Schmutztitel

Andreas Haupt

Aufbau einer Kerberos5-Infrastruktur in einer OpenAFS Umgebung

Diplomarbeit

Aufbau einer Kerberos5-Infrastruktur in einer OpenAFS Umgebung

(OpenAFS Migration auf Heimdal-Kerberos5)

Diplomarbeit

im Studiengang

Wirtschaftsinformatik
Technische Fachhochschule Wildau
Fachbereich Wirtschaftsinformatik – Betriebswirtschaft

Andreas Haupt

Matrikelnummer: 99 277 0352
Seminargruppe: I2/99

Erstbetreuer: Prof. Dr. Christian Müller
Zweitbetreuerin: Prof. Dr. Ulrike Tippe

Angefertigt in der Zeit vom 11. April 2003 bis 10. Juli 2003

Zeuthen, den 26. Februar 2004

Inhaltsverzeichnis

Abbildungsverzeichnis

Prolog

Aufgabe dieser Diplomarbeit ist es aufzuzeigen, wie am DESY Zeuthen die Migration von OpenAFS als Authentisierungsinstanz zur Kerberos5 Implementation Heimdal von statten ging. Zusätzlich wird in den ersten beiden Kapiteln der theoretische Hintergrund von Kerberos und AFS dargestellt. Es werden die beiden großen Implementationen von Kerberos5 vorgestellt, welche Unterschiede zwischen beiden existieren und warum Heimdal den Vorzug bekam, eingesetzt zu werden. Desweiteren wird ein Rückblick auf die Fähigkeiten von Kerberos4 gegeben und die Vorteile seines Nachfolgers dargelegt.

Auch wenn es nicht zu meiner Aufgabenstellung gehört, werde ich versuchen aufzuzeigen, ob und wie Windows (2000/XP) in eine heterogene Kerberos5 Umgebung eingebettet werden könnte. Am DESY Zeuthen werden allerdings nur die Unixsysteme umgestellt. Das betrifft neben Linux nur noch Solaris. Andere Plattformen werden hier nicht mehr unterstützt.

Zum Zeitpunkt dieser Arbeit ist Heimdal bei Version 0.6 angekommen, OpenAFS liegt in Version 1.2.9 vor. Daß Heimdals Versionsnummer mit einer 0 beginnt, zeigt, daß es sich noch stark in der Entwicklungsphase befindet und dementsprechend noch nicht alle Features zu 100% implementiert sind. Welche davon hier zu Problemen führten, wird auch in Kapitel 3 erläutert. Trotzdem ist Heimdal zum jetzigen Zeitpunkt, als stabil und für den Produktiveinsatz geeignet anzusehen.

Danksagung

Ich möchte an dieser Stelle einigen Personen meinen Dank aussprechen, ohne die ich die Migration wohl nicht so schnell geschafft hätte. Zum ersten Wolfgang Friebel, der dieses Thema überhaupt erst vorgeschlagen hat und bei Verständnisfragen immer mit Rat und Tat zur Seite stand. Desweiteren möchte ich Waltraut Niepraschk danken, die mich auf viele Probleme aufmerksam gemacht hat und von der ich eine Menge praktischer Hilfe, besonders unter Solaris, bekommen habe. Zu guter letzt hat mich Fatima Streit bei Kompilations- und Linkerproblemen sowie Karin Pipke bei LaTeX-Problemen immer wieder auf den Pfad der Tugend zurück gebracht.

Im Zuge dieser Diplomarbeit wurde ich an das Rechenzentrum der italienischen Univer-

sität CASPUR[1] eingeladen. An dieser Stelle möchte ich Andrei Maslennikov noch einmal meinen Dank dafür ausdrücken.

[1]Consorzio interuniversitario per le Applicazioni di Supercalcolo per Università e Ricerca (http://www.caspur.it)

Kapitel 1

Kerberos

1.1 Was ist Kerberos

Als das Internet in den 70er Jahren des letzten Jahrhunderts von den US-amerikanischen Militärs entwickelt wurde, ahnte niemand, welche Verbreitung es eines Tages haben würde. Da der Nutzerkreis in diesen Anfangstagen stark eingeschränkt war, wurden Sicherheitsmechanismen auf das allernötigste beschränkt. Doch mit der Zeit drängten immer mehr Nutzer in das Netz. Mit ihnen wuchs die Menge der sensiblen Daten, die über das Internet ausgetauscht wurden. Hacker hatten ein relativ leichtes Spiel, sie mitzulesen oder sogar zu modifizieren. So wurden z.B. Paßwörter mitgeschnitten und zum Einbruch in fremde Systeme genutzt. Desweiteren vertrauten bei vielen Client/Server Applikationen die Server den Clients „einfach so“, was die Identität der Nutzer anging. Es war somit ziemlich einfach, dem Server eine andere als die eigene Identität „vorzugaukeln“.

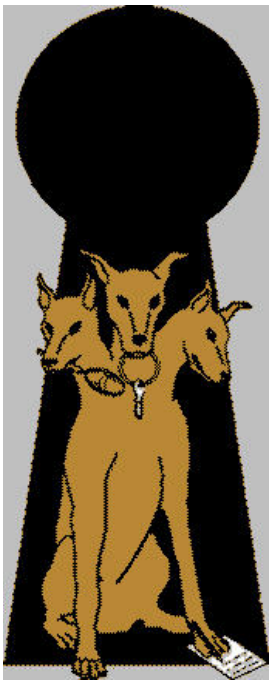

Abbildung 1.1: Der dreiköpfige Hund Cerberus (© MIT)

Bei den Geschädigten entstand der Ruf nach einer höheren Sicherheit. Nach und nach wurden Verschlüsselungstechnologien entwickelt und zunehmend eingesetzt. Im Zuge dessen entstand darauf aufbauend am MIT[1] ein Protokoll, welches die zwei zuvor beschriebenen Probleme aus der Welt schaffen konnte – Kerberos. Während die Versionen 1 - 3 nur intern am MIT genutzt wurden, um das Projekt „Athena“ zu sichern, verbreitete sich die Version 4 über viele Sites auf der ganzen Welt. Durch deren hohen Verbreitungsgrad wurden einige Einschränkungen und fehlende Features gefunden. In Version 5 sind diese Defizite behoben worden (siehe auch [4]). Letztendlich wurde das Kerberos Protokoll in seiner aktuellen Version im September 1993 in RFC 1510[5] standardisiert.

[1] Massachusetts Institute of Technology

Der Name Kerberos stammt aus der griechischen Mythologie. Er bezeichnet den dreiköpfigen Hund, der den Eingang zum Hades[2] bewachte. Da in der lateinischen Sprache der Buchstabe „K“ normalerweise nicht genutzt und durch den Buchstaben „C“ ersetzt wurde, ist heute der Name „Cerberus“ gebräuchlicher. Für die Namensgebung des Authentisierungsprotokolls wurde jedoch die originale griechische Schreibweise herangezogen (Vgl. [3]).

Kerberos ist ein Netzwerkauthentisierungsprotokoll. Es wurde entworfen, um in hohem Maße Sicherheit bei der Authentisierung zwischen Client und Server durch das Nutzen von Secret-Key Verschlüsselung zur Verfügung zu stellen. Das Kerberos Protokoll basiert auf der Annahme, daß Nachrichten über ein unsicheres Netzwerk übertragen und somit abgehört, modifiziert und von Angreifern wiederholt gesendet werden können. Die Hosts selber, auf denen kerberisierte Netzwerkdienste laufen, gelten jedoch als sicher.

1.2 Terminologie

Zum Verständnis des theoretischen Teils sind einige Begriffserklärungen erforderlich:

Principal: ist ein eindeutiger Eintrag in der Kerberos Datenbank. Ein Principal kann einen Nutzer (Account), einen Host oder einen Netzwerkdienst repräsentieren.

Realm: alle Hosts, die unter der selben zentralen Administration stehen und auf denen die selben Principals verfügbar sind, werden zusammengefaßt als Realm bezeichnet. Der Name einer Realm ist im Prinzip frei wählbar, es wird jedoch dringend empfohlen, den Namen der eigenen Internet Domain – nur in Großbuchstaben – zu nehmen. Die beiden Namen lassen sich so leicht unterscheiden. Am DESY Zeuthen mit dem Domainnamen *ifh.de* heißt die Kerberos Realm *IFH.DE*. Es ist zu beachten, daß bei DNS Domainnamen keine Unterscheidung zwischen Groß- und Kleinschreibung gemacht wird – bei Realmnamen jedoch schon.

Key Distribution Center: ist der Kerberos Server, der die Schlüssel aller Principals einer oder mehrerer Realms verwaltet.

Ticket: Mit Hilfe eines Tickets kann ein Principal seine Identität gegenüber einem Dienst beweisen.

Keytab Datei: Serviceprincipals bekommen üblicherweise einen zufälligen Schlüssel zugeordnet – also keinen der auf einem Paßwort basiert. Damit ein Dienst für ihn bestimmte Nachrichten entschlüsseln kann, wird sein geheimer Schlüssel in einer Keytab Datei abgelegt. Diese sollte nur für den Dienst selbst lesbar sein.

[2] in der griechischen Mythologie die Bezeichnung für die Unterwelt

1.3 Die Funktionsweise von Kerberos

1.3.1 Das Kerberos Modell

Kerberos wurde entwickelt, Netzwerkdiensten die sichere Identifikation ihrer Clients in einem unsicheren Netz zu erlauben. Um dies zu erreichen, muß ein Client einen Drei-Parteien-Nachrichtenaustausch initiieren. Er kann somit seine Identität einem einen Dienst anbietenden Server beweisen. Für diese Authentisierung benutzt der Client ein Ticket. Es enthält die Namen der beteiligten Principals (den des Clients und den des Dienstes bei dem er sich authentisieren will), einen Session Key (über den die Kommunikation zwischen Client und Server verschlüsselt wird) und andere Informationen wie z.B. die Gültigkeitsdauer des Tickets. Dieses Service Ticket fordert der Client von einer dritten Partei an, der sowohl der Client als auch der Server vertrauen: dem KDC.

1.3.2 Starke Kryptographie mit Kerberos

Kerberos benutzt für die Verschlüsselung symmetrische Kryptoverfahren. Diese haben den Vorteil, die Kodierung und Dekodierung von Nachrichten, im Vergleich zu asymmetrischen Verfahren, sehr viel schneller zu bewerkstelligen. Das spart Rechenzeit auf den Systemen.

Der Unterschied zwischen beiden Verfahren ist vereinfacht folgender: Bei symmetrischer Verschlüsselung besitzen beide Kommunikationspartner den selben geheimen (also nur den beiden Parteien bekannten) Schlüssel, der sowohl für die Kodierung als auch für die Dekodierung genutzt wird. Asymmetrische Verfahren nutzen für die Ver- und Entschlüsselung zwei unterschiedliche Schlüssel, die jedoch zusammen erzeugt wurden. Der erste ist öffentlich und für mögliche Kommunikationspartner frei verfügbar. Der zweite ist geheim und nur dem Empfänger der Nachrichten bekannt. Möchte eine Partei A eine geheime Botschaft an einen Empfänger B schicken, chiffriert A die Nachricht mit dem öffentlichen Schlüssel von B. Diese kann dann nur mit dem privaten Teil des Schlüsselpaars (den nur B besitzt) dechiffriert werden.

Unter Kerberos4 werden Nachrichten grundsätzlich mit DES[3] verschlüsselt. Dieser gilt jedoch aufgrund seiner relativ kurzen Schlüssellänge von 64 Bit (von denen jedoch nur 56 zur Chiffrierung genutzt werden) seit längerer Zeit als unsicher. Bei Kerberos5 wurden die Verschlüsselungsroutinen in selbständige Module ausgelagert. Dies erlaubt dem Programmierer, die verwendeten Verschlüsselungsalgorithmen auszutauschen. Damit die Kommunikationspartner den verwendeten Algorithmus erkennen können, enthalten die Nachrichten ein Feld, das diesen eindeutig identifiziert. Die einzelnen Schlüssel für die verschiedenen Chiffrierungsalgorithmen werden natürlich auch in der Kerberos Datenbank vorgehalten.

[3] Data Encryption Standard

1.3.3 Der Ticket-Nachrichtenaustausch im Detail

Möchte ein Client C einen Dienst S auf einem Host nutzen, stellt C eine Ticketanfrage an das Key Distribution Center. Darin enthalten sind die Namen von C und S. Nach dem Erhalt dieser Anforderung überprüft das KDC, ob beide Principalnamen gültig und in der Kerberosdatenbank vorhanden sind. Trifft beides zu, erstellt das KDC ein Ticket mit dem Namen von C und S, der IP-Adresse von C (C_{ADDR}), der aktuellen Zeit T, der Lebensdauer des Tickets L sowie einem Zufallsschlüssel, der zum Session-Key von C und S wird ($K_{C,S}$). Das komplette Ticket sieht wie folgt aus (Vgl. [6]):

$$T_{C,S} = (C, S, C_{ADDR}, T, L, K_{C,S})$$

Es wird mit dem Principalschlüssel des Dienstes (K_S) aus der Kerberosdatenbank verschlüsselt. Auf diese Weise ist sicher gestellt, daß nur S das Ticket lesen kann, da er als einziger ebenfalls den Schlüssel kennt. Als Antwort an C wird neben jenem chiffrierten Ticket der Name von S, die aktuelle Zeit T, die Lebensdauer des Tickets L auch der Session-Key $K_{C,S}$ zusammengefügt. Vor dem Zurücksenden kodiert das KDC sie aber noch mit dem Principalschlüssel von C (K_C):

$$\{S, T, L, K_{C,S}, \{T_{C,S}\}K_S\}K_C$$

Der Client C kann nun die Antwort mit Hilfe seines Principalschlüssels K_C, der z.B. von seinem Paßwort abgeleitet ist, die Antwort entschlüsseln. Er erhält somit S, T, L und $K_{C,S}$. Es besteht für ihn aber keine Möglichkeit $T_{C,S}$ zu entschlüsseln, da es mit dem für ihn unbekannten Principalschlüssel des Dienstes kodiert ist. All diese Daten werden im Credentials- (oder Ticket-) Cache für eine spätere Wiederbenutzung aufgehoben.

Bevor C eine Authentisierungsanfrage an S schickt, wird von ihm noch ein Authenticator A_C generiert. Dieser enthält den Namen von C, seine IP-Adresse C_{ADDR}, die aktuelle Zeit T und eine Prüfsumme PS. Durch das Einfügen des Zeitstempels sollen Replay-Attacken[4] vermieden werden, jedoch sind sie innerhalb eines Zeitfensters zumindest unter Kerberos4 nicht ausgeschlossen. A_C wird vor dem Senden an den Client mit $K_{C,S}$ verschlüsselt:

$$A_C = \{C, C_{ADDR}, T, PS\}K_{C,S}$$

Geimeinsam mit dem kodierten Ticket $\{T_{C,S}\}K_S$ wird der Authenticator an den Dienst S geschickt. Er kann nun mit seinem Principalschlüssel das Ticket $T_{C,S}$ entschlüsseln. Dabei erhält S unter anderem den Session-Key $K_{C,S}$, mit dem der Authenticator dekodiert wird. S vergleicht nun die Inhalte von A_C und $T_{C,S}$. Stimmen die Werte überein und liegen die Zeiten T nicht mehr als einen kleinen Betrag (meist fünf Minuten) auseinander, ist C für den Dienst S authentisiert. In Abbildung 1.2 ist der erläuterte Ablauf graphisch zu sehen.

Der Dienst kann nun noch eine mit dem Sitzungsschlüssel chiffrierte Botschaft (oft den um eins erhöhten Zeitstempel, der im Authenticator steht) an den Client zurückschicken. Durch diese Vorgehensweise wird eine gegenseitige Authentisierung erreicht. Der Client selbst kann sich so sicher sein, mit dem „richtigen“ Dienst zu kommunizieren. Der Session-Key ist, wie schon beschrieben, nur über den geheimen Principalschlüssel des Dienstes erhältlich.

