JOHANNES KEPLER
UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Darstellung von Rechtestrukturen in Verzeichnisdiensten (am Beispiel Active Directory)

DIPLOMARBEIT

zur Erlangung des akademischen Grades

DIPLOMINGENIEUR

in der Studienrichtung

INFORMATIK

Angefertigt am *Institut für Informationsverarbeitung und Mikroprozessortechnik*

Betreuung:
o. Prof. Dr. Mühlbacher Jörg R.
Dipl.-Ing. Hörmanseder Rudolf

Eingereicht von:
Helml Thomas

Linz, *November 2001*

Abstract

Directory Services, e.g. Novell Directory Services or Microsofts Active Directory Services (ADS) get more and more relevance for computer networks. Often the entire network structure of an enterprise is stored in the directory. Therefore the management of authorisation and access-control is an important part of the security administration.
Questions like "Who has permission on a dedicated object?" can be answered easily by standard tools. In the context of this diploma thesis an ADS security analysis tool has been designed which also can answer questions of the type "Where does a dedicated user have access-permission?" in a short and meaningful way. This additional view on permissions makes it easier to detect security holes.
This work describes the functionality and structure of directory services in general. After a special part concerning Microsoft Active Directory and its security mechanisms a description of the implementation is given. The last section describes results of the security analysis.
The diploma thesis has been developed as part of the project *Security analysis Tool (SAT)* at the Institute for Information Processing and Microprocessor Technology (FIM). The SAT-project is sponsored by Microsoft Research (MSR) Cambridge (UK).

Zusammenfassung

Verzeichnisdiensten, wie z.B. den Novell Directory Services oder Microsofts Active Directory Services (ADS) kommt in Netzwerken ein immer größerer Stellenwert zu. Im Verzeichnis selbst ist oft die gesamte IT-Struktur des Unternehmens wie in einem Organigramm abgebildet. Die korrekte Rechtevergabe auf diese Firmendaten ist damit eine wesentliche Aufgabe der IT-Sicherheitsadministration.
Das im Rahmen der Diplomarbeit erstellte ADS-Analysetool zeigt auf, wie neben der standardmäßig verfügbaren Sicht „Wer hat auf ein ausgewähltes Objekt im ADS welche Rechte?" auch Fragen wie: „Wo hat ein bestimmter User im ADS welche Rechte?" prägnant und aussagekräftig beantwortet werden können. Durch diese zusätzliche Sicht auf die Rechte können Sicherheitslücken leichter erkannt werden.
Die Arbeit beschreibt zuerst Funktion und Aufbau von Verzeichnisdiensten allgemein. Dann wird ein Schwerpunkt auf den Verzeichnisdienst Active Directory von Windows 2000 und dessen Sicherheitsfunktionen gelegt. Nach der darauf folgenden Implementierungsbeschreibung werden einzelne Ergebnisse der Sicherheitsanalyse vorgestellt.
Die vorliegende Diplomarbeit entstand im Zuge des Projekts *Security Analysis Tool (SAT)* am Institut für Informationsverarbeitung und Mikroprozessortechnik (FIM). Dieses Projekt wird von Microsoft Research (MSR) Cambridge (UK) finanziell unterstützt.

Danksagung

Mein Dank gilt allen, die zur Erstellung dieser Arbeit beigetragen haben:

- Meinem Betreuer o. Prof. Dr. Mühlbacher Jörg für das mir entgegengebrachte Vertrauen, sowie für die Bereitstellung von Räumlichkeiten und Betriebsmitteln.

- Dem Instituts-Mitarbeiter, Dipl.-Ing. Hörmanseder Rudolf, der viel Zeit und Energie in dieses Projekt investiert hat. Von ihm stammen auch viele der hier beschriebenen und implementierten Überlegungen.

- Microsoft Research (MSR) Cambridge (UK) für die finanzielle Unterstützung und ihr Interesse am Security Analysis Tool (SAT).

- Meinen Eltern für ihre Hilfe in den letzten Jahren. Sie stellten ihre eigenen Bedürfnisse zurück, um mir eine akademische Ausbildung zu ermöglichen. Der Dank, der ihnen gebührt, kann nicht in Worte gefasst werden.

- Besonders wichtig war auch die Unterstützung meiner Lebensgefährtin Claudia. Sie und auch meine Tochter Julia mussten in der letzten Zeit sehr oft auf ein geregeltes Familienleben verzichten. Ohne sie wäre ich nicht, wo ich heute bin. Danke.

Inhaltsverzeichnis

1 Einleitung und Motivation

In einer von KPMG durchgeführten Studie [Kpmg] zum Thema „E-fraud - is technology running unchecked?“ nahmen Unternehmensverantwortliche aus 12 Ländern rund um den Globus teil. Die Auswertung der 1253 ausgefüllten und retournierten Fragebögen ergab, dass eine Mehrheit von 79% der Befragten der Auffassung sind, Angriffe könnten nur von außen - also aus dem Internet - kommen.

Daten müssen jedoch vor unerlaubtem Zugriff geschützt werden, egal ob von außen oder innen. Der Rechtevergabe kommt somit ein besonders hoher Stellenwert zu. Sie regelt, was System-Benutzer dürfen und was nicht. Selbst kleine Änderungen an bestehenden Rechtestrukturen haben eventuell gravierende Auswirkungen und führen im schlimmsten Fall zu Sicherheitslücken im System.

Die Kombination aus wachsenden Netzwerken, steigenden Benutzerzahlen und den vielen Möglichkeiten der heutigen Betriebsysteme sind verantwortlich für die zunehmend höhere Komplexität in der Administration. Der Überblick geht dabei leider oft verloren. Aus diesen Gründen werden Sicherheits-Analyse-Programme für Administratoren immer wichtiger. Das Projekt SAT – Security Analysis Scanner – wurde aus eben diesen Gründen ins Leben gerufen.

Diese Arbeit behandelt einen speziellen Schwerpunkt der Sicherheitsanalyse: die Analyse des Active Directory.

Übersicht

Im ersten Kapitel wird ein Überblick über das SAT 2-System gegeben. Die Entstehungsgeschichte von SAT wird präsentiert und in weiterer Folge auf die Architektur des Analyseprogramms eingegangen. Kapitel 1 sollte als Einstieg in die vorliegende Arbeit gewählt werden.

Das zweite Kapitel bildet einen Hauptschwerpunkt dieser Arbeit. Zu Beginn wird eine grundlegende Einführung in die Thematik der Verzeichnisse bzw. Verzeichnisdienste gegeben (Kap. 2.1). Im nächsten Teil (Kap. 2.2) wird X.500 als „Urvater“ aller Verzeichnisdienste besprochen. Interessant für den weiteren Verlauf ist dieses Kapitel insofern, da LDAP ursprünglich als „Bereicherung“ für X.500 konzipiert wurde, bevor daraus ein eigener Verzeichnisdienst entstand. Kapitel 2.3 beschäftigt sich ausschließ-

lich mit den Grundlagen von LDAP und vervollständigt somit den Überblick über Verzeichnisdienste. Der darauffolgende Teil (Kap. 2.4) beschäftigt sich mit den konkreten Implementierungen: Microsoft Active Directory bzw. OpenLDAP für Unix / Linux. Vertiefend werden die Bereiche „Security“ und das „Schema“ vorgestellt. Diese beiden Themen bilden wichtige Grundlagen für den ADS-Scanner (Kap. 3), darum sollten sie zum besseren Verständnis vorher studiert werden.

Kapitel 3 stellt neben Kapitel 2 den zweiten Schwerpunkt dieser Arbeit dar. Die beiden einführenden Teile (Kapitel 3.1 und 3.2) bilden einen Überblick über die Implementierung eines Sicherheitsanalyse-Programms „ADS-Scanner“ für das Active Directory. Die verbleibenden Teile des dritten Kapitels vertiefen sich sehr stark in die Implementierung des ADS-Scanners und sind als Referenz für zukünftige Änderungen und Verbesserungen am SAT-System gedacht. Der interessierte Leser ist natürlich eingeladen, sich über das Datenbankkonzept (Kapitel 3.4), die Komprimierung (Kapitel 3.5) sowie Implementierungsdetails (Kapitel. 3.6) ausgiebig zu informieren.

Abschluss dieser Arbeit bildet Kapitel 4, worin anhand praktischer Testläufe des ADS-Scanners Aufschlüsse über die Beschaffenheit des Active Directory sowie über die Effizienz des ADS-Scanners gegeben werden sollen. Im Zuge der Fallstudie wurden auch Untersuchungen über die Erweiterung des Active Directorys durch Drittprogramme angestellt. Die Ergebnisse werden ebenfalls in diesem Teil vorgestellt. Kapitel 4 kann somit auch für jenen Teil der Leserschaft, welche den Implementierungsteil ausgelassen hat, von Interesse sein.

1.1 Security Analysis Scanner v 2.0

1.1.1 Motivation

Eine der Hauptaufgaben eines Administrators ist es, den Zugriff auf Systemressourcen zu verwalten. Die Rechtevergabe erfolgt dabei meist auf Objektebene, d.h. für jedes Objekt wird einzeln festgelegt, wer darauf mit welchen Rechten Zugriff erhält. Im Dateisystem beispielsweise können Objekte, auf die man den Zugriff kontrolliert, Dateien und auch Verzeichnisse sein. Zusätzliche Komplexität wird noch durch einen Vererbungsmechanismus für solche Zugriffsrechte geschaffen.

Die zur Verfügung stehenden Tools (z.B. Windows-Explorer) ermöglichen es dem Administrator, diese Zugriffskontrolle schnell und einfach zu implementieren. Ein Computersystem ist aber leider nicht statischer Natur, vielmehr „lebt“ es auf seine Weise und unterliegt ständigen Änderungen. Je größer ein System wird, je mehr Benutzer darauf zugreifen können, desto höher wird auch die Wahrscheinlichkeit, dass sich Fehler einschleichen und unerkannt bleiben.

Von Zeit zu Zeit wäre es daher wünschenswert, einen Gesamtüberblick über den derzeitigen Zustand zu bekommen. Interessant wäre so eine Funktionalität vor allem, wenn man es mit einem neuen System zu tun hat, bzw. wenn ein neuer Mitarbeiter eingewiesen werden soll. Das Betriebssystem unterstützt solch ein Vorhaben allerdings nur in beschränktem Ausmaß. Um einen genauen Überblick zu bekommen, müsste jedes Verzeichnis und eventuell sogar jede Datei einzeln überprüft werden. Beschränkt man sich alleine auf einen kleinen Server, so sieht man sich schnell mit optimistisch geschätzten 100.000 Dateien und Verzeichnissen konfrontiert. Dabei wurden aber andere sicherheitsrelevante Systeme, wie Datenbanken oder auch das Active Directory, außen vor gelassen. Hilfreich für den Administrator wäre hier eine sehr gute Dokumentation, welche aber in der Erstellung und Wartung auch sehr aufwendig ist und deren Erfolg zudem von der akribischen Genauigkeit des Verantwortlichen abhängt. Um ein System mit mehreren hundert oder sogar tausend ständig wechselnden Benutzern zu verwalten, wird aber auch diese Methode über kurz oder lang versagen.

Einziger Ausweg aus diesem Problem ist somit eine softwareseitige Unterstützung, also ein Analyseprogramm. Da ein solches nicht im Lieferumfang von Windows enthalten ist, wurde das Projekt SAT – Security Analysis Scanner – ins Leben gerufen, siehe dazu auch [Hoe].

1.1.2 Geschichte

Die Idee zum SAT-System stammt bereits aus 1995. Nach einer Machbarkeitsstudie folgte die erste Implementierung des Security Analysis Scanner. Diese wurde vom damaligen Studenten Hanner Kurt im Jahr 1998 fertiggestellt. Seine Diplomarbeit „Analyse und Archivierung von Benutzerberechtigungen in einem Windows NT Netzwerk“ [Han] beschreibt das damalige SAT-System sowie dessen Implementierung.

Mit Anfang 1999 wurde SAT 1.0 Beta der Öffentlichkeit vorgestellt und ist seit diesem Zeitpunkt im Internet unter der Adresse http://www.fim.uni-linz.ac.at/sat/ kostenlos zu beziehen. In der Sicherheitsanalyse beschränkt sich SAT 1.0 auf NTFS in der damaligen

Version 4. Es hat mehrere kleine „Schönheitsfehler“, wobei ein wirkliches großes Problem bestand. SAT 1.0 ist nur für Windows NT 3.x und 4.0 konzipiert, es arbeitet jedoch nicht mehr unter Windows 2000. Grund war die neue Version des Dateisystems NTFS 5, welches mit Windows 2000 eingeführt wurde.

Dem Projektleiter und –initiator Hrn. Dipl.-Ing. Rudolf Hörmanseder gelang es in Gesprächen mit MSR - Microsoft Research (Cambridge, UK) [Msr] diese als Sponsoren für das Projekt SAT zu gewinnen. Bedingung seitens Microsoft Research war, dass sämtliche Forschungen und Ergebnisse des Projektes der Öffentlichkeit frei zugänglich gemacht werden müssen.

Der offizielle Startschuss für das neue Projekt SAT 2 erfolgte im Winter 2000/2001. Das Projektteam wuchs auf insgesamt 4 Mitarbeiter an.

Das zum damaligen Zeitpunkt noch relativ neue Windows 2000 brachte eine Vielzahl an Neuerungen: NTFS 5, die Active Directory Services sowie den breiten Einsatz der Administrationsoberfläche Microsofts Management Console (MMC). Diese Neuerungen legten eine fast vollständige Neuimplementierung von SAT nahe. In die Sicherheitsanalyse wurde neben NTFS 5 die Windows-Registrierung (Registry) sowie das Active Directory aufgenommen. Zur Visualisierung und Steuerung des Systems sollten MMC-SnapIns implementiert werden. Für das Projektteam bedeutete dies eine relativ lange Einarbeitungszeit in diese neuen Technologien, da hier noch wenige Erfahrungen verfügbar waren.

Das Projektteam und deren Aufgabenbereich sei an dieser Stelle kurz vorgestellt:

- Die Projektleitung blieb bei Herrn Dipl.-Ing. Rudolf Hörmanseder.
- Der Zuständigkeitsbereich von Achleitner Michael wurde das neue NTFS 5, sowie die Analyse der Windows-Registrierung. Für Details sei der Leser auf [Ach] verwiesen.
- Herr Zarda Gerald wurde mit der Implementierung der Auswertungsfunktionalität sowie der Visualisierung mittels MMC-SnapIns beauftragt. Die Gruppen- und User-Analyse wurde ebenfalls von ihm implementiert. Seine Erfahrungen und Ergebnisse werden in [Zar] präsentiert.
- Der Autor dieser Arbeit übernahm die Aufgabe, das Active Directory in die Sicherheitsanalyse zu integrieren. Weiters erforderte ein Umstieg von Access (SAT 1.0) zur Datenspeicherung auf Microsofts SQL-Server 2000 eine Neuimplementierung der Datenbankschnittstelle. Diese wurden ebenfalls vom Autor

übernommen. Für Details dazu sei auf das Kapitel 3, in welchem die Implementierung beschrieben wird, verwiesen.

1.1.3 Architektur

Abstrakt betrachtet besteht das SAT 2 System nur aus 2 Komponenten: dem Analysesystem und einer Auswertungskomponente. Das Analysesystem lässt sich wieder in 2 Teilen unterteilen: dem Scanner und einem sogenannten Controller-Programm.

Der Scanner erledigt hier die eigentliche Hauptarbeit – er ist für die Analyse im engeren Sinn zuständig. Pro Rechner, den man analysieren will, muss diese Scanner-Komponente installiert sein. Diese wurde als Windows-Dienst (auch genannt „Service“) implementiert, welcher “on-demand” startet (siehe dazu [Spe/Seite 668]: *“Service: A program, routine, or process that performs a specific system function to support other programs, particularly at a low […] level”*).

Die Daten der Analyse werden vom Scanner direkt in die Datenbank übertragen (von SAT wird derzeit nur der SQL-Server unterstützt).

Jeder Scanner besteht aus mehreren Komponenten – jede davon ist spezialisiert auf die Analyse eines bestimmten (Windows-) Systembereiches:

- User-/Gruppenanalyse: Lokale als auch globale User und Gruppen werden ausgelesen und die Zugehörigkeit ermittelt (welcher User gehört zu welcher Gruppe, usw.)
- NTFS-Analyse: Das Dateisystem der ausgewählten Platten wird analysiert
- Registry-Analyse: untersucht die Windows Registrierung (auch genannt „Registry“, siehe dazu [Kb]: *„The Windows Registry is a central hierarchical database used in Microsoft Windows […] to store information necessary to configure the system for one or more users, applications and hardware devices.”*)
- Active Directory-Analyse: analysiert das Active Directory - wird sehr ausführlich in Kapitel 3 beschrieben.

Anzumerken ist an dieser Stelle, dass die eben genannten Komponenten nicht parallel auf einem Rechner starten. Die einzelnen Scanner-Komponenten werden sequentiell ausgeführt, um die Netzwerklast und somit auch den Datenbank-Server zu entlasten.

Zweiter Bestandteil des Analyse-Systems ist das Controller-Programm. Das Controller-Programm ist für die Steuerung des SAT-Systems zuständig, es fungiert als Schalt-

zentrale des Analysesystems. Vom Controller ausgehend werden die Scanner, welche auf den zu untersuchenden Maschinen installiert sind, gesteuert. Neben dem Startzeitpunkt der Analyse wird vom Controller-Programm auch festgelegt, welche Scanner-Komponente auf welchem Rechner startet.

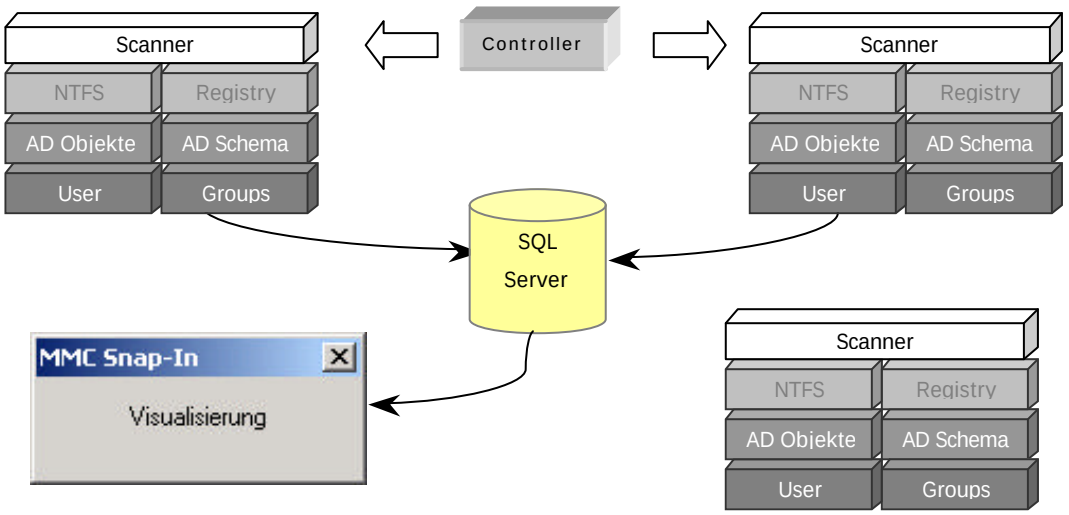

Abb. 1: Architektur SAT-System

Der zweite Teil des SAT-Systems ist das Auswertungsprogramm. Dieses wird derzeit von Herrn Zarda als MMC-SnapIn (Microsofts Management Console) entwickelt, siehe [Zar]. Der Benutzer dieses Programms (i.d.R. der Administrator) wird dabei die Möglichkeit haben, zwischen mehreren Visualisierungsarten auszuwählen. Die Daten für die Auswertung kommen direkt aus der Datenbank, d.h. das Auswertungsprogramm selbst benötigt keine Rechte auf das analysierte System.

2 Lightweight Directory Access Protocol (LDAP)

2.1 Einleitung und Übersicht

Dieses Kapitel soll dem Leser einen Überblick über Verzeichnisdienste vermitteln. Neben den Grundlagen wird vor allem auf den LDAP-Verzeichnisdienst – zu dem auch das Microsofts Active Directory gehört - eingegangen. Der Erweiterungsfähigkeit von Verzeichnisdiensten (Stichwort: das Schema) wurde ein eigenes Unterkapitel gewidmet. Eine Beschreibung der Sicherheitskonzepte, welche in Verzeichnisdiensten Anwendung finden, runden letztendlich das Bild ab. Als konkrete Implementierungen werden Microsofts Active Directory und der frei verfügbare Verzeichnisdienst OpenLDAP vor- und gegenübergestellt.

2.1.1 Verzeichnis

In der einschlägigen Literatur findet man mehrere, meist sehr ähnliche Definitionen zum Thema „Verzeichnis“. Zum Beispiel definiert Microsoft dies folgendermaßen [Spe/Seite 17]: *„Ein Verzeichnis ist eine gespeicherte Ansammlung von Information über - in irgendeiner Art und Weise zusammenhängende – Objekte“* (sinngemäß übersetzt).

[Ldap/Seite 2] beschreibt Verzeichnisse so: *„Ein Verzeichnis ist eine Auflistung von Informationen über - in irgendeiner Weise angeordnete - Objekte, welches (das Verzeichnis) Details über jedes einzelne Objekt beinhaltet“* (sinngemäß übersetzt).

Telefonbuch – Beispiel für Verzeichnisse

Ein Verzeichnis lässt sich am Besten durch einen kleinen Vergleich erklären. In der Fachliteratur findet sich immer wieder das Beispiel „Telefonbuch“, dieses darf natürlich auch in dieser Arbeit nicht fehlen. Ein Telefonbuch ist im Prinzip nichts anderes als ein einfaches Verzeichnis. Betrachten wir es kurz in Anlehnung an die [Ldap] Verzeichnis-Definition:

Die in einem Telefonbuch gespeicherten Objekte sind üblicherweise Personen. Diese werden alphabetisch geordnet, wobei die Details für jede Person (Adresse, Telefonnummer) einsehbar sind. Dies entspricht genau dem, was man im deutschsprachigem Raum unter dem Begriff Telefonbuch kennt.

Eine andere Art eines „Telefonbuches“ sind die sogenannten „Gelben Seiten“, auch bekannt als Branchenverzeichnis. Dieses Beispiel soll verdeutlichen, wie die Verwendung eines Verzeichnisses aussieht. Die Suche nach einer Telefonnummer gestaltet sich, je nachdem über welche Information man verfügt, anders. Kennt man den Namen einer Person, so kann man die Nummer direkt nachschlagen. Verzeichnisse können aber nicht nur solche Fragen beantworten, sie lassen auch eine Suche zu wie: „Suche alle Computerfachgeschäfte in Wien, 2. Bezirk“, eine Suchabfrage wie sie z.B. die Gelben Seiten anbieten.

Tagtäglich werden Telefonbücher benutzt, um die Telefonnummer oder auch die Adresse einer Person ausfindig zu machen. Die gedruckte Version hat leider einen kleinen Schwachpunkt: Der Inhalt ist begrenzt durch die Größe des Buchs (Anzahl der Seiten) und das Suchen gestaltet sich als relativ zeitaufwendig. Ein Telefonbuch mit 10.000 Seiten würde kaum auf eine sehr hohe Akzeptanz stoßen. Möchte man wirklich ein globales, weltweites Telefonbuch, so hätte dies einen noch wesentlich größeren Seitenumfang. Zudem wäre die Handhabung schier unmöglich und der Inhalt eines gedruckten Buches ist sowieso nach relativ kurzer Zeit völlig überholt. Elektronische Verzeichnisse hingegen unterliegen diesen physikalischen Begrenzungen nicht. Stellen wir uns vor, es gäbe so ein globales Telefonbuch, welches technisch durchaus realisiert werden könnte. Verfügbar über das Internet könnte es sämtliche gedruckten Telefonbücher der Welt ersetzen. Per Mausklick erhielte man zusätzlich zur Telefonnummer die E-Mail Adresse der gesuchten Person. Wer einmal versucht hat herauszufinden, ob eine Person überhaupt eine E-Mail Adresse besitzt, würde sich über eine solches Verzeichnis sicherlich freuen.

Dies alles hat jedoch nicht nur positive Seiten, man muss auch die negative Seite betrachten. Treiben wir das Beispiel eines globalen, erweiterten Telefonbuches ein wenig zur Spitze. Nehmen wir an, es gäbe so ein „Bürgerverzeichnis“ in welchem die Daten für jeden Staatsbürger eines Landes gespeichert sind. Dieses beinhaltet neben Adresse und Telefonnummer, E-Mail und Web-Adresse auch sämtliche persönliche Daten über einen Bürger. Angefangen beim Jahresverdienst, Familienstand, Religionsbekenntnis sowie die Krankengeschichte und Steuererklärungen wird alles in diesem vom Staat

kontrollierten Verzeichnis gespeichert. In der einen oder anderen Form wird dies vielleicht sogar bald Realität werden - in manchen Bereichen ist es vielleicht sogar schon soweit. Faktum ist jedoch, dass Verzeichnisse jeder Art vor unerlaubten, nicht-berechtigtem Zugriff geschützt werden müssen.

2.1.2 Vergleich: Relationale Datenbank – Verzeichnis

Eines muss ganz klar festgehalten werden: Ein Verzeichnis kann nicht mit einer relationalen Datenbank gleichgestellt werden, obwohl dies auf den ersten Blick so erscheinen mag. Ein guter Vergleich wurde in [Ldap/S. 2ff.] angestellt, die folgenden Punkte fassen dies kurz zusammen:

- Ein Verzeichnis ist eine hoch spezialisierte Datenbank, mit dem Unterschied zu einer „reinen“ Datenbank, dass der Zugriff auf Verzeichnisse hauptsächlich lesend (inkludiert die Suche) stattfindet. Nur ein kleiner Prozentsatz der Zugriffe ist schreibend – Telefonbücher werden oft durchsucht, die Telefonnummern selbst ändern sich dagegen relativ selten. Datenbanken hingegen müssen sowohl Lese-, als auch Schreibzugriffe sehr schnell erledigen. Verzeichnisse werden hauptsächlich auf den Lesezugriff optimiert, sowohl Server als auch die Clientprogramme erfahren diese Optimierung gleichermaßen.

- In Verzeichnissen werden eher statische Informationen gespeichert, sie sind nicht dafür ausgelegt um dynamische – sich ständig ändernde – Informationen zu verwalten.

- Ein Muss in modernen Datenbanken ist eine Transaktionsverwaltung. Verzeichnisse können Transaktionen unterstützen, müssen dies aber nicht. In den wenigsten Verzeichnissen sind sie tatsächlich verfügbar, erhöhen sie doch die Komplexität bei der Implementierung drastisch. Die Frage, warum Transaktionen nicht benötigt werden, beantwortet sich von selbst, wenn man den Hauptanwendungsbereich von Verzeichnissen betrachtet: Es werden kaum Schreiboperationen durchgeführt. Dort wo (fast) nichts geschrieben wird, kann auch nichts (nicht viel) dabei schief gehen.

- Verzeichnisse müssen nicht strikt konsistent sein. Da die Replikation (siehe Kapitel 2.2.2) von Informationen zwischen Verzeichnis-Server nicht immer unmittelbar nach einer Änderung stattfindet, kann es vorkommen, dass Daten kurzfristig inkonsistent sind. Dies stellt in der Regel aber kein Problem dar.

- Die Zugriffsmethoden auf Verzeichnisse und Datenbanken unterscheiden sich gravierend. Datenbanken verwenden sehr mächtige, standardisierte Abfragesprachen wie z.B. die Structured Query Language (SQL), die komplexe Zugriffe auf Datenbanken ermöglichen. LDAP Verzeichnisse hingegen verfügen über ein einfaches und optimiertes Zugriffsprotokoll.

Erwähnenswert ist vielleicht noch die Tatsache, dass die Datenbanken hinter Verzeichnissen sehr oft relationale Datenbanken sind. Als Beispiel sei hier IBM's eNetwork LDAP Directory (heute: IBM SecureWay Directory [Ibm]) angeführt.

2.1.3 Verzeichnisdienst

Es wurde bereits sehr ausführlich der Begriff „Verzeichnis“ erklärt. Was ist nun ein eigentlich ein Verzeichnisdienst?
Die Online-Enzyklopädie „Webopedia“ [Web1] findet zum Stichwort „Verzeichnisdienst“ folgende Definition: *„Ein Verzeichnisdienst ist ein Netzwerkdienst, der alle Ressourcen eines Netzwerks kennt, und diese sowohl Benutzern als auch Anwendungen zugänglich macht. Ressourcen beinhalten dabei E-Mail Adressen, Computer, und Peripherie wie z.B. Drucker. Idealerweise verbirgt der Verzeichnisdienst sowohl die physikalische Netzwerktopologie als auch die verwendeten Protokolle, sodass ein Benutzer eines Netzwerks Ressourcen verwenden kann, ohne zu wissen, wo oder wie diese physikalisch angeschlossen sind.“ (frei übersetzt)*

Eine etwas allgemeinere Definition des Begriffs „Verzeichnisdienst“ könnte so aussehen: Ein Dienst, welcher den Zugriff auf ein Verzeichnis ermöglicht. Der Zugriff kann sowohl schreibend als auch lesend erfolgen, wobei der Verzeichnisdienst dafür Sorge zu tragen hat, dass der Zugreifende zum auszuführenden Zugriff berechtigt ist.

Beispiel für einen Verzeichnisdienst

In Kapitel 2.1.1 wurde der Begriff „Verzeichnis“ auf ein reales, leicht vorstellbares Beispiel umgemünzt: das Telefonbuch bzw. die Gelben Seiten. Dies soll nun ebenfalls mit dem Verzeichnisdienst versucht werden.

Ein Telefonbuch entspricht also einem simplen Verzeichnis. Die Telefonauskunft wäre in diesem Vergleich ein passender Verzeichnisdienst. Die Auskunft ist ein Dienst, welcher den Zugriff auf ein Verzeichnis (Telefonbuch) ermöglicht. Man ruft die Auskunft an und kann sowohl die Telefonnummer eines Teilnehmers als auch seine Adresse erfragen. Diese Auskunft erhält man aber nur, wenn man dazu befugt ist. Geheimnummern beispielsweise werden nicht weitergegeben. Die Telefonauskunft erlaubt nur einen „lesenden“ Zugriff, die Telefonbucheinträge können nicht geändert werden.

Der wichtigste Verzeichnisdienst im Internet ist zweifelsohne das Domain Name System (DNS). Das DNS-Verzeichnis speichert Domain-Namen sowie die dazugehörigen Rechner-Adressen (IP-Adressen). Sehr vereinfacht dargestellt beantwortet ein DNS-Server Anfragen der Art: „Welche IP-Adresse hat www.uni-linz.ac.at?“. Die Funktionsweise des Domain Name System ist in den RFCs 1034 und 1035 definiert.

Ebenso wie DNS ist auch der „finger“-Dienst als Verzeichnisdienst zu sehen. Bereits im Jahr 1977 wurde dieser Dienst in RFC 742 definiert.

2.1.4 Architektur - Verzeichnisdienst

Die typische Architektur eines Verzeichnisdienstes lässt sich folgendermaßen beschreiben: Ein Klient (Directory Client) nimmt vom Benutzer des Verzeichnisdienstes eine Anfrage (Request) entgegen. Somit bildet der Directory Client die Mensch-Maschine Schnittstelle. Der Directory Client greift nicht direkt auf das Verzeichnis zu, sondern reicht seine Anfragen durch Funktionsaufrufe an ein spezielles API (Application Program Interface) weiter. Dieses generiert eine Nachricht, welche an einen anderen Prozess, den Directory Server, geschickt wird. Der Directory Server verarbeitet diese Nachricht und greift auf das Verzeichnis zu. Je nach Art des Zugriffs kann dies sowohl lesend oder auch schreibend sein. Nach Ermittlung des Ergebnisses wird eine Antwort (Reply) generiert, welche dem Client zurückgeschickt wird.

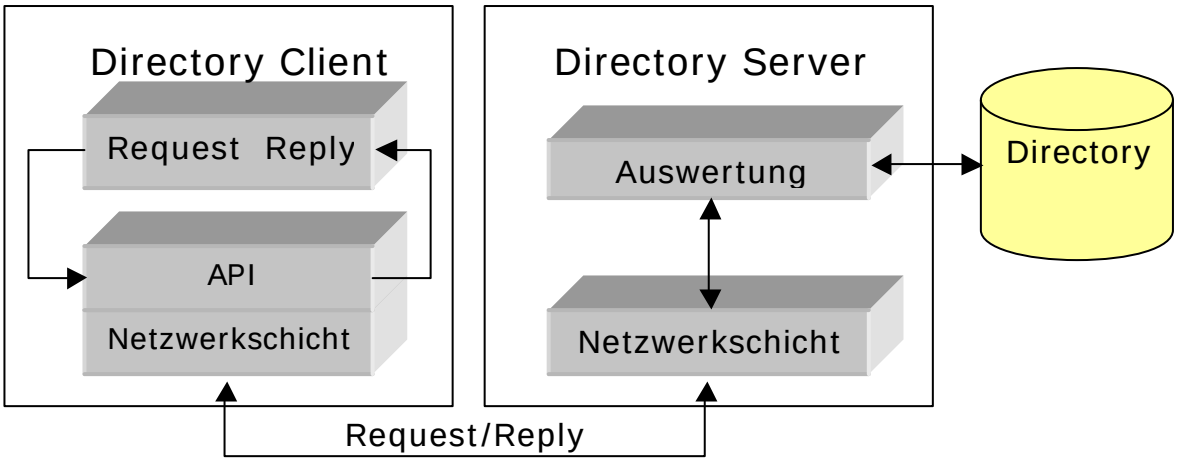

Abb. 2: Architektur eines Verzeichnisdienstes

2.2 X.500 – "The Directory"

Als „Urvater" aller modernen Verzeichnisdienste kann man ruhigen Gewissens X.500 bezeichnen. Dieses Kapitel soll die Geschichte der Standardisierung schildern, sowie die grundlegenden Konzepte des Standards aufzeigen. Der Einstieg wird am Besten mit einer kurzen Definition begonnen:

X.500 [Web2]: *"Ein ISO und ITU Standard, welcher festlegt, wie globale Verzeichnisse strukturiert werden sollen. X.500 Verzeichnisse sind hierarchisch angeordnet, mit verschiedenen Ebenen für jede Kategorie von Informationen. X.500 unterstützt X.400 Systeme."*

2.2.1 Die X.500 Standardisierung

X.500 ist das Ergebnis eines jahrelangen Bemühens, eine Standardisierung für Verzeichnisse herbeizuführen. Ursprünglich arbeiteten 3 Institutionen parallel an der Standardisierung, für Details sei der Leser auf [Cha/Kap.1] verwiesen, hier kurz zusammengefasst:

- CCITT (Comité Consultatif International Télégraphique et Téléphonique), die heutige ITU-T (International Telecommunication Union - Telecommunication Standardisation Bureau [Itu1]) arbeitete an einem Standard für einen „Telefonbuch-Dienst" (white page service)

- Die ISO (International Organisation for Standardization [Iso]) hatte sich als Ziel gesetzt, OSI (Open Systems Interconnection) Anwendungen einen Name-Server-Dienst zur Verfügung zustellen.

