unsigned_char_p_t) "ncacn_ip_tcp",
            rpc_c_protseq_max_reqs_default, &status);
            dce_error(status, "rpc_server_use_protseq tcp failed");
            break;
        }
    }
}


// DCE_server::inq_server_bindings()
// FUNCTIONALITY:
// wrapper for rpc_inq_binding to get server binding vector
//
void DCE_server::inq_server_bindings() {
    rpc_server_inq_bindings(&bv, &status);
    dce_error(status, "Unable to get server bindings");
}



// DCE_server::register_interface()
// FUNCTIONALITY:
// wrapper for rpc_server_register_if to register interface
// specification with RPC runtime
//
void DCE_server::register_interface() {

rpc_server_register_if(

if_handle,

mgr_type_uuid_of_obj,

manager_epv,

&status);

dce_error(status, "Unable to register server interface with RPC
runtime");
}

// DCE_server::register_endpoint()
// FUNCTIONALITY:
```

```
// wrapper for rpc_ep_register() to register with endpoint mapping
interface
//
void DCE_server::register_endpoint() {
    rpc_ep_register(
    if_handle,
    bv,
    NULL, // obj uuid
    (unsigned_char_t *) principal,
    &status);
    dce_error(status, "Unable to register with endpoint map");
}

// DCE_server::export_to_namespace()
// FUNCTIONALITIES:
// export the server binding to name space
// PREREQUISITE:
// ENVIRONMENT VARIABLE
//     SERVER_ENTRY: server entry in CDS
//     GROUP_ENTRY: group of server entry in CDS
//     PROFILE_ENTRY: profile of group entry in CDS
//
void DCE_server::export_to_namespace() {
    const int SIZE = 256;

    if (DEBUG) cout << "start to allocate memory for server_entry\n";

    server_entry = new unsigned_char_t [SIZE];
    group_entry = new unsigned_char_t [SIZE];
    profile_entry = new unsigned_char_t [SIZE];

    char * server_temp = NULL;
    char * group_temp = NULL;
    char * profile_temp = NULL;

    if (server_temp = getenv("SERVER_ENTRY"))
        strcpy((char *)server_entry, server_temp);

    if (group_temp = getenv("GROUP_ENTRY")) {
        strcpy((char *)group_entry, group_temp);
        if (DEBUG) cout << "temp is not NULL\n";
    }

    if (profile_temp = getenv("PROFILE_ENTRY")) {
        strcpy((char *)profile_entry, profile_temp);
        if (DEBUG) cout << "temp is not NULL\n";
        }

    if (DEBUG) {
        cout << "start to export to namespace\n";
        cout << "server_entry: " << server_entry << endl;
        cout << "group_entry: " << group_entry << endl;
        cout << "profile_entry: " << profile_entry << endl;
    }



    rpc_ns_binding_export(
        rpc_c_ns_syntax_default,
        (unsigned_char_p_t) server_entry,
        if_handle,
        bv,
        NULL, // object UUID
```

```
        &status);
    dce_error(status, "Unable export to name space");
    if (DEBUG) cout << "rpc_ns_binding_export complete\n";

    if (group_temp) {
        rpc_ns_group_mbr_add(
            rpc_c_ns_syntax_default,
            (unsigned_char_p_t) group_entry,
            rpc_c_ns_syntax_default,
            (unsigned_char_p_t) server_entry,
            &status);
        dce_error(status, "Unable export to group entry");
    }
    if (DEBUG) cout << "rpc_ns_group_mbr_add complete\n";

    if (profile_temp) {
        rpc_if_inq_id(if_handle, &if_id, &status);
        dce_error(status, "Unable to get interface identifier");
        rpc_ns_profile_elt_add(
            rpc_c_ns_syntax_default,
            (unsigned_char_p_t) profile_entry,
            &if_id,
            rpc_c_ns_syntax_default,
            (unsigned_char_p_t) server_entry,
            0, // priority
            (unsigned_char_p_t) principal, // annotation actually
            &status);
        dce_error(status, "Unable export to profile entry");
     }
}

// DCE_server::sec_login()
// FUNCTIONALITY:
// setup server principal identity
// validate and certify the server principal's identity
// setup security login context
// server register authentication information with RPC runtime
//
void DCE_server::sec_login() {
    setup_identity();
    valid_and_cert_identity();

    sec_login_set_context(login_context, &status);
    dce_error(status, "Unable to setup server login context");

    rpc_server_register_auth_info(
        principal,
        rpc_c_authn_dce_secret,
        NULL, // use default keytab retrieval method
        keytab_file,
        &status);
   dce_error(status, "Unable to register server auth info with RPC
runtime");
}

// DCE_server::setup_identity()
// FUNCTIONALITY:
// wrapper for sec_login_setup_identity()
//
void DCE_server::setup_identity() {
    sec_login_setup_identity(
        principal,
        sec_login_no_flags,
```

```
        &login_context,
        &status);
    dce_error(status, "Unable to setup identity for server principal");
}

// DCE_server::valid_and_cert_identity()
// FUNCTIONALITY:
// get server key
// validate the server's identity
// free server key to make sure no one have enough time to get server
key
// certify the server's identity
//
void DCE_server::valid_and_cert_identity() {
    void * key_data;
    sec_login_auth_src_t authn_src;
    boolean32 reset_password;

    sec_key_mgmt_get_key(
        rpc_c_authn_dce_secret,
        (unsigned_char_p_t) keytab_file,
        (unsigned_char_p_t) principal,
        0, // latest key version
        &key_data,
        &status);
    dce_error(status, "Unable to locate server key");

    sec_login_validate_identity(
        login_context,
        (sec_passwd_rec_t *) key_data,
        &reset_password,
        &authn_src,
        &status);
    dce_error(status, "Unable to validate identity");

    sec_key_mgmt_free_key(key_data, &status);
    dce_error(status, "Unable to free server key");

    sec_login_certify_identity(
        login_context,
        &status);
    dce_error(status, "Unable to certify identity for server");
}

// DCE_server::~DCE_server()
// FUNCTIONALITY:
// remove binding information at CDS
// remove binding information at endpoint mapping service
// unregister server binding information from RPC runtime
// kill key management thread
// kill refresh ID thread
// kill signal catcher thread
// purge security login context
// free resouce acquired
//
DCE_server::~DCE_server() {

    rpc_if_id_t  if_id; rpc_if_inq_id(if_handle, &if_id, &status);
    dce_error(status, "Unable to get interface identifier");

    if(profile_entry) {
        rpc_ns_profile_elt_remove(
            rpc_c_ns_syntax_default,
```

```
            (unsigned_char_p_t) profile_entry,
            &if_id,
            rpc_c_ns_syntax_default,
            (unsigned_char_p_t) server_entry,
            &status);
        dce_error(status, "Unable to remove entry from profile", 0);
    }

    if (group_entry) {
        rpc_ns_group_mbr_remove(
            rpc_c_ns_syntax_default,
            (unsigned_char_p_t) group_entry,
            rpc_c_ns_syntax_default,
            (unsigned_char_p_t) server_entry,
            &status);
        dce_error(status, "Unable to remove entry from group", 0);
    }

    if (server_entry) {
        rpc_ns_binding_unexport(
            rpc_c_ns_syntax_default,
            (unsigned_char_p_t) server_entry,
            if_handle,
            NULL, // should be an object UUID
            &status);
        dce_error(status, "Unable to unexport server entry", 0);
    }

    if (endpoint) {
        rpc_ep_unregister(
            if_handle,
            bv,
            NULL, // should be an object UUID
            &status);
        dce_error(status, "Unable to remove entry point", 0);
    }

    rpc_server_unregister_if(
        if_handle,
        mgr_type_uuid_of_obj,
        &status);
    if (key_mgmt_thread)
        kill_thread(key_mgmt_thread, "key management");
    if (refresh_id_thread)
        kill_thread(refresh_id_thread, "refresh identity");
    if (sigcatch_thread)
        kill_thread(sigcatch_thread, "signal catcher");
    dce_error(status, "Unable to unregister interface with RPC runtime");

    sec_login_purge_context(&login_context, &status);
    dce_error(status, "Unable to purge server security login context",
0);

    rpc_binding_vector_free(&bv, &status);
    dce_error(status, "Unable to free binding vector", 0);

    if (server_entry)
        delete [] server_entry;
    if (group_entry)
        delete [] group_entry;
    if (profile_entry)
        delete [] profile_entry;
// if (mgr_type_uuid_of_obj)
```

```
//      delete mgr_type_uuid_of_obj;
}


// DCE_server::start_mgmt_thread
// FUNCTIONALITY:
// start refresh_identity thread to refresh identity periodically
// start key_mgmt thread to manage key
//
void DCE_server::start_mgmt_thread() {
    create_thread(
        refresh_identity,
        "refresh identity",
        &refresh_id_thread,
        (pthread_addr_t) this);

    create_thread(
        key_mgmt,
        "key management",
        &key_mgmt_thread,
        (pthread_addr_t) this);
}