[4] die Pakete werden von einem Angreifer im Netz mitgelesen und von ihm erneut gesendet

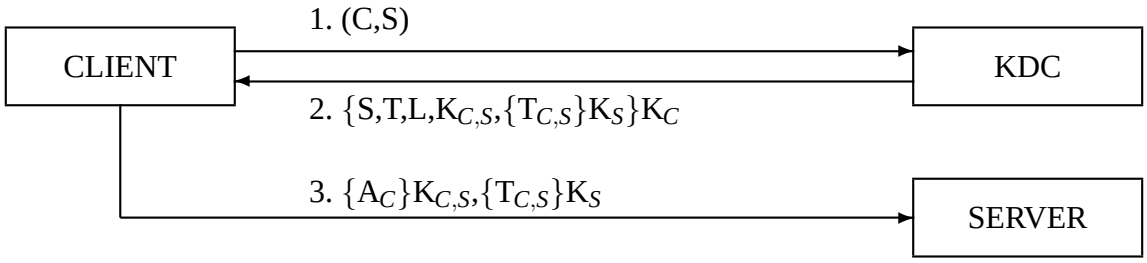

Abbildung 1.2: Nachrichtenaustausch bei der Authentisierung mit Kerberos (nach [6])

1.3.4 Der Ticket Granting Service

Der eben unter 1.3.3 beschriebene Ablauf ist jedoch nur der erste Schritt. Würde auf diese Weise vorgegangen, müßte ein Nutzer für jede Dienstanforderung sein Paßwort eingeben oder das Zwischenspeichern seines Schlüssels (oder Paßworts) wäre notwendig. Dies ist jedoch nicht erstrebenswert. Stattdessen wird der erwähnte Vorgang dazu genutzt ein spezielles Ticket – das Ticket Granting Ticket (TGT) – zu erhalten. Es ist ein Ticket für den Ticket Granting Service (TGS) und kann für alle weiteren Service Ticket Anfragen genutzt werden. Aus diesem Grund ist es dem Client möglich, den geheimen Schlüssel K_C nach Erhalt des TGS zu verwerfen. Die Chance eines Diebstahls wird somit verringert.

Das KDC besteht aus zwei Teilprozessen, die beide auf dem selben Host laufen und auf die selbe Datenbank zugreifen: dem Authentication Server (AS) sowie dem Ticket Granting Server (TGS). Der erste wird vom Client dazu genutzt, ein TGT zu erhalten. Weitere Anfragen für Service Tickets gehen an den TGS. Die Antworten des TGS sind dann mit dem Session-Key aus dem TGT verschlüsselt. Dies ermöglicht das Löschen von K_C, da der Client nur $K_{C,S}$ aufbewahren muß. Die erhaltenen Service Tickets können nun zur Authentisierung an Netzwerkdiensten wie z.B. einem Mailserver genutzt werden. Die nun vollständige Authentisierung ist in Abbildung 1.3 anschaulich dargestellt.

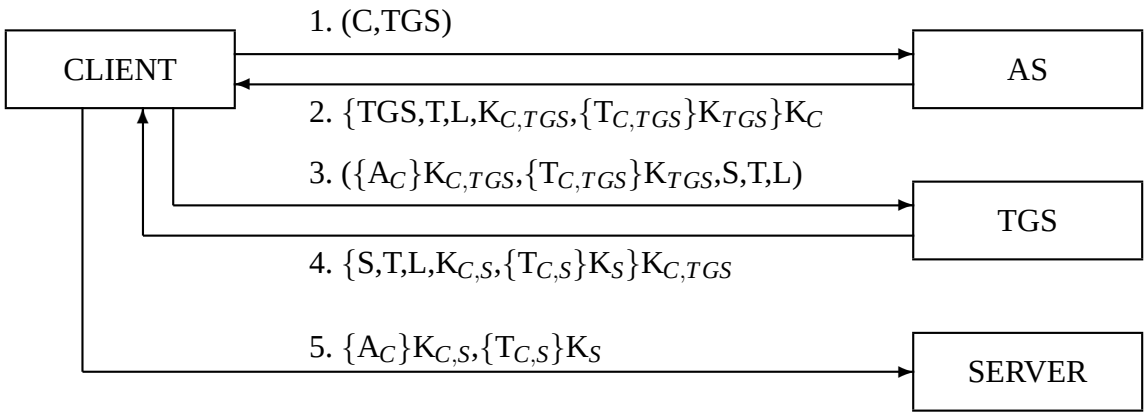

Abbildung 1.3: Nachrichtenaustausch zum Erhalt eines TGT und eines Service-Tickets (nach [6])

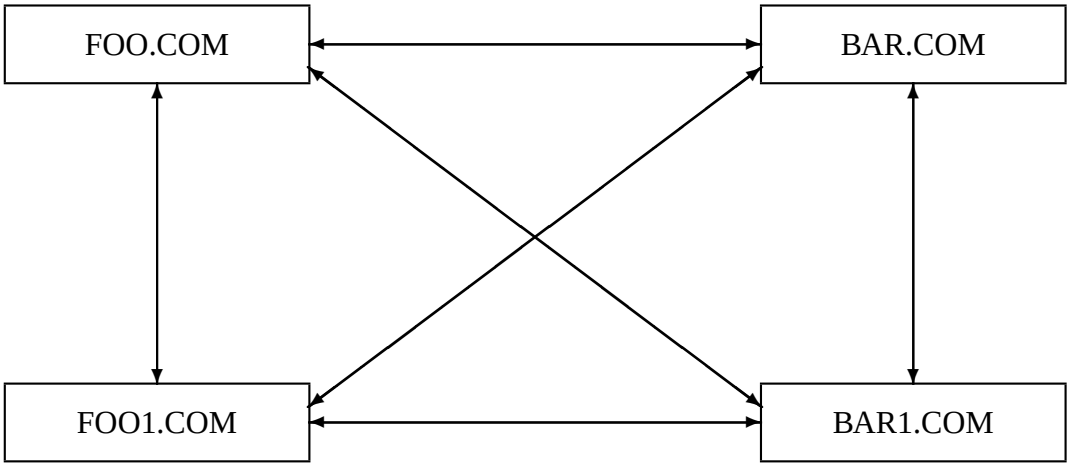

Abbildung 1.4: Cross-Realm Authentisierung unter Kerberos4

1.3.5 Cross-Realm Authentisierung

Es ist für einen Client, der in einer Realm A angemeldet ist, möglich, sich bei einem Dienst, der in einer Realm B läuft, zu authentisieren. Dies muß jedoch von den Administratoren in beiden Realms eingerichtet werden. Dazu ist ein Cross-Realm Schlüssel zu generieren, der in den Datenbanken von Realm A und B hinterlegt wird. Über diesen Schlüssel ist es möglich, alle verfügbaren Dienste gegenseitig in Anspruch zu nehmen.

Sollen zwischen mehreren Realms Authentisierungen erlaubt sein, sind zwischen allen Paaren Schlüssel auszutauschen (Abbildung 1.4). Die Zahl der Schlüssel steigt somit quadratisch zur Anzahl der beteiligten Realms.

1.3.6 Paßwort-zu-Schlüssel Konvertierung

In der Kerberos Datenbank sind nur z.B. DES Schlüssel gespeichert. Ein Mensch ist jedoch nicht in der Lage, sich solche langen alphanumerischen Reihen zu merken. Sie benutzen Paßwörter stattdessen. Um aus einem Paßwort einen Schlüssel zu generieren, gibt es eine „String-to-Key“ Methode. Dies ist eine Einwegfunktion. Das heißt, daß es nicht mehr möglich ist, aus dem generierten Schlüssel das Originalpaßwort zu extrahieren[5].

Diese Methode wird bei jeder TGT Authentisierung sowie bei jeder Paßwortänderung genutzt, den geheimen Schlüssel zu erzeugen. Unter Kerberos4 kommt zur Schlüsselgenerierung nur das Paßwort zum Einsatz:

K_{V4} `= String-To-Key(Paßwort)`

[5] Außer natürlich über eine Brute-Force Attacke

1.3.7 Schlüsselversionsnummern

Da es natürlich möglich ist, Principalschlüssel zu ändern, kann es bei der Authentisierung zu Problemen kommen. Hält ein Client ein Service Ticket für einen bestimmten Dienst im Ticket Cache und ändert dieser Dienst währenddessen seinen Schlüssel, wäre er nicht mehr in der Lage, das Ticket, welches ihm der Client schickt, zu entschlüsseln.

Aus diesem Grund gibt es Schlüsselversionsnummern (oder Key Version Numbers – kvno). Bei einer Schlüsseländerung wird der alte nicht aus der Datenbank entfernt. Zusätzlich zu diesem wird nun der neue Schlüssel mit einer um eins höheren Schlüsselversionsnummer ebenfalls abgespeichert. Zusammen mit dem Ticket schickt das KDC immer die kvno des verwendeten Schlüssels. Der Server kann mit dem alten das Ticket somit immer noch lesen. Jedoch werden neue Tickets mit dem Schlüssel mit der höchsten Schlüsselversionsnummer ausgestellt.

1.4 Kerberos4 vs. Kerberos5

1.4.1 Generelle Schwächen im Kerberos Protokoll

Das Kerberos Protokoll hat mit seinem Versionsschritt viele Verbesserungen erfahren. Außerdem sind viele störende Einschränkungen der Version 4 in Kerberos5 behoben worden. Ein sehr gute Zusammenstellung darüber ist [1], worauf ich mich zu einem großen Teil beziehe. Auch in [4] ist die Entwicklung des Protokolls übersichtlich dargelegt. Die Evolution erfolgte in Zusammenarbeit mit vielen Nutzern und Administratoren von Kerberos4, die ihre Wünsche mit einfließen ließen. Das Ergebnis ist der Anforderungskatalog für Kerberos5 RFC 1510[5]. Diese Verbesserungen in der Nachfolgeversion sind die Hauptmotivation für die Umstellung des Authentisierungsschemas auf Kerberos5 am DESY Zeuthen.

Wie schon erwähnt, wurde Kerberos entwickelt, das Athena Projekt zu sichern. Aufgrund der dort vorhandenen Infrastruktur kam Kerberos4 als Netzwerkauthentisierungsprotokoll zum Einsatz, das die Identität eines Nutzer vor dem Zugriff auf einen Server beweisen sollte. Zu diesem Zweck mußte er stets sein Paßwort eingeben. Dadurch ist es nicht ohne weiteres möglich, einen Daemon-Prozeß gegen einen anderen Dienst zu authentisieren. Kurz gesagt: Kerberos ist kein Host-zu-Host Protokoll.

Dies ist eine generelle Schwäche, die auch in Version 5 nicht behoben wurde. Kerberos basiert auf der Annahme, die Integrität der Hosts sei sicher – nur die Kommunikation über das Netzwerk eben nicht. Schlüssel in Keytabdateien sowie Session-Keys liegen ungeschützt auf der lokalen Festplatte[6]. Falls Computer überhaupt keine lokales Speichermedium besitzen, sieht es noch schlimmer aus. In diesem Fall liegt z.B. */tmp* auf einem

[6] es existiert nur die einfache Sicherheit, die das Dateisystem bietet

Fileserver: ein sehr hohes Sicherheitsrisiko. Es ist zwar möglich, Schlüssel im Shared Memory unterzubringen; wird eine Maschine jedoch gehackt, sind sie trotzdem ein offenes Geheimnis.

1.4.2 Behobene Probleme und Angriffsziele

Während die eben angeschnittenen Probleme auch in Version 5 noch nicht vernünftig korrigiert wurden, hat sich in vielen anderen Bereichen eine Menge getan. Die wichtigsten Punkte sollen jetzt erläutert werden.

Verschlüsselung

Kerberos4 beherrscht, wie in 1.3.2 beschrieben, nur DES-Verschlüsselung. In Version 5 ist der Kodierungsalgorithmus in generische Module ausgelagert worden. Kodierte Nachrichten enthalten angehängt ein Indikatorfeld, das den verwendeten Algorithmus eindeutig kennzeichnet. Auf diese Weise kann ein Client erkennen, wie und mit welchem Schlüssel die Nachricht dekodierbar ist.

Die einzelnen Verschlüsselungsalgorithmen sind selbst dafür verantwortlich, die Integrität der Nachricht zu sichern. Sollte einer dies nicht zur Verfügung stellen, kann vor der Kodierung eine Checksumme an die Klartextnachricht gehängt werden, um die Fälschungssicherheit zu erhöhen.

Nachrichtenkodierung

Unter Kerberos4 werden Nachrichten grundsätzlich in der Byte-Order des Senders über das Netzwerk verschickt. Dies macht die Übertragung zwischen Maschinen gleicher Architektur leicht. Sollten diese jedoch abweichen, hat der Empfänger dafür Sorge zu tragen, daß er die Botschaft lesen kann.

Dies gehört mit Kerberos5 der Vergangenheit an. Im Netzwerkprotokoll kommt nun ASN.1[7] zum Einsatz. Die Protokollbeschreibung sieht dadurch stark verändert im Vergleich zu Version 4 aus. Das betrifft aber nur die Präsentation der Nachricht, der Inhalt bleibt größtenteils unverändert.

Netzwerkadressen

Ein weiteres Problem ist die Tatsache, daß in den Tickets nur eine IP-Adresse gespeichert wird. Computer mit mehreren Netzwerkkarten kommen damit schlecht zurecht. Da auf den meisten PCs, auf denen Nutzer normalerweise arbeiten, dies nicht zutrifft, ist es

[7] Abstract Syntax Notation One

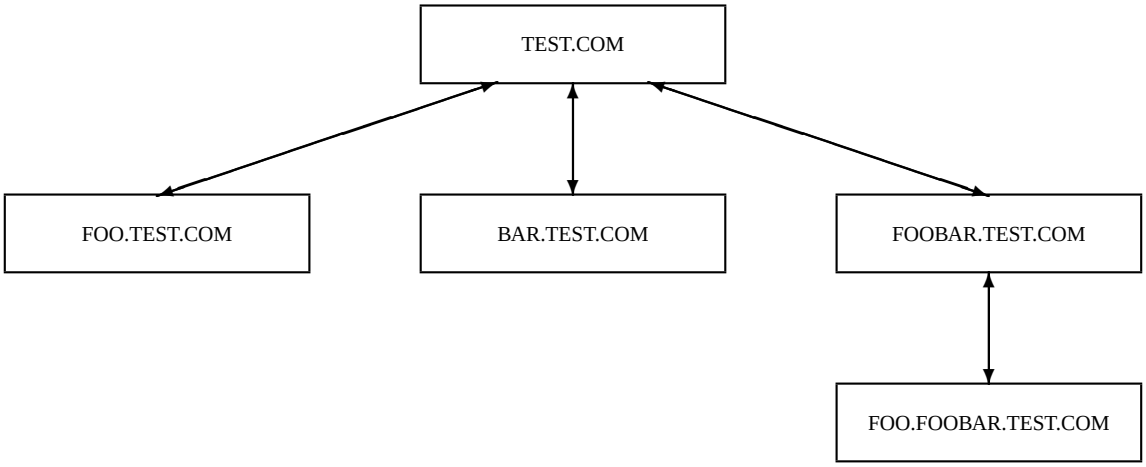

Abbildung 1.5: Eine Kerberos5 Realmhierarchie

oft nur ein theoretisches Problem. Das Ticketformat von Kerberos5 läßt es zu, mehrere Adressen in einem Ticket abzuspeichern. Auch Adressen verschiedener Netzwerkprotokolle sind möglich.

Ticket Forwarding

Ein TGT, das auf einer Maschine generiert wurde, kann auf einer anderen aufgrund der enthaltenen Adresse nicht mehr benutzt werden. Dies soll zwar die Sicherheit erhöhen, macht heutzutage aber Probleme, falls man sich mit z.B. **ssh** kaskadierend über mehrere Rechner einloggt. In OpenAFS' Kerberos Implementation ist eine Adressüberprüfung deshalb gar nicht erst vorhanden.

Unter Kerberos5 wurde das Konzept des Ticket Forwardings eingeführt. Verbindet sich ein Nutzer über Kerberos Authentisierung auf eine andere Maschine, kann er unter Vorlage eines gültigen TGT vom KDC ein neues TGT erhalten. Dieses enthält dann die Adresse des neuen Hosts. Das erste TGT muß ein Flag enthalten, welches es als weiterleitbar identifiziert. Fehlt es, wird das KDC kein neues TGT ausstellen.

