TECHNISCHE UNIVERSITÄT CHEMNITZ
Fakultät für Informatik
Professur für Betriebssysteme

Diplomarbeit

Sicherheitsaspekte von Instant Messaging

Verfasser	Holger Schildt <holger.schildt@s1999.tu-chemnitz.de>
Betreuender Hochschullehrer	Prof. Dr. Winfried Kalfa Technische Universität Chemnitz
Betreuer	Dipl. Math. Thomas Häberlen Bundesamt für Sicherheit in der Informationstechnik
Datum	10. Juli 2005

Danksagung

Diese Diplomarbeit wäre ohne die Unterstützung vieler Menschen in dieser Form nicht möglich gewesen:

Spezieller Dank geht an Herrn Prof. Kalfa, der kurzfristig die Betreuung dieser Diplomarbeit übernommen hat. Seine Anregungen und Hinweise haben die Qualität dieser Diplomarbeit entscheidend verbessert.

Ich danke auch meinem Mentor Thomas, der mir nicht nur während der Diplomarbeit hilfreich zur Seite stand. Zwischen dem Beginn meines Praktikums und dem Abschluss dieser Diplomarbeit hat er Einfluss auf mein Studium genommen und mich insbesondere für den Bereich IT-Sicherheit sensibilisiert.

Ohne Connie würde diese Diplomarbeit auch anders aussehen, wofür ich ihr sehr danke. Sie hat mich nicht nur auf inhaltliche Unklarheiten hingewiesen und viel Zeit mit Korrekturlesen verbracht, sondern hat mir während der gesamten Bearbeitung den Rücken freigehalten.

Natürlich danke ich auch allen Freunden, die während der gesamten Diplomarbeit Interesse an deren Zustand gezeigt haben. An Heiko und Michael geht mein besonderer Dank.

Schließlich möchte ich meinen Eltern für die finanzielle und ideelle Unterstützung des Studiums danken.

Abkürzungsverzeichnis

(S)FTP	(Secure) File Tranfer Protocol
AES	Advanced Encryption Standard
AIM	AOL Instant Messenger
ARP	Address Resolution Protocol
ASCII	American Standard Code for Information Interchange
BOS	Basic Oscar Service
DES	Data Encryption Standard
DNS	Domain Name Service
DoS	Denial of Service
DSS	Digital Signature Standard
HMAC	Hashed Message Authentication Code
HTTP(S)	HyperText Transfer Protocol (secure)
ID	Identificator
IM	Instant Messaging
IRC	Internet Relay Chat
IV	Initialisierungsvektor
JID	Jabber Identificator
LDAP	Lightweight Directory Access Protocol
MAC	Message Authentication Code *oder* Media Access Control
MIME	Multipurpose Internet Mail Extensions
MSN	Microsoft Network
NAT	Network Address Translation
OSCAR	Open System for Communication in Realtime
PAM	Pluggable Authentication Module
PGP	Pretty Good Privacy
PKCS	Public Key Cryptographic Standard
RFC	Requests for Comments
SASL	Simple Authentication and Security Layer
Silc	Secure Internet Live Conferencing
SKE	Silc Key Exchange
SMS	Short Message Service
SSI	Server Side Information
SSL	Secure Sockets Layer
TLS	Transport Layer Security
TLV	Type Length Value
TOC	Talk to OSCAR
TrID	Transaction Identificator
URL	Uniform Resource Locator
UTF	Unicode Transformation Format
WLAN	Wireless Local Area Network
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

Abstract

Instant Messaging besitzt zweifelsohne eine sehr große Zukunft. Es ist vergleichbar mit dem *Short Message Service* (SMS), der einen sehr großen Stellenwert im Mobilfunk hat. Neben der für SMS charakteristischen schnellen Übermittlung von Textnachrichten bietet *Instant Messaging* weitere Vorteile: es ist für den Sender auf den ersten Blick ersichtlich, ob der Gesprächspartner für eine Kommunikation zur Verfügung steht. Neben Textnachrichten können aber auch Informationen wie Bilder, Videos oder andere Formate übermittelt werden.

Auch für Organisationen ist ein Einsatz von *Instant Messaging* sehr interessant. Besonders in diesem Zusammenhang spielt die Sicherheit eine sehr wichtige Rolle. Daher beschäftigt sich diese Diplomarbeit vorrangig mit *Instant Messaging*-Systemen hinsichtlich der Prinzipien der Systemsicherheit wie Integrität, Vertraulichkeit und Verfügbarkeit.

Ein weiterer Aspekt ist das Fehlen eines standardisierten *Instant Messaging*-Protokolls. Viele Anbieter von *Instant Messaging*-Systemen halten ihr System für das Beste und versuchen, soviele Kunden wie möglich an sich zu binden. Daher befasst sich diese Diplomarbeit ebenfalls mit den verbreitetsten *Instant Messaging*-Systemen, um deren Vor- und Nachteile herauszukristallisieren. Im Weiteren wird die Funktionsweise der Protokolle erklärt.

Gerade im privaten Bereich spielen Konferenzsysteme („Chat“) eine wichtige Rolle. Neben den *Instant Messaging*-Systemen wird auch diese Möglichkeit der Kommunikation beleuchtet.

Diese Diplomarbeit wird vom Bundesamt für Sicherheit in der Informationstechnik (BSI), Referat „Grundschutz und Systemsicherheit“, betreut.

Aufgabenbeschreibung

Instant Messaging (IM) ermöglicht den Benutzern eine zeitnahe Kommunikation. Da IM in einigen Anwendungsfällen das Potential hat, effizienter als E-Mail zu sein, soll dieser Dienst anhand der Grundprinzipien der IT-Sicherheit Verfügbarkeit, Vertraulichkeit und Integrität untersucht werden.

Von verschiedenen Anbietern wurden zahlreiche, untereinander nicht kompatible *Instant Messaging*-Protokolle entwickelt. Im Rahmen dieser Arbeit sollen die bekanntesten Protokolle analysiert werden. Hierzu zählen ICQ, AIM, MSN und *Jabber*. Obwohl es sich bei IRC und SILC um keine klassischen IM-Protokolle handelt, soll eine kurze Betrachtung dieser beiden Kommunikationsprotokolle die Analyse abschließen.

Die Spezifikationen einiger IM-Protokolle sind nicht frei verfügbar. Um die Grundfunktionalitäten dieser Protokolle dennoch zu analysieren wird auf *Reverse Engineering* zurück gegriffen. Damit die Untersuchung transparent gehalten werden kann, wird frei verfügbare Analysesoftware verwendet.

Nach der Analyse der Protokolle sollen anhand von einfachen Beispielen die Schwächen der Protokolle hinsichtlich Verfügbarkeit, Vertraulichkeit und Integrität betrachtet werden. Wie viele Netzprotokolle sind IM-Protokolle durch *Denial-of-Service*-Angriffe und Lauschangriffe verwundbar, wobei letzteres besonders für nicht kryptographisch geschützte Protokolle relevant ist. Um diese Angriffe nachvollziehbar zu machen, werden die Angriffsmethoden, wie *Spoofing*, kurz vorgestellt. Vorhandene (öffentlich zugängliche) Analysen zu Sicherheitsaspekten sollen recherchiert und die daraus resultierenden Erkenntnisse in die Arbeit einbezogen werden.

Angriffe auf Protokolle, die durch kryptographische Verfahren geschützt werden, sind möglich, wenn die verwendeten Algorithmen oder die Implementierung Schwachstellen aufweisen. Bei diesen Protokollen werden die verwendeten Algorithmen und ihr Einsatz kurz vorgestellt.

Neben der Analyse der Protokolle soll mindestens ein IM-Server betrachtet werden. Für die Analyse wurde *Jabber* ausgewählt, da dieser Server als freie Software im Quellcode zur Verfügung steht. Dabei wird besonderer Wert auf die sichere Konfiguration des Servers gelegt. Falls Schwachstellen entdeckt werden, sollen diese ebenfalls beleuchtet werden. Eine Fähigkeit von *Jabber* ist die Interoperabilität zu anderen IM-Protokollen. Diese soll ebenfalls untersucht werden. Zusätzlich spielt für eine Sicherheitsanalyse die Kommunikation zwischen mehreren *Jabber*-Servern eine entscheidende Rolle.

Neben der technischen Funktionalität von IM-Diensten sollen ebenfalls organisatorische und personelle Aspekte der Nutzung von IM in Einrichtungen betrachtet werden. Hierbei sollen auch Gefährdungen wie *Social Engineering* betrachtet werden. Außerdem soll anhand von Beispielen demonstriert werden, welche Sicherheitsaspekte bei der Benutzung von IM in Unternehmensnetzen zu beachten sind und wie die Nutzung gegebenenfalls unterbunden werden kann.

Diese Arbeit wird in Zusammenarbeit mit dem Bundesamt für Sicherheit in der Informationstechnik (BSI) bearbeitet.

Inhaltsverzeichnis

Teil I.
Allgemeines

1. Instant Messaging und Konferenzsysteme

Diese Diplomarbeit befasst sich mit der Beschreibung und Analyse von Kommunikationsprotokollen. Daher muss frühzeitig geklärt werden, was hier unter einem Kommunikationsprotokoll zu verstehen ist.

In erster Linie dienen Kommunikationsprotokolle zur Übermittlung von Textnachrichten über ein Netz. Im Gegensatz zu E-Mail können die Verzögerungen bei der Nachrichtenzustellung viel geringer sein, da diese Systeme auf einen direkten Dialog zwischen den Kommunikationspartnern ausgerichtet sind. Dabei kann es sich um eine 1:1-Verbindung (also zwischen zwei Benutzern) oder um eine 1:n-Kommunikation (Gruppendiskussion) handeln.

Textnachrichten reichen aber sehr oft nicht für eine effiziente Kommunikation aus. Skizzen oder Gesten des Gesprächspartners können zur sinnvollen Verständigung beitragen. Daher bieten viele der hier vorgestellten Technologien die Möglichkeit, neben Text auch andere Formate zu übermitteln. Da die Systeme aber mit dem Ziel, Textnachrichten zu übermitteln, entworfen wurden, konzentriert sich diese Diplomarbeit auf die Übertragung dieser.

In dieser Arbeit werden die Kommunikationssysteme in zwei Gruppen betrachtet: ***Instant Messaging*-Systeme** und **Konferenzsysteme**. Beide Systeme werden im Laufe dieses Kapitels näher beleuchtet. Der Unterschied zwischen beiden Kommunikationssystemen ist nicht immer klar zu definieren. Einige *Instant Messaging*-Systeme bieten auch Funktionalitäten von Konferenzsystemen an.

Der Einsatz von Kommunikationssystemen ist nicht auf Computer im klassischen Sinne beschränkt. Auch Laptops und PDAs können mit entsprechender Netzanbindung (beispielsweise *Wireless LAN*) diese Systeme nutzen. In naher Zukunft könnten auch Mobiltelefone in diesen Einsatzbereich vordringen und *Instant Messaging* könnte den heute sehr beliebten *Short Message Service* (SMS) auf Grund seines größeren Funktionsumfangs verdrängen.

Umfang eines Kommunikationssystems Endanwender unterscheiden nicht zwischen Klienten, Protokollen und Servern. Analog hierzu wird dieses gesamte Gebilde in dieser Diplomarbeit als **Kommunikationssystem** bezeichnet (siehe Abbildung 1). Hierzu gehören die Softwareschnittstelle, an der der Benutzer seine Anweisungen übergibt, die Instanz, die diese in Protokollanweisungen umwandelt (**Klienteninstanz**), das eigentliche Protokoll und eine Gegenstelle, die analog zu der Klienteninstanz die übertragenen Informationen in Anweisungen umwandelt und ausführt (**Serverinstanz**).

In dieser Diplomarbeit werden die ***Instant Messaging*-Systeme** ICQ, *AOL Instant Messenger* (AIM), *Microsoft Network Messenger* und *Jabber* beleuchtet. Die Betrachtung der **Konferenzsysteme** beschränkt sich auf IRC und Silc.

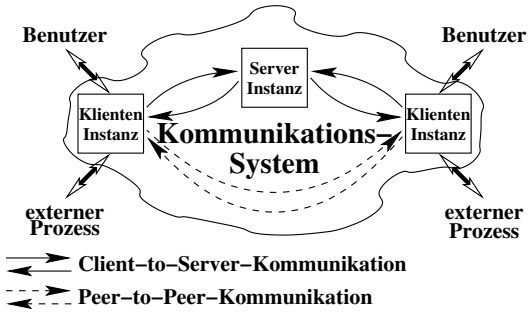

Abbildung 1: Prinzipieller Aufbau eines Kommunikationssystems

Netze Abbildung 2 zeigt eine mögliche Struktur eines Netzes mit öffentlichen Servern, die häufig bei proprietären Systemen vorzufinden sind. Zu unterscheiden ist hierbei zwischen lokalen Netz (LAN, *Local Area Network*), in dem sich die Klienteninstanzen befinden und dem Netzsegment mit den Serverinstanzen. Das lokale Netz befindet sich in der Regel unter der Kontrolle eines **lokalen Netzbetreibers** beziehungsweise Administrators und analog hierzu wird das Server-Netzsegment vom **Serverbetreiber**/Administrator gepflegt.

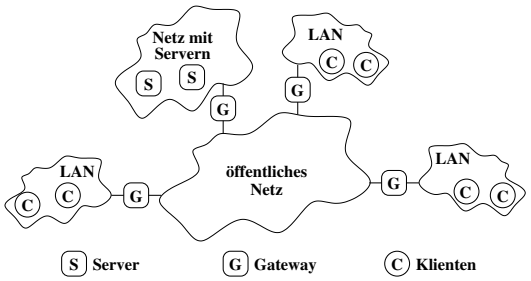

Abbildung 2: Beispiel eines öffentlichen Netzes

Es existieren mehrere unabhängige lokale Netze mit Klienteninstanzen, von denen Verbindungen zu den Servern aufgebaut werden. Je nach System befinden sich alle Server in einem Netz oder in verschiedenen Segmenten. Erlaubt es das Kommunikationssystem, einen eigenständigen Server zu betreiben, kann sich dieser auch im lokalen Netz befinden. Die genaue Struktur dieser Netze wird für die einzelnen Kommunikationssysteme in späteren Kapiteln beleuchtet.

Zwischen diesen Netzen befindet sich ein öffentliches Netz, auf das weder der Betreiber des lokalen Netzes noch der des Servers Einfluss besitzen. Daher wird das öffentliche Netz auch als unsicheres Netz bezeichnet, da ein Angreifer dort leichter Angriffe auf die Integrität, Verfügbarkeit und Vertraulichkeit gegen die Verbindung durchführen kann. Die Anbindung des lokalen Netzes beziehungsweise des Netzes mit den Servern an das öffentliche Netz geschieht über *Gateways*.

Sowohl Server als auch Klienten sind die Endpunkte einer Kommunikation und Instanzen des Kommunikationssystems. Der Unterschied zwischen beiden Systemen besteht darin, dass ein Klient mit einem **Benutzer** oder einem **externen Prozess** kommuniziert, wogegen ein Server die Kommunikation der einzelnen Klienteninstanzen untereinander regelt. Des Weiteren verwaltet

ein Server die Daten der Klienten, die ihm nach oder während der Registrierung bekannt gegeben wurden.

Bei der Kommunikation über Kommunikationssysteme sind zwei Architekturmodelle, die ebenfalls in Abbildung 1 zu sehen sind, zu unterscheiden. In der Regel ist einer der (Gesprächs)-Partner der Klient und der zweite ein Server. Bei dieser **Client-to-Server**-Architektur wird eine Nachricht vom Klienten zum Server geschickt, der diese Nachricht dann bearbeitet. In der Regel umfasst die Bearbeitung das Weiterleiten der Nachricht zu dem vom Sender gewünschten Empfänger.

Diese **Client-to-Server**-Anordnung ist nicht immer sinnvoll. Wenn sich zwei Kommunikationspartner in einem LAN befinden, schickt der eine Klient die Nachricht zu dem Server, der die Nachricht wieder an den anderen Klienten zurückschickt. Da der Weg zum Server über viele Router gehen kann, hätte ein Angreifer viele Möglichkeiten, die Nachricht abzufangen. Im Weiteren ist es nicht besonders elegant, wenn die Nachrichten zwischen zwei Klienten, die vielleicht nur wenige Meter voneinander entfernt sind, sehr weite Strecken zurücklegen müssen.

Im Gegensatz dazu bieten einige Kommunikationssysteme die Möglichkeit, Nachrichten über **Peer-to-Peer**-Verbindungen zu übertragen. Dabei wird zwar in den meisten Fällen ein Server benötigt, um die IP-Adressen und Portnummern der Klienten auszutauschen, aber der eigentliche Nachrichtenaustausch findet direkt zwischen den beiden Klienten statt. Es ist zu beachten, dass *Network Address Translation* (NAT)-Router solche Verbindungen behindern können.

1.1. Instant Messaging

Instant Messaging (IM) kann sehr gut mit dem E-Mail-Dienst verglichen werden: der Sender schickt dem Empfänger, der über einen eindeutigen Identifikator bestimmt wird, eine Nachricht, die dieser dann direkt lesen kann. Neben der direkten Zustellung spielt die geringe Verzögerung ebenfalls eine entscheidende Rolle. Bei vielen E-Mail-Implementationen kann es hingegen vorkommen, dass die E-Mail über einen längeren Zeitraum auf einem Server zwischengespeichert wird, wenn eine der Zwischenstationen nicht erreichbar ist.

Da *Instant Messaging* im Gegensatz zu E-Mail noch nicht so lang verbreitet ist, hat sich noch kein einheitliches und ausgereiftes Protokoll herauskristallisiert.

IM-Instanzen Eine Klienteninstanz ermöglicht dem Benutzer die Nutzung des *Instant Messaging*-Systems. Auf dieses können aber auch externe Prozesse zugreifen: es könnte eine Benachrichtigung von einem Script beim Eintreten eines bestimmten Ereignisses, wie einen Systemausfall oder einen erkannten Angriff, erfolgen.

Daher ist der von außen sichtbare Teil einer Klienteninstanz die **externe Schnittstelle**, mit der der Benutzer oder eine andere Applikation kommunizieren können. Dies kann ein Fenster sein, in das der Benutzer seine Anweisungen eingibt oder Softwarebibliotheken, die die Einbindung in fremden Applikationen ermöglicht.

Über diese Schnittstelle erfolgt die Kommunikation mit der eigentlichen **Verwaltungsinstanz**. Hierzu gehören Elemente zum Versand und Empfang von Nachrichten und Zuständen. Auf den Begriff Nachricht (und dessen Format) und auf Zuständen wird in den folgenden Abschnitten näher eingegangen.

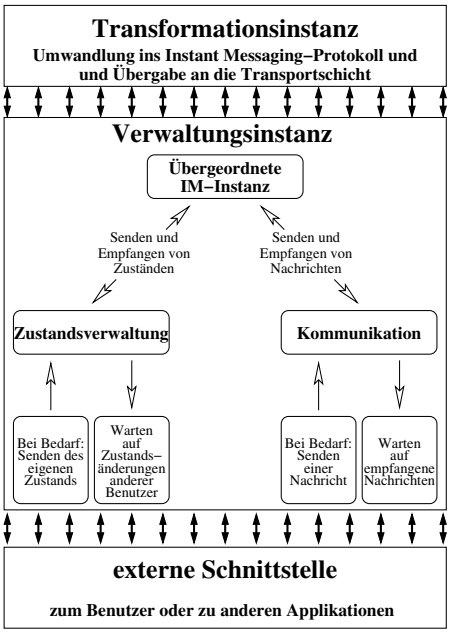

Abbildung 3: Bestandteile einer IM-Instanz

Die Verwaltungsinstanz übermittelt die vom Benutzer eingegebenen Informationen an eine **Transformationsinstanz**, die diese in Anweisungen für das *Instant Messaging*-Protokoll umwandelt und an eine Instanz der Transportschicht weiterleitet. Informationen für den Benutzer gelangen ebenfalls über diese Schnittstelle zur Verwaltungsinstanz. Ein Modell einer IM-Instanz zeigt Abbildung 3.

In der Regel wird die gesamte Applikation direkt von dem Betreiber des *Instant Messaging*-Systems dem Benutzer zur Verfügung gestellt, damit dieser diese auf seinem Rechner installieren kann. Alternativ könnte auch ein webbasierter Klient genutzt werden.

Bei **webbasierten Klienten** handelt es sich um Instanzen, die dem Anwender über einen HTTP-Server Zugang zu dem Kommunikationssystem ermöglichen. Dabei greift der Benutzer mit seinem *Browser* auf den Server zu, der wiederum eine Verbindung zu dem eigentlichen Server des Kommunikationssystems aufbaut. Durch dynamisch generierte HTTP-Seiten stehen dem Benutzer alle Funkionalitäten des Systems zur Verfügung, ohne dass dieser eine Klienteninstanz installieren muss. Diese Server können direkt vom Betreiber des Systems oder von Dritten bereit gestellt werden.

Bei vielen Implementationen läuft die Klienteninstanz im Hintergrund - beim Empfang einer Nachricht erscheint diese dann direkt auf der virtuellen Arbeitsfläche. Die Nachrichten werden dabei im so genannten *Push*-Verfahren an den Empfänger übermittelt. Anders als bei E-Mail muss der Empfänger den Server nicht explizit nach neuen Nachrichten fragen. Solange die Klienteninstanz am Server angemeldet ist, benachrichtigt sie dieser eigenständig über neu eingetroffene Mitteilungen.

Nachrichtenformat Ursprünglich wurden IM-Systeme zur Übermittlung von Textnachrichten zwischen Benutzern entwickelt. Viele Systeme bieten aber zusätzlich noch die Möglichkeit, mit mehreren Benutzern gleichzeitig zu kommunizieren (*Chat*), Dateien zu übermitteln, Bilder zu versenden, zu telefonieren oder eine Videokonferenz zu führen. Möglich wäre auch noch die Bereitstellung einer *Whiteboard*-Funktionalität, die durch die Erstellung von Skizzen die Kommunikation vereinfacht. In dieser Arbeit wird aber primär auf die Grundfunktionalität eingegangen, dem Versenden von Textnachrichten.

Kontaktlisten und Zustände Bei IM spielen die so genannten Kontaktlisten (oder auch *Buddy*-Listen) eine zentrale Rolle: der Benutzer kann seine Gesprächspartner (**Kontakte**) in diesen Listen verwalten. Sobald sich einer der eingetragenen Kontakte an dem Server anmeldet, wird dies dem Benutzer signalisiert. So sieht der Benutzer ohne eine weitere Anfrage (zum Beispiel durch ein Telefongespräch oder einen Blick ins Büro des Kontakts), ob der Kontakt kommunikationsbereit ist. Wenn der Benutzer nun eine Nachricht an den Kontakt, der sich angemeldet hat, sendet, ist die Wahrscheinlichkeit hoch, dass dieser sofort antworten kann.

Die verwalteten Kontakte können nach bestimmten Eigenschaften zu Benutzergruppen zusammengefasst werden. Beispiele hierfür sind Gruppen für Familienangehörige und Freunde oder eine Anordnung nach geschäftlichen Strukturen. Der Vorteil dieser Anordnung besteht in der Übersichtlichkeit und in der vereinfachten Möglichkeit, eine Nachricht direkt an eine Gruppe und nicht an jede Person einzeln zu schicken. Einige *Instant Messaging*-Protokolle sehen diese Integration von Gruppen direkt vor - bei anderen wird diese Funktionalität nur vom Klienten ausgeführt.

Um über IM zu kommunizieren, müssen die Gesprächspartner bei vielen *Instant Messaging*-Systemen nicht dauerhaft mit dem zu dem System gehörenden Server verbunden sein. Verschickt ein Benutzer eine Nachricht an einen anderen, der zur Zeit nicht angemeldet ist, wird die Nachricht wie bei E-Mail auf dem Server zwischengespeichert.

Zusätzlich kann jeder Benutzer, der sich angemeldet hat, seinen persönlichen Status ändern. Beispiele hierfür sind „Nicht Verfügbar“, „Beschäftigt“ oder „Abwesend“. Ist er nicht an dem Server angemeldet, erscheint er den Benutzern, die ihn in seiner Kontaktliste eingetragen haben, als „Offline“.

1.2. Konferenzsysteme

Konferenzsysteme werden umgangssprachlich auch als *Chat*-Systeme bezeichnet und dienen primär der parallelen Kommunikation zwischen mehreren Benutzern. Dabei gruppieren sich einige Benutzer, um dann unter einem gemeinsamen Identifikator angesprochen zu werden. Gruppierungen können auf Grund von Interessenschwerpunkten oder geographischen Eigenschaften entstehen.

Ein solcher Identifikator ist ein **Channel**, den mehrere Benutzer betreten können. Versendet ein Mitglied dieses Channels eine Nachricht an diesen, kann die Nachricht von allen anderen Mitgliedern gelesen werden. Dies ermöglicht eine einfache und effiziente Kommunikation zwischen mehreren Benutzern. Zusätzlich ist bei den meisten Kommunikationssystemen eine direkte Kommunikation zwischen einzelnen Benutzern möglich.

Beim Einsatz von öffentlichen Channels, in die alle interessierten Benutzer eintreten können, ist es für jeden einzelnen Benutzer sehr leicht, neue soziale Kontakte zu knüpfen. Die Hemmschwelle, in einer anonymen Gruppe zu kommunizieren, ist geringer, als einen einzelnen, fremden Benutzer direkt anzusprechen.

Durch die Vielzahl von Kontakten, die geknüpft werden können, leiden viele Benutzer von privat eingesetzten Konferenzsysteme unter einer „Sucht“. Einige Benutzer können sich hier Respekt und Anerkennung verschaffen, die sie im realen Leben nicht bekommen.

Um die Kommunikation innerhalb der hier vorgestellten Konferenzsysteme zu ermöglichen, müssen die Benutzer dauerhaft an einem Server angemeldet sein. Hierfür muss die Netzverbindung über die gesamte Zeit aufgebaut bleiben. Die Struktur der **Instanz** eines Konferenzsystems ist bis auf die fehlende Zustandsverwaltung analog zu der Instanz eines IM-Systems.

2. Zielgruppen von Instant Messaging

Instant Messaging lässt sich sehr vielseitig einsetzen. Es ermöglicht eine Kommunikation zwischen zwei oder mehreren Endpunkten. Diese Endpunkte müssen nicht immer Menschen sein. Es ist es auch möglich, dass bei bestimmten Systemereignissen, wie Ausfällen, der zuständige Administrator über ein *Instant Messaging*-System informiert wird.

Zusätzlich ist zu unterscheiden, ob das *Instant Messaging*-System nur für den Heimgebrauch oder innerhalb einer Organisation eingesetzt werden soll.

2.1. Privater Einsatz

Ursprünglich wurde *Instant Messaging* für den privaten Einsatz konzipiert. Viele Anbieter von Klientensoftware versuchen daher, die Software so komfortabel wie möglich zu gestalten.

Die Verbreitung eines *Instant Messaging*-Systems geschieht oft über Weiterempfehlungen. Zu Beginn probiert ein einzelner Benutzer ein System aus, für das er wiederum oft sein privates oder berufliches Umfeld begeistern kann. Diese Benutzer überzeugen wiederum neue Benutzer.

Für den Heimgebrauch von *Instant Messaging* spielt der Schutz nur eine untergeordnete Rolle. Die Benutzer möchten mit ihren Familien oder Freunden nur plaudern, in der Regel werden wichtigere Gespräche persönlich über Telefon oder per E-Mail geführt.

Die Risikoabschätzung für *Instant Messaging* entspricht ziemlich genau der von E-Mail. Beide Systeme sind anfällig für *Phishing* (siehe Anhang B.10) oder für die Übermittlung von bösartigen Code. Außerdem besteht auch die Gefahr des Abhörens, eine Übermittlung sensibler Informationen ist bei beiden Systemen zu vermeiden. Sind dem Benutzer diese Gefahren bewusst, kann er *Instant Messaging* analog zu E-Mail verwenden.

2.2. Einsatz in Unternehmen

Auch in Organisationen bieten *Instant Messaging*-Systeme viele Vorteile. Die folgenden wurden von *freiheit.com technologies* in einer Studie[3] zusammengefasst.

2.2.1. Einsatz eines zentralen Instant Messaging-Systems

Um mit allen Kontakten über *Instant Messaging* kommunizieren zu können, werden oft mehrere unabhängige IM-Systeme benötigt. Da jedes dieser Systeme eine eigenständige Verbindung aufbaut, wird die Kontrolle und Steuerung dieser erschwert.

Dieses Problem kann durch den Einsatz eines lokalen IM-Systems, das eine Kommunikation über viele andere Protokolle ermöglicht und über das die gesamte Kommunikation läuft, gelöst werden. Das *OpenSource*-System *Jabber* erfüllt diese Anforderungen.

Die Vorteile sind offensichtlich: da die gesamte Kommunikation an einer Stelle zentral zusammenfließt, müssen die Klienten keine eigenständige Verbindung zu öffentlichen IM-Servern aufbauen. Hinzu kommt, dass alle empfangenen und gesendeten Nachrichten protokolliert werden können. Dies ist vorteilhaft, um unerwünschte Kommunikation (zum Beispiel den Austausch von Geschäftsgeheimnissen) zu erkennen und darauf entsprechend zu reagieren.

2.2.2. Instant Messaging als Kommunikationswerkzeug

Instant Messaging ermöglicht eine zeitnahe und direkte Kommunikation, wovon besonders Organisationen profitieren können. Die Benutzer können direkt erkennen, welchen Status der Gesprächspartner zur Zeit besitzt. Im Gegensatz zum Telefon, bei dem erst sehr lange nach einem Partner, der gerade Zeit hat, gesucht werden muss, erfolgt dies bei *Instant Messaging* direkt auf den ersten Blick.

2.2.3. Instant Messaging als Kooperationswerkzeug

Der Einsatz von *Instant Messaging* ist nicht nur auf die textbasierte Kommunikation beschränkt. Vielmehr kann es tief in die Geschäftsprozesse der Organisation integriert werden. Beispielsweise können Dateien wie Quellcode oder Dokumentation direkt ausgetauscht werden. Diese Aufgabe, die vorher von intern versendeten E-Mails erfüllt wurde, kann *Instant Messaging* komplett übernehmen.

Durch den Einsatz von Audio- oder Videokonferenzen in Verbindung mit einer *Whiteboard*-Funktionalität kann *Instant Messaging* viele bestehende Aufgabenfelder vereinen.

3. IM-Protokollunabhängige Angriffe

Zahlreiche Ansätze ermöglichen es einem Angreifer, die Grundprinzipien der IT-Sicherheit (Verfügbarkeit, Vertraulichkeit und Integrität) zu verletzen.

Der Schwerpunkt dieser Arbeit liegt auf einer Sicherheitsanalyse der übertragenen Informationen auf der Verarbeitungsschicht. Ein Angreifer kann aber auch die untergeordneten Schichten angreifen. In der Literatur sind viele Angriffe auf der Transport-, Netz-, Sicherungs- sowie Bitübertragungsschicht zu finden.

Da der Umfang dieses Themenbereichs eine eigenständige Studie rechtfertigt, werden diese Angriffe nur schematisch vorgestellt. Kurze Beispiele verdeutlichen die Gefahr, die von diesen ausgehen kann.

3.1. Angriffe auf die Verfügbarkeit

Das Ziel von Angriffen auf die Verfügbarkeit ist die Arbeitsunfähigkeit des Opfers. Diese Art von Angriffen wird auch als *Denial of Service* (DoS) bezeichnet. Das Opfer eines DoS-Angriffs kann ein einzelner Netzdienst, ein Rechner oder ein gesamtes Netz sein.

Im Gegensatz zu anderen Techniken kann in der Regel bei einem DoS-Angriff der Angreifer keine vertraulichen Informationen einsehen. Ein DoS-Angriff kann aber als Grundlage für weitere Angriffe dienen, die die Vertraulichkeit und Integrität verletzen sollen.

Obwohl physikalischer Vandalismus ebenfalls als DoS-Angriff angesehen werden kann, sollen an dieser Stelle nur netzbasierte DoS-Angriffe betrachtet werden.

Ein bei Angreifern sehr beliebter, netzbasierter Angriff auf die Verfügbarkeit besteht darin, eine Überlastung des Systems oder des Dienstes zu erzeugen. In diesem Fall muss in der Regel die Anbindung des Angreifers eine höhere Bandbreite aufweisen als die des Opfers. Um dies zu erreichen, wird in der Praxis der Angriff von mehreren Punkten aus durchgeführt. Hier müssen die Angreifer in der Summe eine bessere Anbindung an das Netz besitzen. Eine von mehreren Angriffspunkten aus durchgeführte DoS-Attacke wird in der Literatur als *Distributed Denial of Service* bezeichnet.

Ein Beispiel für einen *Denial of Service*-Angriff durch Überlastung ist *Flooding*. Diese Art des Angriffs wird im Anhang B.4 vorgestellt. Vertiefende Informationen sind unter [5] zu finden. Eine weitere Möglichkeit, einen TCP-Datenstrom zwischen zwei Systemen zu unterbrechen, ist das Senden eines Datagramms mit gesetztem *Reset*-Flag an mindestens einen der beiden Kommunikationspartner.

Ein weiterer Ansatz, neben der Überlastung der Netzanbindung, besteht im Ausnutzen von Schwachstellen der Applikationen. Bietet ein System einen Netzdienst an, bei dem Schwachstellen bekannt sind, kann dies ein Angreifer von einem anderen System über das Netz ausnutzen. Auch hier kann das Ergebnis von einem ausgefallenen Dienst bis zu einem ausgefallenen Netz reichen.

Um ein komplettes Netz außer Kraft zu setzen, sind Angriffe auf zentrale Dienste, von denen andere Systeme abhängen, ein sehr beliebtes Ziel von Angreifern. Gelingt es einem Angreifer, den einzigen File-, DNS- oder DHCP-Server im Zielnetz außer Dienst zu stellen, würde die Verfügbarkeit aller Systeme nicht mehr garantiert sein. Weniger kritisch, aber dennoch effizient, ist ein Angriff auf den einzigen Proxy-, Mail-, IM- oder Konferenzserver. Obwohl die Benutzer im angeschlossenen Netz ihre Systeme weiter nutzen könnten, sind sie von der Außenwelt abgeschnitten.

Ein Schutz vor DoS-Angriffen ist nur sehr schwer umzusetzen. In der Regel muss ein bestimmter Dienst durch mehrere Personen genutzt werden können. Eine Einschränkung des Kreises der autorisierten Personen ist zwar durch Paketfilterregeln möglich, aber IP-Adressen können leicht gefälscht werden.

Ein sehr effizienter Schutz vor der Ausnutzung von Softwareschwachstellen ist die regelmäßige Aktualisierung der Software.

3.2. Angriffe auf die Vertraulichkeit

In sehr vielen Fällen interessieren sich die Angreifer für den Inhalt der übertragenen Informationen. Dies kann aus Neugierde oder anderen Interessen wie Spionage sein.

Angriffe auf die Vertraulichkeit können auch weitere Angriffe einleiten. So kann ein Angreifer aus einer Netzkommunikation Passwörter erspähen, die er für weitere Zwecke einsetzen kann. Angriffe auf die Vertraulichkeit von Netzverbindungen werden auch als *Sniffen* (siehe Anhang B.16) bezeichnet. Hier liest der Angreifer den gesamten oder ausgewählten Netzverkehr mit, um diesen dann auszuwerten.

Führt die Route der versendeten Datagramme direkt an dem System des Angreifers vorbei, ist das Sniffen auch für Angreifer mit sehr geringen Vorkenntnissen möglich. Als Beispiel ist das graphische Programm *ethereal*[1] zu nennen, mit dem direkt Angriffe auf die Vertraulichkeit vorgenommen werden können. Verläuft die Route nicht direkt über das System des Angreifers, befindet er sich aber in einem Netzsegment eines beteiligten Systems, kann er sehr oft durch *Spoofing* (siehe Anhang B.17) den Datenstrom umleiten.

Grundkenntnisse über TCP/IP werden hierbei höchstens für die Auswahl der auszuwertenden Verbindungen benötigt. Dadurch kann sich der Angreifer direkt auf die gewünschten Verbindungen konzentrieren.

Einen Schutz vor *Sniffen* bietet nur eine Verschlüsselung der übertragenen Informationen. Um diese Informationen auszuwerten, muss der Angreifer entweder Kenntnis über den verwendeten Schlüssel erlangen oder direkt das eingesetzte Kryptographieverfahren angreifen. Da in der Praxis hauptsächlich Verschlüsselungsverfahren eingesetzt werden, von denen angenommen wird, dass sie sicher sind, führt der Weg, den Schlüssel zu ermitteln, eher ans Ziel.

3.3. Angriffe auf die Integrität

Hier sollen nur solche Angriffe auf die Integrität betrachtet werden, die Informationen während der Übertragung zwischen dem Sender und dem Empfänger verändern können. Diese Veränderungen reichen vom Einfügen sinnloser Informationen bis zur gezielten Manipulation. Durch die falschen Informationen kann der Empfänger zu Handlungen bewegt werden, die er ohne diese Informationen nicht ausführen würde.

Eine gebräuchliche Variante von Angriffen auf die Integrität der übertragenen Informationen sind *Man-In-The-Middle-Attacks*. Im Normalfall baut der Sender eine Verbindung zu dem Empfänger auf. Bei einer *Man-In-The-Middle-Attack* versucht der Angreifer, sich direkt zwischen die beiden Kommunikationspartner zu schalten. Indem sich der Angreifer als der Empfänger ausgibt, wird die Verbindung nicht zu dem Empfänger, sondern direkt zu ihm aufgebaut. Im nächsten Schritt gibt sich der Angreifer als der Sender aus und baut wiederum eine Verbindung zu dem Empfänger auf. Auf alle Datagramme, die von dem Sender zu den Empfänger geschickt werden, hat der Angreifer direkten Zugriff. Es steht ihm frei, die übertragenen Informationen nur einzusehen oder direkt zu verändern. Ein Beispiel für die Umsetzung einer *Man-In-The-Middle-Attack* ist im Kapitel 6.3 zu finden.

Mit Hilfe einer *Man-In-The-Middle-Attack* lässt sich auch *TCP Hijacking* (siehe Anhang B.18) betreiben. Hier wird beispielsweise die SSH- oder Telnetverbindung, die von dem Sender

[1] **Verweis auf Applikation:** http://www.ethereal.com

aufgebaut wurde, von dem Angreifer übernommen.

Auch die Verwendung einer Verschlüsselung bietet keinen Schutz vor einer Übernahme, da die Schlüsselaushandlung nicht mehr zwischen Sender und Empfänger stattfindet, sondern jeweils zwischen Sender und Angreifer sowie zwischen Angreifer und Empfänger.

Theoretisch sollten die Gesprächspartner die Identitäten auf Grund des *Fingerprints* des öffentlichen Schlüssels überprüfen, jedoch wird dies in der Praxis oft nicht ausreichend umgesetzt.

Um diesen *Fingerprint* vergleichen zu können, muss im Vorfeld eine Vergleichszeichenkette übertragen werden. Gelingt es dem Angreifer, diese ebenfalls abzufangen und durch seine eigene auszutauschen, kann er weiterhin eine *Man-In-The-Middle-Attack* durchführen.

Um das Opfer zu täuschen, muss aber nicht notwendigerweise der *Fingerprint* abgefangen werden. Viele Benutzer schenken dem Vergleich der *Fingerprints* wenig Beachtung und überprüfen diesen überhaupt nicht oder nur teilweise. Daher wird ein Angreifer mit der Erstellung eines Schlüssels, dessen *Fingerprint* dem des Opfers ähnlich ist, mit hoher Wahrscheinlichkeit ebenfalls sein Ziel erreichen. Diese Prüfsumme, die der des Schlüssels ähnlich sieht, wird als *Fuzzy Fingerprint* bezeichnet.

3.4. Lokale Angriffe auf den Klienten

Aus der Sicht eines Angreifers besteht eine der wenigen Möglichkeiten, eine kryptographisch geschützte Verbindung zu kompromittieren, darin, den Angriff direkt an einen Endpunkt vorzunehmen.

Verantwortungsvolle Serverbetreiber stecken in der Regel sehr viel Energie in die Absicherung ihrer Systeme, daher ist der Klient ein leichteres Ziel für einen Angriff. Da der Mensch grundsätzlich das schwächste Glied in der IT-Sicherheitskette ist, kann der Angreifer seine Tätigkeiten auf ihn konzentrieren. Einige Angriffsmöglichkeiten auf diese Schwachstelle werden in diesem Abschnitt vorgestellt.

Besitzt der Angreifer Administratorenrechte auf dem Computersystem, ist es dem Benutzer unmöglich, seine Kommunikation vor dem Angreifer zu schützen. Damit sind dem Angreifer auch effiziente Angriffe auf die Verfügbarkeit des Systems möglich. Dazu kann er beispielsweise wichtige (System-)Dateien löschen.

Gelingt es dem Angreifer, den Klienten gegen eine präparierte Version auszutauschen, kann er die geschützten Nachrichten entweder vor dem Verschlüsseln oder nach dem Entschlüsseln einsehen. Für einen Angriff auf den Klienten muss der Angreifer nicht die vollen Administratorenrechte besitzen. Hier reicht in der Regel eine entsprechende Schwachstelle in der benutzen Software oder Schreibrechte.

Alle Systeme, die auf einen kryptographischen Schutz setzen, benötigen oft eine explizite Authentisierung. Dies kann zum Beispiel eine *Passphrase* (siehe Anhang B.8) zu einem privaten Schlüssel sein. Das größte Problem dabei ist die Übergabe der *Passphrase* an die Kommunikationsinstanz. Eine Möglichkeit besteht darin, diese in einer Datei abzulegen. Hat der Angreifer hierauf Zugriff, ist dieser Schutz nutzlos.

Eine zweite Möglichkeit, die in der Praxis viele Anhänger findet, ist das Merken der *Passphrase*. Das Problem besteht darin, dass die *Passphrase* an die entsprechende Instanz übergeben werden muss. Für einen erfahrenen Angreifer sollte es keine Schwierigkeit sein, die *Passphrase* auf diesem Weg abzufangen. Sein Waffenarsenal reicht von *Keyloggern* (sowohl Hardware als auch

Software) über Überwachungskameras, die die Schnittstelle zwischen Benutzer und Hardware überwachen, bis hin zu kompromittierter Software.

Einen Schutz dagegen können Einmalpasswörter oder externe, abgeschlossene Systeme bieten, die eine Signatur nach einer erfolgreichen Authentisierung übertragen. Letzteres ist vom *Homebanking* bekannt, bei dem die Bankkarte und eine Zahlenkombination an das Gerät übergeben werden müssen.

Weiterhin ist es problematisch, dass der Benutzer bei jeder gesendeten oder empfangenen Nachricht die *Passphrase* erneut eingeben muss. Daher wird diese in der Regel im Hauptspeicher für eine mehrmalige Verwendung abgelegt. Besitzt der Angreifer Zugriff auf den Hauptspeicher, kann er auf die abgelegte *Passphrase* zugreifen.