// DCE_server::start_signal_thread()
// FUNCTIONALITY:
// setup signal handling for the server
//
void DCE_server::start_signal_thread() {
    create_thread(
        signal_catcher,
        "signal catcher",
        &sigcatch_thread,
        NULL);
}


// Helper function of DCE_server
// key_mgmt()
// FUNCTIONALITY:
// given a server, manager this server's key
//
pthread_addr_t key_mgmt(pthread_addr_t arg) {
    unsigned32 status;
    DCE_server * theServer = (DCE_server *) arg;
    if (DEBUG) cout << "The key_mgmt thread is starting to work...\n";
    sec_key_mgmt_manage_key(
        rpc_c_authn_dce_secret,
        theServer->keytab_file,
        theServer->principal,
        &status);
    dce_error(status, "Unable to manage server keys");
}

// refresh_identity
// FUNCTIONALITY:
// given a server, refresh the server's identity periodically
//
pthread_addr_t refresh_identity(pthread_addr_t arg) {
    timeval now;
    timespec sleep_time;
    unsigned32 expiration;
    unsigned32 status;
    DCE_server * theServer = (DCE_server *) arg;
```

```
    if (DEBUG) cout << "the refresh_identity thread is starting to
work...\n";
    while (true) {
        sec_login_get_expiration(
            theServer->login_context,
            (long int *) &expiration,
            &status);
        dce_error(status, "Unable to get context expiration");
        gettimeofday(&now, NULL);
        sleep_time.tv_sec = expiration - now.tv_sec - 600;
        sleep_time.tv_nsec = 0;
        if (sleep_time.tv_sec < 0) {
            cerr << "Login context expires too soon" << endl; exit(0);
            }
        pthread_delay_np(&sleep_time);
        sec_login_refresh_identity(theServer->login_context, &status);
        dce_error(status, "Unable to refresh identity");
        theServer->valid_and_cert_identity();
    }
}


// set up signal handling for the application depend on long-lived or not
//
pthread_addr_t signal_catcher(pthread_addr_t arg) {
    sigset_t mask;
    unsigned32 status;
    if (DEBUG) cout << "start to setup signal handler...\n";

    sigemptyset(&mask);
    sigaddset(&mask, SIGTERM);
    sigaddset(&mask, SIGINT);
    sigaddset(&mask, SIGHUP);

    sigwait(&mask);
    if (DEBUG)
        cerr << "signal catched and I am going to stop the server...\n";

    rpc_mgmt_stop_server_listening(NULL, &status);
    dce_error(status, "Unable to stop server listening");
}


// general helper function
// create_thread():
// FUNCTIONALITY:
// create thread
//
void create_thread(pthread_startroutine_t thread_routine,
        char *thread_name,
        pthread_t *thread_id,
        pthread_addr_t thread_arg) {
    unsigned32 status;
    status = pthread_create(thread_id, pthread_attr_default,
thread_routine, thread_arg);
    if (status != 0) {
        cerr << "Unable to create " << thread_name << " thread\n";
        exit(0);
    }
    pthread_yield(); /* Allow thread to run now */
}
```

```
// kill_thread
// FUNCTIONALITY:
// kill thread
//
void kill_thread(
       pthread_t thread_id,
       char * thread_name) {
   unsigned32 status;
   status = pthread_cancel(thread_id);
   if (status != 0) {
      cerr << "Unable to kill " << thread_name << " thread\n";
   }
}


// is_server_there
// FUNCTIONALITY:
// Is server alive ?
//
boolean32 is_server_there(
       rpc_binding_handle_t h,
       rpc_if_handle_t ifspec) {
   boolean32 listening;
   unsigned32 status;

   listening = rpc_mgmt_is_server_listening(h, &status);
   if (status == rpc_s_binding_incomplete) {
      rpc_ep_resolve_binding(h, ifspec, &status);
      if ((status == ept_s_not_registered) ||
            (status == rpc_s_comm_failure) ||
            (status == rpc_s_connect_rejected))
         return(false);
      dce_error(status, "Unable to resolve binding");
      listening = rpc_mgmt_is_server_listening(h, &status);
   }

   if ((status == rpc_s_comm_failure) ||
         (status == rpc_s_connect_rejected))
      return(false);
   dce_error(status,"Unable to check if server is listening");
   return (listening);
}


// dce_errro()
// FUNCTIONALITY:
// Is DCE server listening for incoming calls
//
void dce_error(unsigned32 status, char *errmsg, boolean32 abort) {
   unsigned char errstr[dce_c_error_string_len];
   int error_status;

   if (status != error_status_ok) {
      dce_error_inq_text(status, errstr, &error_status);
      if (error_status == error_status_ok) {
         cerr << errstr << endl;
      } else {
         cerr << "Could not get error text for status " << status <<
endl;
      }
      if (abort) exit(0);
   }
}
```

```
// DCE_client difinition
// DCE_client::DCE_client()
// FUNCTIONNALITY:
// get binding handle
// do DCE login using client principal identified
// setup authentication information
//
DCE_client::DCE_client(
       rpc_if_handle_t ih,
       unsigned_char_p_t srv_prin,
       unsigned_char_p_t path,
       unsigned32 pt,
       unsigned32 authn,
       unsigned32 authz,
       unsigned_char_t * clnt_prin,
       unsigned_char_t * clnt_kt,
       Src_binding sb,
       int exp)
: if_handle(ih), server_principal(srv_prin), binding_path(path),
protection_level(pt), authentication(authn),
authorization(authz), client_principal(clnt_prin), keytab_file(clnt_kt),
src_binding(sb), expiration(exp)
{
   get_binding_handle();
   sec_login();
   setup_auth_info();
}


// DCE_client::get_binding_handle()
// FUNCTIONALITY:
// check source of the binding
// translate from string binding to normal binding when the source is
// string binding get binding from name space if the search entry is
// provided
//
void DCE_client::get_binding_handle() {
   rpc_ns_handle_t import_context;
   if (src_binding == str) {
      rpc_binding_from_string_binding(binding_path,
          &binding_handle,
          &status);
      dce_error(status, "Unable to get binding handle from string
binding");
   } else if (src_binding == ns) {
      rpc_ns_binding_import_begin(
          rpc_c_ns_syntax_default,
          binding_path,
          if_handle,
          NULL, // for object UUID
          &import_context,
          &status);
      dce_error(status, "Unable to create an import context in CDS db");
      rpc_ns_binding_import_next(import_context,xi
          &binding_handle,
          &status);
      if (status == rpc_s_no_more_bindings) {
          cerr << "No server bindings available\n";
          exit(0);
          }
```

```
        dce_error(status, "Unable to import next binding from server", 0);
        rpc_ns_binding_import_done(&import_context, &status);
        dce_error(status, "Unable to complete binding import from
server");
    }
}


// DCE_client::get_binding()
// FUNCTIONALITY:
// return server biding handle for next remote call
//
rpc_binding_handle_t & DCE_client::get_binding() const {
    return binding_handle;
}


// DCE_client::sec_login()
// FUNCTIONALITY:
// wrapper for setup_identity
//
void DCE_client::sec_login() {
    setup_identity();
}


// DCE_client::setup_identity()
// FUNCTIONALITY:
// setup client principal
// validate and certify client principal
// Or get client principal from current login context if client
//     principal is not specified
//
void DCE_client::setup_identity() {
    if (client_principal) {
        sec_login_setup_identity(
            client_principal,
            sec_login_no_flags,
            &login_context,
            &status);
        dce_error(status, "Unable to setup identity for client
principal");
        valid_and_cert_identity();
        dce_error(status, "Unable to validate and certify client
principal");
    } else {
        sec_login_get_current_context(&login_context, &status);
        dce_error(status, "Unable to get login context from the inherited
principal");
    }
}


// DCE_client::valid_and_cert_identity()
// FUNCTIONALITY:
// get client principal's key
// validate identity
// free the key
// certify the client's identity
//
void DCE_client::valid_and_cert_identity() {
    void * key_data;
    sec_login_auth_src_t authn_src;
```

```
    boolean32 reset_password;

    sec_key_mgmt_get_key(
        rpc_c_authn_dce_secret,
        (unsigned_char_p_t) keytab_file,
        (unsigned_char_p_t) client_principal,
        0, // latest key version
        &key_data,
        &status);
    dce_error(status, "Unable to locate server key");

    sec_login_validate_identity(
        login_context,
        (sec_passwd_rec_t *) key_data,
        &reset_password,
        &authn_src,
        &status);
    dce_error(status, "Unable to validate identity");

    sec_key_mgmt_free_key(key_data, &status);
    dce_error(status, "Unable to free server key");

    sec_login_certify_identity(
        login_context,
        &status);
    dce_error(status, "Unable to certify identity for server");
}


// DCE_client::setup_auth_info()
// FUNCTIONALITY:
// annotate the security on the binding in order to authenticate with
the server
//
void DCE_client::setup_auth_info() {
rpc_binding_set_auth_info(
binding_handle,
(unsigned_char_p_t) server_principal,
protection_level,
authentication,
(rpc_auth_identity_handle_t) login_context,
authorization,
&status);
dce_error(status, "Unable to setup client auth info");
}