Cross-Realm Authentisierung

Um die Zahl der Schlüssel bei der Authentisierung zwischen mehreren Realms zu verringern, wurde für Kerberos5 das Konzept der hierarchischen Realms eingeführt. Dadurch entsteht eine Art „Realm Baum“ (siehe Abbildung 1.5). Ein KDC speichert nun den Schlüssel der übergeordneten Realm sowie die aller untergeordneter Realms. Die Schlüsselanzahl steigt nun nur noch logarithmisch zur Zahl der beteiligten Realms. Bei einer Cross-Realm Authentisierung wird der Weg über mehrere KDC bis zur Zielrealm „abgewandert“.

Principal Benennung

Unter Kerberos4 sind Principalnamen aus drei Teilen aufgebaut:

1. Der eigentliche Name
2. Eine (optionale) Instanz
3. Die Kerberos Realm

Jeder dieser Bestandteile kann bis zu 39 Zeichen lang sein. Für einige Anwendungen ist das jedoch zu kurz. Als Trenner zwischen Namen und Instanz dient der Punkt „.“. Aus diesem Grund vertragen sich Principals nicht mit Accountnamen, die einen Punkt enthalten. Alles zusammen ergibt folgendes Format für den Aufbau:

`Name.Instanz@REALM`

Unter Kerberos5 sieht der Aufbau leicht modifiziert aus. Ein Principal besteht jetzt aus mehreren Komponenten die durch einen Slash „/„ getrennt sind:

`Komponente1/Komponente2/.../KomponenteN/@REALM`

Im allgemeinen wird die Komponente 1 immer noch als Name und Komponente 2 als Instanz bezeichnet. Mehr als zwei Komponenten sind nicht gebräuchlich.

Lebenszeit der Tickets

Tickets in Kerberos4 enthalten einen Startzeitpunkt und eine Zeitspanne, die das Gültigkeitsende definiert. Dies ist ein 8-Bit Wert, der in der Originalversion von Kerberos4 mit fünf Minuten multipliziert wurde. Daraus ergab sich eine Maximallebensdauer von $21\frac{1}{4}$ Stunden. Dies war auch für damalige Verhältnisse zu kurz, sodaß seit 1988 eine neue Methode genutzt wird. Bis zu einem Wert von 128 erfolgt weiterhin eine Multiplikation mit fünf. Danach steigt dieser Faktor auf Basis einer Tabelle an. Es ergibt sich auf diese Weise eine Maximallebensdauer von 30 Tagen.

In Kerberos5 Tickets ist sowohl die Start- als auch die Endezeit ein Zeitstempel. Dies ermöglicht theoretisch unbegrenzte Lebenszeiten.

Doppelte Verschlüsselung im TGT

Wenn man sich die Antwort des KDC nach einer TGT-Anfrage in 1.3.2 noch einmal ansieht, ist zu erkennen, daß das eigentliche Ticket $T_{C,S}$ zweimal verschlüsselt wird:

$\{S, T, L, K_{C,S}, \{T_{C,S}\}K_S\}K_C$

Dies ist unnötig und kann wertvolle Rechenzeit vergeuden. Unter Kerberos5 sieht die Antwort deshalb so aus:

$\{S, T, L, K_{C,S}\}K_C, \{T_{C,S}\}K_S$

Paßwort-zu-Schlüssel Konvertierung

Wie schon unter 1.3.6 erläutert, kommt bei Kerberos4 nur das Paßwort zur Erstellung des Schlüssels zum Einsatz. Da Nutzer häufig das selbe Paßwort für viele Dinge benutzen, kann es zu einem Problem kommen.

Gelingt es Angreifern in einer fremden Realm einen Principalschlüssel zu stehlen, ist dieser bei gleichem Paßwort auch in der eigenen Realm gültig. Der Account ist gleich doppelt kompromittiert. Unter Kerberos5 wird deshalb zur Schlüsselgenerierung auch der Principalname genutzt. Dieser enthält unter anderem als Teil den eindeutigen Realmnamen. Der Schlüssel ist aus diesem Grund in jeder Realm – auch bei gleichem Paßwort – verschieden.

K_{V5} `= String-To-Key(Paßwort, Principalname)`

Replay-Attacken

Dieses Thema wurde unter 1.3.3 schon kurz angeschnitten. In einem unsicheren Netz können Nachrichten mitgehört und somit auch erneut gesendet werden. Anfragen an Server, die gültige Tickets beinhalten, sind so angreifbar. Indem die Identität des Nutzers, der die originale Anfrage abschickte, vorgetäuscht wird, kann sich Zugang zum Dienst erschlichen werden, ohne die Nachricht selbst zu entschlüsseln. Um dies zu verhindern, wird bei der Originalanfrage der Authenticator mitgeschickt, der unter anderem einen Zeitstempel und die IP-Adresse enthält. Innerhalb der maximal zulässigen Zeitspanne (normalerweise 5 Minuten) ist eine Replay Attacke aber trotzdem möglich, da IP-Adressen auch gefälscht werden können.

Kerberos5 besitzt als Gegenwehr einen Replay-Cache. Dabei werden beim Server alle Authenticators bis zum Ablauf ihrer Gültigkeitsdauer aufgehoben und beim erneuten Eintreffen als Replay-Attacke enttarnt.

Da Kerberos als Transportprotokoll standardmäßig UDP[8] benutzt, kann der Replay-Cache auch Probleme bereiten. UDP gewährt keine Sicherheit, daß Pakete ihr Ziel wirklich erreichen. Sollte die Antwort des Servers verloren gehen, wird der Client das Ticket und den Authenticator erneut schicken. Jetzt wird diese zweite Anfrage beim Server als Replay gewertet! Das führt zu Dienstverweigerungen für eigentlich berechtigte Nutzer und zu Fehlalarmen[1]. In Heimdal wurde das Problem bisher noch nicht gelöst – der Replay-Cache ist hier unbrauchbar.

Ein weiteres Konzept zum von Replay-Attacken sind Sequenznummern. Im Authenticator sind sie an Stelle der IP-Adresse und des Zeitstempels enthalten. Der Applikationsserver muß nun darauf achten, daß die Nummern in der richtigen Reihenfolge und ohne Lücken bei ihm eintreffen.

[8] wird durch RFC 1510 vorgeschrieben

Systemzeitattacken

Der letzte Punkt zeigt ein anderes Angriffsziel auf. Der Timeservice, über den sich alle Maschinen synchronisieren, läuft unauthentisiert und unverschlüsselt ab. Angreifer können diese Nachrichten modifizieren und somit erreichen, daß auf dem Opfersystem eine von ihnen frei gewählte Zeit läuft. Die Zeitspanne für Replay Attacken kann somit ins Unendliche ausgedehnt werden. Jedoch sind auch Denial-Of-Service Attacken denkbar, bei dem die Zeit des KDC von denen aller Clients abweicht. Ein sicherer Timeservice ist für Kerberos unabdingbar.

Paßwortattacken

Der Erwerb von TGTs unter Kerberos4 ist gegen Offline Attacken anfällig. Die initiale Authentisierungsanforderung wird vom KDC mit einem TGT beantwortet, das mit dem Schlüssel des Principals kodiert ist. Mittels einer Brute-Force Attacke kann ein Angreifer systematisch Schlüssel testen, die dieses TGT vernünftig lesbar machen: Der Principalschlüssel ist nicht länger geheim. Das Paßwort, worauf der Schlüssel basiert, ist jedoch nicht ausspionierbar.

Um dieses Angriffsziel auszumerzen, ist in Version 5 das Konzept der Preauthentication eingeführt worden. Ist dies für die Anmeldung zwingend vorgeschrieben, muß der Principal bei der Authentisierungsanforderung zuerst einen Beweis seiner Identität mitschicken, bevor das KDC ein TGT ausstellt. Im einfachsten Fall ist das der mit dem Principalschlüssel kodierte aktuelle Zeitstempel.

1.4.3 Weitere Neuerungen in Kerberos5

Erneuerbare Tickets

Gerade in Batchsystemen sind die maximalen Lebenszeiten von rund 21 Stunden oft viel zu kurz. Unter Kerberos5 sind zwar längere Zeiten möglich, jedoch sollen Tickets nicht grundsätzlich viele Tage gültig bleiben, nur um alle Eventualitäten auszuschließen.

Es gibt nun ein zusätzliches Flag im TGT, das angibt, ob die Lebensdauer über den eigentlichen Zeitraum verlängert werden kann. Zusätzlich steht im Ticket, bis zu welchem Zeitpunkt diese Verlängerung regelmäßig stattfinden darf. Am DESY Zeuthen beträgt der Zeitraum 30 Tage. Vor Ablauf der Gültigkeit des TGT kann nun beim KDC unter Vorzeigen des Tickets ein erneuertes TGT angefordert werden, welches wieder die Standardlebensdauer hat.

Vordatierte Tickets

Ein andere Erweitung, die auch primär für Batchsysteme geschaffen wurde, sind vordatierte Tickets. Ein Principal kann bei einer TGT-Anforderung angeben, ab welchem

Zeitpunkt in der Zukunft das TGT gültig werden soll. Davor ist es nicht benutzbar.

Nutzer-zu-Nutzer Authentisierung

Kerberos ist in erster Linie ein Protokoll, welches Nutzern (also Menschen vor z.B. PCs) die Möglichkeit bietet, sich bei Netzwerkdiensten zu authentisieren. Es ist jedoch unter Kerberos4 nicht möglich, die Identität von Nutzern gegenüber anderen Nutzern zu beweisen. Dies gehört mit Kerberos5 der Vergangenheit an.

Oft bieten auch normale Nutzer Dienste an, bei denen sich andere sicher authentisieren können sollen. Eine Möglichkeit wäre es, dem Dienstanbieter einen langlebigen Service-Schlüssel zu geben, der ihn in einer Keytabdatei abspeichert. Dieser wäre jedoch ein Angriffsziel für Spione. Bei der Nutzer-zu-Nutzer Authentisierung nimmt des dienstanbietenden Nutzers kurzlebiger TGT Session-Key die Rolle des Service-Schlüssels ein. Das TGT muß deshalb natürlich periodisch erneuert werden.

Nutzer-zu-Nutzer Authentisierung bedarf einer speziellen Behandlung in den Programmen. Kerberos bietet keine API dafür an. Der Ablauf kann kurz wie folgt beschrieben werden. Ein Nutzer (Client), der von einem Dienst eines anderen Nutzers (Server) Gebrauch machen möchte, fordert von diesem das TGT an. Der Client kann dies nicht weiter nutzen, da ihm der passende Session-Key fehlt. Stattdessen schickt er es weiter zum KDC und erhält Credentials zurück. Einen Teil kann der Client entschlüsseln und erhält einen Session-Key für die Verbindung, den anderen schickt er zum Server Nutzer, der auch den Session-Key erhält (ist mit seinem TGT Session-Key verschlüsselt). Nun können sich Client und Server über den neu erworbenen Session-Key verschlüsselt unterhalten. Da beide Parteien dies tun können, wenn sie jeweils einen gültigen TGT Sitzungsschlüssel besitzen, sehen sich beide als gegenseitig authentisiert an.

1.5 MIT-Kerberos5 vs. Heimdal

Unter den unixoiden Betriebssystemen haben sich zwei größere Kerberos5 Implementationen etabliert: MIT und Heimdal. Während die Version von MIT die erste Kerberos5 Distribution war, begann die Arbeit an Heimdal erst Mitte der 90er Jahre des letzten Jahrhunderts. Dafür gab es Gründe. Der schwerwiegendste ist das Ausfuhrverbot für starke Kryptographiealgorithmen in den USA, wie z.B. DES. Verschlüsselungsverfahren fallen unter die selbe Kategorie wie Waffen. Zwar wurden die Beschränkungen in letzter Zeit gelockert, jedoch muß vor dem Download von MIT Kerberos5 eine Erklärung abgegeben werden, daß man amerikanischer oder kanadischer Staatsbürger ist. Ob der Einsatz außerhalb des nordamerikanischen Kontinents rechtlich gesehen in Ordnung ist, wissen nicht einmal die Personen in der Mailingliste genau. Sie empfehlen die Konsultation eines Anwalts. Da Heimdal außerhalb der USA entwickelt wird (KTH[9] Schweden), ist es völlig frei erhältlich.

[9] Kungliga Tekniska Högskolan

Ein weiteres Thema ist die mangelnde AFS Unterstützung in MITs Implementierung. Es sind Zusatzwerkzeuge wie das AFS-Krb5 Migration Kit (ftp://ftp.cmf.nrl.navy.mil/pub/kerberos5/afs-krb5-2.0.tar.gz) notwendig. Heimdal arbeitet von sich aus mit voller AFS Unterstützung.

Die letzten beiden Punkte gaben den Ausschlag, am DESY Zeuthen die Heimdaldistribution einzusetzen. Heimdal hat jedoch nicht nur Vorteile. Da es relativ jung ist, sind einige Features noch nicht vollständig implementiert (z.B. GSSAPI). Auch die Dokumentation ist ziemlich dürftig. Es folgt eine kleine Übersicht über die Vor- und Nachteile der beide Distributionen.

Heimdal:

+ Völlig freie Implementation
+ Sehr gute AFS Unterstützung
+ Einfache Konvertierung der AFS Datenbank in eine native Heimdaldatenbank
+ Inkrementelle Datenbankpropagierung (siehe 3.3.5)
- schlechte Dokumentation
- wenige Anwendungen lassen sich direkt gegen Heimdal bauen
- Features teilweise noch nicht implementiert

MIT:

+ Ausgereifte Distribution
+ Viele Anwendungen nur hiermit kompilierbar
+ Weit verbreitet (besonders in USA)
- AFS Unterstützung nur mit Zusatztools
- Keine inkrementelle Datenbankpropagierung

1.6 Komponenten der Heimdaldistribution

1.6.1 Serverprogramme

kdc: Das Key Distribution Center läuft auf allen Datenbankservern. TGT- und Service Ticket Anfragen werden von ihm abgearbeitet.

kadmind: Der Administrationsserver läuft nur auf dem Master KDC. Er stellt den Zugang (Abfragen und Änderungen) zur Kerberosdatenbank über das Netzwerk her.

kpasswdd: Der Paßwortänderungsserver läuft ebenfalls nur auf dem Master KDC. Über das Programm **kpasswd** können Nutzer ihr Paßwort in der Datenbank ändern.

hpropd: Dies ist der Datenbank Propagierungsserver. Im Zusammenspiel mit dem Programm **hprop** kann die komplette Kerberos Datenbank verschlüsselt über das Netzwerk kopiert werden.

ipropd-master: Bei kleinen Änderungen an der Masterdatenbank braucht diese nicht immer wieder komplett über das Netz geschickt zu werden. Besser wäre es, nur diese Aktualisierungen zu übertragen. Dies nennt sich inkrementelle Propagierung. Am **ipropd-master** können sich Clients diese Änderungen holen, er läuft nur auf dem Master KDC

ipropd-slave: Dies ist der Client der inkrementellen Propagierung. Er läuft natürlich nur auf den Slave KDCs.

Es werden außerdem kerberisierte Servervarianten von RSH, POP3, FTP und Telnet angeboten. Diese sollten jedoch aufgrund ihrer unsicheren Natur nicht eingesetzt und durch sicherere Dienste wie SSH und IMAP ersetzt werden.

1.6.2 Administrationsprogramme

kadmin: Mit diesem Programm werden (fast) alle Änderungen an der Datenbank vorgenommen. Dazu gehören Anlegen, Modifikation und Löschen von Principals. Diese Änderungen sollten immer auf der Masterdatenbank durchgeführt werden. **kadmin** kommuniziert mit dem Dienst **kadmind** kann aber über den Schalter *-l* auch lokal die Datenbank modifizieren.

ktutil: Hiermit werden Einträge in Keytabdateien verwaltet. Es können Prinipalschlüssel angezeigt, gelöscht und auch hinzugefügt werden. Vorsicht: Beim Hinzufügen eines Schlüssels zur Datei mittels **ktutil get** <**Principal**> wird dieser in der Masterdatenbank neu angelegt. Im Falle, daß der Principal noch nicht existiert ist das kein Problem. Gibt es ihn jedoch schon in der Datenbank, bekommt er einen neuen Schlüssel – der alte wird ungültig! Falls der aktuelle Schlüssel in die Keytabdatei extrahiert werden soll, sollte **kadmin ext** genutzt werden. Mit **ktutil change** ist ein Abgleich der Schlüssel in der Keytabdatei mit denen der Datenbank möglich.

kstash: Dieses Programm erzeugt den Masterschlüssel der Kerberosdatenbank, mit dem sie verschlüsselt wird. Der Masterschlüssel befindet sich danach unter */var/heimdal/m-key*.

hprop: Mit diesem Programm kann, wie schon erwähnt, die Datenbank über das Netz kopiert werden. Außerdem ist es möglich Datenbanken verschiedener Formate (Kerberos4, MIT, AFS KA-Datenbank) in eine für Heimdal lesbare Form umzuwandeln.