Wichtige Nachrichten sollten für eine spätere Verwendung protokolliert werden. Viele Klienten stellen hierfür Automatismen zur Verfügung, die dies dem Benutzer erleichtert. Alle gesendeten und empfangenen Informationen werden hierbei auf der Festplatte abgelegt. Wird diese Protokollierung genutzt und ein Unberechtigter hat Zugriff auf die Dateien, in die die Informationen geschrieben werden, ist jede noch so starke Verschlüsselung der Verbindung wirkungslos.

4. Probleme durch Kommunikationssysteme

Sowohl beim Einsatz von *Instant Messaging* als auch bei Konferenzsystemen können Probleme auftreten. Leider sind besonders die *Instant Messaging*-Systeme sicherheitstechnisch nicht ausgereift. Dies ist vor allem durch den großen Funktionsumfang bedingt. Erschwerend kommt hinzu, dass die IM-Klienten sehr oft direkt nach dem Systemstart automatisch aktiviert werden. Unerfahrene Benutzer können dieses Verhalten nicht unterbinden.

Im Folgenden werden mögliche Gefährdungen im Zusammenhang mit *Instant Messaging* auf Grundlage eigener Beobachtungen skizziert. Als Strukturierungsvorlage dienten die Gefährdungskataloge des Grundschutzhandbuchs des Bundesamts für Sicherheit in der Informationstechnik.

4.1. Organisatorische Mängel

4.1.1. Regelungen für den dienstlichen und privaten Einsatz in Unternehmen

Sehr oft treffen die Regelungen für die Nutzung von Computersystemen in Unternehmen keine Aussagen über die Verwendung von *Instant Messaging*. Kann der Benutzer die zu dem Kommunikationssystem gehörende Software installieren oder bietet der Betreiber des Kommunikationssystems einen webbasierten Zugang an, ist die Nutzung problemlos möglich. Untersuchungen des Markforschungsunternehmens *Meta Group*[1] haben gezeigt, dass *Instant Messaging* in Unternehmen sehr oft für private Gespräche genutzt wird. Soll der Benutzer aber auch mit externen Gesprächspartner kommunizieren können, so muss eine Trennung von privaten und geschäftlichen *Instant Messaging*-Nachrichten umgesetzt werden. Dies ist jedoch technisch nur sehr schwer realisierbar.

Da die Nutzung von *Instant Messaging* mit vielen Problemen behaftet ist, sollte daher ein Abschnitt über *Instant Messaging* in der *Policy* des Unternehmens aufgenommen werden. Für den Einsatz in einem Unternehmensnetz kann zwischen vier Fällen unterschieden werden:

1. Der private Einsatz von *Instant Messaging* wird gestattet und ein geschäftlicher Einsatz ist nicht vorgesehen.
2. Die private Nutzung wird gestattet und *Instant Messaging* wird zusätzlich für den geschäftlichen Einsatz verwendet.
3. Die private Nutzung wird nicht gestattet aber für den geschäftlichen Einsatz wird *Instant Messaging* verwendet.
4. *Instant Messaging* wird im Unternehmen nicht eingesetzt und eine private Nutzung ist verboten.

4.1.2. Protokollierung

Viele Institutionen schreiben eine Aufzeichnung aller ausgehenden Nachrichten vor. Diese Protokollierung reicht von der Information, wer eine Nachricht an wen gesendet hat, bis zu den kompletten Inhalten der Nachricht. Durch eine zentrale Protokollierung von E-Mail auf einem lokalen Server ist dies sehr leicht zu implementieren.

Anders sieht es aber bei der Verwendung eines öffentlichen *Instant Messaging*-Server aus. Da die Klienteninstanz direkt mit diesem Server in Verbindung treten, könnte nur dieser die Informationen protokollieren. Mit ziemlicher Sicherheit sind aber die Betreiber dieser Systeme nicht bereit, diese Informationen weiterzugeben.

Daher müssen andere Wege gefunden werden, dieses Ziel zu erreichen. Hierfür kann die Protokollierung direkt auf dem Klienten durchgeführt werden. Eine weniger elegante Möglichkeit ist die Protokollierung des kompletten Datagrammstroms mit einer anschließenden Filterung der gewünschten Informationen.

Im Weiteren stehen hierfür auch kommerzielle Lösungen zur Verfügung[2].

4.1.3. Integration des Klienten in die Anwendungslandschaft

In vielen Organisationen wird der Benutzer zentral autorisiert, um sich am Rechner anzumelden, seine Benutzerverzeichnisse von einem Server einzubinden und um E-Mail nutzen zu können. Ein Vorteil davon ist die Änderung des Passworts an einer zentralen Stelle.

Durch den fehlenden Protokollstandard sind bisher keine Lösungen bekannt, die *Instant Messaging* in diese Architektur integrieren.

4.1.4. Schutz der Privatsphäre

Die direkte und sehr einfache Möglichkeit, mit einem anderen Benutzer in Kontakt zu treten, ist auch mit Nachteilen verbunden. Beispiele hierfür sind das unverlangten Zusenden von Werbung (*SpIM*, entstanden aus *Spam* und IM) oder die ständige Belästigung und Bedrohung durch andere Personen (*Stalking*). Des Weiteren kann durch die Auswertung des Status ein detailliertes Benutzerprofil erstellt werden. Ein Angreifer, der eine Analyse über den Benutzer anfertigt, kann somit leicht in Erfahrung bringen, wann der Benutzer arbeitet, wann er zu Hause ist und

[2] **Vertiefung:** Akonix L7 Enterprise http://www.akonix.com/products/l7enterprise.asp

wann er Zeit hat. IM erleichtert diese Analyse, da der Benutzer eindeutig an seinem Identifikator erkannt werden kann, auch wenn sich seine IP-Adresse ändert.

Viele *Instant Messaging*-Systeme bieten Mechanismen, die die Kontaktaufnahme und Statuseinsicht durch unbekannte oder unerwünschte Personen verhindern sollen. Dazu unterscheiden *Instant Messaging* drei Benutzergruppen: unbekannte, erwünschte und unerwünschte. Für einen sinnvollen Einsatz von *Instant Messaging* muss die Kommunikation mit erwünschten Kontakten garantiert werden.

Für die beiden anderen Benutzergruppen ist je nach *Instant Messaging*-System zu unterscheiden, ob der Status des Benutzers von der entsprechenden Gruppe eingesehen werden kann oder nicht. Idealerweise sollte dies nicht möglich sein.

Weiterhin ist es wichtig, dass unerwünschte Kontakte dem Benutzer keine Nachrichten senden dürfen. Hierfür wird der eindeutige Bezeichner des Kontakts in einer Liste eingetragen. Diese wird in der Regel auf den Server übertragen, um die Zustellung der Nachrichten zu unterbinden.

Dieser Schutz kann aber sehr leicht durch einen Wechsel des Identifikators umgangen werden. Daher ist das Verhalten gegenüber bisher noch nicht bekannten Kontakten entscheidend. Um den Schutz der Privatsphäre zu garantieren, ist die Option, dass alle unbekannten Kontakte dem Benutzer Nachrichten schicken können, nicht sehr ideal.

Dennoch muss eine Möglichkeit existieren, bei der ein neuer Kontakt mit dem Benutzer in Verbindung treten kann. Dies kann durch einen speziellen Dialog, der die Erlaubnis des Benutzers einholt, realisiert werden.

Erschwerend kommt hinzu, dass bei einigen *Instant Messaging*-Systemen diese Erlaubnis nur schwer wieder zurückgezogen werden kann. Es besteht zwar die Möglichkeit, die Nachrichten einiger Kontakte zu ignorieren, aber die Einsicht in den Benutzerstatus ist durch den unerwünschten Kontakt weiterhin möglich.

Dieses Problem kann durch eine zusätzliche Liste, auf der der Server alle Kontakte verwaltet, die den aktuellen Benutzer auf ihrer Kontaktliste führen, gelöst werden. Existiert eine Differenz zwischen der eigentlichen Kontaktliste des Benutzers und dieser Liste, kann der Benutzer hierüber informiert werden und kann bei Bedarf den anderen Kontakt für eine Kommunikation freischalten.

4.1.5. Datenschutz

Neben dem Benutzernamen und dem Status können andere Anwender weitere Informationen über den Benutzer einsehen. Für einen privaten Einsatz können bei zahlreichen *Instant Messaging*-Systemen auch persönliche Informationen eingetragen werden. Beispiele hierfür sind Geburtsdatum, Freizeitinteressen, Geschlecht und Adressen. Diese Informationen können für eine Suche nach anderen Kontakten mit entsprechenden Eigenschaften genutzt werden. Sie können aber auch für die unerwünschte Zusendung von Werbung missbraucht werden. Wird eine Kontaktaufnahme durch neue Anwender nach diesen Aspekten nicht gewünscht, sollten so wenig Informationen wie möglich dem *Instant Messaging*-System übermittelt werden.

Neben diesen Informationen, die oft nicht zwingend angegeben werden müssen, sind aber noch weitere Eigenschaften für Angreifer von Interesse. Die IP-Adresse, die einen *Denial of Service*-Angriff ermöglicht oder die auf den Standort des Benutzers schließen lässt, ist hierfür ein Beispiel.

Für die Abfrage dieser Informationen muss die Gegenseite in den meisten Fällen nicht bei dem *Instant Messaging*-System registriert sein, da viele Systeme die Abfrage von Benutzerinformationen über eine Web-Schnittstelle anbieten.

4.2. Menschliche Fehlhandlungen

4.2.1. Probleme auf der Seite des Benutzers

Falls dies in der Policy des Unternehmens nicht direkt verboten wird, installieren viele Benutzer *Instant Messaging*-Software auf ihren Arbeitsplatzrechnern. Da sie dabei keine Risiken sehen, ist dies für die Benutzer eine komfortable Lösung, um private Kontakte zu pflegen. Dabei sind ihnen die netztechnischen Gefahren nicht bewusst, die der Einsatz von *Instant Messaging* mit sich bringt.

So kann der Benutzer durch andere Personen belästigt werden. Diese können ihm Werbung zuschicken oder unseriöse Angebote unterbreiten, die ihn von der Arbeit abhalten. Durch die direkte Kommunikation kann ein Angreifer zusätzlich den Versuch unternehmen, bösartigen Code auf dem Computer des Benutzers ausführen zu lassen.

Da auch Dateien zwischen den Benutzern ausgetauscht werden können, ohne dass dies der Netzbetreiber verhindern kann, bringt dies noch weitere juristische Probleme mit sich. Bei einem Tausch von urheberrechtlichen geschützten Material, der von einem Firmennetz ausgeht, wird zunächst dessen Betreiber belangt.

4.3. Technisches Versagen

4.3.1. Nutzung von öffentlichen Servern

Bei vielen Kommunikationssystemen ist die Nutzung eines öffentlichen Servers für den Betrieb des Systems notwendig. Bei diesen Systemen muss sich der Benutzer anmelden, kann dort seine Statusinformationen ändern und über diesen Server mit anderen Benutzern kommunizieren.

Der Einsatz eines öffentlichen Kommunikationssystems kann sehr sinnvoll sein. Um mit Geschäftspartnern oder Kunden, deren Klienteninstanzen sich nicht direkt im lokalen Netz befinden, einen Dialog aufbauen zu können, müssen diese Zugang zu einem Server erlangen, der dieses System anbietet.

Die Benutzer dieser öffentlichen Server besitzen aber keinen Einfluss auf die Administration dieser Systeme. Somit ist es möglich, dass die zuständigen Betreiber die Server nicht ausreichend vor Angriffen schützen.

Der öffentliche Server kann aber auch bewusst von den Betreibern modifiziert werden. Ein Beispiel hierfür ist die Protokollierung aller verarbeiteten Nachrichten, um diese im schlimmsten Fall weiterzugeben.

Es ist auch nicht auszuschließen, dass bestimmte Informationen zwischen dem Sender und dem Empfänger bewusst durch den Serverbetreiber manipuliert werden. Auch dies ist für die Übermittlung von sensiblen Geschäftsgeheimnissen fatal.

Zusätzlich sind den Benutzern eines solchen Systems bei einem Ausfall des Servers die Hände gebunden. Sie haben keinen Einfluss auf die Verfügbarkeit und bei einem finanziellen Schaden werden sie auch nicht auf eine Entschädigung durch den Betreiber hoffen können.

4.3.2. Schutz der übertragenen Informationen

Da viele *Instant Messaging*-Systeme auf einen Schutz der übertragenen Informationen verzichten, können diese auf dem Weg zwischen Server und Klienten abgehört werden. Dabei sind sowohl Vertraulichkeit als auch Integrität und Verfügbarkeit der Informationen gefährdet.

Wenige Systeme setzen auf eine Verschlüsselung zwischen den Klienten und dem Server. Bei einem kompromittierten Server ist aber auch diese nutzlos. Daher sollten die übertragenen Informationen direkt bei dem Sender verschlüsselt und beim Empfänger entschlüsselt werden. Bisher hat sich aber noch kein Standard hierfür durchgesetzt. In der Regel kann eine Nachricht nur mit einer Klienteninstanz entschlüsselt werden, deren Verschlüsselung kompatibel zu der der Klienteninstanz des Senders ist.

Kann ein *Instant Messaging*-Server in einem geschützten Netz, in dem die sensiblen Informationen ausschließlich übertragen werden, betrieben werden, spielt dieses Problem nur eine untergeordnete Rolle. Außenstehenden kann der sichere Zugang zu der Kommunikation durch ein *Virtuelles Privates Netz* (VPN) oder einen *Tunnel* ermöglicht werden.

4.3.3. Verfügbarkeit der Serversoftware

Um Geschäftsgeheimnisse zu schützen, die zwischen zwei Klienten in einem LAN ausgetauscht werden, sollte darauf geachtet werden, dass diese Informationen das geschützte Netz nicht verlassen. Hierfür existieren zwei Lösungsansätze: die Nachrichten werden direkt zwischen den Klienten ausgetauscht (*Peer-To-Peer*) oder die Organisation betreibt einen lokalen Server. Oft ist der Einsatz eines selbst administrierten Servers die elegantere Lösung.

Jedoch sind solche Server vieler proprietäre *Instant Messaging*-Systeme nicht verfügbar. Sie werden zwar von dem Betreiber des Systems eingesetzt, aber auf fremden Computersystemen dürfen sie nicht installiert werden. Eine Alternative hierzu stellt die *OpenSource*-Software *Jabber* dar, die zusätzlich als Gateway zu den proprietären IM-Systemen genutzt werden kann.

4.3.4. Einsatz von Paketfiltern

Sollen Paketfilter eingesetzt werden, um eine Nutzung von *Instant Messaging* zu verhindern, so ist dies mit einigen Problemen behaftet. Einige Systeme bieten durch die freie Wahl des Serverports die Möglichkeit, an bestehende Paketfilterregeln angepasst zu werden.

Zusätzlich bieten viele Betreiber webbasierte Systeme an, die eine Kommunikation über eine HTTP-Verbindung ermöglichen. Diese Verbindungen kann ein Paketfilter nur sehr schwer von zulässigen unterscheiden.

Wird der Einsatz von *Instant Messaging*-Systemen von der Unternehmenspolicy gestattet, bereitet dies ebenfalls viele Probleme. Soll die komplette Funktionalität zur Verfügung stehen, kann dies für einen Angriff ausgenutzt werden. Für eine *Peer-To-Peer*-Verbindung muss eine TCP-Verbindung zwischen den Gesprächspartnern aufgebaut werden. Dieser direkter Zugriff ermöglicht eine Vielzahl von Angriffen.

Neben den Mechanismen, die in jedem Kapitel zu den einzelnen *Instant Messaging*- Systemen vorgestellt werden, gibt es oft auch kommerzielle Lösungen. An dieser Stelle ist die Applikation

Akonix L7 Enterprise[3] zu nennen, die neben dem Blocken auch Protokollierungsmöglichkeiten bietet.

4.3.5. Vielfalt an Protokollen

Sehr viele Betreiber von *Instant Messaging*- und anderen Kommunikationssystemen stellen ihre Produkte dem Endbenutzer kostenlos zur Verfügung. Die Vorteile dieser vielfältigen Auswahl sind offensichtlich: Der Benutzer kann theoretisch frei entscheiden, welches Produkt er einsetzen will. Das von dem Benutzer verwendete Betriebssystem, seine Anforderungen an die Sicherheit des Produkts sowie seine Vorliebe für ein Lizenzsystem kann seine Auswahl beeinflussen.

Aber diese Auswahl, die ein Monopol eines Betreibers verhindern kann, hat entscheidende Nachteile. Am gravierendsten ist die Tatsache, dass die Protokolle untereinander nicht kompatibel sind. Ein Beispiel für dieses Problem ist ein Szenario, bei dem innerhalb eines Unternehmens ein spezielles *Instant Messaging*-System verwendet wird, aber die Kommunikation mit Geschäftspartnern die Nutzung anderer Systeme erfordert. Besonders deutlich wird dieses Problem, wenn jeder außenstehende Kontakt ein anderes System verwendet. Hier muss ein Benutzer, der mit allen Beteiligten kommunizieren will, jeweils einen entsprechenden Klienten aktiviert haben.

Auf der Seite des Benutzers ist dies nicht sehr problematisch, da der Markt einige Multiprotokollklienten bereitstellt. Aber für jedes System muss eine Verbindung von dem Klienten zu dem jeweiligen Server aufgebaut werden. Der Schutz jeder Verbindung vor einem Angreifer ist sehr schwer zu realisieren. Erschwerend kommt hinzu, dass diese Multiprotokollklienten sehr komplex sind und bei einer Modifikation eines einzelnen Protokolls der gesamte Klient aktualisiert werden muss.

Eleganter wurde diese Problematik bei E-Mail gelöst: durch einen Protokollstandard kann so gut wie jeder Klient verwendet werden, um eine Nachricht zu einem anderen Klienten zu senden. Ein IM-Protokoll, das von allen Betreibern unterstützt wird, ist aber zu Zeit noch ein Zukunftstraum. Erschwerend kommt hinzu, dass dies beispielsweise von AOL nicht erwünscht wird (Quelle: [2]).

4.4. Vorsätzliche Handlungen

4.4.1. Bösartiger Code

Die Vergangenheit hat gezeigt, dass sehr viel bösartiger Code oder Werbung über E-Mail verteilt wird. Zur Zeit der Erstellung dieser Diplomarbeit traten einige Fälle auf, in denen diese Verbreitung auch über *Instant Messaging* erfolgte.

Die Versender können sowohl die Kontakte des Benutzers als auch Angreifer sein, die sich als die Kontakte ausgeben. Besitzt die Klientensoftware hiergegen keine ausreichenden Schutzmechanismen, können auch unbekannte Personen ihren Code auf das System des Benutzers einschleusen.

Die durch diese Applikationen verursachten Schäden können bis zu dem kompletten Datenverlust oder Systemausfall führen. Aber auch Hintertüren im System, die einen weiteren Angriff

[3] **Vertiefung:** Akonix L7 Enterprise http://www.akonix.com/products/l7enterprise.asp

ermöglichen, sind ein Ziel von Angreifern.

Als weitere Möglichkeit können Programmfragmente übermittelt werden, die Schwachstellen in der verwendeten Klientensoftware ausnutzen. Im Falle von E-Mail existieren Mechanismen, die die E-Mails zentral filtern. Um die Benutzer zu schützen, die *Instant Messaging*-Systeme einsetzen, müsste ein entsprechender Proxy eingesetzt werden. Für die proprietären Systeme ist aber kein offizieller Proxy verfügbar.

4.4.2. Social Engineering

Die effizienteste Möglichkeit des Angreifers besteht im *Social Engineering*. Die Praxis hat gezeigt, dass es für einen Angreifer am einfachsten ist, den Benutzer direkt nach den gewünschten Informationen zu fragen. Die Möglichkeiten, die einem *Social Engineer* zur Verfügung stehen, sind sehr weit gefächert.

Da die Kontaktaufnahme über Kommunikationssysteme sehr einfach ist, ist *Social Engineering* jederzeit möglich. Ein Angreifer könnte sich ein Opfer auf Grund von diversen Eigenschaften aussuchen und angreifen. Er kann von dem Verhalten in einer Gruppendiskussion (*Chat*) auf die Gutgläubigkeit des Opfers schließen, was den Aufwand des Angreifers sehr verringert. Um seine Fähigkeiten zu trainieren, könnte sich der Angreifer einen misstrauischen Benutzer aussuchen. Möchte der Angreifer in ein bestimmtes Netz eindringen, kann er im voraus die Opfer anhand ihrer IP-Adresse auswählen. Zusammengefasst kann der Angreifer sich sein Opfer direkt ohne eine vorherige Kontaktaufnahme auswählen.

Anstelle hier auf weitere Beispiele einzugehen, wird auf „Die Kunst der Täuschung“ von Kevin D. Mitnick[4] verwiesen, das einen Einblick in diese Technik bietet.

5. Marktanteile

Das Hauptargument für die Auswahl eines *Instant Messaging*-Systems ist die Akzeptanz auf dem Markt. Jedes noch so ausgereifte *Instant Messaging*-System ist wertlos, wenn es nur von wenigen Benutzern verwendet wird. Dies wird besonders deutlich, wenn alle Benutzer, mit denen kommuniziert werden soll, ein anderes als das lokal eingesetzte *Instant Messaging*-System verwenden. Daher haben es besonders neue Implementationen schwer, auf dem Markt Fuß zu fassen. Verallgemeinert gesagt ist ein Kommunikationssystem umso attraktiver, desto mehr Benutzer es einsetzen.

Auch die Entwickler der *Instant Messenger* kennen diese Problematik. Daher werden auf den Homepages die Benutzerzahlen vermutlich „schöngerundet“. Hinzu kommt die Tatsache, dass bei AOL jeder, der einen Benutzernamen für die Einwahl ins Internet besitzt, diesen ebenfalls für *Instant Messaging* nutzen kann.

Bei Microsoft sieht es auch nicht anders aus. Jeder Benutzer, der ein *hotmail*-Konto besitzt, kann dieses Konto automatisch für *Instant Messaging* nutzen. Microsoft geht sogar einen Schritt weiter: bei der Installation von Windows-Betriebssystemen stößt der Benutzer auf die Möglichkeit, ein *hotmail*-Konto (und damit einen *Instant Messaging*-Benutzernamen) anzulegen. Gerade

[4] **Vertiefung:** „Die Kunst der Täuschung“ von Kevin D. Mitnick und William L. Simon, ISBN: 3826609999

unerfahrene Benutzer nehmen diese Möglichkeit gerne an. Daher kann Microsoft alle *hotmail*-Kunden in ihren Statistiken, unabhängig ob sie *Instant Messaging* nutzen, als *Instant Messenger*-Benutzer verzeichnen.

Neben dem Ausstatten aller Kunden mit einem *Instant Messaging*-Benutzernamen für die Statistik kommt ein weiteres Problem hinzu. In der Gesamtanzahl der Benutzer werden auch die Benutzer aufgenommen, die dieses System lange nicht mehr verwendet haben. Jedes Konto, das zum Ausprobieren angelegt und nicht entfernt wurde, vergrößert die Gesamtanzahl.

Eine Benutzerumfrage über den Einsatz von *Instant Messaging* ist selten repräsentativ. Benutzer, die das Internet schon sehr lange nutzen, würden vermutlich ICQ als ihren Favoriten angeben, da dieses *Instant Messaging*-System am längsten verfügbar ist. Benutzer, die Microsoft Windows als Betriebssystem einsetzen, werden öfters als Benutzer von *OpenSource*-Software zu dem *MSN Messenger* greifen.

Obwohl die von den Betreibern bereitgestellten Benutzerzahlen als auch die durch Umfragen gewonnen Ergebnisse nicht unbedingt die aktuelle Verbreitung widerspiegeln, wird dem Microsoft-Produkt die größte Verbreitung zugesprochen (Quelle: [4]).

6. Werkzeuge und Szenarien

Um die Kommunikationssysteme nach den Gesichtspunkten der IT-Sicherheit (Verfügbarkeit, Integrität und Vertraulichkeit) untersuchen zu können, muss im ersten Schritt deren Funktionsweise ermittelt werden. Dazu werden aus Interoperabilitätsgründen die Protokolle des Informationsaustauschs durch *Reverse Engineering* untersucht. Die dabei verwendeten Methoden sind der Schwerpunkt des ersten Abschnitts.

Die Analyse der Kommunikationssysteme wird zeigen, dass nicht alle benötigten Informationen durch einfaches *Sniffen* gewonnen werden können. Daher werden Szenarien gezeigt, wie diese Probleme umgangen werden können.

Schwachstellen der Kommunikationssysteme, die für Angriffe genutzt werden könnten, werden diese ebenfalls in diesem Kapitel vorgestellt.

6.1. Angriff auf ungeschützte Verbindungen

Dieser Abschnitt soll die Anfälligkeit der ungeschützten Kommunikationsprotokolle gegenüber Angriffen zeigen. Ziel dieses Fallbeispiels soll eine Kompromittierung der Vertraulichkeit, Integrität und Verfügbarkeit sein.

Um den Angriff interessanter zu gestalten, wird er innerhalb eines geswitchten Netzes vorgenommen, wie es in Abbildung 4 dargestellt ist. Dieses beinhaltet einen Klienten („Alice“), ein Gateway, der das öffentliche Netz von dem lokalen trennt („Bob“), einen Angreifer im lokalen Netz („Eve“) und eine entfernte Instanz des IM-Servers. Auf „Alice“ könnte eine der im Anhang C vorgestellten Instanzen die Verbindung zu der Serverinstanz aufbauen. Der Switch stellt dem Angreifer im Normalfall nur Datagramme zu, die für ihn bestimmt sind. Daher basiert der Angriff auf *ARP Spoofing* (siehe Anhang B.17). Als Werkzeug wird *hunt*[5] eingesetzt, das das Netz mit unbekannten MAC-Adressen überflutet. Versendet ein System eine Nachricht an solch eine

[5] **Verweis auf Applikation:** http://lin.fsid.cvut.cz/~kra

unbekannte MAC-Adresse, verwerfen alle bis auf das System des Angreifers diese Nachricht. Der Angreifer empfängt, bearbeitet und sendet die Nachricht an den vorgesehenen Empfänger weiter.

Mit Hilfe dieser Applikation kann die Verbindung über ein beliebiges Protokoll sehr elegant abgehört, verändert oder zurückgesetzt werden. Anstatt die folgenden Seiten mit dem Menü-Dialog von *hunt* zu füllen wird nur auf relevante Punkte näher eingegangen.

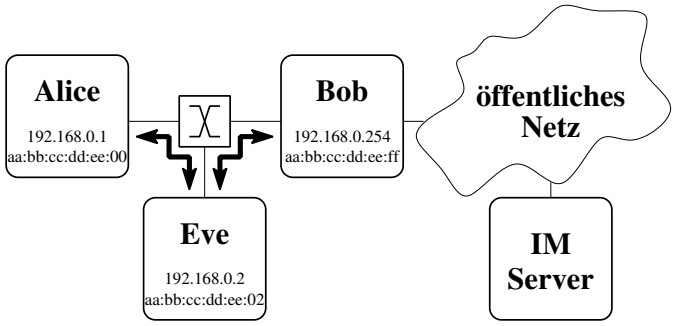

Abbildung 4: Geswitches Beispielnetz für Angriff

6.1.1. Vorbereitung

Die Situation vor dem Angriff ist die folgende:

```
alice$ arp 192.168.0.254
bob (192.168.0.254) at aa:bb:cc:dd:ee:ff on fxp0 [ethernet]
```

```
bob$ arp 192.168.0.1
alice (192.168.0.1) at aa:bb:cc:dd:ee:00 on fxp0 [ethernet]
```

Nun startet der Angreifer (in diesem Fall Eve) *hunt* und trägt die anzugreifenden Systeme (192.168.0.1 und 192.168.0.254) als Ziel ein. Dies kann sie über den Befehl „add host to host arp spoof“ im Abschnitt des „*arp spoof + arp relayer daemon*“ durchführen. Neben den IP-Adressen werden an dieser Stelle MAC-Adressen eingetragen, die nicht im Netz vorkommen dürfen. Die in der Voreinstellung eingetragenen MAC-Adressen *ea:1a:de:ad:be:01* für Alice und *ea:1a:de:ad:be:02* für Bob können problemlos übernommen werden. Anschließend muss im gleichen Abschnitt der *Relay*-Dienst aktiviert werden („*start relayer daemon*“), der die empfangenen Datagramme an den ursprünglichen Empfänger weiterleitet.

Das Ergebnis könnte mit den folgenden Befehlszeile auf den Rechner von Bob und Alice überprüft werden:

```
alice$ arp 192.168.0.254
bob (192.168.0.254) at ea:1a:de:ad:be:02 on fxp0 [ethernet]
```

```
bob$ arp 192.168.0.1
alice (192.168.0.1) at ea:1a:de:ad:be:01 on fxp0 [ethernet]
```

Ab diesem Zeitpunkt laufen alle Datagramme, die Bob und Alice austauschen, über den Rechner von Eve.

In der Voreinstellung zeigt *hunt* nur *telnet*-Verbindungen im Menüeintrag „*list/watch/reset connections*“ an. Unabhängig davon, ob der Angriff in einem geswitchten Netz über *ARP-Spoofing* stattfindet oder *hunt* auf einem der Zielrechner betrieben wird, bei dem nicht *gespooft* werden muss, sollte diese Voreinstellung in dem Abschnitt „*options*“ geändert werden. Wird eine Regel von den Rechner $0.0.0.0/0$ zu $0.0.0.0/0$ 5190 9898 1863 5222 6667 durch den Anweisung „*add conn policy entry*“ aufgenommen, kann später der Angriff auf diese Kommunikationssysteme erfolgen.

Nun können alle aktiven Verbindungen mit der Anweisung „list connections“ eingesehen werden.

6.1.2. Angriff auf die Vertraulichkeit durch *Sniffen*

Unabhängig davon, ob der Angriff innerhalb eines geswitchten Netzes mit *ARP Spoofing* oder direkt auf einem der beteiligten Systeme ausgeführt wurde, können alle aktiven Verbindungen mittels „*list connections*“ in *hunt* angezeigt werden.

Um Einsicht in den Datenstrom zu erhalten, muss mit der Anweisung „*watch connections*“ eine der verfügbaren Verbindungen ausgewählt werden. Eine weitere Möglichkeit bietet die Applikation *ethereal*[6]. Neben Mechanismen zur reinen Protokollierung können die empfangenen Informationen durch Filter übersichtlich visualisiert werden.

6.1.3. Angriff auf die Verfügbarkeit durch Reset-Datagramme

Mit der Anweisung „*reset connections*“ in *hunt* kann ein Angreifer eine bestehende Verbindung beenden. Dabei hat er die Wahl, ob ein TCP-Datagramm mit einem gesetzten *RST-Flag* an den Sender, an den Empfänger oder an beide gesendet wird.

6.1.4. Angriff auf die Integrität durch *Hijacking*

Der komplexeste Angriff mit *hunt* ist das Übernehmen einer TCP-Verbindung. Dabei kann ein Angreifer eigene Anweisungen, wie Textnachrichten oder die Authentisierung zu einer Kontaktlisteneintragung, in den Datenstrom schreiben.

hunt unterscheidet dabei zwei Methoden: *ARP Hijack* oder *Simple Hijack*. Bei *ARP Hijack* wird *ARP-Spoofing* für die Umsetzung verwendet. Befindet sich der Zielrechner außerhalb des geswitchten Netzes, kann diese Methode nicht eingesetzt werden: Es ist unmöglich, ARP-Datagramme zu diesem zu senden. Im Weiteren hat das *Spoofing* der entfernten Zieladresse auch keinen Einfluss auf lokale Rechner, da diese nur ARP-Adressen von direkt erreichbaren Systemen speichern. Daher muss hier die Anweisung *Simple Hijack* verwendet werden.

Bevor die Verbindung übernommen wird, besteht die Möglichkeit, den Datenstrom mitzulesen. Dies ist insbesondere für Protokolle interessant, die eigene *Session IDs* oder andere dynamische Felder führen. Beispielsweise ist bei dem *MSN Messenger Protocol* die *Transaction ID* wichtig. Daher empfiehlt es sich, erst einige Datagramme abzuhören, um bei der Generierung neuer Datagramme die nächste *Sequence Number* oder *Transaction ID* vorauszusagen. Hierfür muss die letzte empfangene Nummer mindestens um Eins inkrementiert werden.

[6] **Verweis auf Applikation:** `http://www.ethereal.com`

6.2. Nachweis der Abhängigkeit des *Authorization Cookies* von der IP-Adresse des Klienten

Die *Instant Messaging*-Systeme ICQ und AIM trennen die Authentisierung von der eigentlichen Anmeldung. Details zu der Anmeldung sind in der Protokollanalyse im Anhang A.1.2 zu finden. Im ersten Schritt baut der Klient eine Verbindung zum *Authentification Server* auf, von dem er als Nachweis seiner Identität ein *Authorization Cookie* und die Adresse des BOS-Servers erhält. Die Anmeldung wird nun am BOS-Server fortgesetzt.

Diese verteilte Anmeldung wirft die Frage auf, ob ein Angreifer den Informationsaustausch des Klienten zum *Authentification Server* abhören und sich mit den abgefangenen Informationen anmelden kann. Folgt dem eine *Denial of Service-Attack* gegenüber dem Klienten, kann sich der Angreifer ohne Kenntnis des Passworts direkt am BOS-Server anmelden.

Dieser Angriff setzt voraus, dass der *Authentification Server* das *Authorization Cookie* nicht an die IP-Adresse des Klienten bindet. Daher soll im Folgenden überprüft werden, ob die Authentisierung und Anmeldung von unterschiedlichen IP-Adressen vorgenommen werden kann. Abbildung 5 zeigt den Informationsaustausch, bei dem der authentisierte Klient die im *Close Connection*-Datagramm des *Authentification Server* empfangenen Informationen über eine TCP-Verbindung an den zweiten Klienten übermittelt. Beide Klienten dürfen nicht über den selben NAT-Gateway oder Proxy-Server ans Internet angeschlossen sein, da sonst die Anmeldung mit identischen IP-Adressen erfolgt.

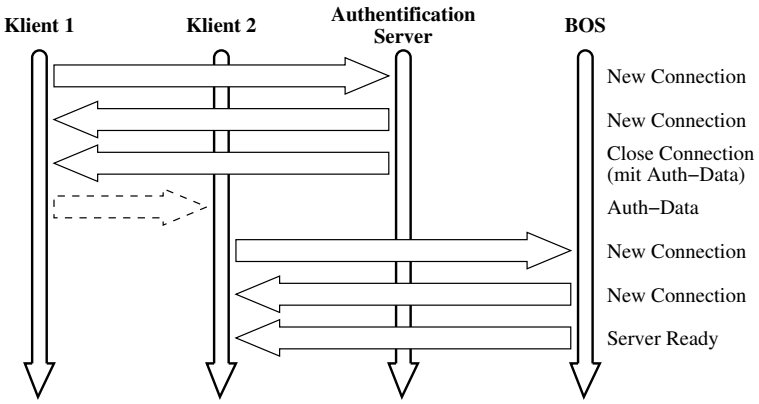

Abbildung 5: Verteilte Anmeldung von OSCAR

Der in Abbildung 5 gezeigte Informationsaustausch kann mit dem selbst entwickelten, unter Anhang C.1.1 zu findenden Klienten umgesetzt werden.

Der Einsatz des Klienten hat gezeigt, dass eine getrennte Anmeldung von zwei verschiedenen IP-Adressen nicht möglich ist. Im Gegensatz dazu traten keine Probleme auf, wenn beide Komponenten des Klienten auf einem System ausgeführt werden. Im Folgenden werden die Aspekte skizziert, die dieses Verhalten hervorrufen.

BOS weist unbekannte IP-Adresse ab Nachdem die Authentisierung erfolgreich durchgeführt wurde, werden die Verbindungs- und Authentisierungsinformationen an den zweiten

Klienten übermittelt. Dieser baut nun eine Verbindung zum BOS-Server auf. Der Server weist jedoch den Verbindungsaufbau mit einem Datagramm mit gesetzten RST/ACK-Flag ab.

Da die Verbindung ohne den Empfang des *Authorization Cookie* vom Server abgewiesen wird, kann davon ausgegangen werden, dass der *Authentification Server* den BOS-Server über die IP-Adresse des Klienten informiert hat. Alle unbekannten IP-Adressen, dass heißt, solche, die sich nicht am *Authentification Server* angemeldet haben, werden sofort abgewiesen.

Dieser Schutz kann aber umgangen werden, indem der zweite Klient sich zuvor mit einem beliebigen Account am *Authentification Server* anmeldet. Somit kennt der BOS-Server die IP-Adresse und weist Verbindungsanfragen nicht von vornherein ab.

In der Praxis führt dieses Vorgehen zu einem sehr großen Problem: Jedem Benutzername wird ein anderer BOS-Server zugewiesen, auf dem er sich anmelden soll. Daher ist es wichtig, dass eine Authentisierung mit einer BOS-Adresse gefunden wird, die der der Authentisierungskomponente (Klient 1) entspricht. Diese Zuweisung von Benutzername zu der IP-Adresse bleibt über einen längeren Zeitraum bestehen (mehr als 24 Stunden).

Für diese Demonstration mussten achtzehn unabhängige Benutzeridentifikatoren ermittelt werden, die bei der Authentisierung jeweils verschiedene BOS-Server lieferten, bis zwei Adressen identisch waren. Diese Anzahl ist nur ein zufälliger Wert.

An diese neuen Erkenntnisse musste die Softwarekomponente, die sich am BOS-Server anmeldet, angepasst werden.

***Authorization Cookie* stimmt nicht mit der IP-Adresse des Klienten überein** Das Lösen des zuvor aufgetretenen Problems mit der unbekannten IP-Adresse führt aber nicht zum Erfolg. Der Server akzeptiert nun zwar die TCP-Verbindung und empfängt das *Authorization Cookie*, lehnt jedoch die weitere Kommunikation mit einem RST-Datagramm ab.

Daher kann davon ausgegangen werden, dass das *Authorization Cookie* mit der entsprechenden IP-Adresse des authentisierten Klienten verknüpft ist. Es kann aber keine Aussage getroffen werden, ob der *Authentification Server* die IP-Adresse vom Klienten explizit überträgt oder ob sie in dem *Authorization Cookie* kodiert ist.

6.3. SSL-Man-In-The-Middle-Attack

Für die vollständige Analyse der IM-Protokolle musste jeder Datagrammstrom zwischen dem Sender und dem Empfänger mitgelesen werden. Einige Protokolle wie das des *MSN Messengers* und des klassischen *Jabbers* schützten ihre Verbindungen jedoch teilweise oder komplett mit dem *Secure Sockets Layer*-Protokoll (SSL, siehe Anhang B.15). Dieser Datagrammstrom musste daher entschlüsselt werden, um die Funktionsweise dieser Protokolle zu verstehen. Dieser Abschnitt soll zeigen, auf welchem Weg die Informationen gewonnen wurden.

Der Server authentisiert sich gegenüber dem Klienten mit einem Zertifikat. Einige Klienten ignorieren aber die Korrektheit des benötigten Zertifikats. Diese Tatsache vereinfachte den Angriff sehr stark. Zunächst muss ein Schlüssel erstellt werden, der durch ein Zertifikat bestätigt wird. Das OpenSSH-Paket, das für viele Unix-Derivate verfügbar ist, beinhaltet alle hierfür benötigten Werkzeuge.

Das OpenSSH-Paket beinhaltet ein Script namens „CA.sh“ oder „CA.pl“ zur Erstellung von

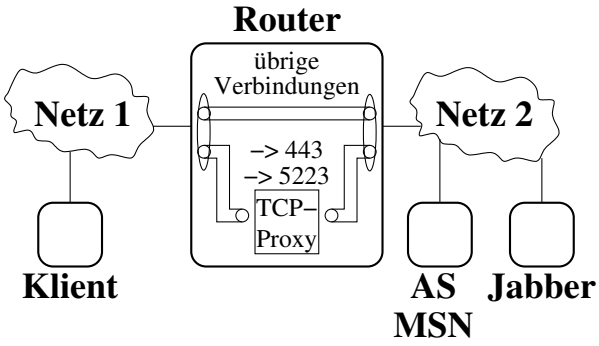

Abbildung 6: Mitlesen einer SSL-geschützten HTTP- oder *Jabber*-Verbindung

Zertifikaten. Für die folgenden Anweisungen muss sich das Script in dem Verzeichnis „/usr/local/openssl/misc/“ befinden.

Im ersten Schritt muss ein privater Schlüssel erstellt werden. Dies kann sehr leicht mit der folgenden Anweisung durchgeführt werden:

```
bash-2.05b$ /usr/local/openssl/misc/CA.sh -newca
```

Nach Eingabe der benötigen *Passphrase* (siehe Anhang B.8) und einiger weiterer Informationen ist die Generierung des Schlüssels abgeschlossen.

Nun muss aus dem Schlüssel ein Zertifikat erstellt werden. Dies kann mit

```
bash-2.05b$ /usr/local/openssl/misc/CA.sh -newreq
```

umgesetzt werden.

Idealerweise muss nun das Zertifikat durch eine vertrauenswürdige Stelle unterzeichnet werden. Dies würde aber keine dieser Stellen für die Unterstützung eines Angriffs tun. Daher wird das Zertifikat selber unterschrieben. Dies geschieht mit der Anweisung

```
bash-2.05b$ /usr/local/openssl/misc/CA.sh -sign
```

Nun sollte sich im aktuellen Verzeichnis die Datei „newreq.pem“ mit dem öffentlicher Schlüssel und die Datei „newcert.pem“ mit dem unterschriebenen Zertifikat befinden.

Diese Dateien müssen nun auf den Rechner, von dem die *Man-In-The-Middle-Attack* durchgeführt werden soll, übertragen werden. In dem hier beschriebenen Szenario handelt es sich um einen Router, der sich zwischen dem Klienten und dem Server befindet. Diese Anordnung soll den Angriff so unkompliziert wie möglich gestalten. Der Angriff könnte auch direkt auf dem Klienten, Server oder an Systemen, die an deren Netzen direkt angeschlossen sind, durchgeführt werden.

Für den eigentlichen Angriff wird der TCP-Proxy „SoCAT“ (*Socket Cat*[7]) eingesetzt. Dieser nimmt die gewünschten TCP-Verbindungen an, gibt die empfangenen Informationen aus und leitet die Daten an den eigentlichen Empfänger weiter.

Da „SoCAT“ in einem SSL-Strom geschaltet werden soll, wird der Applikation das erstellte Zertifikat übergeben. Dieses wird an den Klienten herausgereicht, der die Informationen mit dem

[7] **Verweis auf Applikation:** http://www.dest-unreach.org/socat

Schlüssel schützt, der an das Zertifikat gebunden ist. Mit Hilfe des selbst erstellen Schlüssels können diese problemlos entschlüsselt werden. Die empfangenen Daten werden nun mit dem Schlüssel des eigentlichen Servers geschützt und an diesen übertragen.