- Die ECMA (European Computer Manufacturers Association) sei nur der Vollständigkeit halber erwähnt. Sie spielte im weiteren Standardisierungsprozess keine größere Rolle mehr, ihre Interessen wurden von denen der ISO abgedeckt.

Ein Zusammenschluss zur „ISO/CCITT working group on directories“ sollte eine gemeinsame Basis für die letztendliche Standardisierung bilden.
Die Entstehungsgeschichte des X.500 Standards reicht ins Jahr 1984 zurück. In jenem Jahr begannen sowohl die CCITT, als auch die ISO unabhängig voneinander ihre ersten Schritte in der Verzeichnis-Technologie. Im April 1986 wurde der Grundstein für eine Zusammenarbeit bei einem Treffen in Melbourne gelegt. Bereits im November desselben Jahres wurde das erste ISO Draft Protocol (DP) vorgestellt. Die Zusammenarbeit brachte im Januar 1990 die offizielle CCITT Version des - auch als „X.500 ´88 Standard“ - bekannten Verzeichnisstandards, die „CCITT X.500 Blue Book Recommendations“ heraus. Ein Jahr später, im Januar 1991, erschien das dazugehörige ISO Pendant: „ISO/IEC 9594 – The Directory“. Nach dem X.500 ´88 Standard folgten mit dem X.500 1993 und dem X.500 1997 zwei weitere verabschiedete Standards. In Arbeit befindet sich derzeit noch X.500 2001, eine Vorabversion ist bereits unter [Itu1] verfügbar.
Die offiziellen X.500 ITU/ISO Standards können kostenpflichtig per Internet bezogen werden, siehe [Itu1] bzw. [Iso]. Kostenlos verfügbar sind nur einige Drafts, hier handelt es sich aber nicht um die vollständigen, verabschiedeten Standards.

Eine Gegenüberstellung der verschiedenen Bezeichnungen der ITU und ISO Versionen des X.500 ´93 Standards wird in Abb. 3 gezeigt.

Titel	ITU-T (CCITT)	ISO
Overview of Concepts, Models and Services	X.500	9594.Part 1
Models	X.501	9594.Part 2
Abstract Service Definition	X.511	9594.Part 3
Procedures for Distributed Operations	X.518	9594.Part 4
Protocol Specifications	X.519	9594.Part 5
Selected Attribute Types	X.520	9594.Part 6
Selected Object Classes	X.521	9594.Part 7
Authentication Framework	X.509	9594.Part 8
Replication	X.525	9594.Part 9

Abb. 3: "The Directory", CCITT REC. X.500.X.521, ISO/IEC Standard 9594

2.2.2 X.500 - Architektur

In den folgenden Abschnitten wird die Architektur eines X.500 Verzeichnisdienstes vorgestellt (siehe dazu auch [Fer]).

Directory Clients werden im X.500 Standard als Directory User Agents (DUA) bezeichnet. Ein Directory Server ist ein sogenannter Directory System Agents (DSA). Die Kommunikation zwischen Client und Server erfolgt über das standardisierte Directory Access Protocol (DAP). DAP ermöglicht es Klienten, den Kontakt zu einem Server herzustellen, Suchanfragen an das Verzeichnis zu senden oder auch auf einzelne Objekte des Verzeichnisses zuzugreifen.

Ein X.500 Verzeichnis kann auch aus mehreren DSAs bestehen. Diese können untereinander mittels des Directory System Protocol (DSP) kommunizieren. Dieses Protokoll dient dazu, Anfragen von Klienten beantworten zu können, wenn beispielsweise die erforderlichen Daten nicht lokal gespeichert sind. Der DSA erfährt über das DSP, auf welchem anderen DSA die erwünschten Daten gespeichert sind. In so einem Fall gibt es zwei Möglichkeiten: Eine Variante ist, den Klienten auf diesen DSA zu verweisen. Die zweite Möglichkeit ist es, selbst eine Anfrage an den entsprechenden DSA zu senden und die retournierte Antwort dem Klienten zu senden. Annähernd das gleiche Konzept kennt man vom Domain Name Service (DNS), vgl. dazu „Iterative Query“ und „Recursive Query“ in [Moc].

Replikation

X.500 Verzeichnisse sind stark skalierbar. Das Verzeichnis lässt sich durch Hinzunahme eines einzelnen oder auch mehrerer DSAs erweitern, sollten die vorhandenen DSAs überlastet sein. Daten, welche im Verzeichnis gespeichert sind, werden entweder aufgeteilt (partitioniert) oder redundant gespeichert (repliziert).

Bei der Partitionierung speichert jede DSA nur einen Teil der gesamten Verzeichnisinformation. Fällt ein DSA aus, so ist die darauf befindliche Information temporär nicht verfügbar.

Die Replikation hingegen speichert Information mehrfach, also auf mehreren DSAs gleichzeitig. Fällt ein einzelner DSA aus, so ist dennoch die gesamte Verzeichnisinformation verfügbar. Ein weiterer Vorteil dabei ist, dass durch die Replikation und die sich dadurch ergebende mehrfache Datenhaltung ein Lastausgleich einfach erfolgen kann (da nun mehrere Server um diese Daten gefragt werden können). Man darf jedoch auch die Nachteile nicht unbeachtet lassen. Replikation verbraucht unter Umständen sehr viel an Bandbreite, wenn die Daten ausgetauscht werden. Änderungen am Datenbestand werden üblicherweise auch nicht sofort repliziert. Daraus resultiert unter Umständen, dass vorübergehende Inkonsistenzen auftreten können.

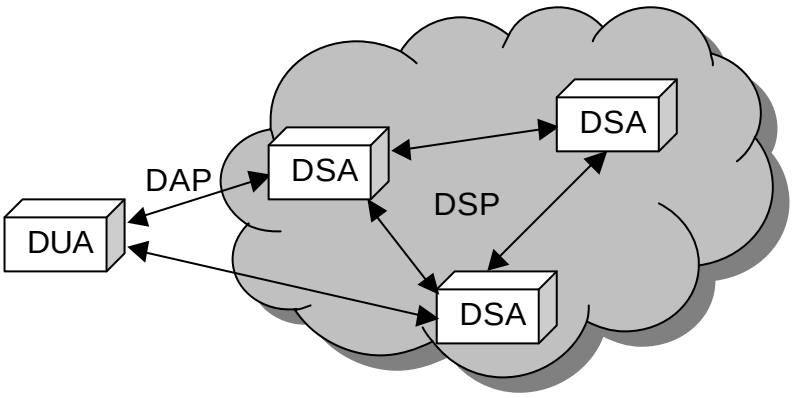

Abb. 4: X. 500 Architektur

Um das Konzept der Replikation zu ermöglichen, wurden im 1993er X.500 Standard zwei weitere Protokolle eingeführt:

- Directory Operational Binding Management Protocol (DOP)
- Directory Information Shadowing Protocol (DISP)

Die Replikation erfolgt logischerweise nicht zwischen Klient und Server, sondern nur zwischen den Servern untereinander, also von DSA zu DSA.

2.2.3 X.500 - Informationsmodell

Jeder Eintrag in einem X.500 Verzeichnis wird als Objekt bezeichnet. Jedes Objekt gehört zu einem oder mehreren Typen, den sogenannten Objektklassen. Objekte können z.B. Personen, Computer oder auch Drucker sein, mit den dazugehörigen Objektklassen „Person“, „Computer“ und „Drucker“. Für jede Objektklasse ist genau festgelegt, welche Attribute ein Objekt dieser Objektklasse beinhalten kann. Diese Attribute werden auch als Objektklassenattribute bezeichnet. Die Objektklassenattribute der Objektklasse „Person“ könnten z.B. „Vorname“, „Nachname“, „Adresse“, „Telefon“ und „E-Mail Adresse“ sein. Für ein Drucker-Objekt hätten diese Attribute wenig Sinn, die Objektklassenattribute der Objektklasse „Drucker“ könnten vielleicht so aussehen: „Name“, „Netzwerkadresse“ und „Standort“.
Jedes Objekt im Verzeichnis hat mehrere Eigenschaften, die sogenannten Attribute. Die Anzahl der Attribute wird durch die Objektklassen festgelegt, zu denen ein Objekt gehört. Attribute können je nach Definition einen Wert annehmen, oder aber auch mehrere. Beispiel: Eine Person kann über mehrere Telefonanschlüsse verfügen, es ergibt auch Sinn, diese alle zu erfassen.

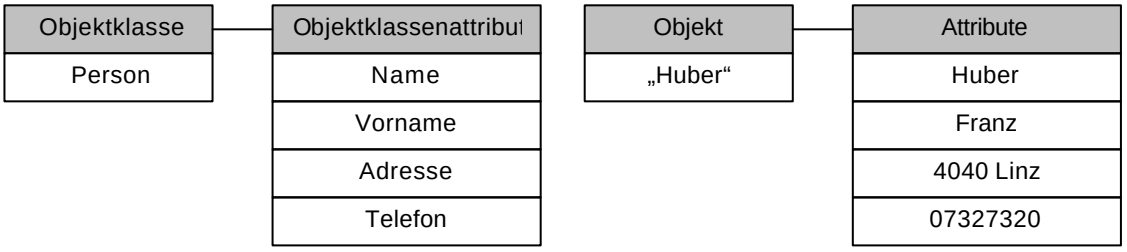

Abb. 5: Beispiel Objektklasse/Objekt

Man unterscheidet zwischen 2 Arten von Objekt-Attributen: solche, die einen Wert haben müssen (mandatory), und solche, die einen Wert haben können (optional). Wird ein Objekt angelegt, so wird geprüft, ob alle Attribute, die einen Wert haben müssen, diesen auch haben. Sollte dies nicht der Fall sein, so wird die Objektgenerierung nicht erlaubt.

Die Gesamtheit aller Informationen, die in einem Verzeichnis gespeichert ist, bezeichnet man im X.500 Standard als Directory Information Base (DIB). Die Anordnung der Objekte der DIB erfolgt in einer hierarchischen, baumartigen Struktur, dem so-

genannten Directory Information Tree (DIT). Der DIT verfügt über einen im X.500 Standard festgelegten Ausgangspunkt genannt „Root“. Alle Objekte im Verzeichnis haben diesen Knoten als ihre Wurzel.

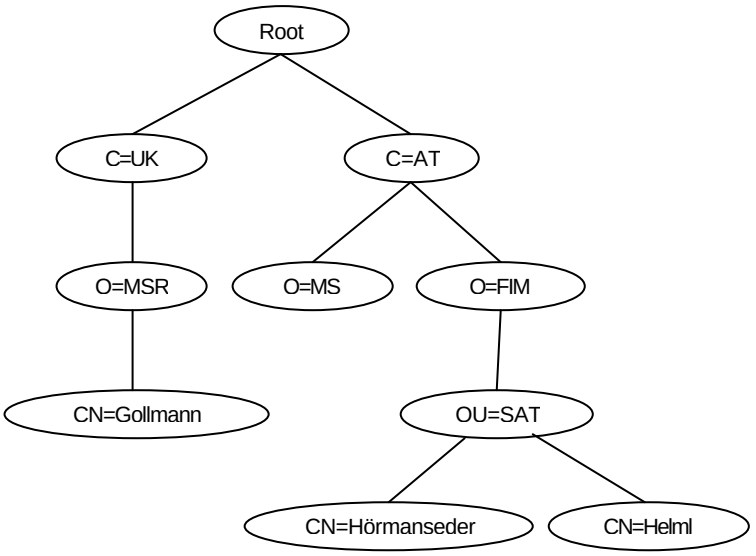

Abb. 6: Möglicher Directory Information Tree (DIT)

Anmerkung:

- C … Country
- O ... Organisation
- OU .... Organisational Unit
- CN ... Common Name)

2.2.4 X.500 - Namensmodell

Jeder Knoten im DIT hat einen Namen, den Relative Distinguished Name (RDN). Betrachten wir z.B. das Objekt „FIM“ im DIT aus Abb. 6, so ist der dazugehörige Relative Distinguished Name: „O=FIM“, das Objekt „Hörmanseder“ hat den RDN: „CN=Hörmanseder“. Vergleichen könnte man einen RDN mit einer relativen Pfadangabe unter MS- DOS, z.B.: „Windows“ oder „System32“.

Diese Namenskonvention alleine reicht jedoch noch nicht, um Objekte im DIT eindeutig zu identifizieren. Um dies zu gewährleisten bedarf es des Distinguished Name. Um den Distinguished Name eines Objekts zu bestimmen geht man wie folgt vor: Ausgehend vom Wurzelknoten „Root“ wird der kürzeste Pfad zu dem Objekt, dessen Name

man bestimmen möchte, genommen. In der selben Reihenfolge wie man den DIT durchwandert, hängt man die RDNs aller passierten Objekte aneinander. Der so entstandene Name nennt sich Distinguished Name und muss innerhalb des DITs eindeutig sein. Der DN des Objekts Hörmanseder aus Abb. 6 wäre z.B.: „C=AT;O=FIM; OU=SAT;CN=Hörmanseder“. Ein RDN kann mit einem absoluten Pfad unter MS-DOS verglichen werden, z.B.: „C:\Windows\System32\“.

2.2.5 X.500 - Funktionales Modell: Directory Access Protocol (DAP)

In den letzten Kapiteln haben wir uns mit verschiedenen Details des X.500 Standards auseinandergesetzt. Der aufmerksame Leser sollte an dieser Stelle die Grundzüge eines Verzeichnisdienstes verstehen, die Architektur des X.500 Verzeichnisdienstes ansatzweise kennen und wissen, wie Informationen in einem X.500 Verzeichnis gespeichert sind. Der wichtigste Teil fehlt jedoch: eine Möglichkeit, die gespeicherte Information abzufragen. Diese Möglichkeit bietet das X.500 Directory Access Protocol (DAP), welches in diesem Kapitel vorgestellt werden soll. Für Details sei der Leser auf die entsprechenden ITU/ISO Standards [Itu2], [Iso] sowie auf die sehr gute Referenz von D. Chadwick, [Cha] verwiesen.

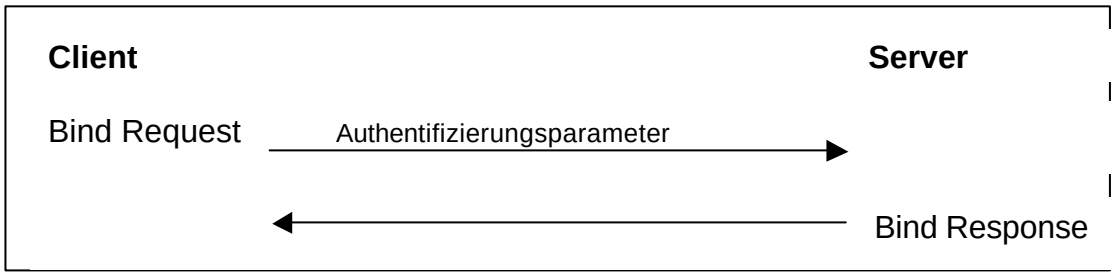

Abb. 7: DAP-Verbindungsaufbau

DAP dient als Kommunikationsprotokoll zwischen dem Klienten (DUA) und dem Verzeichnisserver (DSA). Im OSI-Referenzmodell lässt sich DAP auf Applikationsebene (Schicht 7) einordnen. DAP ist ein verbindungsorientiertes Protokoll, d.h. vor dem eigentlichen Datenaustausch muss erst eine Verbindung hergestellt werden, die während der ganzen Sitzung bestehen bleibt. Den Vorgang des Verbindungsaufbaus nennt man „binding“ (siehe dazu Abb. 7). Zusätzlich kann, je nach gewünschter Verbindung, zwischen folgenden Authentifizierungsmethoden gewählt werden [Has]:

- Keine Authentifizierung
- Einfache (simple) Authentifizierung
- Starke (strong) Authentifizierung
- Externe (external) Authentifizierung

Erst nachdem die Verbindung aufgebaut ist, kann der eigentliche Datenaustausch erfolgen.
DAP kennt insgesamt zehn verschiedene Operationen, welche man in zwei Kategorien unterteilen kann [Chap/Kap.4]:

- „Interrogation Operations“ dienen dem lesenden Zugriff auf ein X.500 Verzeichnis
- „Modification Operations“ werden zur Modifikation von Verzeichnisinhalten benötigt

Die angeführten Beispiele zu den Operationen beziehen ausnahmslos auf den DIT aus Abb. 6.

Interrogation Operations:

- **Read Operation**: Benötigt man, um Information zu einem bestimmten Verzeichnis-Objekt auszulesen, üblicherweise Objekt-Attribute.
  Beispiel: Lies das Attribut „Vorname“ aus dem Objekt: „C=AT;O=FIM; OU=SAT;CN=Hörmanseder“.

- **Compare Operation**: Dient dazu, Benutzeranfragen (Attributwerte) mit Objektattributen zu vergleichen. Die Antwort auf diese Art von Anfrage wird mit TRUE oder FALSE beantwortet. Diese Operation findet z.B. bei der Passwort-Abfrage Verwendung, das Passwort ist im Verzeichnis als Objekt-Attribut gespeichert.
  Beispiel: Vergleiche das Attribut „Vorname“ aus dem Objekt „C=AT;O=FIM; OU=SAT;CN=Hörmanseder“ mit dem Wert „Thomas“.

- **List Operation**: Benötigt man, um ausgehend von einem Objekt einen Teilbaum des DIT aufzulisten.

Beispiel: Liste die Elemente der gesamten Organisationseinheit „SAT“ mit dem Distinguished Name „C=AT;O=FIM;OU=SAT“ an.

- **Search Operation**: Die Suchoperation wird verwendet, um Objekte eines Teils des DITs zu finden, welche bestimmte Suchkriterien erfüllen.
  Beispiel: „Zeige alle Objekte im Teilbaum „C=AT;O=FIM;OU=SAT“ an, welche im Attribut „Vorname“ den Wert „Rudolf“ haben.

- **Abandon Operation**: Mit dem „Abandon“ Kommando kann eine der vorher genannten Operationen abgebrochen werden. Dies kann z.B. dann Sinn ergeben, wenn die Suchanfrage zu lange dauert.

Modification Operations:

- **AddEntry Operation**: Diese Operation hängt ein Objekt an den DIT an, wobei das neu angefügte Objekt immer „Endknoten“ (Blatt) sein muss.
  Beispiel: Hänge an das Objekt „C=AT“ ein Objekt „O=UNI-LINZ“ an.

- **RemoveEntry Operation**: Dient zum Löschen eines Objekts des DIT mit der Einschränkung, dass nur Endknoten (Blätter) gelöscht werden können. Soll ein ganzer Teilbaum gelöscht werden, so muss dies Objekt für Objekt, von „unten nach oben“, also rekursiv, geschehen.
  Beispiel: Entferne das Objekt „C=AT;O=FIM;OU=SAT;C=Helml“.

- **ModifyRDN Operation** (nur X.500 1988): Damit ist es möglich, den Relative Distinguished Name eines Blattes des DIT umzubenennen.
  Beispiel: Ändere den RDN des Objekts „Helml“ an der Stelle „C=AT;O=FIM; OU=SAT“ auf „Zarda“.

- **ModifyDN Operation** (ab X.500 1993): Ändert sowohl für Blätter als auch Nicht-Blätter eines DIT den Distinguished Name (und somit implizit den RDN auch). Damit wird es auch möglich, Teilbäume umzubenennen. Teilbäume können somit verschoben werden.

Beispiel: Ändere den DN von „C=AT;O=FIM“ auf „C=AT;O=UNI-LINZ; OU=FIM“.

- **ModifyEntry Operation**: Mit dieser Operation ist es möglich, Attribute und Attribut-Werte eines Objekts hinzuzufügen, zu löschen oder zu ändern.
  Beispiel: Ändere das Attribut „Telefonnummer“ des Objekts „C=AT;O=FIM; OU=SAT;C=Helml“ auf „+430676676034“.

2.3 LDAP

LDAP [Web3]: *“Kurzform für Lightweight Directory Access Protokoll, eine Sammlung von Protokollen zum Zugriff auf Verzeichnisse. LDAP ist eine Vereinfachung des X.500 Standards. Im Gegensatz zu X.500 benutzt LDAP TCP/IP, was für die Kommunikation über das Internet unumgänglich ist. Da LDAP eine einfachere Version von X.500 darstellt, wird es oft auch als X.500-lite bezeichnet. […] Dank der offenen Implementierung von LDAP, brauchen sich LDAP-Anwendungen nicht darum kümmern, welcher Server das Verzeichnis beinhaltet.“*

2.3.1 Die LDAP-Geschichte

Die Notwendigkeit eines neuen Zugriffsprotokolls für Verzeichnisdienste ergab sich aus einem Grund: DAP war zu schwerfällig. Internetstandards werden heute fast immer von der „Internetgemeinde“ selbst – sprich in RFCs festgelegt. Institutionen wie die ITU oder ISO haben auf jeden Fall ihre Daseinsberechtigung, oft setzen sich trotzdem andere als ihre Standards durch. Ähnlich wie DAP erging es auch dem OSI 7-Schichten Modell. Es hat zwar einen brauchbaren Ansatz, durchgesetzt hat sich aber als „das Internetprotokoll“ letztendlich TCP/IP – kein „waschechtes“ 7-Schichten Protokoll.
DAP wurde immer schon seine Schwerfälligkeit vorgehalten. Der Name des neuen Protokolls sollte daher LDAP – Lightweight Directory Access Protocol werden. Dieses „schlankere“ DAP sollte auch den Spitznamen „DAP-lite“ erhalten.
Das Standardisierungsgremium für LDAP ist nicht wie bei X.500 ITU/ISO, sondern die Internet Engineering Taskforce [Ietf]. Im Juli 1993 wurde das erste LDAP-RFC von

einem Forscherteam der University of Michigan als RFC 1487: „Lightweight Access Protocol“ veröffentlicht. Von der University of Michigan sollte auch die erste Implementierung des LDAP-Standards kommen. Dieses unter dem Namen „OpenLDAP“ bekannte Paket beinhaltet neben einem LDAP-Client auch einen Stand-alone LDAP-Server (slapd, erschienen im Dez. 1995) sowie das Frontend für einen X.500 DSA (ldapd). OpenLDAP 1.0 ist im August 1998 erschienen. Der erste kommerzielle LDAPv3 Server wurde erst im Januar 1998 verfügbar, Hersteller ist Netscape.

LDAP v2 (1995) besteht aus mehreren RFCs:

- **RFC 1777**: „Lightweight Directory Access Protokoll“: löst RFC 1487 ab, beschreibt das eigentliche LDAP-Protokoll für den Zugriff auf X.500 Verzeichnisse.
- **RFC 1778**: „The String Representation of Standard Attribute Syntaxes”: beschreibt ein Regelwerk zur Umwandlung der X.500 Attribut-Syntax in eine für LDAP brauchbare Form.
- **RFC 1959**: “An LDAP URL Format”: beschreibt die das Format für den LDAP-URL, welche als Grundlage für einen LDAP-Standalone-Server (ohne X.500) nötig ist.
- **RFC 1960**: “A String Representation of LDAP Search Filters”: definiert ein für Menschen lesbares Format um LDAP- Suchfilter anzugeben.

Im Dezember 1997 wurde RFC 2251, welcher das neue LDAP v3 definiert, publiziert. LDAP v3 soll zu LDAP v2 nicht mehr 100% kompatibel sein und dieses als Standard ablösen. Ähnlich wie LDAP v2 besteht es auch aus mehreren RFCs:

- **RFC 2251**: „Lightweight Directory Access Protocol (v3)“: beschreibt ein Zugriffsprotokoll für den lesenden und schreibenden Zugriff auf Verzeichnisse, welchem die X.500 Modelle zugrunde liegen.
- **RFC 2252**: “Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions”: definiert Attribut-Syntaxen und Regeln für die Repräsentation von Attributwerten für das LDAPv3 Protokoll.
- **RFC 2253**: “Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names”: definiert die Umwandlung von ASN.1 kodierten X.500-Distinguished Names in das von LDAP verwendete Format.

- **RFC 2254**: "The String Representation of LDAP Search Filters": definiert ein für Menschen lesbares Format für LDAPv3-Suchfilter.
- **RFC 2255**: "The LDAP URL Format": beschreibt ein Format für LDAPv3-URLs.
- **RFC 2256**: "A Summary of the X.500(96) User Schema for use with LDAPv3": bietet einen Überblick über die wichtigsten Attributtypen und Objekt-Klassen-definitionen und stellt somit eine Basis für ein mögliches LDAP-Schema dar.

Eine Programmierschnittstelle sollte auch in einem RFC spezifiziert werden - RFC 1823: „The LDAP Application Program Interface". RFC 1823 spezifiziert ein API für die Programmiersprache C. Dieses gilt allerdings nur für LDAPv2 – eine LDAPv3 Version ist bis zum heutigen Zeitpunkt nicht verfügbar. Lediglich einige Drafts wurden bereits publiziert.

2.3.2 LDAP: Protokoll, Verzeichnis oder Server?

Wie bereits in den vergangenen Kapiteln beschrieben ist X.500 ein Verzeichnis mit DAP als zugehörigem Zugriffsprotokoll. Bei LDAP sieht die Sache ein wenig anders aus. Die einen sprechen vom Protokoll LDAP, die anderen vom LDAP Server und Dritte gar vom LDAP Verzeichnis. Dieses Kapitel soll Klarheit schaffen, warum diese Definitionen für LDAP alle zutreffend sind.

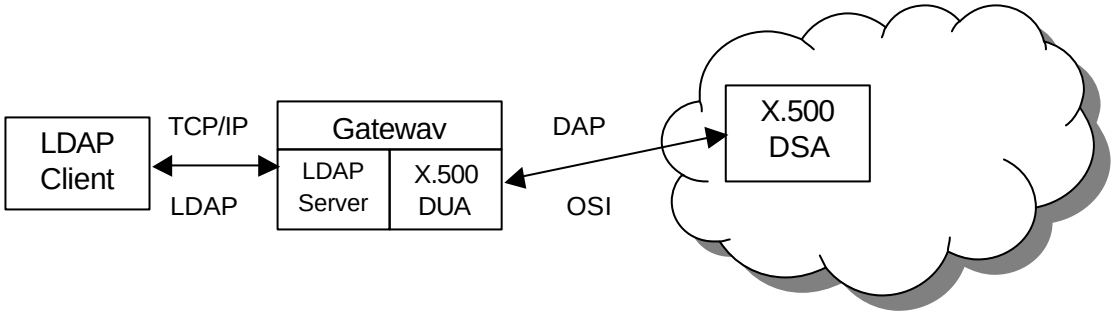

Abb. 8: LDAP Gateway

LDAP, das Protokoll

In den RFCs, in welchen LDAP definiert wird, ist vom Zugriffsprotokoll LDAP die Rede. LDAP sollte ursprünglich DAP als neuen Standard für den Zugriff auf X.500 Verzeichnisse ablösen.

LDAP, der Server

Zwischen LDAP-Klienten und X.500 Verzeichnis schaltet man eine Art „Umsetzer", ein sogenanntes Gateway, dazwischen. Dieses Gateway (siehe Abb. 8) wird auch als LDAP-Server bezeichnet. Es erfüllt zwei Funktionen:

- Das Gateway ist LDAP-Server für Anfragen des LDAP-Klienten.
- Zugleich ist das Gateway aber ein X.500 DUA, welcher Anfragen auf DAP „konvertiert" und an den X.500 Server weiterleitet.

LDAP, das Verzeichnis

LDAP kann man dann als Verzeichnis/-dienst bezeichnen, wenn X.500 komplett weggelassen wird. Ein LDAP-Server kann direkt auf die Information im Verzeichnis zugreifen, dies muss nicht mehr über den Umweg DAP-X.500 geschehen.
Implementierungen, welche ohne X.500 System im Hintergrund auskommen, findet man bereits mehrere am Markt, z.B.:

- OpenLDAP
- Netscape Directory Server
- Microsofts Active Directory Services

Fazit

Der ursprünglich geplante Einsatz von LDAP als Protokoll für den Zugriff auf X.500 Verzeichnisse brachte nicht nur eine Menge Probleme, sondern auch Vorteile mit sich. Der größte Vorteil war sicherlich die hohe Verbreitung von X.500 Verzeichnissen, sowie die „schlanken" LDAP-Klienten. Um auf ein X.500 Verzeichnis zugreifen zu können, musste man jedoch den Umweg über ein Gateway nehmen. Dies brachte zwei Probleme mit sich:

1) X.500 DSAs verstehen nur DAP, LDAP hingegen nicht.
2) X.500 basiert auf eine Kommunikation nach dem OSI-Referenzmodell, LDAP hingegen verwendet TCP/IP zur Datenübertragung.

Sinnvoll erschien es, X.500 wegzulassen und einen eigenen Datenbestand zu schaffen. Somit wurde der Weg für das LDAP-Verzeichnis gelegt. Vorteil der ganzen Sache war, dass die Kommunikation direkt über TCP/IP läuft und Anfragen an einen Verzeichnis-Server somit schneller ausgeführt werden konnten.

2.3.3 LDAP-Architektur

Die Architektur eines LDAP-Systems ist im wesentlichen seinem Vorbild X.500 nachempfunden (siehe Abb. 9). LDAP basiert wie X.500 auch auf dem Client/Servermodell. Eine Applikation (der Klient) startet eine Abfrage an das LDAP-Verzeichnis. Dazu muss der Klient eine Nachricht über das TCP/IP Netzwerk an den LDAP-Server richten. Dieser bearbeitet die Anfrage und leitet das Ergebnis zurück an den Klienten. LDAP-Verzeichnisse eignen sich auch hervorragend für den Zugriff über eine Web-Schnittstelle, mitunter ein Grund für die Popularität von LDAP.

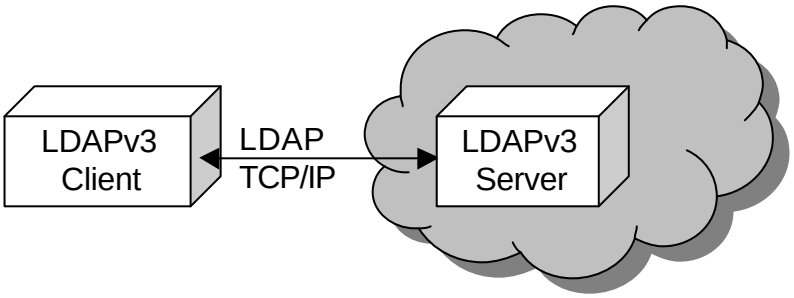

Abb. 9: LDAP-Architektur

LDAP basiert auf den Modellen [Ldap/S. 24]:

- Informationsmodell
- Namensmodell
- Funktionales Modell
- Sicherheitsmodell.

Jedes dieser vier Modelle wird nun in einem Kapitel beschrieben. Dabei sollen auch Unterschiede und Gemeinsamkeiten von LDAP zum X.500 Standard kurz angeführt werden.

2.3.4 Informationsmodell

Das LDAP Informationsmodell basiert auf dem Informationsmodell des X.500 Standards. Der LDAP-Verzeichnisbaum wird aus Objekten aufgebaut. Objekte bilden somit die kleinste Informationseinheit in einem Verzeichnis.

Jedes Objekt im Verzeichnis setzt sich aus einzelnen Attributen zusammen. Attribute besitzen einen „Typ“, die sogenannte Attribut-Syntax. Diese legt Datentyp fest und zusätzlich kann ein gültiger Wertebereich bestimmt werden. Es gibt Attribute, die nur einen Wert annehmen können (einwertig) oder mehrere (mehrwertig). Zusätzlich unterscheidet man zwischen Attributen, welche einen Wert haben müssen (mandatory) und solchen, die einen Wert haben können (optional).

Syntax	**Description**
bin	binary: Binärdaten
ces	case exact string: es wird zwischen Groß- und Kleinbuchstaben bei Vergleichen unterschieden
cis	case ignore string: Groß- und Kleinbuchstaben werden bei Vergleichen nicht berücksichtigt
dn	Distinguished name

Abb. 10: Ausschnitt aus möglichen LDAP-Attributen [Ldap/S. 26]

Ein kleines Beispiel soll die Beziehung Attribut-Objekt besser verdeutlichen. Der Einfachheit halber nehmen wir an, dass uns lediglich folgende Attribut-Syntaxen zur Auswahl stehen:

- String (Zeichenketten)
- Integer (Ganze Zahlen)

Des weiteren soll im folgenden ein Personen-Objekt lediglich die Attribute „Vorname“, „Nachname“, „Adresse“, „Gehalt“, „DN“ und „RDN“ besitzen.

Eine konkrete Ausprägung eines Personen-Objekts, in diesem Fall das Objekt „C=AT; O=FIM;OU=SAT;CN=Hörmanseder“ (siehe Abb. 6), könnte folgendermaßen aussehen:

Attributname	Wert
Vorname	Rudolf
Nachname	Hörmanseder
Telefonnummer	0676/xxxxxxxx 0732/2468-8438
Gehalt	kein Wert
DN	C=AT;O=FIM;OU=SAT;CN=Hörmanseder
RDN	CN=Hörmanseder

Abb. 11: Attributwerte des Objekts "Hörmanseder"

Das Schema

Dem sogenannten Schema kommt eine besondere Rolle zu: seine Aufgabe ist es festzulegen, wie die im Verzeichnis gespeicherten Daten aussehen müssen und welchen Regeln sie zu gehorchen haben. Das Schema bildet somit als Beschreibung der Daten die Meta-Ebene des Verzeichnisses.