1.6.3 Clientprogramme

kinit: Ein Nutzer kann hiermit ein neues TGT anfordern. Wenn die Unterstützung dafür mit einkompiliert ist, werden automatisch ein Version 4 TGT und ein AFS Token generiert.

klist: Dient dem Anzeigen der gespeicherten Credentials im Ticket Cache des Nutzers. TGTs und Service Tickets sind damit einsehbar.

kdestroy: Ein Nutzer kann hiermit seinen Ticket Cache zerstören. Auch das AFS Token (falls vorhanden) im Kernel Cache wird dabei gelöscht.

afslog: Kann zum Erzeugen eines AFS Token (bei vorhandenem TGT) genutzt werden. Dies geschieht jedoch bei einem korrekt konfigurierten System automatisch.

kpasswd: Hiermit können Nutzer ihr Paßwort in der Datenbank ändern[10].

xnlock: Dies ist ein Bildschirmschoner für den X-Server. Es sperrt den Bildschirm. Bei erfolgreicher Paßworteingabe werden automatisch die Credentials im Ticket Cache erneuert.

Zu den bereits erwähnten kerberisierten Serverprogrammen existieren natürlich auch Nutzerprogramme, deren Einsatz auch hier nur bedingt zu empfehlen ist. Sie dienen mehr oder weniger nur als Anleitung, wie verschiedene Dienste kerberisiert werden können.

1.7 Kerberos5 unter Windows

Mit der Einführung von Windows 2000 wird unter den professionellen Betriebssystemen aus dem Hause Microsoft Kerberos5 zur Authentisierung an der Netzwerkdomäne genutzt. Es ist Teil des zentralen Verzeichnisdienstes „Active Directory“. Die Kerberos Implementation basiert auf RFC 1510, enthält aber einige Erweiterungen, wie z.B. Public-Key Authentisierung. Auch sind in den Tickets einige Zusatzinformationen enthalten, die bislang nur ungenutzt in der Spezifikation waren. Diese Daten sind jedoch nur unter dem Active Directory selbst zugänglich.

Aus diesem Grund ist es nicht möglich das KDC, welches im Active Directory enthalten ist, durch das einer anderen Distribution auszutauschen. Eine Authentisierung ist zwar möglich, doch sind nicht alle Features des Active Directory nutzbar, da dafür eben diese Zusatzinformationen benötigt werden. Auf der anderen Seite bietet die Windows Implementation keine Möglichkeit, die Beschaffung von AFS Tokens bei der Anmeldung zu integrieren. Eine Verschmelzung der Nutzerumgebung (selbes Homedirectory unter Windows und UNIX) ist somit unmöglich.

Am italienischen CASPUR hatte ich die Möglichkeit zu untersuchen, wie Windows Clients mit einem MIT oder Heimdal KDC zusammen arbeiten können. Dort ist genau die eben beschriebene Vereinheitlichung der Nutzerverzeichnisse in beiden Betriebssystemwelten vorgesehen. Das Resultat sieht nicht unbedingt schlecht aus! Sowohl Heimdal als auch MIT sind als Server bei der Netzwerkanmeldung von Windows 2000/XP Clients benutzbar. Dazu muß der auf den Windows CDs enthaltene Kerberos Client installiert und so konfiguriert werden, daß das alternative KDC benutzt wird.

[10] Genauer gesagt: sie ändern ihren Schlüssel, der nun auf dem neuen Paßwort basiert

Eine automatische AFS-Token Generierung über den OpenAFS Client für Windows ist mit Heimdal ohne Probleme möglich. Da in MIT die eingebaute AFS Unterstützung fehlt, muß das auch für UNIX notwendige AFS-Krb5 Migration Kit auf dem Server installiert sein.

Auch eine Paßwortänderung ist von Windows aus über den eingebauten Mechanismus „<Ctrl-Alt-Delete> → Change Password...“ mit einem MIT KDC möglich. Nach Rücksprache mit den Entwicklern von Heimdal ist die Implementation des Protokolls zum Ändern von Kerberos Paßwörtern unter Windows nach RFC 3244 im **kpasswdd** in einer der nächsten Versionen geplant. Ein vorläufiger Patch liegt unter http://www.stacken.kth.se/lists/heimdal-discuss/2003-06/msg00026.html. Somit sind die wichtigsten Funktionen für Nutzer auch unter Windows bei einem Heimdal KDC vorhanden.

Kapitel 2

AFS

2.1 Was ist AFS

AFS ist ein verteiltes Dateisystem. Es basiert auf einem Client/Server Modell. Das heißt, daß Dateien auf mehreren Hosts (Fileserver) liegen können, wobei Benutzer transparenten Zugriff auf diese Dateien von allen Client Hosts im Netzwerk haben, als ob sie auf dem lokalen Computer gespeichert wären. Ein weiteres Dateisystem, welches netzwerktransparenten Zugriff auf Dateien ermöglicht, ist NFS[1]. Im Vergleich dazu bietet AFS viele zusätzliche Features, die in 2.2 erläutert werden.

Der Name AFS ist ein Akronym für „Andrew File System“. Es wurde ab 1984 an der Carnegie Mellon University entwickelt. 1989 entstand die Firma Transarc, die es von nun an kommerziell weiter entwickelte und vertrieb. Im Jahre 1998 wurde Transarc von IBM übernommen, das die Sourcen zwei Jahre später frei gab. Aus diesem Code ging die OpenAFS Distribution hervor. In der Zeit als der AFS Quellcode noch nicht frei war, wurde an der KTH Schweden ein eigenes AFS-Projekt, genannt „Arla“, aus der Taufe gehoben. Auch nach der Veröffentlichung der Sourcen seitens IBM wird an diesem Projekt noch weiter gearbeitet, es ist zur Zeit jedoch nur eingeschränkt nutzbar.

Die OpenAFS Distribution kommt mit einer hervorragenden Dokumentation einher. Diese wurde zu Zeiten der kommerziellen Vermarktung geschrieben und ist damit teilweise etwas veraltet. Trotzdem bietet sie eine Fülle von Informationen und Anleitungen, sowohl für den normalen Nutzer als auch für den Administrator. Die meisten nun aufgeführten Punkte wurden dieser Dokumentation entnommen, die über die OpenAFS Homepage unter http://www.openafs.org/doc/index.htm erhältlich ist.

[1] Network File System

2.2 Features von AFS

AFS hat im Vergleich zu NFS viele Vorteile. Aus diesem Grund stellt es am DESY Software sowie die Homeverzeichnisse der meisten Nutzer bereit. Einige der Features sollen nun vorgestellt werden.

2.2.1 Zellen

AFS arbeitet als eine Erweiterung des lokalen Dateisystems. Es existiert ein Verzeichnis im lokalen Baum, das per Konvention */afs* heißt. Dies ist der Eintrittspunkt in die AFS Welt.

AFS ist in Verwaltungsdomänen unterteilt, die „Zellen“ genannt werden. Eine Zelle ist die Zusammenfassung von Fileservern und Clients, die unabhängig von anderen Zellen administriert wird. Ein einzelner Host kann gleichzeitig nur zu einer Zelle gehören, die „lokale Zelle“ heißt. Alle anderen sind fremde Zellen. Clients können die Dateien, die eine Zelle anbietet, anfordern, indem sie auf den Verzeichnisbaum unter */afs/<Zellenname>* zugreifen.

Der Zellenname ist meist identisch mit dem DNS Domainnamen der Hosts, die zu dieser Zelle gehören. Er kann jedoch auch frei gewählt werden. Dies bedeutet oft aber erhöhten Administrationsaufwand.

2.2.2 Volumes

Die AFS Fileserver speichern die Dateien in separaten Partitionen. Sie werden für gewöhnlich unter */vicepx* im Verzeichnisbaum des Servers eingehängt, wobei „x zur eindeutigen Identifizierung dient. Auf diesen Partitionen kann der Administrator Volumes anlegen. Sie stellen eine Art Container dar, in der zusammengehörige Daten gespeichert werden. Im AFS sind Dateien grundsätzlich auf Volumes abgelegt.

Die Volumenamen haben eine Maximallänge von 31 Zeichen. Da jedoch noch zusätzlicher Platz für die Replikaerweiterung (siehe 2.2.3) *.readonly* gelassen werden muß, beschränkt sich der Name auf nur 22 Zeichen.

Damit auf die Daten von Clients auch zugegriffen werden kann, ist für jedes Volume ein Mountpoint anzulegen. Dieser beschreibt die Position im AFS Verzeichnisbaum, über die die auf dem Volume enthaltenen Dateien adressierbar sind. Im Falle von Homedirectories ist als Volumename *user.<Accountname>* mit dem Mountpoint */afs/<Zelle>/user/<erster Buchstabe des Accountnamens>/<Accountname>* weit verbreitet.

Volumes sind nicht direkt an Partitionen gebunden. Aus diesem Grund ist es möglich, sie (im laufenden Betrieb!) zwischen einzelnen Partitionen und sogar Fileservern zu verschieben. Der Nutzer merkt davon nichts! Dies ist hervorragend dazu geeignet, ein gut

ausgeklügeltes Verfahren für den Lastenausgleich innerhalb der Fileserver zu entwickeln: Volumes, die oft genutzt werden, können auf Server, die weniger zu tun haben, verschoben werden.

2.2.3 Volume Replikation

Eine weitere Möglichkeit, Last von einzelnen Servern zu nehmen, sind Replika. AFS kann von stark frequentierten Volumes (nur lesbare) Kopien erstellen und diese auf anderen Fileserver unterbringen. Die Zugriffslast auf Dateien dieser Volumes verteilt sich somit auf die Fileserver. Das funktioniert natürlich nur solange Dateien zum Lesen geöffnet werden. Modifikationen an Daten sind nur am Original vornehmbar.

Replika steigern auch die Ausfallsicherheit. Ist ein Fileserver nicht verfügbar, wären im Normalfall alle auf ihm untergebrachten Daten nicht erreichbar. Da sie nun aber auf anderen Servern ebenfalls angeboten werden, merkt der Benutzer im Idealfall nicht einmal den Ausfall.

Werden Änderungen am Original vorgenommen, so sind diese nicht automatisch in den Replika verfügbar. Der AFS Administrator muß zu diesem Zweck **vos release** <**volume**> ausführen. Gibt es zu einem Volume Replika, ist der Mountpoint des Originals */afs/.*<*Zelle*>*/...* Die nur lesbaren Replika sind über */afs/*<*Zelle*>*/...* erreichbar.

2.2.4 Quotas

Um zu verhindern, daß Volumes in ihrer Größe zu stark anwachsen, gibt es auch im AFS Quotas. Diese werden vom Administrator festgelegt. Ein Nutzer kann die Quota eines Volumes über den Befehl **fs lq** <**mount point**> abfragen.

2.2.5 Cache Manager

Programme, die auf das AFS zugreifen wollen, können dies nicht direkt tun. Es muß auf jeder Maschine, die ein AFS Client ist, ein Daemon laufen – der Cache Manager. Über ihn läuft jede Kommunikation mit den Fileservern.

Möchte ein Nutzer eine Datei lesen, fordert er diese (natürlich völlig transparent – das heißt: er merkt dies eigentlich überhaupt nicht) beim Cache Manager an. Dieser sieht zuerst nach, ob sich die geforderte Datei nicht schon im Cache befindet. Ist sie nicht vorhanden, wird sie vom Fileserver angefordert, lokal zwischengespeichert und zuletzt dem Nutzer gegeben. Bei erneuter Anforderung kann nun gleich diese Kopie genutzt werden. Das spart Netzwerkressourcen und Leistung auf den Servern.

Mit der Datei erhält der Cache Manager eine „Rückrufversicherung“ (oder Callback). Sollte das Original der angeforderten Datei auf dem Server modifiziert werden, erhält

er vom Fileserver diesen Rückruf. Jetzt muß sie vor der erneuten Auslieferung an ein Programm neu geholt werden. Auch beim Speichern der Datei auf dem Client wird sie nicht sofort an die Server übertragen. Dies erfolgt erst nach ihrem Schließen.

2.2.6 Authentisierung

AFS benutzt zur Authentisierung Kerberos4. Es ist jedoch eine leicht modifizierte Variante, da die Implementation in AFS vor der Standardisierung von Kerberos der Version 4 fertig gestellt wurde.

Nach der Anmeldung erhält der Nutzer ein sogenanntes AFS Token. Dies ist im Sinne von Kerberos ein Service Ticket, nur mit kleiner Abwandlung. Service Tickets werden normalerweise im Ticket-Cache auf der Festplatte aufbewahrt. Das AFS Token jedoch liegt im RAM und wird dort vom Cache Manager verwaltet. Bei jedem Zugriff auf AFS Ressourcen dient das Token als Beweis der Identität des Nutzers.

Wie schon erwähnt unterscheidet sich die Kerberos Implementation im AFS etwas vom Standard der Version 4. Nach einer Authentisierung wird normalerweise das TGT aufgehoben und für die Anforderung weiterer Service Tickets benutzt. Dies ist bei AFS unnötig, da der einzige Service das AFS ist. Aus diesem Grund wird nach der Generierung des AFS Tokens das TGT verworfen, da es im Prinzip nutzlos ist.

Für Nutzer, die später andere kerberisierte Dienste ansprechen wollen, gibt es aber auch die Möglichkeit, das TGT zu behalten. Dazu ist statt des üblichen **klog**, das etwas modifizierte **klog.krb** zu benutzen. Dies hat sogar einen weiteren Vorteil. Während es unter Kerberos4/5 nicht vorgesehen ist, mehrere TGT verschiedener Realms gleichzeitig im Ticket-Cache aufzubewahren, funktioniert dies mit **klog.krb**. Natürlich kann ein Nutzer auch mehrere Tokens verschiedener Zellen gleichzeitig besitzen. Es ist jedoch unmöglich, verschiedene Tokens ein und der selben Zelle zu halten.

Ein weiterer Unterschied zum Kerberos4 Standard ist die verwendete String-To-Key Methode, die aus Nutzerpaßworten Schlüssel generiert. Bei AFS kommt neben dem Paßwort auch der Zellenname zum Einsatz:

K_{AFS} = String-To-Key(Paßwort, Zelle)

Da auf einem Host mehrere Nutzer (oder ein Nutzer mehrmals) angemeldet sein können, muß der Cache Manager die vorgehaltenen AFS Token eindeutig zuordnen. Dazu bedient er sich der PAG[2]. Es wäre theoretisch auch möglich die Zuordnung über die Nutzeridentifikationsnummer (UID) zu regeln. Dies wirft jedoch Sicherheitsprobleme auf, da lokale Superuser ohne Probleme die Identität anderer Nutzer annehmen können (z.B. mittels **su**). Das Sicherheitsniveau würde auf einfache Betriebssystemsicherheit sinken.

Da AFS zur Authentisierung an seinen Diensten nur das AFS Token benötigt, ist es egal über welchen Mechanismus dieses erworben wurde. Dies macht es möglich, die einge-

[2] Process Authentication Group

baute Kerberos4 Authentisierung durch eine der Version 5 zu ersetzen, die weitaus mehr Sicherheit und Features bietet. Darauf basiert letztendlich diese Diplomarbeit.