// DCE_client::~DCE_client()
// FUNCTIONALITY:
// purge login context and free acquired resources
//
DCE_client::~DCE_client() {
    sec_login_purge_context(&login_context, &status);
    dce_error(status, "Unable to purge client login context", 0);

    free(client_principal);
    free(server_principal);
    free(binding_path);
}
```

myapp.h

```
#ifndef _MYAPP_H_
```

```
#define _MYAPP_H_

// MODULE: myapp.h
//
// DESCRIPTION:
//    This module contains declaration of class DCE_server, DCE_client and
//    some helper functions of these two classes like dce_error,
is_server_there,
//    create_thread, kill_thread, refresh_identity, key_mgmt,
signal_catcher
//

// INCLUDE FILES
#include <iostream.h>
extern "C" {
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <dce/nbase.h>
#include <dce/dce_error.h>
#include <pthread.h>
#include <dce/sec_login.h>
#include <dce/keymgmt.h>
#include <dce/rpc.h>
#include <pthread.h>
}

// HELPER FUNCTION DECLARATIONS
/* dce_error:
    status     - DCE routine status
    errmsg     - descriptive information about the DCE routine that
returned the status
    abort      - depend on critical of the error, exit system or continue.
default is exit system
*/
void dce_error(unsigned32 status, char *errmsg, boolean32 abort=1);

/* is_server_there:
    h   - binding handle
    ifspec - interface specification

    PURPOSE: Given a binding handle and interface specification, test if
the server is listening
*/
boolean32 is_server_there(rpc_binding_handle_t h, rpc_if_handle_t
ifspec);

/* create_thread():
    thread_routine   - the main body of the routine that the thread will
execute
    thread_name      - assign a name to this thread
    thread_id     - the ID return by the system to identity the thread
    thread_arg       - extra information that can pass the thread when
executed

    PURPOSE: Helper function to create a create, given a function to
execute
*/
void create_thread(
    pthread_startroutine_t thread_routine,
    char *thread_name,
    pthread_t *thread_id,
    pthread_addr_t thread_arg);
```

```
/* kill_thread:
    thread_id     - thread ID got when create the thread
    thread_name      - the name assigned to thread when the thread is
created

    PURPOSE: Helper function to kill a thread, given a thread ID and
thread name
*/
void kill_thread(pthread_t thread_id, char * thread_name);

/* refresh_identity():
    arg    - expect a (DCE_server *) arg

    PURPOSE: Helper thread to refresh DCE credentials in predefined time
*/
pthread_addr_t refresh_identity(pthread_addr_t arg);

/* key_mgmt():
    arg    - expect a (DCE_server *) arg

    PURPOSE: Helper thread to manage the server key
*/
pthread_addr_t key_mgmt(pthread_addr_t arg);

/* signal_catcher():
    arg- not used

    PURPOSE: Helper function to setup signals and wait for any of these
signals happen
*/
pthread_addr_t signal_catcher(pthread_addr_t arg);

/* class DCE_server:
    PURPOSE: encapsulate the functionality of setting up a DCE server
    most of the default parameters are provided, users can
    leave it as it is and create the class or he/she can change
    the parameters to the value he/she want.

    PREREQUISITE: (required only when the server will export to name
space)

    ENVIRONMENT VARIABLE: this version requires you to set up the
following
    environment variables before running the server. You can write
    a script to setup these environment variable and then invoke the
    server
    SERVER_ENTRY: entry name of CDS to export
    GROUP_ENTRY(optional):  group entry of CDS that will contain the
server entry
    PROFILE_ENTRY(optional): profile entry of CDS that will contain the
group entry

    CONSTRUCTOR PARAMETERS:
    ih     -  interface handle (mandatory)
    prin      -  server principal name (mandatory)
    mgr_type_uuid - default to NULL ( not used in this version )
    mgr_epv   -  default to NULL ( not used in this version )
    p      -  protococol used, default to "all", the others are "udp" or
"tcp"
    authn_lg      -  authentication login (default is no)
    protection    - protection level, default is "rpc_c_protect_level_none
       possible values are:
```

```
            rpc_c_protect_level_none(no authentication)
            rpc_c_protect_level_default(same as rpc_protect_pkt_integ)
            rpc_c_protect_level_connect(authenticate when client connect)
            rpc_c_protect_level_call(authenticate at each remote call)
            rpc_c_protect_level_pkt(authenticate at each packet received)
            rpc_c_protect_level_pkt_integ(ensure none of the data have been
 modified)
        rpc_c_protect_level_pkt_privacy(enscription plus previous)
     authn  - authenication service, default to "rpc_c_authn_none"
        possible values are:
            rpc_c_authn_none(no authentication)
            rpc_c_authn_dce_secret(DCE share-secret key authentication
        rpc_c_authn_default(the same as rpc_c_authn_dce_secret)
        rpc_c_authn_dce_public(reserved for future release)
     authz  - authorization service, default to "rpc_c_authz_none"
        possible values are:
            rpc_c_authz_none(no authorization)
            rpc_c_authz_name(authorization based on client principal)
            rpc_c_authz_dce(perform authorization using client's PAC)
     kt     - keytab file path (mandatory)
     ep     - export to endpoint(default yes)
     ns     - export to namespace(default yes)
     is_listen - does server start listening immediately(default to yes)

     PUBLIC MEMBER FUNCTIONS:
        listen()       - allow server to do customization before actually
 listen for incoming calls
 */
 class DCE_server {
     friend pthread_addr_t refresh_identity(pthread_addr_t arg);
     friend pthread_addr_t key_mgmt(pthread_addr_t arg);
     public:
        //  type declarations first
        enum Myproto {all, udp, tcp};
        DCE_server(
            rpc_if_handle_t ih,
            unsigned_char_t * prin,
            uuid_t *mgr_type_uuid=NULL,
            rpc_mgr_epv_t mgr_epv=NULL,
            Myproto p= all,
            int authn_lgn = 0,
            unsigned32 protection=rpc_c_protect_level_none,
            unsigned32 authn = rpc_c_authn_none,
            unsigned32 authz = rpc_c_authz_none,
            unsigned_char_p_t kt = NULL,
            int ep = 1,
            int ns = 1,
            int is_listen = 1
        );

        ~DCE_server();
        void listen();
    private:
        // methods second
        void register_interface(); void use_protocol(); void
 inq_server_bindings(); void sec_login(); void setup_identity();
        void register_endpoint(); void export_to_namespace(); void
 start_mgmt_thread(); void start_signal_thread();
        void valid_and_cert_identity();

        // attributes last
        rpc_if_handle_t if_handle; uuid_t *mgr_type_uuid_of_obj;
 rpc_mgr_epv_t manager_epv;
```

```
        Myproto myproto; unsigned32 protection_level; unsigned32
authentication; unsigned32 authorization;
        unsigned_char_t * principal; unsigned_char_t * keytab_file;
rpc_binding_vector_t * bv; sec_login_handle_t
        login_context; int authn_login; int endpoint; int name_space;
rpc_if_id_t if_id; unsigned_char_t * server_entry;
        unsigned_char_t * group_entry; unsigned_char_t * profile_entry;
pthread_t refresh_id_thread; pthread_t
        key_mgmt_thread; pthread_t sigcatch_thread; unsigned32 status;
};


/* class DCE_client
   PURPOSE: encapsulate the action of set up server

   CONSTRUCTOR PARAMETERS:
      ih -  interface handle of the interface that a specific server
support
      srv_prin      - the server principal name
      path  -  a string binding or a NSI entry in CDS to locate the
server
      pt -  protection level, default is "rpc_c_protect_level_none
         possible values are:
            rpc_c_protect_level_none(no authentication)
            rpc_c_protect_level_default(same as rpc_protect_pkt_integ)
            rpc_c_protect_level_connect(authenticate when client
connect)
            rpc_c_protect_level_call(authenticate at each remote call)
            rpc_c_protect_level_pkt(authenticate at each packet
received)
            rpc_c_protect_level_pkt_integ(ensure none of the data have
been modified)
            rpc_c_protect_level_pkt_privacy(enscription plus previous)
      authn -  authenication service, default to "rpc_c_authn_none"
         possible values are:
            rpc_c_authn_none(no authentication)
            rpc_c_authn_dce_secret(DCE share-secret key authentication
            rpc_c_authn_default(the same as rpc_c_authn_dce_secret)
            rpc_c_authn_dce_public(reserved for future release)
      authz -  authorization service, default to "rpc_c_authz_none"
         possible values are:
            rpc_c_authz_none(no authorization)
            rpc_c_authz_name(authorization based on client principal)
            rpc_c_authz_dce(perform authorization using client's PAC)
      clnt_prin           - client principal name
      clnt_kt   - client keytab file
      sb     - source of binding handle
         possible values are:
            DCE_client::ns (import the binding from name space)
            DCE_client::str(the source is string binding)
      exp    - if the CDS cache data is out of data, then force high
confidence
            (not implemented yet, so put in the default value)

   PUBLIC METHODS:
      rpc_binding_handle_t & get_binding() const:
         return a binding handle in order for client to call the remote
methods supported by the remote server