Jedes Objekt in einem Verzeichnis besitzt einen Typ, die sogenannte Objektklasse. Sie legt fest, welche Attribute ein Objekt haben kann bzw. besitzen muss. Sämtliche Objektklassen werden im Schema abgespeichert. Alle Objekte in einem Verzeichnis dürfen nur von Objektklassen aus dem Schema abgeleitet werden.
Eine einfache Objektklassen-Definition für eine - zum Beispiel aus Abb. 11 passende – Objektklasse „Person“ könnte so aussehen:

Objektklasse “Person”
Vorname
Nachname
Telefonnummer
Gehalt
DN
RDN

Abb. 12: Objektklasse "Person"

Objektklassen legen fest, welche Attribute ein Objekt bilden. Diese Attribute und deren Eigenschaften müssen ebenso definiert werden. Dafür zuständig sind die sogenannten Attributtypen, welche mit den Objektklassen das Schema bilden.

Ein Attributtyp bestimmt folgende Eigenschaften eines Attributs:

- Attributsyntax und Wertebereich
- Attribut ist einwertig oder mehrwertig
- müssen Attribute einen Wert haben (mandatory) oder können sie einen Wert haben (optional)

Die passenden Attributtypen für jene Attribute, welche in der Objektklasse „Person“ (Abb. 12) verwendet wurden, könnten folgendermaßen aussehen:

Attributname	**ein-/mehrwertig**	**optional/mandatory**	**Syntax**
Vorname	mehrwertig	mandatory	String, Länge 30
Nachname	einwertig	mandatory	String, Länge 20
Telefonnummer	mehrwertig	optional	String, Länge 50
Gehalt	einwertig	optional	Integer, >0
Distinguished Name	einwertig	mandatory	String, Länge 1024
Relative Distinguished Name	einwertig	mandatory	String, Länge 256

Abb. 13: Attributtypen

Jedem LDAP-Verzeichnis steht es frei, das Schema selbst festzulegen. Es wird aber davon ausgegangen, dass die meisten LDAP-Verzeichnisse eine gemeinsame Klassen-Basis verwenden (siehe dazu auch [Wah2] und [Wah3]).

2.3.5 Namensmodell

Das LDAP-Namensmodell definiert, wie Objekte im Verzeichnis gespeichert und organisiert werden. Dies ist fast völlig mit dem X.500 Standard identisch. Objekte werden hierarchisch im sogenannten DIT (Directory Information Tree) gespeichert. Jedes Objekt im DIT hat einen Namen, den Relative Distinguished Name, der Objekte jedoch nicht eindeutig identifiziert. Dies wird erst durch den sogenannten Distinguished

Name (DN) erreicht. Den DN erhält man, wenn man vom Wurzelknoten (root) des DIT der Reihe nach bis zum Objekt durchwandert und die RDNs aneinander reiht.

Ein DIT muss nicht auf einem Server gespeichert sein, durch Partitionierung kann ein Teilbaum auf einem (physisch) anderen Server liegen. Der Klient merkt von alledem nichts, nach außen sieht man nur ein Verzeichnis mit einem DIT. Dies wird erreicht durch sogenannte „referrals“ – Platzhalter-Objekte, die auf die „echten“ Objekte an einen anderen Server verweisen.

Abb. 14 zeigt, wie der DIT aus Abb. 6 auf zwei Server aufgeteilt werden könnte und durch ein Referral miteinander verbunden wird.

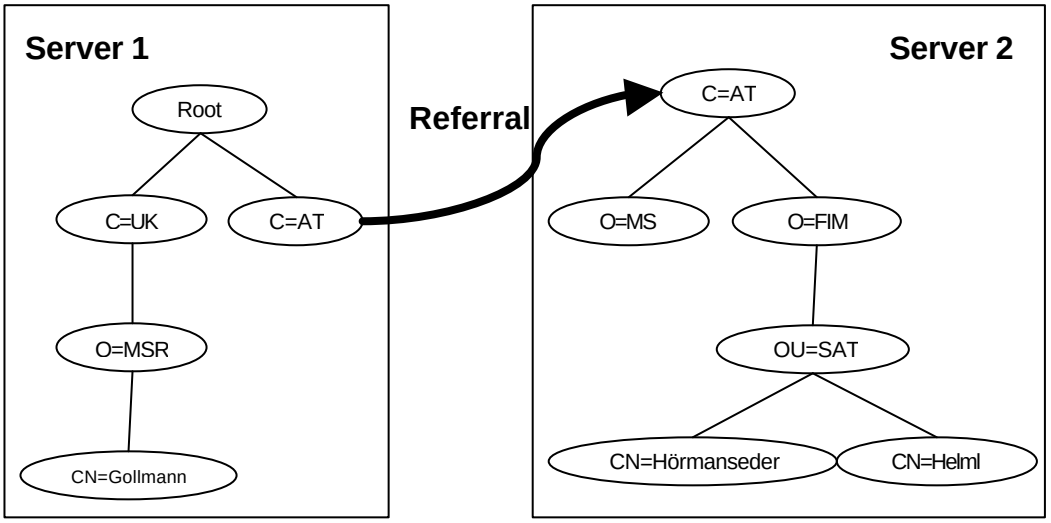

Abb. 14: Referral – Verweise auf Teilbäume

2.3.6 Funktionales Modell

Das LDAPv3 Protokoll ist sehr ähnlich wie das X.500 Pendant DAP aufgebaut, nur kommt man mit weniger Operationen aus. Auch hier ist es notwendig, vor dem Datenaustausch eine Verbindung mittels „bind“ herzustellen. Das Bind-Kommando im LDAP erlaubt nur 3 Authentifizierungs-Optionen:

- Keine Authentifizierung
- Einfache (simple) Authentifizierung
- SASL (Secure Authentication and Security Layer, [Mey]) Authentifizierung

Die LDAP-Sitzung wird, genau wie im DAP, mit der Operation “Unbind” beendet.

Eine Gegenüberstellung (Abb. 15) der DAP und LDAP Operationen sollte den Unterschied und die Reduzierung auf die notwendigen Operationen zeigen. LDAP kommt mit lediglich sieben Operationen (+eine zusätzliche) aus, DAP hat die Nase mit zehn Operationen eindeutig vorne.

Operation-Type	DAP-Operation	LDAP-Operation
Interrogation	Compare	Compare
	Abandon	Abandon
	Search	Search
	Read	
	List	
Modification	ModifyEntry	Modify
	AddEntry	Add
	RemoveEntry	Delete
	ModifyDN	Modify DN
	ModifyRDN	
Extended		Extended

Abb. 15: Gegenüberstellung LDAP - DAP

Betrachten wir die LDAP-Operationen genauer, so sollte es dem Leser selbst möglich sein, sich ein Bild über die Verbesserung des LDAP-Protokolls zu machen (siehe dazu auch RFC 2251 [Wah1]). Alle Beispiele beziehen sich auf den DIT aus Abb. 6.

Interrogation Operations:

- **Search Operation**: Search ist die mächtigste Abfragefunktion des LDAP-Protokolls. Sie vereint die DAP Operationen Search, Read und List. Es ist somit möglich, eine Suche durchzuführen auf:
    - Attribute von einzelnen Objekten
    - alle direkten Söhne eines bestimmten Objekts
    - auf ganze Teilbäume

  Beispiel: Alle Objekte, welche im Attribut „Gehalt“ einen Wert größer als „€2.000.-„ haben und die sich im Teilbaum „C=AT;O=FIM;OU=SAT;“ befinden.

- **Compare Operation**: Diese Operation entspricht der DAP Compare-Operation. Sie vergleicht einzelne Attribute eines Objekts mit einer vom Benutzer festgelegten Vorgabe.

- **Abandon Operation**: Führt zum Abbruch der zuletzt gestarteten Operation, selbe Funktion wie bei DAP.

Modification Operations:

- **Modify Operation**: Mit dieser Operation ist es möglich, Attribute und Attribut-Werte eines Objekts, hinzuzufügen, zu löschen oder zu ändern – identisch mit der DAP-Operation ModifyEntry.
  Beispiel: Ändere im Objekt „C=AT;O=FIM;OU=SAT;CN=Hörmanseder“ den Wert des Attributs „extraLongPassword“ auf „ichbineinextralangespasswort“.

- **Add Operation**: Fügt ein Objekt in den DIT ein. Der Vaterknoten des einzufügenden Objekts muss vorhanden sein. Im Gegensatz zu AddEntry (DAP) können auch Nicht-Blätter in den Baum eingefügt werden.
  Beispiel: Füge das Objekt „C=AT;O=UNI-LINZ“ an der Stelle „C=AT“ ein.

- **Delete Operation**: Dient zum Löschen eines Objekts des DIT mit der Einschränkung, dass nur Endknoten (Blätter) gelöscht werden können. Hat die gleiche Funktion wie RemoveEntry (DAP).

- **Modify DN Operation**: Erlaubt die Umbenennung eines Objektes. Dadurch ist es auch möglich, den Distinguished Name eines ganzen Teilbaums zu ändern, bzw. ganze Teilbäume zu verschieben. Gleiche Funktion wie ModifyDN im DAP.
  Beispiel: Benenne das Objekt „C=AT;O=FIM“ um auf „C=AT;O=UNI-LINZ; OU=FIM“. Der Teilbaum „C=AT;O=FIM“ wird somit verschoben!

Extended Operations:

- **Extended Operation**: Wurde erstmals im LDAPv3 Standard aufgenommen. Dieser Mechanismus erlaubt die Implementierung zusätzlicher Operationen, welche nicht Bestandteil der LDAP-Definition sind. Ein mögliches Beispiel wären digital signierte Funktionen, welche eine höhere Sicherheit als herkömmliche bieten würden.

2.3.7 Sicherheitsmodell

Für die Unterteilung des LDAP-Sicherheitsmodell gibt es verschiedene Ansichten. Wir beschränken uns auf eine Unterteilung in lediglich 2 Teile:

- Authentifizierung
- Autorisierung

Die Authentifizierung stellt sicher, dass eine Person (bzw. ein Computer) auch der ist für den er sich ausgibt. Auf diesen Teil wird im Standard ausführlich eingegangen.
Die Autorisierung soll sicherstellen, dass die Berechtigung für eine Operation auch vorliegt. Das Design der Autorisierung wird im Standard an die Produkthersteller delegiert.

2.3.7.1 Authentifizierung

Die Authentifizierung wurde ansatzweise im „Funktionalen Modell“ (siehe 2.3.6) besprochen. Bereits beim Verbindungsaufbau einer LDAP-Verbindung wird in der „bind“-Operation die Möglichkeit zur Authentifizierung geboten. Ein Parameter, welcher beim „bind“ übergeben wird, erlaubt die Auswahl zwischen drei verschiedenen Authentifizierungsmethoden:

Keine Authentifizierung

Der Titel „Keine Authentifizierung“ verrät bereits alles. Diese Option findet dann Verwendung, wenn Sicherheitsüberlegungen keine Rolle spielen. Bei öffentlich zugänglichen Verzeichnissen, wie z.B. öffentliche Telefonbücher, macht dies durchaus Sinn. Wünscht der Klient-Prozess keinerlei Authentifizierung, so geht der LDAP-Server automatisch von einem anonymen Zugriff aus. Der Klient erhält somit lediglich jene Rechte, welche für anonymen Zugriff erlaubt wurden.

Einfache (Basic) Authentifizierung

Der Name „Einfache Authentifizierung“ entspricht nicht ganz dem, was man sich unter einer einfachen Authentifizierung vorstellt. Treffender formuliert müsste der Name „Unsichere Authentifizierung“ heißen. Der LDAP-Klient authentifiziert sich durch Übergabe eines Distinguished Name (DN) und eines Passwortes. Der LDAP-Server sucht sich das Objekt, dessen Distinguished Name übergeben wurde und vergleicht das dort gespeicherte Passwort mit dem übertragenen. Wird eine Übereinstimmung gefunden, so ist der Klient authentifiziert. Der Haken an der Sache ist die Übertragung des DN und des Passworts selbst. Sie erfolgt üblicherweise im Klartext, die einzige „Verschlüsselung“, die man laut Standard vornehmen kann, ist die Verwendung des „Base64 encoding“ nach MIME (Multipurpose Internet Mail Extenstions, [Bor]). Dass diese Art der Kodierung keine Sicherheitsmerkmale bietet, versteht sich von selbst.

SASL (Simple Authentication and Security Layer) Authentifizierung

SASL [Mey] wurde ursprünglich entworfen, um IMAP mit einer stärkeren Authentifizierung auszustatten. Im Laufe der Zeit entwickelte es sich aber zu einem universell einsetzbaren Authentifizierungsmechanismus. SASL findet erstmals im LDAPv3 Standard Anwendung.

Der Authentifizierungsprozess erwartet auch hier – wie bei der Einfachen Authentifizierung - die Angabe eines Distinguished Name (also eines Objekts, welches das Passwort beherbergt) und eines Passwortes. Zusätzlich wird als Parameter angegeben, welche der folgenden Authentifizierungsmethoden verwendet werden soll:

- **Kerberos Version 4**, [Stei]
- **S/Key**, [Hal1
- **GSSAPI**, [Lin]
- **CRAM-MD5**, [Kle]
- **EXTERNAL**
- **ANONYMOUS**

Die beiden Methoden ANONYMOUS und EXTERNAL stellen – verglichen mit den anderen - eine Ausnahme dar, da sie kein bestimmtes Verfahren festlegen.

- ANONYMOUS:

  Ein Klient meldet sich als Anonymer Benutzer an (vergleiche dazu: „Keine Authentifizierung“ weiter oben). Der Unterschied zu „Keine Authentifizierung“ besteht darin, dass man explizit einen Anonymen Benutzer anmeldet.

- EXTERNAL:

  Ermöglicht es, andere „externe“ Authentifizierungsmethoden zu verwenden. Dies ist die meistverwendete Option, denn das am häufigsten in LDAPv3 verwendete Verfahren ist SSL (Secure Socket Layer), bzw. dessen Nachfolger TSL (Transport Security Layer).

2.3.7.2 Autorisierung

Nach erfolgter Authentifizierung ist sichergestellt, dass ein Benutzer (der Klient) der ist, für den er sich ausgibt. Dies ist der erste Schritt, um sich vor unberechtigtem Zugriff zu schützen. Der zweite Teil besteht darin, zu überprüfen, ob ein Klient auch die erforderlichen Rechte besitzt, um Operationen am LDAP-Server auszuführen. Diese Überprüfung nennt man Autorisierung. Die Autorisierung selbst wird oft mittels Zugriffskontrolllisten, genannt ACL (Access Control List), implementiert. ACLs legen fest, wer worauf welchen Zugriff erhält.

Im LDAPv3 Standard wird der Autorisierungsmechanismus nicht definiert. Es wird den Herstellern von LDAP-Servern freigestellt, die Autorisierung zu implementieren. In Kapitel 2.4.1.3 wird dazu der Autorisierungsmechanismus von Microsofts Active Directory vorgestellt.

Zum Thema Zugriffskontrolle und LDAPv3 existiert ein RFC mit dem Titel: „Access Control Requirements for LDAP“ [Sto1]. Hier wird beschrieben, welche Voraussetzungen für eine Implementierung einer LDAP-Zugriffskontrolle gegeben sein müssen.

Zum Zeitpunkt des Entstehens dieser Arbeit war lediglich ein Internet-Draft mit dem Titel „Access Control Model for LDAPv3“ in Arbeit [Sto2].

2.4 LDAP Implementierungen

Mittlerweile gibt es eine Vielzahl von LDAP-Implementierungen am Markt. Zwei Produkte sollen in diesem Kapitel vorgestellt werden: Microsofts Active Directory Services und das frei verfügbare OpenLDAP.
Das Hauptaugenmerk dieser Arbeit liegt eindeutig auf Microsofts Active Directory. Der Grund sollte den Leser nicht zu sehr verwundern: diese Arbeit entstand ja im Zuge der Entwicklung eines Security-Scanners für das Active Directory. OpenLDAP wird nur kurz im Überblick, quasi als Vergleichsstudie, vorgestellt.

2.4.1 Microsoft Active Directory Services

Die Grundkonzepte von Verzeichnisdiensten wurden in den vorausgegangen Kapitel über X.500 und LDAP bereits ausgiebig besprochen. Dieses Kapitel stellt einen Teil des Verzeichnisdienstes Active Directory Services vor. Zwei Bereiche sind für diese Arbeit besonders interessant:

- **Active Directory Schema**: Die Funktionsweise des Schemas und die dahinterliegenden Konzepte und Mechanismen werden am Beispiel des Active Directory Schemas aufgezeigt.
- **Active Directory Security**: Der LDAPv3 Standard schreibt keine konkreten Mechanismen für die Zugriffskontrolle vor. Die Konzepte, die im Active Directory Verwendung finden, werden ausführlich vorgestellt.

Diese beiden Punkte wurden deshalb ausgewählt, weil sie in Kapitel 3 – Active Directory Security – Scanner (ADS-Scanner) – noch eine wichtige Rolle spielen werden. Daher ist es unbedingt notwendig, mit diesem Tiefgang an das Thema heranzugehen.

2.4.1.1 Überblick

Das Erscheinen von Windows 2000 ist zugleich auch die Geburtsstunde von Microsofts erstem Verzeichnisdienst: Active Directory Services.
Active Directory unterstützt sowohl den LDAPv2 als auch den neueren LDAPv3 Standard. Trotz einiger nicht implementierter Details des LDAPv3-Standards kann man

es als LDAP-Verzeichnisdienst kategorisieren, wenn auch nicht zu 100% LDAPv3 kompatibel.

Microsoft verwendet zum Teil eigene Bezeichnungen für seinen Verzeichnisdienst. So etwa werden Objekte, welche Wurzelknoten für einen Teilbaum sind, als Container bezeichnet. LDAP-Server bezeichnet man im Active Directory als Domain-Controller (DC), siehe dazu [Ms/Seite 39 ff.]. Domain-Controller sind nicht ausschließlich LDAP-Server, sie erfüllen zusätzlich noch Funktionen im gleichen Kontext. Bereits unter Windows NT waren DCs im Einsatz. Um weiter abwärts kompatibel zu sein, kann die Funktionsweise eines Windows NT DCs in einem „Kompatibilitätsmodus“ unter Windows 2000 nachgebildet werden. Eine Windows 2000 Domain bezeichnet eine Gruppe von Rechnern, die über ein Netzwerk verbunden sind und sich ein zentrales Verzeichnis teilen. Jeder Domain-Controller innerhalb einer Domain speichert eine Kopie des gesamten Active Directory der Domäne.

Mehrere Windows 2000 Domains können hierarchisch zusammengeschlossen werden, dies wird als „Tree“ bezeichnet. Ein Tree entsteht, wenn man eine oder mehrere sogenannte „Child Domains“ an eine bestehende Domain, der sogenannten „Parent Domain“ anhängt. Alle Domains eines Trees teilen sich ein gemeinsames Schema.

Miteinander verbundene Trees wiederum ergeben einen „Forrest“. Forrests entstehen durch einen (möglicherweise hierarchischen) Zusammenschluss von voneinander unabhängigen Trees. Alle Trees innerhalb eines Forrests haben miteinander ein gemeinsames Schema.

2.4.1.2 Active Directory Schema

Das Schema in seinen Grundzügen wurde bereits in Kapitel 2.3.4. erklärt. Eine kurze Zusammenfassung der wichtigsten Begriffe soll als Wiederholung an dieser Stelle den Einstieg in die Problematik erleichtern.

- **Objekt**: Den kleinsten „Eintrag“ in einem Verzeichnis nennt man Objekt. Die hierarchische Anordnung der Objekte ergibt den DIT.
- **Container**: Objekt, welches Wurzelknoten für einen Teilbaum ist bzw. sein kann.
- **Objekt-Klasse**: Legt fest, aus welchen Attributen ein Objekt besteht. Jedes Objekt ist von einer bestimmten Objekt-Klasse „abgeleitet“.
- **Attribut**: Jedes Objekt setzt sich aus mehreren Attributen (Eigenschaften) zusammen. Man unterscheidet zwei Arten von Attributen: solche, die einen Wert

haben müssen (mandatory) und solchen, die einen Wert haben können (optional). Attribute können je nach Definition (Attribut-Syntax) nur einen Wert haben (single-value) oder mehrere (multi-value).

- **Attribut-Syntax**: Bestimmt, welche Werte ein Attribut mit dieser Attribut-Syntax annehmen darf.

Das Schema kann man sich am besten als eine spezielle Datenbank vorstellen, konkret ist es ein spezieller Container im Active Directory: "CN=Schema,CN=Configuration, DC=MyDomain".
Jedes Objekt dieses Containers hat dabei eine weltweit eindeutige Nummer, welche es identifiziert: den OID (Object Identifier). Einen OID kann man sich bei der IANA (Internet Assigned Numbers Authority, [Ian]) registrieren lassen. Ein OID besteht aus einer Reihe von Zahlen, die durch Punkte voneinander getrennt sind. Der OID, den man bekommt, stellt den Wurzelknoten für beliebig viele Nummern dar (durch Anhängen von Punkt-Zahl-Kombinationen). Für die Verwaltung unterhalb der zugeteilten OID ist man selbst zuständig.

Das Schema legt fest, welchen Regeln Objekte im Verzeichnis zu gehorchen haben. Im Schema finden sich sowohl Objekt-Klassen- als auch Attribut-Definitionen. Objekte im Schema haben somit nur einen von zwei möglichen Objekttypen:

- **classSchema**: Objekte dieser Klasse definieren die verschiedenen Objekt-Klassen im Verzeichnis
- **attributeSchema**: Objekte dieser Klasse definieren die Attribute, welche in Objekt-Klassen verwendet werden können

Fügt man neue Objekte des Typs classSchema bzw. attributeSchema in den Schema-Container ein, so wird das Schema dadurch erweitert. Durch Ändern der Eigenschaften eines solchen Objekts kann das Schema nachträglich angepasst werden.
Die Definition von neuen Attributen und Objekt-Klassen ist nicht ganz so einfach, es gilt eine Reihe von Regeln zu beachten. Im Überblick soll nun dieser Definitions-Prozess erläutert werden:

Attribut Syntax

Es ist nicht möglich, die Attribut-Syntaxen zu ändern – diese sind im Active Directory (bzw. auch in anderen Verzeichnisdiensten) fest verankert. Jede Attribut-Syntax wird durch einen OID eindeutig identifiziert. Active Directory verwendet zum Großteil die vom X.500 Standard definierten Attribut-Definitionen. Einige Microsoft-spezifische Definitionen (z.B. die SID-Attribut-Syntax) wurden jedoch hinzugefügt und sind somit im „Verzeichnis-Universum" einzigartig. Für eine detaillierte Beschreibung der 18 zur Verfügung stehenden Attribut-Syntaxen siehe [Kir/Seite 148 ff.].

Attribut Definition

Wie bereits bekannt ist, wird ein neues Attribut durch das Anlegen eines Objekts vom Typ „attributeSchema" im Schema Container definiert. Bei der Erstellung eines solchen Objekts müssen bestimmte Regeln eingehalten werden. Die wichtigsten sind im folgenden Absatz zusammengefasst, für eine detaillierte Beschreibung sei auf [Kir/Seite 142 ff.] verwiesen.

Jede Attribut-Definition, also jedes Schema-Objekt mit dem Typ „attributeSchema", besteht selbst aus einer Reihe von Attributen, hier ein Ausschnitt der wichtigsten:

- **attributeID**: beherbergt eine weltweit eindeutige Nummer, den OID – durch ihn wird das Attribut definiert.
- **schemaIDGUID**: 128 Bit großer Global Unique Identifier (GUID), welcher diese Klasse im Active Directory eindeutig identifiziert.
- **attributeSyntax**: bestimmt durch Angabe einer OID die Attribut-Syntax.
- **rangeLower, rangeHigher**: schränken den Wertebereich, den ein Attribut haben kann, zusätzlich zur Attribut-Syntax ein.
- **isSingleValued**: regelt, ob ein Attribut ein- oder mehrwertig ist.
- **systemOnly**: kennzeichnet Attribute, welche nur vom System (Active Directory) selbst geändert werden dürfen.

Der Relative Distinguished Name eines attributeSchema-Objekts bestimmt den Attributnamen, welcher später in Objekten angezeigt wird.

Beispiel: Das Attribut cn (common name) ist definiert durch das attributeSchema-Objekt, welches man an der Stelle: „CN=cn,CN=Schema,CN=Configuration, DC=MyDomain" findet. Der Relative Distinguished Name dieses Objekts ist bekanntlich „cn".

Objekt-Klassen-Definition

Die Klassendefinition ist ein wenig komplizierter, als die bereits vorausgegangene Attribut-Definition. In den folgenden Absätzen werden die wichtigsten Punkte, die es einzuhalten gilt, zusammengefasst - für eine detaillierte Darstellung siehe [Kir/Seite 132ff.].

So wie es für jedes Attribut einen Attribut-Typ gibt, so existiert auch für jedes Objekt im Active Directory ein Objekt-Typ – man nennt diesen auch die Objekt-Klasse eines Objektes. Die Definitionen der Objekt-Klassen befinden sich im gleichen Speicherort wie attributeSchema-Objekte: „CN=Schema,CN=Configuration,DC=MyDomain“.

Objekt-Klassen-Definitionen sind selbst als Objekte vom Typ „classSchema“ abgespeichert.

Die Identifizierung einer Objekt-Klasse geschieht durch eine Reihe von Attributen. Hier eine Auflistung der wichtigsten:

- **Relative Distinguished Name (RDN)**: Der RDN eines classSchema-Objekts bestimmt den Klassennamen.
- **governsID**: OID, welcher diese Klasse weltweit eindeutig identifiziert
- **schemaIDGUID**: 128 Bit großer Global Unique Identifier (GUID), welcher diese Klasse im Active Directory eindeutig identifiziert

Objekt-Klassen bestimmen die „Eigenschaften“ (Attribute), aus welchen jedes Objekt im Verzeichnis besteht. Zuständig dafür sind vier Attribute, welche jede Objekt-Klassen- Definition haben muss:

- **mustContain**: In diesem Attribut sind all jene Attribute, welche eine Objektinstanz dieser Klasse haben müssen. Es ist nicht zulässig, ein Objekt zu erzeugen, welches keine oder ungültige Werte in den hier aufgeführten Attributen hat. In bestehenden Objekten dürfen diese Attribute unter keinen Umständen gelöscht werden. Beispiel: mustContain für ein User-Objekt könnte z.B. sein: „Vorname“ und „Nachname“
- **mayContain**: Im Gegensatz zu jenen Attributen, welche im „mustContain“ aufgelistet sind, müssen die Attribute aus dem „mayContain“ in Objektinstanzen keinen Wert bekommen. Sie können einen Wert bekommen, dies ist jedoch optional – je nach Bedarf. Beispiel: mayContain für User-Objekte könnte sein: „Initialen“, „Titel“, etc.

- **systemMustContain**: Unterscheiden sich von „mustContain“-Attributen dadurch, dass sie nur vom System (Active Directory) geändert werden können.
- **systemMayContain**: Dürfen nur vom System geändert werden – können einen Wert bekommen, müssen dies aber nicht.

Es gibt noch eine Reihe von Attributen, aus welchen ein classSchema-Objekt besteht [Kir/Seite 137 ff.]. Wir beschränken uns auf zwei weitere (auch für den ADS-Scanner) wichtige Attribute:

- **defaultSecurityDescriptor**: Die „Standard-Security“ eines Objekts wird ebenfalls durch seine Klasse festgelegt. Das Attribut „defaultSecurityDescriptor“ wird bei der Erzeugung eines Objektes herangezogen und sein Inhalt auf das neue Objekt abgebildet. So bekommt jedes Objekt je nach seinem Objekttyp seine eigenen, passenden Standard-Sicherheitseinstellungen.
- **possSuperiors**: Dieses Attribut legt fest, in welchen Containern eine Instanz dieser Klasse gespeichert werden kann (zusätzlich existiert eine vom System verwaltete Version: „systemPossSuperiors“).

Bevor nun die verschiedenen Objekt-Klassen und deren Unterschiede betrachtet werden, müssen an dieser Stelle noch zwei Konzepte erklärt werden. Das Konzept der Vererbung sowie das Konzept der Inklusion sind beides Möglichkeiten, um Objekt-Klassen zu erweitern.

Vererbung

Im Active Directory gibt es nur eine Einfach-Vererbung, d.h. jede Klasse muss genau von einer anderen abgeleitet werden. Die Objektklasse „top“ stellt die Wurzel der Vererbungshierarchie dar. Sie besteht lediglich aus einer minimalen Anzahl von Attributen.

Bei der Vererbung geschieht nun folgendes:

- Jedes Objektklassen-Objekt im Schema hat ein Attribut „objectClass“. Bei der Vererbung wird in diesem Objekt die gesamte Vererbungshierarchie eingetragen, d.h. alle Objektklassen, von denen direkt und indirekt abgeleitet wurde.

- In die Attribute „mustContain“, „systemMustContain“, „mayContain“ und „systemMayContain“ der abgeleiteten Klasse werden die entsprechenden

„mustContain“, „sytemMustContain“, „mayContain“ und „systemMyContain“ Attribute der Basisklasse hinzugefügt. Somit muss eine abgeleitete Objektklasse über dieselben Attribute wie deren Basisklasse verfügen, es können jedoch noch zusätzliche aufgenommen werden. Eine abgeleitete Klasse ist deshalb immer eine Spezialisierung gegenüber der Basisklasse.

- Die Attribute „possSuperior“ und „systemPossSuperior“ der Basisklasse werden zu den bereits vorhandenen der abgeleiteten Klasse hinzugefügt. Dies bewirkt, dass Instanzen dieser Klasse auch in all jenen Containern gespeichert werden dürfen, wo dies Instanzen der Basisklasse erlaubt ist.

Abb. 16 zeigt eine (frei erfundene) Vererbungshierarchie ausgehend von der Klasse „top“.

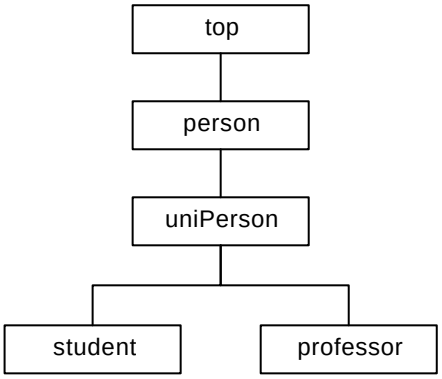

Abb. 16: Vererbungshierarchie

Ein kleines Beispiel soll nun den eigentlichen Vererbungsprozess besser verdeutlichen. Abb. 17 zeigt mögliche Attribute der verschiedenen Klassen, der Einfachheit halber werden die vom System vergebenen Attribute „systemPossSuperior“, „systemMayContain“ und „systemMustContain“ im folgenden Beispiel weggelassen.

Objektklasse	mustContain	mayContain	PossSuperior
top	DN RDN	-	-
person	Name Vorname	Adresse	OU (organisational unit)
uniPerson	UniName	UniAdresse	OU (organisational unit)
student	MatrNr SKZ	-	OU (organisational unit)
professor	Institut	Gehalt	OU (organisational unit)

Abb. 17: Objektklassen-Attribute

Das Ergebnis der Kombination der Klassen-Hierarchie aus Abb. 16 mit den Objektklassen-Attributen aus Abb. 17, sowie den bereits erläuterten Regeln für die Vererbung, wird in Abb. 18 gezeigt.

Hier wird klar deutlich, wie den einzelnen Objektklassen Attribute von ihren Basisklassen hinzugefügt (veerbt) werden.

Objektklassen-Name	objectClass	mustContain	mayContain	possSuperior
top	-	DN RDN	-	-
person	top	DN RDN Name Vorname	Adresse	OU
uniPerson	top person	DN RDN Name Vorname UniName	Adresse UniAdresse	OU
student	top person uniPerson	DN RDN Name Vorname UniName MatrNr SKZ	Adresse UniAdresse	OU
professor	top person uniPerson	DN RDN Name Vorname UniName Institut	Adresse UniAdresse Gehalt	OU

Abb. 18: Vererbungsprozess

Inklusion

Die Eigenschaften anderer Klassen können durch einen Mechanismus genannt „Inklusion“ hinzugefügt werden. Inklusion ist vergleichbar mit den sogenannten „Interfaces“ in Java, siehe dazu [Jav].

Inklusion wird durch das Eintragen einer oder auch mehreren Klasse(n) in das Attribut „auxiliaryClass“ bzw. „systemAuxiliaryClass“ des Objekttyps realisiert. Unterschiede zur Vererbung ergeben sich zwangsläufig folgende:

Das Attribut „objectClass“ wird nicht geändert, da kein Eingriff in die Vererbungshierarchie vorliegt.

Die Attribute von „mustContain“, „systemMustContain“, „mayContain“ und „sytemMustContain“ der Basisklasse werden den entsprechenden Attributen der inkludierten Klasse hinzugefügt. Dies entspricht dem Verhalten der Vererbung.

Die Attribute „possSuperior“ und „systemPossSuperior“ bleiben auch unbeeinflusst.

Abb. 19 zeigt die Definition der Objektklasse „mailEmpfänger“. Diese wird direkt von „top“ abgeleitete und beinhaltet als einziges, zusätzliches Attribut zu „DN“ und „RDN“ lediglich eine E-Mail Adresse „Email“.

Objekt-klassenname	objectClass	mustContain	mayContain	possSuperior
mailEmpfänger	top	DN RDN Email	-	-

Abb. 19: Objektklasse "mailEmpfänger"

Würden nun die Eigenschaften der Klasse „mailEmpfänger“ benötigt, hat man aber schon von einer anderen Klasse abgeleitet, so kann mittels der Inklusion hier nachgeholfen werden. Als Grundlage für ein Beispiel dient die Objektklasse „student“ (aus Abb. 18). Diese Objektklasse wurde bereits in den Vererbungsprozess involviert, d.h. es wurde direkt von der Objekklasse „uniPerson“ abgeleitet. Die Eigenschaften (Attribute) der Objektklasse „mailEmpfänger“ soll nun mittels Inklusion hinzugefügt werden. In Abb. 20 wurde dieser Vorgang bereits durchgeführt.

Objekt-klassen-name	auxiliaryClass	objectClass	mustContain	mayContain	Poss-Superior
student	mailEmpfänger	top person uniPerson	DN RDN Name Vorname UniName MatrNr SKZ Email	Adresse UniAdresse	OU

Abb. 20: Inklusion

Interessant sind vor allem die Attribute „auxiliaryClass“ und „objectClass“. Das Attribut „auxiliaryClass“ findet man in allen Objektklassen, es ist aber üblicherweise ohne Inhalt. Sobald eine Inklusion durchgeführt wird, enthält dieses Attribut alle „inkludierten“ Objektklassen. Im vorausgegangen Beispiel wurde hier die Objektklasse „mailEmpfänger“ eingetragen. Das Attribut „objectClass“ wurde hingegen nicht verändert, da es sich bei der Inklusion eben um keine Vererbung handelt.