Um die TCP-Verbindung zu dem Authentisierungsserver des *MSN Messenger*-Systems (*login.passport.com:https* beziehungsweise 65.54.183.198:443) anzugreifen, öffnet „SoCAT“ einen lokalen TCP-Port (hier: 5555). Alle Verbindungen, die an diesen Port adressiert sind, leitet „SoCAT“ mit dem folgenden Befehl an den MSN-Authentisierungsserver weiter:

```
bash-2.05b$ socat -v OPENSSL-LISTEN:5555,key=newreq.pem,cert=newcert.pem,\
verify=0,fork,method=SSLv2,cipher=MD5 OPENSSL:65.54.183.198:443,verify=0,\
method=SSLv2,cipher=MD5
```

Ist das Ziel des Angriffs der *Jabber*-Server *jabber.org*, muss „SoCAT“ mit dem folgenden Befehl aktiviert werden:

```
bash-2.05b$ socat -v OPENSSL-LISTEN:5556,key=newreq.pem,cert=newcert.pem,\
verify=0,fork,method=TLSv1,cipher=3DES OPENSSL:208.245.212.67:5223,      \
verify=0,method=TLSv1,cipher=3DES
```

Um jeweils einen „SoCAT“ für MSN als auch *Jabber* parallel nutzen zu können, lauscht dieser Dienst auf Port 5556.

Um den Angriff zu starten, müssen die Verbindungen direkt an „SoCAT“ umgeleitet werden. Da es sich bei dem System mit „SoCAT“ um einen Router zwischen dem Klienten und dem Server handelt, ist das durch einfache Paketfilterregeln möglich.

Diese Paketfilterregeln sind abhängig vom verwendeten Betriebssystem. Für die Analyse wurde OpenBSD eingesetzt, da das Regelwerk auch für nicht OpenBSD-Benutzer sehr verständlich ist.

Durch das Aktivieren folgenden Paketfilterregeln ist die Vorbereitung des Angriffs abgeschlossen.

```
rdr on fxp0 inet proto tcp to 65.54.183.198  port 443 ->127.0.0.1 port 5555
rdr on fxp0 inet proto tcp to 208.245.212.67 port 5223->127.0.0.1 port 5556
```

6.4. Abhören einer Verbindung zwischen *Jabber*-Servern durch Kompromittierung der Server Dialbacks

Opfer dieses Angriffs, der für diese Diplomarbeit erstellt wurde, sind *Server Dialbacks*. Wird er erfolgreich ausgeführt, kann sich ein Angreifer für einen anderen *Jabber*-Server ausgeben, um in Verbindung mit einer *Man-In-The-Middle-Attack* Angriffe auf die Integrität, Vertraulichkeit und Verfügbarkeit durchzuführen.

Der folgende Angriff basiert auf theoretischen Überlegungen. Da eine Vielzahl von Servern (sowohl DNS- als auch *Jabber*-Server) und Adressbereichen benötigt werden, wurde er nicht in die Praxis umgesetzt.

Zum besseren Verständnis des Angriffs wird der Ablauf eines *Server Dialbacks*, der im Kapitel 10.2.3 vorgestellt wird, grob skizziert. In der folgenden Zusammenfassung, die durch Abbildung 7 unterstützt wird, werden die DNS-Abfragen integriert:

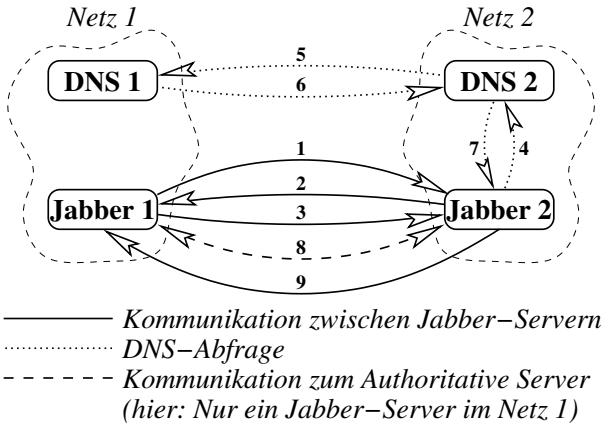

Abbildung 7: Ablauf eines *Server Dialbacks*

1. „Jabber 1“ baut eine TCP-Verbindung zu „Jabber 2“ auf. Diesem folgt das initialisierende „<stream:stream>“.
2. „Jabber 2“ beantwortet das „<stream:stream>“.
3. „Jabber 1“ generiert einen *Dialback Key* und übermittelt diesen „Jabber 2“.
4. Um den für das „Netz 1“ zuständigen *Authoritative Server* zu ermitteln, stellt „Jabber 2“ eine entsprechenden Anfrage an den lokalen „DNS 2“.
5. „DNS 2“ leitet die Anfrage an den für das „Netz 1“ zuständigen „DNS 1“ weiter.
6. „DNS 1“ beantwortet die Anfrage von „DNS 2“.
7. „DNS 2“ speichert in der Regel diese Antwort für spätere Anfragen zwischen und sendet sie an den „Jabber 2“, der die Anfrage gestellt hat.
8. Mit der vom „DNS 2“ erhaltende Adresse baut nun der „Jabber 2“ eine Verbindung zum *Authoritative Server* im „Netz 1“ auf. Nun wird der *Dialback Key* überprüft und die Verbindung wieder abgebaut.
9. „Jabber 2“ übermittelt das Resultat der *Dialback*-Überprüfung an „Jabber 1“. Bei Erfolg wird die Anmeldung abgeschlossen.

Die DNS-Abfrage ermöglicht einen klassischen Angriff durch *DNS Spoofing* (siehe Anhang B.17.4). Wird davon ausgegangen, dass sich das *Netz 1* unter der Kontrolle eines Angreifers befindet, kann ein Angreifer eine korrekte Authentisierung durchführen, bei der in der DNS-Antwort zusätzlich die IP-Adresse eines anderen Servers, für den er sich ausgeben möchte, übertragen wird. In Abbildung 8 wird ein *Server Dialback* mit den möglichen Angriff darstellt.

Generell ist der Informationsaustausch in Abbildung 8 mit dem aus Abbildung 7 identisch. Hinzugekommen ist der *Jabber*-Server „Eve b“, über dem die Kommunikation zwischen „Alice“ und „Bob“ laufen soll. Die Nachrichten, die für „Eve a“ benötigt werden, sind in der Abbildung

8 durch den Zusatz „a“ diesem Server zugeordnet. Analog gilt dies für den *Jabber*-Server „Eve b“ und der Nachrichtenzuordnung „b“.

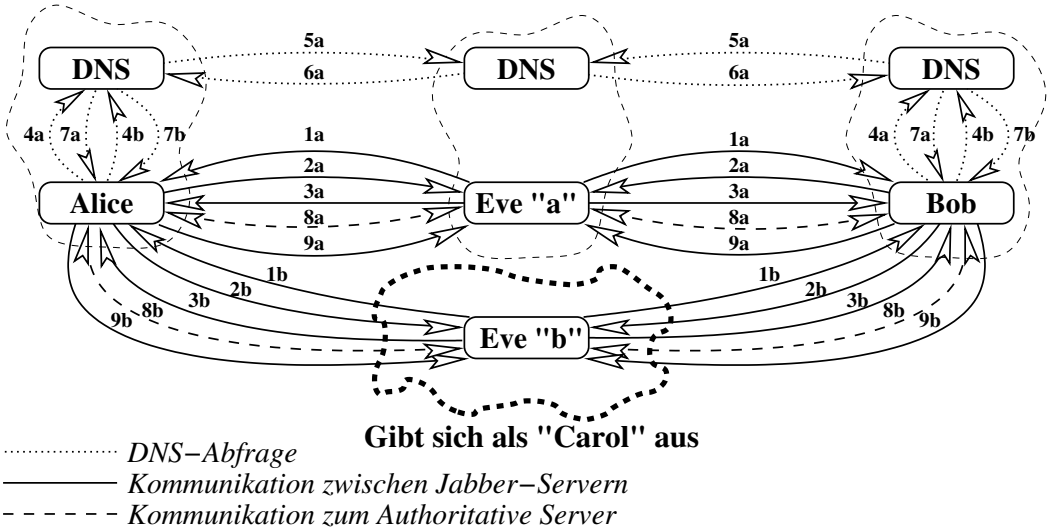

Abbildung 8: *Man-In-The-Middle-Attack* auf *Server Dialbacks*

Im Unterschied zu dem obigen Beispiel wurde die Antwort von dem DNS-Server von „Eve“ geändert:

6a Der DNS-Server schickt nicht nur die IP-Adresse zu „Eve a“, sondern zusätzlich auch die von „Eve b“, die dem Hostnamen von „Carol“ zugeordnet wird.

Diese nicht angeforderte IP-Adresse wird nun auf den DNS-Servern von „Alice“ und “Bob“ abgelegt. Wird nun kurz darauf eine Anfrage auf die Adresse „Carol“ gestartet, wird diese Adresse ohne weitere Abfrage an den für „Carol“ zuständigen DNS-Server aus dem Speicher geholt. Die Überprüfung des *Dialback Keys* findet mit der Adresse von „Eve b“ statt.

Abbildung 8 zeigt den Informationsaustausch einer *Man-In-The-Middle-Attack*. Dabei wurde die Abbildung so gestaltet, dass die Kommunikation sowohl zwischen „Alice“ und „Eve“ als auch zwischen „Eve“ und „Bob“ symmetrisch abläuft. Zum leichteren Verständnis ist die Betrachtung zwischen zwei Kommunikationspartnern ausreichend.

Sorgt nun der Angreifer dafür, dass die TCP-Verbindung bestehen bleibt (zum Beispiel durch regelmäßigen Informationsaustausch, um einen *Timeout* zu vermeiden), wird für eine Kommunikation eines Benutzers von „Alice“, mit einem Benutzer, den er auf „Carol“ vermutet, diese TCP-Verbindung benutzt.

7. Zusammenfassung

In Tabelle 1 werden die Kommunikationssysteme, die in dieser Diplomarbeit vorgestellt werden, hinsichtlich der in den vorigen Kapiteln untersuchten Aspekte gegenübergestellt.

	ICQ	AIM	MSN	Jabber	IRC	Silc
1. Instant Messaging und Konferenzsysteme						
Instant Messaging	Ja	Ja	Ja	Ja	Nein	Nein
Konferenzsystem	Nein	Nein	Nein	Nein	Ja	Ja
2. Zielgruppen von Instant Messaging						
Geeignet für privaten Einsatz	Ja	Ja	Ja	Ja	Ja	Ja
Geeignet für Einsatz in Unternehmen	Nein	Nein	Nein	Ja	Ja	Ja
3. IM-Protokollunabhängige Angriffe						
	kein Vergleich möglich, da tiefere Schichten Ziel der Angriffe sind					
4. Probleme durch Kommunikationssysteme						
Organisatorische Mängel	Betrifft alle Kommunikationssysteme					
Menschliche Fehlhandlungen	Betrifft alle Kommunikationssysteme					
Tech. Versagen: Nutzung von öffentlichen Servern	Betrifft alle Kommunikationssysteme					
Tech. Versagen: Schutz der übertragenen Informationen	Kein Schutz integriert	Kein Schutz integriert	Kein Schutz integriert	Ja	Kein Schutz integriert	Ja
Tech. Versagen: Verfügbarkeit der Serversoftware	Nein	Nein	Nein	Ja	Ja	Ja
Tech. Versagen: Einsatz von Paketfiltern	Schwer	Schwer	Ja	Ja	Ja	Ja
Tech. Versagen: Vielfalt an Protokollen	nur kompatibel zu AIM	nur kompatibel zu ICQ	inkompatibel	Kompatiblität möglich	inkompatibel	inkompatibel
Vorsätzliche Handlungen	Betrifft alle Kommunikationssysteme					
5. Marktanteile						
	keine präzise Aussage möglich					
6. Werkzeuge und Angriffszenarien						
Angriff auf ungesch. Verbindungen	Ja			krypto-graphisch geschützt	Ja	krypto-graphisch geschützt
Abhängigkeit des Authorization Cookies	Ja	Ja	entfällt			
SSL-Man-In-The-Middle-Attack	entfällt		Ja	Ja	entfällt	
Kompromittierung der Server Dialbacks	entfällt			Ja	entfällt	

Tabelle 1: Zusammenfassung - Allgemeine Aspekte

Teil II.
Kommunikationssysteme

Der folgende Abschnitt beschäftigt sich mit den verschiedenen *Instant Messaging*- und Konferenzsystemen. Für eine genauere Untersuchung dieser Systeme, die unter der Bezeichnung Kommunikationssysteme zusammengefasst werden, wurden die Systeme mit den höchsten Marktanteilen ausgewählt.

Vertiefende Informationen zu der Funktionsweise sind im Anhang A zu finden. Dort werden die Protokollvorschriften vorgestellt, mit denen Nachrichten gebildet und über ein Netz ausgetauscht werden. Diese Informationen sind für das Verständnis der Kommunikationssysteme und der Angriffe notwendig.

Die Kommunikationssysteme werden in den folgenden Kapiteln anhand der aufgeführten Kriterien verglichen:

Architektur Im Abschnitt zu der Architektur des Systems werden vorrangig die beteiligten Komponenten vorgestellt. Hierzu gehört die Anordnung von Servern, Klienten, Netzen und wenn vorhanden, weiterer Bestandteile wie *Gateways*. Wenn dies möglich ist, wird die Serverarchitektur ebenfalls skizziert.

Schutz der Kommunikation Einige Kommunikationssysteme stellen Mechanismen zum Schutz vor Angriffen bereit. Die in dieser Diplomarbeit betrachteten sind in der Regel direkt im Protokoll implementiert. Dies bedeutet, dass die übertragenen Informationen während des Nachrichtenaustausch durch kryptographische Verfahren geschützt werden.

Da aber nur sehr wenige Systeme Schutzmechanismen bereitstellen, fehlt dieser Abschnitt in den Kapiteln der ungeschützten Systeme.

Mögliche Angriffe In diesem Abschnitt werden mögliche Ansätze für Angriffe, insbesondere auf die Kommunikation der einzelnen Komponenten der Kommunikationssysteme, vorgestellt. Bei vielen Systemen reichen schon die bekannten Ansätze aus, wie *Sniffen* oder *TCP-Hijacking*, deren Anwendung im Kapitel 6.1 beschrieben wird. Aber auch für die Systeme, die einen Schutz der Kommunikation bieten sollen, wurden Szenarien für Angriffe erarbeitet. In der Regel werden in diesem Abschnitt nur theoretische Überlegungen zu einem Angriff vorgestellt. In Kapitel 6 wurde die praktische Umsetzung einiger dieser Überlegungen demonstriert.

Maßnahmen zur Verhinderung der unautorisierten Nutzung in Organisationen Im gewerblichen Umfeld ist die Nutzung von Kommunikationssystemen nicht immer erwünscht. Dies gilt vorrangig für die private Nutzung, die Angriffe ermöglichen kann oder verlorene Arbeitszeit zur Folge hat.

Daher sind in diesem Zusammenhang Maßnahmen zur Verhinderung wünschenswert. Einige Betreiber der Kommunikationssysteme versuchen aber, dies zu unterdrücken. Wie es dennoch möglich ist, zeigt dieser Abschnitt zu dem jeweiligen Kommunikationssystem.

Fazit Diese abschließende Zusammenfassung fasst jedes Kommunikationssystem nach den oben genannten Gesichtspunkten zusammen.

8. ICQ und AIM

Die Protokolle des *AOL Instant Messenger* (AIM) und von ICQ haben sehr viele Gemeinsamkeiten, da sie von den gleichen Entwicklern stammen. Daher wird die Analyse dieser beiden *Instant Messaging*-Systeme in einem Kapitel zusammengefasst.

ICQ war das erste *Instant Messaging*-System überhaupt und wurde 1996 von der Firma Mirabilis veröffentlicht. ICQ, das ausgesprochen so viel wie „I seek you“ („Ich suche Dich“) bedeutet, ist durch die frühe Veröffentlichung sehr weit verbreitet.

Im Jahre 1998 wurde Mirabilis von AOL aufgekauft. Bis zu diesem Zeitpunkt setzte AOL auf seinen Telegrammdienst, der bis zum Jahre 1997 AOL-Kunden vorbehalten war. Durch den hohen Marktanteil des Providers und die Möglichkeit, mit den Kunden über den Telegrammdienst zu kommunizieren, besaß er eine sehr große Bedeutung.

Nach dem Aufkauf von ICQ war AOL bemüht, die beiden *Instant Messaging*-Systeme zu einem zusammenzufassen. Die beiden Systeme besitzen aber einen gravierenden Unterschied, der das Zusammenfassen erschwert: Die Identifikatoren der Benutzer bestehen bei ICQ nur aus Zahlen, während bei AIM auch Buchstaben möglich sind. Durch die vielen Benutzer war es unmöglich, nachträglich eine einheitliche Gestalt der Identifikatoren festzulegen. So war die Neuvergabe von Identifikatoren unzumutbar, da diese Identifikatoren auch allen Kontakten mitgeteilt werden mussten. Daher entschied sich AOL für ein neues *Instant Messaging*-Protokoll, das sowohl ICQ als auch AIM beherrscht: OSCAR.

Um die Vormachtstellung auszubauen, wurde OSCAR im Laufe der Zeit einigen Änderungen unterzogen, die die Integration in die Klienten anderer großer Softwarehersteller verhindern sollten. Um aber die Vielzahl der Benutzer freier Software nicht auszuschließen, wurde ein weiteres Protokoll entwickelt: TOC. Dessen Funktionalität ist im Vergleich zu OSCAR eingeschränkt, da nur die zwingend benötigte Kommunikation unterstützt wird.

Es existieren nicht nur Unterschiede zwischen TOC und OSCAR, sondern auch zwischen AIM und ICQ. ICQ kennt eine Vielzahl von verschiedenen Online-Zuständen. Der Benutzer kann seinen Kontakten mitteilen, ob er beispielsweise „Online“, „Nicht Anwesend“ oder „Beschäftigt“ ist. Im Gegensatz dazu bieten AIM-Klienten nur die Möglichkeit, nach längerer Inaktivität des Benutzer bei einer ankommenden Nachricht mit einer „Away“-Mitteilung zu reagieren und dies durch einen entsprechenden Status den Kontakten mitzuteilen.

Die Analyse des OSCAR- und TOC-Protokolls im Anhang A.1 und A.2 zeigt einige Probleme, die anschließend in separaten Kapiteln beleuchtet werden. Zur Protokollanalyse wurden hauptsächlich Klienten genutzt, die für diese Diplomarbeit erstellt wurden.

8.1. Architektur

8.1.1. Klienten

Klienten sind Instanzen, die dem Benutzer die Kommunikation ermöglicht. Prinzipiell baut die Klienteninstanz eine Verbindung zu der Serverinstanz auf, über die die gesamte Kommunikation

stattfindet. Dabei ist es bei den beiden Protokollen OSCAR und TOC unerheblich, ob diese Systeme direkt am Internet angeschlossen sind oder ob dazwischenliegende Systeme wie (NAT)-Gateways oder Proxys den Zugang bereitstellen.

Neben der offiziellen Klientensoftware für *MacOS* und *Windows* existieren zahlreiche weitere Klienten für alle verbreiteten Betriebssysteme sowohl für das OSCAR- als auch das TOC-Protokoll. Zusätzlich sind webbasierte Klienteninstanzen verfügbar, die die Nutzung des IM-Systems ermöglichen. Beispiele für den Einsatz dieser Klienteninstanzen sind fremdadministrierte Systeme, bei denen der Benutzer keine Software installieren darf oder bei denen die Netzarchitektur eine herkömmliche Nutzung verhindert.

Viele Klienten unterstützen direkt die Registrierung der Benutzer über die Klienteninstanzen. Zusätzlich können die Benutzer über eine Webseite eine Zugangskennung erhalten. ICQ verwendet numerische Kennungen, die folgend als Identifikator oder Benutzernamen bezeichnet werden. Dabei muss dieser Identifikator mindestens fünf Stellen umfassen. Bei AIM muss die erste Stelle hingegen ein Buchstabe sein, dem neben Zahlen auch weitere Buchstaben folgen können. Bei ICQ können die Passwörter maximal acht Zeichen und unter AIM sechzehn Zeichen lang sein.

Die später folgende Protokollanalyse berücksichtigt nur den Informationsaustausch zwischen zwei Klienteninstanzen (*Peer-To-Peer*) oder zwischen einer Klienteninstanz und einer Serverinstanz - Verbindungen zwischen Serverinstanzen werden nicht betrachtet. Neben den offiziellen Klienten wurden zwei selbst entwickelte Klienten eingesetzt, die sich im Anhang C.1 (OSCAR) und C.2 (TOC) befinden.

8.1.2. Server

Je nach Protokoll (OSCAR oder TOC) kommunizieren die Klienteninstanzen mit verschiedenen Serverinstanzen. Die Kommunikation zwischen diesen Serverinstanzen kann in der Protokollanalyse nicht berücksichtigt werden, da die Serversoftware nicht frei verfügbar ist und die Kommunikation der entfernten Serverinstanzen untereinander nicht abgehört werden kann. Daher kann auch die Struktur des Servers nicht analysiert werden. Die bekannten Server werden im Folgenden kurz vorgestellt:

login.oscar.aol.com Die Klienteninstanz muss sich bei der Nutzung von OSCAR gegenüber diesem System authentisieren. Obwohl der TCP-Port 5190 für die Nutzung reserviert wurde, steht dem Klienten die Auswahl eines TCP-Ports am Server frei. Verläuft die Authentisierung erfolgreich, erhält die Klienteninstanz die Adresse vom *Basic Oscar Service* (BOS)-Server und einen Schlüssel, um sich darauf anzumelden.

BOS Nachdem die Klienteninstanz die Adresse und Portnummer von *login.oscar.aol.com* erhalten hat, muss sie sich für die eigentliche Nutzung des *Instant Messaging*-Systems am BOS-Server anmelden. Während der gesamten Nutzung des *Instant Messaging*-Systems bleibt die Verbindung bestehen.

toc.oscar.aol.com Bei der Nutzung eines Klienten, der nur das TOC-Protokoll unterstützt, muss sich dieser bei *toc.oscar.aol.com* anmelden. Eine Trennung zwischen Authentisierung und

eigentlicher Kommunikation ist nicht vorgesehen - für beides ist diese Serverinstanz zuständig. Für die TCP-Kommunikation wurde der Port 9898 reserviert, aber wie bei den OSCAR-Servern läuft die OC-Serverinstanz auf einer Vielzahl weiterer Ports.

Die aufgebaute TCP-Verbindung zwischen der Klienten- und der Serverinstanz bleibt während der gesamten Nutzung des *Instant Messaging*-Systems bestehen.

8.1.3. Aufbau eines Netzes

Abbildung 9 zeigt die Struktur eines Netzes. In dem Beispiel kommunizieren jeweils zwei Klienteninstanzen („3“ und „5“) direkt miteinander, wofür im Vorfeld die IP-Adresse und Portnummer über eine Server-Kommunikation ausgehandelt wurde. Im Weiteren sind eine angemeldete Klienteninstanz („4“) und zwei Klienteninstanzen („1“ und „2“) zu erkennen, die über entsprechende Serverinstanzen eine Kommunikation führen.

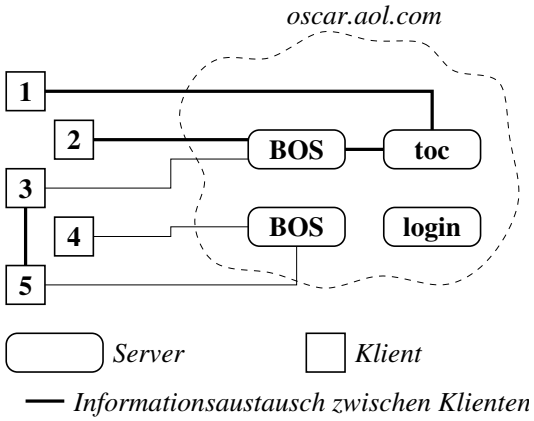

Abbildung 9: ICQ/AIM - Architektur eines Netzes

8.2. Mögliche Angriffe

Ein Angriff ist denkbar einfach, da keine Schwachstellen im Protokoll ausgenutzt werden müssen. Gegen einen Angriff auf einer tieferen Ebene des ISO/OSI-Referenzmodells bieten diese Protokolle kaum Schutzmechanismen. Somit können bis auf das Benutzerpasswort bei der Anmeldung alle übertragenen Informationen durch *Sniffen* (siehe Anhang B.16, eventuell in Verbindung mit *Spoofing*, Anhang B.17) abgehört werden. Auch durch *Session Hijacking* (Anhang B.18) kann ein Angriff auf die Integrität durchgeführt werden. Diese Defizite zeigt der im Kapitel 6.1 beschriebene Angriff, der für die IM-Protokolle TOC und OSCAR angewendet werden kann.

8.2.1. Bedrohung

Abhören der Nutzdaten Die Analysen von OSCAR und TOC haben gezeigt, dass diese Protokolle nur einen unzureichenden Schutz bieten. ***Durch die nicht verfügbare Ver-***

schlüsselung ist es jedem Angreifer, der die Datagramme protokollieren kann, möglich, diese zu lesen.

Die Analyse der abgehörten Informationen ist sehr einfach: Bei der Auswertung fallen dem Angreifer sofort die Nachrichten zwischen den Steueranweisungen ins Auge. Hierfür muss dieser nur sehr wenige Kenntnisse über das Protokoll besitzen.

Erschwert wird das *Sniffen* durch die Verwendung anderer TCP-Ports. Dazu werden entweder die Grundeinstellungen des Klienten verändert oder eine *Peer-To-Peer*-Verbindung genutzt, bei der die Portnummer variieren kann. Dies stellt aber keinen wirklichen Schutz dar, da der Angreifer die Portnummern ermitteln kann.

Abhören des Passworts Problematisch ist nur das Abhören des Passwortes, da dies geschützt übertragen wird. Wird das Passwort mit einem festen Schlüssel XOR-kodiert (TOC und OSCAR/ICQ) übertragen und kennt der Angreifer den Schlüssel, kann er das Passwort sehr leicht ermitteln. Anders sieht es bei Passwörtern aus, die als *md5*-Prüfsumme (OSCAR/AIM) übertragen werden. Über diesen Weg kann das Passwort nur mit höheren Aufwand ermittelt werden.

Obwohl die OSCAR-Entwickler mehr oder weniger effektive Schutzmechanismen der Passwortübertragung während der Authentisierung umgesetzt haben, wurde der Schutz bei der Aktualisierung des Passworts vernachlässigt.

Viele Klienten bieten die Möglichkeit, das Passwort direkt durch die Instanz zu ändern. Da in OSCAR hierfür *SNACs* vorgesehen sind, ist diese Änderung für den Benutzer sehr leicht durchzuführen. Das TOC-Protokoll sieht Änderungen direkt durch den Klienten nicht vor.

Die Protokollanalyse von OSCAR hat aber gezeigt, dass bei einer Passwortänderung durch die Klienteninstanz das Passwort sehr oft ungeschützt übertragen wird. Da dieses Defizit sehr vielen Benutzern nicht bekannt ist, könnte dies für einen Angriff ausgenutzt werden. Gelingt es ihm durch einen* Social Engineering*-Angriff, dass der Benutzer sein Passwort über den Klienten ändert, kann er es abfangen.

Dem Benutzer steht auch eine sichere Möglichkeit zur Verfügung. Besonders lizenzierte Klienten verweisen auf die Änderung des Passworts mittels HTTPS.

Vortäuschen einer fremden Identität Dazu muss es einem Angreifer gelingen, nach dem Empfang des *Authorization Cookie* und des Abbaus der Verbindung zum Authentisierungsserver die neue Verbindung des Klienten zum BOS-Server zu verhindern. Hat er zusätzlich das *Authorization Cookie* abgefangen, so könnte er sich an Stelle des authentisierten Benutzers beim BOS-Server anmelden. Der für diese Diplomarbeit erstellte Versuch im Kapitel 6.2 zeigte, dass der Server das *Authorization Cookie* an eine IP-Adresse bindet. Daher muss ein Angreifer zusätzlich seine Adresse fälschen (*IP-Spoofing*, siehe Anhang B.17.3).

Vortäuschen einer fremden Identität bei *Peer-To-Peer*-Verbindungen Die Möglichkeit, Nachrichten direkt an den Empfänger zu senden, wirft die Frage auf, ob sich ein Angreifer als ein direkter Kommunikationspartner („Peer“) ausgeben kann. Hierfür müsste er theoretisch nur die *Direct Connection*-Parameter des Benutzers abhören und könnte diesen direkt

kontaktieren, ohne sich gegenüber dem Server authentisieren zu müssen.

In der Praxis sieht dies aber anders aus. Der Server benachrichtigt den Benutzer über das Schreiben der Nachricht. Daher ist es die Aufgabe der Klienteninstanz, den Benutzer zu warnen, wenn dieser eine Nachricht direkt von einem Kontakt erhält, ohne dass dies die Serverinstanz angekündigt hat. Viele Klienten unterstützen dies, es kann aber jederzeit eine neue Klientenversion ohne diese Sicherheitsfunktionalität veröffentlicht werden.

Problematisch wird dies, wenn die (inoffizielle) Klientensoftware des Kontakts die OSCAR-Spezifikation verletzt und grundsätzlich auf das Versenden dieser Benachrichtigungen verzichtet. In diesem Fall könnte die Klienteninstanz des Empfängers keinen Unterschied feststellen.

8.2.2. Schutzmechanismen

Einige Klienten bieten die Möglichkeit, die Nachrichten zu verschlüsseln. Diese Option, die primär von *OpenSource*-Klienten geboten wird, kann meist nur genutzt werden, wenn der Kommunikationspartner die gleiche Software einsetzt. Als Ansatz für eine verschlüsselte Übertragung könnte PGP (siehe Anhang B.11) dienen, da die Verschlüsselungsapplikation auch außerhalb der Klienteninstanz eingesetzt werden kann. So kann ein Benutzer, dessen Klientensoftware keine Verschlüsselung unterstützt, das Chiffrat durch eine externe Applikation erstellen und in die IM-Klienteninstanz importieren.

Da die externe Verschlüsselung mit mehr Aufwand verbunden ist, sind nur sehr wenige Benutzer bereit, ihre Nachrichten zu schützen. Aber ohne eine Verschlüsselung zwischen dem Sender und dem Empfänger können alle dazwischenliegenden Systeme die übertragenen Informationen einsehen.

8.3. Maßnahmen zur Verhinderung der unautorisierten Nutzung in Organisationen

Der Einsatz von *Instant Messaging*-Systemen ist nicht immer erwünscht. Besonders wenn die Gesamtsicherheit oder die Produktivität der Mitarbeiter darunter leidet, ist die technische Unterdrückung der Nutzung von ICQ und AIM wichtig.

Die Protokollanalysen haben gezeigt, dass für die TCP-Kommunikation mittels OSCAR der Port 5190 und für TOC der Port 9898 vorgesehen ist. Diese Portnummern sind aber nur als Empfehlung anzusehen und daher nicht bindend.

Die Serverinstanzen, die die Kommunikation über OSCAR oder TOC realisieren sollen, lauschen auf einer Vielzahl von TCP-Ports. Testreihen haben gezeigt, dass bis auf den Port 80 (HTTP) keine Probleme bei der Verwendung von Portnummern weit verbreiteter Dienste auftraten. Bei Port 80 war zwar eine Authentisierung unter OSCAR möglich, aber die Verbindung zum BOS-Server scheiterte.

Dies hat für den Schutz eines LANs durch einen Paketfilter gravierende Auswirkungen. Sobald direkte Verbindungen neben HTTP aus dem LAN ins Internet gestattet werden, können die Benutzer ICQ beziehungsweise AIM einsetzen.

Das Blocken aller ausgehenden Verbindungen in Kombination mit dem Einsatz eines Proxy-Servers führt nicht zum gewünschten Erfolg. Sobald der Proxy neben HTTP auch FTP oder

HTTPS unterstützt, können die Benutzer über den Proxy eine Verbindung zu dem IM-Server aufbauen.

Diese Problematik verhindert eine Filterung nach IM-Portnummern. Der mit dieser Maßnahme verwandte Ansatz, die Filterung nach Hostnamen vorzunehmen, ist auch sehr schwer zu realisieren. Es ist sehr aufwändig, die Listen mit den IM-Servern aktuell zu halten. Sobald irgendein vorher nicht bekannter Server oder ein Server, der die Verbindungen zu einem IM-Server weiterleitet, hinzukommt, ist dieser Schutz unwirksam.

Zusammengefasst ist die eigenständige, technische Unterdrückung der Kommunikation zu IM-Servern nur sehr schwer zu realisieren. Wenn es die Policy der Institution zulässt, könnte eine Auswertung der Datagramme, die das LAN verlassen, zum Ziel führen, da OSCAR- und TOC-Datagramme einen nahezu identischen Header besitzen.

Dem TCP-Header eines OSCAR- oder TOC-Datagramms folgt der Wert 0x2a. Diesem Wert folgt die *Channel ID*, die auch nur eine begrenzte Anzahl an Werten annehmen kann. Je mehr Datagramme mit dieser Charakteristik das LAN verlassen, desto Wahrscheinlicher ist es, dass es sich um eine AIM- oder ICQ-Verbindung handelt.

Von einer automatischen Sperrung aller Verbindungen, die dieses Merkmal aufweisen, ist abzuraten, da auch andere Verbindungen dieser Charakteristik aufweisen können. Aber nach der Analyse steht es dem Verantwortlichen frei, alle Verbindungen zu diesem neuen IM-Server zu unterdrücken. Verstößt die Nutzung gegen die Policy, kann der hierfür Verantwortliche entsprechend reagieren.

8.4. Fazit

Obwohl die AOL-Produkte den Markt der *Instant Messaging*-Systeme nicht dominieren, besitzen sie einen großen Kundenstamm. Gerade ICQ-Benutzer, die seit der ersten Stunde *Instant Messaging* nutzen und eine dementsprechend große Kontaktliste führen, sehen keinen Grund, ein anderes Produkt zu verwenden.

Auch für die Vergabe der Identifikatoren hat ICQ eine sehr elegante Lösung gefunden. Da diese direkt von ICQ vergeben werden, besitzen die Benutzer keinerlei Einfluss auf die Gestalt. Bei AIM werden diese vom Benutzer vorgeschlagen und wenn sie frei sind, dem Benutzer übergeben. Da beliebte AIM-Namen schnell vergeben sind, müssen später hinzukommende Benutzer einen weniger aussagekräftigen Identifikator wählen.

Die von AOL gewählte Netzarchitektur ist bei vielen proprietären IM-Systemen zu finden. Durch das Bündeln vieler Server in einem Netzsegment können sehr effizient Ausfälle einzelner Systeme ausgeglichen werden. Außerdem kann durch den Einsatz weiterer Server die Kapazität erhöht werden. Da aber keine Server außerhalb des Netzsegments bekannt sind, muss immer eine Verbindung zu diesen Systemen aufgebaut werden. Die Verwendung von amerikanischen Servern ist für europäische und asiatische Benutzer nicht effizient.

Ein großes, aber bisher noch weitgehend unbekanntes Problem ist die unverschlüsselte Passwortübertragung während einer Änderung. Bis auf diesen Nachteil erfüllen die Protokolle ihre Aufgabe zuverlässig, obwohl die Unterstützung von zwei verschiedenen Protokollen (OSCAR und TOC) nicht sehr elegant ist. Warum die Kommunikation zwischen ICQ-Benutzern, die OSCAR nutzen, und AIM-Benutzern, die TOC verwenden, nicht erwünscht ist, konnte nicht geklärt werden.

Auch die Lösung, den zu nutzenden Zeichensatz vom Benutzer wählen zu lassen, ist nicht sonderlich elegant. Kommunizieren asiatische und mitteleuropäische Benutzer mit individuellen Zeichensätzen, kann eine eindeutige Interpretation der übertragenen Zeichen nicht garantiert werden.

Wird während der Authentisierung *md5* eingesetzt, das in dieser Form nur bei AIM mit OSCAR möglich ist, ist das Abhören des Passworts nur sehr schwer möglich. Warum dieser Weg aber nicht auch bei der ICQ-Authentisierung in Verbindung mit OSCAR eingeschlagen wurde, ist nicht nachvollziehbar. Da TOC nicht ausreichend vor Angriffen schützt, sollte OSCAR favorisiert werden.

Da für den Verbindungsaufbau zum Server eine Vielzahl von Ports gestattet sind, können die Betreiber der lokalen Netze die Verwendung des IM-Systems nur sehr schwer unterbinden, was eine unkontrollierte Nutzung von ICQ oder AIM in Unternehmen zur Folge haben kann.

9. MSN Messenger

Die Betrachtung der proprietären *Instant Messenger* schließt der ***M**icro**s**oft **N**etwork Messenger* ab. Diese Klientenapplikation, die auch als *.NET Messenger* bezeichnet wird, basiert auf dem *MSN Messenger Protocol*. Ein weiterer offizieller Softwareklient, der auf diesem Protokoll basiert, ist der direkt in neuere Windowsversionen integrierte *Windows Messenger*. Im Folgenden werden all diese Klienten als *MSN Messenger* zusammengefasst.

Das *MSN Messenger Protocol* wurde im August 1999 mit der Versionsnummer 1 direkt von Microsoft veröffentlicht (siehe [11]). Ab der Versionsnummer 7, die im Oktober 2003 veröffentlicht wurde, ist das Protokoll nicht mehr abwärtskompatibel. Um die Grundfunktionalitäten zu verstehen, eignete sich jedoch die zu neueren Versionen inkompatible Protokollspezifikation mit der Nummer 1.

Zu allen der Version 1 folgenden Versionen existieren keine offiziellen Spezifikationen von Microsoft. Als Hilfe konnte jedoch die inoffizielle Spezifikation der Version 7 von Mike Mintz herangezogen werden, die unter [12] verfügbar ist. Die eigentliche Analyse (Siehe Anhang A.3) berücksichtigt die Versionsnummer 9, die der in diesem Zusammenhang erstelle Klient unter C.3 unterstützt. Anzumerken ist, dass die Unterschiede zwischen den Protokollen ab der Version 7 zu vernachlässigen sind.

Die in dieser Analyse vorgestellten Grundlagen wurden aus Interoperabilitätsgründen primär durch *Reverse Engineering* gewonnen. Diese Ergebnisse wurden anschließend mit den oben genannten Spezifikationen verglichen.

9.1. Architektur

9.1.1. Klienten

Bei Klienten handelt es sich um Instanzen, die mit den zu dem *Instant Messaging*-System gehörenden Serverinstanzen kommunizieren. Sie ermöglichen die Anmeldung und die eigentliche Nutzung des *Instant Messaging*-Systems, indem sie die Schnittstelle zwischen Benutzer und IM-System bilden.

Dem Benutzer stehen zahlreiche Klienten zur Verfügung. Neben dem offiziellen, direkt von *Microsoft* stammenden Klienten gibt es für alle verbreitete Betriebssysteme inoffizielle Klienten. Um aber den Umfang der versendeten Datagramme auf das nötigste zu beschränken, wurde für die im Anhang Anhang A.3 folgende Analyse hauptsächlich der unter Anhang C.3 zu findende Klient verwendet.

Der *MSN Messenger* ist wesentlich endkundenorientierter als die bisher vorgestellten Systeme. Es existieren mehr Statusmöglichkeiten und dem Benutzer werden beispielsweise netzbasierte Spiele angeboten, die direkt in dem offiziellen Klienten integriert sind. Um die Nutzung von verschiedenen Rechnern aus zu ermöglichen, werden viele Konfigurationseinstellungen, wie die Kontaktlisten, direkt auf dem Server abgelegt. Um multimediale Informationen zu übermitteln, werden die Nachrichten im *MIME*-Format übertragen. Hierfür muss aber während der Übertragung der Empfänger angemeldet sein - eine Übermittlung an nicht angemeldete Benutzer und die damit verbundene Zwischenspeicherung ist nicht möglich.

Als eindeutiger Identifikator wird die E-Mail-Adresse genutzt. Besitzt der Benutzer ein Konto bei *hotmail*[8], so kann er den *MSN Messenger* ohne Registrierung mit diesem nutzen. Der Vorteil hierbei ist, dass das E-Mail-Konto vollständig in den *MSN Messenger* integriert wird. Benutzer, die kein *hotmail* E-Mail-Konto besitzen, können den *MSN Messenger* nutzen, indem sie sich explizit registrieren. E-Mails können bei dieser Option aber nicht mit dem *MSN Messenger* abgerufen oder versendet werden.

Da die E-Mail-Adresse in Kontaktlisten nicht immer die optisch schönste Lösung ist, kann jeder Benutzer einen Alias-Namen verwenden, der in den Kontaktlisten der anderen Benutzer geführt wird. Dieser Alias muss nicht eindeutig gewählt werden.

9.1.2. Server

Auch der *MSN Messenger* benötigt wie die *Instant Messaging*-Systeme AIM/ICQ mehrere Serverinstanzen für eine Kommunikation. Diese werden im Folgenden kurz vorgestellt:

Dispatch Server (DS) Nach einem Verbindungsaufbau und einer entsprechenden Anfrage der Klienteninstanz liefert der *Dispatch Server* die IP-Adresse und die Portnummer des *Notification Servers*. Sind diese Informationen der Klienteninstanz von einer anderen Informationsquelle bekannt, muss sie diese nicht vom *Dispatch Server* abfragen.

Notification Server (NS) Der *Notification Servers* steuert über den TCP-Port 1893 den gesamten Datenfluss, der während der Nutzung des *MSN Messenger* anfällt. Dazu gehören die Statusverwaltung und die Benachrichtigungen über eine ankommende Nachricht, die zu der Anmeldung am *Switchboard Server* führt. Daher muss diese TCP-Verbindung während der gesamten Nutzung des *Instant Messaging*-Systems bestehen bleiben.

Authentification Server (AS) Die Adresse des *Authentification Servers* erhält die Klienteninstanz während der Anmeldung am *Notification Server*. Der *Authentification Server* kennt alle Benutzerinformationen, die zur Authentisierung benötigt werden. Kann der Benutzer dem

[8] *Hotmail.de:* http://login.passport.net/uilogin.srf?id=2

Authentification Server seine Identität nachweisen, erhält er einen Schlüssel, mit dem er die Anmeldung am *Notification Server* abschließen kann. Die TCP-Verbindung wird direkt nach der Authentisierung beendet.

Switchboard Server (SBS) Über den *Switchboard Server* können Klienteninstanzen Textnachrichten austauschen - eine *Peer-To-Peer*-Kommunikation ist nicht vorgesehen. Um mit einem anderen Benutzer in Kontakt zu treten, übermittelt der Sender eine entsprechende Nachricht an den *Notification Server*. Dieser übermittelt dem Sender darauf hin die Adresse des *Switchboard Server* und informiert den gewünschten Empfänger. Nachdem sich dieser ebenfalls am *Switchboard Server* angemeldet hat, können beide Gesprächspartner die gewünschte Kommunikation führen.