2.2.7 Zugriffsrechte

Anders als die unter UNIX üblichen drei Zugriffskriterien (Besitzer, Gruppenzugehörigkeit, Rest) gibt es unter AFS sieben davon: (in Klammern die Abkürzung, die **fs listacl** benutzt)

- Recht, Dateien im Verzeichnis aufzulisten (l)
- Leserecht (r)
- Schreibrecht (w)
- Recht, Dateien zu löschen (d)
- Recht, Dateien anzulegen (i)
- Recht, ACLs zu verändern (a)
- Recht, exclusiv Dateien zu locken (k)

Diese Rechte sind jedoch auf ganze Verzeichnisse beschränkt. Das heißt, daß alle Dateien im selben Verzeichnis die selben Rechte besitzen. Ein weiterer Vorteil ist die volle Kontrolle des Besitzers über die Zugriffrechte. Er kann für jedes Verzeichnis eine Liste von Principals und Gruppen mit jeweils individuellen Rechten festlegen. Diese werden „Access Control Lists“ (kurz ACL) genannt. Dabei steht es dem Nutzer frei, eigene neue Gruppen zu erstellen und Nutzernamen oder sogar andere Gruppen dieser hinzuzufügen. Es können außerdem Gruppen erstellt werden, die nur IP-Adressen enthalten. Somit sind Zugriffbeschränkungen auf einzelne Hosts möglich!

2.3 Die Serverdienste

Die OpenAFS Distribution enthält viele Serverdienste, die jeweils eine spezielle Aufgabe erfüllen. AFS wurde konzipiert, den täglichen Wartungsbedarf durch den Systemadministrator so gering wie möglich zu halten. Auf der anderen Seite soll der Dienst nur sehr seltene – und wenn doch, sehr kurze Ausfallzeiten haben. Die einzelnen Prozesse sollen nun vorgestellt werden.

2.3.1 Basic Overseer Server

Der „Basic Overseer Server“ ist im Programm **bosserver** enthalten. Er überwacht alle anderen Prozesse, die unter seiner Obhut auf dem Fileserver laufen. Sollte ein Dienst abstürzen, versucht der **bosserver** selbständig, ihn neu zu starten. Somit erspart er dem Administrator die Arbeit, ständig auf alle Prozesse zu achten. Auch ist er der Garant für sehr kurze Ausfallzeiten der AFS Dienste. **bosserver** kann bei dem Ausfall mehrerer Prozesse auch auf die Reihenfolge achten, in der die einzelnen Dienste nacheinander gestartet werden müssen.

Gesteuert wird der „Basic Overseer Server“ über das Administrationsprogramm **bos**.

2.3.2 File Server

Der „File Server“ ist das Herzstück der Serverdienste, da er den Zugriff auf die Daten gewährt. Folgende Aufgaben laufen über ihn:

- Auslieferung von Dateien an AFS Clients. Außerdem werden Änderungen an diesen Dateien abgespeichert.
- Organisation von Dateien in Verzeichnisstrukturen wie der Nutzer es unter UN*X gewohnt ist.
- Sämtliche anderen Dateioperationen: Kopieren, Bewegen, Löschen von Dateien und ganzen Verzeichnisbäumen.
- Anlegen von symbolischen und harten Links[3].
- Er überwacht und aktualisiert alle Statusinformationen der Dateien und Verzeichnisse, wie z.B. Größe und Änderungsdatum.
- Vor jeder Operation findet eine Überprüfung statt, ob der Client berechtigt ist, die geforderte Aktion auszuführen (Authentisierung über das AFS Token und Authorisation gegen die ACL).
- Setzen von exklusiven Locks nach Wunsch.

2.3.3 Kerberos Authentication Server

Der **kaserver** ist die Authentisierungsintanz der OpenAFS Distribution. Bevor ein Nutzer als angemeldet gilt, muß er sich über den Kerberos Mechanismus beim **kaserver** mit

[3] Harte Links sind an einigen Stellen der AFS Dokumentation als nicht unterstützt angegeben. Ich konnte sie jedoch auf einem OpenAFS Fileserver (Version 1.2.8) ohne Probleme anlegen.

seinem Paßwort authentisieren. Dieser stellt ihm ein AFS-Service Ticket aus, welches das AFS Token enthält.

Im Authentication Server sind einige Mechanismen enthalten, die zusätzliche Sicherheit bringen sollen. Um die Chance eines Diebstahls zu verringern (oder besser: den Diebstahl sinnloser zu machen), ändert der **kaserver** regelmäßig den Schlüssel des TGS. Eine weitere Maßnahme ist die automatische Sperrung von Principals nach einer gewissen Anzahl von fehlerhaften Authentisierungsversuchen. Diese wird nach einer vom Administrator vorgegebenen Zeitspanne automatisch wieder aufgehoben. Er kann den Account aber auch selbst mit dem Programm **kas** vor Ablauf der Sperre freigeben.

2.3.4 Protection Server

Der „Protection Server“ oder kurz **ptserver** überwacht den Zugriff auf der Dateien. Während der **kaserver** für die Nutzerauthentisierung zuständig ist, übernimmt der **ptserver** die Authorisationskontrolle. Er verwaltet die ACL auf Verzeichnisse sowie die Zuordnung der Principals zu den einzelnen Gruppen. Vor jeder Dateitransaktion wird er vom Fileserver befragt, ob die geforderte Aktion eines Clients auch zulässig ist.

2.3.5 Volume Server

Der „Volume Server“ dient zur Verwaltung der Volumes. Über ihn werden Volumes angelegt, gelöscht und verschoben. Außerdem können mit ihm Volumes für eine Archivierung vorbereitet werden.

2.3.6 Volume Location Server

Der **vlserver** führt die Liste von AFS Volumes mit der Angabe, auf welchem File Server sie sich befinden. Diese heißt „Volume Location Database“ (kurz VLDB). Immer wenn der Cache Manager auf dem Client eine Datei aus dem AFS holen möchte, fragt er zuerst den **vlserver**, auf welchem File Server sich die Datei überhaupt befindet. Danach stellt er die Anfrage auf Auslieferung an den Dateiserver, den er als Antwort bekommen hat.

Im Grunde ist der **vlserver** der Schlüssel zu allen Dateien. Sollte die VLDB zerstört sein, besteht für den Client keine Möglichkeit, eine bestimmte Datei zu finden.

2.3.7 Backup Server

Der „Backup Server“ ist für die Verwaltung der Backup Datenbank zuständig. Beide zusammen machen sie das Sicherungssystem von OpenAFS für die Dateiarchivierung aus.

Einmal konfiguriert laufen Sicherungen wichtiger Volumes automatisch ab. Diese können auch inkrementell sein. Sollte eine Wiederherstellung bestimmter Dateien nötig sein, kann der Administrator diese z.B. vom Band zurückholen. Dabei ist es sogar möglich, den Zustand zu einem bestimmten Zeitpunkt zu rekonstruieren.

Weil die Backup Datenbank der Schlüssel zu allen Sicherungen ist, empfiehlt es sich, diese ebenfalls zu archivieren.

2.3.8 Update Server

Da es zu Problemen führen kann, wenn auf verschiedenen AFS Servern unterschiedliche Versionen der selben Software laufen, sorgt der „Update Server“ dafür, daß dies nicht vorkommt.

Dazu muß auf einer Maschine, die „Binary Distribution Machine“ (BDM) genannt wird, die Software installiert werden, die auf allen Servern laufen soll. Die Update Server der anderen Maschinen fragen nun regelmäßig die BDM ab, ob eine neue Version vorhanden ist und installieren diese bei Bedarf. Der Administrator muß sich darum nicht weiter kümmern.

2.3.9 Salvager

Der „Salvager“ ist kein Server im eigentlichen Sinne. Er wird vom Basic Overseer Server angestoßen, falls eine Fileserver- oder Volume-Server Instanz mit einem Fehler ihren Dienst abgebrochen haben. Dabei können ganze Volumes oder auch nur einzelne Daten beschädigt werden. Der Salvager versucht nun diese zu beheben. Falls benötigt, ist er auch vom Administrator direkt startbar.

Kapitel 3

Die Migration

3.1 Vorüberlegungen

Um die Umstellung auf Kerberos5 so problemlos wie möglich vonstatten gehen zu lassen, sind ein paar Vorüberlegungen notwendig. Es sollen keine Einschränkungen für die Nutzer während dieser Zeit entstehen. Um auch die Vorteile von Kerberos5 nutzen zu können, heißt das, alle Dienste während der Migrationsphase sowohl mit Kerberos4- als auch Kerberos5-Unterstützung laufen zu lassen. Hier folgt nun die Schritt-für-Schritt Anleitung unserer Migration:

1. Kompilation der Software (Neuübersetzung Athena, Heimdal)
2. Starten eines Testservers, Initialisierung der Heimdaldatenbank und Anlegen von Testnutzern
3. Testen der Funktionalität des Servers
4. Einfügen der aktuellen AFS-Datenbank in die Heimdaldatenbank
5. Testen, ob Nutzer gültige TGT und AFS Token bei der Authentisierung gegen das Heimdal KDC bekommen
6. Aufsetzen eines KDC-Slaveservers
7. Konfiguration und Verteilung der notwendigen Software auf alle Clienthosts (PAM, OpenSSH).
8. Abschaltung der **kaserver**, Migration der KDCs auf die OpenAFS Server

3.2 Kompilation der benötigten Software

Software, die für viele Plattformen und viele Hosts zur Verfügung gestellt werden soll, wird am DESY Zeuthen unter */afs/ifh.de/products/<name>* installiert. Die Konfiguration der Hosts erfolgt über Cfengine[1]. Wichtige Software, die auch ohne laufendes AFS funktionieren soll, wird dann von diversen Cfengine-Features auf die lokalen Platten der entsprechenden Hosts kopiert. Diese angestrebte Unabhängigkeit vom AFS bringt jedoch die Notwendigkeit mit, Software statisch zu linken. Deshalb ist es nötig, für Software, die gegen Bibliotheken aus dem AFS gelinkt wird, im Makefile immer die statische Variante dieser Bibliothek anzugeben. Im Klartext heißt das, alle Vorkommen von *-l<libname>* durch */pfad/zu/statischer/Bibliothek* zu ersetzen. Nach dem Linken ist der Erfolg dieser Aktion immer mit **ldd** <**Programmname**> zu überprüfen.

3.2.1 KTH's Version von Kerberos4 („Athena")

Um Heimdal die Möglichkeit zu geben, Kerberos4 Anfragen zu beantworten, muß es gegen die Kerberos4 Bibliothek der KTH gelinkt werden. Am DESY Zeuthen war bereits eine Version installiert, jedoch hoffnungslos veraltet. Aktuell ist zum Zeitpunkt dieser Abhandlung Version 1.2.2. Unter Linux ist das Kompilieren kein Problem. Hier kommt der gcc 2.95.3 zum Einsatz. Unter Solaris muß mit SUNs Workshop Compiler gebaut werden, da es zu Problemen mit anderer Software kommt, die gegen diese Bibliothek gelinkt wird, falls sie sich nicht ebenfalls mit dem gcc kompilieren läßt.

Die AFS-Bibliotheken werden automatisch erkannt (solange sie sich an Standardpositionen im Verzeichnisbaum befinden). Daß die Suche erfolgreich war, erkennt man an der zusätzlich gebauten libkafs. Diese stellt die Schnittstelle zu AFS her (z.B. erzeugt die Funktion *krb_afslog* aus einem gültigen Kerberos4 TGT ein AFS Token).

Die **configure** Parameter unter Linux sehen wie folgt aus:

```
CFLAGS="-I/products/BerkeleyDB/<BDB Version>/include" \
LDFLAGS="-L/products/BerkeleyDB/<BDB Version>/lib" \
./configure --prefix=/products/kerberos/<Krb4 Version> \
--enable-rxkad --without-ipv6 \
--with-openssl=/products/ssl/<OpenSSL Version>
```

Da wir eine eigene Version von BerkeleyDB pflegen und nicht die Systembibliotheken benutzen wollen, muß dies dem Preprozessor und dem Linker mitgeteilt werden. Die aktuelle Version von Kerberos4 sowie auch von Heimdal lassen sich (ungepatcht) nur gegen Version 3 oder älter von BerkeleyDB linken. In Version 4 gibt es Modifikationen im API, die ein Binden verhindern. Desweiteren ist auch von OpenSSL eine eigene Version vorhanden.

[1] http://www.iu.hio.no/cfengine/

3.2.2 KTH's Version von Kerberos5 („Heimdal")

Nun kann mit der eigentlichen Kerberos5 Suite fortgefahren werden. Eingesetzt wird die Version 0.5.1. Beim **configure** ist zu beachten, daß es nicht reicht, *–with-krb4=path/to/krb4* anzugeben. Die Kerberos4 Bibliotheken werden zumindest in dieser Version nicht ordentlich eingebunden, falls *–with-krb4-config=path/to/krb4-config* fehlt.

Wenn die Kerberos4 Bibliotheken AFS erfolgreich gefunden haben, gibt es nun auch keine Probleme damit. Auch jetzt wird wieder eine libkafs gebaut. Diese steht jedoch in Konkurrenz zu der Version von Kerberos4. Wenn weitere Software kompiliert wird, ist darauf zu achten, daß immer gegen die Heimdalversion gelinkt wird. Diese stellt unter anderem die Funktion *krb5_afslog* zur Verfügung, die aus einem gültigen Kerberos5 TGT ein AFS Token generiert. Falls also gegen die alte Version gelinkt wird, entstehen unaufgelöste Funktionsaufrufe.

Es folgt der **configure** Aufruf unter Linux:

```
CFLAGS="-I/products/BerkeleyDB/<BDB Version>/include" \
LDFLAGS="-L/products/BerkeleyDB/<BDB Version>/lib" \
./configure --without-ipv6 \
--with-openssl=/products/ssl/<OpenSSL Version>
--with-krb4=/products/kerberos/<Krb4 Version>
--with-krb4-config=/products/kerberos/<Krb4 Version>/bin/krb4-config \
--prefix=/products/heimdal/<Heimdal Version>
```

Auch hier sind wieder die Verweise auf die eigenen BerkeleyDB und OpenSSL Bibliotheken zu finden.

3.2.3 MIT's Version von Kerberos5

Ein Großteil der Kerberos5-fähigen Software läßt sich jedoch nicht gegen die Heimdalbibliotheken bauen. Deswegen ist eine Nutzung der MIT-Kerberos5 Distribution oft notwendig. Diese kann und sollte von http://www.crypto-publish.org bezogen werden. Grund sind die schon in 1.5 erwähnten US-amerikanischen Ausfuhrbestimmungen für starke Kryptographie. Diese wurden zwar gelockert, jedoch wird beim Download von MIT's Homepage eine Erklärung verlangt, daß man sich in US-amerikanischen bzw. kanadischen Staatsgebiet aufhält. Auf der „Cryptography Publishing Project" Seite fällt dies weg.

Das Übersetzen von MIT's Kerberos5 gestaltet sich problemlos. Mit dem **configure**-Schalter *–with-krb4* werden auch Bibliotheken gebaut, um abwärtskompatibel mit Kerberos4 zu bleiben.

```
./configure --prefix=/products/kerberos5/<Krb5 Version> --with-krb4
```

3.2.4 Ein Kerberos5-fähiges PAM Modul

Um den Nutzer beim Anmelden immer ein gültiges TGT sowie AFS Token zur Verfügung zu stellen, wird ein PAM-Modul benötigt. PAM[2] steht am DESY Zeuthen für alle Linux- und Solarisplattformen zur Verfügung. Es wurde entworfen um PAM-fähige Applikationen mit den unterschiedlichsten Authentisierungsmethoden kompatibel zu machen. Kerberos bildet da keine Ausnahme.

Wir entschieden uns für Nalin Dahyabhai's Modul, das von Balazs Gal mit Heimdal kompatibel gemacht wurde. Es kann von http://sourceforge.net/projects/pam-krb5/ bezogen werden. In der aktuellen Version arbeitet es sowohl unter Linux als auch unter Solaris. Gebaut wurde es wie folgt:

```
./configure --prefix=/usr \
--with-krb5=/products/heimdal/<Heimdal Version> \
--with-krbafs=/products/heimdal/<Heimdal Version> \
--with-krb4=/products/kerberos/<Krb4 Version>
```

3.2.5 OpenSSH

Die vorher am DESY Zeuthen eingesetzte OpenSSH-Version war nur mit Kerberos4 kompatibel. Außerdem waren einige Patches vom CERN[3] enthalten, die Probleme beim AFS Tokenforwarding mit älteren OpenSSH-Versionen aus dem Weg räumten.