Arten von Objekt-Klassen

Insgesamt unterscheidet man drei Arten von Objekt-Klassen: Abstract Classes, Auxilary Classes und Structural Classes. Der Unterschied wird in der Klassendefinition durch das „objectClassCategory“-Attribut festgelegt. Dieses Attribut muss in jedem classSchema-Objekt vorhanden sein, es ist daher ein mandatory Attribut in der Klasse „top“. Sehen wir uns diese verschiedenen Arten von Objekt-Klassen genauer an:

- **Abstract Classes**: Aus diesen Objektklassen können keine Objekte direkt abgeleitet werden. Es lassen sich aber sowohl Abstract Classes als auch Structural Classes (siehe unten) ableiten.

- **Auxiliary Classes**: Diese Objekt-Klassen haben eine Gemeinsamkeit mit den Abstrakten Klassen: Es können keine direkten Objekt-Instanzen dieser Klassen abgeleitet werden. Die Eigenschaften einer Auxiliary Class werden „inkludiert“. Wie bereits erwähnt gibt es keine Mehrfachvererbung im Active Directory, mit Inklusion wird dies teilweise umgangen.

- **Structural Classes**: Der Großteil der Objekte im Active Directory ist von diesen Klassen abgeleitet. Von ihnen können direkt Objekt-Instanzen abgeleitet werden. Structural Classes können selbst von einer anderen Structural Class oder auch von einer Abstract Class abgeleitet bzw. die Eigenschaften von mehreren Auxiliary Classes mittels Inklusion „hinzugefügt" werden.

Betrachtet man die Vererbungshierarchie aus Abb. 16, so könnte man folgende Zuordnung der vorkommenden Objektklassen vornehmen:

- **top**: Die Klasse „top" könnte rein theoretisch sowohl eine Abstract als auch eine Structural Class sein. Sowohl von Abstract, als auch von Structural Classes kann abgeleitet werden. Wie Abb. 16 entnommen werden kann, wird von der Klasse „top" direkt abgeleitet. Praktisch muss „top" aber eine Abstract Class sein, da sie an der Spitze jeder Vererbungshierarchie steht. Von „top" sind alle Objektklassen direkt oder auch indirekt (durch Vererbung) abgeleitet.

- **person, uniPerson**: Diese beiden Klassen könnten ebenfalls sowohl Abstract, als auch Structural Classes sein. Von beiden Klassen ausgehend wird abgeleitet, deswegen fällt die Zuordnung zur Auxiliary Class bereits weg. Zulässig wären somit sowohl sie Zuordnung zu Abstract, als auch zur Structural Class. Die Entscheidung darüber hängt vor allem davon ab, ob Verzeichnisobjekte von diesem Klassentyp gebildet werden sollen. Dann nämlich würde nur mehr die Structural Class in Frage kommen. Für den Fall, dass von diesen Klassen unter Garantie keine Objekt-Instanzen abgeleitet werden sollen, ist dies ein klassischer Fall für eine Abstract Class.

- **student, professor**: Diese beiden Klassen müssen Structural Classes sein. Beide dienen als Basis für Verzeichnisobjekte, welche im Verzeichnis abspeichert werden können.

Als Beispiel für eine Auxiliary Class sei die Klasse „mailEmpfänger" (Abb. 19) angeführt. Die Inkludierung dieser Klasse macht immer dann Sinn, wenn man ein Objekt mit den Eigenschaften eines „Mail-Empfängers" ausstatten möchte, wie dies in Abb. 20 geschehen ist.

Fazit

Das Schema ist das zentrale Herzstück jedes Verzeichnisdienstes. Unüberlegte Änderungen können weitreichende Konsequenzen haben, deshalb sollte ein Eingriff in dieses System sehr gut durchdacht sein.

Ein interessanter Punkt sei zum Abschluss noch erwähnt: Werden dem Active Directory Schema Objekte hinzugefügt, so ist das Löschen dieser Objekte nicht mehr möglich. Schema-Objekte können zwar deaktiviert werden, eine endgültige Löschung kann jedoch nur durch eine Neuinstallation erreicht werden.

2.4.1.3 Active Directory Security

Spricht man von Security im Kontext Active Directory, so muss man immer zwei Seiten betrachten, die Authentifizierung und die Autorisierung. Vom LDAPv3 Standard wird der Authentifizierungsprozess vorgeschrieben, wobei es - Dank sehr offener Implementierung - so gut wie keine Einschränkungen bei der Wahl des Authentifizierungsverfahrens gibt. Die Autorisierung hingegen wird dem Verzeichnisdienst laut LDAPv3 Spezifikation freigestellt. Für diese Arbeit hat die Autorisierung einen immensen Stellenwert, da der entwickelte Security-Scanner auf dieser Ebene seinen Dienst versieht.

Authentifizierung

Im wesentlichen finden drei Verfahren in der Active Directory Authentifizierung Verwendung:

- **NT LAN Manager (NTLM)** : NTLM Authentifizierung ist bereits von Windows NT her bekannt, es wird dort als primärer Authentifizierungsmechanismus verwendet. Windows 2000 kennt einen Kompatibilitätsmodus zu Windows NT, genannt „Mixed-Mode“. In diesem Modus kann nur eine Authentifizierung verwendet werden, welche abwärtskompatibel zu Windows NT ist.

- **Kerberos v5**: Sobald das Active Directory auf den reinen Windows 2000-Modus („Native Mode“) umgestellt wurde, steht Kerberos v5 als Authentifizierungsmethode zur Verfügung. Kerberos v5 stellt zusätzlich zur „normalen“ Authentifizierung mittels Passwort eine alternative Methode vor.

Mit sogenannten „Smart Cards“ - auch bekannt als „Chipkarte“ – und geeignetem Kartenlesegerät kann ein Benutzer auch durch die Daten in seiner Smart Card identifiziert werden.

- **Secure Sockets Layer (SSL)**: Für eine externe Authentifizierung über das Internet bietet das Active Directory noch die Möglichkeit von SSL an.

Autorisierung

Wie bereits in der Einleitung erwähnt, wird im LDAPv3 Standard die Implementierung der Autorisierung völlig dem Verzeichnisdienst-Hersteller überlassen. Üblicherweise wird die Autorisierung mittels Zugriffskontrolllisten (Access Control List, kurz ACL) realisiert. Die Funktionsweise der Autorisierung im Active Directory ist nun Gegenstand einer genaueren Untersuchung.

Security Identifier (SID)

Subjekte, welche auf geschützte Bereiche zugreifen können, nennt man „Security Principals“. Als Beispiele für Security Principals seien User, Computer und Gruppen erwähnt. Windows 2000 generiert für jedes dieser Security Principals eine eigene, systemweit eindeutige Nummer, den SID (siehe [Ms/Seite 644 ff.]). Ein solcher SID berechnet sich aus Server- und Domain-IDs. Ein möglicher SID könnte z.B. so aussehen: S-1-5-21-4352345-3453245-124.

Nach der erfolgreichen Authentifizierung wird dem authentifizierten Prozess (kann auch ein User sein) ein sogenanntes Access Token zugewiesen. Dieses Token enthält u.a. den eindeutigen SID des Besitzers.

Global Unique Identifier (GUID)

Jedes Objekt im Active Directory ist durch einen eindeutige ID - dem GUID - identifiziert. Der GUID wird dem Objekt bei seiner Erzeugung zugewiesen (jedes Objekt im Active Directory hat ein Attribut „objectGUID“, dort wird der GUID gespeichert). Solange das Objekt existiert, bleibt auch der GUID bestehen.

Access Mask

Die 32-Bit große Access Mask definiert die möglichen Berechtigungen, die auf ein Objekt vergeben werden können. Soll eine Operation ausgeführt werden, so wird an-

hand dieser Bit-Maske überprüft, ob die Operation durchgeführt werden darf oder nicht. Abb. 21 zeigt die Aufteilung der Access Mask. Eine genaue Beschreibung jedes einzelnen Bits wird hier dem Leser vorenthalten, er sei an dieser Stelle auf [Kir/Seite 61 ff.] verwiesen.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Generic Rights				Reserved		Maximum Rights	SACL Rights	Standard Rights								Object-Specific Rights															

Abb. 21: Windows 2000 Access Mask

Access Control Entry (ACE)

Eine ACE ist eine einfache Datenstruktur. Sie beschreibt im wesentlichen:

Wer hat **worauf welche** Berechtigung(en)?

Für jedes Objekt im Active Directory kann man eine oder mehrere ACEs definieren. Diese regeln dann den Zugriff auf das Objekt.

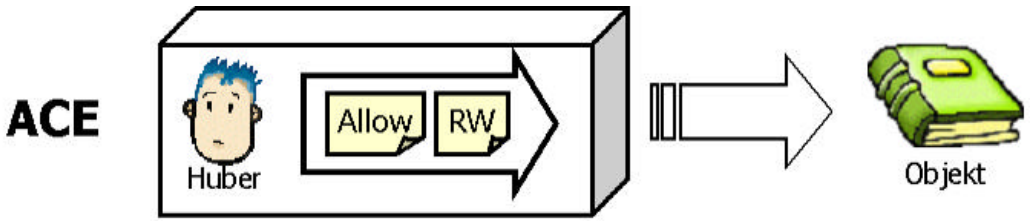

Abb. 22: Access Control Entry

Abb. 22 zeigt eine sehr vereinfachte Darstellung einer ACE, bringt die grundlegende Funktionsweise aber genau auf den Punkt. Ein User namens „Huber“ erhält schreibenden (W=write) und lesenden (R=read) Zugriff auf ein Objekt.

Die tatsächliche Windows 2000 Implementierung ist eine Spur komplexer. Abb. 23 zeigt den vollständigen Inhalt einer ACE.

	Bytes
ACE Type	1
ACE Flags	1
ACE Size	2
Access Mask	4
Trustee SID	?

Abb. 23: Allgemeine ACE

Eine ACE dient unter Windows 2000 nicht nur der Vergabe von Berechtigungen. Mittels ACEs können auch Objekte definiert werden, die besonders „beobachtet“ werden, d.h. der Zugriff auf diese Objekte wird mitprotokolliert. Man nennt diesen Vorgang auch Auditing. Auf die speziellen Teile der ACE, welche für das Auditing notwendig sind, wird hier nicht näher eingegangen. Für das Projekt SAT spielen diese Teile keine Rolle, sie werden nur der Vollständigkeit halber erwähnt.

Trustee SID: Dieser SID gibt an, für wen der Zugriff ermöglicht wird. Im vorausgegangen Beispiel (Abb. 22) würde hier der SID des Users „Huber“ stehen. Die Länge eines SIDs kann variieren, sie hängt u.a. von der Anzahl der Sub-Domains ab in denen der SID generiert wurde. Als Trustee bezeichnet man die Person bzw. das Objekt, zu dem der Trustee SID gehört.
Bei der eigentlichen Überprüfung, ob eine Operation durchgeführt werden kann oder nicht, werden der Trustee SID aus dem ACE und der SID aus dem Access Token verglichen. Besteht eine Übereinstimmung, so wird die Erlaubnis erteilt, die entsprechende Operation durchzuführen.

Access Mask: Die Access Mask wurde bereits vorgestellt. Sie ist eine 32-Bit große Maske, welche die eigentlichen Berechtigungen, d.h. die möglichen Operationen definiert. Sie bestimmen, welcher Zugriff gewährt wird.

Ace Type: man unterscheidet drei ACE-Typen (eine davon für das Auditing):

- **ADS_ACETYPE_ACCESS_ALLOWED**: Dieser Typ erlaubt es dem Trustee, die durch die Access Mask definierte Operation auszuführen.
- **ADS_ACETYPE_ACCESS_DENIED**: Will man einem Trustee explizit den Zugriff verbieten, so muss man diesen ACE-Typ wählen. Verboten wird jene Operation, die durch die Access Mask definiert wird.
- **ADS_ACETYPE_SYSTEM_AUDIT**: Auditing

ACE Size: gibt die Größe der ACE-Datenstruktur in Bytes an.

ACE Flags: Sieben verschiedene Flags (zwei davon für das Auditing) ermöglichen es, Berechtigungen zu „vererben“. Der Vererbungsmechanismus erlaubt es, Be-

rechtigungen in der Verzeichnishierarchie nach unten weiterzureichen. Dies kann u.U. die Administration erheblich erleichtern.

- **ADS_ACEFLAG_INHERIT_ACE**: Alle in der Hierarchie darunter liegenden Objekte "erben" diese ACE. Die Vererbung kann an bestimmten Stellen (absichtlich) unterbrochen werden (siehe Security Descriptor weiter unten).
- **ADS_ACEFLAG_NO_PROPAGATE_INHERIT_ACE**: Dieses Flag ist nur in Verbindung mit ADS_ACEFLAG_INHERIT_ACE gültig. Die Vererbung bezieht sich dann nur auf die direkten Sohnobjekte jenes Objekts, auf welches die ACE vergeben wurde.
- **ADS_ACEFLAG_INHERIT_ONLY_ACE**: Dieses Flag legt fest, dass sich die Vererbung auf den ganzen Teilbaum unterhalb des Objekts bezieht, dem diese ACE zugewiesen wurde. Das Objekt selbst ist davon ausgenommen, es erhält die Berechtigungen der ACE nicht. Auch dieses Flag ist nur in Verbindung mit dem ADS_ACEFLAG_INHERIT_ACE gültig.
- **ADS_ACEFLAG_INHERITED_ACE**: Wird automatisch vom System vergeben, wenn die ACE durch Vererbung von „oben" entstanden ist.
- **ADS_ACEFLAG_VALID_INHERIT_FLAGS**: Wird ebenfalls vom System gesetzt, zeigt an, ob die ACE Flags korrekt gesetzt wurden.
- **ADS_ACEFLAG_SUCCESSFUL_ACCESS**: Auditing
- **ADS_ACEFLAG_FAILED_ACCESS**: Auditing

Um die Vererbung von Berechtigungen an einem konkreten Beispiel zu sehen, soll als Ausgangsbasis wieder der DIT aus Abb. 6 dienen. Zusätzlich werden folgende Annahmen getroffen:

- SID für „Zarda": S-1-5-21-123456
- Accessmask für Lesezugriff: „0000 0001"

Drei Aufgabenstellungen zeigen sämtliche verschiedene Möglichkeiten, die Vererbung von Berechtigungen anzuwenden:

1) User „Zarda“ soll Leserechte in der gesamten Organisation „FIM“ erhalten. Folgende ACE wird dazu zum Objekt „C=AT;O=FIM;“ gespeichert:

Trustee SID	S-1-5-21-123456
Access Mask	0000 0001
ACE-Type	ADS_ACETYPE_ACCESS_ALLOWED
ACE-Flags	ADS_ACEFLAG_INHERIT_ACE

2) User „Zarda“ soll Leserechte in der gesamten Organisation „FIM“ erhalten, in dem Organisationsobjekt „FIM“ selbst aber nicht. Folgende ACE wird dazu zum Objekt „C=AT;O=FIM;“ gespeichert:

Trustee SID	S-1-5-21-123456
Access Mask	0000 0001
ACE-Type	ADS_ACETYPE_ACCESS_ALLOWED
ACE-Flags	ADS_ACEFLAG_INHERIT_ACE ADS_ACEFLAG_INHERIT_ONLY_ACE

3) User „Zarda“ soll Leserechte in „AT“ erhalten. Die Berechtigung soll jedoch nur eine Hierarchieebene nach unten reichen (in dem Fall, nur auf die Organisationen „SAT“ und „MS“). Folgende ACE wird dazu zum Objekt „C=AT;“ gespeichert:

Trustee SID	S-1-5-21-123456
Access Mask	0000 0001
ACE-Type	ADS_ACETYPE_ACCESS_ALLOWED
ACE-Flags	ADS_ACEFLAG_INHERIT_ACE ADS_ACEFLAG_NO_PROPAGATE_INHERIT_ACE

Access Control Entry (ACE) – Teil 2

Auf den letzten Seiten wurde der „Standard-ACE“ vorgestellt. Dieser findet auch im restlichen Windows-Sicherheitssystem Verwendung. Zusätzlich zu dieser ACE-Struktur gibt es noch eine zweite, erweiterte Version. Anhand eines Beispiels soll auf die Mängel, die unser bisheriges System hatte, aufmerksam gemacht werden:

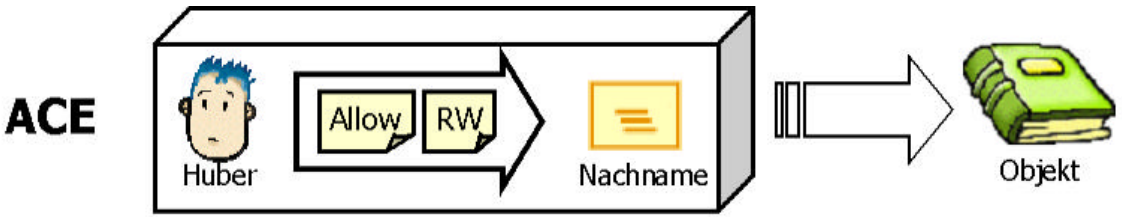

Abb. 24: Objektspezifische ACE

Abb. 24 zeigt ein Szenario, welches wir mit unserer bisherigen ACE-Definition nicht lösen können. Mit der vorgestellten „Standard"-ACE können wir nur Berechtigungen auf ganze Objekte vergeben. Besser wäre es jedoch, einem User auf dem ganzen Objekt Leserechte zu gewähren, aber schreiben soll er nur einen Teil der Attribute dürfen – wie z.B. seinen Vor- und Nachnamen. Das zweite Problem, welches sich zwangsläufig ergibt, ist folgendes: Der Vererbungsprozess wirkt sich – vereinfacht gesagt - immer auf alle Objekte eines Teilbaums aus, welche sich in der Hierarchie unterhalb des Objekts befinden. Nehmen wird an, die Vererbung würde für unser Beispiel aus Abb. 24 eingeschaltet. Die Konsequenz wäre, dass alle Objekte, die sich im Teilbaum unterhalb des Objekts, für das wir die ACE betrachten, die Berechtigung bekommen: „Erlaube Lesen und Schreiben auf das Attribut Nachname". Für User-Objekte macht dies Sinn. Diese Berechtigung auf ein Drucker-Objekt anzuwenden ist jedoch unsinnig. Deswegen sollte es auch möglich sein, den Vererbungsprozess auf eine Objekt-Klasse, d.h. auf bestimmte Objekttypen, einzuschränken. Um dies zu realisieren bedarf es einer Erweiterung der ursprünglichen ACE-Datenstruktur (Abb. 25).

	Bytes
ACE Type	1
ACE Flags	1
ACE Size	2
Access Mask	4
Object ACE Flags	2
Object GUID	16
Inherited Object GUID	16
Trustee SID	?

Abb. 25: Objektspezifische ACE

Die Teile: “ACE Flags”, „ACE Size“, „Access Mask“ und „Trustee SID“ bleiben völlig unverändert gegenüber der herkömmlichen ACE-Strukur. Eine zusätzliche Unterscheidung wird im Feld „ACE Type“ getroffen. Neu hinzugekommen sind die Einträge „Object ACE Flags“, „Object GUID“ und „Inherited Object GUID“.

ACE Type

Da wir es hier mit neuen ACE-Typen zu tun haben, wurde das Feld ACE Type um folgende neuen Möglichkeiten ergänzt:

- **ADS_ACETYPE_ACCESS_ALLOWED_OBJECT**: Diese ACE gewährt Zugriff auf jenen Teil des Objektes, welcher durch das Feld „Object GUID“ bzw. „Inherited Object GUID“ identifiziert ist.
- **ADS_ACETYPE_ACCESS_DENIED_OBJECT**: Die ACE verbietet den durch “Object GUID” bzw. “Inherited Object GUID” identifizierten Teil.
- **ADS_ACETYPE_SYSTEM_AUDIT_OBJECT**: Auditing

Object ACE Flags

Mit “Object GUID” und „Inherited Object GUID“ stehen nunmehr zwei GUID-Einträge zur Verfügung. Da diese nicht immer beide benötigt werden, regelt „Object ACE Flags“ ob „Object GUID“ bzw. „Inherited Object GUID“, oder sogar beide einen Wert beinhalten. „Object ACE Flags“ kann die folgende Werte annehmen:

- **ADS_FLAG_OBJECT_TYPE_PRESENT**: Signalisiert, dass ein GUID im “Object GUID” Eintrag vorliegt.
- **ADS_FLAG_INHERITED_OBJECT_TYPE_PRESENT**: Zeigt an, dass der GUID im Feld “Inherited Object GUID” einen Wert enthält.

Object GUID

Dieser Eintrag beinhaltet lediglich einen GUID, also einen Verweis auf ein anderes Objekt. Wie bereits erwähnt hat jedes Objekt im Verzeichnis ein Attribut, welches den GUID beinhaltet. Dieser identifiziert es im System eindeutig. Je nachdem, auf welches Objekt dieser GUID verweist, ergibt sich daraus eine andere Bedeutung. Der GUID kann sich auf folgendes beziehen:

- **Attribute**: Der GUID verweist auf die Attributdefinition im Schema, d.h. auf ein attributeSchema-Objekt. Auf dieses Attribut des Objektes kann dann Lese- oder Schreib-Erlaubnis gewährt oder verboten werden.
  Beispiel: Zugriff auf das Attribut „Nachname“ eines Objekts soll gewährt werden. Jener GUID, welcher in „Objekt GUID“ eingetragen wird, verweist auf das attributeSchema-Objekt „Surname“ aus dem Schema.

- **Objekt-Klassen** (Objekttypen): Wird ein GUID angegeben, welcher auf eine Objekt-Klasse aus dem Schema (also ein classSchema-Objekt) verweist, so wird es damit erlaubt/verboten, Objekte dieses Typs im Container anzulegen.

- **Extended Rights**: Um diesen Punkt zu erklären muss ein bisschen weiter ausgeholt werden. Das Active Directory ist so konzipiert, dass es erweiterbar ist (siehe Schema). Anwendungen können sich durch die Verwendung von sogenannten „Extended Rights“ das Active Directory Sicherheits-System zu Nutze machen. „Extended Rights“ werden in einem speziellen Container abgespeichert („CN=Extended-Rights,CN=Configuration,DC=MyDomain“) und beinhalten zwei spezielle Attribute:

  1) Einen GUID, welcher auf jene Objektklasse verweist, für welches diese Berechtigung gelten darf. Damit kann das Sicherheitssystem von Windows dem Drittprogramm signalisieren, welche Operation(en) auf welches Objekt ausgeführt werden darf (dürfen).

  2) Das zweite Attribut ist ein GUID, der zur Identifizierung des eigentlichen Extended Rights dient. Dieser GUID wird in der ACE im „Object GUID“-Feld eingetragen und identifiziert das Extended Right im Container „Extended-Rights“.

Abb. 26 zeigt, wie Extended Rights in eine ACE eingebunden werden und auf eine Objektklasse verweisen.

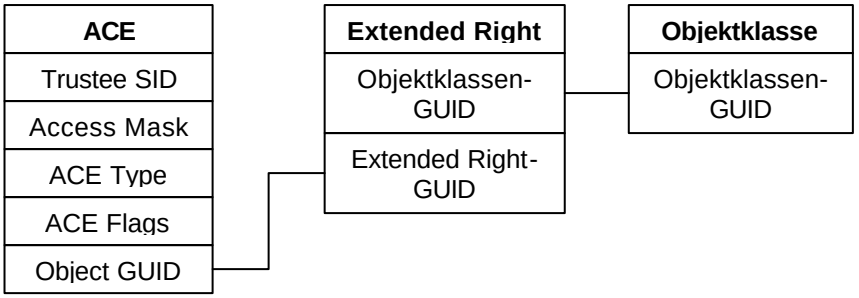

Abb. 26: Extended Right

Somit wäre es z.B. möglich, einen Terminkalender zu implementieren. Dazu wird ein Extended Right „addMeeting“ definiert, welches man nur auf Objekte der Klasse „MeetingItem“ vergeben darf. Die Objektklassen-GUID des Extended Rights „addMeeting“ müsste somit auf die Objektklasse „MeetingItem“ verweisen. Vergibt man nun Rechte für einen User, so muss innerhalb der ACE lediglich der Object GUID auf das entsprechende Extended Right verweisen (in diesem Fall wird die GUID des Extended Rights eingetragen).

Inherited Object GUID

Dieser GUID wird verwendet, um die Vererbung der Rechte nur auf bestimmte Objekttypen einzuschränken. Der Inherited Object GUID verweist auf ein classSchema-Objekt aus dem Schema.

Beispiel: Die Vererbung einer Berechtigung soll sich nur auf User-Objekte auswirken. Dazu trägt man den GUID der „User-Klasse“ aus dem Schema ein.

Access Control List (ACL)

Könnte man pro Objekt im Verzeichnis lediglich eine Berechtigung – sprich eine ACE – vergeben, so wäre dies alles andere als wünschenswert. Auf Verzeichnis-Objekte werden üblicherweise mehrere Berechtigungen, sprich ACEs, vergeben. Diese fasst man zu sogenannten Zugriffskontrolllisten (ACLs) zusammen.

Es wird zwischen zwei Arten von ACLs unterschieden:

- **Discretionary Access Control List (DACL)**: Eine DACL ist eine Liste von ACEs, in welchen die Berechtigungen für ein Objekt zusammengefasst sind. DACLs sind die für diese Arbeit interessanten ACLs.

- **System Access Control List (SACL)**: Eine SACL ist genau wie ein DACL eine Liste von ACEs. Der Unterschied besteht darin, dass im SACL jene ACEs zusammengefasst sind, welche für das Auditing zuständig sind. SACLs werden in der SAT-Security Analyse derzeit nicht berücksichtigt.

Security Descriptor (SD)

ACLs gibt es also in zwei Ausprägungen: SACL und DACL. Beide können gleichzeitig auf ein Objekt angewandt werden, deswegen bedarf es einer „größeren" Datenstruktur, die man einem Objekt zuweisen kann. Ein Security Descriptor erfüllt genau diese Bedingungen, in ihm werden SACL und DACL vereint. Zusätzlich zu DACL und SACL werden im SD der SID des Objektbesitzers sowie Kontrollflags gespeichert. Ein Security Descriptor beinhaltet somit die kompletten Sicherheitseinstellungen für ein einzelnes Objekt. Die wichtigsten der insgesamt 13 Flags seien an dieser Stelle aufgelistet:

- **ADS_SD_CONTROL_SE_DACL_PRESENT**: Dieses Flag zeigt an, ob ein DACL vorhanden ist. Dies hat unter Umständen gravierende Auswirkungen (siehe weiter unten).
- **ADS_SD_CONTROL_SE_DACL_AUTO_INHERITED**: Dieses Flag wird vom System automatisch gesetzt und signalisiert, dass ein Teil des DACL – also einige ACEs – durch Vererbung dazugekommen ist.
- **ADS_SD_CONTROL_SE_DACL_PROTECTED**: Vererbung von „oben" kommend wird an diesem Objekt nicht berücksichtigt. Die Vererbung wird an dieser Stelle unterbrochen, auch alle Objekte unterhalb sind davon betroffen.

Auf einen gravierenden Unterschied sei an dieser Stelle noch hingewiesen: Es macht sehr wohl einen Unterschied, ob eine DACL leer ist oder ob einfach kein DACL vorhanden ist (vgl. dazu SD-Flag: ADS_SD_CONTROL_SE_DACL_PRESENT). Ein fehlender DACL bedeutet, dass keine Sicherheitseinstellungen für dieses Objekt existieren, d.h. zugleich, dass jeder auf dieses Objekt mit maximalen Berechtigungen zugreifen darf. Ein leerer DACL hingegen bedeutet, dass sich der Zugriff auf die leere Menge bezieht, d.h. niemand darf auf das Objekt zugreifen.

Standard Zugriffsrechte

Nachdem nun die Funktionsweise der Zugriffskontrolle im Active Directory sehr ausführlich behandelt wurde, stellt sich nur noch die Frage, woher Objekte ihre Rechte bekommen. Die offensichtlichste Antwort wäre natürlich: explizit durch den Administrator. Daraus ließe sich aber sofort folgern, dass ein neu aufgesetztes System völlig unzureichend geschützt wäre. Daher gibt es eine Reihe von Regeln, welche die Standard Zugriffsrechte für neu angelegte Objekte bestimmen, siehe [Kir/Seite 72]:

1) Bei der Objekt-Erzeugung kann ein optionaler Security Descriptor explizit angegeben werden. Wurde dieser angegeben, so wird er auch verwendet. Zusätzlich werden vererbte Berechtigungen dazuaddiert.

2) Wurde kein Security Descriptor bei der Objekt-Generierung übergeben, so werden lediglich vererbte Berechtigungen in den Security Descriptor des Objektes eingetragen.

3) Sollten keinerlei Berechtigungen durch den Vererbungsprozess hinzugekommen sein, so wird im Schema nach dem Objekt-Typ des Objekts gesucht. In jedem classSchema-Objekt existiert ein Attribut mit dem Namen „defaultSecurityDescriptor“, welcher einen Standard-Security Descriptor enthält. Sofern vorhanden, wird dieser in den SD des Objekts kopiert.

4) Falls auch im Schema kein passender Standard-SD vorhanden ist, wird im Access Token des Objekt-Erzeugungs-Prozesses nachgesehen, ob dort ein SD vorhanden ist. Falls dieser existiert, so wird er verwendet.

5) Für den Fall der Fälle, dass sich auch im Access Token kein SD befindet, wird das Objekt ohne DACL erzeugt. Dies führt dann dazu, dass jeder mit vollem Zugriff auf dieses Objekt zugreifen kann.

2.4.2 OpenLDAP

1998 war das Gründungsjahr der Non-profit-Organisation „The OpenLDAP Foundation“ [Ope1]. Die OpenLDAP Foundation hat sich als ehrgeiziges Ziel gesetzt, stabile, frei verfügbare und voll zum LDAPv3 Standard kompatible LDAP Anwendungen (Server und Clients) und dazupassende Programmiertools zu entwickeln. Die sogenannte OpenLDAP Suite erscheint im Zuge der Open Source Initiative, d.h. der komplette Source-Code ist gratis und vollständig für jedermann zugänglich. OpenLDAP Foundation-Mitarbeiter auf der ganzen Welt arbeiten unentgeltlich am Projekt, die Kommunikation untereinander erfolgt über das Internet.

Bereits im August 1998 erschien mit OpenLDAP v1.0 einer der ersten LDAPv3- Verzeichnisdienste überhaupt. Die derzeit aktuelle Version 2.0.15 ist unter [Ope2] zu beziehen.

OpenLDAP wurde für die UNIX-/LINUX-Welt sowie deren Derivate entwickelt, mittels Anpassungen lässt es sich auch unter Windows NT/2000 kompilieren und ausführen.

2.4.2.1 OpenLDAP Schema

Das Schema eines jeden Verzeichnisdienstes ist mit einer Grundausstattung an Definitionen von Klassen und Attributen ausgestattet – für OpenLDAP trifft dies jedoch nicht zu. OpenLDAP wird mit mehreren vorkonfigurierten Schema-Auswahlmöglichkeiten ausgeliefert. Durch eine einfache (Um-)Konfiguration wird ein zusätzliches Schema oder auch mehrere Schemen in das System eingebunden. In der OpenLDAP Suite sind folgende Möglichkeiten inkludiert:

- OpenLDAP core (Basis-Schema, wird benötigt)
- Cosine and Internet X.500
- InetOrgPerson
- Assorted
- North American Directory Forum
- Network Information Services
- OpenLDAP Project

Das OpenLDAP Schema ist in einer sogenannten Schema-Datei untergebracht. In dieser Datei finden sich Attributtyp- und Klassen-Definitionen. Die Syntax dieser Definitionen

wird noch genauer betrachtet werden. Schema-Dateien befinden sich üblicherweise in UNIX/LINUX Systemen am Speicherort: „/usr/local/etc/openldap/schema“.

Das OpenLDAP Schema ist natürlich auch individuell erweiterbar. Es können sowohl neue Attributtypen als auch neue Klassen definiert werden. Im Gegensatz zum bereits erwähnten Active Directory bietet es aber einen großen Unterschied: Es kann auch eine neue Attribut-Syntax definiert werden. Dies erfordert einen „gröberen Eingriff“ in das System: Die Erweiterung muss programmiert werden, sie lässt sich nicht mit den Standard-Tools erledigen.
Der erste Schritt, um Schema-Erweiterungen vorzunehmen, besteht darin, sich einen OID bei der Internet Assigned Numbers Authority (IANA, siehe [Ian]) zu besorgen. Dieser OID stellt quasi die Wurzel jener OIDs dar, die man sich selbst daraus ableiten kann. Durch Anhängen einer Punkt-Zahl-Kombination (Beispiel-OID: 1.3.1.4.1.123) kann man selbst beliebig viele, weltweit eindeutige Nummern generieren. Jeder Attributtyp und jede Klassendefinition müssen einen eindeutigen OID haben.