9.1.3. Aufbau eines Netzes

Abbildung 10 zeigt den groben Aufbau eines *MSN*-Netzes. Neben den zur Zeit nicht benutzten *Authentification Server* und *Dispatch Server* führen zwei Klienteninstanzen („2“ und „3“) ein Gespräch über den *Switchboard Server*. Zusätzlich ist eine weitere Klienteninstanz („1“) am *Notification Server* angemeldet.

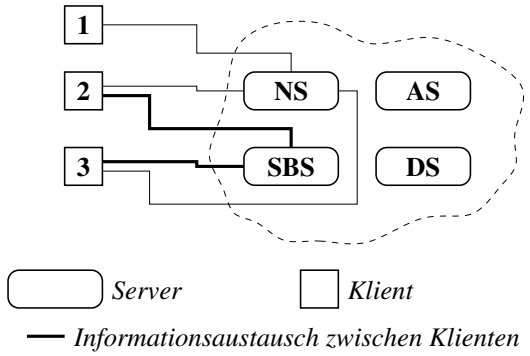

Abbildung 10: MSN - Architektur eines Netzes

9.2. Mögliche Angriffe

9.2.1. Bedrohung

Der im Kapitel 6.1 vorgestellte Angriff auf die Verfügbarkeit, Integrität und Vertraulichkeit konnte sehr einfach für das *MSN Messenger*-Protokoll übernommen werden. Insbesondere das ASCII-Format der übertragenden Informationen ermöglicht ein effizienten Angriff durch *TCP-Hijacking* (siehe Anhang B.18) - nur der *Request Identifikator* musste angepasst werden. Im Folgenden werden einige der gewonnenen Erkenntnisse vorgestellt.

Abhören der Nutzlast Wie viele andere IM-Systeme bietet auch das *MSN Messenger*-Protokoll keine Verschlüsselung der Nutzlast. ***Durch* Sniffen *(siehe Anhang B.16) kann***

ein Angreifer, der sich zwischen dem Sender und dem Empfänger befindet, sämtliche Nachrichten zwischen diesen beiden abhören.

Die Verbindung zwischen *Switchboard Server* und Klienten ist genauso gefährdet wie der Informationsaustausch des Klienten mit dem *Notification Server*. Ein Angreifer kann sie wie im Kapitel 6.1 beschrieben abhören, übernehmen und beenden.

Erschwert wird das Lauschen durch die Verwendung einer eigenständigen TCP-Verbindung zum *Switchboard Server*, die für jede Kommunikation neu ausgehandelt wird. Da die TCP-Ports und die Serveradressen variieren können, ist das zielgerichtete Abhören eines Opfers sehr umständlich. Bei der entsprechenden Motivation des Angreifers kann dieser Mehraufwand aber nicht als Schutz angesehen werden.

Dieses Problem ist Microsoft bewusst. Kein anderer Klient informiert den Benutzer darüber, dass die Kommunikation unsicher ist. Bei Microsoft wird dies mit der Mitteilung „Übermitteln Sie bei Sofortmeldungen niemals Ihr Kennwort oder Ihre Kreditkartennummer“ innerhalb des Fensters für die Nachrichteneingabe umgesetzt.

Abhören des Passworts Die Kommunikation zwischen Klienteninstanz und dem *Authentification Server* wird durch eine HTTPS-Verbindung geschützt. ***Die Implementierung der SSL-Verschlüsselung ist jedoch mangelhaft und daher anfällig für eine* Man-In-The-Middle-Attack*.***

Obwohl der Server ein unterschriebenes Zertifikat besitzt, wurde es in keinster Weise vom Klienten überprüft. Nicht einmal der Benutzer besitzt Zugriff auf das aktuelle Zertifikat. Die Klienteninstanz bemerkte noch nicht einmal, dass das Zertifikat von dem Originalen zu dem Gefälschten geändert wurde. Ein Angreifer kann durch den in der verwendeten Literatur in diesem Zusammenhang nicht eingesetzten und im Kapitel 6.3 beschriebenen Angriff das im Klartext übermittelte Passwort abfangen, ohne dass der Benutzer dies überhaupt bemerken oder verhindern kann.

9.2.2. Schutzmechanismen

Schutz vor dem *Sniffen* bietet nur eine Verschlüsselung, die direkt auf den Klienteninstanzen vorgenommen wird. Hier gibt es wie bei allen vorgestellten Systemen keine standardisierte Möglichkeit, da viele verschiedene Klienten (offizielle und inoffizielle) auf dem Markt verfügbar sind.

Die bei Browsern bekannte Strategie könnte auch hier das Problem mit den Zertifikaten lösen: Der Benutzer bekommt bei seiner ersten Anmeldung das Zertifikat angezeigt und muss dieses annehmen. Diese Verlagerung des Problems von Microsoft zu dem Benutzer ist aber auch nicht besonders wirkungsvoll. Eleganter wäre es, wenn Informationen zu dem Zertifikat direkt mit der Klientensoftware ausgeliefert werden würde. Dies wird insbesondere dadurch erleichtert, dass nur sehr wenige *Authentification Server* verwendet werden. Ein Angreifer könnte diesen Ansatz nur kompromittieren, indem er die Software beim Download zwischen dem Anbieter der Software und dem Benutzer manipuliert.

9.3. Maßnahmen zur Verhinderung der unautorisierten Nutzung in Organisationen

Im Gegensatz zu den vom AIM und ICQ verwendeten *OSCAR*-Servern lauschen die *MSN*-Server nur auf dem vorgesehenen TCP-Port 1863. Diese Tatsache ermöglicht eine unproblematische Filterung über entsprechende Paketfilterregeln. Es ist aber nicht auszuschließen, dass Microsoft irgendwann die AOL-Strategie übernimmt, um genau dieses Filtern durch lokale Netzbetreiber zu unterbinden.

Bei der Verwendung eines HTTP-Proxys im lokalen Netz sind entsprechende restriktive Regeln ebenfalls unproblematisch umzusetzen. Alle *MSN Messenger*-Anfragen besitzen den *Content-Type* „application/x-msn-messenger“ (Bei *Squid*: „req_mime_type“). Entsprechende Anfragen können entweder geblockt oder nur protokolliert werden.

Wird jedoch von dem Endbenutzer ein fremder Proxy verwendet, ist eine entsprechende Filterung nur sehr schwer umzusetzen.

9.4. Fazit

Zusammengefasst kann gesagt werden, dass der *MSN Messenger* - besonders die offizielle Windowsversion - von allen *IM*-Systemen am benutzerfreundlichsten ist. Er lässt sich beispielsweise gut in Windows integrieren, da beides vom selben Hersteller stammt. Gerade Benutzer, die nur über eine sehr geringen Bandbreite für den Anschluss ans Internet verfügen, versuchen größere Downloads wie einen *Instant Messenger* von einem anderen Anbieter zu vermeiden.

Weniger erfahrene Benutzer werden zuerst den *MSN Messenger* ausprobieren, da dieser für sie am wenigsten Aufwand mit sich bringt. Zum Nachteil anderer *Instant Messaging*-Systeme ist die Integration des *MSN Messenger* in die Standardinstallation des Betriebssystems marktpolitisch ein sehr wichtiger Schritt von Microsoft.

Keines der anderen Kommunikationssysteme benötigt so viele verschiedene Server wie der *MSN Messenger*. Die Funktionalität wird zwischen einer Serverinstanz für die Anmeldung, für die Verwaltung und für die eigentliche Kommunikation aufgeteilt. Einer der größten Vorteile ist die Skalierbarkeit, da jederzeit weitere Server aufgenommen werden können.

Die Vielzahl der benötigen Server führt theoretisch aber zu einem größeren Ausfallpotential. Kann beispielsweise der *Switchboard Server* wegen eines Ausfalls oder Netzproblemen nicht kontaktiert werden, dann können keine Nachrichten ausgetauscht werden, wenn der *Notification Server* nicht kurzfristig auf diesen Ausfall reagiert. Außerdem kann die zusätzliche Kommunikation Ziel eines Angriffs sein.

Das Protokoll ist sehr durchdacht und die Datagramme, die mit dieser Vorschrift generiert werden, sind sehr effizient. Durch die Kodierung der Nachrichten im MIME-Format treten in der Praxis keine Probleme mit Zeichensätzen auf.

Auch der Schutz der Privatsphäre durch verschiedene Listen (siehe Anhang A.3.3) ist eine sehr elegante Möglichkeit. Dieses Verfahren bietet viele Vorteile gegenüber expliziten Anfragen, die sonst der Benutzer stellen beziehungsweise beantworten müsste.

Um das gesamte System so einfach wie möglich zu gestalten, besitzt der Benutzer keinerlei Einfluss auf die Sicherheit. Und hier liegt das größte Defizit, das weniger auf der Protokollarchitektur als auf deren Umsetzung beruht. Obwohl SSL wirkungsvoll implementiert werden

kann, wurde dies von Microsoft nicht umgesetzt. Dadurch, dass die Klientensoftware das Zertifikat des Authentisierungsserver ignoriert, kann ein Angriff sehr leicht durchgeführt werden. Wenn schon der Benutzer die SSL-Zertifikate überhaupt nicht betrachten kann, so sollte er zumindest über ein Wechsel der Zertifikate benachrichtigt werden. Ansonsten kann ein Angreifer direkt das Passwort mitlesen, ohne dass der Benutzer hierauf Einfluss besitzt.

Zur Zeit lässt sich die unkontrollierte Nutzung beispielsweise in Organisationen durch Paketfilterregeln sehr effizient unterbinden. Im Gegensatz zu den von AOL betriebenen Systemen können die MSN-Server nur über einen Port kontaktiert werden. Weiterhin bleibt aber die Nutzung von webbasierten Klienten für die unautorisierte Nutzung problematisch.

Zusammenfassend kann gesagt werden, dass bei der vorliegenden Implementierung das Entfernen der SSL-Verschlüsselung den Schutz der Authentisierung nur wenig verringern würde. Für Benutzer, die keine Anforderungen an den Schutz Ihrer Daten stellen, ist die Verwendung des *MSN Messengers* eine komfortable Alternative.

10. Jabber

Neben den schon sehr lange auf dem Markt etablierten IM-Systeme wie ICQ, AIM und *MSN Messenger* gewinnt *Jabber* insbesondere beim Einsatz in Organisationen und Unternehmen zunehmend an Bedeutung. Der Hauptvorteil liegt in der Verfügbarkeit von Serversoftware, die die Integration eines lokalen *Instant Messaging*-Systems ermöglicht. Sensible Informationen, die zwischen Benutzern im lokalen Netz ausgetauscht werden, müssen hier das gesicherte Netz nicht verlassen.

Sowohl Steuerungsanweisungen als auch Nachrichten werden mit dem ***Ex**tensible **M**essaging and **P**resence **P**rotocol* (XMPP) oder dessen Vorgänger ausgetauscht. Diese XML-basierten Protokolle werden im Anhang A.4 vorgestellt.

Viele Organisationen haben schon die Vorteile von *Jabber* erkannt und umgesetzt. Ein Beispiel ist das „Auswärtige Amt“, das zu dem Zeitpunkt der Erstellung dieser Diplomarbeit für die Kommunikation innerhalb der Zentrale in Berlin einen *Jabber*-Server betreibt (Quelle: [13]). In nachfolgenden Schritten sollen die Außenstellen über die bestehende „Sichere Inter-Netzwerk Architektur“ (SINA) integriert werden. Die geschützte Einbindung entfernter Standorte kann in anderen Fällen auch durch *Virtuelle Private Netze* (VPN) realisiert werden.

Insbesondere für Privatanwender, deren Kontakte jeweils verschiedene IM-Systeme nutzen, bietet *Jabber* ebenfalls Vorteile. Viele öffentliche Server, die teilweise von Providern oder anderen Institutionen gepflegt werden, bieten die Möglichkeit, durch **Gateways** externe IM-Systeme zu nutzen. Da hinter *Jabber* keine bekannten Organisationen stehen, die es vermarkten, konnte sich *Jabber* bisher nicht auf dem Markt durchsetzen.

10.1. Architektur

10.1.1. Klienten

Das Angebot an verfügbarer Klientensoftware[9], die die Kommunikation zu Instanzen auf *Jabber*-Servern ermöglicht, ist sehr umfangreich. Für viele Betriebssysteme sind zahlreiche Kli-

[9] **Vertiefung:** http://www.jabber.org/software/clients.shtml

enten mit unterschiedlichen Lizenzmodellen verfügbar. Neben der reinen *Jabber*-Funktionalität unterstützen viele Klienten weitere IM-Protokolle. Dies wird aber nur benötigt, wenn der verwendete Server keine IM-Gateways anbietet.

Identifikatoren Mit Hilfe der Klienteninstanz kann sich der Benutzer an einem *Jabber*-Server anmelden und diesen für *Instant Messaging* nutzen. Hierfür muss er sich vorher mit einen eindeutigen Identifikator registrieren, der im Folgenden als JID (*Jabber Identificator*) bezeichnet wird.

Wichtige Bestandteile einer JID sind der Benutzername und der Name des Servers. Die JID hat sehr viel Ähnlichkeit mit einer E-Mail-Adresse, da die beiden Bestandteile mit dem Zeichen @ voneinander getrennt werden. Da der Servername Teil der JID ist, muss der Benutzername nur auf diesem Server eindeutig sein. Somit können oft verwendete Benutzernamen wie Nachnamen öfters vergeben werden. Damit ist der Benutzer aber auch an diesen Server gebunden und kann bei einem Ausfall des Servers auf keinen anderen ausweichen.

Ein weiterer Bestandteil einer JID ist die **Ressource**, die optional angegeben werden kann. Möchte sich ein Benutzer mehrmals mit einem Benutzernamen auf einem Server anmelden, da er sowohl Zuhause als auch auf der Arbeit mit dem Server verbunden ist, unterscheidet die Ressource diese angemeldeten Instanzen voneinander.

Zusammengefasst besitzt eine vollständige JID folgende Gestalt:

$$benutzername@serveradresse/ressource$$

Problematisch für die parallele Anmeldung mit einem Benutzernamen auf einem Server ist die Zustellung von Nachrichten. Ein Kontakt, der eine Nachricht versenden möchte, muss wissen, welche Ressource der Benutzer zur Zeit verwendet. Daher wurde eine numerische **Priorität** zwischen 1 und 999 eingeführt. Die Nachrichten werden bei einer fehlenden Ressourcenangabe an die Ressource mit der höchsten Priorität zugestellt. Sind mehrere Ressourcen mit gleicher Priorität angemeldet, stellt der Server die Nachricht der Instanz zu, die sich als letzte an dem Server angemeldet hat.

Um Nachrichten an eine Ressource mit einer niedrigeren Priorität zu schicken, muss der Kontakt die Ressource in der Zieladresse angeben. Um Probleme zu vermeiden, wird in der Regel die Ressource bei einer Antwort auf eine Nachricht durch den Klienten eingefügt.

Klient-Klient-Verschlüsselung Die RFC3923 (*End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol*[15]) definiert einen Standard für die direkte Verschlüsselung zwischen Klienteninstanzen. Obwohl der Datenstrom zu den beteiligten Servern über SSL beziehungsweise TLS verschlüsselt übertragen werden kann, ist dieser alleinige Schutz bei einer Kompromittierung eines Servers wirkungslos.

Dieser Standard wird sich aber nicht durchsetzen, weil er insbesondere bei den Entwicklern sehr unbeliebt ist (Quelle: [14]). Stattdessen wurde die Problematik durch den Einsatz von PGP (siehe Anhang B.11) gelöst.

Bei PGP handelt es sich um eine sehr weit verbreitete Lösung. Durch die Verfügbarkeit von *OpenSource* Bibliotheken kann PGP sehr effizient und problemlos in Klienten integriert werden. Obwohl dieses Verfahren in keiner *Jabber*-RFC berücksichtigt wird, setzen es viele Klienten ein.

10.1.2. Server

Die Aufgabe eines Servers besteht hauptsächlich in der Kommunikation mit den Klienteninstanzen, der Verarbeitung interner Anweisungen und der Kommunikation mit weiteren Serverinstanzen.

Ein *Jabber*-Server sollte ähnlich wie ein E-Mail-Server in dem zuständigen DNS-Server mit einer entsprechenden Zuweisung eingetragen werden. In der Praxis haben sich hierfür SVR-Records mit den *lookup*-Werten *_xmpp-server._tcp* oder *_jabber._tcp* durchgesetzt. Diese Eintragung wird für *Dialbacks* (Siehe Kapitel 10.2.3) zwischen Servern benötigt.

Verfügbarkeit und Unterschiede der Serversoftware Es steht eine Vielzahl von Serversoftware zur Verfügung. Die Auswahl einer geeigneten hängt neben dem unterstützten Betriebssystem und Lizenzmodell entscheidend vom Funktionsumfang ab. Auf den Webseiten zu *jabber.org*[10] gibt es einen Überblick über verfügbare Serversoftware.

Zu den verbreitetsten Servern zählen die *OpenSource*-Applikationen des *jabberd projects*[11]. Da sie eine Vielzahl von Softwarearchitekturen unterstützen, sollen sie in dieser Diplomarbeit genauer betrachtet werden.

Zum Zeitpunkt dieser Arbeit liegt die Serversoftware als Version 1 (Version 1.4.3 vom 15. November 2003) und Version 2 (2.0s6 vom 11. Dezember 2004) vor. Da die zuerst entwickelte Version 1 sehr unübersichtlich wurde und sich schwer erweitern ließ, wurde die Version 2 komplett neu implementiert. Da bei einer Neuimplementation oft zu Beginn viele Schwachstellen auftreten, wurde die Version 1 parallel dazu weiter gepflegt. Zum jetzigen Zeitpunkt sind jedoch für beide Versionen keine Schwachstellen bekannt.

Durch die Neuimplementierung wurden einige Vorteile der Version 2 gegenüber der früheren erreicht. Während in Version 1 ein zentraler Prozess die gesamte Kommunikation abwickelte, übernehmen dies in der Version 2 eigenständige Prozesse, die über TCP kooperieren. Die damit verbundene bessere Skalierbarkeit macht das System flexibler und somit für den Einsatz innerhalb von Organisationen besser geeignet. Des Weiteren ist in Version 2 auch die Integration von externen Authentisierungsmechanismen wie LDAP (*Lightweight Directory Access Protocol*) oder PAM (*Pluggable Authentication Module*) möglich. Benutzerinformationen können in *MySQL*, *PostGreSQL* oder in einer *Berkeley DB* abgelegt werden. Diese Unterstützung bietet Version 1 nur in Kombination mit externer Software.

Ein weiterer Unterschied sind die unterstützten Protokolle. Die ältere Version verwendet nur das klassische *Jabber*-Protokoll, wogegen in der Version 2 zusätzlich XMPP unterstützt wird.

In der Version 2 wurde auf die Integration weiterer IM-Systeme verzichtet. Diese Funktionalität kann nur durch den Einsatz eines „*Foreign IM Gateway*“ erreicht werden. Dieses System, das auf dem gleichen Rechner betrieben werden kann, kann durch einen *Jabber*-Server in der Version 1 bereitgestellt werden. Im Weiteren existieren unabhängige Gateways, die zahlreiche Funktionalitäten zur Verfügung stellen.

Trotzdem ist für den Einsatz in einer Organisationen Version 2 aufgrund der besseren Integration in die Anwendungslandschaft vorzuziehen. Für den Betrieb eines öffentlichen Servers

[10] **Vertiefung:** http://www.jabber.org/software/servers.html
[11] **Vertiefung:** http://jabberd.jabberstudio.org

hingegen, an dem sich die Benutzer selbst registrieren können, bietet die Version 1 elegantere Lösungsansätze.

Aufgrund der Ausrichtung dieser Diplomarbeit auf den vorrangig nicht privaten Einsatz von *Instant Messaging* wird hier daher primär auf die Version 2 eingegangen.

Architektur In diesem Abschnitt werden die Bestandteile eines *Jabber*-Servers betrachtet. Systeme wie der *jabberd2* sehen eine klare Trennung der Komponenten vor, die als unabhängige Prozesse agieren und über eigene Konfigurationsdateien beeinflusst werden. Die Kommunikation dieser Prozesse erfolgt über TCP-Verbindungen. Da dadurch die Prozesse auf verschiedenen Rechner betrieben werden können ist das System skalierbar. Ältere Systeme besitzen diese Trennung, wie sie in Abbildung 11 vorgestellt wird, nicht.

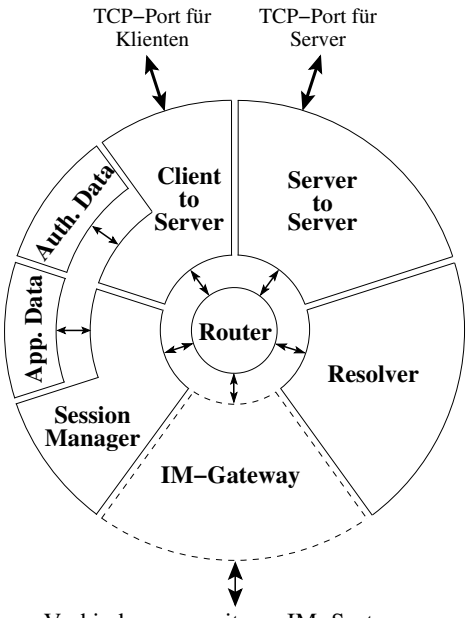

Abbildung 11: *Jabber* - Bestandteile eines Servers

Der zentrale Bestandteil eines *Jabber*-Servers ist der **Router**. Er öffnet einen TCP-Port, über den die anderen Komponenten mit ihm kommunizieren können. Der Informationsaustausch erfolgt über XML-Anweisungen, die die gesamte Steuerung des Servers ermöglichen. Im Beispiel des *jabberd2* wird ein Benutzername und ein Passwort für die TCP-Kommunikation benötigt.

Die **Client to Server**-Instanz ermöglicht der Klienteninstanz die Kommunikation mit dem Server. Hierfür öffnet sie einen TCP-Port (Voreinstellung: 5222), zu dem die Klienteninstanzen eine Verbindung aufbauen können. Zur Authentisierung vergleicht die *Client to Server*-Instanz Benutzername und Passwort mit den Inhalten des *Authentication Data Store*. Diese Informationen werden bei der Registrierung, für die die *Client to Server*-Instanz ebenfalls zuständig ist, eingetragen. Verläuft die Authentisierung erfolgreich, wird dies dem *Router* mitgeteilt, der diese Information an den *Session Manager* weiterleitet.

Der **Session Manager** setzt die eigentliche *Instant Messaging*-Funktionalität um. Dazu greift er auf den *Application Data Store* zu und veranlasst die *Client to Server*- beziehungsweise *Server to Server*-Instanz zur Bearbeitung und Weiterleitung von Nachrichten und Zustandsinformationen. Im Weiteren ist der *Session Manager* für die Pflege der Kontaktlisten mit den dazugehörigen „*Subscriptions*“ zuständig.

Analog zur *Client to Server*-Instanz verwaltet die **Server to Server**-Instanz die Kommunikation zu anderen Servern. Hierdurch kann ein Informationsaustausch zwischen lokalen und entfernten Benutzern realisiert werden. Da nur in Ausnahmefällen eine Authentisierung mit einem Passwort stattfindet, werden zur Überprüfung *Server Dialbacks* (Siehe Kapitel 10.2) verwendet. Dazu greift die *Server to Server*-Instanz über den *Router* auf den *Resolver* zu. Der **Resolver** wird für DNS-Anfragen im Zusammenhang mit *Server Dialbacks* benötigt.

Neben der Möglichkeit, für die Authentisierung der Benutzer *PAM* oder *LDAP* zu nutzen, kann ein separater **Authentication Data Store** mit dem **Application Data Store** in einer Datenbank zusammengefasst werden. *Jabberd2* unterstützt *MySQL*-, *PostGreSQL*- und *Berkeley*-Datenbanken, deren Server separat installiert werden müssen. Neben den optional in der Datenbank gepflegten Authentisierungsinformationen des *Authentication Data Store* werden Einträge für Kontaktlisten, Zeiten und Kontaktinformationen im *Application Data Store* gepflegt.

Konfiguration Nachdem *Jabberd2* durch distributionsabhängige Binärpakete oder manuell übersetzte Pakete erfolgreich installiert wurde, kann mit der Konfiguration begonnen werden.

Dazu wird eine bereits konfigurierte *MySQL*, *PostGreSQL* oder *Berkeley*-Datenbank benötigt. Die Erzeugung der benötigten Tabellen in der Datenbank kann von einem Script durchgeführt werden, das im *Jabberd2* enthalten ist. Im folgenden Beispiel wird *PostGreSQL* verwendet. Des Weiteren müssen auf dem verwendeten System einige spezielle Systemnutzer angelegt sein. Die Anmeldung soll nur dem System bekannten Benutzern über *PAM* gestattet werden.

In dem Verzeichnis für die Konfigurationsdateien befinden sich jeweils eine Datei für den *Router* („*router.xml*“), den *Resolver* („*resolver.xml*“), den *Session Manager* („*sm.xml*“), die *Client-to-Server*- („*c2s.xml*“) und die *Server-to-Server*-Instanz („*s2s.xml*“). Diese sind wie auch XMPP und das klassische *Jabber*-Protokoll in XML erstellt.

Im ersten Konfigurationsschritt müssen der DNS-Name und die zu verwendeten Datenspeicher (hier *PostgreSQL* und *PAM*) dem Programm übergeben werden. Der Hostname muss sowohl in der Datei „*sm.xml*“ als auch in der Datei „*c2s.xml*“ eingetragen werden. Die Daten werden in das XML-Knotenelement <*id*> eingetragen, das sich in der „*c2s.xml*“ innerhalb einer <*local*> Umgebung befindet:

```
<!-- sm.xml -->
  <id>Hostname</id>
```

beziehungsweise

```
<!-- c2s.xml -->
<local>
  ...
  <id>Hostname</id>
  ...
</local>
```

In der „*c2s.xml*“ wird im Abschnitt <*authreg*> als <*module*>-Eintrag *PAM* gewählt.

```
<!-- c2s.xml -->
<authreg>
  <module>pam</module>
  ...
  <!--<enable/>-->
  ...
</authreg>
```

Im selben Abschnitt sollte die Zeile <*enable/*> einkommentiert werden. Damit können sich keine neuen Benutzer mehr am IM-System registrieren. Benutzer, die das System nutzen wollen, müssen bereits als Systembenutzer vorhanden sein.

Zusätzlich muss für *PAM* eine entsprechende Konfigurationsdatei angelegt werden. Im einfachsten Fall kann dies mit

```
bash-2.05b$ cp /etc/pam.d/common-auth /etc/pam.d/jabberd
```

umgesetzt werden. In dem <*storage*>-Abschnitt der „*sm.xml*“ kann die Datenbank, in der die Benutzerinformationen abgelegt werden, definiert werden. Der Eintrag

```
<!-- sm.xml -->
<storage>
  <driver>pgsql</driver>
  ...
  <pgsql> ... </pgsql>
  ...
</atorage>
```

ermöglicht die Nutzung einer *PostgreSQL*-Datenbank. In einem speziellen Tag (hier <*pgsql*>) können Informationen für die Datenbank spezifiziert werden. Beispiele sind der Rechnername, auf dem sich die Datenbank befindet, ein Datenbankbenutzer und dessen Passwort.

Wenn sich ein Benutzer zum ersten Mal am System anmeldet, müssen seine Daten aus dem *PAM* in die Datenbank übernommen werden. Dieser Mechanismus wird durch das Auskommentieren des folgenden Eintrags aktiviert.

```
<!-- sm.xml -->
<user>
  ...
  <auto-create/>
  ...
</user>
```

Dieser Schritt entfällt, wenn eine Registrierung direkt am System möglich ist.

Damit ist die Grundkonfiguation abgeschlossen und der Server könnte gestartet werden. Bisher ist dieser jedoch nicht sicherheitskritisch konfiguriert. So können Klienten nur unverschlüsselt mit dem Server kommunizieren.

Des Weiteren bietet der Server mit der bisher vorgenommenen Konfiguration neben den für die Kommunikation mit Klienten und Servern benötigten Ports auch den TCP-Port 5347 für Verbindungen an. Damit kann eine Kommunikation mit dem *Router* erfolgen. Da sich in diesem Beispiel Server und Router auf einem System befinden, sollte dieser Port für einen Zugriff von

außen aus Sicherheitsgründen geschlossen werden. Zur Kommunikation mit dem *Router* benötigen alle Komponenten ein Passwort. Dieses sollte in allen Konfigurationsdateien auf ein selbst gewähltes geändert werden.

```
<!-- router.xml -->
<local>
  ...
  <ip>127.0.0.1</ip>
  <secret>neues passwort</secret>
  ...
</local>
```

Um sämtliche Verbindungen kryptographisch zu schützen, sollte ein RSA-Schlüssel (siehe B.13) generiert werden, der mit einem Zertifikat unterschrieben werden muss. Dieser Schutz wird auch empfohlen, wenn sich die einzelnen Komponenten auf einem System befinden, da die Kommunikation von einem lokalen Angreifer abgehört werden kann.

Um dieses Beispiel übersichtlich zu gestalten, wird nur ein Schlüssel sowohl für die Verschlüsselung zu Klienten, als auch zu den Servern und dem Router erstellt. Als Alternative sollte für jede dieser Verschlüsselungen ein eigenständiger Schlüssel erstellt werden.

Das *OpenSSL*-Paket, das für den Schutz der Verbindungen benötigt wird, ermöglicht die Generierung des Schlüssels:

```
bash-2.05b$ openssl req -new -x509 -newkey rsa:1024 -keyout privkey.pem\
                    -out server.pem
bash-2.05b$ openssl rsa -in privkey.pem -out privkey.pem
```

Der *Jabber*-Server muss Zugriff auf den Schlüssel und das Zertifikat besitzen - dafür müssen unter Umständen die Zugriffsrechte angepasst werden.

Um den Schlüssel dem Server zu übergeben, muss er in den entsprechenden Konfigurationsdateien („*c2s.xml*“, „*s2s.xml*“ und „*router.xml*“) eingetragen werden. Hierfür muss das folgende XML-Knotenelement angepasst werden, das den vollständigen Pfad des Zertifikats beinhaltet.

```
<!-- c2s.xml, s2s.xml und router.xml -->
<local>
  ...
  <pemfile>/usr/local/etc/jabberd/privkey.pem</pemfile>
  ...
</local>
```

In den Konfigurationsdateien aller Komponenten, die eine geschützt Verbindung zu dem *Router* aufbauen müssen, muss in dem <*router*>-Abschnitt das entsprechende Zertifikat angegeben werden.

```
<!-- c2s.xml, s2s.xml, resolver.xml, sm.xml -->
<router>
  ...
  <pemfile>/usr/local/etc/jabberd/privkey.pem</pemfile>
  ...
</router>
```

Um die Klienten zu der Nutzung der Verschlüsselung zu zwingen, sollte die „*c2s.xml*“ die Zeile

```
<!-- x2s.xml -->
<local>
  ...
  <require-starttls/>
  ...
</local>
```

enthalten.

Im Weiteren können durch eine entsprechende Konfiguration weitere Eigenschaften, wie eine IP-basierte Zugriffsbeschräkung oder die Protokollierung, erzielt werden. Da die Konfigurationsdateien ausführlich kommentiert sind, bleibt dies dem interessierten Leser selbst überlassen.

10.1.3. Externe Komponenten und Gateways

Jabber-Server bieten in der Grundkonfiguration nur die wichtigsten Funktionalitäten an. Sie können jedoch durch Module erweitert werden, die dann zusätzliche Dienste anbieten. Dies können Module für Konferenzen oder für die Kommunikation mit anderen *Instant Messaging*-Systemen sein. Eine Liste mit bekannten Erweiterungen ist auf der *Homepage* des *Jabber*-Projektes[12] zu finden.

Diese Trennung bietet viele Vorteile. Die Entwickler der *Jabber*-Server können sich auf den Softwareentwurf des eigentlichen Servers konzentrieren - bei Änderungen in einem proprietären *Instant Messaging*-Protokoll muss der eigentliche Server nicht überarbeitet werden. Durch die Konzentration auf die Server-Komponente können die Entwickler viele Ressourcen in die Stabilität stecken - für die Erweiterungen sind andere Entwickler zuständig.

Die Erweiterungen arbeiten als eigenständige Prozesse und kommunizieren mit der *Router*-Komponente über eine TCP-Verbindung. Zur Einbindung des Systems wird ein Alias für ihn auf dem *Jabber*-Server angelegt (Beispiel: *icq.jabberserver.org* für den Server *jabberserver.org*). Dieser muss in den Konfigurationsdateien des Dienstes und des Routers aufgenommen werden. Neben dem Eintrag, über welchen Port und IP-Adresse der *Router* erreichbar ist, enthält die Konfigurationsdatei des Dienstes weitere dienstspezifische Einträge.

Funktionsweise eines Gateways Der Benutzer erfährt über seine Klienteninstanz, welche Dienste vom Server angeboten werden. Um einen Gateway-Dienst nutzen zu können, muss sich der Benutzer hierfür registrieren. Dazu übergibt er der Klienteninstanz seinen vorhandenen Identifikator des *Instant Messaging*-Systems, das er nutzen möchte, und das zugehörige Passwort. Mit diesen Informationen kann sich nun der Gateway für den Benutzer an dem gewünschten IM-System anmelden.

Ist der Benutzer angemeldet, kann er nun seine Kontakte, die ebenfalls dieses System nutzen, in seine Kontaktliste eintragen. Diese erhalten JIDs mit dem Aufbau *IM-System_abhängige_ID@Alias_der_Erweiterung*.

Soll nun einem dieser Kontakte eine Nachricht gesendet werden, nimmt der Gateway diese an und übermittelt sie über das zu dem System gehörigen Protokoll an den eingetragenen Server.

[12] **Vertiefung:** http://www.jabber.org/software/components.shtml

Analog hierzu nimmt der Gateway die Nachricht von der IM-Serverinstanz an und benachrichtigt den Benutzer entsprechend.

10.1.4. Aufbau eines Netzes

Abbildung 12 zeigt einen möglichen Zusammenschluss von einem normalen Server („Jabberd 1“), einem „IM-Server“, der nicht auf *Jabber* basiert und einem *Jabber*-Server („Jabberd 2“), der dieses IM-System zusätzlich unterstützt. Für die Gestaltung dieses Beispielnetzes ist es irrelevant, welches *Instant Messaging*-System der „IM-Server“ anbietet. Beispiele sind ICQ, AIM oder der *MSN Messenger*.

Die Klienteninstanz „2“ in Abbildung 12 ist mit zwei Ressourcen („2a“ und „2b“) zur gleichen Zeit an dem „Jabberd 1“ angemeldet und kommuniziert mit der ebenfalls an dem Server angemeldeten Klienteninstanz „1“ (Verbindung „c“) und mit der Klienteninstanz „4“, die den „Jabberd 2“ nutzt (Verbindung „b“).

Weitere Kommunikationspartner sind die Klienteninstanz „3“ und die Klienteninstanz „6“. Dabei ist die Klienteninstanz „3“ an dem „Jabberd 2“ angemeldet, wogegen die Klienteninstanz „6“ einen anderes *Instant Messaging*-System nutzt (Verbindung „a“).

Die TCP-Verbindungen zwischen den Servern werden nur bei Bedarf aufgebaut. Beenden die Klienteninstanz „2a“ und „4“ ihr Gespräch, dann bleibt die Verbindung zwischen den Servern „Jabberd 1“ und „Jabberd 2“ bis zu einem *Timeout* bestehen.

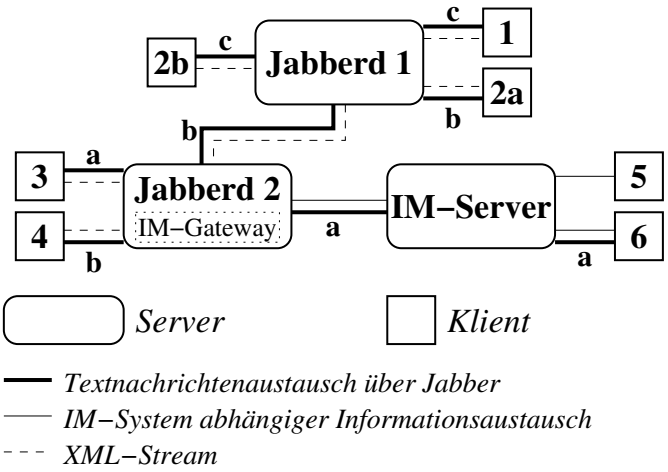

Abbildung 12: *Jabber* - Architektur eines Netzes

10.2. Schutz der Kommunikation

10.2.1. TLS over XMPP

Im Gegensatz zu dem klassischen *Jabber*-Protokoll, das für den Schutz der übertragenen Informationen *Secure Sockets Layer* (SSL, siehe Anhang B.15) verwendet, setzt XMPP vollständig auf den Nachfolger *Transport Layer Security* (TLS). Eine kurze Beschreibung des eigentlichen TLS-Protokolls ist im Anhang B.19 zu finden.

Während bei SSL eine eigenständige Verbindung über einen zusätzlichen Port ausgehandelt wird, übermittelt TLS die Informationen innerhalb der bestehenden TCP-Verbindung. Diese Verbindung setzt wie auch eine unverschlüsselten Verbindung einen initialisierenden XMPP-Stream („<stream:stream>“, siehe Protokollanalyse im Anhang A.4) voraus, dem die Anweisungen zu TLS folgen.

TLS kann für die Verbindung zwischen Klienten- und Serverinstanzen oder nur zwischen Serverinstanzen verwendet werden. Die Serverinstanz legt dabei fest, ob bei einer Anmeldung TLS verwendet werden kann oder muss. In der Regel informiert die Serverinstanz die Klienteninstanz über die Verwendung von TLS mit folgender „<stream:features>“-Anweisung:

```
<!--Sender -> anzumeldendes System -->
<stream:features>
  <starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'>
    <required/>
  </starttls>
  <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
    <mechanism>DIGEST-MD5</mechanism>
    <mechanism>PLAIN</mechanism>
  </mechanisms>
</stream:features>
```

Der „<mechanisms>“-*Tag* spielt für die Authentisierung über SASL eine entscheidende Rolle und wird im Kapitel 10.2.2 vorgestellt.

Das „<starttls>“-*Tag* informiert das anzumeldende System über die Unterstützung von TLS. Das *Tag* „<required/>“ zwingt die Klienteninstanz zu einer Nutzung vom *Transport Layer Security* - fehlt es, kann TLS optional verwendet werden.

Fordert die Serverinstanz die Nutzung von TLS, so ignoriert sie alle übertragenen Anweisungen des anzumeldenden Systems bis dieses die Verwendung von TLS signalisiert:

```
<!-- anzumeldendes System -> Server -->
<starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
```

Dies bestätigt die Serverinstanz mit folgender Anweisung:

```
<!-- Server -> anzumeldendes System -->
<proceed xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
```

Bei Problemen bricht die Serverinstanz den Verbindungsaufbau mit folgender Mitteilung ab:

```
<!-- Server -> anzumeldendes System -->
  <failure xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
</stream:stream>
```

Nachdem die vorangegangen Schritte erfolgreich ausgeführt wurden, handeln die beiden beteiligten Systeme über das *TLS Handshake Protocol* (siehe Anhang B.19) die Optionen und Zertifikate für die Verwendung von TLS aus.

Nachdem die Aushandlung des *TLS Handshake Protocol* erfolgreich abgeschlossen wurde, wird durch eine neue „<stream:stream>“-Instanz die Kommunikation fortgesetzt. Danach folgt die Authentisierung über SASL oder *Server Dialbacks*.

Nachdem diese Schritte ebenfalls ausgeführt wurden, werden alle folgenden Informationen verschlüsselt über das *TLS Record Protocol* (siehe ebenfalls Anhang B.19) übertragen.

10.2.2. SASL over XMPP

Simple Authentication and Security Layer (SASL) wird von vielen Protokollen neben XMPP wie POP3, SMTP, IMAP, oder LDAP zur Authentisierung verwendet. Der Gesprächspartner, bei dem sich der Sender authentisieren muss, sendet eine oder mehrere *Challenges*, die durch die gleiche Anzahl *Responses* beantwortet werden müssen. Dies kann sowohl eine Authentisierung von einem Klienten an einem Server als auch eine Authentisierung zwischen zwei Servern sein.

Um Probleme mit den Zeichensätzen zu vermeiden, werden die Zeichenketten, die den *Challenges* und *Responses* entsprechen, BASE64[13] kodiert. Zu schützende Informationen wie Passwörter können als Klartext oder als Prüfsummen („DIGEST-MD5“) übertragen werden.

Wird SASL von dem verwendeten *Jabber*-Protokoll unterstützt, wird dieses Verfahren direkt nach dem „<stream:stream>“-*Tag* des Empfängers eingeleitet. Zuerst übermittelt der Empfänger, der bei einer Klient-zu-Server-Kommunikation dem Server entspricht, die von ihm unterstützten Prüfsummenverfahren an den zu authentisierenden Gesprächspartner:

```
<!-- Server -> zu authentisierendes System -->
<stream:features xmlns:stream='http://etherx.jabber.org/streams'>
  <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
    <mechanism>DIGEST-MD5</mechanism>
    <mechanism>PLAIN</mechanism>
  </mechanisms>
</stream:features>
```

Aus dieser Auswahl wählt nun der zu Authentisierende ein Verfahren aus und übermittelt dies dem Gesprächspartner:

```
<!-- zu authentisierendes System -> Server -->
<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl' mechanism='DIGEST-MD5'></auth>
```

Nachdem das Verfahren ausgehandelt wurde, sendet der Gesprächspartner, der die Authentisierung erwartet (hier: Server), die erste *Challenge* an den Initiator. Der Server besitzt in dem folgenden Beispiel den Namen *jabber.clowntown.priv*:

```
<!-- Server -> zu authentisierendes System -->
<challenge xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
  cmVhbG09ImphYmJlci5jbG93bnRvd24ucHJpdiIsbm9uY2U9ImZkZGNlM2E3YjA3MTUzMTAzYzk
  5MTJkMTMxMzU2MzRiOGNhYzkzMGQiLHFvcD0iYXV0aCIsY2hhcnNldD11dGYtOCxhbGdvcml0aG
  09bWQ1LXNlc3M=
</challenge>
```

Die *Challenge* wird BASE64-kodiert übertragen. Umgewandelt in BASE8 entspricht dies:

```
realm="jabber.clowntown.priv",nonce="fddce3a7b07153103c9912d13135634b8cac93
0d",qop="auth",charset=utf-8,algorithm=md5-sess
```

Nachdem der zu Authentisierende diese Informationen erhalten hat, kann er mit einer *Response* antworten:

```
<!-- zu authentisierendes System -> Server -->
<response xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
```

[13] **Vertiefung:** RFC3548 http://www.ietf.org/rfc/rfc3548.txt

```
    Y2hhcnNldD11dGYtOCxjbm9uY2U9ImYyMTNkZDEyMmFlZmMxOWMyYjViMWM2ODBiZTFlYmUwIi
    xkaWdlc3QtdXJpPSIvIixuYz0wMDAwMDAwMSxub25jZT0iZmRkY2UzYTdiMDcxNTMxMDNjOTkx
    MmQxMzEzNTYzNGI4Y2FjOTMwZCIscW9wPWF1dGgscmVhbG09ImphYmJlci5jbG93bnRvd24ucH
    JpdiIscmVzcG9uc2U9NjQ3MGYwZGIzOWE4YTZiYTk3OTgwYTk4YWYxZWVlZjIsdXNlcm5hbWU9
    ImJlaXNwaWVsIg==
</response>
```

Auch diese BASE64-kodierte *Response* lässt sich umwandeln:

```
charset=utf-8,cnonce="f213dd122aefc19c2b5b1c680be1ebe0",digest-uri="/",
nc=00000001,nonce="fddce3a7b07153103c9912d13135634b8cac930d",qop=auth,
realm="jabber.clowntown.priv",response=6470f0db39a8a6ba97980a98af1eeef2,
username="beispiel"
```

Neben dem Benutzernamen (*username=*) spielt die Zeichenkette der eigentlichen *Response* (*response=*) eine zentrale Rolle: Diese beinhaltet unter anderem die Prüfsumme über das Passwort. Der Algorithmus, der die *Response* aus der von der Serverinstanz empfangenen Informationen wie *nonce* generiert, ist im Anhang C.5 zu finden.