Nun kommt die Version 3.6.1p1 mit einigen Modifikationen zum Einsatz. Die erste Erweiterung ist der Patch http://meta.cesnet.cz/software/heimdal/openssh-heimdal.patch, der es ermöglicht, problemlos gegen Heimdal zu bauen. Dieser wurde eigentlich für die OpenSSH Version 3.4p1 geschrieben, läßt sich aber ohne Probleme auch auf die zur Zeit aktuelle Version 3.6.1p1 anwenden. Ohne diese Erweiterung ist es nur möglich, gegen MIT zu linken. Dies bringt jedoch Probleme mit dem Ticketforwarding mit sich: Einmal geforwardete Tickets sind nicht mehr forwardable – ein inakzeptabler Zustand.

Doch selbst mit Patch funktioniert die Kerberos- und AFS-Unterstützung nur über Protokollversion 1. Diese hat jedoch Sicherheitsprobleme (Vgl. http://www.openssh.com/de/security.html). Es ist über einen weiteren Patch (http://meta.cesnet.cz/software/heimdal/gssapi_ext.en.html) möglich, Kerberos5-Unterstützung über GSSAPI und Protokoll 2 in OpenSSH einzupflegen. Ich habe dies jedoch noch nicht probiert und weiß nicht, wie und ob das AFS Token Forwarding damit funktioniert.

Die aktuelle Version von OpenSSH besitzt noch immer einen Bug im Code des Ticketforwardings. Besitzt ein Nutzer ein TGT einer Realm A und verbindet sich mit einem Host B in Realm C, wird nach erfolgreicher Authentisierung versucht, das Ticket von Realm A

[2] Pluggable Authentication Modules
[3] Conseil Européen pour la Recherche Nucléaire (http://www.cern.ch)

auf Host B zu forwarden. Dies schlägt jedoch fehl und wird auf der Serverseite mit einem fatalen Programmabruch quittiert. Als Workaround bleibt dem Nutzer nur übrig, vor dem Verbinden **kdestroy** aufzurufen. Ich habe den Code in *sshconnect1.c* soweit modifiziert, daß nur noch nach erfolgreicher Kerberos5 Authentisierung das TGT weitergeleitet wird.

OpenSSH wird hier mit folgenden Optionen konfiguriert:

```
./configure --prefix=/usr --sysconfdir=/etc/ssh \
--libexecdir=/usr/bin --with-ipv4-default \
--with-tcp-wrappers --with-pam \
--with-kerberos4=/products/kerberos/<Krb4 Version> \
--with-kerberos5=/products/heimdal/<Heimdal Version> \
--with-ssl-dir=/products/ssl/<OpenSSL Version> \
--with-afs --with-cflags=-Wno-pointer-arith
```

Configure setzt neben der OpenSSL libcrypto auch die libdes in das Makefile als einzubindene Bibliotheken. Das sie aber beide teilweise die selben Funktionen zur Verfügung stellen, gibt es Probleme mit doppelt vorhandenen Symbolen. Ein Entfernen aller Vorkommen von *-ldes* aus dem Makefile bringt Abhilfe, da libcrypto alle benötigten Funktionen allein zur Verfügung stellt.

3.3 Aufsetzen des KDC

Es wird nun beschrieben, wie das Heimdal KDC im Testbetrieb am DESY Zeuthen aufgesetzt wurde. Die nun folgenden ersten Schritte können noch nicht auf dem selben Host ausgeführt werden, auf dem auch der OpenAFS Server läuft. OpenAFS' **kaserver** belegt den gleichen Port, den auch das Heimdal KDC belegen will. Dies funktioniert natürlich nicht! Trotzdem sollte zum Testen ein Rechner genommen werden, der ähnlich aufgesetzt ist, wie der spätere Produktionsserver.

Da, wie bereits erwähnt, selbstkompilierte Software hier standardmäßig im AFS liegt und für wichtige Dienste keine Abhängigkeit davon vorhanden sein darf, besteht der erste Schritt aus dem Kopieren der kompletten Heimdaldistribution auf die lokale Platte des Servers nach */usr/heimdal*. Dies geschieht hier automatisch durch ein Cfengine-Feature.

3.3.1 Initialisierung der Kerberos5 Datenbank

Auf Heimdalservern liegen die Daten standardmäßig unter */var/heimdal*. Es empfielt sich, dieses Verzeichnis nur für root zugänglich zu machen (Rechtemaske 0700). Zum Initialisieren der Datenbank wird das Programm **kadmin** mit dem Schalter *-l* (für lokal) benutzt. Mit dem Kommando **init** <**REALNAME**> wird eine neue Realm in der Datenbank */var/heimdal/heimdal.db* angelegt. Dabei werden eine systemnotwendige Principals hinzugefügt:

- **krbtgt/<REALNAME>@<REALNAME>:**
  Ticket-Granting-Service (TGS)

- **kadmin/admin@<REALNAME>:**
  Datenbankadministrationsservice (benutzt von **kadmind**)

- **kadmin/changepw@<REALNAME>:**
  Paßwortänderungsservice (benutzt von **kpasswdd**)

- **default@<REALNAME>:**
  Default Principal

Es wird nun empfohlen, die Datenbank mittels eines Masterkeys zu verschlüsseln. Dieser wird mit dem Programm **kstash** erzeugt und unter */var/heimdal/m-key* abgelegt. Auf diese Weise kann die Datenbank (oder Sicherungskopien dieser) auch an weniger gesicherten Stellen aufbewahrt werden – auch wenn dies trotzdem nicht zu empfehlen ist. Ein unverschlüsseltes Kopieren der Datenbank ist bei Heimdal unnötig. Mit **hprop** kann sie problemlos verschlüsselt über ein ungesichertes Netzwerk kopiert werden.

3.3.2 Konvertierung der existierenden AFS Datenbank

Zum Konvertieren der vorhandenen AFS KA-Datenbank in eine Heimdaldatenbank habe ich das Perlskript **heimdalsync** von Wolfgang Friebel genutzt. Es kann von ftp://ftp.ifh.de/pub/unix/kerberos/heimdalsync herunter geladen werden und arbeitet zusammengefaßt wie folgt:

1. Kopieren der AFS Datenbank auf den lokalen Host.
2. Dumpen der aktuellen Heimdaldatenbank.
3. Umwandlung der AFS Datenbank in eine Heimdaldatenbank mittels **hprop**.
4. Vergleich der einzelnen Schlüssel der alten Datenbank mit der neuen Datenbank: geänderte sowie neue Einträge werden in eine Mergedatei namens *dbdiff* geschrieben, nicht mehr vorhandene zum Löschen vorgemerkt.
5. Zum Löschen vorgemerkte Einträge werden aus der Originaldatenbank mit **kadmin delete** entfernt.
6. Aufruf von **kadmin merge** → die Einträge in der Originaldatenbank werden mittels der Mergedatei *dbdiff* aktualisiert.

Ein großer Vorzug von Heimdal ist, daß dies während des laufenden Betriebes des KDC stattfinden kann. Auf diese Weise ist es möglich die Heimdaldatenbank periodisch auf den neuesten Stand zu bringen, ohne den Service zu unterbrechen.

Durch das Zusammenführen der beiden Datenbanken werden auch die Schlüssel der Dienste *afs@IFH.DE* (zum Erwerb von AFS Tokens) und *krbtgt/IFH.DE@IFH.DE* in die Heimdaldatenbank übernommen bzw. überschrieben. Bei letzterem kommt es dadurch zu einem Problem, da AFS, wie in 2.3.3 beschrieben, den Schlüssel periodisch ändert. **kadmin merge** kann den alten Schlüssel nicht aufbewahren, er ist unwiederbringlich verloren. Fordert ein Nutzer nun kurz vor dem Zusammenführen ein neues TGT via z.B. **kinit** an, ist dieses danach für die Anforderung von Kerberos5 Service Tickets unbrauchbar – der entsprechende Schlüssel fehlt in der Datenbank. Der Nutzer merkt dies natürlich nur, wenn er Dienste nutzen möchte, die nur eine Kerberos5 Authentisierung anbieten. Ein Fallback auf Kerberos4 Authentisierung funktioniert weiterhin, da AFS' **kaserver** den alten Schlüssel bis zum Ablauf seiner Maximallebensdauer in einer speziellen Warteschlange noch vorrätig hält. Muß der Nutzer jedoch einen Dienst nutzen, der nur Kerberos5 versteht, hilft nur ein erneutes **kinit**. Dieses Problem besteht natürlich nur solange zwei getrennte Datenbanken existieren.

3.3.3 Anpassen der Konfigurationsdatei */etc/krb5.conf*

Jetzt muß */etc/krb5.conf* an die lokalen Gegebenheiten angepaßt werden. Eine Musterdatei liegt der Heimdaldistribution bei. Sie ist als Vorlage benutzbar. Die Datei besteht aus mehreren Abschnitten, in denen Attribute gesetzt sind. Die wichtigsten Punkte sollen jetzt erläutert werden. Eine Komplettbeschreibung ist mit **man krb5.conf** erhältlich.

[libdefaults]
Hier stehen alle Konfigurationsparameter, die bereits von den Kerberosbibliotheken genutzt werden.

default_realm
NÖTIG! Der Name der lokalen Kerberos Realm.

default_keytab_name
Die Datei, aus der Dienste ihre Principalschlüssel holen können. Default: */etc/krb5.keytab*

ticket_lifetime
Die standardmäßige Lebenszeit von Tickets. Endungen wie h → Stunden und d → Tage sind möglich. Ohne Endung ist es die Zeit in Sekunden.

renew_lifetime
Die standardmäßige Zeit, in der Tickets erneuert werden können. Auch hier sind Endungen möglich. Die aktuelle Version des PAM-Moduls aus 3.2.4 kommt damit seltsamerweise nicht klar. Die Tickets sind dann überhaupt nicht erneuerbar. Wenn hier die Sekunden angegeben werden, funktioniert es jedoch.

krb4_get_tickets
Gibt an, ob bei einer TGT-Anforderung über z.B. **kinit** automatisch auch ein TGT der Version 4 generiert wird.

forwardable
Falls gesetzt, werden standardmäßig weiterleitbare TGTs ausgestellt.

default_etypes
Liste von Verschlüsselungstypen, die bei einer Ticketanforderung verwendet werden.

clockskew
Bestimmt die maximale Zeitdifferenz, die der Zeitstempel im Ticket und der im Authentikator auseinander liegen dürfen. Standardmäßig sind das fünf Minuten.

capath
Hier kann der Pfad von Realms bei einer Cross-Realm Authentisierung angegeben werden: Auflistung von „Zielrealm = nächster Sprung (Realmname) auf dem Weg zur Zielrealm“. Siehe hierzu auch 1.4.2 und Abbildung 1.5.

[realms]
Hier kommen die nötigen Daten hin, die ein Client braucht, Tickets für die aufgelisteten Realms anzufordern. Heimdalclients unterstützen auch die dynamische Konfiguration über Nameserviceeinträge (siehe 3.4.3).

kdc
Mit Leerzeichen getrennte Liste von Hosts auf denen ein KDC läuft.

kadmin_server
Hostname des Master-KDC (nur dort läuft der **kadmind**).

kpasswd_server
Hostname des Master-KDC

default_domain
Name der Domain. Wird genutzt, Namen von Service-Principals der Version 4 (besitzen keine FQHN[4]) in Principalnamen der Version 5 (besitzen FQHN) und umgekehrt umzuwandeln.

v4_name_convert
Gegenüberstellung, wie Principalnamen der Version 4 in der Version 5 heißen. Hauptsächlich für den Hostservice genutzt: *rcmd* → *host.*

check-ticket-addresses
Legt fest, ob Adressen im TGT bei der Anforderung eines Service Tickets überprüft werden sollen. Ist nur für Kerberos4 und Kerberos5 gemeinsam möglich!

check-ticket-addresses
Gibt an, ob Tickets ohne enthaltene Adresse erlaubt sind.

[domain_realm]
Mapping von DNS Domainnamen auf Realmnamen (Hosts in einer Domain gehören zu welcher Realm?).

[4]Fully qualified host names

[kdc]

require-preauth
Aktiviert grundsätzlich Preauthentication. Da Clients der Version 4 (z.B. klog) dies nicht beherrschen, muß es abgeschaltet bleiben!

enable-kerberos4
Aktivierung der Kerberos4 Emulation im KDC.

enable-kaserver
Aktivierung der AFS KA-Server Emulation im KDC.

enable-524
Aktivierung des 524-Dienstes im KDC (Umwandlung von TGTs der Version 5 in Version 4.

v4-realm
Nötig solange Kerberos4 Emulation eingeschaltet ist. Der Realmname für Kerberos4 TGTs bei nativen TGT Anforderungen der Version 4.

[kadmin]

default_keys
Liste von Saltmethoden bei der Schlüsselerstellung. Solange *v4* dabei ist, können auch native Kerberos4 Clients Tickets entschlüsseln.

[logging]
Hier kann das Loggingverhalten der einzelnen Dienste angegeben werden (Level, Datei oder Syslog). Auch ein Defaulteintrag für alle Dienste ist möglich.

[appdefaults]
Unter diese Sektion kommen Einstellungen für verschiedene kerberisierte Dienste (z.B. PAM), die diese Datei über Kerberos Bibliotheksfunktionen auswerten.

3.3.4 Starten der notwendigen Dienste

Damit auf dem Server alle Dienste angeboten werden können, müssen zuerst die systemnotwendigen Schlüssel mittels **kadmin ext** in die lokale Keytabdatei extrahiert werden. Dies sind folgende:

- kadmin/admin@IFH.DE
- kadmin/hprop@IFH.DE
- kadmin/changepw@IFH.DE
- changepw/kerberos@IFH.DE

Nun muß eine gültige Konfigurationsdatei unter */etc/krb5.conf* abgespeichert werden. Wie sie am DESY Zeuthen aussieht, ist im Anhang B.1 nachlesbar. Desweiteren wird ein symbolischer Link */var/heimdal/kdc.conf* angelegt, der auf */etc/krb5.conf* zeigt.

Jetzt kann ein erster Testlauf stattfinden: Mit **/usr/heimdal/libexec/kdc &** wird das Key Distribution Center hochgefahren. Da der Serverhost nun zum Anfordern von Kerberos5-Tickets fertig konfiguriert ist, sollte dies jetzt auch getestet werden:

```
[zeus] ~ % kinit ahaupt
ahaupt@IFH.DE's Password:
[zeus] ~ % klist
Credentials cache: FILE:/tmp/krb5cc_oZ6688
        Principal: ahaupt@IFH.DE

  Issued           Expires          Principal
May 20 11:34:22  May 21 12:34:22  krbtgt/IFH.DE@IFH.DE
May 20 11:34:22  May 21 12:34:22  afs@IFH.DE

   V4-ticket file: /tmp/tkt9132_6688
        Principal: ahaupt@IFH.DE

  Issued           Expires          Principal
May 20 11:34:22  May 21 13:00:43  krbtgt.IFH.DE@IFH.DE
```

Wenn es im großen und ganzen wie die obige Ausgabe aussieht, ist das KDC korrekt konfiguriert! Sollten Probleme auftreten, kann in */var/heimdal/kdc.log* bzw. im Syslog – je nachdem wie es konfiguriert ist – nach der Ursache gefahndet werden.