Attributtyp-Definiton

Im OpenLDAP Schema müssen Attribute gemäß der Attribute Type Description (RFC 2252, [Wah2]) definiert werden. Die dazugehörige Backus-Naur-Form (BNF) sieht folgendermaßen aus [Ope2/Kap. 8.2.4]:

```
attributetype <RFC2252 Attribute Type Description>

AttributeTypeDescription = "(" whsp
        numericoid whsp              ; AttributeType identifier
      [ "NAME" qdescrs ]             ; name used in AttributeType
      [ "DESC" qdstring ]            ; description
      [ "OBSOLETE" whsp ]
      [ "SUP" woid ]                 ; derived from this other
                                     ; AttributeType
      [ "EQUALITY" woid              ; Matching Rule name
      [ "ORDERING" woid              ; Matching Rule name
      [ "SUBSTR" woid ]              ; Matching Rule name
      [ "SYNTAX" whsp noidlen whsp ] ; see section 4.3
      [ "SINGLE-VALUE" whsp ]        ; default multi-valued
      [ "COLLECTIVE" whsp ]          ; default not collective
      [ "NO-USER-MODIFICATION" whsp ]; default user modifiable
```

```
      [ "USAGE" whsp AttributeUsage ]; default userApplications
      whsp ")"

AttributeUsage =
    "userApplications"     /
    "directoryOperation"   /
    "distributedOperation" / ; DSA-shared
    "dSAOperation"          ; DSA-specific, value depends on server
```

Folgende Abkürzungen bedürfen noch zusätzlicher Erklärung:

- **whsp**: Leerzeichen
- **numericoid**: weltweit eindeutige OID
- **qdescrs**: ein oder mehrere Namen
- **woid**: entweder Name oder OID
- **noidlen**: optionale Längenangabe, Beispiel {10}

Beispiele (aus dem OpenLDAP Core.Schema):

Definition des Attributtyps „name“:

```
attributeType ( 2.5.4.41 NAME 'name'
                EQUALITY caseIgnoreMatch
                SUBSTR caseIgnoreSubstringsMatch
                SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{32768} )
```

Erklärung:

- **2.5.4.41:** Gibt einen weltweit eindeutigen OID, welcher den Attributtyp „name“ identifiziert, an.
- **NAME ‘name’:** Der Name des Attributs ist „name“
- **EQUALITY caseIgnoreMatch:** Wird dieses Attribut mit einem Wert auf Gleichheit überprüft, so sollen die Groß- und Kleinschreibung und Leerzeichen ignoriert werden.
- **SUBSTR caseIgnoreSubstringMatch:** Bei einer Suche nach einem Teil (Substring) dieses Attributs sollen Groß- und Kleinschreibung und Leerzeichen ignoriert werden.
- **SYNTAX 1.3.x.x.x. {32768}:** SYNTAX definiert die Syntax des Attributs „name“ durch Angabe eines OIDs. Dieser OID bedeutet, dass es sich um einen

Directory String handelt. „{32768}“ bedeutet, dass einem Attribut „name“ eine Zeichenkette mit der maximalen Länge 32768 zugewiesen werden darf.

```
attributeType ( 2.5.4.3 NAME
              ( 'cn' $ 'commonName' ) SUP name )
```

Erklärung:

- **2.5.4.3:** Gibt einen weltweit eindeutigen OID, welcher den Attributtyp „cn“ identifiziert, an.
- **NAME (’cn’ $ ’commonName’)**: Ein zulässiger Attribut-Namen ist sowohl “cn” als auch „commonName“.
- **SUP name**: das Attribut „cn“ wird vom Typ „name“ (siehe Definition weiter oben) „abgeleitet“, d.h. die Eigenschaften werden vom Attributtyp „name“ geerbt.

Objekt-Klassen Definiton

Die Objekt-Klassen Definition ist ebenfalls im RFC 2252 festgelegt. Hier die Definition in BNF [Ope2/Kap. 8.2.5]:

```
objectclass <RFC2252 Object Class Description>
ObjectClassDescription = "(" whsp
        numericoid whsp      ; ObjectClass identifier
        [ "NAME" qdescrs ]
        [ "DESC" qdstring ]
        [ "OBSOLETE" whsp ]
        [ "SUP" oids ]       ; Superior ObjectClasses
        [ ( "ABSTRACT" / "STRUCTURAL" / "AUXILIARY" ) whsp ]
                ; default structural
        [ "MUST" oids ]      ; AttributeTypes
        [ "MAY" oids ]       ; AttributeTypes
        whsp ")"
```

Die verwendeten Abkürzungen seien noch erwähnt:

- **whsp**: Leerzeichen
- **numericoid**: weltweit eindeutiger OID in numerischer Form
- **qdescrs**: einer oder mehrere Namen
- **oids**: ein oder mehrere OIDs

Folgendes Beispiel gibt eine Klassendefinition für die Objektklasse „student“ an (siehe Abb. 17, Abb. 18):

```
objectClass ( 1.1.2.2.3 NAME 'student'
        DESC 'Student'
        SUP uniPerson
        MUST ( 'MatrNr' $ 'SKZ' ) )
```

Erklärung:

- **1.1.2.2.3**: Gibt einen weltweit eindeutigen OID, welcher die Objektklasse „student“ identifiziert, an.
- **NAME 'student'**: Der Name der Objektklasse ist „student“.
- **DESC 'Student'**: DESC (description) gibt eine Beschreibung der erzeugten Objektklasse an.
- **SUP uniPerson**: Mit SUP (superior objectclass) wird die Basisklasse, von der abgeleitet wird, spezifiziert. Ohne zusätzliche Parameter wird eine Structural Class erzeugt.
- **MUST ('MatrNr' $ 'SKZ')**: Hier werden jene Attribute definiert, welche eine Objektinstanz der Klasse „student“ haben muss.

Fazit

Vergleicht man die Active Directory Objekt-Klassen Definition und die von OpenLDAP, so findet man doch einige Übereinstimmungen. So etwa gibt es auch im Active Directory Schema 3 Arten von Klassen: Abstract, Auxiliary und Structural.

„Must“- und „May“- Attribute bestimmen sowohl im Active Directory als auch unter OpenLDAP die Bestandteile der Objektklasse. Ob das Active Directory hier dem Standard folgt, ist fraglich. Die Attribute „systemMustContain“ und „systemMay-Contain“ kommen beispielsweise in der OpenLDAP Definition nicht vor. Ein weiterer Punkt, der sofort auffällt: Unter OpenLDAP gibt es so keine GUIDs, man kommt sehr gut mit den weltweit eindeutigen OIDs aus.

2.4.2.2 OpenLDAP Security

Den Aufbau der Zugriffskontrolle überlässt der LDAPv3 Standard völlig den Herstellern von LDAP-Software. OpenLDAPs Implementierung basiert auf relativ ein-

fachen Regeln, die dennoch sehr mächtig sind. Sehen wir uns das Regelwerk in BNF an [Ope2/Kap. 8.2.5]:

```
<access directive> ::= access to <what>
        [by <who> <access> <control>]+
<what> ::= * | [ dn[.<target style>]=<regex>]
        [filter=<ldapfilter>] [attrs=<attrlist>]
<target style> ::= regex | base | one | subtree | children
<attrlist> ::= <attr> | <attr> , <attrlist>
<attr> ::= <attrname> | entry | children
<who> ::= [* | anonymous | users | self |
       dn[.<subject style>]=<regex>]
         [dnattr=<attrname> ]
         [group[/<objectclass>[/<attrname>][.<basic style>]]=<regex> ]
         [peername[.<basic style>]=<regex>]
         [sockname[.<basic style>]=<regex>]
         [domain[.<basic style>]=<regex>]
         [sockurl[.<basic style>]=<regex>]
         [set=<setspec>]
         [aci=<attrname>]
<subject style> ::= regex | exact | base | one | subtree | children
<basic style> ::= regex | exact
<access> ::= [self]{<level>|<priv>}
<level> ::= none | auth | compare | search | read | write
<priv> ::= {=|+|-}{w|r|s|c|x}+
<control> ::= [stop | continue | break]
```

Die OpenLDAP Zugriffskontrolle wird im wesentlichen von drei Parametern bestimmt:
`access to <what> by <who> <access>`
Also: "Auf **welches** Objekt soll von **wem wie** zugegriffen werden dürfen?". Diese drei Parameter sollen nun näher besprochen werden.

<what>: Welches Objekt soll geschützt werden?

Die zu schützenden Objekte müssen näher spezifiziert werden. Dies erfolgt durch Angabe eines der folgenden Parameter:

- ***** : Alle Einträge des Verzeichnisbaums werden ausgewählt.
- **Distinguished Name**: Mit dem DN wird ein bestimmtes Objekt im Verzeichnis angegeben.

- **Filter**: Mit einem LDAP-Filter lässt sich eine Teilmenge des Verzeichnisses spezifizieren. Diese sehr effiziente Methode ermöglicht es somit, größere Bereiche mit nur einem Kommando abzudecken. Der verwendete LDAP-Filter muss dem „LDAP search filter“ aus RFC 2254 entsprechen.
- **Attribute**: Berechtigungen auf einzelne Attribute eines Objektes können mit diesem Parameter angegeben werden.

<who>: Wer soll Zugriff erhalten?

Fünf Möglichkeiten stehen zur Verfügung, um zu regeln, wer Zugriff auf ein geschütztes Objekt bekommen soll:

- *****: Alle – dies beinhaltet sowohl anonyme als auch authentifizierte Benutzer.
- **Anonymous**: Zugriff für anonyme Benutzer wird freigegeben.
- **Users**: Nur authentifizierte User erhalten Zugriff.
- **Self**: Der Zugriff wird freigegeben für den Bereich, der mit dem <what>-Parameter definiert wurde.
- **Distinguished Name**: Der DN kann sich beziehen auf Objekte, Domains, Gruppen oder auch auf Attribute, in denen ein DN gespeichert ist, welcher dann weiter verweist.

<access>: Welcher Zugriff wird gewährt?

Das Sicherheitskonzept von OpenLDAP kommt mit lediglich sechs Berechtigungen aus. Die Zugriffsberechtigungen sind geordnet, wobei das unterste die meisten, das obere die wenigsten Rechte beinhaltet. Jede Berechtigung beinhaltet alle darüber liegenden (kein Zugriff – „none“ - ausgenommen), siehe Abb. 27.

- **none**: Kein Zugriff wird erlaubt.
- **auth**: Die Berechtigung zur Authentifizierung wird gewährt.
- **compare**: Damit wird der Vergleich von Objekten erlaubt (beinhaltet das Recht „auth“).
- **search**: Die Suche im Verzeichnisbaum ist durch diese Berechtigung noch nicht freigegeben, lediglich ein Suchfilter darf angewandt werden. Zum Lesen der Suchergebnisse bedarf es einer zusätzlichen Berechtigung (siehe „read“). „Search“ inkludiert sowohl „compare“ als auch „auth“.

- **read**: Erst mit „read“ dürfen Sucherergebnisse auch eingesehen, sprich gelesen werden. „Read“ beinhaltet „search“, „compare“ und „auth“.
- **write**: Der Schreibzugriff stellt die höchste Zugriffsebene dar. Damit ist es erlaubt, Einträge zu ändern und umzubenennen. „Write“ enthält alle anderen Berechtigungen, die wären: „read“, „search“, „compare“ und „auth“.

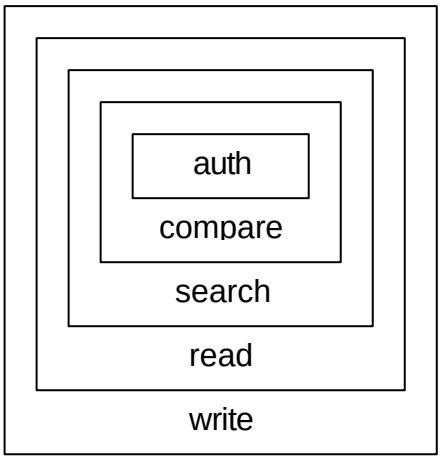

Abb. 27: Zugriffsberechtigungen

Ein paar simple Beispiele für Zugriffsberechtigungen machen die Anwendung dieses Regelwerks besser verständlich:

```
access to * by * write
```

Diese Regel erlaubt den schreibenden Zugriff auf das vollständige Verzeichnis für alle.

```
access to *
by self write
by anonymous auth
by * read
```

Diese Berechtigung erlaubt den schreibenden Zugriff für jedes Objekt des Verzeichnisses auf sich selbst, sowie den lesenden Zugriff für alle. Anonyme Benutzer bekommen die Berechtigung, sich zu authentisieren.

```
access to dn=“.*, O=FIM,C=at”
       by dn=”CN=Helml,OU=SAT,O=FIM,C=at” read
```

Diese Regel erlaubt dem User „Helml” lesenden Zugriff auf den Teilbaum „O=FIM,C=at“, wobei die Wurzel (Objekt „FIM“) selbst davon ausgenommen ist.

3 Active Directory Security - Scanner (ADS-Scanner)

Dieses Kapitel beschreibt den praktischen Teil der Arbeit: die Implementierung des ADS-Scanners im Zuge des SAT-Projektes. In den ersten Unterkapiteln wird die Architektur des ADS-Scanners erläutert, seine Einbettung in das SAT-System gezeigt, sowie das dahinterliegende Datenbankschema erklärt. Der Großteil dieses Kapitels wird jedoch die Implementierung selbst zum Inhalt haben. Probleme, die sich während der Arbeit am Projekt ergaben, werden aufgezeigt und deren Lösungen präsentiert. Dieses Kapitel soll zugleich als Nachschlagewerk für zukünftige Änderungen und Verbesserungen des Scanners dienen.

3.1 Motivation

Mit Einführung von Windows 2000 kam die wohl interessanteste Neuerung seit dem Erscheinen von Windows NT: Microsofts Active Directory Services. Die Implementierung dieses Verzeichnisdienstes, welcher Bestandteil der Windows 2000 Serverpalette wurde, sorgte weltweit für Aufsehen. Für die Administration großer Netzwerke und Serversysteme bringt diese Neuerung eine Menge an Erleichterungen mit sich. Die Benutzerverwaltung insbesondere für große Netze war bisher alles andere als einfach, und wegen globaler und lokaler Gruppen nicht gerade flexibel. Microsofts Active Directory verspricht den Zusammenschluss ganzer Firmennetze, die weltweit verteilt sein können. Das Active Directory soll Millionen von Einträgen handhaben können. Darf man der Marketingabteilung Microsofts Glauben schenken, so wird die Rolle des Active Directory künftig eine noch größere werden. Die Erweiterbarkeit des Systems ermöglicht es anderen Anwendungen, das Active Directory inklusive dessen Sicherheitssystem zu nutzen. Die Benutzerverwaltung von E-Mail-, FTP- und Datenbank-Servern könnte so in naher Zukunft an einer Stelle zentral von einem Administrator erledigt werden. All diese aufgezählten Punkte bringen ganz klare Verbesserungen und zum Teil auch eine erhöhte Sicherheit mit sich.

Alle Gründe für das Active Directory lassen sich aber auch umkehren. „Zentrale Verwaltung“ setzen viele mit „zentrales Sicherheitsloch“ gleich. In der Tat birgt die zentrale Administration ein großes Risiko: erhält ein Benutzer durch Unachtsamkeit zu viele Rechte, so können die Auswirkungen ungleich größer sein als in Systemen mit einer dezentralen Verwaltung. Ein weiterer Schwachpunkt des Systems ist – genau wie im Dateisystems NTFS – das Fehlen einer einfachen Möglichkeit, sich die Rechte eines Benutzers im System anzeigen zu lassen. Die Zugriffskontrolle des Active Directory ist sehr ähnlich wie im Dateisystem aufgebaut. Der Administrator hat lediglich die Möglichkeit, sich die Rechte auf ein einzelnes Objekt anzeigen zu lassen. Sollten man es zukünftig wirklich mit Tausenden, ja vielleicht Hunderttausenden von Einträgen zu tun haben, so kann man sich sehr leicht vorstellen, wie schwer es wird, Fehler zu erkennen. Selbst wenn man sich sicher ist, einen Fehler begangen zu haben, wird es zu einer Herausforderung, diesen zu lokalisieren und zu eliminieren.
SAT hat es sich zur Aufgabe gemacht, dazu beizutragen, diese Schwachstellen zu entschärfen. Mit SAT 2 wird das Active Directory in die Sicherheitsanalyse mit aufgenommen. Die Implementierung des dafür verantwortlichen Teils – des ADS-Scanner – wird in diesem Kapitel beschrieben.

3.2 Architektur

Das SAT-System ist so entworfen, dass auf jedem zu untersuchenden Server ein SAT-Scanner installiert wird. Der SAT-Scanner wurden als Windows-Dienst (siehe [Spe/Seite 668]) implementiert. Von einer zentralen Stelle - dem Controller Programm - aus werden die SAT-Scanner aktiviert und starten ihre Analyse.
Ein SAT-Scanner besteht aus mehreren Teilen: User-/Gruppenanalyse [Zar], NTFS- und Registry-Analyse [Ach] und der ADS-Analyse, welche im folgenden beschrieben wird. Ob ein Scanner-Teil startet oder nicht, hängt alleine von der Konfiguration im Controller-Programm ab.
Ein Punkt unterscheidet den ADS-Scanner jedoch von den anderen Teilen des SAT-Scanners. NTFS wird wahrscheinlich auf jedem Rechner im Netzwerk analysiert werden, das Active Directory wird aufgrund seiner Struktur nur auf wenigen Rechnern analysiert werden müssen.

Der ADS-Scanner unterteilt sich in 2 Programmkomponenten:

- ADS-Object Scanner
- ADS-Schema Scanner

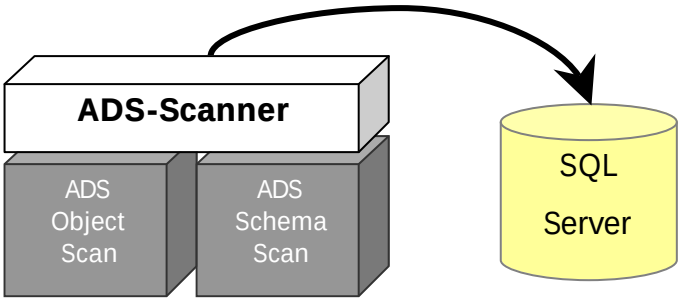

Abb. 28: Architektur ADS-Scanner

Die ADS-Object Komponente und die ADS-Schema Komponente stellen zwei unabhängige Teile dar. So ist es möglich, auf einem Server nur die Active Directory Objekte zu analysieren und auf einem anderen das Schema zu analysieren. Der Grund für diese Teilung liegt in der Natur des Active Directorys selbst. Das Verzeichnis kann auf mehrere Domain-Controller aufgeteilt werden (Partitionierung). Bei der Analyse muss auch bedacht werden, dass eine Domain aus mehreren Domain-Controllern bestehen kann. Alle DCs innerhalb einer solchen Domain speichern dabei den gleichen Teil des Active Directory (Replikation). Das Schema hingegen wird generell repliziert, d.h. es ist auf allen Domain-Controller innerhalb des Active Directory gleich. Damit wäre eine Analyse des Schemas auf jedem Domain-Controller unsinnig. Die Objekt-Analyse sollte für jeweils einen Domain-Controller pro Domain durchgeführt werden.

Die Schema-Analyse unterscheidet sich gravierend von der Objekt-Analyse. Bei der Schema-Analyse werden die „Objekt-Typen“ sowie die dazugehörige „Standard-Security“ ausgelesen. Die Objekt-Analyse hingegen speichert die Security (Zugriffskontrolllisten) für jedes Objekt. Daher kann es vorkommen, dass das Schema zweimal gescannt wird: einmal in der Schema-Analyse, um die Objekttypen zu erhalten und ein zweites Mal in der Objekt-Analyse, um den eigentlichen Security-Scan durchzuführen.
Der ADS-Scanner ist so konzipiert worden, dass die Reihenfolge - Objekt-Analyse / Schema-Analyse – keine Rolle spielt.

Die analysierten Daten wandern direkt – ohne Umweg - in die Datenbank. SAT 2 verwendet zur Zeit der Entstehung dieser Arbeit Microsofts SQL-Server 2000 für die Speicherung. SAT 1 verwendete noch eine Access-Datenbank, welche sich aber bei zunehmendem Analysevolumen als langsam erwiesen hatte.

3.3 Registry-Werte

Die Windows Registry wird als Steuereinheit für die ADS-Scanner benutzt. Jede SAT-Scanner-Instanz erfährt dadurch, welche Teile des Scanners gestartet werden müssen. Somit kann auf einzelnen Rechnern nur das NTFS analysiert werden, auf anderen das Active Directory. Die Unterteilung, welcher Teil des Active Directory (DomainNC, Configuration, Schema) untersucht werden soll, wird ebenfalls in der Registry hinterlegt.
Unter „HKEY_LOCAL_MACHINE\SOFTWARE\FIM\Sat2\“ findet man sämtliche für die Scanner notwendigen Daten.
Für den ADS-Scanner müssen folgende Werte vorhanden sein:

- **adsCompression**: gibt die Komprimierungsstufe an; zulässige Werte: 0,1,2,3; Default-Wert: 0
- **cacheLimit**: maximale Anzahl der im Cache befindlichen ACLs; Default-Wert: 300
- **creationTime**: gibt an, ob die Erzeugungszeit eines Objekts (Wert: 0) gespeichert werden soll oder der Zeitpunkt der letzten Änderung (Wert: 1); Default-Wert: 0
- **dbServer**: gibt die Netzwerkadresse des SQL-Servers, an den die analysierten Daten geschickt werden sollen, an (zulässig: Netbios-Name, DNS-Name / IP-Adresse); Default-Wert: localhost
- **scanAdsDomain**: gibt an, ob die Sicherheitsanalyse für den DomainNC-Containers gestartet (1) werden soll oder nicht (0); Default-Wert: 0
- **scanAdsConfig**: gibt an, ob die Sicherheitsanalyse für den Configuration-Containers gestartet (1) werden soll oder nicht (0); Default-Wert: 0
- **scanAdsSchemaRights**: gibt an, ob die Sicherheitsanalyse für den Schema-Containers gestartet (1) werden soll oder nicht (0); Default-Wert: 0

- **scanAdsSchema** gibt an, ob die Schema-Analyse gestartet (1) werden soll oder nicht (0); Default-Wert: 0

3.4 Datenbank

Aus der Datensicht betrachtet, läuft die SAT-Analyse in zwei Schritten ab:

1) Daten sammeln
2) Daten auswerten (und visualisieren)

Aufgabe des ADS-Scanners ist es lediglich, die ADS-Daten zu sammeln und in einer Datenbank abzuspeichern. Ein Teil der Datenaufbereitung wird durch den ADS-Scanner selbst erledigt (siehe Kapitel 3.5 Komprimierung), sodass eine strikte Trennung nicht ganz möglich ist. In diesem Kapitel wird das Datenmodell des ADS-Scanners vorgestellt, welches die Grundlage für die eigentliche Implementierung bildet.

Wie bereits mehrmals erwähnt setzt SAT 2 auf SQL-Server 2000 auf. Der Zugriff auf den Datenbankserver wurde mittels ActiveX Data Objects (ADO) realisiert. Die Implementierung der Datenbank-Zugriffsklasse sowie die Beschreibung der Schnittstelle wird im Kapitel 3.6.1.1 noch ausführlich behandelt. Dieses Kapitel beschränkt sich lediglich auf die Beschreibung der einzelnen Tabellen, in denen die Daten abgespeichert werden. Dabei wird allerdings nur auf jenen Teil eingegangen, der für den ADS-Scanner relevant ist. Die SAT 2-Datenbank beinhaltet zusätzliche Tabellen, auf die an den entsprechenden Stellen verwiesen wird.

Der ADS-Scanner kommt im wesentlichen mit sechs Tabellen aus (siehe Abb. 29):

- **acl**
- **ace**
- **membership**
- **sidref**
- **objTypes**
- **objects**

Die nächsten Seiten beschreiben detailliert die Attribute jeder Tabelle, sowie die Werte, welche darin gespeichert werden. Die in den Tabellen angeführten Datentypen ent-

sprechen den Definitionen des SQL-Servers. Sämtliche Konstanten, welche ausnahmslos mit „SAT_“ beginnen, stammen aus dem Header-File „satdefs.h“. Festgehalten sei an dieser Stelle noch, dass die Datenbank in keiner Normalform vorliegt. Das SAT-Team ist dieser Tatsache sehr wohl bewusst. Optimierungen am SAT-System könnten mit Sicherheit einige an der Datenbank vorgenommen werden. Diese wurden aber noch nicht durchgeführt und bleiben somit künftigen SAT-Versionen vorbehalten. Manche Redundanz wurde absichtlich zugunsten von Performance-Steigerungen und manchmal auch aus Gründen der Einfachheit eingebaut. So gab es z.B. die Überlegung, für SIDs und GUIDs eigene Tabellen einzuführen. Anstatt des SIDs/GUIDs könnte dann – mittels einem ID - auf den entsprechenden Eintrag in der SID-/GUID-Tabelle verwiesen werden. Auf den ersten Blick erscheint dies als die bessere, „elegantere“ Lösung. Man darf jedoch nicht vergessen, dass für jeden SID/GUID zuerst überprüft werden muss, ob dieser nicht bereits gespeichert ist. Die Reihenfolge, in der SIDs/GUIDs ausgelesen werden, lässt sich nicht vorhersagen. Dies bedeutet eine enorme Anzahl an zusätzlichen Datenbankzugriffen.

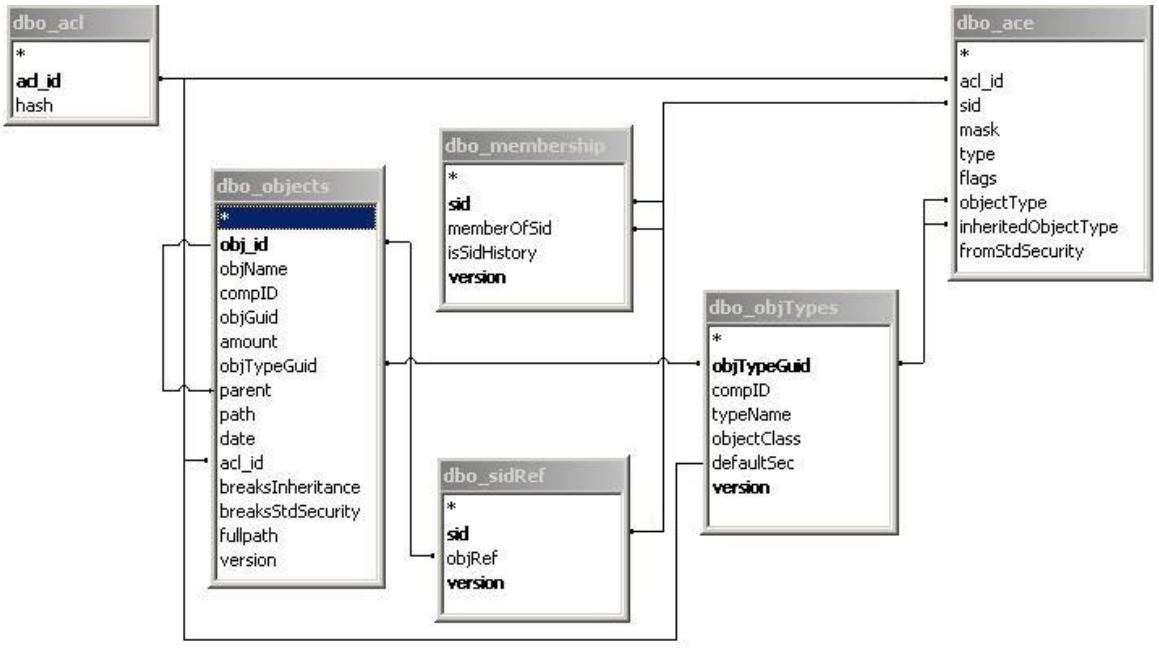

Abb. 29: Tabellen der SAT-DB, welche für den ADS-Scanner relevant sind

3.4.1 Tabelle „acl“:

acl	
acl_id	int
hash	int

Beschreibung:

ACLs werden in der Datenbank nur einmal abgelegt. Gleiche ACLs kommen relativ oft vor, daher spart es Platz, wenn diese nur einmal abgespeichert werden. Bevor eine ACL gespeichert wird, wird vorher überprüft, ob sich diese ACL nicht schon in der Datenbank befindet. Dazu wird aus der gegebenen ACL ein Hash-Wert generiert. Die Wahrscheinlichkeit, dass sich aus zwei verschiedenen ACLs der gleiche Hash-Wert berechnet, ist relativ gering. Sollte dieser Fall dennoch auftreten, so müssen alle ACLs mit gleichem Hash-Wert aus der Datenbank ausgelesen und verglichen werden. Somit kann eine doppelte Speicherung ausgeschlossen werden. Bei schlechter Wahl des Hashing-Algorithmus steigt die Anzahl der Datenbankzugriffe entsprechend an.

Abb. 30 zeigt einen Ausschnitt aus Tabelle „acl“ nach einem durchgeführten Analyselauf. (Anmerkung: Die ACL mit ID 1 ist ein Sonderfall. Diese wird von SAT generiert, wenn der ADS-Scanner nicht die entsprechenden Berechtigungen hatte um die ACL auszulesen)

	acl_id	hash
▶	1	0
	2	2098322
	3	4065424
	4	2098322
	5	3081873

Abb. 30: Ausschnitt aus Tabelle „acl“

Attribute:

- **acl_id**

  Der acl_id ist Primärschlüssel der Tabelle acl und wird automatisch von der Datenbank generiert. Die kleinste ID hat den Wert „1“, jeder weitere ID wird um eins erhöht.

- **hash**

  Aus der gesamten ACL wird ein numerischer Hash-Wert durch Addition der Access-Masks der einzelnen ACEs berechnet. Die Anzahl der Datenbank-zugriffe wird somit minimiert. Der Hash-Wert wird initialisiert mit SAT_NO_HASH.

3.4.2 Tabelle „ace“

Ace	
acl_id	int
sid	varchar(SAT_SID_LEN)
mask	tinyint
type	tinyint
flags	tinyint
objectType	varchar(SAT_GUID_LEN)
inheritedObjectType	varchar(SAT_GUID_LEN)
fromStdSecurity	bit

Beschreibung:

Jede ACL setzt sich aus mindestens einer, in den meisten Fällen aber mehreren ACEs zusammen. In der Datenbank werden ACLs und ACEs in zwei Tabellen gespeichert. Genau aus diesem Grund bewirkt die Speicherung einer ACL in der Datenbank zugleich die Speicherung mehrerer Einträge in der „ace“-Tabelle.

In Abb. 31 wird ein Ausschnitt aus einer gefüllten „ace“-Tabelle gezeigt.

acl_id	sid	mask	type	flags	objectType	inheritedObj	fromStdSec
30	S-1-5-21-1343024091-158043	16	5	26	{037088F8-0A	{BF967ABA-0	0
30	S-1-5-21-1343024091-158043	16	5	26	{4C164200-20	{BF967ABA-0	0
30	S-1-5-21-1343024091-158043	16	5	26	{59BA2F42-7	{BF967ABA-0	0
30	S-1-5-21-1343024091-158043	16	5	26	{5F202010-79	{BF967ABA-0	0
30	S-1-5-21-1343024091-158043	16	5	26	{BC0AC240-7	{BF967ABA-0	0

Abb. 31: Ausschnitt aus Tabelle "ace"

Attribute:

- **acl_id**

  Der hier gespeicherte acl_id verweist auf eine ACL aus der „acl"-Tabelle. Es existieren in der Regel mehrere ACEs in der Datenbank, bei denen der acl_id gleich ist.

- **sid**

  Der SID einer ACE wird in diesem Feld gespeichert. SIDs können fast beliebig lang werden. Die Länge des Feldes wurde mit der Konstante SAT_SID_LEN festgelegt. Sollte der SID in einer ACE fehlen, so wird der Initialwert SAT_ACE_NO_SID gespeichert.

- **mask**

  Die Access-Mask eines ACEs wird vor der Speicherung noch transferiert – bei diesem Prozess werden die „Generic Rights" aufgelöst (siehe Kap. 3.6.3.3). Dieser berechnete Wert wird im Feld „mask" abgespeichert. Der Initialwert wird hier durch die Konstante SAT_ACE_NO_MASK festgelegt.

- **type**

  Wir unterscheiden vier verschiedene ACE-Typen. Diese werden jeweils durch eine Bit-Maske repräsentiert und so in der Datenbank gespeichert:

ACCESS_ALLOWED_ACE_TYPE (ADS_ACETYPE_ACCESS_ALLOWED)	0x00
ACCESS_DENIED_ACE_TYPE (ADS_ACETYPE_ACCESS_DENIED)	0x01
ACCESS_ALLOWED_OBJECT_ACE_TYPE (ADS_ACETYPE_ACCESS_ALLOWED_OBJECT)	0x05
ACCESS_DENIED_OBJECT_ACE_TYPE (ADS_ACETYPE_ACCESS_DENIED_OBJECT)	0x06

  Initialwert des ACE-Typs ist der Wert SAT_ACE_NO_TYPE.

- **flags**

  In diesem Flag werden die ACE-Flags gespeichert. Diese geben hauptsächlich Auskunft über den Vererbungsprozess. Initialwert ist hier SAT_ACE_NO_FLAGS. Das Feld „flags“ speichert ebenso wie „type“ die Werte kodiert in einer Bitmaske als Integerwert ab. Hier ein Überblick über die Konstanten (die Erklärung der Konstanten wurde bereits in Kapitel 2.4.1.3 vorgenommen):

ADS_ACEFLAG_INHERIT_ACE	0x02
ADS_ACEFLAG_NO_PROPAGATE_INHERIT_ACE	0x04
ADS_ACEFLAG_INHERIT_ONLY_ACE	0x08
ADS_ACEFLAG_INHERITED_ACE	0x10
ADS_ACEFLAG_VALID_INHERIT_FLAGS	0x1F

  Zusätzlich zu den ACE-Flags werden die Object ACE Flags ebenfalls in im Feld „flags“ gespeichert:

SAT_ACE_OBJECT_TYPE_PRESENT	0x100
SAT_ACE_INHERITED_OBJECT_TYPE_PRESENT	0x200

  Anhand der ACE Flags kann man bestimmten, ob die Felder „objectType“ und „inheritedObjectType“ einen Wert haben müssen oder nicht. Dies macht im Falle eines leeren Feldes sehr wohl einen Unterschied, da hier je nach Access Mask eine andere Interpretation getroffen werden muss. So kann z.B. ein leeres „objectType“-Feld alle Attribute eines Objektes bezeichnen – sofern ein entsprechendes ACE-Flag vorliegt.

- **objectType**

  Im Feld „objectType” wird der Object-GUID – sofern vorhanden - gespeichert. Dieser GUID verweist auf das Feld „objTypeGuid“ aus der Tabelle „objTypes“. Die Länge der GUID ist mit der Konstante SAT_GUID_LEN, der Initialwert mit SAT_ACE_NO_OBJTYPE festgelegt. Der Object-GUID einer ACE kann sowohl auf ein Attribut, eine Objektklasse als auch auf ein „Extended Right“ verweisen.

- **inheritedObjectType**

  Für das Feld „inheritedObjectType“ gilt generell dasselbe wie für „objectType“: Der gespeicherte GUID verweist ebenfalls auf ein Attribut, eine Objektklasse oder auf ein „Extended Right“ aus der Tabelle „objTypes“. Unterschied zum „objectType“ ist jedoch, dass im Feld „inheritedObjectType“ ein Inherited Object GUID gespeichert wird – siehe dazu auch Kap. 2.4.1.3.

- **fromStdSecurity**

  Dieses Bit zeigt an, ob der gespeicherte ACE aus der Standard-Security des Schemas kommt. TRUE bedeutet in diesem Fall, dass die ACE aus dem Schema kommt. FALSE (Initialwert) bedeutet, dass es sich um eine „normale“ ACE handelt.