Nachdem der Empfänger die Korrektheit das Passwortes überprüft hat, sendet er eine *Challenge*, die folgenden Aufbau besitzen kann, an den zu authentisierenden Gesprächspartner.

```
<!-- Server -> zu authentisierendes System -->
<challenge xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
    cnNwYXV0aD05ODIwNTNiMTQ2MDRhM2U1NDcxN2QwODYwMTdiZTE1Zg==
</challenge>
```

Transformiert lässt sich diese Zeichenkette wie folgt darstellen:

```
rspauth=982053b14604a3e54717d086017be15f
```

Der *rspauth=* wird ähnlich wie die vorige *Response* gebildet. Da die Klienteninstanz sowohl die Vorschrift als auch die Inhalte kennt, kann sie ebenfalls diese *rspauth=* generieren. Dadurch, dass ebenfalls das Passwort des Klienten (das nur der Klient und der Server kennen sollten) berücksichtigt wurde, kann die Klienteninstanz davon ausgehen, dass es sich bei dem Server wirklich um den gewünschten Server handelt.

In dem Fall, in dem die Authentisierung nicht erfolgreich verlief, bricht die Serverinstanz die Kommunikation mit folgender Nachricht ab:

```
<!-- Server -> zu authentisierendes System -->
  <failure xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
    <temporary-auth-failure/>
  </failure>
</stream:stream>
```

Konnte aber der zu authentisierende Gesprächspartner seine Identität der Serverinstanz nachweisen, kann er die letzte *Response* des Servers mit folgender Anweisung beantworten:

```
<!-- zu authentisierendes System -> Server -->
<response xmlns='urn:ietf:params:xml:ns:xmpp-sasl'></response>
```

Im letzten Schritt der Authentisierung sendet der Server folgende Mitteilung, um die SASL-Anmeldung abzuschließen:

```
<!-- Server -> zu authentisierendes System -->
<success xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>
```

Nachdem dieser Dialog erfolgreich abgeschlossen wurde, generiert der Initiator eine neue „<stream:stream>“-Instanz, um die eigentliche Kommunikation zu beginnen.

10.2.3. Server Dialback

Um Angriffen durch *DNS-Spoofing* (Siehe Anhang B.17.4) entgegenzuwirken, kann der Empfänger die Korrektheit des DNS-Namens des Sender durch einen *Server Dialback* überprüfen. Dies soll verhindern, dass ein Angreifer eine Verbindung zu dem Empfänger aufbaut und sich als ein anderer Server ausgibt. Dieser Mechanismus kann nur zwischen Servern verwendet werden.

In der Regel sollte innerhalb einer Domain SASL als Authentisierung zwischen mehreren Servern verwendet werden. Da aber nicht immer die Passwörter aller externen Server bekannt (und erforderlich) sind, stellen *Server Dialbacks* eine Alternative dar.

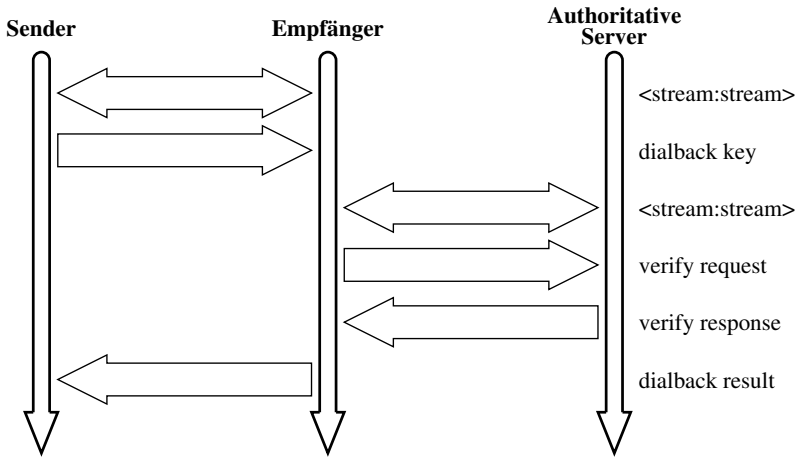

Abbildung 13: *Jabber* - Protokollablauf eines *Server Dialbacks*

Ein Protokollablaufdiagramm für ein *Server Dialback* ist in Abbildung 13 zu finden. Neben dem Sender und Empfänger spielt der *Authoritative Server* eine entscheidende Rolle: er ist der zentrale XMPP-Server in der DNS-Domain des Senders. Dafür muss er durch passende SVR-Records (*_xmpp-server._tcp* oder *_jabber._tcp*) im DNS-Server eingetragen sein. In kleinen Netzstrukturen sind in der Regel der ursprüngliche Sender und der *Authoritative Server* in einem System vereint.

Nachdem eine TCP-Verbindung zwischen Sender und Empfänger aufgebaut wurde, übermittelt der Sender innerhalb des „<stream:stream>“-*Headers* die Unterstützung von *Server Dialbacks*:

```
<!-- Sender -> Empfaenger -->
<stream:stream
  xmlns:stream='http://etherx.jabber.org/streams'
  xmlns='jabber:server'
  xmlns:db='jabber:server:dialback'>
```

Der initialisierende „<stream:stream>“-*Header* des Empfängers beinhaltet nun eine *ID*:

```
<!-- Empfaenger -> Sender -->
<stream:stream
    xmlns:stream='http://etherx.jabber.org/streams'
    xmlns='jabber:server'
    xmlns:db='jabber:server:dialback'
    id='457F9224A0...'>
```

Auf Grundlage dieser *ID*, dem DNS-Namen des Senders und einem Geheimnis, das nur der Sender und der *Authoritative Server* kennen, generiert der Sender einen *Dialback Key* und übermittelt diesen dem Empfänger:

```
<!-- Sender -> Empfaenger -->
<db:result
  to='Empfaenger'
  from='Sender'>
  98AF014EDC0...
</db:result>
```

Nachdem der Empfänger über den entsprechenden DNS-Server die Adresse des *Authoritative Servers* abgefragt hat, baut er eine Verbindung zu ihm auf. Der initialisierenden „<stream: stream>“-*Header* hat einen ähnlichen Aufbau. Nachdem die Verbindung ausgehandelt wurde, übermittelt der Empfänger die ID, die er vom Sender erhalten hat, und den erhaltenen *Dialback Key*:

```
<!-- Empfaenger -> Authoritative Server -->
<db:verify
  from='Empfaenger'
  to='Sender'
  id='457F9224A0...'>
  98AF014EDC0...
</db:verify>
```

Da der *Authoritative Server* den DNS-Rechnernamen und das Geheimnis kennen sollte, kann er die Korrektheit des *Dialback Key* bestätigen:

```
<!-- Authoritative Server -> Empfaenger -->
<db:verify
  from='Sender'
  to='Empfaenger'
  type='valid'
  id='457F9224A0...'/>
```

Stimmt der *Dialback Key* nicht überein, übermittelt der *Authoritative Server* stattdessen eine eine „<db:verify>“ -Nachricht mit *type='invalid'*.

Nun informiert der Empfänger den ursprünglichen Sender über das Ergebnis des *Servers Dialbacks*.

```
<!-- Empfaenger -> Sender -->
<db:result
  from='Empfaenger'
  to='Sender'
  type='valid'/>
```

10.2.4. Zusammenspiel der Sicherheitsmechanismen

Es ist zwischen einer Kommunikation zwischen Serverinstanzen und zwischen einer Server- und einer Klienteninstanz zu unterscheiden. Beide Verbindungsarten können, wie in Tabelle 2 zusammengefasst, verschiedene der vorgestellten Sicherheitsmechanismen nutzen.

	Serverinstanz	**Klienteninstanz**
Serverinstanz	*Server Dialbacks* und SASL	SASL
Klienteninstanz	SASL	PGP und S/MIME

Tabelle 2: *Jabber* - Gegenüberstellung der Sicherheitsmechanismen

Bei einer Kommunikation zwischen einer Klienten- und einer Serverinstanz besitzt in der Regel die Klienteninstanz ein Passwort, mit dem sie sich gegenüber dem Server authentisieren kann. Damit dieses Passwort nicht durch einen Angreifer abgehört werden kann, muss es geschützt übertragen werden. Hierfür wird unter der Verwendung von XMPP das Authentisierungssystem SASL eingesetzt.

SASL kann ebenfalls für eine Kommunikation zwischen Serverinstanzen eingesetzt werden. Dies setzt aber voraus, dass die beteiligten Server die Passwörter der anderen Server kennen. Für die Kommunikation zwischen bekannten Servern bietet SASL eine sichere Authentisierung.

Sehr oft muss aber auch eine Verbindung zu einem unbekannten Server aufgebaut werden, um einen Gesprächspartner zu kontaktieren, der diesen Server nutzt. In diesem Fall scheidet die Authentisierung über SASL aus, da eine sichere Verteilung aller Passwörter nur sehr schwer umzusetzen ist.

Die einfachste Möglichkeit eines Angriffs wäre DNS-Spoofing (Siehe Anhang B.17.4), bei den sich der Angreifer für einen anderen Server ausgibt. Da in der Regel eine Verbindung nach dem Hostteil einer JID, der einen DNS-Rechnernamen präsentiert, aufgebaut wird, könnte sich der Angreifer direkt zwischen die beteiligten Server schalten. Dies kann aber durch *Server Dialbacks* verhindert werden, da hier die Zuordnung des DNS-Namens bestätigt werden muss.

Bei beiden Verbindungsarten (Klient-Server und Server-Server) müssen die übertragenen Informationen vor *Sniffen* (Siehe Anhang B.16) geschützt werden. Diesen Schutz bietet die Nutzung von TLS, bei dem die beteiligten Server ein durch eine vertrauenswürdige Instanz unterschriebenes Zertifikat besitzen müssen.

Die in diesem Abschnitt nicht beleuchteten Schutzmechanismen zwischen den Endgeräten wurden auf Seite 54 vorgestellt.

10.3. Mögliche Angriffe

10.3.1. Bedrohung

Für eine Kommunikation über *Jabber* werden drei Arten von TCP-Verbindungen benötigt:

- Kommunikation der einzelnen Bestandteile des Servers untereinander
- Verbindungen zwischen Klienten- und der Serverinstanzen
- Verbindungen zwischen Servern untereinander

Die im Folgenden vorgestellten TCP-Verbindungen sind besonders angreifbar, wenn sie über unsichere Netze laufen. Die hier beschriebenen Bedrohungen können aber durch den Einsatz einer Verschlüsselung von den Endpunkten aus (Beispielsweise PGP, siehe Kapitel B.11) minimiert werden.

Kommunikation der Serverinstanzen Befinden sich die einzelnen Bestandteile eines *Jabber*-Servers nicht auf einem Rechner, so können die Verbindungen zwischen ihnen Ziel eines Angriffs sein. Schutz davor bieten ein entsprechendes Netzdesign und eine sicherheitskritische Konfiguration des Servers.

Die Bestandteile sollten sich in einem Netz befinden, das durch Paketfilterregeln zusätzlich geschützt wird. Auch der Einsatz eines VPNs kann die Sicherheit erhöhen. Zusätzlich sollte die Konfiguration des Servers die Verwendung von Passwörtern und TLS ermöglichen.

Angriffe auf diese Kommunikation können damit nur durch eine unsachgemäße Konfiguration ermöglicht werden. Diese soll für die weitere Betrachtung ausgeschlossen werden.

Kommunikation zwischen Klient und Server Soll im lokalen Netz kein selbständiger Server betrieben werden, muss ein externer genutzt werden. Die Verbindungen über das öffentliche Netz können dabei angegriffen werden.

Durch die Verwendung von TLS und SASL können die Kommunikation und die Passwörter geschützt werden. Wurde eine TLS-Verbindung erfolgreich aufgebaut, so kann ein Angreifer diese nur schwer kompromittieren.

Die einzige Möglichkeit für einen Angriff besteht beim Verbindungsaufbau. Gelingt es dem Angreifer, dem Benutzer gefälschte Zertifikate vorzutäuschen, so kann er sich durch eine *Man-In-The-Middle-Attack* zwischen Server und Benutzer schalten.

Dieser Angriff, der in Verbindung mit SSL im Kapitel 6.3 vorgestellt wurde, ist jedoch nicht *Jabber*-spezifisch. Da bei den Implementierungen von TLS und SASL ebenfalls keine Schwachstellen bekannt sind, kann davon ausgegangen werden, dass ein Angriff mit ausschließlich technischen Maßnahmen erfolglos bleibt.

Kommunikation zwischen Servern Die Verbindungen zwischen *Jabber*-Servern sollten grundsätzlich mit TLS (siehe Anhang B.19) geschützt werden. Da dieses Verschlüsselungsverfahren als sicher gilt, sollte ein Angriff nicht zum Erfolg führen.

Um es nutzen zu können, müssen sich die Server gegenseitig authentisieren, wenn die Benutzer der Server miteinander kommunizieren wollen. Da durch die Vielzahl von Servern eine Passwortverteilung problematisch ist, entfällt die Authentisierung über SASL (Siehe Kapitel 10.2.2).

Auch die von der Authentisierung einer Klienteninstanz an einen Server bekannte Lösung mit Zertifikaten kann nicht verwendet werden. Dazu müssten entweder alle Zertifikate der anderen Server auf jedem Server hinterlegt werden, alle von einer vertrauenswürdigen Instanz unterschrieben worden sein oder der Benutzer müsste bei der aufzubauenden Verbindung zwischen den Servern das Zertifikat manuell überprüfen.

Die Verteilung aller Zertifikate an jeden Server ist genauso problematisch wie die Hinterlegung aller Passworte. Alle Betreiber von *Jabber*-Servern werden sich nur schwer auf eine vertrauenswürdige Instanz einigen können.

Auf den ersten Blick wäre eine manuelle Überprüfung des verwendeten Zertifikats durch den Benutzer denkbar. Da die so bestätigte TCP-Verbindung aber auch für folgende Kommunikationen anderer Benutzer verwendet wird, stellt dies keine sichere Lösung da. Ein Angreifer könnte das Zertifikat selbst bestätigen.

Die hier vorgestellten Probleme sollen durch* Server Dialbacks *(Siehe Kapitel 10.2.3) gelöst werden. Wie der für diese Diplomarbeit entwickelte, im Kapitel 6.4 zu findende Angriff auf* Server Dialbacks *zeigt, ist dieser Ansatz auch nicht als sicher anzusehen. Es ist möglich, dass sich ein Server an dem anzugreifenden System anmeldet und sich für einen Anderen ausgibt. Geschieht dies analog an einem weiteren Server, könnte ein Angreifer durch eine* Man-In-The-Middle-Attack *den verschlüsselten Datenstrom mitlesen.

10.3.2. Schutzmechanismen

Die Analyse von *Jabber* lässt die Aussage zu, dass die Entwickler viel Energie in die Erhöhung der Sicherheit gesteckt haben. Die Kombination von TLS und SASL beziehungsweise TLS und *Server Dialbacks* bietet nur sehr wenige Ansätze für einen Angriff.

Der in dieser Diplomarbeit vorgestellte Angriff auf *Server Dialbacks* kann aber nur sehr schwer verhindert werden. Eine der wenigen Möglichkeiten wäre eine passwortgestützte Authentisierung, bei der aber alle Passwörter an alle Server verteilt werden müssten.

Eine weitere Art sich vor diesem Angriff zu schützen ist der Einsatz einer *End-To-End*-Klientenverschlüsselung. Als einziges vorgestelltes *Instant Messaging*-System setzt *Jabber* auf eine PGP-Verschlüsselung. Alle verbreiteten Klienten unterstützten dieses asymmetrische Verschlüsselungsverfahren.

Durch den Einsatz dieses Verfahren treten nur die bekannten Probleme der asymmetrischen Verschlüsselung auf. Ein Beispiel ist die Manipulation des öffentlichen Schlüssels während der Übertragung.

Obwohl ebenfalls eine standardisierte XMPP-Lösung für eine *End-To-End*-Verschlüsselung zwischen Klienteninstanzen veröffentlicht wurde, wird diese in der Praxis nicht eingesetzt.

10.4. Maßnahmen zur Verhinderung der unautorisierten Nutzung in Organisationen

Reine *Jabber*-Verbindungen von Klienten zu externen Servern können durch einen Paketfilter verhindert werden, der alle Verbindungen mit den TCP-Portnummern 5222 (XMPP und ungeschützte Verbindungen des klassischen Protokolls) und 5223 (Verschlüsselte Verbindungen des klassischen Protokolls) blockiert. Es muss berücksichtigt werden, dass auf diese Weise Verbindungen mit *Jabber*-Servern, die auf anderen Ports lauschen, oder die Nutzung webbasierter Klienteninstanzen nicht unterbunden werden kann.

Soll ein interner Server eingesetzt werden, der nicht mit externen kommunizieren darf, muss durch einen entsprechenden Paketfilter die Kommunikation zu den TCP-Ports 5269 und 5270

verhindert werden. Auch wenn kein zentraler Server vorgesehen ist, sollten ein- und ausgehende Verbindungen zu diesen TCP-Ports unterbunden werden. Sonst könnten die Benutzer im Netz einen eigenen, lokalen Server aufsetzen und mit anderen *Jabber*-Servern kommunizieren.

Um einen öffentlichen *Jabber*-Server zu betreiben, auf den sowohl von dem internen als auch von dem öffentlichen Netz zugegriffen werden darf, sollte dieser in der demilitarisierten Zone betrieben werden. Durch entsprechende Paketfilterregeln auf den angrenzenden Netz-Routern kann ein Missbrauch erschwert werden.

Hierfür ist zu betrachten, ob die Benutzer im lokalen Netz andere *Instant Messaging*-Server kontaktieren dürfen. Zusätzlich muss noch berücksichtigt werden, ob sie andere *Instant Messaging*-Systeme über den in der demilitarisierten Zone bereitgestellten Server nutzen dürfen.

Es dürfen nur Verbindungen zu dem *Instant Messaging*-System, dessen Verwendung gestattet wird, von dem zentralen *Jabber*-Server aufgebaut werden. Alle Verbindungen zu anderem *Instant Messaging*-Servern sollten dann verhindert werden. Die dazu notwendigen Schritte wurden zu jedem *Instant Messaging*-System skizziert, das in dieser Diplomarbeit vorgestellt wurde.

10.5. Fazit

Die Nutzung von *Jabber* bietet sowohl für einen privaten als auch einen gewerblichen Einsatz mehr Vor- als Nachteile. Gerade für Organisationen ist die Option, einen zentralen Server im lokalen Netz zu betreiben, für die Auswahl von *Jabber* als *Instant Messaging*-System entscheidend. Dadurch können sensible Informationen über *Instant Messaging* versendet werden, ohne dass diese das abgesicherte Netz verlassen müssen.

Die Wahl des öffentlichen Servers kann die Sicherheit beeinflussen. Um die Angriffspunkte auf die Kommunikation zu verringern, kann ein geographisch naher Server (zum Beispiel der des verwendeten *Internet Service Providers*) genutzt werden.

Die Nutzung des Klartextprotokolls, das die Informationen in einem standardisierten XML-Dialekt überträgt, ermöglicht den Einsatz auf einer Vielzahl von Systemen. Die Kodierungsvorschrift UTF-8 verhindert dabei Probleme bei der Kommunikation zwischen Klienten mit verschiedenen Zeichensätzen.

Die hohen Sicherheitsanforderungen, die bei der Entwicklung von *Jabber* umgesetzt wurden, machen *Jabber* zu dem sichersten *Instant Messaging*-System, das in dieser Diplomarbeit vorgestellt wird. Hiervon profitieren neben gewerblichen auch private Benutzer.

Für Privatanwender wird die geringe Verbreitung von *Jabber* zu dem größten Defizit gegenüber anderen Systemen. *Jabber* ist am wenigsten verbreitet und den Serverbetreibern stehen meist keine finanziellen Mittel für Marketing zur Verfügung. Und der Erfolg eines *Instant Messaging*-Systems ist direkt von der Benutzeranzahl abhängig.

Um diesen Nachteil auszugleichen, können weitere *Instant Messaging*-Systeme integriert werden. Bei keinem anderen IM-System ist es möglich, mit nur einer TCP-Verbindung zwischen Klienten- und Serverinstanz die Benutzer mehrerer IM-Systeme zu kontaktieren. Im allgemeinen bietet somit die *Jabber*-Architektur nur Vorteile gegenüber anderen Systemen.

Dem lokalen Netzbetreiber stehen eine Vielzahl von Möglichkeiten zur Verfügung, die Kommunikation über dieses *Instant Messaging*-System zu beeinflussen. Durch entsprechende Paketfilterregeln kann *Jabber* unterdrückt oder durch den Einsatz eines lokalen Servers kontrolliert werden.

11. IRC

Das älteste Protokoll, das in dieser Diplomarbeit vorgestellt wird, ist das *Internet Relay Chat* (IRC)-Protokoll. IRC ist das am meisten verbreitete Konferenzsystem im Internet. Es wurde von Jarkko Oikarinen entwickelt und am Anfang unter dem Namen *Relay Chat* im *BITNET* eingesetzt. 1988 wurde das offene Protokoll für der Verwendung im Internet übernommen.

Durch die einfache Handhabung besitzt IRC ein hohes Suchtpotential. Auf Grund der weiten Verbreitung und der dadurch bedingten großen Anzahl von Benutzern bilden sich schnell Interessengruppen, die sich in unterschiedlichen Channels (siehe Kapitel 12.1.2) zusammenfinden.

Im Gegensatz zu *Instant Messaging*-Systemen kann über IRC auch mit fremden Personen kommuniziert werden. Es bietet die Möglichkeit, im Internet soziale Kontakte zu pflegen.

Durch den hohen Bekanntheitsgrad von IRC und der Vielzahl verfügbarer Software (sowohl Klienten als auch Server) wird IRC auch sehr oft für einen lokalen, temporären Einsatz verwendet. Beispielsweise ist es üblich, dass während einer LAN-Party (Computerspieler treffen mit ihren Computern an einem Ort zusammen) die gesamte Kommunikation über einen IRC-Server stattfindet.

Das Zusammentreffen vieler, verschiedener Charaktere ist aber auch mit Nachteilen verbunden. So erleichtert IRC illegale Aktivitäten wie zum Beispiel den Austausch von pädophilen Materialien oder rechtsradikalen Gedankengut.

Die erste RFC zu IRC wurde im Jahre 1993 unter der Nummer 1459 veröffentlicht. 2000 wurde eine überarbeitete Spezifikation in den RFCs mit den Nummern 2810 (*Internet Relay Chat Protocol*), 2811 (*Internet Relay Chat: Channel Management*), 2812 (*Internet Relay Chat: Client Protocol* und RFC 2813 (*Internet Relay Chat: Server Protocol*) veröffentlicht. Alle RFCs sind unter [16] zu finden.

Da das IRC-Protokoll keinen Schutz der Integrität und Vertraulichkeit der übertragenen Informationen bietet, wird auf dieses Protokoll nur kurz eingegangen. Der interessierte Leser findet in den RFCs zu IRC viele weitere Informationen zu dem Protokoll.

Im Laufe dieses Kapitels werden die Bestandteile eines IRC-Netzes und Beispiele für Angriffe vorgestellt. Zusätzlich wird auf Möglichkeiten eingegangen, wie IRC nach den Aspekten der IT-Sicherheit (Verfügbarkeit, Integrität und Vertraulichkeit) abgesichert werden kann.

11.1. Architektur

11.1.1. Klienten

Der Klient hat die Aufgabe, die Kommunikation mit dem Server zu realisieren. Da IRC sehr weit verbreitet ist, gibt es Klienten in allen möglichen Variationen. Für jedes Betriebssystem, das von einem größeren Benutzerkreis verwendet wird, existieren Klienten. Zusätzlich hat der Benutzer die Wahl zwischen graphischen oder textbasierten Klienten. Sowohl Klienten, die in einem Web-Browser laufen, als auch graphische, bei denen die Mitteilungen durch Sprechblasen wie in einem Comic dargestellt werden, haben einen Benutzerkreis gefunden.

Die Benutzer werden an Hand von eindeutigen *Nicknames* identifiziert. Hier entsteht ein gewaltiges Problem, auf das detailliert im Abschnitt 11.2.1 und im Anhang B.7 eingegangen wird: *Nickname Wars*.

Bot Äußerlich kann ein Bot nur schwer von einem Klienten, an dem ein Mensch sitzt, unterschieden werden. Ein Bot ist aber vielmehr ein automatisierter Klient. Der Begriff Bot wird von „Robot“ abgeleitet, das vermutlich wiederum von dem tschechischen Wort für Arbeit „robota“ stammt.

Es gibt ein großes Einsatzgebiet für Bots. Wenn zum Beispiel ein Benutzer einen Channel gründet und ihn wieder verlässt, nachdem sich andere Benutzer angemeldet haben, kann der Benutzer seine vorher erworbenen Privilegien im Channel von sich aus nicht mehr erlangen. Hat er vor dem Verlassen einem anderen Benutzer diesen privilegierten Status gegeben, so kann er diesen nur mit dessen Hilfe wieder erlangen.

Um von anderen Benutzern unabhängig zu sein, könnte dieser Benutzer einen Bot in den Channel stellen. Wenn er nun dem Bot diese Privilegien gibt, kann er den Bot so programmieren, dass der Benutzer beim Eintreten in den Channel automatisch seine Rechte zurückerhält.

Bots können aber auch mit anderen Funktionalitäten belegt werden. So sind Bots verbreitet, die zu einer bestimmten Uhrzeit Informationen wie zum Beispiel den Sonnenaufgang ausgeben oder die nach Anfrage den Kantinenspeiseplan in den Channel schreiben.

Besitzt ein Bot Privilegien im Channel, so wird er für Angreifer attraktiver. Ein Bot kann aber auch zum Angriff auf andere Benutzer missbraucht werden, indem er beispielsweise Viren und Würmer verbreitet.

Ein Programm, um einen Bot zu erzeugen, ist die Software *Eggdrop*[14].

11.1.2. Server

Der Server hat die Aufgabe, Verbindungen von Klienteninstanzen anzunehmen und damit die Kommunikation dieser in Channels zu ermöglichen. Dazu muss der Server an einem TCP-Port lauschen. Hierfür wurde die offizielle Portnummer 194 und die inoffizielle Portnummer 6667 reserviert. Zusätzlich können mehrere Server untereinander verbunden werden, um zusammen zu arbeiten.

Ein Vorteil von IRC ist es, dass dem Betreiber eine große Auswahl an Serversoftware zur Verfügung steht. Für die weiter verbreiteten Betriebssysteme existieren sogar verschiedene Implementierungen der Serversoftware. Die Administration eines Servers ist dabei auch für weniger geübte Benutzer keine große Hürde.

11.1.3. Aufbau eines Netzes

Um ein umfangreiches, geographisches Gebiet abzudecken und um die Last aufzuteilen, besteht unter IRC die Möglichkeit, mehrere Server zu verbinden. Diese bilden ein Netz, in dem Informationen untereinander ausgetauscht werden können.

Neben der möglichen Aufteilung innerhalb eines LANs wurde diese Aufteilung im Internet schon sehr früh vorgenommen. Da dort aber die Netze schnell unübersichtlich und beinahe unverwaltbar wurden, existieren im Internet verschiedene, unabhängige Netze. So wurden die Netze nach Sprachen und geographischen Schwerpunkten aufgeteilt. Damit wird auch eine Untergliederung nach kulturellen Gesichtspunkten erreicht. Beispielsweise wäre in einem arabischen Netz die Toleranz gegenüber sexuellen Inhalten sehr gering.

[14] **Verweis auf Applikation:** http://www.eggheads.org

Beispiele für die großen Netze sind QuakeNet, EFnet, IRCNet und Undernet, wobei das IRCNet in Europa am verbreitetsten ist. Um die Bestandteile des IRCNet zu verdeutlichen, lohnt sich ein Blick auf die IRC-Landkarten für Deutschland[15] oder für andere geographische Gebiete[16].

11.2. Mögliche Angriffe

11.2.1. Bedrohung

Der im Kapitel 6.1 beschriebene Angriff auf ungeschützte Verbindungen lässt sich auch für IRC leicht durchführen. Dies betrifft sowohl Angriffe auf die Verfügbarkeit und Integrität als auch auf die Vertraulichkeit. Im Folgenden werden weitere Aspekte vorgestellt.

Angriffe auf Verfügbarkeit Bis auf wenige Ausnahmen werden die öffentlichen IRC-Netze für einen privaten Informationsaustausch genutzt. Da der zu erwartende finanzielle Schaden bei einem Ausfall eher gering ist, stellt sich die Frage, welche Motivation ein Angreifer neben den üblichen Punkten wie Anerkennung verleitet.

Sehr oft ist der Zusammenbruch eines IRC-Netzes oder einzelner Bestandteile dessen nicht das primäre Ziel des Angreifers. Der gewöhnliche Angreifer, der nur ein geringes Hintergrundwissen besitzt, sieht seine Vorteile ganz woanders: Wie schon erwähnt, ist ein Problem im IRC die Identifikation durch eindeutige *Nicknames* und die Bestrebung, in einem Channel über den „gewöhnlichen“ Benutzer zu stehen. Dies kann im Channel durch den Besitz übergeordneter Privilegien geschehen, die den Angreifer als Gott fühlen lassen, da er die Benutzer beliebig aus dem Channel entfernen (*kicken*) kann.

Die Übernahme eines *Nicknames* durch einen Angreifer hat meistens den Grund, dass der von einem anderen benutzte *Nickname* aus der Sicht des Angreifers entweder besser zum eigenen Namen passt oder aus anderen Gründen besser gefällt. Um an einen schon belegten *Nickname* zu gelangen, muss der Angreifer dafür sorgen, dass die Verbindung von Benutzer mit dem gewünschten *Nickname* zu dem Server unterbrochen wird. Das hat den Hintergrund, dass der *Nickname* nur von einem angemeldeten Benutzer belegt werden kann. Wird die Verbindung zum Benutzer beendet, verliert dieser den Anspruch auf den *Nickname*.

Diese Problematik des *Nickname War* wird noch einmal im Anhang B.7 diskutiert. Im Weiteren ist zu beachten, dass in einigen Serverimplementationen der *Nickname* noch für eine kurze Zeitspanne von einigen Minuten reserviert bleibt. Unter Umständen muss dann der Angreifer verhindern, dass das Opfer den *Nickname* wieder zurück erlangen kann.

Die zweite Problematik ist das *Channel Hijacking*. Wie weiter oben erwähnt, sind hier übergeordnete Privilegien im Channel das Ziel des Angriffs. Um diesen Angriff zu verstehen, muss bekannt sein, dass ein Channel bei dem Eintritt des ersten Benutzers erstellt und beim Verlassen des letzten Benutzers beseitigt wird. Der erste Benutzer bekommt dabei übergeordnete Privilegien, die er nach Bedarf an andere Benutzer weitergeben kann.

Möchte nun ein Angreifer den Channel übernehmen, muss er einen Weg finden, dass alle Benutzer den Channel verlassen. Dies kann er zum Beispiel durch einen *Net Split* realisieren. Hierbei stört er die Verbindung eines Servers zu dem IRC-Netz und erstellt auf diesem einen neuen

[15] **Vertiefung:** AIP - Die *.de IRCnet-Server-Map (http://irc.fu-berlin.de/ircmap.html)
[16] **Vertiefung:** IRCnet - Maps (http://irc.leo.org/ircmap)

Channel. Dabei erhält der Angreifer den gewünschten Operator-Status. Nachdem der gesplittete Server nun wieder den Kontakt zum restlichen IRC-Netz aufbaut, werden die beiden doppelten Channel (einer auf dem gesplitteten Server und der Alte im Netz) zu einem zusammengefügt. Nun muss der Angreifer dafür sorgen, dass er mit seinem Operatorstatus die Privilegien der vorhanden Operatoren entfernt und somit die alleinige Kontrolle über den Channel erhält.

Anstatt den üblichen, netzbasierten *Denial of Service*-Angriffen wurde an dieser Stelle auf die Problematik der *Nicknames* und Channels eingegangen. Es ist aber zu berücksichtigen, dass beispielsweise Fehler in der Server- oder Klientensoftware ebenfalls für effiziente Angriffe missbraucht werden können. Auf diese Fehler, die durch eine einfache Softwareaktualisierung behoben werden können, wird in dieser Diplomarbeit nicht tiefer eingegangen.

Angriffe auf Vertraulichkeit und Integrität Ein gravierender Nachteil des *Internet Relay Chat*-Protokolls ist, dass keine Schutzmaßnahmen während der Planung in das Protokoll übernommen wurden. Da die gesamte Kommunikation im Klartext über definierte TCP-Ports abläuft, kann ein Angreifer alle Angriffsarten anwenden, die im Kapitel 3 vorgestellt wurden.

Neben dem Angriff auf die Verfügbarkeit gilt dies besonders für den nicht vorhandenen Schutz der Vertraulichkeit (siehe Kapitel 3.2) und der Integrität (siehe Kapitel 3.3). Ein Angreifer kann durch *Sniffen* (siehe Anhang B.16) oder durch *Spoofen* (siehe Anhang B.17) an sein Ziel gelangen.

Besitzt der Angreifer lokalen Zugriff auf das System des Benutzers, kann die Klientensoftware ebenfalls das Ziel eines Angriffs sein (siehe Kapitel 3.4).

11.2.2. Schutzmechanismen

Die sicherheitsrelevanten Defizite, die IRC mit sich bringt, sind allgemein bekannt und daher haben sich einige Entwickler mit diesem Problem befasst. Es gibt zwei mögliche Ansätze, die im Folgenden vorgestellt werden.

IRC-SSL Es existieren einige Bemühungen, IRC über das *Secure Sockets Layer* (SSL) - Protokoll zu schützen. Die Nutzung von SSL (siehe Anhang B.15) ist mit folgenden Problemen verbunden:

1. Ein Großteil der bestehenden IRC-Server ist jetzt schon sehr stark ausgelastet. Da SSL hohe Kapazitäten erfordert, würden die bisherigen Server unter der Last zusammenbrechen.

2. Prinzipiell verschlüsselt SSL nur zwischen den jeweiligen, direkten Kommunikationseinheiten wie Klient-Server oder Server-Server. Ein Angreifer, der die Kontrolle über den Server erhalten hat oder ein neugieriger Serverbetreiber könnte durch eine Änderung in der Serversoftware die Kommunikation weiterhin mitlesen.

3. Die SSL-Verschlüsselung könnte nur parallel zu IRC verwendet werden. Nutzt nur einer der Gesprächspartner kein SSL, würden alle Informationen zwischen ihm und dem Server auch weiterhin unverschlüsselt übertragen werden.

Zusatzsoftware auf dem Klienten Durch so genannte *Plugins* kann der Funktionsumfang der eingesetzten Klienteninstanz erweitert werden. Beispielsweise kann jede ausgehende Nachricht mittels PGP (siehe Anhang B.11) verschlüsselt werden.

Es ist aber jederzeit möglich, dass für die Klientensoftware ein Sicherheitsupdate nötig wird. Da das *Plugin* ein eigenständiges Programm ist, kann es sein, dass es nicht mehr mit der neuen Software zusammenarbeitet. Bemerkt der Benutzer dies nicht, versendet er Klartext, ohne dies zu bemerken.

11.3. Maßnahmen zur Verhinderung der unautorisierten Nutzung in Organisationen

Die Nutzung von IRC lässt sich sehr einfach durch entsprechende Paketfilterregeln verhindern. Da IRC-Server aber auf einer Vielzahl von TCP-Ports betrieben werden können, führt nur eine Abkopplung des lokalen Netzes vom externen Netz zum Erfolg. Durch entsprechende Proxy-Server kann den Benutzern die Nutzung bestimmter Dienste gestattet werden.

Durch die weite Verbreitung von IRC sind sehr viele Software-Bibliotheken verfügbar. Diese ermöglichen beispielsweise eine sehr einfache Implementierung eines webbasierten Klienten. Wird der Zugriff auf Webseiten gestattet, kann die Nutzung nur sehr schwer verhindert werden.

11.4. Fazit

Primär wird IRC für private Zwecke eingesetzt und die damit verbundenen Ansprüche erfüllt es perfekt. Die einfache Protokollarchitektur ermöglicht es den Entwicklern, für Endbenutzer komfortable Software zu erstellen.

Durch die weite Verbreitung und die Vielzahl von öffentlichen Servern kann sich ein Betreiber nach seinen Vorlieben einen Server auswählen. Das Prinzip der Servernetze ist aber nicht sehr effizient, da alle Informationen über Benutzer und Channel auf jedem Server vorliegen müssen. Dies erfordert eine Synchronisation, deren Informationsfluss angegriffen werden kann und die hohen Netzverkehr erzeugt.

Da IRC in der Grundeinstellung keinen Schutz bietet, ist es höchstens zur Pflege von sozialen Kontakten im Internet geeignet. Ein Großteil der im IRC-Netz verbreiteten Channels sind öffentlich, in denen jeder interessierte Mitleser eintreten kann. Daher ist ein kryptographischer Schutz hier nicht relevant.

Die Angriffe auf die Verfügbarkeit sind zwar lästig, aber einen sehr großen Schaden verursachen sie bei einem nicht kommerziellen Einsatz nicht. Es muss aber jedem Benutzer bewusst sein, dass er keine vertraulichen Daten übermitteln darf.

Anders sieht es in kommerziellen Umfeld aus. Auf Grund der im Vorfeld angesprochenen Defizite sollte unter keinen Umständen IRC verwendet werden. Es gibt zwar zahlreiche Anwendungen und Erweiterungen, die diese Probleme lösen, aber ihr Einsatz ist mit hohem Aufwand verbunden. Hier besteht jederzeit die Gefahr, dass eine der installierten Anwendungen nicht aktiviert wurde. Außerdem ist eine komplette Lösung vorzuziehen, da bei einer Aktualisierung eines Teils der andere unter Umständen nicht mehr effizient arbeitet. In einem geschlossen LAN besteht auch immer die Gefahr, dass ein neugieriger Mitarbeiter oder ein Angreifer die gesamte Kommunikation mitlesen oder stören kann.

Für einen geschäftskritischen Einsatz sollte auf Silc zurückgegriffen werden, das im folgendem Kapitel vorgestellt wird. Eine Nutzung von IRC-Servern, die auf Standardports betrieben werden, lässt sich einfach unterbinden.

12. Silc

Die vorigen Kapitel über *Instant Messaging* haben gezeigt, dass sehr viel Energie in die Entwicklung von Kommunikationssystemen gesteckt wurde. Leider wurde dabei oft die komfortable Bedienung über die Datensicherheit gestellt.

IRC, das die Vorstellung der Konferenzsysteme im vorigen Kapitel eingeleitet hat, ist aus Sicht der IT-Security-Verantwortlichen ebenfalls kritisch. Einen anderen Weg geht das *Secure Internet Live Conferencing* (Silc)-Protokoll, bei dem die Sicherheit der übertragenen Informationen im Vordergrund steht.

Das Silc-Projekt wurde 1997 von Pekka Riikonen gegründet und die erste Version eines Klienten und eines Servers wurde 2000 freigegeben. Seit dieser Zeit ist Silc immer weiter gereift und zahlreiche Silc-fähige Klienten wie Gaim und Plugins wie zu Irssi sind hinzugekommen.

Der größte Vorteil von Silc liegt in der Verschlüsselung. Jede Nachricht wird während der Übertragung kryptographisch geschützt. Dieser Schutz zielt primär auf die Integrität und Vertraulichkeit der Informationen. Dabei werden sowohl die Nutzlast als auch die Header-Informationen der Silc-Datagramme verschlüsselt.

Dem Benutzer ist es dabei nicht möglich, die Verschlüsselung zu deaktivieren. Somit müssen sich der Benutzer und der dafür zuständige IT-Security-Verantwortliche keine Gedanken darüber machen, ob die Verschlüsselung aktiviert wurde. Um die sichere Kommunikation zwischen den Benutzern komfortabel zu gestalten, kann statt eines Passworts eine Public-Key-Authentisierung verwendet werden.

Das Problem des Schlüsselaustauschs, das bei jeder symmetrischen Verschlüsselung auftritt, wird bei Silc elegant gelöst. Das dazu verwendete *Silc Key Exchange*-Protokoll wird im Kapitel 12.3.2 vorgestellt.

Die bisherige Betrachtung der IM-Systeme hat gezeigt, dass bei der Verwendung von Servern, die nicht selber administriert werden, ein Angreifer jede Kommunikation dort mitlesen könnte. Um dieses Problem zu umgehen, kann in Silc ein kryptographischer Schutz aktiviert werden, der nur von den Endpunkten (zum Beispiel Klienten) ausgeht. Diese Maßnahme erschwert eine *Man-In-The-Middle-Attack*. Kurz zusammengefasst, stellt so der sichere Informationsaustausch über unsichere Netze und Server kein Problem dar.

Silc stellt sogar Lösungen bereit, die das dritte Prinzip der IT-Sicherheit, der Verfügbarkeit, gewährleisten soll. Fällt ein Silc-System aus, das für das Weiterleiten der Nachrichten verantwortlich sind, können Backup-Systeme jederzeit die Aufgabe des ausgefallenen Systems übernehmen.