*/
class DCE_client {
   public:
      //  type declarations first
```

```
        enum Src_binding {ns, str};
        DCE_client(
            rpc_if_handle_t ih,
            unsigned_char_t * svr_prin,
            unsigned_char_t * path,
            unsigned32 pt = rpc_c_protect_level_none,
            unsigned32 authn = rpc_c_authn_none,
            unsigned32 authz = rpc_c_authz_none,
            unsigned_char_t * clnt_prin=NULL,
            unsigned_char_t * clnt_kt = NULL,
            Src_binding sb=DCE_client::ns,
            int exp = 0);

        ~DCE_client();
        rpc_binding_handle_t & get_binding() const;

    private:
        //  methods second
        void get_binding_handle();  void sec_login(); void
 setup_identity(); void valid_and_cert_identity(); void setup_auth_info();
        //  attributes last
        rpc_if_handle_t if_handle;
        unsigned_char_t * binding_path;
        Src_binding src_binding;
        rpc_binding_handle_t binding_handle;
        unsigned_char_t * client_principal;
        unsigned_char_t * server_principal;
        unsigned32 protection_level;
        unsigned32 authentication;
        unsigned32 authorization;
        int expiration;
        sec_login_handle_t login_context;
        unsigned_char_t * keytab_file;
        unsigned32 status;
    };

#endif
```

alarm_a.c

```
/*
FILE: alarm_a.c

DESCRIPTION:
    This file contains the implementation of a alarm agent server that
accepts registration from client. After a client registers, the server
will save the client's binding information in a list structure.  The
server can inform all the clients in the list when it receives some alert
using secure DCE RPC.
*/

// INCLUDE FILES

#include "myapp.h"
extern "C" {
    #include "alarm_a.h"
    #include "alarm_c.h"
}

#define DEBUG 0

// list node structure to hold binding information
struct Client_t {
```

```
    rpc_binding_handle_t h;
    struct Client_t * next;
};

Client_t * client_header = NULL;

// void notify_client():
// FUNCTIONALITY:
// foreach client on the list, retrieve the binding handle
// annotate the security information on the binding handle
// call the notify interface that the client supports

void notify_client() {
    Client_t *p = client_header;
    error_status_t status;

    while(p) {
        rpc_binding_set_auth_info(
            p->h,
            (unsigned_char_p_t) "alarm_agent",
            rpc_c_protect_level_default,
            rpc_c_authn_default,
            NULL,
            rpc_c_authz_name,
            &status
        );

        if (status != rpc_s_ok) {
            switch (status) {
                case rpc_s_invalid_binding:
                    cout << "rpc_s_invalid_binding\n";
                    break;

                case rpc_s_wrong_kind_of_binding:
                    cout << "rpc_s_wrong_kind_of_binding\n";
                    break;

                case rpc_s_unknown_authn_service:
                    cout << "rpc_s_unknow_authn_service\n";
                    break;

                case rpc_s_authn_authz_mismatch:
                    cout << "rpc_s_authn_authz_mismatch\n";
                    break;

                case rpc_s_unsupported_protect_level:
                    cout << "rpc_s_unsupported_protect_level\n";
                    break;

                default:
                    cout << "unknown error\n";
            }
            cerr << "alarmClient set auth info failed";
            exit(0);
        }
        dce_error(status, "Unable setup security with srvHandle of
alarm_c");

        notify(p->h);

        p=p->next;
    }
}
```

```
// void insertClientInfo(client_t * pClient)
// FUNCTIONALITY:
// insert the client into internal list
void insertClientInfo(Client_t * pClient) {
    Client_t *t;
    Client_t *p = pClient;
    if (client_header) {
        t=client_header;

        while (t->next) {
            t = t->next;
        }

        t->next=p;
        p -> next = NULL;
    } else {
        client_header=pClient;
        client_header->next = NULL;
    }
}


// client_reg()
// FUNCTIONALITY:
// the implementation of the alarm agent interface that allows client
// to register  and the server can convert the binding and save the
// binding into a list

void client_reg(handle_t h, error_status_t *st)
{
    error_status_t status;
    rpc_binding_handle_t SvrHandle;
    Client_t * pClient;

    rpc_binding_server_from_client(h, &SvrHandle, &status);

    if (status != rpc_s_ok) {
        cerr << "can not convert client handle to server handle\n";
        *st = -1;
        return;
    }

    cout << "successfully converted client binding to server binding
handle\n";

    pClient = (Client_t *) malloc(sizeof(Client_t));

    pClient->h = SvrHandle;

    insertClientInfo(pClient);

    cout << "alarmClient registered successfully\n";
    *st=error_status_ok;
}


// pthread_addr_t driver()
// FUNCTIONALITY:
// a thread that will call client's notify function when a key is
pressed
pthread_addr_t driver(pthread_addr_t arg)
```

```
{
   while (1) {
      printf("Press any key to send notification to clients\n");

      getchar();

      notify_client();

      printf("Press another key to go to next round\n");

      getchar();
   }
}


// main()
// FUNCTIONALITY:
// create a alarm_agent
// setup a driver thread to call alarm client's notify function
// listen for the incoming calls
int main() {
   pthread_t driver_id_thread;
   DCE_server alarm_agent(
      alarm_agent_if_v1_0_s_ifspec,
      (unsigned_char_t *) "alarm_agent", // using know existing
principal
      NULL,
      NULL,
      DCE_server::udp,
      1,
      rpc_c_protect_level_pkt_integ,
      rpc_c_authn_dce_secret,
      rpc_c_authz_name,
      (unsigned_char_t *) "/audit/keytabs/alarm_agent",
      1,
      1,
      0);
   create_thread(driver, "driver thread" ,&driver_id_thread, NULL);

   cout << "alarm_agent is listening...\n";

   alarm_agent.listen();
   return 0;
}
```

alarm_a.h

```
/* Generated by OSF DCE IDL compiler */
#ifndef alarm_agent_if_v1_0_included

#define alarm_agent_if_v1_0_included

#ifndef __STDC__
#define volatile

#ifndef IDLBASE_H
#include <dce/idlbase.h>
#endif

#include <dce/rpc.h>

#ifdef __cplusplus
extern "C" {
```

```
#endif

#ifndef nbase_v0_0_included
#include <dce/nbase.h>
#endif

extern void client_reg(

#ifdef IDL_PROTOTYPES
/* [in] */ handle_t binding,
/* [out] */ error_status_t *status
#endif

);

typedef struct alarm_agent_if_v1_0_epv_t {
void (*client_reg)(

#ifdef IDL_PROTOTYPES
/* [in] */ handle_t binding,
/* [out] */ error_status_t *status
#endif

);

} alarm_agent_if_v1_0_epv_t;
extern rpc_if_handle_t alarm_agent_if_v1_0_c_ifspec;
extern rpc_if_handle_t alarm_agent_if_v1_0_s_ifspec;

#ifdef __cplusplus
    }
#endif

#endif
```

alarm_a.idl

```
/* alarm_a.idl file */
[ uuid(000493e0-b8b9-1444-8360-00a0246561aa), version(1.0) ]

interface alarm_agent_if
{
   void client_reg (
       [in] handle_t binding,
       [out] error_status_t *status
}
```

alarm_c.c

```
/*

FILE: alarm_c.c

DESCRIPTION:
   The file contains implementation of a alarm client that is itself
also a server.  The alarm client implements a call back function for a
alarm server to call when there are important events happen.  The client
itself need to call the alarm server client_reg function in order for the
server to locate the client to inform these special events.
 */
```

```
// INCLUDE FILES
#include "myapp.h"
extern "C" {
    #include "alarm_c.h"
    #include "alarm_a.h"
}

// notify()
// FUNCTIONALITY
// the implementation of the function that the alarm agent can all to
notify
// alarm client

void notify(handle_t h)
{
    cout << "emergency announcement from server\n";
}

// main()
// FUNCTIONS
// create c_alarm_client to setup relationship with alert agent
// create alarm_client that is actually a DCE server that exports
// the notify functionality get the binding from the c_alarm_client
// that contains the binding to alarm agent registered with the alarm
// agent  listen for the call back from the server
int main(int argc, char *argv[]) {
    // setup as server first to make sure
    unsigned32 status;

    DCE_client c_alarm_client(
        alarm_agent_if_v1_0_c_ifspec,
        (unsigned_char_t *) "alarm_client",  // using the client principal
name
        (unsigned_char_t *) "/.:/audit/alarm_net1_profile",  // Namespace
search starting point
        rpc_c_protect_level_pkt_integ,
        rpc_c_authn_dce_secret,
        rpc_c_authz_name,
        (unsigned_char_t *) "alarm_client",  // using the client principal
name
        (unsigned_char_t *) "/audit/keytabs/alarm_client",  // client
keytab
        DCE_client::ns
    );


    DCE_server alarm_client(
        alarm_client_if_v1_0_s_ifspec,
        (unsigned_char_t *) "alarm_client",  // using the client principal
name
        NULL,
        NULL,
        DCE_server::udp,
        1,
        rpc_c_protect_level_default,
        rpc_c_authn_default,
        rpc_c_authz_name,
        (unsigned_char_t *) "/audit/keytabs/alarm_client",  // client
keytab
        1,
        0,  // there is no need to  export to name space for alarm_client
        0   // also do not listen until the client is setup
    );
```