3.4.3 Tabelle “objTypes”

objTypes	
objTypeGuid	varchar(SAT_GUID_LEN)
compID	int
typeName	varchar(SAT_TYPENAME_LEN)
objectClass	char(1)
defaultSec	int
version	int

Beschreibung:

Die Tabelle „objTypes“ dient als Speicher für zwei verschiedene Arten von Daten:

1) sämtliche Objekttypen aus dem Schema werden hier abgespeichert.
2) Extended Rights aus dem Configuration-Container werden ebenfalls hier abgelegt.

Diese Tabelle wird ausschließlich für die Auswertung benötigt. Der Objekttyp jedes Objekts kann somit zurückverfolgt werden.

Die Rechtevergabe ermöglicht es, den Zugriff auf einzelne Attribute zu regeln. Die Attributnamen können in der Visualisierung durch eine Abfrage an diese Tabelle angezeigt werden.

Die Auswertung berücksichtigt zwar keine Extended Rights, es wird aber dennoch angezeigt, ob solche Rechte vorhanden sind. Der Name des entsprechenden Extended Rights muss ebenfalls aus dieser Tabelle kommen.
Der Primärschlüssel setzt sich aus den Attributen objTypeGuid, compID und version zusammen.
Abb. 32 zeigt Tabelle „objTypes“ nach Durchführung der Schemaanalyse.

objTypeGuid	compID	typeName	objectClass	defaultSec	version
{9A0DC332-C100-11D1-BBC5-0080(	1	MSMQ-Sign-Key	a	1	1
{BF967A2E-0DE6-11D0-A285-00AA	1	Search-Guide	a	1	1
{05F6C878-CCEF-11D2-9993-0000F	1	MS-SQL-SQLServer	c	2	1
{93C5D9FE-128F-49A9-8F37-E9A52	1	msFPC-UnlimitedNu	a	1	1
{C3C4DB49-5D01-448D-A440-A46D:	1	Extended-Rights	e	-1	1

Abb. 32: Ausschnitt aus Tabelle "objTypes"

Attribute:

- **objTypeGuid**

  Jedes Objekt im Schema (sowohl Klassen- als auch Attributdefinition) hat einen eindeutigen GUID, welcher in diesem Feld gespeichert wird. Auf diesen GUID wird von anderen Tabellen verwiesen (ace und objects).

- **compID**

  Das Schema des Active Directory wird im Normalfall nur einmal pro Analysedurchgang ausgelesen werden. Um nachträglich rückfolgern zu können, von welchem Rechner (Domain-Controller) das Schema gekommen ist, wird im Feld „compID“ ein Verweis auf den ID der Tabelle „computer“ (siehe [Zar]) gespeichert. Der ADS-Scanner erhält diesen ID als Parameter.

- **typeName**

  Beinhaltet den Namen des Objekt-Typs bzw. des Attributs. Handelt es sich um ein Extended Right, so wird dessen Name gespeichert.

- **objectClass**

  In diesem Feld wird der Objekt-Typ des Eintrages spezifiziert. Folgende Werte stehen derzeit zur Verfügung:

Objekt-Klasse	Wert
Attribut	a
Klasse	c
Extended-Right	e

- **defaultSec**

  Im Schema kann für Objekt-Klassen eine Standard-Security definiert werden, welche für den ADS-Scanner von Interesse sind. Somit ist es möglich zu bestimmen, ob Berechtigungen auf einem Objekt dem vordefinierten Standard für dessen Objekt-Klasse entsprechen. Im Falle einer Ungleichheit kann somit mit Bestimmtheit gesagt werden, dass Änderungen an den Objekt-Berechtigungen vorgenommen wurden. Sollte ein Objekt keine Standard-Security definiert haben, so wird die Konstante SAT_NO_DEFAULT_SD abgespeichert.

  Für Attribute und Extended Rights ist dieses Feld von vorne herein nicht von Bedeutung, sie füllen dieses Feld mit SAT_NO_DEFAULT_SD.

- **version**

  In diesem Feld wird die Version des Security-Scans gespeichert.

3.4.4 Tabelle “objects”

objects	
obj_id	int
objName	varchar(SAT_OBJNAME_LEN)
compID	int
objGuid	varchar(SAT_GUID_LEN)
amount	int
objectTypeGuid	varchar(SAT_GUID_LEN)
parent	int
path	varchar(SAT_COMPR_PATH_LEN)
date	char(17)
acl_id	int
breaksInheritance	bit
breaksStdSecurity	bit
fullpath	varchar(SAT_FULL_PATH_LEN)
version	int

Beschreibung:

Jedes analysierte Objekt erhält einen Eintrag in dieser Tabelle (im Falle der Komprimierung kommt es vor, dass mehrere Objekte zu einem Eintrag zusammengefasst werden). Das gesamte Active Directory Verzeichnis kann anhand dieser Tabelle rekonstruiert werden. Diese Tabelle hat somit eine zentrale Rolle in der SAT-Datenbank.

Der Primärschlüssel besteht lediglich aus dem Feld obj_id.

Abb. 33 zeigt einen möglichen Ausschnitt aus einer durchgeführten ADS-Analyse.

obj_id	objName	compID	objGuid	amo	objTypeGuid	parent	path	date	acl_id	breaksInh	breaksStd	full	vers
12	Users	1	{7D397F47-967	1	{BF967A8B-0DE6-1	1	a-0-1	05-11-01	31	0	-1	_	1
13	Administrator	1	{7508C00F-30F	1	{BF967ABA-0DE6-1	12	a-0-1-0	05-11-01	32	-1	-1	_	1
14	Guest	1	{39562CCB-A6	1	{BF967ABA-0DE6-1	12	a-0-1-1	05-11-01	33	0	-1	_	1
15	TsInternetUser	1	{7703A5A2-A9	1	{BF967ABA-0DE6-1	12	a-0-1-2	05-11-01	33	0	-1	_	1
16	krbtgt	1	{AD2AB82B-75	1	{BF967ABA-0DE6-1	12	a-0-1-3	05-11-01	34	0	0		1

Abb. 33: Ausschnitt Tabelle "objects"

Attribute:

- **obj_id**

  Automatischer ID, der von der Datenbank erzeugt wird. Kleinster ID ist 1, wird schrittweise um eins erhöht.

- **objName**

  Der Common Name (cn) eines jeden gescannten Objekts wird in diesem Feld abgespeichert. Dieser hier gespeicherte Name wird in späterer Folge auch von der Visualisierung verwendet.

- **compID**

  Computer-ID jenes Rechners, von dem die hier gespeicherte Information gelesen wurde.

- **objGuid**

  Jedes Objekt im Active Directory hat einen eindeutigen GUID, welcher im Feld „objGuid“ abgespeichert wird.

- **amount**

  Anzahl der durch die Komprimierung zusammengefassten Objekte. Weitere Details dazu finden sich im Kapitel 3.5.

- **objectTypeGuid**

  Dieses Feld beinhaltet den GUID des Objekt-Typs. Dieser GUID verweist somit auf das Feld „objTypeGuid“ der Tabelle „objTypes“.

- **parent**

  Das Feld “parent” verweist auf das Vaterobjekt des aktuellen Objekts – also des Containers, in welchem sich dieses Objekt befindet. Gespeichert wird die „obj_id“ des Vaterobjekts.

- **path**

  Im Feld „path“ wird der sogenannte „komprimierte Pfad“ abgespeichert. Dies wurde in Hinblick auf einen schnellen Datenbankzugriff gemacht. Eine genaue Erklärung dieses Prinzips wird in Kapitel 3.5.5 gegeben.

- **date**

  Wahlweise wird das Erzeugungsdatum eines Objekts gespeichert oder sein letztes Änderungsdatum. Diese Entscheidung kann im Controller-Programm (siehe dazu auch Kapitel 3.3) getroffen werden.

- **acl_id**

  Für jedes Objekt werden die Berechtigungen ausgelesen und in den Tabellen „ace“ und „acl“ gespeichert. Dieses Feld verweist auf den „acl_id“ der beiden Tabellen.

- **breaksInheritance**

  Wird der Vererbungsmechanismus der Security bei diesem Objekt unterbrochen, so enthält das Feld den Wert TRUE. Sonst wird FALSE gespeichert.

  Jedem Objekt im Active Directory wird üblicherweise eine ACL zugewiesen. Einzelne ACEs einer solchen ACL können in der Baum-Hierarchie nach „unten“ vererbt werden. Der Vererbungsprozess kann an jeder beliebigen Stelle im Baum unterbrochen werden. Dies wird in der ADS-Analyse als Bruch der Vererbung bezeichnet. Im Regelfall entsteht dies durch eine absichtliche, vorsätzliche Änderung einer berechtigten Person, in manchen Fällen auch durch das System selbst. Generell kann man festhalten, dass ein Bruch der Vererbung eine Abweichung der Norm darstellt. Deswegen erscheint es umso wichtiger, diese Ausnahme anzuzeigen und auf die Veränderung hinzuweisen. In der Visualisierung wird dies durch eine Einfärbung des Objekts, welches die Vererbung unterbricht, signalisiert.

- **breaksStdSecurity**

  Im Schema wird für jeden Objekttyp eine ACL, die sogenannte Standard-Security spezifiziert. Wird ein neues Objekt im Verzeichnis erzeugt, so erhält es

automatisch die Standard-Security des entsprechenden Objekt-Typs. Zusätzlich kommen einzelne Berechtigungen durch den Vererbungsprozess hinzu.

Um festzustellen, ob ein Bruch der Standard-Security vorliegt, müssen zuerst (bei diesem Teil der ADS-Analyse) alle ACEs, die durch die Vererbung dazugekommen sind, ignoriert werden. Jene ACEs, die „direkt“ auf das Objekt vergeben wurden vergleicht man mit den entsprechenden ACEs aus dem Schema (der Standard-Security). Kommt es zu einer Übereinstimmung, so wurde an den Berechtigungen des Objekts nichts verändert. Findet man eine Abweichung, so spricht die ADS-Analyse von einem „Bruch der Standard-Security“. Somit stellt „breaksStdSecurity“ neben „breaksInheritance“ einen wichtiger Indikator dafür dar, dass die Berechtigungen eines Objekts verändert wurden – also eine Abweichung von der Norm vorliegt.

Die Visualisierung bedient sich auch hier einer bestimmten Farbe, um einen solchen Bruch anzuzeigen.

In der Datenbank bedeutet TRUE, dass ein Bruch der Standard-Security vorliegt. FALSE signalisiert, dass die Standard-Security nicht verändert wurde.

- **fullpath**

  Der vollständige Distinguished Name eines Objekts wird nur bei bestimmten (neueren) Komprimierungsstufen (Stufe 2 und 3) gespeichert. Bei einer Komprimierung mit Stufe 2 und 3 können unter Umständen mehrere Hierarchiestufen völlig „wegkomprimiert“ werden. Der Verzeichnisbaum kann somit in der Visualisierung nicht vollständig rekonstruiert werden. Darum ist es notwendig, den vollständigen Distinguished Name zu speichern, um das Objekt dem Verzeichnisbaum zuzuordnen.

  Im Normalfall ist dieses Feld mit der Konstante SAT_EMPTY gefüllt (Komprimierungsstufe 0 und 1). Zukünftige SAT-Versionen werden den „fullpath“ in eine eigene Tabelle ausgliedern und lediglich einen Verweis darauf (üblicherweise einen ID) speichern.

- **version**

  In diesem Feld wird die Version des Security-Scans gespeichert.

3.4.5 Tabelle “membership”

membership	
sid	varchar(SAT_SID_LEN)
memberOfSid	varchar(SAT_SID_LEN)
isSidHistory	bit
version	int

Beschreibung:

Die Tabelle “membership” spielt in der Auswertung eine wichtige Rolle. Sie wird sowohl vom ADS-Scanner als auch von der User/Gruppenanalyse verwendet (siehe dazu [Zar]). Der ADS-Scanner verwendet diese Tabelle, um die sogenannte SID-History abzuspeichern – diese Problematik wird in Kapitel 3.6.3.1. weiter ausgeführt.

Attribute:

- **sid**

  Dieses Attribut speichert die Windows 2000 SID eines „Security Principals“ ab.

- **memberOfSid**

  Security Principals unter Windows 2000 können unter Umständen mehrere SIDs haben (vergleiche dazu „SID-History“, Kapitel 3.6.3.1).

  Im Feld „memberOfSid“ werden diese SIDs gespeichert. Die User-/ Gruppen-analyse speichert hier die User- / Gruppen-Beziehung ab.

- **isSidHistory**

  Dieses Bit signalisiert, ob der Eintrag eine SID-History ist (Wert TRUE), oder ob er aus der User-/Gruppenanalyse kommt (Wert FALSE).

- **version**

  In diesem Feld wird die Version des Security-Scans gespeichert.

3.4.6 Tabelle “sidRef”

sidRef	
sid	varchar(SAT_SID_LEN)
objRef	int
version	int

Beschreibung:

Alle Security Principals erhalten bei der Erzeugung eine Domain-weite eindeutige SID zugewiesen. Um bei der späteren Auswertung noch feststellen zu können, woher eine SID gekommen ist, wird die Objekt-ID des Vater-Containers gespeichert.
Somit ist es nachträglich noch möglich, festzustellen, dass z.B. das Objekt (der User) „Hörmanseder“ im Container „SAT“ zu finden ist. In der Datenbank wird mit der SID von „Hörmanseder“, die Objekt-ID des Objekts „SAT“ mitgespeichert.

Attribute:

- **sid**
  SID eines Active Directory Security Principal. Sobald während der Analyse ein Objekt gefunden wird, welches ein SID-Attribut hat, wird dieses ausgelesen und in der Tabelle „sidRef“ gespeichert.

- **objRef:** Wird bei der Analyse ein Objekt mit einem SID gefunden, so wird dies gespeichert. Das Feld „objRef“ verweist auf die Objekt-ID jenes Objekts, in dem sich das Objekt mit dem SID-Attribut befindet.

- **version:** In diesem Feld wird die Version des Security-Scans gespeichert.

3.5 Komprimierung

Das SAT-System wurde als Tool zur Unterstützung des Administrators konzipiert. Die Auswertung der Sicherheitsanalyse soll daher die wichtigsten Informationen übersichtlich und möglichst einfach präsentieren. Damit ist es notwendig, die gesammelten Daten aufzubereiten, also zusammenzufassen. Diesen Vorgang bezeichnen wir als Komprimierung.

Die Komprimierung ist eigentlich das wichtigste Konzept der Visualisierung. Dennoch wurde ein Teil in die Analyse aufgenommen. Der Grund liegt auf der Hand: Die Datenmenge einer vollständigen Analyse (ADS+NTFS+Registry) kann schon bei kleinen Server-Systemen relativ groß werden. Die nachträgliche Komprimierung aus dem Datenbestand würde zu lange dauern. Daher ist es bereits vor dem Analysebeginn möglich, eine Komprimierungsstufe festzulegen. Es sollte jedoch bedacht werden, dass die Komprimierung nicht rückgängig gemacht werden kann. Sollten zum Auswertungszeitpunkt detailliertere Einzelheiten, als die in der Datenbank gespeicherten benötigt werden, so muss ein neuer Scan mit geringerer Komprimierungsstufe durchgeführt werden.

Der ADS-Scanner kennt derzeit vier verschiedene Komprimierungsstufen (eigentlich nur drei, da Komprimierungsstufe 0 nichts komprimiert). In der Visualisierung wird noch zwischen mehreren anderen Komprimierungsstufen unterschieden. Für Details dazu sei auf die Arbeit des Kollegen Zarda [Zar] verwiesen.

3.5.1 Komprimierungsstufe 0: keine Komprimierung

Die einfachste Komprimierungsstufe ist die unkomprimierte Speicherung. Es werden alle Objekte des Active Directory in die Datenbank gespeichert.

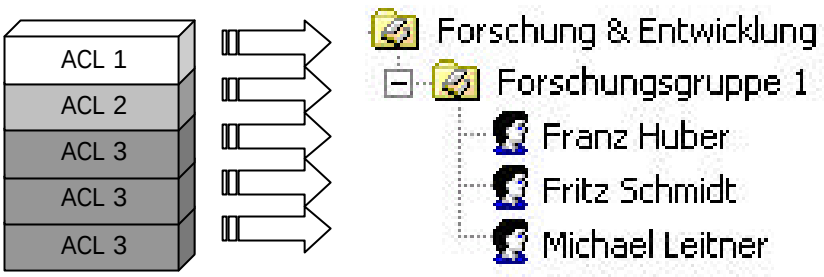

Abb. 34: Komprimierungsstufe 0

Abb. 34 zeigt den Teilbaum „ Forschung & Entwicklung“ aus dem Active Directory. In der Datenbank würden fünf Einträge in der „object“ Tabelle benötigt werden. Wie wir sehen können, gibt es drei verschiedene ACLs. Die Objekte „Franz Huber“, „Fritz Schmidt“ und „Michael Leitner“ weisen dabei die gleiche ACL auf. Aus Administratorsicht haben diese drei Objekte die gleichen Berechtigungen.
Für die Auswertung bedeutet die Komprimierungsstufe 0 vor allem eine 100%ige Genauigkeit. Zugleich handelt man sich aber gleich mehrere Nachteile ein:

1) Die Datenbankgröße wächst stark an, da jedes Objekt im System einem Eintrag in der Datenbank entspricht.
2) Die Visualisierung wird schnell unübersichtlich, wenn zu viele Daten angezeigt werden. Müssen aber für die Visualisierung viele Daten zuerst komprimiert werden, so wird das System entsprechend langsamer.

3.5.2 Komprimierungsstufe 1: Zusammenfassen gleicher Objekttypen

Im Dateisystem NTFS funktioniert die Komprimierungsstufe 1 sehr einfach – siehe [Ach]. Dateien in einem Verzeichnis werden genau dann zusammengefasst, wenn zwei Bedingungen erfüllt sind:

1) Dateien haben die gleichen Berechtigungen, sprich die gleiche ACL.
2) Die Dateitypen der Dateien müssen gleich sein, z.B: alle EXE-Files mit gleicher ACL. So werden z.B. EXE-Dateien nicht mit TXT-Dateien zusammengefaßt.

Im Active Directory sieht die Sache ein wenig anders aus, da es keine Dateitypen gibt. Jedes Objekt im Active Directory wird jedoch von einer Klasse abgeleitet – der Objektklasse, auch Objekttyp genannt. Das Pendant zur NTFS-Komprimierung sieht daher folgendermaßen aus:
Active Directory Objekte werden dann gemäß Komprimierungsstufe 1 zusammengefasst, wenn

1) sich die Objekte in einem Container der selben Hierarchiestufe befinden.
2) sich unter den zusammenzufassenden Objekten kein Container befindet, welcher selbst Objekte beinhaltet.
3) die Objekte eine gleiche ACL haben.
4) die Objekte vom gleichen Objekttyp sind.

Abb. 35 zeigt das Beispiel aus dem letzten Kapitel 3.5.1, nur diesmal mit Komprimierungsstufe 1:

Abb. 35: Komprimierungsstufe 1

3.5.3 Komprimierungsstufe 2: nur Ausnahmen

Für den Administrator sind vor allem Abweichungen vom Regelfall interessant. Die Komprimierungsstufe 2 zielt genau auf diesen Fall ab. In der Datenbank werden nur jene Objekte gespeichert, welche

- die Vererbung der Berechtigungen unterbrechen
- Änderungen gegenüber der Standard-Security aus dem Schema aufweisen.

Diese Art der Zusammenfassung ist „stark verlustbehaftet". Es kann passieren, dass ganze Teilbäume verschwinden und nur einzelne Blätter des Baums gespeichert werden. (Somit wäre es unmöglich im Nachhinein den Verzeichnisbaum aufzubauen und diese Objekte hierarchisch einzuordnen. Genau aus diesem Grund wird bei der Komprimierungsstufe 1 der Distinguished Name eines jeden Objekts mitgespeichert. Dies wird bei der Visualisierung genutzt, um anzuzeigen, wo sich dieses Objekt im Verzeichnisbaum befindet.)

3.5.4 Komprimierungsstufe 3: Kombination aus Komprimierungsstufe 1+2

Die derzeit höchstmögliche Komprimierung wird mit Komprimierungsstufe 3 erreicht. Sie vereint sowohl Stufe 1 (Zusammenfassen von Objekte mit gleicher Security und gleichem Objekttyp) und Stufe 2 (nur Ausnahmen).

Bei dem eigentlichen Komprimierungsvorgang wird zuerst Komprimierungsstufe 1 angewandt. Ausgehend von dieser Liste werden nur mehr jene Objekte in die Datenbank übernommen, welche einen Bruch der Standard-Security bzw. in der Vererbung auf-

weisen. Diese Komprimierungsstufe ist zugleich auch jene, welche am meisten Informationsverlust verursacht. Auch hier wird – genau wie in der Stufe 2 – der Distinguished Name eines jeden Objekts mitgespeichert (siehe oben).

3.5.5 Komprimierter Pfad

Die Tabelle „objects“ (vgl. Kap. 3.4.4) weist ein Feld „path“ auf, in welchem der so-genannte „komprimierte“ Pfad für jedes Objekt abgespeichert wird. Bei der Auswertung lassen sich damit Bereichsabfragen einfach realisieren.

Die Berechnung des komprimierten Pfads für ein Objekt ist relativ simpel:
Jedes Objekt erhält eine Nummer, die es in einer Hierarchiestufe eindeutig identifiziert. Zu Zählen begonnen wird bei der Zahl „0“. Wandert man in der Hierarchiestufe einen Schritt nach unten, so wird dies durch einen Bindestrich „-„ angedeutet. Jedes Teilsystem des SAT-Scanners hat dabei einen eigenen Wurzelknoten: „A“ für Active Directory, „N“ für NTFS und „R“ für Registry. Ein einfaches Beispiel veranschaulicht Abb. 36:

Abb. 36: Beispiel: Komprimierter Pfad

Die Abfrage „Zeig mir den Teilbaum Forschung & Entwicklung“ lässt sich vereinfacht so formulieren: „Zeig mir alle Einträge mit dem komprimierten Pfad A-0-*“

Die Nummerierung erfolgt nicht im Dezimalsystem, der komprimierte Pfad baut auf einem Zahlensystem mit Basis 62 auf. Diese ominöse Zahl 62 lässt sich leicht erklären: Die Idee stammt vom Hexadezimalsystem, welches für die weiteren Ziffern die Buch-

staben „A“-„F“ verwendet. Ein Zahlensystem mit Basis 62 benötigt 62 verschiedene Ziffern, welche gebildet werden durch:

	0-9:	10 Ziffern
+	A-Z:	26 Ziffern
+	a-z:	26 Ziffern
		62 Ziffern

Diese Vorgehensweise spart gegenüber der Verwendung des Dezimalsystems in der Datenbank eine Menge Platz, da die maximale Länge des Feldes nicht so groß wird (man stelle sich unzählige Hierarchiestufen mit jeweils mehreren Tausend Einträgen vor). Die Einschränkung auf Ziffern und Zahlen wurde nur wegen der erhöhten Lesbarkeit getroffen, bei der Verwendung des kompletten ASCII-Satzes wäre dies nicht gegeben.
Die Idee zum komprimierten Pfad stammte von Hrn. Dipl.-Ing. Hörmanseder, der diese in Anlehnung an die sogenannten „abilities“ (vgl. dazu [Gol/Seite 39f]) entwickelte.

3.6 Implementierung

Microsoft stellt den Entwicklern zwei verschiedene APIs für den Zugriff auf den Verzeichnisdienst Active Directory an:

1) LDAP (Lightweight Directory Access Protocol)
2) ADSI (Active Directory Services Interface)

LDAP ist eine für die Programmiersprache C entwickeltes API, es lässt sich im Vergleich zu ADSI eher als Low-Level API bezeichnen. Die Microsoft LDAP Implementierung entspricht weitgehend dem Standard, welcher in RFC 1823 festgelegt ist.
ADSI besteht aus einem Set von COM-Interfaces und ist somit voll objektorientiert. Im Vergleich zu LDAP kann man ADSI als High-Level API kategorisieren. ADSI kann mittels C++, C, Visual Basic und auch von Scriptsprachen wie VBScript verwendet werden. ADSI entspricht keinem international anerkannten Standard, wie beispielsweise

LDAPv3. Interessant hingegen ist die Tatsache, dass von ADSI intern LDAP-Calls verwendet werden. ADSI ist demnach „nur“ eine objektorientierte Kapselung von LDAP Aufrufen. ADSI lässt allerdings auch die Möglichkeit offen, auf andere Verzeichnisdienst-Standards zuzugreifen.

Zum SAT-Projektbeginn stellte sich die Frage, welches dieser beiden Zugriffsprotokolle verwendet werden sollte. Die Entscheidung ging zu Gunsten von ADSI, wofür mehrere Gründe sprachen. Das SAT-System in seiner derzeitigen Implementierung ist ein reines Microsoft Administratoren-Tool. Dies wird sich sehr wahrscheinlich auch in den nächsten Versionen nicht ändern. ADSI ist das von Microsoft favorisierte Protokoll, was sich zum Teil durch bessere Dokumentation von ADSI (Codebeispiele, etc.) bemerkbar macht. Wie sich der Verzeichnisdienst Active Directory weiterentwickelt, wird sich noch zeigen. Von einem kann man jedoch ganz klar ausgehen: Sollte jemals eine Entscheidung zwischen LDAP und ADSI gemacht werden, steht jetzt schon der Sieger fest. Diese Gründe gaben letztendlich den Ausschlag und somit machte ADSI das Rennen.

Dieses Kapitel soll keine Einführung in die Programmierung mit ADSI bzw. LDAP werden. Die interessierte Leserschaft sei an dieser Stelle auf das hervorragende Werk von Gil Kirkpatrick [Kir] verwiesen. In diesem Kapitel geht es vielmehr um die eigentliche Implementierung des ADS-Scanners. Dabei wird die Funktionsweise des Programms aufgezeigt. Probleme, die während der Arbeit am ADS-Scanner auftraten, werden ebenso behandelt wie deren Lösungen. Somit soll eine Referenz für spätere Verbesserungen und Arbeiten an neuen SAT Versionen geschaffen werden.

3.6.1 Klassenbeschreibung

Im folgenden Kapitel werden die implementierten und verwendeten Klassen, welche den ADS-Scanner bilden, vorgestellt. Eine Einarbeitung wird durch die Erklärung der wichtigsten Methoden dieser Klassen enorm erleichtert.

3.6.1.1 Klasse SqlDB

Für den Zugriff auf den SQL-Server wird ADO verwendet. Die Benutzung von ADO unter C++ gestaltet sich leider nicht ganz einfach. Die Alternative wäre eine Implementierung unter Visual Basic gewesen, wo die Verwendung von ADO weitaus besser unterstützt wird (Online-Dokumentation). Aus Sicht der Bequemlichkeit war von einer Implementierung der Datenbank unter Visual C++ völlig abzuraten. Das Projektteam hat sich dennoch für die Kombination ADO/C++ entschlossen, Grund ist die höhere Performance. Die Erfahrungen mit SAT 1 hatten gezeigt, dass damals der Flaschenhals eindeutig bei der Datenbank lag. Alleine schon der Umstieg von ODBC und Access (SAT 1) auf ADO und SQL-Server (SAT 2) bedeutet eine Verbesserung, es sollte aber mit der Entscheidung zugunsten von C++ auf Nummer sicher gegangen werden.

Bevor Daten in die Datenbank gespeichert werden bzw. bevor man Daten aus der Datenbank abfragen kann, muss zuerst eine Verbindung zur Datenbank hergestellt werden. Sobald diese Verbindung hergestellt ist, müssen einzelne Records zum Lesen oder Schreiben geöffnet werden. Erst dann kann der Datenaustausch begonnen werden.

Methoden:

ConnectToDB

Für den Verbindungsaufbau zur Datenbank ist die Methode ConnectToDB zuständig. Die Berechtigungen für den Datenbank-Server müssen vom Datenbank-Administrator entsprechend gesetzt werden. Der ADS-Scanner geht von einer „trusted“-Connection aus, d.h. die Rechner, von denen gescannt wird, erhalten Lese- und Schreibzugriff auf die entsprechende Datenbank. Als Alternative können Usernamen und auch das Passwort für den Zugriff auf die Datenbank als Parameter angegeben werden. Davon ist aber aus Gründen der Sicherheit abzuraten.

OpenRecord

Um in der Datenbank Änderungen vorzunehmen bzw. um Daten auszulesen ist es nötig, den zu ändernden Teil vorweg zu bestimmen. Dies geschieht durch Aufruf der OpenRecord Methode. In diesen so geöffneten Record können dann Daten geschrieben oder gelesen werden. Spezifiziert wird der Record durch

- Angabe eines Tabellennamens, oder
- Selektieren von Attributen durch das SQL SELECT Statement

Der Schreibzugriff im SAT-Scanner erfolgt prinzipiell nur in einzelne Tabellen, welche seriell geschrieben werden. Daher wird die OpenRecord-Methode für den Schreibzugriff durch Angabe eines Tabellennamen geöffnet. Anders sieht es bei der Auswertung und Visualisierung [Zar] aus. Hier ist es sehr wohl nötig, dass Daten mehrerer Tabellen gleichzeitig ausgelesen werden. Dies ist nur mittels einer geeigneten SELECT-Anweisung möglich: Beispiel: „SELECT * FROM sat..ace, sat..acl“
Pro Instanz der Klasse SqlDB kann nur ein Record geöffnet sein. Möchte man in einen zweiten Record schreiben bzw. aus einem zweiten lesen, so muss ein zweites Objekt der Klasse instanziiert sein oder der Record wird geschlossen und neu geöffnet.

CloseRecord

Ein bereits geöffneter Record wird durch den Aufruf der Klasse CloseRecord geschlossen. Dies muss dann geschehen, wenn man auf einen anderen Record zugreifen möchte. Wird der Destruktor der Klasse SqlDB aufgerufen, so wird ein offener Record dort noch geschlossen.

AddRecord

Daten werden mittels AddRecord in die Datenbank übernommen. Mit jedem Aufruf von AddRecord kann genau einen Datensatz gespeichert werden. Bevor AddRecord ausgeführt wird, muss der zu schreibende Record mittels OpenRecord geöffnet worden sein. Neben den Werten, welche gespeichert werden sollen, müssen auch die Feldnamen aus den Tabellen, in die gespeichert wird, bekannt sein. AddRecord bekommt diese Daten in zwei Arrays übergeben, beide müssen vom Typ COleSafeArray sein.

GetRecords

Das Auslesen der Daten aus der Datenbank geschieht mittels GetRecords. Auch hier muss zuerst durch Aufruf von OpenRecord jener Teil der Datenbank selektiert werden, den man auslesen möchte. Zum Selektieren kann man wahlweise einen Tabellennamen oder auch ein SELECT Kommando übergeben. GetRecords ermöglicht es, mehrere Records gleichzeitig auszulesen, welche dann in einem Varianten-Record übergeben werden.

ExecSQL

Die Methode ExecSQL ermöglicht es, SQL-Anweisungen abzusetzen. Somit können auch komplizierte Datenbank-Anweisungen in SQL verwenden werden. Sollte man mit GetRecords und AddRecords nicht das Auslangen finden, so steht der volle Funktionsumfang von SQL-Server trotzdem zur Verfügung. SAT verwendet diese Methode beispielsweise, um die SAT-Datenbank und sämtliche benötigten Tabellen mittels CREATE-Statements anzulegen.

3.6.1.2 Klasse Ace

Die Berechtigungen eines Active Directory Objekts werden ausgelesen und in den Objekten vom Typ Ace und Acl gespeichert. Wie bereits mehrmals erwähnt besteht eine ACL i.d.R. aus mehreren ACEs. SAT hat hier die Speicherstruktur von Windows als Vorbild genommen und nachgebildet. Wird vom ADS-Scanner eine ACL ausgelesen, so wird diese in ihre Bestandteile – sprich ACEs - zerlegt. Für jede ACE wird ein Objekt der Klasse Ace angelegt und mit den entsprechenden Werten gefüllt. Unterschieden wird in weiterer Folge zwischen ACEs, welche aus der Standard-Security aus dem Schema kommen, und den „normalen“ ACEs.

Ace-Objekte werden vor der Speicherung in der Datenbank im Speicher gehalten. Ein Acl–Objekt speichert immer nur einen Zeiger auf das erste Ace-Objekt. Ace-Objekte sind mittels einer einfach verketteten Liste gespeichert.

Methoden:

Get-/SetAttribut

Für „Attribut“ in GetAttribut / SetAttibut kann eingesetzt werden kann: Flags, InheritedObjectType, Mask, ObjectType, Sid und Type. Diese Attribute können außerhalb der Klasse nur über die entsprechenden Methoden mit Get- bzw. Set- ausgelesen bzw. geschrieben werden.

GetHash

Berechnet einen einfachen Hash-Code einer ACE.

SetFromStdSecurity / IsFromStdSecurity

Die Methode SetFromStdSecurity wird genau dann mit dem Parameter TRUE aufgerufen, wenn eine ACE aus dem Schema kommt. Dies spielt für die Auswertung in weiterer Folge eine wichtige Rolle. Mittels IsFromStdSecurity kann abgefragt werden, ob eine ACE aus dem Schema kommt oder nicht.

ClearAce

Überschreibt die im Ace-Objekt gespeicherten Werte und stellt den Zustand der Initialisierung wieder her.

Equal

Vergleicht zwei Ace-Objekte auf Übereinstimmung. Der Rückgabewert TRUE signalisiert eine völlige Übereinstimmung der beiden verglichenen Ace-Objekte.

IsSmallerThan

Ace-Objekte werden vom SAT-System geordnet abgespeichert. Nur so ist ein Vergleich möglich. Um eine Sortierung zu realisieren ist eine Vergleichsoperation zwingend notwendig, was mit der Methode IsSmallerThan erreicht wird.

3.6.1.3 Klasse Acl

Die Klasse Acl dient zum Speichern der ausgelesenen Berechtigungen (ACLs). Sobald eine ACL aus der Sicherheitsanalyse kommt, wird sie in die einzelnen ACEs zerlegt und in Ace-Objekten gespeichert. Diese Ace-Objekte werden dann an ein Acl-Objekt „weitergereicht“ und dort gespeichert. Aus Sicht der Implementierung wird dies durch eine verkettete Liste von Ace-Objekten realisiert, wobei ein Zeiger auf das erste Ace-Objekt im Acl-Objekt gespeichert ist.