Silc ist aber nicht nur als verschlüsseltes IRC zu verstehen. Die Probleme in IRC-Netzen, wie *Nickname-* und *Channel Wars*, wurden in Silc komplett gelöst. Ein *Nickname* ist unter Silc nicht eindeutig. Und durch die Eigenschaft der Silc-Channels, dass dieser nach dem Schließen nicht sofort gelöscht und der Gründer des Channels seine übergeordnete Channel-Rechte ohne fremde Hilfe zurück erlangen kann, werden die IRC-Probleme im Keim erstickt.

Über das Silc-Protokoll können nicht nur Textnachrichten versendet werden. Um Dateien sicher vom Sender zum Empfänger zu übertragen, setzt Silc auf *SFTP*. Bei *SFTP* handelt es sich um die verschlüsselte Form des *File Transfer Protocols*. Zusätzlich können Nachrichten im MIME-Format übertragen werden. Damit wird die Übertragung von Videos, Bildern und Audionachrichten möglich.

Da Silc sehr komplex ist, kann im Rahmen dieser Diplomarbeit nicht das gesamte Protokoll analysiert werden. Daher werden neben den Grundlagen nur einige Teilgebiete im Anhang A.5 näher erläutert. Als Informationsquelle diente die detaillierte Spezifikation von Silc (siehe [17]), welche zur Zeit bei der IEFT zur Standardisierung erneut eingereicht wurde (Stand: Mai 2005).

Soweit dies möglich war, wurde diese Quelle durch eine Analyse der Datagramme bestätigt. Da bei Silc der Großteil der Kommunikation jedoch verschlüsselt wird, war dies nicht so immer möglich.

Terminologie In dieser Diplomarbeit wird der Begriff **Silc** für verschiedene Abstraktionsebenen benutzt. Er bezieht sich in erster Linie auf das Protokoll, aber auch auf das hinter Silc stehende Prinzip.

Die Begriffe **Schlüssel** beziehungsweise **Key** werden gleichbedeutend verwendet. Generell ist der Schlüssel das Geheimnis, das für eine Verschlüsselung notwendig ist. Der Begriff des Schlüssels wird im Anhang B.14 detailliert beleuchtet.

Das Silc-Protokoll (siehe Anhang A.5) verwendet das *Transmission Control Protocol* (TCP) für den Datenaustausch. Die Nutzlast, die innerhalb eines TCP/IP-Datagramms versendet wird, besitzt in Silc eine einheitliche Struktur und wird im Folgenden als **Silc-Datagramm** bezeichnet. Jedes Silc-Datagramm hat einen individuellen Bezeichner (**SILC_PACKET_<paketname>**), der von der Funktion des Datagramms abgeleitet ist.

12.1. Architektur des Systems

12.1.1. Klienten

Die Klienteninstanz ist als Verbindungselement zwischen der Silc-Serverinstanz und dem Benutzer zu sehen. Der Benutzer kann damit Silc-Befehle zur Serverinstanz schicken sowie Nachrichten lesen und schreiben.

Zur Zeit existiert im Gegensatz zu IRC nur eine geringe Auswahl an Klienten. Aber ob graphisch oder textbasiert, für jede Vorliebe des Benutzers ist etwas dabei. In der Vergangenheit sind keine nennenswerten Softwarefehler veröffentlicht wurden, so dass davon auszugehen ist, dass die Klienteninstanzen ihre anfängliche Testphase abgeschlossen haben.

Um Nachrichten zu empfangen wird wie bei IRC eine Art Identifikator benötigt. Dieser wird im Silc-Jargon ebenfalls als *Nickname* bezeichnet. Im Gegensatz zu der Umsetzung von IRC bietet Silc einen gravierenden Vorteil: Die *Nicknames* alleine sind nicht mehr eindeutig und können so an mehrere Benutzer vergeben werden. Damit sind Probleme wie *Nickname Wars* (siehe Anhang B.7) ausgeschlossen.

Die eindeutige Identifikation der Silc-Benutzer ist nur über die *Client ID* möglich. Die *Client ID* besteht aus dem *md5*-Hashwert des *Nickname*, aus der IP-Adresse des Rechners des Benutzers und einer unter Umständen zufällig gewählten Zahl. Diese ermöglicht eine Anmeldung von

mehreren gleichen *Nicknames* an einer Serverinstanz von einem Rechner aus. Der *Nickname* kann 128 Zeichen lang sein.

Das Format der *Client ID* ist von der IP-Version abhängig. Bei IPv4 besteht die *Client ID* aus 32 Bit für die IP-Adresse, 8 Bit für die Zufallszahl und 88 Bit für den Hashwert über den *Nickname*. Bei IPv6 werden 128 Bit für die IP-Adresse reserviert. Dies ergibt eine Gesamtgröße von 128 Bit für die IPv4-*Client ID* und 224 Bit für die IPv6-*Client ID*.

12.1.2. Server

Die Silc-Server sind die wichtigsten Bestandteile eines Silc-Netzes. Von einer Serverinstanz wird die benötigte Infrastruktur (zum Beispiel Channel, siehe Seite 79) für die Kommunikation zwischen den angemeldeten Klienteninstanzen bereitgestellt und verwaltet. Zur Kommunikation mit anderen Serverinstanzen werden die zwischen die Server geschalteten Silc-Router verwendet.

Jede Serverinstanz ist höchstens an einem Router angeschlossen. Wenn der Server nur lokal operieren und nicht mit anderen Servern kommunizieren soll, ist der Anschluss an einen Router nicht erforderlich.

Zusammenfassend ist zu sagen, dass der Server nur lokale Informationen erfasst und alle weiteren Anfragen an den nächsten Router weiterleitet. Um dies zu realisieren, wird der Serverinstanz eine *Server ID* zugewiesen. Die Größe der *Server ID* ist abhängig vom verwendeten Internet-Protokoll und besteht aus der IP-Adresse (IPv4: 32 Bit, IPv6 : 128 Bit), der Portnummer (jeweils 16 Bit) und einer Zufallszahl (16 Bit). Die Zufallsnummer besitzt zur Zeit noch keine Bedeutung, sie wurde aber aufgenommen, um das Format der *Server ID* und das Format der *Client ID* einheitlich zu gestalten. Damit umfasst die *Server ID* eine Gesamtgröße von 64 Bit bei IPv4 und 160 Bit bei IPv6.

Um Verbindungsanfragen von den Klienteninstanzen annehmen zu können, wird die Serverinstanz an einen TCP-Port gebunden. Bei der IANA[17]wurde für das Silc-Protokoll der Port 706 reserviert.

Zur Bereitstellung der Kommunikationsinfrastruktur muss die Serverinstanz bestimmte Informationen führen. Diese Informationen werden jeweils in Listen für angeschlossene Router, Klienten und Channel verwaltet.

Die **server list** beinhalten Informationen über den nächsten Router. Der irreführende Name *server list* wurde gewählt, da die im Router geführte Liste dasselbe Format besitzt und zwei verschiedene Namen für die gleiche Datenstruktur nicht sinnvoll wären.

In der *server list* werden die Namen und IP-Adressen der nächsten Router, deren *Server IDs* und die für den kryptographischen Schutz benötigten Schlüssel abgelegt. Dies umfasst sowohl den symmetrischen als auch den öffentlichen Schlüssel. Auf einem normalen Server ohne Routing-Funktionalität sollte diese Liste sehr kurz ausfallen, da nur ein Router direkt an dem Server angeschlossen sein sollte.

Die an der Serverinstanz angemeldeten Klienteninstanzen werden ebenfalls in einer Tabelle verwaltet. Hierfür ist die **client list** vorgesehen, die die zu den Benutzern zugehörigen Informationen enthält. Neben dem symmetrischen und öffentlichen Schlüssel werden der *Nickname* und

[17] **Vertiefung:** IANA - http://www.iana.org/assignments/port-numbers

die *Client ID* der jeweiligen Klienteninstanzen in der *client list* abgelegt. Um die Identifikation der Benutzer durch andere Benutzer zu vereinfachen, beinhaltet die *client list* Einträge zu dem kompletten Namen des Benutzers, von welchem Rechner er sich angemeldet hat und welchen Benutzernamen er auf dem System besitzt, von dem er sich angemeldet hat.

Wenn sich mehrere Benutzer mit dem gleichen Benutzernamen im Silc-Netz aufhalten, kann ein möglicher Gesprächspartner durch eine Silc-*whois*-Anfrage anhand dieser Informationen den gewünschten Gesprächspartner finden.

Zuletzt müssen alle Channels, die auf dem Server verwaltet werden, in einer Liste geführt werden. Hierfür ist die **channel list** vorgesehen, die für jeden Channel einen Eintrag bestehend aus dem Namen des Channels, der *channel ID*, dem *channel key* und der im Channel angemeldeten Benutzer enthält. Die Benutzer werden an Hand ihrer *Client ID* identifiziert. Außerdem enthält die *channel list* für jeden Benutzer die Rechte, die er im Channel besitzt.

Channel Wie auch bei IRC ist ein Channel ein Zusammenschluss mehrerer Benutzer, die aus bestimmten Gründen untereinander kommunizieren wollen oder müssen. Dies kann das Interesse an einem bestimmten Themengebiet oder die Zugehörigkeit zu einer Arbeitsgruppe sein. Wenn eines der Mitglieder dieses Channels eine Nachricht an diesen sendet, können alle weiteren Mitglieder diese Nachricht lesen. Betritt ein Mitglied als erstes den Channel, wird dieser erstellt. Nach dem Verlassen des letzten Mitglieds wird dieser Channel gelöscht. Der erste Benutzer, der den Channel erstellt, bekommt einen besonderen Status zugewiesen.

Im Vergleich zu IRC hat ein Channel unter Silc einen größeren Funktionsumfang. Wie auch bei IRC gibt es bei Silc spezielle Administratoren, die den Channel pflegen. Diese werden als Operator bezeichnet und können beispielsweise störende Benutzer aus dem Channel entfernen. Zusätzlich existieren unter Silc Channel-**Founder**. Dies sind Operatoren mit erweiterten Rechten.

Die Channel-*Founder* können, anders als beim IRC, diesen Status selbstständig bei jedem Eintritt in den Channel erhalten. Des Weiteren können sie permanente Channel erstellen. Dies bedeutet, dass der Channel auch nach dem Verlassen des letzten Mitgliedes bestehen bleibt und der erste zurückkehrende Benutzer nicht notwendigerweise einen Operatoren-Status erlangt. Durch diese beiden Punkte ist die Problematik der Channelübernahme, die im IRC-Kapitel 11.2.1 beschrieben wurde, unter Silc nicht mehr gegeben.

Um einen solchen Angriff dennoch realisieren zu können, müsste ein beliebiger Operator im Channel dem *Founder* die Rechte aberkennen. Um dies zu verhindern, wurde von den Silc-Entwicklern der *Founder*-Status als permanentes Recht ausgelegt.

Durch die Angabe des eindeutigen Channelnamens, über den die *Channel ID* bestimmt wird, können die Benutzer diesen Channel betreten. Channelnamen können maximal 256 Zeichen lang sein und werden von den Servern verwaltet, auf denen sie erstellt wurden.

Jeder Channelname ist eindeutig und die dazugehörige *Channel ID* ist abhängig von der verwendeten IP-Version. Bei IPv4 beinhaltet die *Channel ID* die 32 Bit große IP-Adresse. Bei IPv6 werden hierfür 128 Bit reserviert. Bei beiden Protokollen werden in der *Channel ID* jeweils 16 Bit für die Portnummer des Servers und für eine Zufallszahl reserviert. Dies ergibt eine *Channel ID*-Größe von 64 Bit bei IPv4 und 160 Bit für IPv6.

Sämtliche Kommunikation im Channel wird durch den *Channel Key* (siehe Kapitel 12.3.3)

kryptographisch geschützt. Wird der Server erfolgreich angegriffen, so ist dieser Schutz jedoch wertlos. Daher besteht zusätzlich die Möglichkeit, die Kommunikation von den Endpunkten aus, also den Klienteninstanzen, ebenfalls kryptographisch zu schützen.

12.1.3. Router

Die Router bilden im Silc-Netz die Bindeglieder zwischen den einzelnen Servern. Wenn dies von dem Betreiber zugelassen wird, können sie zusätzlich Serverfunktionalitäten anbieten.

Im Unterschied zu einem Server verwaltet der Router nicht nur lokale Informationen, sondern besitzt Wissen über das gesamte Silc-Netz. Die dabei verwendeten globalen Datenstrukturen unterscheiden sich im Aufbau nicht von lokalen.

Außerdem muss er das Wissen aller Server in der *Cell* wiedergeben können, um Verbindungsanfragen zu dem betreffenden Server weiterleiten zu können. So muss er wissen, welcher Channel von welcher Serverinstanz betrieben wird und welche Benutzer sich wo angemeldet haben.

Wenn sich Benutzer auf einem Router anmelden können, muss der Router zusätzlich die Listen führen, die jeder Server lokal benötigt. Diese wurden im Abschnitt 12.1.2 beschrieben.

Die *Router ID* wird analog zu der *Server ID* gebildet. Im Weiteren findet die Kommunikation ebenfalls über einen TCP-Port statt. In der Grundeinstellung wird der Port 706 verwendet.

Backup-Router Der Ausfall eines Routers wäre fatal: erstens ist die *Cell* des Routers von der Außenwelt abgeschnitten und der Ring, den mehrere Router bilden sollten, wäre nicht mehr geschlossen. Daher empfiehlt sich der Einsatz eines Backup-Routers, der bei einem Ausfall die Aufgaben des primären Routers übernimmt.

Als Backup-Router kann ein normaler Server eingesetzt werden, der bei Bedarf Routingaufgaben übernehmen kann. Dabei ist es sehr wichtig, dass die Listen des Backup-Routers vom primären Router aktualisiert werden. Außerdem müssen alle Systeme, die mit dem primären Router kommunizieren, auch den Backup-Router kennen. Dies betrifft sowohl den vorhergehenden Router im Ring als auch die Server der lokalen *Cell*.

In einer *Cell* können mehrere Backup-Systeme existieren. Der Ablauf des Umschaltens zwischen Backup-Router und normalem Router wird in Anhang A.5.5 vorgestellt.

12.1.4. Aufbau eines Netzes

Ein selbst administriertes Silc-Netz kann sehr schnell sehr komplex werden. Auch wenn ein eigenes, einfaches Netz nur an das bereits bestehenden, öffentlichen Silc-Netz angeschlossen werden soll, ist ein Grundverständnis der globalen Kommunikation notwendig. Daher beschäftigt sich dieser Abschnitt mit einzelnen Bestandteilen eines Silc-Netzes.

Cell Die kleinste Gruppe eines Silc-Netzes ist eine *Cell*. Sie umfasst alle Klienten- und Serverinstanzen, die an einem Router angeschlossen sind. Dies schließt auch Server mit ein, die keine Verbindungen zu anderen Netzen unterhalten. In Abbildung 15 auf Seite 82 ist eine *Cell* mit einem Router und mehreren Server- und Klienteninstanzen skizziert.

Zusammenschluss mehrerer *Cells* Da sich nicht immer alle Klienteninstanzen innerhalb einer *Cell* befinden, müssen die *Cells* geeignet verbunden werden. Dazu wird ein Netz mit mehreren Routern aufgebaut. Es muss berücksichtigt werden, dass jeder Router Nachrichten, die nicht an seine *Cell* adressiert sind, an genau einen **primären Router** weiterleiten muss. Dies wird durch die Anordnung der Router in einem Ring erreicht. Ein solches Silc-Netz mit einigen angeschlossenen Server- und Klienteninstanzen ist in Abbildung 14 zu sehen.

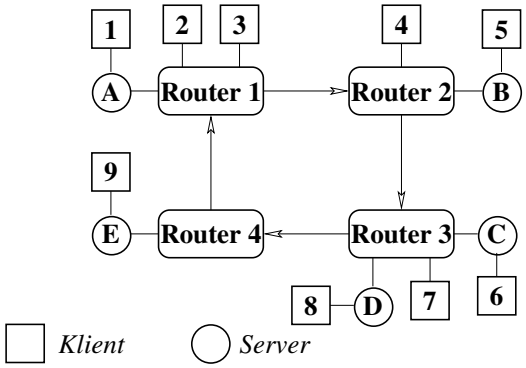

Abbildung 14: Silc - Architektur eines Netzes

Der primäre Router, der verwendet werden soll, wird in der Konfigurationsdatei des Routers eingetragen. Anhand seiner Listen kann der Router prüfen, ob er ein Datagramm in seine *Cell* oder zu seinem primären Router weiterleiten soll.

12.2. Kommunikation

12.2.1. Kommunikation innerhalb eines Netzes

Die Basis der Kommunikation innerhalb eines Silc-Netzes bilden die primären Router. Diese sind in einer Kreisstruktur zusammengeschlossen. Jeder Router besitzt Informationen über seine eigene *Cell* und über seinen primären Router, das heißt den Router der nach ihm in dem Kreis ist.

Erhält ein Router nun eine Nachricht, so prüft er anhand seiner Listen, ob der Empfänger im Netz angemeldet ist. Ist dies nicht der Fall, so erhält der Sender eine Fehlermeldung. Andernfalls schickt der Router das Datagramm an seinen primären Router weiter. Dieser überprüft, ob der Empfänger in seiner *Cell* angemeldet ist. Ist dies nicht der Fall, wird das Datagramm zum nächsten Router weitergesendet. Dies geht solange, bis ein Router das Datagramm an den Empfänger in seiner *Cell* zustellen kann.

12.2.2. Kommunikation innerhalb einer Cell

Wird eine Nachricht von einer Klienteninstanz abgesendet oder vom Router in die *Cell* weitergeleitet, beginnen neue Mechanismen zur Nachrichtenzustellung zu wirken. Diese sollen anhand einer Nachricht zwischen zwei Klienteninstanzen innerhalb einer *Cell* verdeutlicht werden. Die Zustellung anderer Nachrichten erfolgt analog.

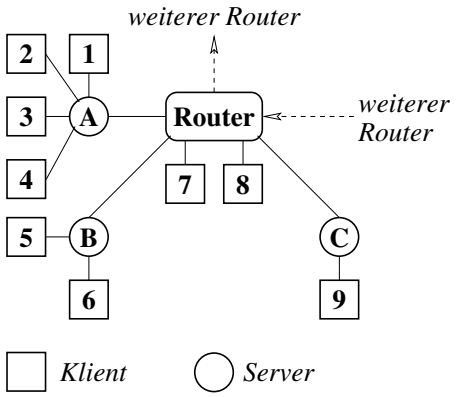

Abbildung 15: Silc - Architektur einer *Cell*

In der *Cell*, die in Abbildung 15 dargestellt wird, möchte die Klienteninstanz „2“ mit der Klienteninstanz „9“ kommunizieren (zum Beispiel mit einem privaten *query*). Die Klienteninstanz „2“ ist direkt an der Serverinstanz „A“ und der Klienteninstanz „9“ an der Serverinstanz „C“ angeschlossen.

Als erstes verschlüsselt die Klienteninstanz „2“ das Datagramm mit dem *Session Key*, den sie mit der angeschlossenen Serverinstanz „A“ ausgehandelt hat und schickt das Datagramm an diese. Nachdem es die Serverinstanz empfangen hat, entschlüsselt sie das Datagramm.

Wäre die Klienteninstanz „9“ bei Serverinstanz „A“ angemeldet, könnte sie die Anfrage wieder mit dem *Session Key* des Klienten „9“ verschlüsseln und ohne Weiteres zu der Klienteninstanz weiterleiten. Da der Server die Klienteninstanz „9“ nicht in seinen lokalen Listen verwaltet, schickt er die Anfrage an den Router in der *Cell* weiter. Hierfür verschlüsselt sie das Datagramm mit dem *Session Key*, den Serverinstanz und Router teilen.

Beim Router angekommen, wird das Datagramm wieder entschlüsselt. Der Router führt zwei Listen: eine globale und eine lokale, in der alle benötigten Informationen der *Cell* zusammengefasst sind. Anhand der gesuchten *Client ID* kann der Router feststellen, ob sich die gesuchte Klienteninstanz in dieser oder einer anderen *Cell* befindet. Hier kann der Fall auftreten, dass die Klienteninstanz „9“ in keiner der beiden Listen geführt wird. Dann wird eine Fehlermeldung an die Klienteninstanz „2“ zurückgesendet.

Ist die gesuchte Klienteninstanz direkt an dem Router angeschlossen, leitet der Router die Anfrage an die Klienteninstanz direkt weiter. Da sich der Server zu der gesuchten Klienteninstanz aber in der lokalen *Cell* befindet, wird die Nachricht dorthin (also zu Server „C“) gesendet. Andernfalls würde der Router die Nachricht zum nächsten Router in der nächsten *Cell* weiter schicken. Auch bei diesem Informationsaustausch werden die Datagramme mit dem zugehörigen *Session Key* geschützt.

Nachdem die Nachricht von der Serverinstanz „C“ empfangen und entschlüsselt wurde, kann der Server nachschlagen, ob die Klienteninstanz „9“ noch an der Serverinstanz angemeldet ist. Wenn dies der Fall ist, kann die Nachricht verschlüsselt und zu ihr gesendet werden.

Dieses Szenario soll verdeutlichen, welche Systeme bei einer Kommunikation Daten austauschen. Um welche Informationen es sich dabei im Einzelnen handelt, zeigt Anhang A.5.

12.2.3. Kommunikation innerhalb eines Channels

Die Kommunikation innerhalb eines Channels ist der Kommunikation zwischen zwei Klienteninstanzen, wie sie im vorigen Abschnitt vorgestellt wurde, sehr ähnlich. Dazu verschlüsselt der Benutzer die Nachricht mit dem ausgehandelten *Channel Key* und sendet diese an den Server, der den gewünschten Channel verwaltet.

Erst dieser Server kennt auch den *Channel Key* und kann die Nachricht entschlüsseln. Alle dazwischenliegenden Systeme leiten das Datagramm einfach weiter, wobei die Information zusätzlich mit den jeweilige *Session Keys* geschützt wird.

Der Server schlägt nun in seiner lokalen Liste nach, welche *Client ID*s zu dem Channel gehören. An jeden dieser Klienten sendet der Server nun eine verschlüsselte Kopie der empfangenen Nachricht. Dies geschieht analog zu den bisher vorgestellten Beispielen.

12.3. Schutz der Kommunikation

12.3.1. Kryptographische Grundlagen

Beim Einsatz von Silc spielt Kryptographie eine wichtige Rolle. Wie schon in der Einleitung erwähnt, ist es unter Silc nicht möglich, unverschlüsselt zu kommunizieren. Die Gefahr des versehentlichen Deaktivierens der Verschlüsselung ist damit nicht gegeben.

In den folgenden Unterkapiteln wird gezeigt, wie Verschlüsselung unter Silc realisiert werden kann.

Unterstützte Verfahren Silc unterstützt eine Vielzahl von Verschlüsselungstechniken. Zum einen können symmetrische Verschlüsselungsverfahren eingesetzt werden. Der dazu notwendige Schlüsselaustausch wird durch das Diffie-Hellmann-Merkle-Verfahren (siehe Anhang B.3) realisiert.

Zum anderen unterstützt Silc die Verwendung von asymmetrischen Public-Key-Verfahren. Diese werden hauptsächlich zur Erstellung von Signaturen verwendet.

Mit Hilfe des privaten Schlüssels wird zu der Nachricht eine Signatur erzeugt. Der Empfänger kann unter Verwendung des zugehörigen öffentlichen Schlüssels prüfen, ob die Signatur und die übermittelte Nachricht zusammenpassen. Damit kann ermittelt werden, ob die Nachricht tatsächlich vom angegebenen Sender stammt.

Zur Sicherung der Integrität können *Message Authentication Codes* (MAC) gebildet werden. Diese MAC wird von der gesamten Nachricht und dem verwendeten symmetrischen Schlüssel erstellt. Da der symmetrische Schlüssel einem Außenstehenden normalerweise nicht bekannt ist, kann so das unbemerkte Abfangen, Manipulieren und Weiterschicken des Datagramms durch einen Angreifer (*Man-In-The-Middle-Attack*) verhindert werden. Da aus der Prüfsumme praktisch nicht auf die Nachricht geschlossen werden kann, stellt dieses Verfahren auch kaum eine Gefahr für die Vertraulichkeit des Schlüssels dar.

Im Folgendem werden die unterstützten Verfahren aufgelistet. Zur Aushandlung der zu nutzenden Verfahren werden die angegebenen Zeichenketten in den Datagrammen übermittelt.

symmetrische Verschlüsselungsverfahren

Zeichenkette	Algorithmus	Modus (<mode>)	Schlüssellänge (<len>)
aes-<len>-<mode>	AES	cbc, ctr oder rcbc	128, 192 oder 256 Bit
twofish-<len>-<mode>	Twofish	cbc, ctr oder rcbc	128, 192 oder 256 Bit
cast-<len>-<mode>	CAST	cbc, ctr oder rcbc	128, 192 oder 256 Bit
serpent-<len>-<mode>	Serpent	cbc, ctr oder rcbc	128, 192 oder 256 Bit
rc6-<len>-<mode>	RC6	cbc, ctr oder rcbc	128, 192 oder 256 Bit
mars-<len>-<mode>	MARS	cbc, ctr oder rcbc	128, 192 oder 256 Bit

Folgende blockorientierte Betriebsmodi des Verschlüssungssystems sind möglich:

cbc Cipher-Block-Chaining: Das Ergebnis der Verschlüsselung des vorhergehendes Blocks wird als Initialisierungsvektor (siehe Anhang B.6) des folgenden Blocks verwendet. Der erste Initialisierungsvektor wird zufällig erzeugt.

ctr Counter-Mode: Der Counterblock ist im Prinzip ein spezieller Initialisierungsvektor, der beiden Seiten bekannt ist. Aus dem Counter-Block wird dann der Key berechnet.

rcbc Randomized-Cipher-Block-Chaining: Für jedes Datagramm wird der Initialisierungsvektor zufällig bestimmt. Treten mehrere Blöcke in einem Datagramm auf, wird ab dem zweiten Block das *inter-packet-chaining* (Der Initialisierungsvektor wird aus dem vorigen Verschlüsselungsblock gebildet) verwendet.

asymmetrische Verschlüsselungsverfahren

Zeichenkette	Algorithmus
rsa	RSA
dss	DSS

Hash- und MAC-Verfahren

Zeichenkette	Algorithmus	Hash-Länge
sha1	SHA-1	20 Byte
md5	MD5	16 Byte
Zeichenkette	**Algorithmus**	**MAC-Länge**
hmac-sha1-96	HMAC-SHA1	12 bytes
hmac-md5-96	HMAC-MD5	12 Bytes
hmac-sha1	HMAC-SHA1	20 Bytes
hmac-md5	HMAC-MD5	16 Bytes

12.3.2. Schlüsselaustausch über das SKE-Protokoll

Der Austausch von Schlüsseln, die für die sichere Authentifizierung und für die Umwandlung von Klartext in ein Chiffrat benötigt werden, kann über das *SILC Key Exchange* (SKE)-Protokoll realisiert werden. Der Schlüsselaustausch ist die Voraussetzung für jeden weiteren Schritt, der in Silc ausgeführt wird. Dadurch stehen geheime Schlüssel zur Verfügung, die für den folgenden Datenaustausch die Integrität und Vertraulichkeit sichern. Der *Session Key*, der zu einem späteren Zeitpunkt benötigt wird, wird ausschließlich durch das SKE-Protokoll generiert.

Der Schlüsselaustausch kann beim Verbindungsaufbau einer Klienten- zur Serverinstanz oder von Serverinstanzen zum Router geschehen. Zusätzlich kann der Schlüsselaustausch zwischen zwei einzelnen Klienteninstanzen, die nicht nut den Schlüssel des Servers nutzen wollen, angeregt werden.

Der mit Hilfe des SKE-Protokolls erstellte Schlüssel ist nur für eine bestimmte Zeitspanne (in der Regel eine Stunde) gültig. Nach einem Abbruch der Kommunikation, wie durch das Abmelden einer Klienteninstanz von einer Serverinstanz, oder bei einer Neuanmeldung muss ein neuer Schlüssel generiert werden. Für die Erstellung kann *Perfect Forward Secrecy (PFS)* (siehe hierzu Anhang B.9) berücksichtigt werden.

Die Struktur der für das SKE-Protokoll benötigten Silc-Datagramme ist im Anhang B.9 zu finden.

Zeitlicher Ablauf des Datagrammaustauschs In der Abbildung 16 wird der Ablauf der *Silc Key Exchange*-Kommunikation vorgestellt. Hierbei werden der TCP-*Handshake*, der für eine Verbindungsinitiierung vorausgesetzt wird, und die TCP-ACK-Datagramme, die jedes Datagramm bestätigen, nicht berücksichtigt.

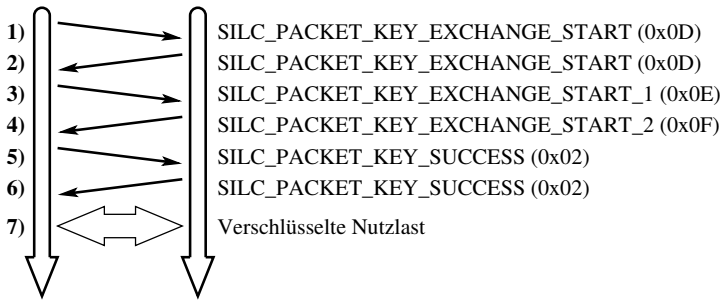

Abbildung 16: Silc - Protokollablauf der Kommunikation zwischen Sender und Empfänger

Als erstes werden in einem *SILC_PACKET_KEY_EXCHANGE_START*-Datagramm die vom Sender unterstützten Kryptographiealgorithmen übermittelt. Als solche kommen PKCS (*Public Key Cryptographic Standard*, siehe Anhang B.12), Verschlüsselungs- und Hash-Algorithmen in Frage. Die Gegenseite, zu der die Verbindung aufgebaut wurde, wählt von den gesendeten Eigenschaften eine aus, die sie selbst ebenfalls unterstützt und teilt diese Entscheidung dem Gegenüber mit.

Nun werden die Schlüssel über Diffie-Hellmann-Merkle (siehe hierzu Anhang B.3) vereinbart. Parallel hierzu tauschen die Kommunikationsseiten ihre öffentlichen Schlüssel und Zertifikate aus. Dies geschieht mit dem *SILC_PACKET_KEY_EXCHANGE_START_1/2*-Datagramm. Dabei wird von jedem Kommunikationspartner ein Datagramm versendet (Siehe Grafik 16, Datagramm 3) und 4)). Tritt dabei ein Fehler auf, schickt der Kommunikationspartner, der diesen bemerkt hat, ein *SILC_PACKET_FAILURE*-Datagramm. Daraufhin wird der Schlüsselaustausch abgebrochen.

Obwohl bisher nur *Key_Exchange_Payload*-Datagramme versendet wurden, fällt auf, dass es sich um verschiedene *Key_Exchange_Payload*-Datagramme handelt. Für den Austausch der unterstützen Algorithmen (siehe Datagramme 1) und 2)) wird im Feld *Packet Type* der Wert

0x0D eingetragen. Bei dem eigentlichen Schlüsselaustausch sendet der Sender ein Datagramm mit dem *Packet Type*-Wert **0x0E** und der Empfänger antwortet mit seinen Schlüsselparametern in einem Datagramm, das den *Packet Type*-Wert **0x0F** enthält.

Nachdem der Schlüsselaustausch erfolgreich stattgefunden hat, wird dies mit einem *SILC_PACKET_KEY_SUCCESS*-Datagramm bestätigt (Siehe Datagramm 5) und 6) in Abbildung 16). Ab diesem Zeitpunkt bis zum Ende der Gültigkeit des Schlüssels wird die Kommunikation zwischen den beiden Partnern mit diesem Key verschlüsselt. Die verschlüsselte Nutzlast wird in Abbildung 16 mit der Nummer 7 dargestellt.

Es ist anzumerken, dass bis zu diesem Zeitpunkt noch keine Verschlüsselung stattgefunden hat, da vorher noch keine Schlüssel ausgehandelt wurden.

12.3.3. Verschlüsselung zwischen verschiedenen Endpunkten

Nachrichten können von Silc mehrmals mit verschiedenen Schlüsseln des ausgehandelten Algorithmuses verschlüsselt werden. Jede Nachricht wird mit dem *Session Key* kodiert. Gehört diese zu einer Kommunikation innerhalb des Channels, so wird sie zusätzlich noch mit einem *Channel Key* geschützt. Den Vorteil der mehrfachen Verschlüsselung zeigt Abbildung 17. Durch die Verwendung des *Channel Keys* können dazwischenliegende Stationen die Ursprungsnachricht nicht ermitteln. Analog zum *Channel Key* können auch die Verbindungen zwischen Benutzern durch eine *Client-To-Client*-Verschlüsselung geschützt werden.

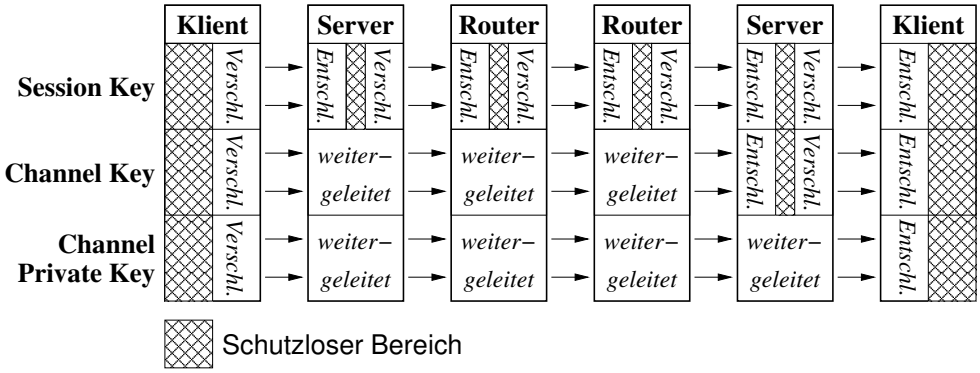

Abbildung 17: Silc - Vergleich Session, Channel und Channel Private Key

Session Key Der *Session Key* wird verwendet, um die direkte Verbindung zwischen zwei Silc-Systemen zu schützen. Alle Informationen zwischen einer Klienten- und der Serverinstanz, einer Serverinstanz und dem Router oder Routern werden mit dem *Session Key* symmetrisch verschlüsselt.

Der Schlüsselaustausch zwischen den beiden Endpunkten wird über das Diffie-Hellmann-Merkle-Verfahren (Siehe Anhang B.3) realisiert und findet sowohl zu Beginn der Verbindung als auch in regelmäßigen Abständen (in der Voreinstellung eine Stunde) statt. Ziel der *Session Key*-Verschlüsselung ist es, die Auswertung des protokollierten Netzverkehrs durch einen Angreifer zu erschweren.

Warum die *Session Key*-Verschlüsselung nicht als alleinige Verschlüsselung ausreicht zeigt Abbildung 17. Jede Verbindung zwischen dem Benutzer und dem Server wird mit dem jeweiligen *Session Key* geschützt. Hierfür wird es auf einem System mit dem entsprechenden *Session Key* verschlüsselt und beim nächsten System entschlüsselt. Dieses System verschlüsselt wiederum das Datagramm mit dem nächsten *Session Key* und schickt es weiter. Zwischen der Entschlüsselung und der Verschlüsselung existiert eine Zeitspanne, in der die Informationen nicht geschützt werden. Dies wird durch dem schraffierten Bereich dargestellt. Um diese Bereiche ebenfalls zu schützen, muss eine weitere Verschlüsselung mit zum Beispiel dem *Channel Key* erfolgen.

Channel Key Der vorige Abschnitt hat gezeigt, dass nur mit dem *Session Key* verschlüsselte Nachrichten kurzzeitig ungeschützt sind. Zu dieser Zeit könnte ein Angreifer, der eines der dazwischen liegenden Systeme unter seiner Kontrolle gebracht hat, alle übermittelten Informationen einsehen und verändern.

Aus diesem Grund verwendet Silc sogenannte *Channel Keys*. In der Praxis wird ein Channel, den mehrere Benutzer betreten können, von einer Serverinstanz generiert und verwaltet. Mit diesen Ansatz ist es möglich, eine Verschlüsselung einzusetzen, die die Informationen auf der gesamten Strecke zwischen der Klienteninstanz und dem Server schützt, auf dem sich der Channel befindet. Die dazwischenliegenden Systeme können die Informationen nicht vollständig entschlüsseln sondern müssen sie direkt weiterleiten.

Diese Art der Verschlüsselung ist transparent für den Benutzer und nicht abschaltbar. Dieser Vorgang wird ebenfalls in Abbildung 17 verdeutlicht. Die regelmäßige Aushandlung des Schlüssels initiiert dabei die Serverinstanz. Dies geschieht in vorher definierten Abständen und zusätzlich, wenn ein Benutzer den Channel betritt oder verlässt.

Es muss beachtet werden, dass ein Angreifer, der den Server mit dem zu nutzenden Channel unter seine Kontrolle gebracht hat, unter Umständen auch alle übertragenen Informationen mitlesen und verändern kann. Daher sollte, wenn möglich, eine Verschlüsselung aktiviert werden, bei der die Ver- und Entschlüsselung nur auf der Seite der Klienten geschehen kann. Dies wird durch das folgende Verschlüsselungsverfahren möglich.

Channel Private Key Am effizientesten ist die Verwendung einer Verschlüsselung, bei der nur die Endpunkte der Kommunikation die zu übertragende Information von dem Klartext in das Chiffrat und wieder zurück umwandeln können. Dazu schützt der Sender die Informationen mit dem *Channel Private Key*. Alle Benutzer im Channel können die Nachrichten wieder entschlüsseln, alle dazwischen liegenden Systeme müssen die Nachricht ohne Kenntnis des Inhalts weiterleiten (Siehe Abbildung 17).

Dazu stehen zwei Möglichkeiten zur Verfügung, die von dem Benutzer explizit aktiviert werden müssen. Zum einen kann dies eine symmetrischen Verschlüsselung, zum anderen eine asymmetrische Public-Key-Verschlüsselung sein. Für die symmetrische Verschlüsselung wird aus einer geheimen Zeichenkette, die im Folgendem als Passphrase (siehe Kapitel B.8) bezeichnet wird, ein Schlüssel erzeugt. Die Passphrase muss dabei allen Benutzern bekannt sein. In der Praxis ist dies schwer umzusetzen, da die Passphrase auf einem sicheren Weg zu den Benutzern gelangen muss.

Hat ein Angreifer den Server, auf dem der Channel verwaltet wird, unter seine Kontrolle gebracht und die Passphrase abgehört, so kann er erneut sämtliche Kommunikation im Channel mitlesen.

Um diese Chance zu minimieren, sollte die Passphrase regelmäßig gewechselt werden. Die Übertragung der neuen Passphrase sollte nicht über Silc stattfinden, da diese Kommunikation bereits von einem Angreifer kompromittiert sein könnte.

Viel sinnvoller ist daher der kryptographische Schutz mit einem asymmetrischen Verschlüsselungsverfahren, da dieses ohne Schlüsselaustausch auskommt. Hierfür müssen alle teilnehmenden Channel-Benutzer ihren öffentlichen Schlüssel allen Mitgliedern im Channel zukommen lassen. Der Sender verpackt seine Nachrichten jeweils mit dem öffentlichen Schlüssel der Channelbenutzer. Jeder Benutzer kann mit seinem privaten Schlüssel nur die Nachricht lesen, die mit seinem öffentlichen Schlüssel verschlüsselt wurde.

Verschlüsselung privater Nachrichten Das Problem bei privaten Nachrichten zwischen zwei Benutzern ist, dass diese nur mit dem *Session Key* geschützt werden. Das bedeutet, dass prinzipiell jeder Serverbetreiber zwischen den Beiden die Nachrichten mitlesen könnte.

Um Nachrichten zu übertragen, die nur an den beteiligten Kommunikationsendpunkten eingesehen werden dürfen, darf die Nachricht zu keiner Zeit bei der Übertragung vollständig entschlüsselt werden. Dies kann nur durch ein Verfahren realisiert werden, bei dem die Zwischensysteme keine Kenntnis von dem Schlüssel haben.

Wie auch bei dem *Channel Private Key*-Verfahren existieren zwei Möglichkeiten, wie diese Anforderung realisiert werden kann. Da beide Verfahren Probleme mit sich bringen, müssen sie vom Benutzer erst aktiviert werden.

Für den Benutzer ist die symmetrische Verschlüsselung mit einer geheimen Passphrase (siehe Anhang B.8) einfacher verständlich. Diese Passphrase muss beiden Partnern bekannt sein und bei jeder Aktualisierung neu ausgetauscht werden. Dieser Vorgang ist jedoch problematisch. Hierfür bietet Silc keine Mechanismen und daher sind andere Kommunikationswege zu wählen. Ein Beispiel ist die Übertragung des Schlüssels über eine verschlüsselte E-Mail. Dabei muss weiterhin beachtet werden, dass ein Angreifer, der den Schlüssel schon bei der Übertragung mitlesen kann, die gesamte Silc-Kommunikation abhören kann.

Um die Sicherheit zu erhöhen, sollten die Passphrasen in regelmäßigen Abständen ausgewechselt werden. Das ist aber für den Benutzer sehr unbequem, da er sich zum einen die neue Passphrase merken muss und zum anderen diese wieder sicher mit dem Partner ausgetauscht werden muss. Daher muss in vielen Fällen davon ausgegangen werden, dass die Kommunikation zwischen den beiden Benutzern immer mit der gleichen Passphrase geschützt wird.

Da die Übermittlung des Schlüssels viele Gefahren für dessen Vertraulichkeit mit sich bringt, ist in den meisten Anwendungsfällen die Aushandlung des Schlüssels über das SKE-Protokoll (Siehe Seite 12.3.2) vorzuziehen.

Es ist wichtig zu wissen, dass für diesen Schlüsselaustausch eine neue, unabhängige TCP-Verbindung aufgebaut werden muss. Dies hat den Vorteil, dass der Schlüsselaustausch von einem Angreifer, der einen Silc-Server oder Router unter seine Kontrolle gebracht hat, nicht mitgehört werden kann. Um dies zu realisieren, muss eventuell bei der Anfrage zum Schlüsselaustausch

und auf jedem Fall bei der Erwiderung die IP-Adresse und die TCP-Portnummer angegeben werden. Und genau an dieser Stelle liegt das Problem des Schlüsselaustauschs. Befindet sich der Sender hinter einem NAT-Gateway und besitzt er nur eine private IP-Adresse, kann der Empfänger keine Verbindung zu ihm aufbauen. Eine Lösung wäre, dass die Kommunikationspartner darauf achten, dass derjenige, der die öffentliche IP-Adresse besitzt, den Schlüsselaustausch anregt. Dieser Lösungsansatz kann aber nicht verwendet werden, wenn sich beide Kommunikationspartner hinter verschiedenen NAT-Gateways befinden. In diesem Fall kann SKE nicht genutzt werden.