Nachdem dies funktioniert, sollten für die zukünftigen Datenbankadministratoren Principals mit der Instanz *admin* angelegt werden. Im Fall des Accountnamen *ahaupt* würde es wie folgt gemacht:

```
[zeus] / # kadmin -l add --use-defaults ahaupt/admin
ahaupt/admin@IFH.DE's Password:
Verifying password - ahaupt/admin@IFH.DE's Password:
```

Mit dem Schalter *–use-defaults* wird der Principal mit den Standardoptionen angelegt. Es folgt die Abfrage nach dem Anfangspaßwort des Principals. Dieses sollte natürlich sorgfältig ausgewählt werden. Ein Check auf zu einfache Paßwörter findet hier nicht statt. Danach muß der neu hinzugekomme Administrationsprincipal noch in die Datei */var/heimdal/kadmind.acl* eingetragen und mit allen Rechten ausgerüstet werden:

```
ahaupt/admin@IFH.DE all
```

Nachdem nun der kadmin Daemon mit **/usr/heimdal/libexec/kadmind &** gestartet wurde, kann in einer anderen Konsole versucht werden, die neu erworbenen Rechte zu testen:

```
[zeus] ~ % kinit ahaupt/admin
ahaupt/admin@IFH.DE's Password:
[zeus] ~ % kadmin get ahaupt
               Principal: ahaupt@IFH.DE
       Principal expires: never
        Password expires: never
    Last password change: never
         Max ticket life: 1 day 1 hour
      Max renewable life: 1 month
                    Kvno: 7
                   Mkvno: 0
                  Policy: none
   Last successful login: never
       Last failed login: never
      Failed login count: 0
           Last modified: 2003-05-20 10:50:09 UTC
                Modifier: kadmin/admin@IFH.DE
              Attributes:
Keytypes(salttype[(salt-value)]): des-cbc-md5(afs3-salt(ifh.de)),
des-cbc-md4(afs3-salt(ifh.de)), des-cbc-crc(afs3-salt(ifh.de))
```

Sollte hierbei keine Fehlermeldung erscheinen, hat auch dies funktioniert! Der Paßwortänderungsservice **/usr/heimdal/libexec/kpasswdd** wird noch nicht gebraucht, da dies noch in der AFS KA-Datenbank geschieht und diese Änderungen erst nachträglich mit **heimdalsync** in die Heimdaldatenbank übernommen werden.

Jetzt ist der Zeitpunkt gekommen, die beiden Dienste automatisch starten zu lassen. Dies wird während der Testphase noch mit einem Init-Skript z.B. */etc/init.d/heimdal* getan. Während des Produktionsbetriebs ist es empfehlenswert, die Kontrolle OpenAFS' **bosserver** zu übergeben.

3.3.5 Aufsetzen eines Slave-KDC

Da im späteren Produktionsbetrieb auch mehrere KDCs nach einem Master/Slave- Konzept arbeiten, ist es ratsam, dieses Zusammenspiel während der Testphase zu untersuchen. Nachdem auf dem Host *zeus* das KDC bereits läuft, soll nun auf *hera* das Slave-KDC installiert werden. Bevor damit begonnen werden kann, sind jedoch einige Vorarbeiten nötig.

Auf *hera* muß eine gültige */etc/krb5.conf* vorhanden sein. Am einfachsten ist es, die vorhandene vom Master zu übernehmen. Desweiteren ist hier das Verzeichnis */var/heimdal* und der Link */var/heimdal/kdc.conf* auf */etc/krb5.conf* zeigend anzulegen. Außerdem muß der Masterkey */var/heimdal/m-key* von *zeus* auf einem sicheren Weg (z.B. via **scp**) auf den Slave transferiert werden. Natürlich wird auch wieder die komplette Heimdaldistribution auf die lokale Platte kopiert. In der Heimdaldatenbank sind nun die Serviceprincipals *iprop/zeus.ifh.de*, *iprop/hera.ifh.de* sowie *hprop/hera.ifh.de* anzulegen:

```
[zeus] ~ % kadmin add -r --use-defaults iprop/zeus.ifh.de \
iprop/hera.ifh.de hprop/hera.ifh.de
```

Die neu erzeugten Schlüssel müssen nun in die Keytabdateien auf *zeus* und *hera* extrahiert werden. Auf *zeus* braucht nur *iprop/zeus.ifh.de* nach */etc/krb5.keytab*:

```
[zeus] ~ # kadmin ext iprop/zeus.ifh.de
```

Auf *hera* müssen alle Principals, die *hera.ifh.de* als Instanz haben, in die Keytabdatei:

```
[hera] ~ # kadmin ext iprop/hera.ifh.de hprop/hera.ifh.de
```

Jetzt wird auf *zeus* die Datei */var/heimdal/slaves* mit folgendem Inhalt angelegt:

```
iprop/hera.ifh.de@IFH.DE
```

Auf *hera* wird nun einmalig **/usr/heimdal/libexec/hpropd** als Nutzer *root* gestartet. Auf *zeus* wird folgendes ebenfalls als Nutzer *root* ausgeführt: **/usr/heimdal/sbin/hprop hera**. Damit wird die Datenbank komplett und verschlüsselt auf *hera* übertragen. Dabei tritt hier der Effekt auf, daß die neu erzeugte Slavedatenbank um ca. 20% größer ist, als die ursprüngliche. Die Ursache dafür habe ich nicht verstanden.

Als letzte Aktion vor dem Start der inkrementellen Propagierung ist eine leere Datei */var/heimdal/log* auf *zeus* und *hera* anzulegen. Diese enthält später ein Journal über die noch ausstehenden Änderungen auf der Slavedatenbank. Auf dem Slave ist dies eigentlich unnötig, jedoch beschwert sich der Master nachdem er die Datei nicht findet. Vielleicht wird das in einer zukünftigen Version behoben.

Jetzt kann der letzte Schritt unternommen und der Propagierungsmaster auf *zeus* mit **/usr/heimdal/libexec/ipropd-master &** gestartet werden. Auf der *hera* geschieht das gleiche, jedoch mit dem Propagierungsslave: **/usr/heimdal/libexec/ipropd-slave zeus.ifh.de &**. Es ist zu beachten, daß hier als Parameter nur der „Fully qualified host name“ – also mit Domainnamen und auch kein Hostalias erlaubt ist. Der Parameter wird 1:1 zum Erstellen des Serviceprincipalnamen genommen – und dieser heißt *iprop/zeus.ifh.de*.

War dieser Schritt erfolgreich wurden auf *zeus* ein Unixsocket */var/heimdal/signal*, über den dem Propagierungsmaster jede Änderung an der Datenbank mitgeteilt wird, sowie die Datei */var/heimdal/slave-stats* angelegt. Liest man sie aus, sollte in etwa folgendes zu sehen sein:

```
Status for slaves, last updated: 2003-05-20 15:54:48

Master version: 4

Name                         Address            Version  Status  Last Seen
iprop/hera.ifh.de@IFH.DE   IPv4:141.34.22.11        4  Up      2003-05-20 14:53:16
```

Im Code der inkrementellen Propagierung gibt es jedoch scheinbar noch einen Bug. Bestätigt hat mir das auf der Heimdal Mailingliste niemand, meine E-Mail blieb unbeantwortet. Solange nach jedem Start des Propagierungsmasters **/usr/heimdal/sbin/truncate_log** nicht aufgerufen wird, wächst das Journal auf dem Slave nach Änderungen an der Masterdatenbank hier nur an, die Slavedatenbank bleibt unverändert. Die Ursache dafür läßt sich durch den Aufruf von **/usr/heimdal/sbin/replay_log** auf dem Slave herausfiltern:

```
ver 3... done
ver 4... replay_log: kadm5_log_replay: No such entry in the database
done
ver 5... done
```

Mit **/usr/heimdal/sbin/dump_log** erkennt man, daß an dieser Stelle ein Principal aus der Datenbank gelöscht werden soll, der in der Slavedatenbank nicht (mehr) existiert. Dieser wurde bereits vor dem Kopieren der gesamten Datenbank auf *hera* aus der Masterdatenbank entfernt. Scheinbar funktioniert die initiale Abstimmung aufeinander nicht richtig – dies wird mit **truncate_log** behoben. Das Journal hat danach eine Größe von 24 Bytes, und diese bleibt zumindest auf dem Slave bei Änderungen an der Masterdatenbank bestehen.

3.4 Kerberisierung der Hosts in der Realm

Nachdem auf zwei Servern Heimdal-KDCs laufen, sollen auch alle anderen Hosts Kerberos5 nutzen können. Die Clientsoftware ist im AFS verfügbar, es reicht nun, die bereits vorhandene */etc/krb5.conf* überall zu verteilen. Damit ist es allen Maschinen möglich, das KDC einheitlich anzusprechen. Es gibt jedoch noch andere Dinge, die getan werden sollten. Diese stelle ich jetzt noch vor.

3.4.1 PAM Konfiguration

PAM bietet, wie schon erwähnt, die Möglichkeit verschiedenste Authentisierungsmechanismen und Loginprogramme zu vereinheitlichen und zentral zu warten. Es bietet dabei den Programmen und dem Nutzer völlige Transparenz. PAM ist eine generische zentrale Schnittstelle, unter deren Überwachung viele voneinander unabhängige Module ihren Dienst versehen. Eine Pflichtlektüre zum Verständnis von Linux-PAM ist http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/. Ich werde auf die Funktionsweise nur grob eingehen.

PAM liegt sowohl für Linux als auch für Solaris vor. Die beiden Implementationen unterscheiden sich jedoch teilweise erheblich, was sich auch auf die Konfiguration auswirkt. Außerdem ist die Variante, die Solaris mitliefert, ziemlich fehlerbehaftet. Die

Arbeitsweise folgt teilweise nicht der Dokumentation, die Solaris zur Verfügung stellt (http://wwws.sun.com/software/solaris/pam/).

Um beim Login (über welche Methode auch immer) dem Nutzer ein Kerberos TGT sowie ein AFS Token zur Verfügung zu stellen, muß hier nur das alte Modul von OpenAFS *pam_afs.krb.so* durch das neue *pam_krb5afs.so* in den Konfigurationsdateien unter */etc/pam.d/* für Linux und in */etc/pam.conf* auf Solaris ersetzt werden. Am DESY Zeuthen ist für eine Übergangszeit die automatische Beschaffung von Kerberos4 TGTs notwendig. Zu diesem Zweck wird in */etc/krb5.conf* im Abschnitt *applications* ein PAM-Eintrag hinzugefügt und dort

```
krb4_convert = true
```

gesetzt. Im Anhang B.1 kann dies nachgeschlagen werden.

Das Modul unterstützt Authentisierung, Accountmanagement, Paßwortänderung und Sessionverwaltung. In letzterem sorgt es automatisch dafür, daß alle Tickets und Tokens nach dem Ausloggen entfernt werden. Eine Beispielkonfiguration für den Loginservice liegt in B.2. Ein Erneuern der Credentials mit einem paßwortgesicherten Bildschirmschoner ist ebenfalls möglich (siehe B.3).

3.4.2 SSH Konfiguration

Nachdem OpenSSH gebaut wurde, sind nun Anpassungen in der SSH-Konfiguration sowohl auf der Client- als auch auf der Serverseite nötig. Eigentlich hatte ich keine Arbeit damit, da bereits vorher Kerberos4 und AFS Unterstützung konfiguriert war und die Einstellungen sich nicht geändert haben. Trotzdem sollen die notwendigen Einstellungen kurz erläutert werden. Es folgen die wichtigen Auszuge aus der Datei */etc/ssh/ssh_config*.

```
Host *
    Protocol 1,2
    KerberosAuthentication yes
    KerberosTgtPassing yes
    AFSTokenPassing yes
```

Wie bereits in 3.2.5 erwähnt, ist Kerberos Unterstützung in OpenSSH nur über Protokoll 1 möglich. Deshalb wird immer versucht, sich zuerst über das veraltete Protokoll zu verbinden. Unterstützt die Gegenstelle das nicht, geschieht ein „Rückfall“ auf Protokoll 2. Der Eintrag *KerberosAuthentication* ermöglicht das paßwortlose Einloggen auf anderen Hosts in der Realm. Dazu wird vom KDC unter Vorlage des TGT ein *host*-Service Ticket ausgestellt, das der Authentisierung dient. *KerberosTgtPassing* schaltet das Weiterleiten des TGT auf den entfernten Rechner ein. Zu guter letzt wird das Forwarden aller AFS Tokens mit *AFSTokenPassing* aktiviert.

All diese Dinge muß aber auch der Server unterstützen. Sonst wird den Bitten des Clients nicht nachgekommen. Die Optionen heißen hier gleich wie auf der Clientseite:

```
Protocol 1,2
KerberosAuthentication yes
KerberosTgtPassing yes
AFSTokenPassing yes
```

3.4.3 DNS Einträge

Eine sehr schöne Sache ist die mögliche Auflösung von Kerberos-Servern über DNS Einträge. Dies ist für die Hosts der eigenen Domain egal, da sie ihre Informationen aus */etc/krb5.conf* beziehen. Für Clients in fremden Domains ist es aber ein Schmankerl, da dort die Daten unserer Realm nicht konfiguriert werden müssen. In diesem Zuge ist es auch ratsam, Aliasnamen für die KDC Hosts zu vergeben. Dadurch ist maximale Flexibilität gesichert. Am DESY Zeuthen sind das *kdc1*, *kdc2* usw.

Nach Internetdraft[2] lauten die Einträge am DESY Zeuthen wie folgt:

```
_kerberos._udp.IFH.DE.  IN      SRV     0 0 88 kdc1.ifh.de.
_kerberos._udp.IFH.DE.  IN      SRV     1 0 88 kdc2.ifh.de.
_kerberos._tcp.IFH.DE.  IN      SRV     0 0 88 kdc1.ifh.de.
_kerberos._tcp.IFH.DE.  IN      SRV     1 0 88 kdc2.ifh.de.
```

Diese Einträge folgen RFC 2052, welches den Aufbau der DNS Konfigurationdatei beschreibt. Da Kerberos die zwei Übertragungsprotokolle TCP und UDP unterstützt, muß für jedes Protokoll und jeden Host ein Eintrag vorhanden sein[5]. Das erste Feld setzt sich aus dem Servicenamen (immer *_kerberos*), dem Protokoll und dem Realmnamen zusammen. Feld 4 beschreibt die Priorität, mit der dieser Eintrag behandelt wird. In Feld 6 ist der Port gesetzt, auf dem das KDC lauscht. Der letzte Eintrag nennt den Namen des KDC.

3.5 Anpassen der vorhandenen Skripte

Das Heimdal KDC kann einen AFS KAServer emulieren, jedoch nicht vollständig. Ihm fehlt eine Implementation des KA_MAINTENANCE_SERVICE. Dieser Service wird vom AFS Programm **kas** genutzt, um die AFS Authentisierungsdatenbank zu modifizieren. Als Schlußfolgerung daraus müssen alle **kas** Aufrufe durch **kadmin** Aufrufe ersetzt werden, welches das Pendant auf der Heimdalseite darstellt.

Am DESY Zeuthen wird **kas** jedoch nicht direkt aufgerufen. Stattdessen existieren Perl-Skripte, die mit Hilfe von Norbert Grüners Perlschnittstelle AFS (http://www.cpan.org/authors/id/N/NO/NOG/AFS-2.03.tar.gz) ansprechen. So werden

[5] In Wirklichkeit ist nur UDP Pflicht. Versteht das KDC jedoch auch TCP, sollte das natürlich nicht fehlen.

u.a. die Funktionen Nutzer anlegen, Nutzer löschen, Nutzer modifizieren und Nutzerpaßwort ändern unter Zuhilfenahme dieses Paketes ausgeführt. Diese mußten nun angepaßt werden. Für die Bearbeitung der Heimdaldatenbank aus Perl heraus gibt es ebenfalls ein Paket: ftp://ftp.su.se/pub/users/leifj/Heimdal-Kadm5-0.2.tar.gz Damit können die ehemaligen KAS-Aufrufe durch native Heimdalaufrufe ersetzt werden. Es besteht nun sogar die Möglichkeit Nutzer zu sperren! Dies ist mit der Authentisierungsdatenbank von AFS nicht möglich. Das Paket hat zur Zeit aber noch Probleme mit der Paßwortauthentisierung. Eine Authentisierung über Keytabdateien, wie sie hier nun eingesetzt wird, funktioniert jedoch problemlos. Auch das Hinzufügen von Principals zur Datenbank funktioniert nicht ohne Patch. Dieser kann von ftp://ftp.ifh.de/pub/unix/kerberos bezogen werden.

Ein weiterer Punkt ist der Paßwortänderungsservice via **kpasswd** aus der OpenAFS Distribution. Dieser muß ebenfalls durch Heimdals eigenes **kpasswd** ersetzt werden. Dies betrifft uns aber nicht, da hier, wie erwähnt, ein anderer Weg zur Paßwortänderung der Nutzer genommen wird.