Eine ACL mit all seinen ACEs kann bei Bedarf in der Datenbank gespeichert werden.

Methoden:

AddAce

AddAce speichert ein Ace-Objekt sortiert in einer einfach verketteten Liste ab. Die Instanzvariable „m_pAceList“ zeigt auf das erste Ace-Objekt in der Liste, ist die Liste leer, so wird auf NULL verwiesen.

CalcHash

CalcHash berechnet aus den momentan gespeicherten Ace-Objekten einen Hashwert. Der Hashwert wird in der Datenbank mit der ACL mitgespeichert und sollte möglichst eindeutig sein. Dies minimiert den Datenbankzugriff enorm.

FoundAce

Mit der Methode FoundAce wird in der Liste der gespeicherten Ace-Objekte nach einer Übereinstimmung gesucht.

Equal

Die Methode Equal vergleicht zwei Acl-Objekte auf Übereinstimmung. Zwei Acl-Objekte sind nur dann gleich, wenn sie denselben Hashwert haben, die Anzahl der gespeicherten Ace-Objekte und auch alle gespeicherten Ace-Objekte gleich sind.

GetNrOfAces

Liefert die Anzahl der gespeicherten Ace-Objekte zurück.

Reset

Löscht alle gespeicherten Ace-Objekte und stellt den Zustand nach der Acl-Objekt-Erzeugung wieder her.

StoreAcl

Berechnet zuerst den Hashwert aller gespeicherten Ace-Objekte. Anschließend wird die Datenbank überprüft, ob nicht dieselbe ACL schon gespeichert worden ist. Sollte dies nicht der Fall sein, so wird die ACL in der Datenbank abgespeichert. Jeder ACL-Eintrag in der Datenbank wird durch einen eindeutigen ACL-ID identifiziert. Dieser wird dann

mit sämtlichen Ace-Objekten in der Datenbank gespeichert. Somit ist ein nachträgliches Zusammensetzen der ACLs aus der Datenbank wieder möglich.
Die Überprüfung, ob die ACL schon in der Datenbank gespeichert ist, wird durch den Aufruf der privaten Methode „StoreAclToDB“ erledigt. StoreAclToDB selbst ruft im Falle der Speicherung die beiden privaten Methoden „StoreAclTable“ und „StoreAceTables“ auf.

3.6.1.4 Klasse AclObj

Die Klasse AclObj ist eigentlich keine Klasse im eigentlichen objektorientierten Sinn, da sie keine Methoden beinhaltet. Sie hätte wohl auch mit dem C-Konstrukt „struct“ implementiert werden können. AclObj wird jedenfalls von der Klasse Cache, siehe Kap. 3.6.1.5, benötigt. Jedes AclObj-Objekt hat einen Zeiger auf ein anderes AclObj-Objekt sowie auf ein Acl-Objekt gerichtet. Weiters wird noch der Acl-ID aus der Datenbank sowie der Hashwert eines Acl-Objekts gespeichert.

3.6.1.5 Klasse Cache

Im Kapitel 3.6.1.3 wurde bereits erwähnt, dass, bevor eine ACL in der Datenbank gespeichert wird, überprüft werden muss, ob die ACL nicht schon gespeichert ist. Geht man von mehreren 10.000 zu untersuchenden Objekten und Dutzenden Scanner-Instanzen, welche alle gleichzeitig in die Datenbank schreiben wollen, aus, so würde der Datenbank-Server sehr schnell zum Flaschenhals.
Das SAT-Team hat aus den Erfahrungen mit SAT 1 gelernt. Aus diesem Grund wurde ein Cache-Mechanismus implementiert. Caching hat allerdings nur für ACLs einen Sinn, das Caching von ADS-Objekten nicht. ADS-Objekte werden alle mit Versions-nummer gespeichert, eine doppelte Speicherung kann somit nicht vorkommen. Der Cache macht hingegen sehr wohl Sinn, wenn es um die Speicherung von ACLs geht. Die Überprüfung, ob eine ACL in der Datenbank schon gespeichert ist, reduziert sich in den meisten Fällen auf ein paar Zugriffe in den Hauptspeicher. Diese belasten aber das Netzwerk und somit auch den Datenbankserver nicht. Wird der Cache genügend groß gewählt, so kann eine Menge an Datenbankzugriffe eingespart werden. Lediglich wenn im Cache nichts gefunden wird, muss die Datenbank überprüft werden.

Da die Wahrscheinlichkeit, dass nach einer ACL gesucht wird, nach der erst kürzlich gesucht wurde, relativ hoch ist, wurde für das Caching (wie in den Vorgängerversionen) eine Move-to-Front List implementiert.
Die Funktionsweise ist sehr einfach: ACLs werden in einer einfach verketteten Liste gespeichert. Ausgehend vom Beginn der Liste werden die Elemente schrittweise durchwandert. Findet man eine Übereinstimmung mit einer bereits in der Liste gespeicherten ACL, so wird dieses Element aus der Liste ausgekettet und an den Anfang der Liste gestellt. Wird eine bestimmte Anzahl an im Cache gespeicherten ACLs erreicht, so wird das letzte Element der Liste gelöscht.

Methoden:

AddToCache

Der Methode AddToCache werden Acl-Objekte übergeben, diese werden im Cache gehalten. Für jedes Acl-Objekt wird intern ein AclObj-Objekt (siehe Kap. 3.6.1.4) erzeugt, welches einen Zeiger auf das Acl-Objekt beherbergt. Die AclObj-Objekte werden miteinander nach dem Move-to-Front Algorithmus verkettet.

FindInCache

Die Methode FindInCache wird dazu benötigt, den Cache nach einer bestimmten ACL zu durchsuchen. Wird diese dort nicht angetroffen, so muss vor der Speicherung der ACL die Datenbank durchsucht werden, ob diese nicht dort schon gespeichert ist.

3.6.1.6 Klasse AdsObject

Die Klasse AdsObject ähnelt insofern der Klasse AclObj, als dass auch sie keine Methoden besitzt. AdsObjekt wird für die Komprimierung von Objekten verwendet. Da während des Analysevorgangs nicht bekannt ist, welche Objekte eines Containers zusammengefasst werden können, muss man sämtliche Objekte zwischenspeichern, bevor sie komprimiert werden können. Ein einzelnes ADS-Objekt wird dabei in einem AdsObject-Objekt gespeichert. Jedes AdsObject-Objekt beinhaltet einen Zeiger auf ein anderes AdsObject.

3.6.1.7 Klasse AdsObjects

In Kap. 3.6.1.6 wurde bereits erklärt, dass bei der Komprimierung von Objekten diese zwischengespeichert werden müssen. Diese Aufgabe wird von der Klasse AdsObjects übernommen. AdsObjects speichert alle direkten Söhne eines Containers des Active Directory, welche in weiterer Folge komprimiert werden sollen. Jedes Objekt aus dem Active Directory wird dabei genau in einem AdsObject-Objekt gespeichert. Die AdsObject-Objekte werden sortiert in einer einfach verketteten Liste gespeichert. Somit ist es sehr schnell feststellbar, ob es überhaupt möglich ist Objekte zusammenzufassen.

Methoden:

Add

Objekte aus dem ADS, welche komprimiert werden sollen, werden mit der Methode Add übergeben. Jedes Objekt wird in einem AdsObject-Objekt abgespeichert.

GetCompressedObjects

Sobald alle Objekte aus dem Active Directory mittels der Methode Add gespeichert wurden, kommt die Komprimierung zum Einsatz. GetCompressedObjects komprimiert die gespeicherten AdsObject-Objekte gemäß Komprimierungsstufe 1 (siehe Kap. 3.5.2). Das durch die Komprimierung neu entstandene Objekt wird zurückgegeben und die zusammengefassten AdsObject-Objekte aus der verketteten Liste gelöscht. Um eine vollständige Komprimierung zu erhalten, muss die Methode GetCompressedObjects so oft aufgerufen werden, bis nichts mehr retourniert wird.

Die nächste ADS-Scanner Version wird zur Erhöhung der Speichereffizienz die beiden Schritte der Speicherung und Komprimierung von Objekten in einem einzigen Durchgang ausführen

3.6.1.8 Klasse Ads

Die umfangreichste Klasse ADS-Scanners ist die Klasse Ads. Die gesamte Logik des ADS-Scanners wurde hier implementiert. Das Kapitel beschreibt diese Klasse die wichtigsten Methoden. Auf den Programmablauf dazu wird im nächsten Kapitel eingegangen. Damit sollte es möglich sein, die Funktionsweise des ADS-Scanners nachzuvollziehen.

Die Klasse Ads erledigt zwei Aufgaben:

- Analyse des Schemas einschließlich der Extended Rights
- Analyse des Active Directory (inklusive Komprimierung)

Im Gegensatz zu den anderen bereits vorgestellten Klassen, welche auch vom NTFS-Scanner (siehe dazu [Ach]) verwendet werden, gilt dies für die Klasse Ads nicht. Es werden lediglich drei Methoden nach außen sichtbar gemacht (Konstruktor/Destruktor nicht mitgezählt), d.h. nur diese können von anderen Programmteilen aus aufgerufen werden. Diese drei Methoden (SetComputerID, SetVersion, Scan) reichen dazu auch völlig aus.

Methoden:

SetComputerID / SetVersion

Diese beiden Methoden müssen vor dem eigentlichen Scan aufgerufen werden. SetComputerID übergibt den ID jenes Rechners, auf dem die Analyse durchgeführt wird. Diese ID stammt aus der Computer/User-Analyse – siehe dazu [Zar].

Jeder Analysedurchgang bekommt durch das Controller-Programm eine eindeutige Nummer zugewiesen – die sogenannte Version. Diese Versionsnummer wird in der Datenbank mit jedem Objekt und jedem Objekttypen (ausgenommen bei den ACLs und ACEs) mitgespeichert. Durch die Versionierung der Scans wird es in Zukunft möglich Vergleiche anzustellen und somit Änderungen festzustellen. Dabei wird nicht überprüft, ob die Methode SetVersion aufgerufen wird. Sollte dies nicht geschehen, so wird in der Datenbank die Version 0 eingetragen.

Scan

Durch Aufruf der Methode Scan wird die eigentliche Analyse gestartet. Welche Teile des Active Directory analysiert werden sollen, wird durch das Controller-Programm festgelegt und in die Windows-Registrierung des jeweiligen Rechners gespeichert. Diese Information veranlasst die Methode Scan entsprechende Teile des Verzeichnisses zu durchlaufen. Die Objekt-Analyse wird zusätzlich unterteilt in

- **Analyse DomainNC Container:** Der Container DomainNC enthält für den Administrator wahrscheinlich den interessantesten Teil. In diesem Container be-

finden sich die Information über Organisationen, Organisationseinheiten, User, Drucker, usw.

- **Analyse Configuration Container:** Der Configuration Container und auch das Schema wurden deswegen in die Analyse aufgenommen, weil auch hier wichtige Information gespeichert wird und die dort gespeicherten Objekte ebenfalls mit ACLs gesichert sind. So befinden sich beispielsweise die Extended Rights (siehe Kap. 2.4.1.3) in diesem Container.

- **Analyse Schema Container:** Das Active Directory Schema befindet sich in diesem Container. Das Schema besteht aus Attributtyp- und Klassen-Definitionen.

Die Reihenfolge der Analyse (sofern alle Teile des Active Directory untersucht werden sollen) ist wie folgt:

1) Schema Analyse: Objekttypen und Extended Rights
2) DomainNC: Analyse der Berechtigungen
3) Configuration: Analyse der Berechtigungen
4) Schema: Analyse der Berechtigungen

Diese Reihenfolge ist durch den Scanner festgelegt, der die einzelnen Teile sequentiell durchläuft, es könnte jedoch genauso gut eine andere Reihenfolge gewählt werden. Durchaus denkbar wäre z.B. auch, dass ein Scanner insgesamt viermal gestartet wird und bei jedem Durchlauf einen anderen dieser vier Teile ausführt. Diese Unabhängigkeit der Reihenfolge ist wichtig, da Scanner auf verschiedenen Rechnern parallel laufen können. Damit sind Fälle wie z.B. dass auf einem Rechner bereits Daten aus dem Container DomainNC in die Datenbank eingetragen werden, bevor von einem anderen (eventuell langsameren) Rechner das Schema selbst in die Datenbank eingespielt wird.

StoreObjectTypes

Die Analyse des Schemas wird von der Methode StoreObjectTypes gesteuert. Gespeichert wird neben den Objekttypen auch die zu den einzelnen Objekttypen gehörige Standard-Security. Damit wird es bei der späteren Objekt-Analyse möglich zu überprüfen, ob ein Objekt eine gegenüber dieser Standard-Security geänderte ACL hat.

StoreObjects

Diese Methode bildet die Logik für die Objektanalyse. Das korrekte Durchlaufen des Active Directory und die gesamte Steuerung des Analysevorgangs wird hier festgelegt. Abfragen an bestimmten Stellen schlagen verschiedene Pfade für die vier Komprimierungsstufen ein, z.B. müssen Objekte für die Komprimierungsstufe 1 und 3 zwischengespeichert werden, für Stufe 2 jedoch nicht.

AnalyzeAcls

AnalyzeAcls untersucht, ob eine gegebene ACL einen Bruch der Standard-Security aus dem Schema verursacht. Dieser Bruch tritt genau dann auf, wenn die Berechtigungen aus dem Schema nicht identisch mit den Berechtigungen des Objekts (ohne Vererbung) sind.

EliminateGenericRights

Dieser Methode wird eine Access Mask übergeben. Um die Auswertung einfacher zu gestalten, werden vom ADS- und auch vom NTFS-/Registry-Scanner die Generic-Rights in die entsprechenden Berechtigungen aufgelöst. Eine genaue Beschreibung dieses Vorgangs findet man in Kapitel 3.6.3.3.

GetSDFromObject / GetSDFromSDDL

Beide Methoden haben eines gemeinsam: Sie lesen einen Security Descriptor (SD) aus. GetSDFromObject kommt bei der Objekt-Analyse zum Einsatz und liest dabei den Security Descriptor direkt aus dem zu analysierenden Objekt.

Anders sieht es jedoch beim Auslesen der Standard-Security aus dem Schema aus. Der Standard-SD wird nämlich in einer als Zeichenkette codierten Art und Weise bei den Schema-Objekten gespeichert. Dieses Microsoft eigene Format nennt sich Security Descriptor Definition Language (SDDL). Aus diesem SDDL-String kann ebenso ein SD gewonnen werden, was mit GetSDFromSDDL realisiert wird.

GetAclFromSD

Das SAT-System speichert – wie bereits bekannt – nicht den vollständigen SD, der ja auch die für SAT unwichtigen SACL beinhaltet. Daher wird der von GetSDFromObject bzw. GetSDFromSDDL ausgelesene SD in seine Bestandteile zerlegt, wobei nur der DACL weiter gespeichert wird. Diese Aufgabe erledigt ein Aufruf der Methode

GetAclFromSD. Innerhalb von GetAclFromSD wird zugleich überprüft, ob die extrahierte ACL die Vererbung unterbricht.

GetInhAcl

GetInhAcl extrahiert aus einer gegebenen ACL jene ACEs, welche durch Vererbung hinzugekommen sind. Dieser Vorgang wird benötigt, um festzustellen, welche ACEs direkt – sei es durch die Standard-Security oder durch den Administrator – vergeben wurden.

GetNamingContexts

Die Distinguished Names der zu analysierenden Container (DomainNC, Schema, Configuration) könnten sich rein theoretisch, wenn Microsoft Änderungen an ihrer Active Directory Implementierung vornimmt, ändern. Ein spezielles Objekt, genannt „RootDSE“ beinhaltet die Distinguished Names dieser Container. GetNamingContexts liest diese Werte, welche den Startpunkt für die Analyse darstellen, aus.

GetNrBase62

Für die komprimierten Pfade (Kapitel 3.5.5) wird ein Zahlensystem mit der Basis 62 verwendet. Die Umwandlung vom Dezimal in dieses System übernimmt die Methode GetNrBase62.

GetObjectCategory

Der Objekttyp eines Objekts wird mit dieser Methode ausgelesen.

GetObjectInfo

Sämtliche benötigte Attribute über ein bestimmtes ADS-Objekt werden von der Methode GetObjectInfo ausgelesen und zurückgegeben.

GetSchemaIDGUID

Der eindeutige GUID eines Objekttypen wird von der Methode GetSchemaIDGUID aus dem Schema ausgelesen.

IsContainer

Überprüft, ob ein Objekt ein Container ist (siehe dazu auch Kapitel 3.6.3.2).

NoAccessAcl

Generiert eine spezielle ACL, die anzeigt, dass der ADS-Scanner nicht die entsprechenden Berechtigungen hatte um die ACL auszulesen.

StoreAcl

Ein Aufruf von StoreAcl veranlasst die Speicherung einer ACL in der Datenbank. Zuvor wird jedoch noch der Cache nach einer Übereinstimmung überprüft.

WriteMembershipTableToDB / WriteObjTableToDB / WriteObjTypeTableToDB / WriteSidRefTableToDB

Diese drei Methoden veranlassen jeweils die Speicherung der dazugehörigen Tabelle: „membership“, „objects“, „objTypes“, „sidRef“ (siehe Kapitel 3.4).

3.6.2 Ablauf der ADS-Analyse aus Implementierungssicht

Nachdem im vorigen Kapitel die implementierten Klassen des ADS-Scanners behandelt und die wichtigsten Methoden derselben besprochen wurden, soll uns nun der Programmablauf des ADS-Scanners beschäftigen. Ziel dieses Kapitels ist, die Funktionsweise des ADS-Scanners zu verstehen und zu wissen, wie die einzelnen Klassen zusammenspielen.

Einstiegspunkt für einen Analyselauf ist die Klasse Ads. Sobald ein Objekt dieser Klasse erzeugt wird, liest der Konstruktor den lokalen Rechnernamen, d.h. jenen Rechnernamen, auf dem der Scanner läuft, aus. Dieser wird benötigt um Zugriff zum Active Directory auf diesem Rechner zu erhalten. Objekte eines Domain Controllers werden durch Vorausstellen von „LDAP://Servername/“ vor dem Distinguished Name angesprochen.

Nachdem vom Initiator des Analysevorgangs - üblicherweise das Controller-Programm – die Scan-Version mittels SetVersion und der Computer-ID mittels SetComputerID festgelegt wurden, kann die Analyse durch Aufruf der Methode Scan gestartet werden.

Bei der Analyse selbst unterscheiden wir zwischen Schema-Analyse und Objekt-Analyse. Ob und welche Teile der Analyse gestartet werden, wird aus der Registry ausgelesen – Details dazu siehe [Ach]. Bevor jedoch mit der Analyse begonnen wird, müssen zuerst die Distinguished Names aller zu analysierenden Container (DomainNC,

Configuration und Schema) aus dem RootDSE-Objekt ausgelesen werden, was mit einem GetNamingContexts auch geschieht.

Schema-Analyse

Die Analyse des Schemas wird durch den Aufruf von StoreObjectTypes initiiert. Der Zugriff auf Objekte des Active Directory geschieht in diesem Fall durch eine Suchanfrage an den Schema-Container, wobei als Suchfilter „*“ spezifiziert wird. Die Schema-Analyse soll alle Objekte des Schema-Containers erfassen und in die Datenbank schreiben. Ausgelesen werden neben dem Namen auch der GUID eines jeden Objekts und es wird festgestellt, ob es sich bei dem Objekt um ein Attribut oder einen Objekt-Typ handelt. Einem Attribut eines jeden Schema-Objekts wird der Standard-Security Descriptor– sofern vorhanden – im SDDL-Format entnommen. Dieser SDDL-String muss zuerst in eine brauchbare Form durch GetSDFromSDDL umgewandelt werden. Da nicht der ganze SD benötigt wird, muss noch der DACL mittels GetAclFromSD herausgefiltert werden, welcher dann in einem Objekt der Klasse Acl abgespeichert wird. Ein Aufruf von StoreAcl veranlasst den Cache Mechanismus (Klasse Cache) die soeben gewonnene ACL in den Cache aufzunehmen und sie in der Datenbank abzuspeichern. Die ACL ist nach Rückkehr vom Aufruf auf jeden Fall duplikatfrei in der Datenbank gespeichert und im Cache ist sie an vorderster Stelle zu finden. Der nächste Schritt in der Schema-Analyse ist das Speichern der Extended Rights. Eine neuerliche Suchabfrage wird auf den entsprechenden Container im Configuration Container gerichtet, somit werden alle Extended Rights retourniert. Die erhaltenen Objekte werden in derselben Tabelle objTypes in der Datenbank abgespeichert, wobei durch das Attribut „objectClass“ eine spätere Unterscheidung möglich ist. Der Hauptunterschied zu den Schema-Objekten ist jedoch das Fehlen der Standard-Security.

Objekt-Analyse

Weitaus komplizierter als die Schema-Analyse gestaltet sich die Objekt-Analyse. StoreObjects wird im besten Fall dreimal mit jeweils einem anderen Start-Container aufgerufen: DomainNC, Configuration und Schema.

Die Objekt-Komprimierung erforderte eine Rekursion, da ein iteratives Durchlaufen des Active Directory für die Komprimierung eher ungeeignet ist. An die Rekursion wird der

DN des zu untersuchenden Objekts/Containers sowie Information zum Vaterknoten (wie komprimierter Pfad, Objekt-ID, usw.) und die Komprimierungsstufe übergeben.
Die Rekursion läuft folgendermaßen ab: Gleich zu Beginn werden durch Aufruf von GetObjectInfo sämtliche Attribute des zu untersuchenden Objektes ausgelesen.
Der nächste Schritt dient dazu, die Security des Objekts zu analysieren. Dazu werden zuerst die Berechtigungen mit den Aufrufen GetSDFromObject und GetAclFromSD ausgelesen, wobei letzteres auch gleich einen eventuellen Bruch der Vererbung feststellt. Ein Bruch der Standard-Security wird durch AnalyzeAcls festgestellt. Die Methode GetInhAcl extrahiert jenen Teil der ACL, welcher durch Vererbung hinzugekommen ist. Diese „inherited-ACL“ wird später als Parameter der Rekursion übergeben, in der derzeitigen Implementierung aber nicht weiter verwendet. Den Abschluss der Security-Analyse einer ACL bildet das Abspeichern derselbigen durch den Aufruf von StoreAcl in der Datenbank. Die zurückgegebene ACL-ID wird benötigt um die Objektinformation in der Tabelle „objects“ abzuspeichern, was der nächste Schritt mit WriteObjTableToDB auch erledigt.
Die Rekursion endet an dieser Stelle, sollte IsContainer feststellen, dass es sich bei dem Objekt, welches gerade analysiert wird, um ein Blatt handelt. Für den Fall, dass es sich um einen Container handelt, wird dieser genauer untersucht. Dazu wird eine Suchabfrage auf die darunter liegende Baum-Ebene spezifiziert. Das Suchergebnis kann dann Objekt für Objekt durchwandert werden. Jedes Objekt, welches man in einem Container findet, wird geprüft, ob es sich um ein Objekt oder einen Container handelt. Wird ein Container gefunden, so wird die Rekursion (StoreObjects) erneut aufgerufen, wobei vorher ein neuer komprimierter Pfad berechnet wird. Anders sieht die Sache aus, wenn wir ein Objekt finden. Dann beginnt die Analyse, wie bereits zu Beginn erklärt, in der Reihenfolge:

- Objektinformation auslesen
- Security Descriptor auslesen
- DACL extrahieren und analysieren
- DACL speichern

Nun müssen noch die verschiedenen Komprimierungsstufen unterschieden werden:

- **Komprimierungsstufe 0:** das Objekt wird sofort abgespeichert, da ohne Komprimierung auch keine Zwischenspeicherung nötig ist.
- **Komprimierungsstufe 1**: die gefundenen Objekte müssen für die Komprimierung zwischengespeichert werden. Dies wird durch die Klasse AdsObjects implementiert. Der gesamte Container wird durchlaufen und die Objekte werden gemäß den Regeln für die Komprimierungsstufe 1 (Objekte mit gleichem Typ und gleicher Security) zusammengefasst.
- **Komprimierungsstufe 2**: das Objekt wird dann sofort abgespeichert, wenn die Vererbung unterbrochen oder die Standard-Security geändert wurde.
- **Komprimierungsstufe 3**: kombiniert Komprimierungsstufe 1 und 2. Es wird eine Komprimierung gemäß Stufe 1 durchgeführt (Objekte mit gleichem Typ und gleicher Security), gespeichert werden dann allerdings nur jene Objekte, welche die Vererbung bzw. die Standard-Security brechen.

3.6.3 Problemlösungen

Dieses Kapitel beschreibt einzelne Fragen, welche während der Implementierungsphase auftraten, und deren Lösung. Es handelt sich hier um Sonderfälle, welche in der Literatur nicht oder zu wenig behandelt wurden bzw. vorerst einfach übersehen wurden.

3.6.3.1 SID-History

Eines der ersten Probleme, die auftraten, war das leidige Thema mit der sogenannten SID-History. Die Planung ging davon aus, dass eine SID eindeutig innerhalb einer Domain ist, was auch korrekt ist. Wir gingen jedoch weiter davon aus, dass jedes Objekt, welches eine SID haben kann - also die sogenannten Security Principals – auch nur eine SID hat.

Das Format, wie sich SIDs berechnen, hat sich mit dem Erscheinen von Windows 2000 geändert. Diese Tatsache alleine wäre noch nicht beunruhigend, geschweige denn ein Problem. Um die Abwärtskompatibilität zu gewährleisten führte Microsoft einen „Kompatibilitätsmodus“ ein, wodurch man z.B. in ein Windows 2000 Netzwerk Windows NT 4 Server integrieren kann. Windows 2000 Domain-Controller verhalten sich dabei wie die Primary Domain Controller (PDC) bzw. Backup Domain Controller (BDC) von Windows NT. Der native Modus von Windows 2000 sieht dieses Konzept

nicht mehr vor, es gibt hier nur mehr Domain Controller. Zählt man nun zwei und zwei zusammen, so haben wir mit Windows 2000 neue SIDs, aber auch einen Kompatibilitätsmodus zu Windows NT. Wird nun z.B. ein User von einer Windows NT Domäne in das Windows 2000 Active Directory transferiert, so wird sich der SID des Users ändern. Somit hätte der User auf einen Schlag alle seine bisherigen Berechtigungen verloren. Damit dies nicht passiert, bedient sich Microsoft eines kleinen Tricks und führte die SID-History ein, worin alle alten SIDs, welche ein Security Principal hatte, gespeichert werden. Diese SID-History wird in einem mehrwertigen Attribut in den dazugehörigen Active Directory Objekten gespeichert.

Für die Analyse selbst macht es keinen Unterschied, ob dieses Attribut ausgelesen wird oder nicht, für die Auswertung sehr wohl. Für die Auswertung ist immer die Benutzersicht interessant, daher die Fragestellung: Wo hat Benutzer X überall Rechte? Ein einzelner Benutzer wird daher als eine Sammlung von SIDs gesehen, welche u.a. alle SIDs der Gruppen, zu denen der Benutzer gehört, beinhaltet. Zu dieser SID-Sammlung gehört auch die SID-History, deswegen ist es wichtig, dass sie ausgelesen und gespeichert wird. Der Zusammenhang, welche SID zu welchen anderen SIDs gehört, wird in der Tabelle „membership“ in der Datenbank gespeichert (siehe dazu Kapitel 3.4.5).

Ausgelesen wird die SID-History mit den anderen Objekt-Attributen. Es wird dabei einfach geprüft, ob es für das aktuelle Objekt das Attribut „sidHistory“ gibt und wenn dieses einen Wert hat, wird er in die Datenbank geschrieben.

3.6.3.2 Objekt oder Container?

Betrachten wir das Dateisystem NTFS, so gibt es nur zwei Arten von „Objekten“:

- Verzeichnisse (Ordner)
- Dateien

SAT 1 analysiert „nur“ die NTFS-Laufwerke mehrere Rechner. Die verschiedenen Komprimierungsstufen wurden ursprünglich auch für das NTFS entworfen. Die Portierung auf das Active Directory brauchte ein paar zusätzliche Überlegungen, welche aber sehr stark an das bereits bestehende Konzept aus der NTFS-Analyse anknüpfen sollte. Das gesamte Projekt-Team war sich aber in einem Punkt einig: das Active Directory besteht aus Containern und Objekten, wobei eine Gemeinsamkeit zum Filesystem gesehen wurde. Auch dort finden wir Container (=Verzeichnisse) und Objekte (=Dateien).

Auf dieser „Erkenntnis“ aufbauend wurde die Komprimierung in Angriff genommen. Im NTFS bedeutet Komprimierungsstufe 1 das Zusammenfassen von Dateien, die sich im selben Verzeichnis befinden und die gleichen Berechtigungen und Dateitypen aufweisen. Dieses Konzept wurde auf das Active Directory umgelegt. Anstatt die Dateien mit gleichem Dateityp zusammenzufassen wird versucht Objekte mit gleichen Objekttypen nach Möglichkeit zu komprimieren.

Zur besseren Veranschaulichung wurde die Komprimierung anhand von User-Objekten durchgespielt, d.h. gleiche User-Objekte in einer Organisationseinheit werden komprimiert, wenn sie die gleichen Berechtigungen haben

In der Implementierungsphase des Projekts wurde – als die Implementierung der Komprimierung auf dem Programm stand - als erstes eine Funktion benötigt, die feststellt, ob ein vorliegendes Objekt ein Container oder ein Objekt ist (siehe IsContainer, Kapitel 3.6.1.8). Dazu wurde das Attribut „allowedChildClasses“ ausgelesen. Wenn dieses Attribut existiert und einen Wert beinhaltet, dann musste es sich um einen Container handeln.

Nach dem erste Probelauf des ADS-Scanners wurde jedoch festgestellt, dass fast ausnahmslos alle Objekte des Active Directory Container sind und somit andere Objekte aufnehmen könnten. Dies gilt interessanterweise auch für User-Objekte, von denen eigentlich angenommen wurde, dass es sich um Objekte und nicht um Container handelt.

Die Probleme, welche sich durch diese Tatsache ergaben waren - aus Sicht der Komprimierung - enorm. Eine einfache Beschränkung, wie: Man fasse alle Objekte zusammen, wenn sie dieselben Berechtigungen haben, reichte hier nicht aus. Was passiert, wenn eines dieser Objekte Wurzel eines Teilbaums ist?

Aus diesem Grund musste eine neue Definition für “Container-Objekt“ erschaffen werden. Der ADS-Scanner sieht derzeit ein Objekt genau dann als Container, wenn das Objekt andere beinhaltet, also selbst Wurzel für einen Teilbaum ist. Realisiert wurde dies, indem die Methode IsContainer für ein gegebenes Objekt eine Suche, nach allen darin befindlichen Objekten absetzt. Liefert diese Suche die leere Menge, so handelt es sich um ein Objekt, andernfalls um einen Container. Komprimierungsstufe 1 darf nur dann Objekte zusammenfassen, wenn die zusammenzufassenden Objekte keinen Container nach dieser neuen Definition beinhalten.

3.6.3.3 Generic Rights

Die Windows 2000 Access Mask beinhaltet vier Bits, welche die sogenannten „Generic Rights“ bilden. Diese Generic Rights stellen eine Zusammenfassung von mehreren Standard-Rechten dar, welche aber je nach Einsatzgebiet unterschiedlich gedeutet werden. Dies macht es für die Auswertung um so schwerer, da dort unterschieden werden müsste, ob es sich bei dem Objekt, welches gerade betrachtet wird, um ein NTFS-Objekt, Registry-Objekt oder ein ADS-Objekt handelt. Daher wurden die Generic Rights – sofern sie überhaupt gesetzt waren – durch ihre entsprechenden Standard-Rechte ersetzt. Abb. 37 fasst zusammen, durch welche Berechtigungen die entsprechenden Generic Rights im Active Directory ersetzt wurden.

GENERIC READ:	ADS_RIGHT_READ_CONTROL ADS_RIGHT_DS_LIST_OBJECT ADS_RIGHT_DS_READ_PROP ADS_RIGHT_ACTRL_DS_LIST
GENERIC WRITE	ADS_RIGHT_READ_CONTROL ADS_RIGHT_DS_WRITE_PROP ADS_RIGHT_DS_SELF
GENERIC_EXECUTE	ADS_RIGHT_READ_CONTROL ADS_RIGHT_ACTRL_DS_LIST
GENERIC_ALL	ADS_RIGHT_DS_CREATE_CHILD ADS_RIGHT_DS_DELETE_CHILD ADS_RIGHT_ACTRL_DS_LIST ADS_RIGHT_DS_SELF ADS_RIGHT_DS_READ_PROP ADS_RIGHT_DS_WRITE_PROP ADS_RIGHT_DS_DELETE_TREE ADS_RIGHT_DS_LIST_OBJECT ADS_RIGHT_DS_CONTROL_ACCESS ADS_RIGHT_DELETE ADS_RIGHT_READ_CONTROL ADS_RIGHT_WRITE_DAC ADS_RIGHT_WRITE_OWNER

Abb. 37: Auflösung der Generic Rights

3.6.4 Ausblick

Die erste Implementierungsphase des ADS-Scanners - Erstellung eines lauffähigen Prototyps - kann als abgeschlossen betrachtet werden. Die Arbeiten am SAT-Projekt - natürlich auch am ADS-Scanner - gehen aber dennoch weiter. Zahlreiche Ver-

besserungen befinden sich bereits in Planung und werden die Effizienz des Systems weiter steigern.
Als Beispiel für künftige Erweiterungen sei auf die sogenannten Owner-SIDs verwiesen, welche in der derzeitigen Analyse noch nicht berücksichtigt werden. Der Besitzer eines Objekts wird von Windows in Form einer (Owner-)SID im dazugehörigen Security Descriptor abgespeichert. Während der Planungsphase wurde überlegt, diese SID mitzuspeichern. Aus Implementierungssicht ergeben sich zwei Probleme, abhängig davon, ob ein Owner-SID in der Datenbanktabelle „Acl“ oder „Objects“ gespeichert wird:

1) Owner-SID in die „Acl“-Tabelle: Die Anzahl der in der Datenbank gespeicherten ACLs und Objekte steigt. Objekte, welche die gleichen Berechtigungen, aber unterschiedliche Owner-SIDs haben, würden dann – aus Datenbanksicht - eine andere ACL besitzen.