12.3.4. SILC Connection Authentification Protocoll

In vielen Fällen wird es von dem Betreiber des Servers gewünscht, dass sich nur eine bestimmte Gruppe von Benutzern auf dem Silc-Server anmelden darf. Wenn der Rechner, von dem sich der Benutzer anmeldet, als Grundlage für die Authentisierung dient, kann dies sehr einfach über Paketfilterregeln realisiert werden.

Doch dieses Vorgehen ist nicht immer sehr elegant. Spätestens, wenn einer der Rechner, die sich anmelden dürfen, auch als NAT-Gateway dient, können sich alle Rechner hinter diesem System auf dem Silc-Server anmelden. Daher ist eine Authentisierung, die auf dem einzelnen Benutzer basiert, die praktikablere Lösung. Natürlich spricht nichts gegen eine Kombination dieser beiden Möglichkeiten.

Die Authentisierung kann nicht nur für die Beziehung zwischen Klienten- und Serverinstanz sinnvoll sein, sondern auch für die Verbindung von einer Serverinstanz zu einem Router und für die Kommunikation von Routern untereinander. Diese Authentisierung wird durch das *SILC Connection Authentication*-Protokoll ermöglicht. Im Gegensatz zu der Klienteninstanz ist die Authentisierung einer Serverinstanz gegenüber einem Router nicht optional. Das *SILC Connection Authentication*-Protokoll wird direkt nach dem *Silc Key Exchange*-Protokoll (SKE) (siehe Kapitel 12.3.2) ausgeführt. Anschließend wird die Einleitung des *SILC Connection Authentication*-Protokolls nur noch auf Wunsch des Klienten, zum Beispiel bei einer Anfrage, eingeleitet.

Die Authentisierung kann auf Grundlage einer Passphrase (siehe Kapitel B.8) oder eines öffentlichen Schlüssels geschehen. Diese Passphrase muss dabei im Vorfeld auf dem Server eingetragen oder der Schlüssel auf den Server kopiert werden.

Die von dem Server unterstützte Methode wurde im Vorfeld durch ein *SILC_PACKET_CONNECTION_AUTH_REQUEST*-Datagramm ausgehandelt.

Da der Schlüsselaustausch im Vorfeld stattgefunden hat, wird die Authentisierung komplett verschlüsselt durchgeführt. Diese Maßnahme verhindert das Mitlesen des Schlüssels oder der Passphrase durch einen Angreifer.

Authentisierung mit einer Passphrase Für dieses Verfahren muss die Passphrase dem Server im Vorfeld bekannt sein. Dies geschieht zum Beispiel, in dem der Betreiber des Servers diese Passphrase in die Konfigurationsdatei aufnimmt.

Daher ist es sehr wichtig, dass die Datei, in der die Passphrase auf dem Server eingetragen wird, geschützt wird. Es wäre für die sichere Authentisierung fatal, wenn ein Angreifer diese Passphrase direkt in der Konfigurationsdatei des Servers lesen könnte.

Für die Durchführung der Authentisierung wird die Passphrase verschlüsselt in der Nutzlast des *SILC_PACKET_CONNECTION_AUTH*-Datagramms übertragen. Der Empfänger entschlüsselt die Passphrase mit dem Schlüssel, der während des Schlüsselaustauschs ausgehandelt wurde und vergleicht sie mit der ihm bekannten Passphrase.

Stimmt die übertragene Passphrase mit dem Vergleichspasswort überein, kann die Authentisierung mit dem *SILC_PACKET_SUCCESS*-Datagramm erfolgreich abgeschlossen werden. Im anderen Fall wird ein *SILC_PACKET_FAILURE*-Datagramm an den abgewiesenen Sender versendet.

Authentisierung mit dem öffentlichen Schlüssel Für die erfolgreiche Authentisierung mit dem öffentlichen Schlüssel des Senders muss dieser auf dem Empfänger (Silc-Server beziehungsweise Router) abgelegt worden sein.

Generell muss bei diesem Verfahren der Sender mit Hilfe seines geheimen Schlüssels, der nur ihm bekannt sein sollte, eine Nachricht signieren und diese dem Empfänger schicken. Dieser kann nun mit Hilfe des hinterlegten Schlüssels feststellen, ob die übertragene Nachricht wirklich mit dem privaten Schlüssels signiert wurde. Ist dies der Fall, wird die Authentisierung als erfolgreich angesehen.

Um die Signatur erfolgreich zu vergleichen, muss beiden Seiten der Inhalt der übermittelten Nachricht bekannt sein. Hierfür wird erst eine Prüfsumme über die Nutzlast des ersten *Silc Key Exchange*-Datagramms (siehe Kapitel 12.3.2) gebildet. Das Hashverfahren, dass die Prüfsumme generiert, wurde während des *Silc Key Exchange*-Verfahrens ausgehandelt. Da beide Kommunikationspartner diese Nutzlast kennen (Der Sender hat das Datagramm gesendet und der Empfänger hat es empfangen) ist dies gewährleistet.

12.4. Praxis

Eine Beschreibung, die es ermöglicht, Silc sofort auszuprobieren, darf dieser Vorstellung nicht fehlen. Obwohl zahlreiche graphische Frontends für Silc existieren, sollen die Grundbefehle von Silc anhand der Kommandozeilenschnittstelle beschrieben werden.

Um diese Anleitung verständlicher zu gestalten, werden Beispielbenutzer verwendet. Wenn man sich die am häufigsten vergebenen Vornamen von Neugeburten seit 1890 auf der Seiten *beliebte-vornamen.de*[18] anschaut, stellt man fest, dass die Namen „Ursula“ und „Peter“ am häufigsten vergeben wurden.

Um die Eigenschaft von Silc, dass die *Nicknames* nicht eindeutig sein müssen, zu demonstrieren und da ein hoher Bedarf an dem *Nickname* „Ursula“ beziehungsweise „Peter“ besteht, werden diese Namen verwendet. Der Name des Benutzers, von dem die Verbindung ausgeht, ist „Peter“, und im Silc-Netz existieren mindestens zwei „Ursulas“.

Als erster Schritt sollte eine Verbindung zum nächsten Silc-Server aufgebaut werden. Dies kann mit dem folgendem Befehl durchgeführt werden:

```
bash-2.05b$ silc --connect=sauna.silcnet.org --nick=peter
```

In diesem Beispiel wird eine Verbindung zu dem Server *sauna.silcnet.org* als Benutzer *peter*

[18] Die beliebtesten Vornamen: http://www.beliebte-vornamen.de/alle_jahre.htm

aufgebaut. Ist dies der erste Aufruf von Silc, wird zuerst ein Schlüsselpaar erstellt, das mit einem Passwort geschützt werden muss.

Nach dem erfolgreichen Verbinden mit dem Server muss anhand des Server-Fingerprints überprüft werden, ob es sich um den gewünschten Server handelt. Dies hat den Hintergrund, dass ein Angreifer seinen eigenen Server bereitgestellt haben könnte und die Anfrage von dem Benutzer auf diesen umgeleitet hat.

Daher ist der Fingerprint vorher auf einem unabhängigen Weg zu beschaffen und zu bestätigen. Nachdem der Fingerprint akzeptiert wurde, wird er für den nächsten Verbindungsaufbau zum Server von der Klienteninstanz lokal abgelegt. Danach ist der Benutzer auf dem Server angemeldet.

12.4.1. Channel-Kommunikation

Channels spielen bei Silc eine zentrale Rolle. Daher möchte „Peter“ im nächsten Schritt einen Channel betreten. Der Befehl hierzu ist schon von IRC bekannt:

```
/join #peters-welt
```

Da er der erste Benutzer im Channel ist, wird dieser automatisch generiert und „Peter“ erlangt übergeordnete Rechte im Channel.

Um bei einem späteren Beitritt in den Channel wieder die Rechte zu erlangen, kann „Peter“ den *Founder*-Status des Channels setzen. Dies geschieht mit der Anweisung:

```
[#peters-welt] /cmode #peters-welt +f
```

Ab diesen Zeitpunkt kann „Peter“ mit dem Befehl

```
[#peters-welt] /cumode #peters-welt +f peter
```

nach einem Wiedereintritt den *Founder*-Status zurück erlangen.

In dem Fall, dass „Peter“ dem Serverbetreiber nicht vertraut, kann er dafür sorgen, dass nur die Benutzer, die die geheime Passphrase kennen, die Nachrichten im Channel lesen dürfen. Dies kann er mit den folgenden Befehlen realisieren:

```
[#peters-welt] /cmode #peters-welt +k
[#peters-welt] /key CHANNEL #peters-welt set geheimnis
```

Die erste Anweisung setzt das Verschlüsselungs-Flag für den Channel und der zweite Befehl setzt die benötigte Passphrase auf „geheimnis“.

Wenn sich nun ein weiterer Benutzer im Channel anmeldet und mitlesen möchte, so kann er dies nur mit dem folgenden Befehl realisieren:

```
/join #peters-welt
[#peters-welt] /key CHANNEL #peters-welt set geheimnis
```

Ohne den */key*-Befehl kann der Benutzer zwar in den Channel eintreten, aber nicht mitlesen oder schreiben.

12.4.2. Benutzer-Kommunikation

Um ein privates Gespräch mit einem anderen Benutzer zu starten, muss ein eindeutiger Bezeichner bekannt sein, der den Gesprächspartner identifiziert. Das folgende Beispiel skizziert „Peters“ Kommunikation mit „Ursula“. Da „Peter“ aber nur den *Nickname* „Ursula“ und deren vollständigen Namen („Die richtige Ursula“) kennt, muss dieser erst weitere Informationen einholen:

```
/whois ursula
***ursula@ircserver (urs@urs-rechner)
*** nickname   :ursula (ursula@urs-rechner1)
*** realname   :Die richtige Ursula
*** idle       :19 seconds
*** fingerprint:D1D5 12EC 4D8D 5BAB 5087  A7EC C489 73E8 722D F70C
***ursula@ircserver (ursel@urs-rechner)
*** nickname   :ursula (ursula@urs-rechner2)
*** realname   :Eine andere Ursula auf Ursulas Rechner
*** idle       :7 seconds
*** fingerprint:599E 8F26 97F4 5032 8B96  580C 217C 2414 C11D B786
```

Bei der Ausgabe der *whois*-Anfrage ist die Entscheidung nicht ganz einfach. Man sieht, dass es zwei „Ursulas“ gibt, wobei anhand des kompletten Namen die gewünschte „Ursula“ ausgewählt werden kann. Da sich beide „Ursulas“ vom gleichen Rechner aus beim gleichen Silc-Server angemeldet haben, werden sie zur Unterscheidung durchnummeriert.

Um der gewünschten, „richtigen“ Ursula eine Nachricht zu schicken, kann „Peter“ folgenden Befehl ausführen:

```
/query ursula@urs-rechner1
```

Da die Verbindung zu „Ursula“ aufgebaut wurde, muss diese bei einer Antwort nichts weiter beachten - der richtige „Peter“ erhält die Nachricht automatisch, auch wenn mehrere existieren würden.

12.5. Mögliche Angriffe

Bei dem Protokolldesign von Silc wurde großer Wert auf die Sicherheit gelegt und durch die verschiedenen Hilfsprotokolle ist es nur sehr schwer anzugreifen. Selbst der Benutzer, der meist die größte Schwachstelle darstellt, kann bestimmte Sicherheitsprinzipien nicht ignorieren.

In diesem Abschnitt sollen die wenigen möglichen Ansätze für einen erfolgreichen Angriff beleuchtet werden. Diese sind jedoch meist prinzipbedingt und können daher nicht durch ein einfaches Softwareupdate aus der Welt geschaffen werden.

Die nichtsystemspezifischen Angriffsmethoden, die im Kapitel 3 vorgestellt wurden, gelten hier natürlich auch, sollen aber nicht weiter betrachtet werden.

12.5.1. Bedrohung

Abhören Ein einfacher Angriff ist das Sniffen (siehe Anhang B.16) der Silc-Datagramme, die dann ausgewertet werden können. Dazu muss allerdings die Verschlüsselung erfolgreich angegriffen werden, in dem entweder der verwendete Schlüssel ermittelt wird oder aber das

Kryptographieverfahren selbst Ziel des Angriff ist. Letztere Methode ist jedoch nicht so einfach umzusetzen und ist in den letzten Jahren (wahrscheinlich) noch niemanden gelungen.

Wie bereits erläutert, werden Silc-Datagramme zwischen direkt verbundenen Silc-Systemen mit dem *Session Key* verschlüsselt. Dieser *Session Key* ist nur für die zugehörige Verbindungsstrecke gültig und somit erfolgt auf jedem dazwischenliegenden Silc-System eine Ent- und erneute Verschlüsselung.

Es muss also davon ausgegangen werden, dass trotz der Verschlüsselung die Datagramme auf den Silc-Systemen kurzzeitig unverschlüsselt vorliegen. Besitzt ein Angreifer nun die Kontrolle über ein solches System, so kann er sämtliche Silc-Datagramme, die über dieses System laufen, ohne weitere Anstrengungen mitlesen.

Man könnte nun annehmen, dass die *Channel Key*-Verschlüsselung Abhilfe schafft. Dies ist aber nicht der Fall, da das Problem dadurch nur verlagert wird. Ein Angriff wird nur dahingehend erschwert, dass nicht ein beliebiges Silc-System unter die Kontrolle gebracht werden muss, sondern der Silc-Server, auf dem der Channel verwaltet wird.

Angriffe auf die Verfügbarkeit Neben den bisher beschriebenen Angriffen auf die Integrität und Vertraulichkeit kann auch die Verfügbarkeit Ziel eines solchen werden.

Dabei muss nicht einmal wie bei den vorher beschriebenen Angriffen die Serverinstanz verwundbar sein. Es genügt, dass andere aktive Netzdienste anfällig für Angriffe sind. Davon sind dann meist sowohl primärer Router als auch Backuprouter betroffen. Daher führt der Einsatz redundanter Systeme nicht immer zum Erfolg.

Dennoch können Fehler in der Software des Servers nicht ausgeschlossen werden. Silc-Server sind nicht so weit verbreitet wie beispielsweise SSH-Server und daher können Softwarefehler sehr lange unbemerkt bleiben.

12.5.2. Schutz

Sicherheitsbewusste Silc-Benutzer setzen auf die direkte Verschlüsselung zwischen den Klienten. Aber selbst dies ist nicht als absolut sicher anzusehen. Der Aufbau einer direkten und verschlüsselten Benutzer-zu-Benutzer-Kommunikation ist stets mit einem höheren Aufwand verbunden. Wenn davon ausgegangen wird, dass der Schlüsselaustausch nicht über Silc ausgeführt werden kann (Beispiel: beide Gesprächspartner sind hinter einem NAT-Gateway), muss der Schlüssel manuell ausgetauscht werden.

Selbst, wenn dieser gelingt, was in der Praxis nur mit hohem Aufwand sicher zu realisieren ist, bleibt noch das Problem des regelmäßigen Schlüsselwechsels. Dieser wird von den meisten Benutzern seltener als nötig durchgeführt werden, da dies die aktive Mitarbeit erfordert, und somit sinkt die Gesamtsicherheit des Systems.

Wünschenswert wäre die regelmäßige und transparente Aushandlung eines symmetrischen Schlüssels wie über das Diffie-Hellmann-Merkle-Verfahren innerhalb der bestehen Silc-Verbindung, wenn der Aufbau einer eigenständigen Verbindung nicht möglich ist. Da ein Angreifer jedoch auf einem kompromittierten Zwischenserver diesen Schlüsselaustausch abfangen kann und die Schlüssel gegen seine eigenen austauschen kann, besteht auch hier die Möglichkeit eines Angriffs.

Dieses Problem kann durch die Verwendung eines asymmetrischen Verschlüsselungsverfahren gelöst werden. Dies erfordert aber einen erfahrenen Benutzer, der diese Möglichkeit kennt und aktivieren kann. Davon kann aber im Normalfall nicht ausgegangen werden.

Zusätzlich muss der öffentliche Schlüssel irgendwie von dem Sender zu dem Empfänger gelangen. Da diese Kommunikation nur mit dem *Session Key* geschützt wird, könnte der Angreifer auf einem Server den öffentlichen Schlüssel manipulieren. Wenn nicht im Vorfeld der Fingerprint des öffentlichen Schlüssels über einen unabhängigen und sicheren Kanal ausgetauscht wurde, würden die Benutzer hiervon nichts merken.

Der Vergleich zwischen dem übertragenen und dem zu verifizierenden Schlüssel ist mit weiteren Problemen behaftet. In vielen Fällen wird dieser Vergleich überhaupt nicht oder nur teilweise durchgeführt. Gelingt es einem Angreifer, einen Schlüssel zu erstellen, dessen Fingerprint dem des Kommunikationspartners nur ähnlich sieht, werden viele Benutzer bei einem oberflächlichen Vergleich keinen Unterschied feststellen. Diese Prüfsummen werden als *Fuzzy Fingerprints*[19] bezeichnet.

Einen gewissen Schutz der eigenen Systeme vor Angriffen können Firewallsysteme bieten, die dem Angreifer nur noch beschränkten Zugang zu dem System gewähren. Dies ist aber in der Praxis nicht vollständig realisierbar, da sonst auch normale Benutzer ausgeschlossen werden würden. Des Weiteren sollte diesem Sicherungssystem nicht uneingeschränkt vertraut werden, da es ebenfalls angegriffen werden könnte.

Diese Beispiele haben gezeigt, dass Silc trotz guter Ansätze bezüglich des sicherheitskritischen Designs nicht als absolut sicher angesehen werden kann. Diese Angriffe scheinen dem einen oder anderen sehr weit her geholt zu sein, doch können sie jederzeit in die Praxis umgesetzt werden. Zusätzlich sind stets systemunspezifische Angriffe, wie sie in Kapitel 3 beschrieben wurden, zum Beispiel auf den Benutzer und seine Klienteninstanz möglich.

Dennoch kann Silc als ein sehr sicheres System angesehen werden, da die Mehrheit der Angriffe die Verwundbarkeit anderer, nicht mit Silc im Zusammenhang stehender Systeme voraussetzt.

12.6. Maßnahmen zur Verhinderung der unautorisierten Nutzung in Organisationen

Um die Nutzung von Silc zu hindern, reicht in der Regel das Blocken des TCP-Ports 706 durch einen Paketfilter aus. Es ist aber zu beachten, dass ein Server auch an einem anderen TCP-Port betrieben werden kann.

Um sicher zu gehen, sollte der gestattete Netzverkehr nur über entsprechende Proxy-Server laufen. Wird kein Silc-Proxy eingesetzt, lässt sich damit die Nutzung verhindern. Es sind keine Lösungen bekannt, bei denen beispielsweise ein HTTP-Proxy für Silc genutzt werden kann.

Falls die Benutzer externe www-Seiten aufrufen dürfen, könnten sie jedoch über einen webbasierten Klienten das Konferenzsystem nutzen.

[19] **Vertiefung:** The THC Fuzzy Fingerprints Project - http://www.thc.org/thc-ffp

12.7. Fazit

Durch die Möglichkeit, im lokalen Netz einen Server zu betreiben, stellt Silc eine sichere Alternative zu IRC da. Bei einem Einsatz in Organisationen ermöglicht Silc eine relativ sichere Kommunikation der einzelnen Benutzer. Auch bei der Verwendung von öffentlichen Servern bietet Silc viele Vorteile gegenüber IRC.

Die vorigen Abschnitte haben gezeigt, dass das Silc-Protokoll nicht ganz frei von Schwachstellen ist. Aber im Vergleich zu den anderen, vorgestellten Systemen bietet Silc den stärksten Schutz bezüglich der Sicherheit.

Durch den Einsatz der Ver- und Entschlüsselung direkt bei dem Benutzer ist es für einen Angreifer so gut wie unmöglich, diese Kommunikation zu kompromittieren. Dies wäre, wenn überhaupt, nur bei dem Schlüsselaustausch oder durch einen Angriff der Kryptographieverfahren möglich.

Unter der Voraussetzung einer benutzerseitigen Verschlüsselung mit einem regelmäßig durchgeführten Schlüsselaustausch und eines regelmäßigen Updates der Software kann Silc in Umgebungen mit einem hohen Schutzbedarf eingesetzt werden.

Für Benutzer, die sich mit einem mittleren Schutzbedarf zufrieden geben, ist Silc ohne großen Aufwand einsetzbar und somit als absoluter Favorit anzusehen. Durch den Einsatz eines lokalen Servers kann in Organisationen die Kommunikation kontrolliert oder mit Paketfiltern unterbunden werden.

Der einzige Grund, der gegen einen Einsatz von Silc als Kommunikationssystem spricht, ist die Tatsache, dass es sich nicht um ein *Instant Messaging* sondern um ein Konferenzsystem handelt. Aber wenn die Kommunikation mit mehreren Gesprächspartner gleichzeitig im Vordergrund steht, ist Silc die beste Wahl.

13. Zusammenfassung

13.1. State of the Art

Änderungen an Protokollen und offizieller Klientensoftware

ICQ 5 Im Februar 2005 wurde die Version 5 des offiziellen ICQ-Klienten veröffentlicht. Die Neuerungen beziehen sich aber hauptsächlich auf die Optik und die Benutzerfreundlichkeit des Klienten - das OSCAR-Protokoll unterscheidet sich kaum von dem in der Analyse betrachteten.

Aus netztechnischer Sicht hat sich primär bei der Übertragung von Dateien einiges geändert: da die Datei dem Empfänger über eine separate TCP-Verbindung zugestellt werden muss, ergaben sich insbesondere für Firewalls Probleme. Neu ist, dass die Festlegung des zu nutzenden TCP-Ports nun offiziell unterstützt wird. Somit kann eine bestehende Firewall besser angepasst werden. Da die Gefährdung von Dateitransfers - es kann schadhafte Software übertragen werden - mit der von E-Mail identisch ist, wurde diese in dieser Studie nicht berücksichtigt. Daher erfordern diese Änderungen keine Anpassung der Protokollanalyse.

Als optische Neuerung lässt sich die Software eleganter in das Betriebssystem *Microsoft Windows* integrieren. Durch „*Xtraz*“ können weitere Funktionalitäten nachträglich eingebunden werden.

Weiterhin ist nun eine *Push2Talk*-Funktion integriert. Diese ermöglicht, wie bei Gesprächen über ein Funkgerät, eine Sprachübermittlung, wenn der Sender eine entsprechende Taste betätigt. So kann ein bandbreitenschonende Übermittlung durchgeführt werden.

MSN Messenger 7 Am 7. April 2005 wurde nach einer Beta-Phase der *MSN Messenger* mit der Versionsnummer 7 veröffentlicht. Das analysierte Protokoll ist hiervon aber nicht betroffen, da diese Aktualisierung der „Personalisierung“ dient und weitere visuelle Reize mit sich bringen soll. Beispielsweise soll die Aufmerksamkeit der Benutzer, denen das im Vordergrund erscheinende Fenster nicht ausreicht, über vibrierende Fenster („*Nudge*“) erregt werden, wenn eine neue Mitteilung eintrifft.

Zusammen mit der neuen Version stellt Microsoft den Benutzern einen neuen Dienst Namens *MSN Spaces* zur Verfügung. Neben der Möglichkeit, Online-Tagebücher zu führen (*Blogs*), können Dateien wie Fotos und Musiklisten online gestellt werden. Der neue *MSN Messenger* integriert diesen Dienst, damit Informationen hierüber leichter zwischen den Benutzern ausgetauscht werden können. Ein Beispiel wäre eine Diashow, bei der der Benutzer die Bilder direkt über *Instant Messaging* kommentieren kann.

Microsoft Office Live Communication Server 2005

Um *Instant Messaging* für Organisationen attraktiver zu gestalten, plant Microsoft für Mitte 2005 die Veröffentlichung des *Microsoft Office Live Communication Server 2005*. Mit dem zugehörigen Klienten *Microsoft Office Communicator 2005* soll es möglich werden, sowohl E-Mails und *Instant Messaging*-Nachrichten zu empfangen als auch Termine abzugleichen, über *Voice Over IP* zu kommunizieren oder Videokonferenzen zu führen. Die benötigte Serversoftware kann käuflich erworben und zentral in einem lokalen Netz installiert werden.

Der Klient soll sich dabei in das Softwarepaket *Microsoft Office* integrieren lassen, womit die Benutzerfreundlichkeit erhöht werden soll. Außerdem ist eine Integration von ICQ, AIM und dem *Yahoo Messenger* geplant. Eine solche wurde bisher von den Betreibern abgelehnt. Dies schien sich jedoch nur auf dem privaten Einsatz zu beziehen.

Weitere Informationen sind unter [18] zu finden. Leider veröffentlicht Microsoft dort nur Dokumentation innerhalb von EXE-Applikationen, die von Benutzern anderer Betriebssystemen nur sehr schwer eingesehen werden kann.

13.2. Allgemeines zum IM-Systemen

Die in dieser Diplomarbeit untersuchten Kommunikationssysteme unterscheiden sich in vielen Punkten voneinander. Ein direkter Vergleich der Systeme ist in Tabelle 3 zu finden.

Die vorgestellten Kommunikationssysteme lassen sich in *Instant Messaging*- und Konferenzsysteme einteilen. Für diese Unterteilung nach der **Art des Kommunikationssystems** wurden die Kurzformen *IM* und *Chat* als Bezeichner verwendet. Da nicht alle Betreiber die Protokollspezifikationen veröffentlicht haben, wird in der **Protokollart** unterschieden, ob die Protokolle offen gelegt wurden oder proprietär vom Betreiber verwendet werden.

Unabhängig davon, ob es sich um einen *Instant Messaging*-System oder um ein Konferenzsystem handelt, besteht bei einigen Systemen die Möglichkeit, eine Kommunikation mit mehreren

	TOC	OSCAR	MSN	Jabber	IRC	Silc
Art des Kommunikationssystems	IM	IM	IM	IM	Chat	Chat
Protokollart	Offen	Proprietär	Proprietär	Offen	Offen	Offen
Konferenz	Nein	Ja	Ja	Optional	Ja	Ja
Identifikatoren	Zeichenkette	Zeichenkette bzw. Nummer	E-Mail-Adresse	JID	Zeichenkette	Zeichenkette
Parallele Kontennutzung	nein	nein	nein	ja	nein	nein
Anonyme Statuseinsicht	Nein	ICQ:Ja[a] AIM:Nein	Nein	Optional[b]	entfällt	entfällt
Authentisierung neuer Kontakte	Ja	Dialog	Listen[c]	Listen[d]	entfällt	entfällt
IP-Adresse sichtbar	Ja	Ja	Nein	optional	Ja	Ja
Kontaktlistenverwaltung auf Server	Nein	Optional	Ja	Ja	entfällt	entfällt
Nutzlast	ASCII	Binär	ASCII	ASCII	ASCII	Binär
Dateitransfer	nein	ja	ja	optional	ja	ja/SFTP
Peer-to-Peer	Nein	Ja	Nein	Nein	Ja (DCC)	nur Dateitransfer und Schlüsselaustausch
offizieller Webklient	Nein	Ja[e]	Ja[f]	keine offizielle Lösung	keine offizielle Lösung	keine offizielle Lösung
Multimedia	nein	Bilder, Sprache, Spiele, SMS, Video	Bilder, Sprache, Spiele, SMS, Video	nur durch Erweiterungen	Nein	Viele in MIME-kodierbare Medien
Internationalisierung	Zeichenkodierung durch HTML	Zeichensatzwahl durch Klienten	*Content Type* der MIME-Nachricht	UTF-8	Abhängig von Serverimplementierung	UTF-8

[a] http://www.icq.com/whitepages ermöglicht die Suche nach Benutzern und eine Einsicht in den Status
[b] Der Betreiber des Servers könnte beispielsweise die Statusinformationen auf einem Webserver veröffentlichen
[c] Forward-, Reverse-, Allow- und Blocklist
[d] Subscriptions: None, To, From und Both
[e] **icq2go:** http://www.icq.com/icq2go
[f] **MSN Web Messenger:** http://webmessenger.msn.com

Tabelle 3: Zusammenfassung - Allgemeine Aspekte der Protokolle

Benutzern zu führen, bei der alle Nachrichten an alle Benutzer übermittelt werden (**Konferenz**).

Auch die **Identifikatoren** der Benutzer auf den jeweiligen Systemen unterscheiden sich in ihrer Struktur. Zusätzlich muss berücksichtigt werden, ob sich ein Benutzer mit einem Konto öfters an dem System anmelden und jede Anmeldung separat Nachrichten empfangen kann (**Parallele Kontennutzung**).

Bei der Verwaltung der Kontaktlisten unterscheiden sich die Systeme ebenfalls. Beispielsweise existieren insbesondere bei *Instant Messaging*-Systemen viele Schutzmechanismen, die die Belästigung durch fremde Kontakte vermeiden sollen. Damit aber erwünschte Gesprächspartner mit dem Benutzer in Verbindung treten können, muss eine **Authentisierung neuer Kontakte** erfolgen. Bei einigen Systemen werden die Kontaktlisten auf dem Server, bei anderen lokal von dem Klienten verwaltet (**Kontaktlistenverwaltung auf Server**).

Wird die IP-Adresse des Benutzers vom *Instant Messaging*-System übertragen, so kann dies von einem Angreifer für *Denial of Service*-Attacken ausgenutzt werden. Er könnte zum Beispiel durch *Flooding* (siehe Anhang B.4) versuchen, den Rechner des Nutzers gezielt außer Gefecht zu setzen. Daher wird dieses Verhalten im Punkt **IP-Adresse sichtbar** betrachtet.

Um die Privatsphäre der Benutzer zu schützen, sollte unterschieden werden, ob nur die Kontakte, die durch eine Eintragung in die Kontaktliste autorisiert wurden, den Status des Benutzerss einsehen dürfen (**Anonyme Statuseinsicht**).

Die Protokolle unterscheiden sich im Weiteren durch das Format der TCP-**Nutzlast**. Einige optimieren die Nachrichten durch eine binäre Kodierung, andere übermitteln die Anweisungen in Klartext.

Viele Klienten bieten die Möglichkeit, neben Textnachrichten auch **Dateien** und andere **Multimedia**-Inhalte zu übertragen. Über diesen Weg kann aber schädlicher Programmcode auf das System des Benutzers übertragen und ausgeführt werden. Die Protokollanalyse berücksichtigte nur die klassische Funktionalität (die Übermittlung von Textnachrichten) dieser Systeme.

Um die Kommunikation über den Server zu minimieren, besteht bei einigen Systemen die Möglichkeit, die Nachrichten direkt an den Gesprächspartner zu übermitteln (**Peer-To-Peer**).

Einige Systeme bieten auch die Möglichkeit, das IM-System über einen **offiziellen Webklienten** zu nutzen. Dadurch kann der Betreiber des lokalen Netzes die Nutzung dieses Systems nur schwer verhindern. Außerdem erfordern diese Systeme keine Installation eines Klienten auf dem System des Benutzers.

Soll eine Kommunikation mit einem Gesprächspartner stattfinden, dessen Schriftsprache andere Zeichen benötigt, spielt die **Internationalisierung** eine wichtige Rolle.

13.3. Server

Bei der Auswahl eines Kommunikationssystems spielen die Server eine wichtige Rolle. Einen Vergleich der Systeme unter verschiedenen Gesichtspunkten zeigt Tabelle 4.

Eine wichtige Frage ist die nach **verfügbarer Serversoftware**. Kann beispielsweise ein Server in einem lokalen Netz installiert werden, müssen die Nachrichten zwischen lokalen Klienten das gesicherte Netz nicht verlassen.

Eine weitere Eigenschaft der Server ist die Zwischenspeicherung von Nachrichten, wenn der Gesprächspartner zur Zeit nicht angemeldet ist. In dem Punkt **Nachrichtenempfang, falls offline** ist zu erkennen, welche Systeme diese Funktionalität anbieten.

	TOC	OSCAR	MSN	Jabber	IRC	Silc
Verfügbare Serversoftware	Nein	Nein	Nein	Ja	Ja	Ja
Nachrichten-empfang, falls offline	Ja	Ja	Nein	Ja	Nein	Nein
Freie Serverwahl	Nein. Nur ein Server	Vielzahl von Auth- und BOS-Server	Vielzahl von Servern	Nein	Ja	Ja
TCP-Port	wählbar (9898)	wählbar (5190)	1863	5222 und 5269	194 und 6667	706
Gateway zu anderen IM-Systemen	Nein	Nein	Nein	Ja, Optional	Nein	Nein
Skalierbar	Nein	Ja	Ja	Ja	Ja	Ja
Datenbestand	unbekannt	unbekannt	unbekannt	Abhängig von Im-plementie-rung[a]	intern und nicht persistent	intern und nicht persistent

[a] Jabberd2: PAM,LDAP, MySQL, PostGreSQL und BerkleyDB

Tabelle 4: Zusammenfassung - Server

Auch die **Freie Serverwahl** spielt eine entscheidende Rolle. Wird das System beispielsweise nur von einem Server angeboten, können die Benutzer bei einem Ausfall nicht auf andere ausweichen. Bei anderen Systemen, die beispielsweise Bestandteil eines großes Netzes sind, ist es nicht immer wichtig, auf welchem Server sich der Benutzer anmeldet, wenn er das System nutzen will.

Die Tabelle 4 zeigt ebenfalls einen Überblick, über welchen **TCP-Port** sich die Klienten in der Standardkonfiguration an dem Server anmelden können. Nachdem sich der Benutzer an einen bestimmen System angemeldet hat, ist die Möglichkeit, andere IM-Systeme direkt nutzen zu können, eine interessante Funktionalität. Hierfür muss der Server einen **Gateway zu anderen IM-Systemen** anbieten.

Für den Betrieb eines Servers kann es wichtig sein, wie sehr sich die Aufgaben beispielsweise auf verschiedene Rechner aufteilen lassen (**Skalierbar**). Auch das Format, in dem der **Datenbestand** auf dem Server abgelegt wird, kann für die Auswahl eines Servers entscheidend sein.

13.4. Verschlüsselung und Sicherheit

Tabelle 5 gibt einen Überblick über die Sicherheit in Bezug auf Integrität und Vertraulichkeit. Neben der Verschlüsselung der Nutzlast und dem Schutz der Passwörter spielt die Länge dieser eine wichtige Rolle. Beim Vergleich der Verschlüsselung der Authentisierung und Nutzlast wird die Kommunikation von Server-Server, Server-Klienten und Klienten-Klienten betrachtet.

	TOC	OSCAR	MSN	Jabber	IRC	Silc
Passwortlänge der Klientenkonten	16	ICQ:8 AIM:16	16	abhängig von Server	abhängig von Server	Pass-phrase[a]
Authentisierung Server&Klient	XOR	ICQ:XOR AIM:md5	SSL (fehlerhaft)	SSL/TLS/ SASL	nein	ja, Verfahren wählbar
Nutzlast Server&Klient	Nein	Nein	Nein	SSL/TLS	nein	ja, Verfahren wählbar
Authentisierung Klient&Klient	entfällt	Nein	entfällt	S/MIME[b] & PGP	nein	ja, Verfahren wählbar
Nutzlast Klient&Klient	entfällt	Nein	entfällt	S/MIME[b] & PGP	nein	ja, Verfahren wählbar
Authentisierung Server&Server	unbekannt	unbekannt	unbekannt	SSL/TLS/ Dialbacks	nein	ja, Verfahren wählbar
Nutzlast Server&Server	unbekannt	unbekannt	unbekannt	SSL/TLS	nein	ja, Verfahren wählbar

[a] Länge ist abhängig von Serversoftware und Verschlüsselungsverfahren
[b] Siehe RFC3923, wird aber nicht eingesetzt

Tabelle 5: Zusammenfassung - Sicherheitsaspekte

13.5. Persönliche Bewertung

Tabelle 6 zeigt eine Bewertung der Kommunikationssysteme nach dem Ermessen des Autors dieser Diplomarbeit. Diese Ergebnisse sind weder messbar noch allgemein gültig.

	TOC	OSCAR	MSN	Jabber	IRC	Silc
Sicherheit	- -	ICQ: - - AIM: 0	-	+ +	- -	+ +
Eignung für Organisationen	- -	-	- -	+ +	0	+
Eignung für privater Einsatz	-	+	+	+ +	+ +	+

Legende: - - Absolut ungeeignet, - ungeeignet, 0 zufriedenstellend, + geeignet , + + gut geeignet

Tabelle 6: Zusammenfassung - Persönliche Bewertung

Ein wichtiger Bestandteil dieser Diplomarbeit ist die Sicherheit der untersuchten Systeme. Hier zeigen insbesondere TOC, OSCAR/ICQ und IRC Schwächen, da sie keine der übertragenden Informationen schützen. Die Verschlüsselung des Passworts bei der Übertragung durch den *MSN Messenger* kann sehr leicht angegriffen werden, daher erfolgt eine negative Bewertung.

Da für TOC, OSCAR und MSN keine Server verfügbar sind, eignen sich diese Systeme nicht

für einen Einsatz in Organisationen. Sensible Informationen müssten insbesondere bei MSN wegen der fehlenden *Peer-to-Peer*-Unterstützung das sichere Netz verlassen, womit ein Angriff möglich wird. Diese Angriffe können sich auf die Vertraulichkeit, Verfügbarkeit und Integrität konzentrieren.

Für IRC existieren zwar eine Vielzahl von Servern, aber durch die Sicherheitsdefizite ist von der Verwendung in Unternehmen abzusehen. Im Weiteren bietet IRC keinerlei Vorteile gegenüber Silc.

Bis auf TOC sind alle Systeme für einen privaten Einsatz geeignet. TOC bietet keinerlei Vorteile gegenüber OSCAR, da auch sehr viele freie Klienten auf OSCAR basieren. Für die breite Masse an Benutzern ist neben Jabber, das eine Vielzahl von Protokollen unterstützt, der *MSN Messenger* zu empfehlen. Durch die weite Verbreitung können viele Kontakte angesprochen werden. Im Weiteren lässt sich der Klient sehr gut in das Betriebssystem Windows integrieren, das zum Zeitpunkt der Erstellung dieser Diplomarbeit den Markt anführt.

Teil III.
Anhang

A. Technische Protokollanalyse

Für eine genauere Untersuchung der Kommunikationssysteme spielt deren Funktionsweise eine wichtige Rolle. Um Angriffe auf die Verfügbarkeit, Integrität und Vertraulichkeit ausüben zu können, bedarf es Informationen über den Aufbau der Nachrichten, die zwischen den einzelnen Instanzen ausgetauscht werden. Die Vorschrift, mit der diese Nachrichten gebildet werden, wird in dieser Diplomarbeit als Protokoll bezeichnet.

In der Regel wird nur der Nachrichtenaustausch betrachtet, an dem ein oder mehrere Klienten beteiligt sind. In wenigen Ausnahmen wird auch der Informationsaustausch anderer Systeme, wie zwischen Servern, betrachtet, um hieraus Rückschlüsse auf das Protokoll zu ziehen. Dies ist nur möglich, wenn durch den lokalen Betrieb eines Servers überhaupt Zugriff auf den Informationsaustausch erlangt werden kann.

Im Folgenden werden die Protokolle von ICQ und AIM (OSCAR beziehungsweise TOC), MSN *Messenger*, *Jabber* (XMPP) und Silc betrachtet.

A.1. OSCAR

Das Resultat der Bemühungen, die beiden *Instant Messaging* Systeme ICQ und AIM zu vereinen, war das *Open System for CommunicAtion in Realtime* (OSCAR). Obwohl der Name auf ein freies Protokoll schließen lässt, trifft dies nicht zu. OSCAR ist auch unter der Bezeichnung ICQv7, ICQv8 oder ICQv9 bekannt.

Bis auf wenige Unterschiede ist die Verwendung von OSCAR im *AOL Instant Messenger* (AIM) und bei ICQ gleich. Abweichungen ergeben sich durch die unterschiedliche Struktur der Identifikatoren der beiden Systeme. ICQ benutzt hierfür eine mindestens fünfstellige Zahl, während der AIM-Name mit einem Buchstaben beginnen muss. Weitere Unterschiede gibt es bezüglich der Länge und Übertragung des Passworts. Die Übermittlung des Benutzerstatus ist im vollen Umfang auch nur unter ICQ möglich. Diese Unterschiede beeinträchtigen eine Kommunikation zwischen Klienten beider Systeme jedoch nicht.

Die Protokollspezifikation zu OSCAR wird von AOL unter Verschluss gehalten. Es existieren zwar viele *OpenSource*-Klienten, die OSCAR unterstützen, diese besitzen jedoch keine Lizenz von AOL für die Verwendung von OSCAR. Neben dem AOL-eigenen Klienten hat nur der *Apple iChat* diese von AOL erhalten.

Trotz der Unterverschlusshaltung der Spezifikation konnte diese in verschiedenen Analysen, die auf *Reverse Engineering* beruhen, teilweise ermittelt werden (Beispiel: [7] von Alexandr Shutko). Leider sind diese sehr inkonsistent und daher wurden in dieser Diplomarbeit aus Interoperabilitätsgründen zusätzlich eigene Erkenntnisse durch *Reverse Engineering* gewonnen. Um herauszufinden, welche Datagramme für eine Kommunikation unbedingt notwendig sind, wurde eine eigene Klientensoftware entwickelt. Diese in Perl implementierte Software ist im Anhang C.1 zu finden.

In den folgenden Abschnitten sollen die grundlegenden Funktionen von OSCAR analysiert werden. Dazu werden Begriffe wie *SNAC* oder *FLAP* verwendet, die aus [8] entnommen wurden.

A.1.1. Datagrammformat

Das OSCAR-Protokoll arbeitet wie alle vorzustellenden Systeme in der Verarbeitungsschicht des ISO/OSI- oder TCP/IP-Referenzmodells. Nur diese Schicht wird auch in die genaue Analyse einbezogen. Als Transportprotokoll wird TCP verwendet, die Kommunikation erfolgt in der Regel über den reservierten Port 5190. Dieser ist jedoch nicht zwingend vorgegeben, da die OSCAR-Instanzen auf den Servern bis auf wenige Ausnahmen Verbindungen an allen Ports annehmen können.

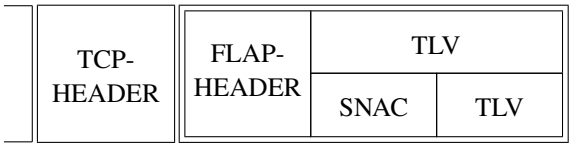

Abbildung 18: OSCAR - Datagrammstruktur

Jedes OSCAR-Datagramm, das von AOL auch als *FLAP* bezeichnet wird, beginnt mit dem Header. Darauf folgt entweder eine *SNAC*-Nutzlast oder ein *Type Length Value* (TLV)-Inhalt. Für den Fall, dass die zu übertragende Nutzlast nicht komplett in das *SNAC*-Feld passt, schließt sich an das *SNAC*-Feld ein weiterer Bereich für externe Nutzlast an. Dieses enthält dann TLV-Daten. Der schematische Aufbau eines OSCAR-Datagramms ist in Abbildung 18 zu finden.