```
    const rpc_binding_handle_t &h = c_alarm_client.get_binding();

    client_reg(h, &status);

    dce_error(status, "Unable to register alarm_client interface\n");

    alarm_client.listen();
    return 0;
}
```

alarm_c.h

```
/* Generated by OSF DCE IDL compiler */
#ifndef alarm_client_if_v1_0_included
#define alarm_client_if_v1_0_included

#ifndef __STDC__
#define volatile
#endif

#include <dce/idlbase.h>
#endif

#include <dce/rpc.h>

#ifdef __cplusplus
extern "C" {
#endif

#ifndef nbase_v0_0_included
#include <dce/nbase.h>
#endif

extern void notify(

#ifdef IDL_PROTOTYPES
    /* [in] */ handle_t binding
#endif

);
typedef struct alarm_client_if_v1_0_epv_t {
void (*notify)(

#ifdef IDL_PROTOTYPES
    /* [in] */ handle_t binding
#endif

);
} alarm_client_if_v1_0_epv_t;
extern rpc_if_handle_t alarm_client_if_v1_0_c_ifspec;
extern rpc_if_handle_t alarm_client_if_v1_0_s_ifspec;

#ifdef __cplusplus
}
#endif

#endif
```

alarm_c.idl

```
/* alarm_c.idl */
```

```
[ uuid(0070ea40-b8b1-1444-8360-00a0246561aa), version(1.0) ]