2) Owner-SID in die „Objects“-Tabelle: Die Komprimierung (Stufe 1 und 3) wird nicht mehr so effizient arbeiten, da nur mehr Objekte mit gleichen Berechtigungen und gleichem Owner-SID zusammengefasst werden könnten.

Für die nächste Version des ADS-Scanners ist bereits geplant, ein zusätzliches Feld „Owner-SID“ in die Tabelle „Objects“ einzubinden. Um dennoch die bestmögliche Komprimierung zu erreichen, wird die Owner-Analyse als eigene Option angeboten werden.

Wie dieses Beispiel gezeigt hat ist sich das SAT-Team sehr wohl mancher „Schwächen“ bewusst. Das Datenbankdesign wird in zukünftigen Versionen ebenso überarbeitet und verbessert werden (neue Tabellen für bestimmte Teile, siehe dazu auch Kapitel 3.4). Einige Kompromisse mussten auch aus Zeitgründen bei der Implementierung eingegangen werden.

4 Fallstudie - Active Directory Services

Die vorausgegangen Kapitel haben den Leser sowohl in die Theorie der Verzeichnisdienste als auch in die Planung und Implementierung des ADS-Scanners eingeweiht. Der nun folgende letzte Teil dieser Arbeit beschäftigt sich mit konkreten Testläufen des ADS-Scanners, welche unter bestimmten Bedingungen vorgenommen wurden, sowie deren Auswertungen.

4.1 Szenarien

Die verschiedenen „Testaufbauten“ sollen sowohl verschiedene Aspekte des Active Directory als auch des ADS-Scanners hervorheben.
In den vorausgegangen Kapiteln wurden Verzeichnisdienste im Allgemeinen, und das Active Directory im Speziellen aus verschiedenen Blickwinkeln betrachtet. Alles in allem fehlt aber noch eine wesentliche Information: Es ist mit dem derzeitigen Wissensstand nicht möglich eine Abschätzung über die Größe eines Verzeichnisses abzugeben. Sprechen wir von 100 Schema-Objekten oder von 10.000? Wie viele Einträge hat ein typisches Verzeichnis?
An mehreren Stellen in der Arbeit wurde darauf hingewiesen, dass Programme ihre Daten im Active Directory speichern können und die Sicherheitsmechanismen des Active Directory verwenden können. Auch hier kann man keine Aussage treffen, wie viele Objekte dem Verzeichnis hinzugefügt werden und wo Änderungen gemacht werden. Verschiedene Szenarien wurden aufgebaut und durch den ADS-Scanner analysiert. Diese Testläufe sollten mehrere Erkenntnisse liefern:

- die Effizienz der Komprimierung
- die Größe des Active Directory
- Änderungen durch Drittprogramme am Active Directory

Anzumerken sei an dieser Stelle noch, dass die hier präsentierten „Messdaten“ lediglich Momentaufnahmen sind, deren Ergebnisse direkt von der installierten Software (IIS, Service Packs, ...) abhängen. Ziel dieser Fallbeispiele ist es aber, nicht exakte Zahlen zu liefern, sondern vielmehr Tendenzen zu erkennen und die Komprimierung zu testen.

Ausgangsbasis

Für die Testläufe standen zwei Rechner zur Verfügung

- Rechner 1: Pentium 2 / 400 Mhz / 256 MB RAM
- Rechner 2: AMD Athlon / 1200 Mhz / 512 MB RAM

Beide Rechner waren über ein 100 MBit Ethernet miteinander verbunden.

Auf Rechner 1 wurde eine Standardinstallation des Windows 2000 Advanced Server inklusive Service Pack 2 durchgeführt. Der zweite Rechner fungiert als Datenbank-Server mit SQL Server 2000.

Für die Tests stand kein Server, welcher sich im täglichen Einsatz befindet, zur Verfügung, daher beziehen sich alle gemachten Angaben auf einen frisch aufgesetzten Server.

In allen Szenarien wurden vier ADS-Scanner Durchläufe – jeweils mit Komprimierungsstufe 0, 1, 2 und 3 – vorgenommen. Nach Beendigung der Testläufe wurde der Windows 2000 Server (SP2) neu installiert. Somit hatten alle Test-Szenarien dieselbe Ausgangsbasis. Insgesamt wurden zwölf Testläufe des ADS-Scanners in drei Szenarien absolviert.

Szenario 1

Der erste Test diente dazu, um ein neu aufgesetztes und quasi „unbenutztes“ Active Directory zu analysieren. Dieser Test hatte eine sehr wichtige Funktion. Durch die gewonnene Information dieses Tests war es erst möglich, Rückschlüsse - betreffend den Umfang der Änderungen von Drittprogrammen - zu ziehen. Weiters sollte sich auch zeigen, ob die Verwendung der Komprimierung entsprechende Erfolge mit sich bringt und eine Reduzierung der Datenmenge ermöglicht. Der ADS-Scanner hatte hier vier Durchgänge mit jeweils Komprimierung 0, 1, 2 und 3 zu absolvieren.

Szenario 2

Das erste Produkt, welches getestet wurde, war Microsofts Exchange 2000 Enterprise Server mit dem zusätzlich installierten Active Directory Connector (ADC). Der ADS-Scanner sollte hier die Änderungen, welche der Exchange-Server am Active Directory vornahm, aufzeigen. Auch hier wurden drei Testläufe mit allen vier Komprimierungsstufen durchlaufen. Nach Abschluss der Tests wurde der Rechner neu aufgesetzt.

Szenario 3

Für Szenario 3 wurde das Produkt Microsoft Internet Security & Acceleration Server 2000 (ISA), Enterprise Edition installiert. Auch hier waren die Ausmaße der Änderungen, welche Microsofts ISA-Server am Active Directory vornahm, interessant.

4.2 Auswertungen und Resultate

Im vorigen Kapitel wurden die verschiedenen Szenarien vorgestellt, welche praktisch durch den ADS-Scanner gestestet wurden. Dieses Kapitel präsentiert nun die Ergebnisse dieser Testläufe. Betrachten wir zunächst die Ergebnisse pro analysiertem Szenario:

Szenario 1

Die Analyse eines neu installierten, noch leeren Active Directory fördert folgende Details zutage:

Komprimierungsstufe	*0*	*1*	*2*	*3*
AD-Objekte gesamt (DB)	2424	164	238	77
davon Bruch Standard Security	238	77	238	77
davon mit Bruch Vererbung	8	6	8	6
Schema-Objekte gesamt (DB)	1056	1056	1056	1056
davon Klassen	142	142	142	142
davon Attribute	863	863	863	863
davon Extended Rights	49	49	49	49
verschiedene ACLs gesamt (DB)	64	64	64	64
bestehen aus ACEs:	632	632	632	632
verschiedene ACLs im ADS	41	41	28	28
bestehen aus ACEs:	504	504	340	340
verschiedene ACLs Standard Security	25	25	25	25
bestehen aus ACEs:	148	148	148	148

Abb. 38: Auswertung Szenario 1

Insgesamt finden sich in den drei Containern „DomainNC“, „Configuration“ und dem „Schema“ 2424 Objekte, wobei rund 43% - 1056 Objekte - auf das Schema entfallen. Auffallend ist die relative hohe Anzahl von Brüchen der Standard-Security. Alleine 238 Objekte bekommen scheinbar vom System selbst einen anderen als die im Schema spezifizierte Standard-Security. Insgesamt benötigen die 2424 Objekte lediglich 41 ver-

schiedene ACLs (bestehend aus 504 ACEs), was auffallend wenig ist. Im Durchschnitt besteht damit eine ACL aus 9,9 ACEs.

Betrachtet man die Zahlenwerte für das Schema, so kann man sofort sehen, dass mehr als 80% der 1056 Schema-Objekte Attribute (863) sind. 142 verschiedene Objekttypen dürften auch für exotische Anwendungen eine hohe Auswahl bieten. Fast die Hälfte der Gesamt-Objekte eines neu aufgesetzten Active Directory entfallen auf das Schema.

Die 142 verschiedenen Objektklassen brauchen immerhin 25 verschiedene ACLs (bestehend aus 148 ACEs) für die Spezifikation ihrer Standard-Security.

Interessant erscheinen auch die 49 verschiedenen Extended Rights, welche von Microsoft quasi mitgeliefert werden.

Die Komprimierung erweist sich als sehr effizient. Von den ursprünglichen 2421 Objekten bleiben lediglich 6,76% (Stufe 1) bzw. 3,17% (Stufe 3) übrig. Dies ist natürlich auf die leicht mögliche Komprimierung des Schemas zurückzuführen.

Szenario 2

Microsofts Exchange Server 2000 war eines der ersten Produkte am Markt, welches das Active Directory verwendete. Aus diesem Grund waren die Auswertung von großem Interesse.

Komprimierungsstufe	*0*	*1*	*2*	*3*
AD-Objekte gesamt	2624	179	240	79
davon Bruch Standard Security	240	79	240	79
davon mit Bruch Vererbung	8	6	8	6
Schema-Objekte gesamt	1244	1244	1244	1244
davon Klassen	151	151	151	151
davon Attribute	1024	1024	1024	1024
davon Extended Rights	49	49	49	49
verschiedene ACLs gesamt	70	70	70	70
bestehen aus ACEs:	661	661	661	661
verschiedene ACLs im ADS	46	46	30	30
bestehen aus ACEs:	529	529	348	348
verschiedene ACLs Standard Security	26	26	26	26
bestehen aus ACEs:	152	152	152	152

Abb. 39: Auswertung Szenario 2

Exchange 2000 Server erweitert das Active Directory um ~200 Objekte, wobei der Großteil auf Schema-Erweiterungen zurückzuführen ist.

Interessant erscheint auch die Tatsache, dass keine neuen Extended Rights eingeführt werden.

Szenario 3

Für das dritte und letzte Szenario wurde der Internet Security & Acceleration 2000 Server (ISA), Enterprise Edition installiert.

Komprimierungsstufe	*0*	*1*	*2*	*3*
AD-Objekte gesamt	3418	538	242	82
davon Bruch Standard Security	242	82	242	82
davon mit Bruch Vererbung	12	10	12	10
Schema-Objekte gesamt	1372	1372	1372	1372
davon Klassen	257	257	257	257
davon Attribute	1064	1064	1064	1064
davon Extended Rights	79	79	79	79
verschiedene ACLs gesamt	71	71	71	71
bestehen aus ACEs:	686	686	686	686
verschiedene ACLs im ADS	46	46	31	31
bestehen aus ACEs:	549	549	371	371
verschiedene ACLs Standard Security	27	27	27	27
bestehen aus ACEs:	156	156	156	156

Abb. 40: Auswertung Szenario 3

Der ISA-Server nahm die meisten Änderungen am Active Directory vor (~1000 neue Objekte). Die Änderungen betrafen jedoch weniger das Schema als das Hinzufügen von neuen Verzeichnis-Objekten.

Vergleich der Szenarien

Die Effizienz der Komprimierung wird in Abb. 41 sehr deutlich hervorgehoben. Besonders deutlich ist dies nach Installation des ISA 2000 Server zu sehen, die Komprimierung (Stufe 3) lässt von 3418 Objekten lediglich nur mehr 82 Objekte übrig (entspricht 2,4%). Diese Reduzierung der Datenmenge bedeutet eine gute Erfüllung der Vorgaben. Die Datenbank bleibt klein, kompakt und übersichtlich. Informationen von geringem Interesse werden erst gar nicht gespeichert. Die Visualisierung kann sich auf die wichtigsten Informationen, welche kurz und prägnant präsentiert werden, konzentrieren. Die Komprimierung ist somit ein wichtiger Schritt in diese Richtung.

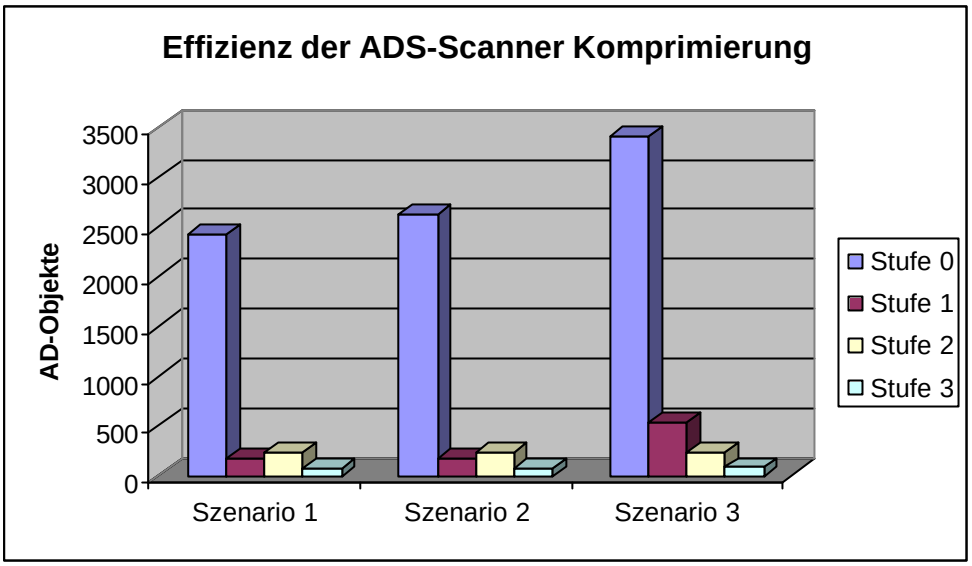

Abb. 41: Komprimierung im Vergleich

Die Vorteile der Komprimierung liegen auf der Hand: Weniger Daten, daher mehr Übersicht. Man darf jedoch auch nicht die Nachteile übersehen. Durch die Komprimierung entsteht auch ein nicht unwesentlicher Informationsverlust. Abb. 42 zeigt, wie durch das „Wegkomprimieren" von Objekten auch die Anzahl der voneinander verschiedenen ACLs schrumpft. Dies bedeutetet einen nicht unwesentlichen Informationsverlust. Vor allem wenn man davon ausgeht, dass durch den Verlust von ACLs (in der Datenbank) wichtige sicherheits-relevante Information verloren geht. Fairerweise muss man auch dazusagen, dass sowohl Komprimierungsstufe 2 als auch 3 eher dazu dient, um auf die „Hotspots" hinzuweisen. Dies heißt, dass jene Objekte des Active Directory hervorgehoben werden sollen, welche einen Bruch in der Standard-Security bzw. in der Vererbung verursachen.

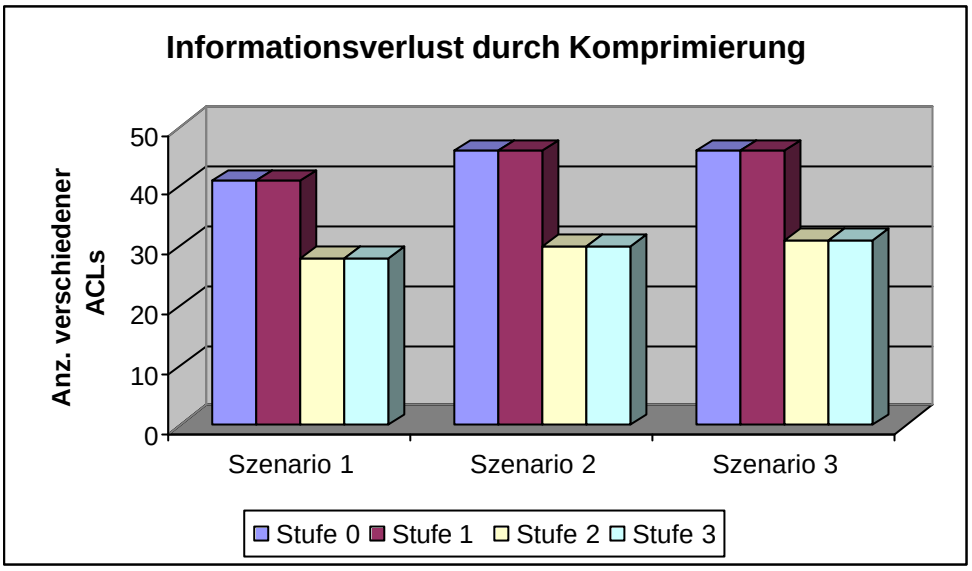

Abb. 42: Komprimierungsstufen 2 und 3 – hoher Informationsverlust

Keineswegs überraschte die Zusammensetzung des Schemas: Mehr als 80% der Schema-Objekte sind Attribute, welche ja die elementarsten Bausteine für die Objektklassen bilden. Die Vielfalt der Objektklassen (142) lässt erahnen, dass hier mit sehr viel Voraussicht geplant wurde.

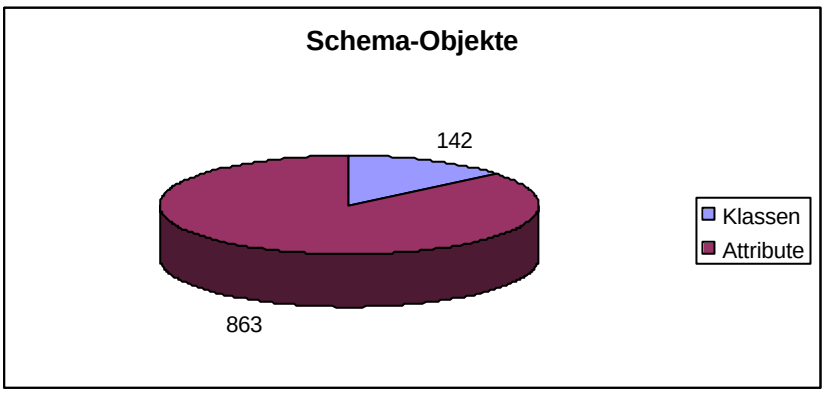

Abb. 43: Schema-Bestandteile, Szenario 1

4.3 Konstruierte Testläufe

Der abschließende Teil dieser Arbeit zeigt die Ergebnisse der ADS-Analyse anhand zweier konstruierter Beispiele als „Differenzanalyse“. Somit konnten die erwarteten Änderungen durch das Analyse-Programm verifiziert werden.

Beispiel 1: Test der Komprimierungsstufe 1

Die Zielvorgabe für das erste Beispiel war es, die Komprimierung (Stufe 1) zu verifizieren und deren Effizienz anhand eines praktischen Beispiel zu zeigen. Für den Test wurde die Organisationseinheit „OU1“ generiert, in welcher verschiedene User, Organisationseinheiten und Gruppen (Local, Global, Universal) angelegt wurden.

Abb. 44 zeigt die Ausgangsbasis für den Testlauf in Form eines Auszugs aus der Standard-Administrationsoberfläche.

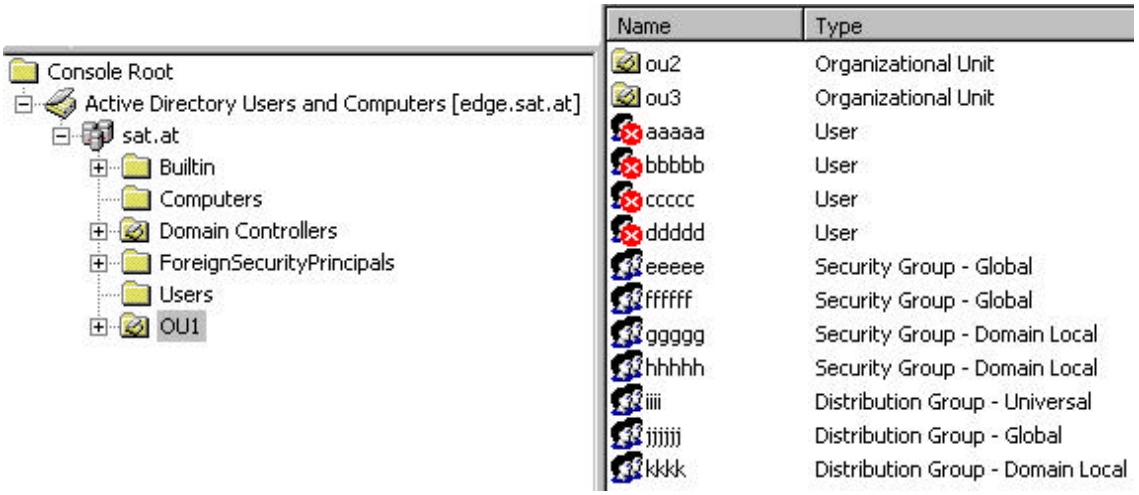

Abb. 44: Testaufbau - Beispiel 1

Von einem Analysedurchgang mit Kompressionsstufe 1 darf man erwarten, dass alle Objekte mit gleichem Objekttyp zusammengefasst werden. Ein Blick in die Datenbank (siehe Abb. 45) verifiziert diese Annahme:

obj_id	objName	c	objGuid	amount	objTypeGuid	parent	path	date	acl_id	breaksInh...	breaksStdSec...
56	OU1	1	{E7...	1	{BF967A...	1	a-0-5	...	29	0	0
57	eeeee	1	{D9...	7	{BF967A...	56	a-0-5-0	...	4	0	0
58	ou2	1	{87...	2	{BF967A...	56	a-0-5-1	...	29	0	0
59	aaaaa	1	{81...	4	{BF967A...	56	a-0-5-2	...	8	0	0

Abb. 45: Analysedaten - Komprimierungsstufe 1

Die Organisationseinheit „OU1“ beinhaltet insgesamt 13 Objekte (2 OU-, 4 User- und 7 Gruppen-Objekte). Die Komprimierung reduziert diese letztendlich auf 3 Einträge in der Datenbank. Interessanterweise werden auch die verschiedenen Gruppentypen (Global, Local, Universal) zusammengefasst.

Beispiel 2: „Delegate Control“

“Delegate Control“ entstand aus der Notwendigkeit heraus, einzelnen Benutzern oder Gruppen (z.B. Abteilungsleiter) die Möglichkeit zu geben, einen Teil des Verzeichnisses (z.B. eine Abteilung) selbst zu administrieren. Der Administrator kann bestimmte Befugnisse an Personen oder/oder Personengruppen weiterreichen – sprich delegieren.

Für den eigentlichen Test wurden zuerst eine Analyse des DomainNC-Containers durchgeführt, um eine Ausgangsbasis für einen Vergleich zu erhalten. Anschließend wurde eine Organisationseinheit „OU1“ (siehe Abb. 46) mit 3 Usern angelegt.

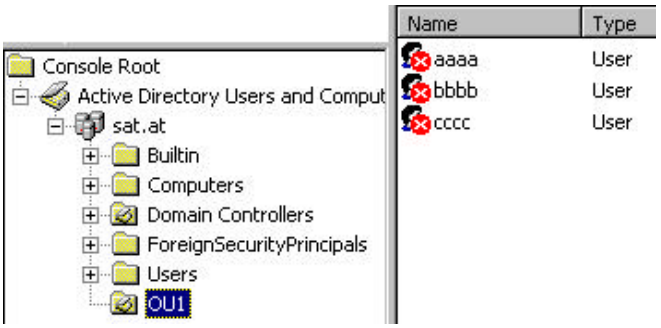

Abb. 46: Testaufbau – Beispiel 2

Diese Organisationseinheit erfuhr ein „Delegate Control“, wobei an eine Gruppe (z.B. Abteilungsleiter) alle zur Auswahl stehenden Aufgaben (siehe Abb. 47) delegiert wurden. Im Anschluss wurde ein weiterer Analysedurchgang gestartet, um die Änderungen in der Datenbank nachvollziehen zu können.

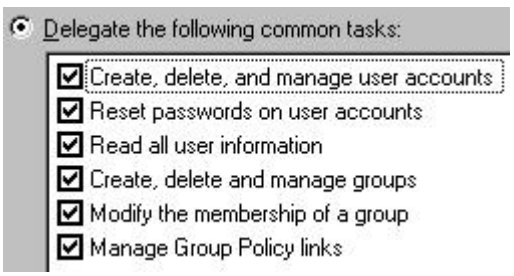

Abb. 47: Delegate Control

Aus Implementierungssicht passiert bei „Delegate Control“ folgendes:

- Mehrere ACEs werden zur ACL des Objekts „OU1“ hinzugefügt und nach „unten“ weiter vererbt

Als Konsequenz zu den obigen Überlegungen müsste gelten:

- Die Standard-Security des Objekt „OU1“ wird gebrochen.
- Die Standard-Security der User-Objekte wird nicht gebrochen, da zusätzliche Berechtigungen (ACEs) nur durch die Vererbung hinzu kommen.

Vergleicht man nun die Ergebnisse der Analysedurchgänge vor und nach dem „Delegate Control“, so scheinen sich die Annahmen zu bestätigen:

Delegate Control	*nein*	*ja*
AD-Objekte gesamt	121	121
davon Bruch Standard Security	89	90
davon mit Bruch Vererbung	8	8
verschiedene ACLs im ADS	28	29
bestehen aus ACEs:	445	488

Abb. 48: Anzahl der ACLs bei einem "Delegate Control"

Die absolute Kontrolle bietet ein Blick in die Datenbank (Abb. 49). Die Organisationseinheit „OU1“ weist einen Bruch in der Standard-Security auf, die „darunter“ liegenden User-Objekte nicht.

obj_id	objName	c	objGuid	amount	objTypeGuid	parent	path	date	acl_id	breaksInh...	breaksStdSec
117	RID Set	1	{F02...	1	{7BFDCB...	114	a-0-7-0-1	...	23	0	0
118	OU1	1	{BF9...	1	{BF967A...	1	a-0-8	...	28	0	1
119	aaaa	1	{7FB...	1	{BF967A...	118	a-0-8-0	...	29	0	0
120	bbbb	1	{548...	1	{BF967A...	118	a-0-8-1	...	29	0	0
121	cccc	1	{6FA...	1	{BF967A...	118	a-0-8-2	...	29	0	0

Abb. 49: Ausschnitt aus der Datenbank nach einem "Delegate Control"

4.4 Ausblick

Die Auswertungen haben gezeigt, dass auch bei einer Erweiterung des Active Directory damit zu rechnen ist, dass die Komprimierung weiter effizient arbeitet. Wie sich die einzelnen Komprimierungsstufen in der Praxis bewähren, kann zum jetzigen Zeitpunkt noch nicht genau vorhergesehen werden.
Diese Arbeit behandelt nur einen Teil des gesamten SAT-Systems. Die NTFS-Analyse [Ach] verwendet ebenfalls verschiedene Komprimierungsstufen, welche aber unter Umständen eine andere Bedeutung haben. In der Auswertung und Visualisierung [Zar] sind noch zusätzliche Komprimierungsstufen vorgesehen, auch hier kann man ohne konkrete Tests nur Vermutungen über die Effizienz anstellen.

An dieser Stelle soll auch noch kurz darauf hingewiesen werden, dass die beiden angeführten Arbeiten [Ach] und [Zar] derzeit noch im Implementierungsstadium sind. Die entsprechenden schriftlichen Arbeiten dazu werden voraussichtlich im 1. Quartal 2002 verfügbar werden.

5 Literaturverweise

[Ach]: Achleichter, M.: „Analyse der Windows 2000 Rechtestruktur in NTFS und Registry“; Diplomarbeit, Joh. Kepler Universität Linz (erscheint voraussichtlich 1Q 2002)

[Bor]: Borenstein, N., Freed, N.: „MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies”; RFC 1521, Sept. 1993

[Cha]: Chadwick, D.: “Understanding X.500 - The Directory”; Chapman & Hall 1994; ISBN 185 0322 813;
http://www.isi.salford.ac.uk/staff/dwc/Version.Web/Contents.htm

[Fer]: Ferris, D.: „A Tutorial on X.500“
http://www.ferris.com/rep/1997090101/default.asp

[Gol]: Gollmann, D.: „Computer Security“; John Wiley & Sons 1999;
ISBN 0 471 97844 2

[Han]: Hanner, K.: „Analyse und Archivierung von Benutzerberechtigungen in einem Windows NT Netzwerk“; Diplomarbeit, Joh. Kepler Universität Linz 1998

[Hal]: Haller, N.; Metz, C.: „A One-Time Password System“; RFC 1938, Mai 1996

[Has]: Hassler, V.: “LDAPv3 versus X.511 DAP security: A comparison and how to sign LDAPv3 operations”;
http://www.infosys.tuwien.ac.at/Staff/vh/papers/

[Hoe]: Hörmanseder, R; Hanner, K.: "Managing Windows NT file system permissions (A security tool to master the complexity of Microsoft Windows NT file system permissions)"; Academic Press, Journal of Network and Computer Applications (1999) 22; Seite 119 ff.

[Ian]: IANA: Internet Assigned Numbers Authority
http://www.iana.com/

[Ibm]: IBM: “IBM SecureWay Directory”
http://www-4.ibm.com/software/network/directory/

[Ietf]: Internet Engineering Task Force: “RFCs”
http://www.ietf.org/rfc.html

[Iso]: ISO (International Organisation for Standardization)
http://www.iso.ch

[Itu1]: ITU (International Telecommunication Union)
http://www.itu.net

[Itu2]: ITU (International Telecommunication Union): “X.500 Standard”;
http://www.itu.int/rec/recommendation.asp?type=products&parent=T-REC-x

[Jav]: Gosling, J.; Joy, B.; Steele, G;. Bracha, G.: „Java Language Specification, Second Edition“
http://java.sun.com/docs/books/jls/second_edition/html/interfaces.doc.html#238680

[Kb]: Microsoft Knowledge Base: “Definition of the Windows Registry”
http://support.microsoft.com/support/kb/articles/Q256/9/86.ASP

[Lang]: Langenscheidt: „Online Fremdwörterbuch“;
http://www.langenscheidt.de/deutsch/index.html

[Ldap]: Johner, H., Brown L., Hinner, F., Reis, W., Westmann, J.: „Understanding LDAP“; IBM Redbook 1998; http://www.redbooks.ibm.com

[Lin]: Linn, J.: “Generic Security Service Application Program Interface”; RFC 1508, Sept. 1993

[Kir]: Kirkpatrick, G.; „Active Directory Programming - The Authoritative Solution”; SAMS Publishing 2000; ISBN 0-672-31587-4

[Kle]: Klensin, J.; Catoe, R.; Krumviede, P.: „IMAP/POP AUTHorize Extension for Simple Challenge/Response“; RFC 2195, Sep. 1997

[Kpmg]: KPMG International: “E-fraud: Is technology running unchecked?”; http://www.kpmg.com/about/press.asp?cid=469

[Mey]: Meyers, J.: “Simple Authentication and Security Layer (SASL)”; RFC 2222, Oct. 1997

[Msr]: Microsoft Research, UK, Cambridge; http://www.research.microsoft.com/labs/cam.asp

[Ms]: Microsoft Press: „Microsoft Windows 2000 Server: Verteilte Systeme. Die technische Referenz“; Microsoft Press 2000; ISBN: 3-86063-273-6

[Moc]: Mockapetris, P.: „Domain Names – Implementation and Specification”, RFC 1035, Nov. 1987

[Ope1]: The OpenLDAP Foundation http://www.openldap.org/foundation/

[Ope2]: The OpenLDAP Foundation: “OpenLDAP” http://www.openldap.org

[Ope3]: The OpenLDAP Foundation: “OpenLDAP 2.0 Administrator’s Guide”
http://www.openldap.org/doc/admin/

[Spe]: Spealman, J.: „Microsoft Windows 2000 Active Directory Services“; Microsoft Press 2000; ISBN 0-7356-0999-3

[Stei]: Steiner, J.; Clifford Neuman, B.; Schiller, J.: „Kerberos: An Authentication Service for Open Network Systems”; Aus: “Proceedings of the Winter 1988 Usenix Conference”; Feb. 1988
ftp://athena-dist.mit.edu/pub/kerberos/doc/usenix.txt

[Sto1]: Stokes, E.; Byrne, D.; Blakley, B.; Behera, P.: “Access Control Requirements for DLAP”; RFC 2820, Mai 2000

[Sto2]: Stokes, E.; Blakley, B.; Byrne, R.; Huber, R.; Rinkevich, D.: “Access Control Model for LDAPv3 <draft-ietf-ldapext-acl-model-08.txt>”; Internet-Draft, Juni 20001
http://www.ietf.org/internet-drafts/draft-ietf-ldapext-acl-mode-08.txt

[Wah1]: Wahl, M., Howes, T., Kille, S.: „Lightweight Directory Access Protocol (v3)“, RFC 2251, Dez. 1997

[Wah2]: Wahl, M., Coulbeck, A., Howes, T., Kille, S.: „Lightweight Directory Access Protocol (v3): Attribute Syntax Definition“, RFC 2252, Dez. 1997

[Wah3]: Wahl, M.: „ A Summary of the X.500(96) User Schema for use with LDAPv3“, RFC 2256, Dez. 1997

[Web1]: Webopedia: Online Computer Dictionary for Internet Terms and Technical Support
http://www.webopedia.com

[Web2]: Webopedia: Online Computer Dictionary for Internet Terms and Technical Support; "X.500 Definition"
http://www.webopedia.com/TERM/X/X_500.html

[Web3]: Webopedia: Online Computer Dictionary for Internet Terms and Technical Support; "LDAP Definition"
http://webopedia.internet.com/TERM/L/LDAP.html

[Zar]: Zarda, G.: „Visualisierung von Rechtestrukturen in Windows 2000 Netzwerken"; Diplomarbeit, Joh. Kepler Universität Linz (erscheint voraussichtlich 1Q 2002)

6 Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

.....

Helml Thomas

7 Lebenslauf

Persönliche Daten

Name	Thomas Helml
E-Mail	thomas@helml.com
Geboren am	1.6.1975 in Vöcklabruck, OÖ
Eltern	Helml Hubert, geb. 1948, Schlosser
	Helml Irene, geb. 1954, Hausfrau
Geschwister	Helml Karin, geb. 1978, Studentin
	Helml Sabine, geb. 1983, Schülerin
Tochter	Fuchs Julia, geb. 1999
Familienstand	ledig

Schulausbildung

1981-1985	Volksschule Ampflwang i.H.
1985-1989	Hauptschule Ampflwang i.H.
1989-1993	Bundesoberstufenrealgymnasium Ried i.I., Ausbildungszweig Informatik, Abschluss: Matura (Wahlfach: Informatik)
1993-1994	Präsenzdienst, Schwarzenbergkaserne Salzburg

Studium

1994-1995	Studium der Mechatronik an der Johannes Kepler Universität Linz
1995-2001	Studium der Informatik an der Johannes Kepler Universität Linz, Wahlfachgruppe: „Network Security“
2001	Mitarbeit am Projekt SAT 2 (FIM, J.K. Universität Linz) Erstellung der Diplomarbeit: „Darstellung von Rechtestrukturen in Verzeichnisdiensten (am Beispiel Active Directory)“ im Zuge des Projekts SAT 2