3.6 Der letzte Schritt

Bis jetzt läuft die Kerberos5 Authentisierung über die Heimdaldatenbank noch völlig unabhängig von AFS. Es wird jedoch eine Integration von Heimdal in OpenAFS angestrebt. Deshalb erfolgt nun die Abschaltung aller **kaserver** Instanzen. Deren Aufgabe übernimmt das Heimdal KDC nativ auf dem AFS Server. Sollte man mehrere AFS Datenbankserver in Betrieb haben, muß überlegt werden, auf welchem Host künftig die Masterdatenbank läuft. Da bei AFS grundsätzlich der Host mit der niedrigsten IP Adresse Master ist, sollte dies für Heimdal ebenso übernommen werden. Das ist *fred. wilma* und *pepples* werden als Slaveserver aufgesetzt.

Bevor mit der eigentlichen Arbeit begonnen wird, sollte sicher gestellt werden, daß die Original Heimdaldatenbank, wie auch die AFS KA-Datenbank von nun an nicht mehr von Skripten, Cron Jobs, etc. verändert wird. Dies bedeutet, daß insbesondere der Paßwortänderungsservice zu unterbrechen ist. Auf diese Weise kann eine minimale Ausfallzeit für den Authentisierungsservice erreicht werden.

Jetzt folgt der Aufruf von **heimdalsync** auf dem alten Masterserver *zeus* ein letztes Mal. Daraufhin werden die Serviceprincipals für die Dienste *hprop* und *iprop* für alle zukünftigen Datenbankserver erstellt und entsprechend dem Bedarf in die Keytabdateien auf *fred*, *wilma* und *pepples* extrahiert. Auf *fred* gehören natürlich auch die für den Masterdatenbankserver notwendigen Schlüssel mit hinein:

- kadmin/admin@IFH.DE
- kadmin/hprop@IFH.DE
- kadmin/changepw@IFH.DE

- changepw/kerberos@IFH.DE

Es folgen die Schritte, wie sie schon unter 3.3 beschrieben wurden: Kopieren der Distribution auf die lokalen Platten, Master- und Slavedatenbankserver für den späteren Betrieb vorbereiten (Konfiguration des */var/heimdal* Verzeichnisses auf den Rechnern entsprechend ihrer späteren Aufgabe).

Nachdem dies erfolgt ist, wird die Heimdaldatenbank von *zeus* zusammen mit dem Masterkey auf die zukünftigen Datenbankserver kopiert. Jetzt ist es an der Zeit alle **kaserver** Instanzen abzuschalten:

```
[fred] ~ # bos stop fred kaserver -localauth
[fred] ~ # bos stop wilma kaserver -localauth
[fred] ~ # bos stop pepples kaserver -localauth
```

Da sie zukünftig auch nicht mehr vom **bosserver** überwacht werden müssen, können sie auch gleich aus *BosConfig* entfernt werden:

```
[fred] ~ # bos delete fred kaserver -localauth
[fred] ~ # bos delete wilma kaserver -localauth
[fred] ~ # bos delete pepples kaserver -localauth
```

Nun werden die Heimdalkomponenten unter die Kontrolle des **bosserver** gestellt:

```
[fred] ~ # bos create fred kdc simple /usr/heimdal/libexec/kdc -localauth
[fred] ~ # bos create wilma kdc simple /usr/heimdal/libexec/kdc -localauth
[fred] ~ # bos create pepples kdc simple /usr/heimdal/libexec/kdc -localauth
[fred] ~ # bos create fred kadmind simple /usr/heimdal/libexec/kadmind -localauth
[fred] ~ # bos create fred kpasswdd simple /usr/heimdal/libexec/kpasswdd \
-localauth
[fred] ~ # bos create fred ipropd-master simple \
/usr/heimdal/libexec/ipropd-master -localauth
[fred] ~ # bos create wilma ipropd-slave simple \
/usr/heimdal/libexec/ipropd-slave fred.ifh.de -localauth
[fred] ~ # bos create pepples ipropd-slave simple \
/usr/heimdal/libexec/ipropd-slave fred.ifh.de -localauth
```

Nachdem dies vollbracht ist, sollte nachgesehen werden, ob alle Services, wie gewollt, gestartet wurden (hier am Beispiel von *fred*):

```
[fred] / # bos status fred
Instance ptserver, currently running normally.
Instance vlserver, currently running normally.
Instance backupusers, currently running normally.
    Auxiliary status is: run next at Fri May 23 01:00:00 2003.
Instance fs, currently running normally.
    Auxiliary status is: file server running.
Instance kdc, currently running normally.
Instance kadmind, currently running normally.
Instance ipropd-master, currently running normally.
Instance kpasswdd, currently running normally.
```

Sieht es ähnlich dieser Ausgabe aus, ist alles gut gegangen! Auf *fred* wird nun noch einmal **/usr/heimdal/sbin/truncate_log** gestartet (Vgl. 3.3.5) und alles läuft soweit.

3.7 Abschließende Bemerkungen

Die letzten Abschnitte sollten eine Schritt-für-Schritt Anleitung für die Integration von Heimdal in OpenAFS darstellen. Falls alles so wie beschrieben funktioniert hat, vielleicht auch ohne einige der angeschnittenen Probleme, ist die Migration geglückt. Heimdal läuft zum Zeitpunkt dieser Niederschrift seit ungefähr zwei Monaten im Produktionsbetrieb ohne Probleme. Es kann also als stabil eingestuft werden. Nichtsdestotrotz funktionieren auch aufgrund der noch niedrigen Versionsnummer einige Dinge nicht so, wie erwartet:

- Wie bereits in 3.3.5 erwähnt, funktioniert hier die inkrementelle Propagierung nicht „einfach so“. Es sind lästige und meiner Meinung nach unnötige Umschiffungen nötig, die hoffentlich in einer der nächsten Versionen wegfallen.
- Die GSSAPI Implementation in Heimdal ist noch nicht vollständig[6]. Viele Applikationen nutzen diesen Mechanismus für die Authentisierung. Sie können jedoch oft nicht gegen Heimdal’s Implementation gelinkt werden. An dieser Stelle muß wieder auf MIT’s Paket zurückgegriffen werden.
- Manche Dinge sind nach außen hin von MIT’s Distribution abgeleitet. So sind die Parameter und Ausgaben von vielen Programmen identisch oder sehr ähnlich auch bei Heimdal anzutreffen. Dies stiftet in manchen Fällen jedoch Verwirrung, wenn einige angezeigte Features überhaupt noch nicht implementiert sind. Folgendes Beispiel soll das verdeutlichen.

```
[fred] ~ % kadmin get ahaupt
               Principal: ahaupt@IFH.DE
       Principal expires: never
        Password expires: never
    Last password change: never                <-- nicht implementiert
         Max ticket life: 1 day 1 hour
      Max renewable life: 1 month
                    Kvno: 7
                   Mkvno: 0
                  Policy: none                 <-- nicht implementiert
   Last successful login: never                <-- nicht implementiert
       Last failed login: never                <-- nicht implementiert
      Failed login count: 0                    <-- nicht implementiert
           Last modified: 2003-05-20 10:50:09 UTC
                Modifier: kadmin/admin@IFH.DE
              Attributes:
Keytypes(salttype[(salt-value)]): des-cbc-md5(afs3-salt(ifh.de)),
des-cbc-md4(afs3-salt(ifh.de)), des-cbc-crc(afs3-salt(ifh.de))
```

Sollten nicht lösbare Probleme auftreten, bei denen nicht einmal Google[7] helfen kann, ist die Heimdal Mailingliste (Instruktionen zum Einschreiben in Anhang A.3) eine sehr gute Anlaufstelle, in der im allgemeinen sehr kompetente Hilfe geboten wird. Es ist sowieso eine gute Idee, als Betreiber eines Heimdal KDC die Liste mit zu verfolgen.

[6] Obwohl mit Erscheinen der Version 0.6 eine starke Verbesserung sichtbar ist

[7] http://www.google.de

Anhang A

Was noch fehlt

A.1 Glossar

ASN.1: Abkürzung für Abstract Syntax Notation One. Es ist eine Abstrahierungssprache für Datentypen und Werte. Mittels ASN.1 können komplexe Objekte durch viel einfachere Typen zusammengesetzt werden.

Dazu gibt es auch Lektüre im Internet:

- Burton S. Kaliski Jr., „A Layman's Guide to a Subset of ASN.1, BER, and DER“
  <ftp://ftp.rsa.com/pub/pkcs/ps/layman.ps>
- Brian Tung, „ASN.1: Wherefore Art Thou?“
  <http://www.isi.edu/ brian/security/asn1.html>

Forwardable Tickets: Werden beschrieben unter 1.4.2. Soll ein TGT auf einem anderen Host, als auf dem es erworben wurde, genutzt werden, muß es weiterleitbar „forwardable“ sein.

GSSAPI: Abkürzung für „Generic Security Service Application Interface“. GSSAPI ist eine generische Schnittstelle für den Programmierer, sichere Authentisierung in seiner Client/Server Applikation zu integrieren. Welche Methode zur Anmeldung letztendlich genutzt wird, braucht ihn nicht zu interessieren. Unter anderem kann Kerberos als Backend für GSSAPI genutzt werden. Somit sind alle Applikationen, die GSSAPI Authentisierung anbieten, kerberisiert.

Siehe dazu auch die RFCs im Anhang A.4.

Heimdal: KTH's Implementation von Kerberos5.

KDC: Abkürzung für Key Distribution Center.

KVNO: Abkürzung für Key Version Number.

KTH: Abkürzung für „Kungliga Tekniska Högskolan“, das Schwedische Technologische Institut. Hier wird Heimdal hauptsächlich entwickelt.

MIT: Abkürzung für „Massachusetts Institute of Technology“. Das MIT bietet ebenfalls eine Kerberos5 Distribution an.

Postdatable Tickets: Wird in 1.4.3 erklärt. Diese Tickets können zu einem späteren Zeitpunkt als der Anforderungszeit gültig werden.

Renewable Tickets: Wird ebenfalls in 1.4.3 erklärt. Diese Tickets sind über ihre normalerweise begrenzte Lebenszeit hinaus benutzbar.

SASL: Abkürzung für „Simple Authentication and Security Layer“. Mittels SASL kann in verbindungsorientierte Protokolle die Unterstützung verschiedener Authentisierungsmechanismen (unter anderem GSSAPI) eingebaut werden. Falls gewünscht, ist es möglich, zwischen dem Protokoll und der eigentlichen Verbindung eine Sicherungsschicht einzufügen.

Näheres ist im Internet unter http://asg.web.cmu.edu/sasl/ nachlesbar. Siehe auch das passende RFC im Anhang A.4.

TGS: Abkürzung für Ticket Granting Service.

TGT: Abkürzung für Ticket Granting Ticket. Dient zum Anfordern von weiteren Service Tickets.

A.2 Links im Internet

- Homepage von Heimdal:
  <http://www.pdc.kth.se/heimdal/>
- Homepage von MIT Kerberos:
  <http://web.mit.edu/kerberos/www/>
- Homepage von OpenAFS:
  <http://www.openafs.org/>
- Ein gute Einleitung zu Kerberos ist „The Moron’s Guide to Kerberos“:
  <http://www.isi.edu/gost/brian/security/kerberos.html>
- Eine weitere Einleitung zu AFS und Kerberos findet man hier:
  <http://grand.central.org/twiki/bin/view/AFSLore/?topic=KerberosAFSInstall>
- Eine Studienarbeit, die das gleiche Thema wie diese Diplomarbeit hat, wurde an der TU-Chemnitz geschrieben:
  <http://archiv.tu-chemnitz.de/pub/1998/0042/data/starbeit.ps>
- Patches und Skripte, die am DESY Zeuthen für bestimmte Aufgaben nötig wurden, sind auf dessen FTP Server zu finden:
  <ftp://ftp.ifh.de/pub/unix/kerberos/>

A.3 Mailinglisten

- Heimdal Diskussionsliste:
  - Einschreiben über mailto:heimdal-discuss-request@sics.se mit „subscribe“ im Body.
  - Posten an die Liste: mailto:heimdal-discuss@sics.se
- MIT Diskussionsliste:
  - Es existieren mehrere Listen: siehe http://web.mit.edu/kerberos/www/mail-lists.html
  - Beiträge der allgemeinen Liste werden gleichzeitig ins Usenet (*comp.protocols.kerberos*) gepostet und umgekehrt.
- OpenAFS Diskussionsliste:
  - Einschreiben über ein Webformular auf https://lists.openafs.org/mailman/listinfo/openafs-info.
  - Posten an die Liste: mailto:openafs-info@openafs.org

A.4 RFCs zum Nachlesen

- Alle Anforderungen an Kerberos5 sind in RFC 1510 zusammengefaßt: <http://www.ietf.org/rfc/rfc1510.txt>
- In folgendem Dokument sind die GSSAPI Dienste beschrieben: <http://www.ietf.org/rfc/rfc1508.txt>
- Von vorherigem RFC gibt es auch eine neuere Version: <http://www.ietf.org/rfc/rfc2078.txt>
- ...und noch ein weiteres Update: <http://www.ietf.org/rfc/rfc2743.txt>
- Der Kerberos5 GSS-API Mechanismus ist hier aufgeführt: <http://www.ietf.org/rfc/rfc1964.txt>
- SASL ist in RFC 2222 spezifiziert: <http://www.ietf.org/rfc/rfc2222.txt>

A.5 Selbständigkeitserklärung

Hiermit erkläre ich, daß ich die vorliegende Diplomarbeit selbständig angefertigt und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Zeuthen, den 26. Februar 2004

Anhang B

Beispielkonfigurationsdateien

B.1 /etc/krb5.conf

```
[libdefaults]
    default_realm = IFH.DE
    default_etypes = des-cbc-crc
    default_etypes_des = des-cbc-crc
    ticket_lifetime = 25h
    renew_lifetime = 30d
    krb4_get_tickets = true
    forwardable = true
[realms]
    IFH.DE = {
        kdc = kdc1.ifh.de kdc2.ifh.de kdc3.ifh.de
        admin_server = kdc1.ifh.de
        kpasswd_server = kdc1.ifh.de
        default_domain = ifh.de
        v4_name_convert = {
            host = {
                rcmd = host
                ftp = ftp
                imap = imap
            }
        }
    }
[domain_realm]
    .ifh.de = IFH.DE
[kadmin]
    default_keys = v4 v5
[kdc]
    enable-kerberos4 = true
    enable-kaserver = true
    enable-524 = true
    v4-realm = IFH.DE
[logging]
    kdc = 0-1/SYSLOG:INFO:AUTH
    default = 0-1/SYSLOG:INFO:USER

[appdefaults]
    pam = {
        krb4_convert = true
    }
```

B.2 /etc/pam.d/login

```
#%PAM-1.0
auth        sufficient /lib/security/pam_unix.so    nullok #set_secrpc
auth        required   /lib/security/pam_krb5afs.so use_first_pass ignore_root
# dependencies: /etc/securetty file
auth        required   /lib/security/pam_securetty.so
# dependencies: /etc/nologin
auth        required   /lib/security/pam_nologin.so

account     required   /lib/security/pam_unix.so

password    required   /lib/security/pam_unix.so    strict=false

session     optional   /lib/security/pam_krb5afs.so
session     required   /lib/security/pam_unix.so    none # debug or trace
session     required   /lib/security/pam_limits.so
```

B.3 /etc/pam.d/xscreensaver

```
#%PAM-1.0
auth     sufficient /lib/security/pam_unix.so
auth     required   /lib/security/pam_krb5afs.so use_first_pass ignore_root \
force_creds refresh_creds
```

Literaturverzeichnis

[1] Steven M. Bellovin and Michael Merritt. Limitations of the kerberos authentication system. Technical report, AT&T Bell Laboratories, 1991.

[2] Ken Hornstein and Jeffrey Altman. *Distributing Kerberos KDC and Realm Information with DNS*, September 2000. Internet Draft draft-ietf-cat-krb-dns-locate-02.txt.

[3] B. Jaspan. *Kerberos users' frequently asked questions*, 2.0 edition, August 2000. Periodically posted to Usenet newsgroup comp.protocols.kerberos.

[4] B. Clifford Neuman John T. Kohl and Theodore Y. T'so. The evolution of the kerberos authentication system, 1994.

[5] John Kohl and B. Clifford Neuman. *The Kerberos Network Authentication Service (Version 5)*, September 1993. Internet Request for Comments RFC-1510.

[6] Norman Schulze. Afs-kerberos-5-migration, Oktober 1998. Studienarbeit.