interface alarm_client_if
{
    [maybe] void notify(
        [in] handle_t binding
    );
}
```

This page intentionally left blank

Bibliography

Alfarez Abdul-Rahman. *A Distributed Trust Model* Department of Computer Science, University College London, Gower Street, London WC2E 6BT
http://www.cs.ucl.ac.uk/staff/F.AbdulRahman/docs/

Jonathan Chinitz and Steve Sonnenberg. *DCE/Snare™: A Transparent Security Framework for TCP/IP and Legacy Applications* August 1996 Intellisoft White Paper, IntelliSoft Sorporation, P.O. Box 2645, Acton, MA 01720
http://www.isoft.com/snare/snarewp/snarewp1/snarewp1-contents.html

Mark Crosbie and Eugene H. Spafford. *Defending a Computer System using Autonomous Agents.* 11 March 1994 Proceedings of the 18th National Information Systems Security Conference, October 1995. Technical Report 95-022, Department of Computer Services, Purdue University.

Mark Crosbie and Gene Spafford. *Applying Genetic Programming to Intrusion Detection.* COAST Laboratory, Department of Computer Services, Purdue University.

Naji Habra, Baudouin Le Charlier, Abdelaziz Mounji, and Isabelle Mathieu. *Preliminary report on Advanced Security Audit trail analysis on UNIX (ASAX also called SAT-X).* Technical report, Institut D'Informatique, FUNDP, rue Grangagnage 21, September 26, 1994.

Jeremy Frank. *Artificial Intelligence and Intrusion Detection: Current and Future Directions.* 9 June 1994 Division of Computer Science, University of California at Davis, Davis, CA 95616 under NSA URP MDA904-93-C-4085 and ARPA DOD/DABT63-93-C-0045.

Neil R. Fraser, Curriculum Corporation. *The DCE Audit Service*, Proceedings of the OSF DCE Developer and User Conference, London, England, March 1996

Thomas D. Garvey and Teresa F. Lunt. *Model-Based Intrusion Detection* 1991 Artificail Intelligence Center and Computer Science Laboratory, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025

Lawrence R. Halme and R. Kenneth Bauer. *AINT misbehaving - a taxonomy of anti-intrusion techniques.* In Proceedings of the 18th National Information Systems Security Conference, pages 163-172, October 1995.

Richard Heady, George Luger, Arthur Maccabe, Mark Servilla, and John Sturtevant. *The prototype implementation of a network level intrusion detection system.* Technical Report CS91-11, Department of Computer Science, University of New Mexico, April 1991.

L. T. Heberlein, K. N. Levitt, B. Mukherjee. *A Method to Detect Intrusive Activity in a Networked Environment* Computer Security Laboratory, Division of Computer Science, University of California at Davis, Davis CA 95616

Judith Hochberg, Kathleen Jackson, Cathy Stallings, J. F. McClary, David DuBois, and Josephine Ford. *NADIR: An automated system for detecting network intrusion and misuse.* Computers & Security, 12(3):235-248, May 1993.

Wei Hu. *DCE Security Programming* (ISBN 1-56592-134-8)

Kathleen A. Jackson, David H. DuBois, Cathy A. Stallings. *An Expert System Application for Network Intrusion Detection* Computer Networki Engineering Group, MS B255, Computing and Communications Division, Los Alamos National Laboratory, Los Alamos, New Mexico 87545

C. Kaufman, R. Perlman and M. Speciner. *Network Security* (ISBN 0-13-061466)

Sandeep Kumar, Eugene H. Spafford. *An Application of Pattern Matching in Intrusion Detection.* Technical Report CSD-TR-94-013, The COAST Project, Department of Computer Services, Purdue University.

Sandeep Kumar, Eugene H. Spafford. *A Pattern Matching Model for Misuse Intrusion Detection.* The COAST Project, Department of Computer Services, Purdue University as funded by the Division of INFOSEC Computer Science, U. S. Department of Defense

Sandeep Kumar, Eugene H. Spafford. *A Software Achitecture to support Misuse Intrusion Detection.* Technical Report CSD-TR-95-009, The COAST Project, Department of Computer Services, Purdue University.

Jeffrey D. Kuhn. *Research toward intrusion detection through automated abstraction of audit data.* In Proceedings of the 9th National Computer Security Conference, pages 204-208, September 1986.

Teresa F. Lunt *Automated Audit Trail Analysis and Intrusion Detection: A Survey.* 11th National Computer Security Conference, October 1988, Outstanding Paper Award.

Teresa F. Lunt.. *IDES: An intelligent system for detecting intruders.* In Proceedings of the Symposium: Computer Security, Treat and Countermeasures, Rome, Italy, November 1990.

Teresa F. Lunt and Debre Anderson. *Software Requirements Specification: Next-Generation Intrusions Detection Expert System (NIDES)* March 1993, Computer Sciences Laboratory, SRI International, Meno Park, CA 94025 produced under U.S. Government contract number N00039-92-C-0015 funded by the U.S. Navy SPAWAR.

Victor H. Marshall. *Summary of the Trusted Information Systems (TIS) Report on Intrusion Detection Systems.* 29 January 1991 TIS Report #348 Booz, Allen, Hamilton, Inc.

Abdelaziz Mounji, Baudouin Le Charlier, Denis Zampunieris, and Naji Habra. *Distributed Audit Trail Analysis.* RP-94-007 Nov 1994, In ISOC '95 Symposium on Network and Distributed System Security, 1995.

Phillip A. Porras and Peter A. Neumann. *EMERALD: Event Monitoring Enabled Responses to Anomalous Live Disturbances - Conceptual Overview* 18 December 1996 Computer Science Laboratory, SRI International, 333 Ravenswood Avenue, Menol Park, CA 94025

Phillip A. Porras and Peter A. Neumann. *EMERALD: Event Monitoring Enabled Responses to Anomalous Live Disturbances* Computer Science Laboratory, SRI International, 333 Ravenswood Avenue, Menol Park, CA 94025 paper submission to the 20th National Information Systems Security Conference.

Michael Reiter, Kenneth Birman, Li Gong. *Integrating Security in a Group Oriented Distributed System.* DARPA/NASA Subcontract NAG2-593 and appears in Proceedings of the 1992 IEEE Symposium on Research in Security and Privacy.

Jeffrey I. Schiller MIT I/T Integration Team. *Toward A Safe and Secure Distributed Computing Environment.* http://big-screw.mit.edu:8001/~jis/mitsec/

W. Olin Sibert. *Auditing in a distributed system: SunOS MLS audit trails.* In Proceedings of the 11th National Computer Security Conference, pages 82-90, October 1988.

W. Olin Sibert. *Malicious Data and computer Security.* InterTrust Technologies Corporation, 460 Oakmead Parkway, Sunnyvale, CA 94086

Steven R. Snapp, James Brentano, Gihan V. Dias, Terrance L. Goan, Tim Grance, L. Todd Heberlein, Che-Lin Ho, Karl N. Levitt, Biswanath Mukherjee, Douglass L. Mansur, Kenneth L. Pon, and Stephen E. Smaha. *A system for distributed intrusion detection.* In COMPCOM Spring '91 Digest of Papers, pages 170-176, February/March 1991.

Steven R. Snapp, James Brentano, Gihan V. Dias, Terrance L. Goan, L. Todd Heberlein, Che-Lin Ho, Karl N. Levitt, Biswanath Mukherjee, Stephen E. Smaha, Tim Grance, Daniel M. Teal, and Doug Mansur. *DIDS (Distributed Intrusion Detection System) - Motivation, Architecture, and An Early Prototype.* Computer Security Laboratory, Department of Computer Science, University of California at Davis, Davis California 95616

Steven E. Smaha, *Haystack: An Intrusion Detection System*, Proceedings of the 4th Aerospace Computer Security Applications Conference, Dec. 1988, pp37-44.

S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, D. Zerkle. *GrIDS - A Graph Based Intrusion Detection System for Large Networks* Department of Computer Science, University of California at Davis under DARPA Contract DOD/DABT 63-93-C-0045

Aurobindo Sundaram. *An Introduction to Intrusion Detection* http://www.acm.org/crossroads/xrds2-4/intrus.html

Diego M. Zamboni. *SAINT: A security analysis integration tool.* In Systems Administration, Networking and Security Conference, May 1996.

DCE Future Directions http://osiris.wu-wien.ac.at/inst/zid/abteil/azi/service/aix/htmlbooks/gg242543.00/2543ch8.html

History of Intrusion Detection at SRI/CSL Computer Science Laboratory (CSL), Stanford Research Institute (SRI) International, Menlo Park CA 94025-3493 USA http://www.csl.sri.com/intrusion.html

Intrusion Detection for Large Networks 31 August 1995, University of California at Davis http://www.ito.darpa.mil/Summaries95/A785--UCDavis.html

Security in a Distributed Environment http://osiris.wu-wien.ac.at/inst/zid/abteil/azi/service/aix/htmlbooks/gg242543.00/2543ch4.html#ch4

National Computer Security Center (NCSC) Publication *A Guide to Understanding Audit in Trusted Systems* 28 July 1987 NCSC-TG-001, Library Number S-228,470 National Computer Security Center, 9800 Savage Road, Fort George G. Meade, MD 2755-6000

OSF RFC 63.3 DCE 1.2 Contents Overview http://www.opengroup.org/tech/rfc/mirror-rfc/rfc63.3.txt

OSF RFCs 68.3 DCE 1.2.2 Public-Key Login -- Functional Specification http://www.opengroup.org/tech/rfc/mirror-rfc/rfc68..txt

OSF RFC 70.0 Single Sign-On (not yet published)

OSF RFC 80.1 DCE 1.2 Certification API -- Functional Specification http://www.opengroup.org/tech/rfc/mirror-rfc/rfc80.1.txt

OSF RFC 93.0 DCE 1.2 Security Scalability and Performance -- Functional Specification http://www.opengroup.org/tech/rfc/mirror-rfc/rfc93.0.txt

IBM Publication GG24-2543-00 *IBM DCE Cross-Platform Guide*

IBM Publication GG24-2526-00 *MVS/ESA OpenEdition DCE: RACF and DCE Security Interoperation*

IBM Publication GG24-4482-00 *MVS/ESA OpenEdition DCE: Application Support Servers CICS and IMS*

IBM Publication GG24-4240-00 *MVS/ESA OpenEdition DCE Presentation Guide Volume I*

IBM Publication GG24-3810-00 *Open Software Foundation DCE Global Directory Services and OfficeVision/MVS Enterprise Address Book Overview and Comparison*

This page intentionally left blank

Glossary

/... - The DCE root directory. Every DCE object can be addressed under this directory. Immediately under this directory, you would expect to see the names of DCE cells. This effectively makes every DCE cell "connectable" to every other DCE cell in the world since /... is a global root directory.

/.: - A shortcut that refers to the CDS path /.../cell_name (where cell_name is the name of the local cell). This is used to refer to the local cell.

/: - A shortcut that refers to the CDS path /.../cell_name/fs (where cell_name is the name of the local cell). This is used to refer to the local cell's DFS namespace.

Access Control Lists or ACLs - Based on POSIX 1003.6 ACL Authorization, DCE ACLs are maintained by a resource owner in order to protect its resources. The ACL contains a list of identities that may be authorized to access the resource in some way, and a set of permissions granted to each identity. This is similar to the permission bits on UNIX files.

ACL key - The name of the identity to which the ACL permissions apply. Some ACL types do not have an ACL key.

ACL manager - Application developers may chose to develop an ACL manager as part of the server process that they design. An ACL manager enforces the ACL security mechanisms. It compares the identity of the client making a request with the entries in the ACL database protecting that resource, and either grant or deny the request.

ACL permissions - A set of characters where each character represents a particular permission (to the object that the ACL protects). For example, the permission character "r" may grant "read" permission to the object being protected, while "c" may determine that a principal may change the ACL itself.

ACL type - The type of identity specified in the next field in this ACL entry. For example,. a type might be "user", indicating that the next field in the ACL is the name of a user.

Account (or login account) - Information related to a user/system/server application, required for DCE login. For example, a user called "Neil" may have a login account in the DCE Security Server Registry database which holds his password, home directory etc. This is very similar to the UNIX /etc/passwd file in form and contents.

acl_edit - acl_edit (the ACL editing program) is a DCE version 1.0 utility that displays & modifies ACLs on DCE objects. Much of the functionality of acl_edit has been migrated to the newer "dcecp" command suite.

Application Code - The core logic of the application.

Authentication - Users in a DCE cell are considered to have authenticated once they are logged in. Authentication is essentially proving your identity to the DCE Security Server.

Availability - A high-availability computing solution is one where failure of no single component can interrupt operation.

Binding or RPC binding handle - The connection information shared by a client and server. This information includes: the UUID that defines the interface that client & server share, .an address of the server, a protocol sequence that client & server can communicate with, the port number associated with the server's interface, and optionally the object UUID uniquely identifying that server interface instance. (see also String Binding)

CDS_Convergence - CDS directories and objects hold data in the form "attribute = value". For example, the CDS directory /.:/subsys may have an attribute called CDS_Convergence whose value may be "high," "medium," or "low". This CDS_Covergence value (of meduim) specifies the approximate time period within which changes to a master will be propagated to readonly replicas (copies) of that directory.

cdsadv - The CDS advertiser program. On a CDS server system, cdsadv periodically sends out a UDP broadcast to advertise that this system has a portion of the CDS database. This is the one method through which, clients learn about the location of CDS servers. On a client system, the cdsadv process is the parent process for each cdsclerk process.

cdsclerk - cdsclerk is the process that makes client requests into the CDS database. When used there will be one cdsclerk process per-userid & groupid combination on a DCE system.

cdscp - cdscp (the CDS control program) is a DCE version 1.0 utility that accesses the CDS database. Much of the functionality of cdscp has beed migrated into the newer "dcecp" command suite.

cdsd - cdsd is the name of the Cell Directory Service (CDS) server process. This process manages the clearinghouses that comprise the CDS database.

Cell - A DCE cell is a collection of logically related DCE computer systems that share a common security service and a directory service.

Cell Directory Service Server (CDS server) - The process (cdsd) that maintains a database containing addresses of other DCE processes. Client access to this server is through the CDS advertiser (cdsadv) and CDS clerk (cdsclerk) processes.

cell-profile - A profile entry in the CDS database. It is a special entry in that it is used to help a client locate other information in the CDS database i.e. it is part of the "search engine" used internally by CDS. The cell-profile may contain references to server entries (binding information) for server processes that exist ANYWHERE in a cell.

Child pointer - The pointer that links a parent directory to its children. This is a downward linked list from parent to child throughout the CDS namespace. In this way navigation can span clearinghouses and replicas.

Clearinghouse - A physical portion of the CDS namespace. The CDS namespace can be split across several clearinghouses and any or all of those portions can be replicated in other clearinghouses. The clearinghouse is just an instance of the CDS database.

Client - A DCE client is the process that requests a service from a DCE server process. The client requests this service by means of a Remote Procedure Call (RPC) and the server responds to the RPC in accordance to the interface specification for that RPC binding.

Clock drift or Drift - The local time on a computer is based on an oscillator in the computer's hardware. This oscillator is not perfect and therefore the system time drifts either ahead or behind true UTC.

Clock skew or Skew - When two separate systems clock values drift apart with respect to each other, the difference between their clock values is defined as their clock skew. Clock skew is computed taking into account network propagation delay as much as practicable.

Compatible binding - a client and server are considered compatible if their interface UUIDs match and their version numbers are similar.

Complete binding - see String Binding

Courier Server or courier - A courier server is a special kind of DCE local time server that, when it synchronizes (to set it's own local clock) always takes at least one time value from a global server (usually located in a different location). Then the courier server gives it's own clock time to clients in it's own or LAN segment. Hence, by taking time from an external server and providing time to a set of local systems, the courier ensures that the global server influences the time values held by clerks (clients) in a LAN.

Credentials - Credentials uniquely define your DCE identity. Your credentials include your principal name as well as the groups that you belong to. Credentials are stored in a Extended Privilege Attribute Certificate (EPAC) and are issued at login time.

Dynamically Linked Libraries or DLLs - A Dynamically Linked Library is a file that resides on disk and is used by the main part of an application program whenever it is needed. It is not permanently joined to the application in order to reduce executable size etc and to de-couple the application from it's library. DLL is a Windows specific term though the same concept applies to most modern operating systems.

dcecp - dcecp (the DCE control program) is a DCE version 1.1 programmable shell (based in tcl version 7.3) that allows an administrator to manage DCE services. This control program largely replaces the older control programs cdscp, rpccp, dtscp, rgy_edit, acl_edit, and sec_admin.

dced - dced is the name of the most fundamental DCE process that runs on every DCE system. The process dced performs several tasks :
1) It is the client interface to the security service, i.e. it helps obtain tickets for the client host. dced is the process that logins into the DCE Security Service as hosts/hostname/self.
2) It performs the endpoint mapping task. When a server process starts up, it must register the endpoint that it is listening for requests on with the dced endpoint mapper interface. Later, when a client looks for a particular interface, dced provide the endpoint.
3) It performs the local location broker task. This is a similar "endpoint mapping" service but for an older technology when dced still supports.
4) It performs a data file management surrogate role that enables remote administration.
5) It performs a process management role that enables remote administration.

Directory Replica - A physical instance of a CDS directory (within a particular clearinghouse)

Domain Name Service or DNS - The domain name service (DNS) is the most common naming service used by systems on the internet. DNS helps turn a logical name of a system into a physical address. The domain naming service (DNS) may be used by DCE in order for a client in one cell to locate a server in another cell.

Drift - see Clock Drift

DTS or Distributed Time Service - This is time service that DCE uses to synchronize the local systems clocks throughout a DCE cell. Like Network Time Protocol (NTP), DTS ensures that time will be synchronized without ever allowing time to move backward.

dtscp - dtscp (the DTS control program) is a DCE version 1.0 utility that accesses time service related information. Much of the functionality of dtscp has been migrated to the newer "dcecp" command suite.

dtsd - dtsd is the name of the DCE Time Service (DTS) server process or client process. Since the same binary name may behave as either a server or a client, flags are normally specified to dtsd at startup to define whether to behave as a client or a server.

Dynamic endpoint - Most DCE processes do not use a fixed endpoint (one that never changes). Most DCE processes are assigned an endpoint number (port number) when they start up - this is a dynamic endpoint (since it may change every time the process starts up). This gives rise to the need for an endpoint mapper function supported by dced.

Encrypted - An encrypted message, sent from one computer to another, is a message where all of the data contained in the message is scrambled before transmission. Only those who know the secret key (password) that was used to scramble (encrypt) the message may unscramble (de-crypt) the message.

Endpoint or Port - In DCE, the terms endpoint and port are synonymous. An endpoint (or port) is a refinement of a network address. Since many processes may run simultaneously on a system with an IP address, endpoints allow connections to specific processes. Each server process "listens" for requests on one or more endpoints. Clients then dynamically discover what endpoints the server is listing on for a particular interface, and then contacts the server at the IP address and endpoint (e.g. 100.200.100.200[3456]).

Extended Privilege Attribute Certificate or EPAC - The DCE privilege service is active at login, by providing an authenticated principal (user) with a Extended Privilege Attribute Certificate (EPAC). This EPAC is a ticket that proves the identity of the principal (user) logging in by indicating not only his principal (singular identity) but also the registry groups that the principal belongs to.

Full Binding - see String Binding

Function - Logic executed within a program that has discrete inputs and outputs.

Global Directory Agent (GDA) - A DCE process (called gdad) that provides the interface from the CDS to either DNS or GDS in order to locate a remote cells and their resources within.

Global Directory Service or GDS - The CCITT X.500/ISO 9594 Directory Service is a general purpose naming database that provides a similar service to the Domain Naming Service (DNS). DCE cells can be registered in GDS, DNS or both naming services in order that clients in one cell can locate servers in another cell.

Global Server - A DTS global server is able to provide time values to any system in a DCE cell and registers its services in the /.:/cell-profile object in the CDS namespace.

Group - A group (in the context of the DCE registry) is a collection of related principal identities. This is very similar to the UNIX /etc/group file in form and contents.

Group Entry or NSI group entry - An object in the CDS database that contains a list of CDS names in the CDS database. This effectively is a "network node" in the networked fabric of pointers contained within the hierarchical CDS database.

hostname - When a system is installed in a DCE cell, the directory /.:/hosts/hostname is automatically created in the CDS database and the principal & account hosts/hostname/self is created in the Security Registry database. Example, when the host "earth" is configured into a DCE cell, the directory /.:/hosts/earth will be created in the CDS and the account hosts/earth/self will be created in the Registry.

hosts - the directory (container) called "hosts" in the Cell Directory Service (CDS) database is a database directory (not a filesystem directory) containing other directories that represent each DCE system installed in a particular cell.

IDL - Interface Definition Language. This is a language (similar to C) that defines the data types that are passed between a particular type of client and it's server. Usually the interface is described (created) by the author of the server program and authors of client programs use the definition when compiling their client side programs. The IDL concept was contributed to the OSF/DCE technology suite by HP/Apollo.

IDL Compiler - An IDL Compiler is a compiler that takes files written in a language called IDL and creates C header files. These header files are then compiled into both the client and the server part of a DCE application. Also generated are "stub" files that are object files used by the DCE runtime library to support the RPC communications tasks. The sub files are linked into the final executable application program. These stub files make the calls to the DCE runtime library at execution time.

Intercell - When DCE cells are connected together using GDS/DNS through the GDA, cell-to-cell communication is possible. This intercommunication is referred to as intercell communication.

Interface Identifier - This specifies the interface UUID and interface version that a particular server supports. It uniquely defines that interface in the entire globe. The interface id is hardcoded into the IDL files speechifying the interface, thus clients locate servers using the interface id.

Interface UUID - An interface UUID is a UUID that refers to an interface that a server process supports. For example, suppose we have a client/server image viewing application. The designer of this application may specify that the server must receive a set of character data (to indicate the pathname of the file to display) and the server must return to the client floating point numbers that make up the basis of an image. This definition of in and out data types supported by an application is the applications' interface. This interface is uniquely identitifed with a UUID (an interface UUID).

Interface Version - An interface version is added to the interface identifier to enable enhancements to the interface without disrupting pre-existent clients. This effectively conserves the use of interface UUIDs and allows one to become familiar with frequently used interface UUIDs.

Interoperability - A computing system is said to be interoperable if computers in that system are able to interact with other similar and dissimilar computer systems successfully.

Kerberos - Kerberos is an authentication scheme that was originally developed at MIT. The DCE security service implements version 5 of this protocol and is incompatible with earlier versions of the protocol. DCE has extended the protocol with the use of Priviledge Attribute Certificates.

Key - In DCE, the term key often refers to a binary representation of a password.

Key Files or Keytab - A keytab file (or key table file) is a file that stores a password (key) for a principal/account (typically for a server process or machine host). Server processes and computers, like ordinary users, are required to log in in a DCE environment. They need to store their login passwords in some form of stable storage - hence a file is used. The file is weakly encrypted and is created using the dcecp control program.

LAN - Local Area Network implemented as a ethernet, token-ring, Fiber Distributed Data Interchange (FDDI) or similar technology. LANs have the characteristic of being high speed and fairly reliable versus WANs.

lan-profile - A profile entry in the CDS database. It is a special entry in that it is used to help a client locate other information in the CDS database i.e. it is part of the "search engine" used internally by CDS. The lan-profile may contain references to server entries (binding information) for server processes that exist in a single LAN.

Leap Seconds - UTC time is not based on the rotation of the earth. UTC time therefore must be re-synchronized with the rotation of the earth periodically. This adjustment in UTC is performed by moving the UTC time by one second. DCE systems do not handle gross time adjustments like this well - so at the time UTC moves by a second, DCE time does not. Instead DCE increases it's inaccuracy by a second and gradually adjusts to remove the 1 second error.

Local Server - A DTS local server is one that provides time values to systems within a Local Area Network (LAN). By default, local servers register their interfaces identifiers in the /.:/lan-profile object of the CDS anmespace.

Maximum Inaccuracy or maxinaccuracy - Maximum inaccuracy is a configurable parameter in DTS. It controls how tightly coupled systems are with respect to time.

Member Name - In the context of an NSI profile, the member name is the CDS path that this element of the profile points to (i.e. what server does this element of the profile point to).

Namespace - a namespace is the logical view of a database (of some kind). For example, the CDS namespace is a hierarchical directory structure representing the contents of the CDS database.

ncacn_ip_tcp - This is a DCE reference to the protocol sequence within a binding handle used by an RPC client/server. In this case, the IP protocol has two main flavors: 1) tcp/ip and 2) udp/ip. The former is a connection oriented protocol (cn) and the latter is an datagram protocol (dg). The string ncacn_ip_tcp is a reference to the former (tcp/ip). The character string is made up as follows :
nca - network computing architecture (this just means DCE RPC)
cn - a connection-oriented protocol
ip - the IP protocol (Internet Protocol) used to connect systems
tcp - the TCP protocol (Transmission Control Protocol) used on top of IP

ncadg_ip_udp - This is a DCE reference to the protocol sequence within a binding handle used by an RPC client/server. In this case, the IP protocol has two main flavors: 1) tcp/ip and 2) udp/ip. The former is a connection oriented protocol (cn) and the latter is an datagram protocol (dg). The string ncadg_ip_udp is a reference to the latter (udp/ip). The character string is made up as follows :
nca - network computing architecture (this just means DCE RPC)
dg - a datagram (connectionless) protocol
ip - the IP protocol (Internet Protocol) used to connect systems
udp - the UDP protocol (User Datagram Protocol) used on top of IP

Network Protocol or protocol - A communications transport such as UDP or TCP.

NFS - the "Network File System" from Sunsoft Inc. NFS is an alternate file sharing environment like DFS or Novell's Netware.

NSI - NSI stands for Name Service Interface - The search and retrieval engine inside the CDS database that helps a client request reach appropriate binding information. There are three kinds of NSI entries in CDS:
1) A server entry - a "boundary node" that points to a system where an interface is supported
2) A group entry - a "network node" that points to other names within the CDS database
3) A profile entry - a "network node" with prioritized pointers to other names within the CDS database
Using these three entry types, a network can be established within the hierarchical CDS database that a client can easily navigate to locate applicable servers.

NTP - NTP stands for the Network Time Protocol - A standard used to synchronize time between systems on the internet. DTS is a similar protocol but makes full use of the DCE Directory and Security Services whereas NTP is not DCE aware.

Object UUID - Within a single interface, a server may support several different types of service, each having the same interface. In order to differentiate between these services, each service may have a unique name (an object UUID).

Organization – An organization (like a registry group) is a collection of related principal identities. What differs an organization from a group is that a registry policy can be applied to every member of an organization e.g. every member of a specific organization may have to re-initialize their identity (log in again) every six hours (rather than a default of 10 hours).

Partial Binding - see String Binding

Password - A string of characters that must be typed by a user wishing to log in (authenticate) to a DCE Security Server.

Periodic skulking - CDS will automatically try to send updates from a read/write master copy of a CDS directory to the copies (or replicas) of that directory stored on other CDS server systems - this is called periodic (or automatic) skulking.

Policy - Registry policies control such things as "what is the default lifespan of a DCE ticket" or "what is the minimum number of characters that are required for a DCE password. These policies may be altered for the registry as a whole, or per registry organization.

Port - see Endpoint

Privilege - The DCE privilege service is active at login, by providing an authenticated principal (user) with an Extended Privilege Attribute Certificate (EPAC). This EPAC is a ticket that proves the identity of the principal (user) logging in by indicating not only his principal (singular identity) but also the registry groups that the principal belongs to.

Profile Entry or NSI profile entry - An object in the CDS database that contains a prioritized list of interface identifiers and the related CDS names in the CDS database. This effectively is a "network node" in the networked fabric of pointers contained within the hierarchical CDS database.

Property - Properties of the DCE registry apply to the registry as a whole. An example of a property of a registry is whether the database is marked as read only.

Registry or Password Registry - A centralized database containing login account information for all users, processes and machines in a DCE cell.

Release skulking - Release skulking is the manual process of forcing an update from a read/write master copy of a CDS directory to the readonly copies (or replicas) of that directory stored on other CDS server systems.

Replica or Replication – Certain centralized DCE databases may be replicated on multiple systems for availability and performance reasons. That is, multiple copies (or replicas) of the database may be created. Replication of DCE databases reduces the likelihood that DCE services may become unavailable because of a failure of a single system since other systems (holding copies of the DCE databases) are still available. The DCE Security and Directory services can be replicated.

rgy_edit - this registry editor was a DCE version 1.0 utility that accesses the DCE registry database. Much of the functionality of rgy_edit has been migrated to the newer "dcecp" command suite in DCE 1.1.

RPC - Remote Procedure Call - A call to a procedure or function that executes on the local or another computer system.

RPC Runtime or RPC Runtime Library - Code that executes on *every* DCE system. This code deals with the complexities of connecting to a remote process over a network of some kind. The RPC runtime is needed in order to execute any DCE based RPC.

rpccp - rpccp (the RPC control program) is a DCE version 1.0 utility that accesses RPC-related information in CDS and in a hosts endpoint database. Much of the functionality of rpccp has been migrated to the newer "dcecp" command suite.

Scalability - A computing solution is scalable if it can be extensively expanded without significant modification.

secd - secd is the security server process. It manages accesses to the security database (called the registry) and all centralized security functions within the DCE cell.

Secure - In DCE terms, a secure message is a message that may not be altered during transmission over a network without the receiver knowing that the message was altered. This is done using cryptographic checksums.

Security Server - The process (secd) that maintains a database of login accounts for DCE identities and other security related information. It provides an authentication, privilege, and certification service to the DCE cell.

Service - A server process provides a service of some kind to clients. This service involves executing a procedure on the CPU where the server process runs and returning the result to the client.

Server - A DCE server is a process that provides a service to clients who request services. This service usually involves executing logic on the server's local CPU and returning the results to a client (often over a network).

Server Entry or NSI server entry - An object in the CDS database where a server process registers its binding information (primarily address information about how to locate the server process). This effectively is a "boundary node" in the networked fabric of pointers contained within the hierarchical CDS database.

Server process or server - A process that provides a service to a requesting client process. The service involves execution of a procedure (or function) on the machine running the server process. Server processes may listen for client requests indefinitely, or they may be invoked only when a client request is made.

Session key - A session key is a temporary key issued by the security service to both a client and a server so that they may have a common key (or password) with which they may encrypt data (if they choose to). The first session key obtained by as user is a session key obtained at login.

Skew - see Clock Skew

Skulk - the skulk operation copies the contents of a read/write CDS directory to any readonly replica sites that have been defined for that directory i.e. a skulk operation might copy the CDS directory "/.:/hosts" to all clearinghouses that hold a readonly copy of "/.:/hosts".

Soft link - A soft link is the CDS equivalent of a symbolic link in a Unix filesystem. It appears to be a directory in that it can be traversed. A soft link provides an alternate path to a CDS entity.

String binding - A string binding is an character representation of the information needed for a DCE client to connect to a DCE server. It contains a protocol sequence, an IP address, and possibly a port number. Binding information is stored in CDS to help a client locate a server process. This binding information may be partial or complete. Complete binding information includes (among other things) the IP address of a computer (where the server process runs) AND an endpoint address (on which the server process listens). Partial binding information does not include this endpoint address. By convention, only partial binding information is usually stored in the CDS since port information changes with each restart of a server.

Stub file - A stub file is an object file (.o file) that is linked into a DCE application. The stub file is generated by compiling an IDL file with an IDL compiler - a stub file and a header file are generated. Since the IDL file includes a definition of the data types to be passed between a client and a server, the stub file also includes these definitions. Hence the stub file is the method that DCE uses to "compile in" the interface definition (between client and server) into a DCE executable. Both a DCE client and a DCE server will have their respective stub file linked into their executable programs. This stub file is what actually calls the DCE runtime library at execution time.

Threads - POSIX Pthreads - A specification for an industry-standard application programming interface for executing multiple parallel tasks in a single process. DCE threads are based on the POSIX 1003.4a Draft 4 specification.

Ticket - The DCE security service issues tickets to various identities. These tickets prove the identity of the person or process receiving the ticket. Tickets usually expire or become invalid some time after they are issued. Tickets are used rather than having to reenter the DCE secret key (password) every time proof of identity is required.

Ticket cache - A ticket cache is a directory on the local disk of a DCE client system that houses the tickets that are obtained from the DCE security service on behalf of a user. The permissions on the elements of this cache (the tickets themselves) need to be carefully controlled by local operating system security. If you are able to read another persons ticket, you may masquerade as that person for a limited period of time.

Ticket Granting Ticket or TGT - The first ticket issued to a user upon successfull login. This ticket proves the identity of a principal. It may be used to obtain other tickets (for other client/server transactions).

Time clerk - In reference to DTS, a clerk is a client.

Time Differential Factor or TDF - The difference between UTC time and the local time zone time. For example, time in New York City in December is five hours less than the UTC time.

Time provider - A time provider is a process that obtains time from an external source and provides that time to a DTS server (often a global server).

Time source - The place a time provider gets an accurate time value from.

Time inaccuracy – Since time calculations can never be totally accurate, DTS specifies that the true time lies within a range (say 4pm +/- 1 sec). In this example the current inaccuracy is 1 sec.

UTC or Universal Coordinated Time - An international time standard that has replaced Greenwich Mean Time (GMT). UTC does not base its time on the rotation of the earth, but rather, on an atomic measurement.

UUID - stands for Universal Unique IDentifier. UUIDs are 128 bit numbers that are used internally by DCE as unique names for any kind of resource. Once created, UUIDs are unique in time and space since they include temporal and physical information as part of the coding scheme. For example a user name such as "neil" will have associated with it a UUID such as 01e363e43-1234-5536-0802b37421e. This uniquely identifies "neil" through the entire globe so there will never be ambiguity who this really is.

Wide Area Network or WAN - A wide area network provides computer connectivity over a greater distance than is involved in local area network connections. Wide area networks generally are unlikely to be available 100% of the time i.e. they are less reliable than Local Area Networks (LANs).

X.500 - The reference to the CCITT standard with which DCE can inter-operate.