Um das Protokolloverhead zu minimieren, können pro TCP-Datagramm mehrere OSCAR-Datagramme übertragen werden. Diese werden dazu in der TCP-Nutzlast aneinander gereiht. Anhand des Startbyte (0x2a) des OSCAR-Headers und der angegebenen Datagrammlänge können sie beim Empfänger getrennt werden.

Header Der Header ist sehr kurz gehalten (siehe auch Abbildung 19). Er beginnt mit dem Startbyte, das den Wert 0x2a besitzt. Als nächstes Byte folgt der **Channel-Identifikator**, der den Inhalt des Datagramms näher bestimmt. Die möglichen Werte für dieses Feld sind, soweit sie durch *Reverse Engineering* ermittelt werden konnten, in Tabelle 7 zu finden.

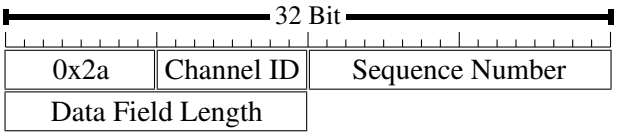

Abbildung 19: OSCAR - Format des Headers

Die **Sequence Number** ist vermutlich der eindeutige Identifikator des *FLAPs*. Die Analyse der Datagramme hat gezeigt, dass die Klienteninstanz bei einer Verbindung zu der Serverinstanz mit der *Sequence Number* 1 beginnt und diese bei jedem gesendeten *FLAP* inkrementiert.

Die Serverinstanz hingegen verwendet einen größeren Startwert. Da bei einer Klienteninstanz bei jedem von der Serverinstanz empfangenen *FLAP* die *Sequence Number* um verschiedene

Wert	Bedeutung
0x01	Aufbau einer neuen Verbindung. Die benötigten Informationen werden dabei in *TLVs* übertragen.
0x02	Dieses Datagramm beinhaltet *SNAC*-Steueranweisungen, die in der *SNAC*-Nutzlast übertragen werden.
0x03	Übermittlung von Fehlermeldungen
0x04	Abbau der bestehenden Verbindung. Beim Abbau der Verbindung zwischen Authentisierungsserver und Klient überträgt der Server Informationen wie die Adresse des BOS-Servers und das *Authorization Cookie*.
0x05	Regelmäßige Überprüfung, ob die Verbindung noch besteht. Es wird keine Nutzlast übertragen.

Tabelle 7: Channelwerte des OSCAR-Headers

Werte erhöht wurde, ist anzunehmen, dass dieser Wert bei jedem versendeten *FLAP* unabhängig vom Kommunikationspartner inkrementiert wird. Dieser Wert ermöglicht anscheinend eine Identifikation unter allen gesendeten *FLAPs*.

Das Feld **Data Field Length** gibt die Größe der folgenden Nutzlast an. Zusammen mit der festen Größe des Headers (sechs Byte) ergibt sich die Gesamtgröße des *FLAP*s. Für diese Größe werden zwei Byte reserviert.

Type Length Value (TLV) Die Übermittlung der Nutzlast erfolgt in *Type Length Value*-Strukturen. In einem *FLAP* können mehrere *TLVs* übertragen werden. Da der Aufbau trivial ist, wurde hier auf eine Grafik verzichtet. Für die Nutzlast werden jeweils zwei Byte zur Angabe der Funktion (**Type**) und der Größe (**Length**) des Inhalts übertragen. Die eigentliche Nutzlast steht dann im **Value**-Feld.

Wert des *Type*-Feldes	Inhalt des *Value*-Feldes
0x0001	Benutzername
0x0002	Passwort
0x0003	Vom Benutzer verwendeter Klient und Zeichensatz
0x0005	IP-Adresse und Portnummer des BOS-Servers
0x0006	Cookie für die Authentisierung am BOS-Server
0x0025	*md5*-Hash des Passworts

Tabelle 8: OSCAR - Beispiele für *Type*-Werte eines *TLV*-Datagramms

Der Wert des *Type*-Feldes gibt die Funktion des übertragenen Inhalts an. Einige Beispiele für die wichtigsten *Type*-Werte sind in Tabelle 8 zu finden.

SNAC *SNAC*-Nutzlasten werden für die Übertragung von Steuerbefehlen zwischen Server- und Klienteninstanzen verwendet. Dies können Anweisungen für die Verwaltung von Kontaktlisten oder für die Übertragung von Benutzerinformationen sein.

Umfangreichere Informationen werden in der Praxis mittels eines *FLAP* mit *SNAC*-Informationen versendet, an das sich TLV-Nutzlasten anschließen.

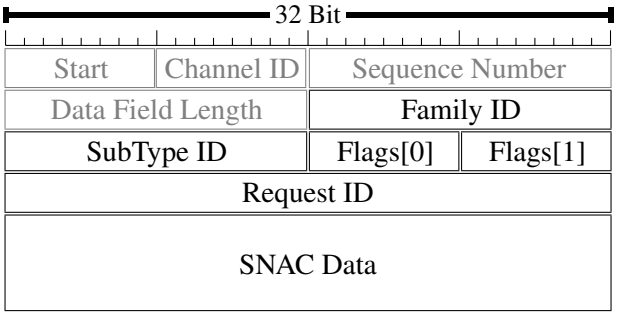

Abbildung 20: OSCAR - Format eines SNAC-Datagramms

Entsprechend Abbildung 20 besteht ein *SNAC* aus dem Startbyte (0x2a), jeweils zwei Byte für die *Family ID* und die *Subtype ID*, jeweils einem Byte für zwei Flagfelder, vier Byte für die *Request ID* und der eigentlichen Nutzlast (*SNAC Data*).

Durch die Felder **Family ID** und **Subtype ID** lassen sich die *SNACs* nach ihrer Funktion in Gruppen einteilen. Einige Beispiele sind in Tabelle 9 zu finden. Ausführliche Listen enthalten die inoffiziellen OSCAR-Spezifikationen [7]. Diese Felder bestimmen auch, ob die übertragenen Informationen im *SNAC Data*-Feld oder in einer oder mehreren *TLV*-Nutzlasten übertragen werden.

Zur Funktion der **Flags** sind keine Informationen bekannt. Im praktischen Einsatz wurde keines der Flags gesetzt und daher konnten auch keine Rückschlüsse auf die Funktion der Flags gezogen werden.

Um Informationen von dem Server zu erhalten, muss die Klienteninstanz diese über *SNACs* anfordern. Alle *SNACs*, die zu einem Informationsaustausch gehören, besitzen die gleiche **Request ID**. Da der Server sehr viel Anfragen verwalten muss, werden hierfür vier Byte reserviert.

A.1.2. Anmeldung

Authentisierung gegenüber login.oscar.aol.com OSCAR sieht eine Trennung zwischen der Authentisierung und der eigentlichen Anmeldung vor. Zunächst muss eine Verbindung zu einer Instanz des Authentisierungsservers aufgebaut werden. Nach einer erfolgreichen Authentisierung erhält die Klienteninstanz die Adresse des Kommunikationsservers und einen Schlüssel zurück.

Family ID	**Subtype ID**	**Bedeutung**
0x0001	0x001e	Ändern des ICQ-Benutzerstatus
0x0004	0x0006	*Outgoing*, Versenden einer Nachricht
0x0004	0x0007	*Incoming*, Empfangen einer Nachricht
0x0017	0x0002	Authentisierung mit *md5*-Prüfsumme
0x0017	0x0006	*Sign-on*, Einleitung *md5*-Authentisierung
0x0017	0x0007	*Sign-on Reply*, Antwort auf *md5*-Authentisierung

Tabelle 9: OSCAR - Beispiele für *Family*- und *Subtype IDs*

Einer der meistfrequentierten Server, der dafür verwendet werden kann, ist unter dem Namen *login.oscar.aol.com* erreichbar. Dieser Server ist vermutlich über ein *Load-Balancing*-System erreichbar, da bei verschiedenen Anmeldungen die IP-Adresse des Servers variierte und dies bei der Vielzahl der Anmeldungen auch vernünftig erscheint. Die Praxis hat gezeigt, dass die Authentisierung über einen nahezu beliebigen TCP-Port erfolgen kann.

Bei der Authentisierung sind zwei Verfahren zu unterscheiden. Bei der Anmeldung mit einer ICQ-Nummer wird die Plaintext-Authentisierung verwendet. Obwohl das übertragene Passwort dabei kodiert übermittelt wird, ist das Ausspähen sehr einfach möglich.

Wird der *AOL Instant Messenger* verwendet, so erfolgt die Authentisierung über eine *md5*-Prüfsumme, die mit Hilfe des Passworts gebildet wird. Dieses Verfahren ist im Vergleich zur Klartextauthentisierung zwar sicherer, aber auch aufwändiger.

Die vom ICQ-Protokoll verwendete **Klartextauthentisierung**, die neben der *md5*-Authentisierung in Abbildung 21 dargestellt wird, ist sehr einfach. Nach dem Verbindungsaufbau zur Serverinstanz sendet diese der Klienteninstanz ein *FLAP* mit dem *Channel*-Eintrag 0x01 (*New Connection*). Dieses Datagramm enthält keine weitere Nutzlast.

Dieses *New Connection*-Datagramm wird von der Klienteninstanz ebenfalls mit einem *New Connection*-Datagramm beantwortet. In den *TLVs* dieses Datagramms werden der Benutzername, das Passwort und Informationen über den Klienten übertragen.

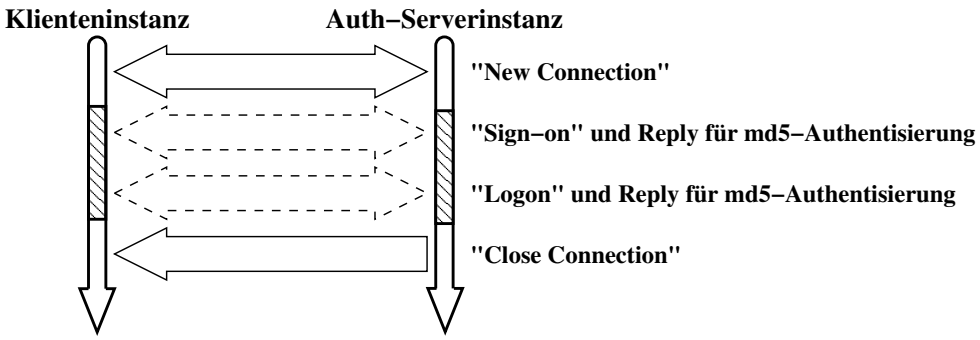

Abbildung 21: OSCAR - Protokollablauf einer Klartext- und *md5*-Authentisierung

Das übertragene Passwort stimmt dabei jedoch nicht mit dem eigentlichen Passwort überein. Durch eine Analyse der Anmeldung mit verschiedenen Benutzerprofilen konnte aber festgestellt werden, dass das übertragene Passwort die gleiche Länge besitzt wie das eigentliche. Nach einigen Tests konnte die Vermutung bestätigt werden, dass sich das übertragene Passwort durch eine XOR-Verknüpfung des eigentlichen Passworts mit einem festen Wert erzeugen lässt.

Da dieser Wert nicht mit übertragen wird, muss er beiden Partnern bereits bekannt sein. Eine erste Vermutung, dass dieser Schlüssel aus den Klienteninformationen gewonnen wird, hat sich durch Tests mit verschiedenen Benutzerprofilen mit dem gleichem Passwort unter Verwendung verschiedener Klienteninstanzen nicht bestätigt.

Die Vermutung, dass der Schlüssel fest in der Software einkodiert ist und von allen Klienten verwendet wird, bewahrheitete sich. Da XOR umkehrbar ist ($a \; xor \; b = c$ und $c \; xor \; b = a$) kann aus dem übertragenen und dem tatsächlichen Passwort der Schlüssel generiert werden. Bei verschiedenen Testreihen wurde damit immer der gleichen Wert ermittelt.

Durch verschiedene Versuche hat sich gezeigt, dass die oben angegebene Reihenfolge der *New Connection-FLAPs* auch vertauscht werden kann. Der Tausch der Reihenfolge ist für die eigentliche Authentisierung irrelevant, da keine der übertragenen Informationen von der anderen abhängig ist. Der selbst entwickelte OSCAR-Klient aus Anhang C.1 verwendet jedoch die erste Variante.

Nachdem die beiden *New Connection-FLAPs* ausgetauscht wurden, schließt der Server die Verbindung mit einem *Connection Closed-FLAP* (0x04). Dieses beinhaltet einige *TLVs*, die den Benutzernamen sowie die Daten des BOS-Servers (IP-Adresse und Portnummer) und ein *Authorization Cookie* enthalten. Dem Klienten steht es frei, ebenfalls mit einem *Connection Closed-FLAP* zu antworten. Mit dem so erhaltenen *Authorization Cookie*, das in der Regel 256 Byte groß ist, kann sich die Klienteninstanz nun direkt an dem BOS-Server anmelden.

Die Authentisierung über eine ***md5*-Prüfsumme** wird nur vom AIM verwendet. Da dieses Verfahren sicherer ist als die Plaintext-Authentisierung, ist es verwunderlich, dass es nicht auch bei ICQ angewendet wird. Der Ablauf dieser Authentisierung ist ebenfalls in Abbildung 21 dargestellt.

Wie auch bei der Plaintext-Authentisierung werden zunächst zwei *New Connection-FLAPs* ausgetauscht. Jedoch enthält hier das Datagramm der Klienteninstanz ebenfalls keine relevante Nutzlast. Im Anschluss daran übermittelt die Klienten- der Serverinstanz den Benutzernamen. Dieser wird in der Nutzlast eines *SNAC* mit dem *Family ID*-Feld „Signon“ (0x0017) und dem *Subtype ID*-Eintrag „Sign-on“ (0x0006) übermittelt.

Dieses wird von der Serverinstanz mit einem *SNAC*, das als *Family ID* ebenfalls den Wert „Signon“ (0x0017) besitzt, bestätigt. Als *Subtype ID*-Feld des *FLAPs* wird der hexadezimale Wert 0x0007 für „Sign-on Reply“ eingetragen. In der Nutzlast übermittelt die Serverinstanz eine vermutlich zufällig erzeugte numerische *Signon challenge*.

Aus der *Signon challenge* und seinem Passwort muss die Klienteninstanz nun eine *md5*-Prüfsumme bilden. Die *Signon challenge* wird für jede Anmeldung jeder Klienteninstanz neu generiert. Dadurch ist es einem Angreifer nicht möglich, die Antwort auf die *Signon challenge* abzufangen, um sie für eine spätere, unabhängige Anmeldung zu verwenden.

Daraufhin antwortet die Klienteninstanz mit einem *SNAC* mit „Signon“ (0x0017) als *Family ID* und zusätzlich als *Subtype ID* „Logon “ (0x0002). Daran anschließend werden *TLV*s mit der *md5*-Prüfsumme (*Value:* 0x0025) und weiteren Informationen über den Klienten übermittelt.

Nun überträgt die Serverinstanz die bei ihr hinterlegten Informationen wie die E-Mail-Adresse des Benutzers, eine URL, unter der er alternativ sein Passwort ändern kann, Versionsinformationen zu dem verwendeten Protokoll, die BOS-Serveradresse und das *Authorization Cookie*. Diese Inhalte, die bei der Plaintext-Authentisierung in einem *Close Connection-FLAP* übermittelt wurden, werden bei der *md5*-Authentisierung in einem separaten *SNAC* („Logon Reply“, 0x0003) übertragen.

Die Authentisierung wird mit einem *Close Connection-FLAP* durch die Serverinstanz abgeschlossen. Die Klienteninstanz kann im Anschluss nun ihrerseits ebenfalls mit einem *Close Connection-FLAP* antworten.

Verbindungsaufbau zum BOS-Server Nachdem die Klienteninstanz ihre Identität gegenüber dem Authentisierungsserver nachgewiesen hat, erhält sie von diesem ein *Authorization*

Cookie, mit dem sie sich nun an einer Instanz des *Basic OSCAR Service* (BOS)-Servers anmelden kann. Hierfür baut die Klienteninstanz eine TCP-Verbindung zu der BOS-Serverinstanz auf, deren IP-Adresse und Portnummer sie ebenfalls von der Instanz des Authentisierungsservers erhalten hat. Dabei wird in der Regel die gleiche Portnummer verwendet wie bei der Verbindung zu der Instanz des Authentisierungsservers.

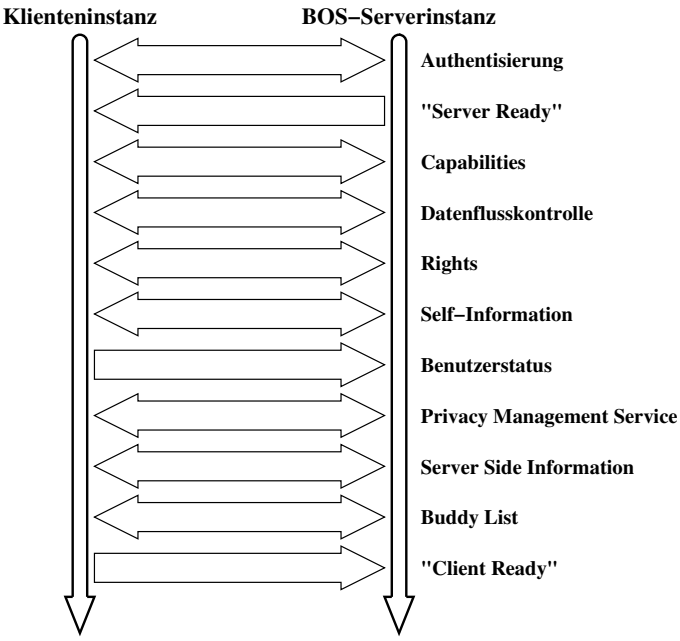

Abbildung 22: OSCAR - Protokollablauf einer Anmeldung am BOS

Zur Anmeldung sind eine Vielzahl von *FLAPs* notwendig. Diese werden in der folgenden Analyse nicht einzeln, sondern gruppenweise betrachtet. Nur solche Datagramme, die für eine sicherheitskritische Betrachtung notwendig sind, werden ausführlich behandelt. Soweit nicht anders angegeben, werden alle Informationen innerhalb einer *SNAC*-Nutzlast übermittelt. Ein schematischer Protokollablauf ist in Abbildung 22 zu finden.

Jeder Informationsaustausch besteht aus einem entsprechenden *Request*-Datagramm der Klienteninstanz, das die Serverinstanz mit den gewünschten Informationen beantwortet. Die Werte der *Family* und *SubType ID* sind [7] zu entnehmen.

Authentisierung Analog zur Authentisierung gegenüber dem Login-Server werden für die Anmeldung jeweils zeitnah ein *New Connection-FLAP* von der Server- und der Klienteninstanz versendet. Das von der Klienteninstanz versendete Datagramm beinhaltet das von der Instanz des Authentisierungsservers erhaltene *Authorization Cookie*.

Dies muss in der Zwischenzeit ebenfalls von der Authentisierungs- zu der BOS-Serverinstanz gelangt sein, so dass dieser die beiden *Authorization Cookies* vergleichen kann. Im Falle einer Übereinstimmung sendet die Serverinstanz der Klienteninstanz ein **Server Ready**-*SNAC*.

Capabilities Im nächsten Schritt tauschen Klienten- und Serverinstanz Listen mit den von ihnen unterstützten *Capabilities* aus. *Capabilities* können die für ein IM-System zwingend benötigte Unterstützung von *Messaging* oder die Verwendung der Kontaktlisten sein. Zusätzlich kann darüber auch die Unterstützung des *Privacy Management Service* oder *Server Side Information* - Dienstes (SSI) übermittelt werden. Diese Dienste benötigten einen zusätzlichen Informationsaustausch, der später beleuchtet wird. Die beiden Datagramme beinhalten neben der eigentlichen Liste auch die Versionsnummer der entsprechenden *Capabilities*.

Des Weiteren kann die Server- der Klienteninstanz die parallele Nutzung von AIM und ICQ anbieten. Die Verfügbarkeit der *Capabilities* ist meist vom verwendeten IM-System abhängig.

Datenflusskontrolle Klienteninstanzen dürfen pro Zeiteinheit nur eine bestimmten Anzahl von Datagrammen senden. Um diese Größe zu regeln, gibt es den Parameter *Rate*. Nach jedem vom Server empfangenen *FLAP* eines Benutzers berechnet die Serverinstanz, unter Berücksichtigung des Zeitabstandes zu den vorigen Datagrammen dieses Benutzers, die *Rate* neu.

Aufgrund dieser *Rate* wird der Klienteninstanz der Status „clean“ (keine Beschränkungen), „limited“ (von der Klienteninstanz werden zur Zeit keine Anfragen mehr angenommen) oder „alert“ (Klienteninstanz wird vor einer Überschreitung ihrer *Rate* gewarnt) zugeordnet. Die Klienteninstanz kann ihre aktuelle *Rate* von der Serverinstanz abfragen.

Rights Im nächsten Schritt werden die Rechte ausgehandelt, die für die Verwaltung der Kontaktliste und des *Server Side Information*-Service (SSI) benötigt werden. Unter anderem überträgt die Server- der Klienteninstanz nach einer entsprechenden Anfrage die Beschränkungen für die Kontakt-, *Invisible*- oder *Visible*-Listen.

Self-Information Daten, die auf dem Server verwaltet werden, können von der Klienteninstanz abgerufen und aktualisiert werden. Bei diesen Daten ist zwischen dynamischen Informationen, die bei der Anmeldung ermittelt werden, und statischen Informationen, die der Server ständig besitzt, zu unterscheiden.

Beispiele für statische Informationen sind der Name, das Geburtsdatum oder eine alternative Kontaktadresse. Diese Informationen wurden entweder bei der Anmeldung eingegeben oder vom Klienten während einer früheren Anmeldung aktualisiert und sind nur für die Kontaktaufnahme (Adresse, Telefon, E-Mail, ...) durch einen anderen Benutzer relevant. Dynamische Informationen werden während der Anmeldung der Klienteninstanz aktualisiert. Dazu gehört der verwendete Zeichensatz der Klienteninstanz, die verwendete IP-Adresse, die dann als letzte bekannte abgespeichert wird und der Zeitpunkt der letzten Anmeldung an diesem System. Für eine spätere Abfrage ist der Eintrag interessant, wieviele Sekunden der Benutzer schon in dieser Sitzung angemeldet ist oder wann er die letzte Nachricht geschrieben hat.

Zusätzlich kann die Klienteninstanz Informationen hinterlegen, die für den Aufbau einer **Direct Connection** (DC) notwendig sind. Damit wird es anderen Klienteninstanzen möglich, *Peer-To-Peer*-Verbindungen zu ihr aufzubauen. Dafür müssen mindestens die IP-Adresse und die Portnummer, auf der die Klienteninstanz Verbindungen annimmt, angegeben werden. Zusätzlich kann übermittelt werden, wie die Klienteninstanz am Netz angeschlossen ist (direkt oder Art und Adresse des Proxys). Die IP-Adresse, unter der *DC*-Verbindungen angenommen werden, muss

nicht notwendiger Weise mit der übertragenen IP-Adresse der Klienteninstanz übereinstimmen. Dies kann eintreten, wenn die IP-Adresse der Klienteninstanz aus einem privaten Adressbereich stammt. Dann wird als *DC*-Adresse die des am öffentlichen Netz angeschlossenen Systems, zum Beispiel eines NAT-Gateways, angegeben.

Benutzerstatus Damit in allen Kontaktlisten, die den Benutzer enthalten, dessen Status angezeigt werden kann, muss dieser ebenfalls bei der Anmeldung übertragen werden. Der Statuswechsel eines Benutzers wird durch das Versenden eines *Set Status-SNACs* (0x001e) erreicht.

Die Statusmöglichkeiten eines ICQ- oder eines AIM-Benutzers sind sehr unterschiedlich. AIM kennt neben dem Status „Online“ nur den Status „Away“. Dieser wird durch das Setzen einer Abwesenheitsmeldung angezeigt. Kehrt der AIM-Benutzer an seine Klienteninstanz zurück, kann er diese einfach löschen.

Das Setzen des ICQ-Status ist viel komplexer. Die Informationen über den Status werden innerhalb von vier Byte in einer dem *SNAC* folgenden *TLV*-Kombination übertragen. Dabei sind die ersten beiden Byte für zusätzliche Informationen und die anderen beiden für den ICQ-Status vorgesehen.

Diesen Status sendet die Klienteninstanz ohne Aufforderung an die Serverinstanz. Folgende Werte können dabei übertragen werden: „online“ (0x0000), „away“ (0x0001), „do not disturb (DND)“ (0x0002), „not available (N/A)“ (0x0004), „occupied (BUSY)“ (0x0010), „free for chat“ (0x0020) oder „invisible“ (0x0100).

Zusätzlich kann das Verhalten bei einer Kontaktaufnahme durch einen unbekannten Benutzer definiert werden. Beinhalten die verbliebenen zwei Byte beispielsweise den Wert 0x1000, kann ein neuer Kontakt den Benutzer nur mit dessen Erlaubnis in seine Kontaktliste eintragen.

Nur wenn der Benutzer auf der Kontaktliste steht, erfährt der Kontakt dessen Status. Dies ist möglich, da bei der Anmeldung die Kontaktliste auf den Server übertragen wird und dieser somit weiß, welche Benutzer er über den Status informieren darf.

Der Übermittlung dieses Datagramms kann auch unabhängig von der Anmeldung bei einem späteren Statuswechsel erfolgen.

Privacy Management Service Während der Anmeldung am BOS-Server kann der *Privacy Management Service* durchgeführt werden. Dabei werden dem Server die *Invisible List* beziehungsweise die *Visible List* des Benutzers übergeben.

Die *Visible List* beinhaltet die Kontakte, die den Benutzer sehen können, wenn dieser den Status *invisible* hat. Alle in der *Invisible List* Aufgeführten können den Benutzer, auch wenn er einen „Online“-Status besitzt, nicht in ihrer Kontaktliste sehen. Ziel ist es in beiden Fällen, dass bei unerwünschten Kontakten der Benutzer als „offline“ erscheint.

Die Praxis hat gezeigt, dass bei der Anmeldung entweder nur die *Visible List* oder die *Invisible List* übertragen wird. Dies ist abhängig vom Status des Benutzers. Wenn sich der Benutzer mit dem Status *invisible* anmeldet, wird nur die *Visible List* übertragen. Bei einem „Online“-Status wird die *Invisible List* übertragen.

Ändert der Benutzer nach dem Anmelden seinen Status von *invisible* auf *Online*, so wird die *Invisible List* erst zu diesem Zeitpunkt übertragen. Das selbe gilt für den Wechsel auf den *invisible*-Status, bei dem die *Visible List* übermittelt wird.

Server Side Information Für Benutzer, die aus bestimmten Gründen sehr oft ihre Klienten wechseln müssen, besteht die Möglichkeit, ihre Kontakte direkt auf dem Server abzulegen. Dies wird mit Hilfe des *Server Side Information* (SSI) - Dienstes realisiert.

Um SSI nutzen zu können, muss die Kontaktliste auf den BOS-Server übertragen und dort gespeichert worden sein. Das Ablegen auf dem Server geschieht jedoch automatisch, sobald der Server die Kontaktliste erhält.

Für die Verwendung von *SSI* sendet die Klienteninstanz einen *Request* an die BOS-Serverinstanz, die mit der Liste der Identifikatoren der Kontakte antworten. Diese Liste kann nun die Klienteninstanz auswerten und die ihr noch nicht bekannten Kontakten zu ihren lokalen Listen hinzufügen.

Buddy Lists Nach der eigentlichen Anmeldung kann nun die Klienteninstanz ihre Kontaktliste an die Serverinstanz übertragen. Diese Liste kann sie lokal abgespeichert oder von SSI-Dienst empfangen haben. Die Serverinstanz überprüft den Status der jeweiligen Kontakte und gibt ihn für die angemeldeten Benutzer zurück.

Die eigentliche Anmeldung wird mit dem Versenden eines **Client Ready**-*FLAPs* abgeschlossen. Tests haben gezeigt, dass die lizenzierten Klienten dieses *FLAP* nicht übertragen. Wieso die restlichen Klienten dieses *FLAP* benötigen, konnte nicht bestimmt werden.

A.1.3. Einfügen eines Benutzers in die Kontaktliste

Um auf den ersten Blick erkennen zu können, ob jemand für einen Informationsaustausch zur Verfügung steht, kann diese Person in die Kontaktliste eingetragen werden. Sobald sich dieser Kontakt bei dem benutzten IM-System anmeldet, würden alle Benutzer, die diese Person auf ihrer Kontaktliste stehen haben, dies angezeigt bekommen.

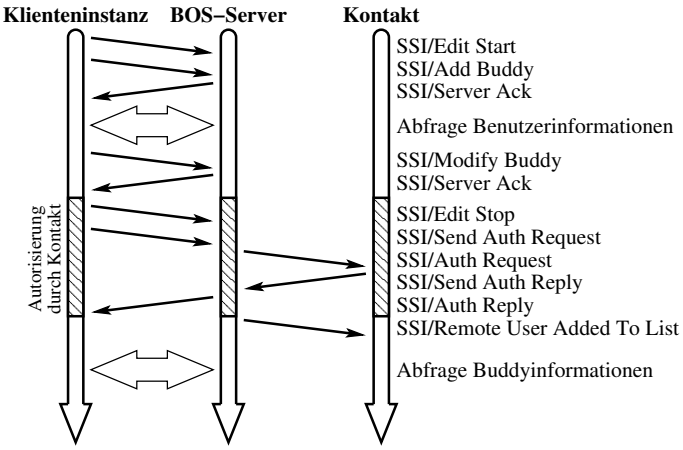

Abbildung 23: OSCAR - Protokollablauf für das Einfügen in die Kontaktliste

Diese Funktionalität besitzt allerdings auch Nachteile für die Kontakte in der Kontaktliste, da deren Benutzerstatus immer sichtbar ist. Einen gewissen Schutz davor bietet eine Einstellung

im Klienten, dass Benutzer eine Erlaubnis des Kontakts benötigen, um ihn in ihre Kontaktliste einfügen zu können.

Das verwendete Authentisierungsverfahren wird von beiden IM-Systemen (ICQ oder AIM) unterstützt, jedoch bieten nicht alle Klienten (zum Beispiel der Original-AIM-Klient) diese Möglichkeit an.

Daher wird im Folgenden zwischen dieser Möglichkeit und der, bei der jeder beliebige Kontakt den Benutzer in seine Kontaktliste eintragen darf, unterschieden. Das Ablaufdiagramm für beide Möglichkeiten ist in Abbildung 23 zu finden.

Alle hierfür benötigten Nachrichten sind der *Server Side Information* (SSI, 0x0013)-*SNAC*-Familie zuzuordnen. Bei beiden Verfahren sendet der Benutzer, der einen Kontakt in die Kontaktliste einfügen will, eine Mitteilung, dass er Änderungen vornehmen will. Hierzu nutzt er ein „Edit Start“-*SNAC* (0x0011). Diesem *FLAP* folgt eine entsprechende „Add Buddy“-Anfrage (*Subtype*: 0x0008) an die Serverinstanz. Als Nutzlast wird der Identifikator des Kontakts angegeben.

Die Serverinstanz bestätigt dies nun mit einem „Server Ack“ (0x000e). In diesem Datagramm besitzen die letzten beiden übertragenen Byte (*result code*) eine besondere Bedeutung, die in Tabelle 10 zu finden ist.

Wert	**Bedeutung**
0x0000	Es sind keine Fehler aufgetreten
0x0002	Benutzer wurde nicht gefunden
0x0003	Eintrag ist schon vorhanden
0x000A	Ein Fehler ist aufgetreten
0x000C	Maximale Kontaktanzahl erreicht
0x000D	ICQ-Kontakt wurde in AIM-Liste eingefügt
0x000E	Nicht eingefügt, da Authentisierung benötigt wird

Tabelle 10: OSCAR - *result codes*

Wenn es sich bei dem Kontakt prinzipiell um einen ICQ-Kontakt handelt, versucht die Klienteninstanz, weitere Informationen zu dem Kontakt zu erhalten. Dies geschieht analog zu der in Anhang A.1.4 vorgestellten Methode.

Danach wird bei beiden Verfahren ein „Modify Buddy“-*SNAC* (0x0009) gesendet, das wiederum mit einem „Server Ack“ bestätigt wird. Wird von dem Kontakt keine Erlaubnis gefordert, so kann die Eintragung in die Kontaktliste mit einem „Edit Stop“-*SNAC* (*Subtype ID*: 0x0012) abgeschlossen werden.

Danach benachrichtigt die Serverinstanz den Kontakt darüber, dass er auf der Kontaktliste des Benutzers eingetragen wurde mit einem *SSI-SNAC* mit der *Subtype ID* „0x001c“ (*Remote User Added Client To List*) und sendet die Statusinformationen des neuen Kontakts an die Klienteninstanz. Andernfalls unterbricht die Klienteninstanz die Eintragung zunächst mit einem „Edit Stop“-*SNAC* (*Family ID* „0x0013“), um sich die Erlaubnis von dem Kontakt einzuholen.

Authentisierung für eine Kontaktlisteneintragung Um die benötigte Erlaubnis des Kontakts einzuholen, sendet die Klienteninstanz ein „Send Authentication Request“ (0x0013)

aus der *SSI-Family* an die Serverinstanz. Dieser *Request* kann neben dem Identifikator des Kontakts auch eine persönliche Nachricht enthalten.

In der Praxis können nun zwei Verfahren für die Kommunikation zwischen der Serverinstanz und dem Kontakt angewendet werden. Das genutzte Verfahren wird von dem Server vermutlich in Abhängigkeit von der Klienteninstanz des Kontakts ausgewählt. Neben dem Einsatz von *SSI-SNACs* kann die Anfrage auch in gewöhnlichen *SNACs* für Textnachrichten übermittelt werden.

Für das *SSI*-basierte Verfahren übermittelt die Serverinstanz dem Kontakt ein „Authentication Request“ (*Family ID*: 0x0013, *Subtype ID*: 0x0019). Neben dem Identifikator des Benutzers, der den Kontakt auf seine Kontaktliste eintragen möchte, wird im Weiteren eine persönliche Nachricht übermittelt. Nun hat der Kontakt die Möglichkeit, die Anfrage abzulehnen oder ihr zu entsprechen. In beiden Fällen antwortet dieser mit einem „Send Authentication Reply“-*SSI-SNAC* (0x001a). Dieses beinhaltet den Identifikator des Benutzers, auf den sich diese Antwort bezieht und ebenfalls eine optional vom Kontakt verfasste Begründung. Das auf den Identifikator direkt folgende Zeichen ist vom besonderen Interesse. Enthält es eine „1“ wird der Kontakt gestattet, andernfalls abgelehnt.

Das andere Verfahren wird mittels *SNACs*, mit denen auch Textnachrichten zwischen den Benutzern versendet werden, realisiert. Hier erhält der Kontakt von der Serverinstanz ein *Incoming-SNAC* mit dem *Message Type* „MTYPE_AUTHREQ“ (0x06, siehe hierfür Abschnitt A.1.5). Dieses Datagramm enthält die optionale Nachricht, wer und warum derjenige ihn auf seiner Kontaktliste haben möchte. Der Kontakt kann die Erlaubnis nun erteilen oder verweigern.

Hierüber informiert er den Server mit einer *Outgoing*-Nachricht. Neben einem Text, den der Kontakt als Begründung nutzen kann, beinhaltet der *Message Type* den Wert „0x07“ (abgewiesen) oder „0x08“ (Erlaubnis erteilt).

Wurde das Verfahren abgeschlossen, informiert die Serverinstanz den Benutzer über das Ergebnis. Hierfür versendet die Serverinstanz ein „Authentication Reply“-*SSI-SNAC* (0x001b), mit dem Identifikator des Kontakts und einem dem Kontakt direkt folgenden Zeichen, ob die Anfrage abgewiesen oder die Erlaubnis erteilt wurde und der optionalen Begründung des Kontakts.

Die Eintragung wird nun wie in dem oben beschriebenen Fall ohne Authentisierung abgeschlossen.

A.1.4. Statusverwaltung

Anmeldung eines Kontakts aus der Kontaktliste Meldet sich ein Kontakt aus der Kontaktliste an, muss die Klienteninstanz des Benutzers darüber informiert werden. Erst durch diese Benachrichtigung sieht der Benutzer den Status des Kontakts.

Zu vergleichen ist dieser Vorgang mit dem Anmelden des Benutzers, bei dem die Informationen der anderen, aktiven Kontakte übertragen werden. Der BOS-Server versendet dazu an alle Kontakte des Benutzers eine entsprechende Nachricht.

Für die eigenständige Benachrichtigung wird eine *SNAC*-Nutzlast aus der *Buddy List*-Familie (0x0003) versendet. Das Datagramm besitzt den *Subtype Incoming Buddy* (0x000b) und die eigentliche Nutzlast wird in *TLVs* übertragen. Die Informationen bestehen hauptsächlich aus dem *Online Status*, der IP-Adresse des am öffentlichen Netz angeschlossenen Systems und aus der *DC Info*.

Die *Direct Connection Information* wird für *Peer-to-Peer*-Verbindungen benötigt. Über diese IP-Adresse und die Portnummer des Kontakts kann der Benutzer eine direkte Verbindung zu diesem aufbauen. Bei einer solchen Verbindung wird der Server für den Informationsaustausch nicht mehr im vollem Umfang benötigt.

Explizite Abfrage von ICQ-Benutzerinformationen Jedem ICQ-Benutzer steht es frei, persönliche Informationen auf dem ICQ-Server zu hinterlegen. Dies kann nützlich sein, um von anderen Benutzern gefunden zu werden. Diese Informationen können sowohl bei der Registrierung als auch zu einem späteren Zeitpunkt übergeben werden.

Für AIM-Benutzer stehen ähnliche Routinen zur Verfügung. Da diese Informationen jedoch nicht so umfangreich sind, wird primär auf die übertragenen ICQ-Informationen eingegangen.

Zur Abfrage dieser Informationen werden *SNAC*-Datagramme mit dem *Family ID*-Eintrag „ICQ“ (0x0002) verwendet. Dabei initiiert die Klienteninstanz die Abfrage mit einem „Login Request“ (*Subtype ID* (0x0002)), worauf die Serverinstanz mit mehreren „Login Responses“ (*Subtype ID* (0x0003)) antwortet.

Die eigentliche Nutzlast, die in Abbildung 24 skizziert wird, wird in *TLVs* übertragen. Hierbei spielt der Aufbau des *Value*-Feldes eine entscheidende Rolle. Nach einer zwei Byte großen Längenangabe (**data chunk size**) und dem vier Byte großen Eintrag, der den Identifikator des Senders beinhaltet (**request owner UID**), folgt der zwei Byte große Datentyp. Alle in diesem Zusammenhang vorgestellten *TLVs* besitzen hier den Eintrag 0xda07, der für *META_DATA_REQ* steht.

Nach einer zwei Byte großen Sequenznummer (**request sequence number**) wird wiederum ein zwei Byte großer **Data Subtype** übertragen. Dieser spezifiziert die Funktion des *TLVs*. Beispiele hierfür werden in dem folgenden Informationsaustausch vorgestellt. Diesen Informationen folgt die eigentliche **Nutzlast**.

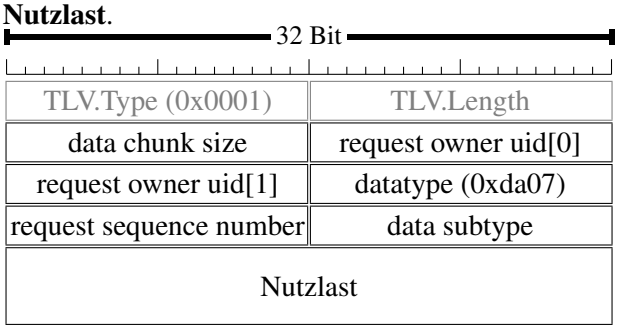

Abbildung 24: OSCAR - Aufbau eines „META_DATA_REQ“-*TLV*

Wie bereits erwähnt, initiiert die Klienteninstanz die Abfrage der kompletten Benutzerinformationen mit einem „Login Request“. Das *TLV* besitzt dabei den *Data Subtype* „META_ FULLINFO_ REQUEST“ (0xB204). Hierbei muss in der Nutzlast der Identifikator des Benutzers, dessen Informationen abgerufen werden sollen, eingetragen werden. Da es sich bei ICQ-Identifikatoren um numerische Werte handelt, kann die komplette Zahl in einen vier Byte großen hexadezimalen Wert transformiert werden. Dieser Wert wird in der *Network Byte Order* (auch bekannt als *Big Endian*), bei der das niederwertigste Byte als erstes übertragen wird, in dem Datagramm verpackt.

Nun überträgt die Serverinstanz mittels einer Vielzahl von „Login Response“-*SNACs* die gewünschten Informationen. Anstatt in die *TLVs* werden die Informationen direkt in das Datenfeld der *SNACs* geschrieben, wobei dieses die gleiche Struktur wie das oben und in Abbildung 24 vorgestellte *TLV* besitzt.

In einem *SNAC* werden grundlegende Informationen („META_BASIC_USERINFO“ beziehungsweise 0xC800) übertragen. Diese umfassen Einträge zu dem Namen, der Adresse, der Zeitzone und weiteren Kontaktmöglichkeiten. Interessant ist zusätzlich das ein Byte große „authorization flag“. Dieses gibt an, ob das Hinzufügen zu einer Kontaktliste einer Autorisierung bedarf. Falls vorhanden und von dem Klient unterstützt, folgen diesem *SNAC* weitere mit privaten und geschäftlichen Daten.

A.1.5. Klientenkommunikation

Kommunikation zwischen zwei Klienteninstanzen über einen Server Die einfachste Form der Kommunikation zwischen Benutzern ist die Verwendung der *Client-to-Server*-Architektur. Hierfür sendet ein Benutzer über eine Klienteninstanz eine Nachricht an die Serverinstanz, die diese im nächsten Schritt an die gewünschten Klienteninstanz weiterleitet.

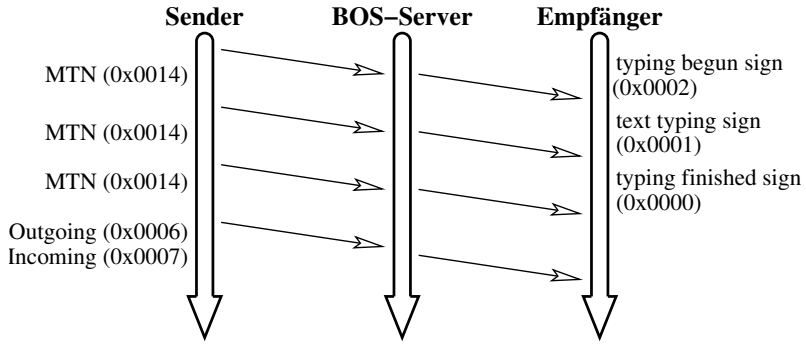

Abbildung 25: OSCAR - Protokollablauf für *Messaging*-Datagramme einer Klientenkommunikation

Die Übermittlung der Nachricht erfolgt in *SNAC*-Nutzlasten. Die Datagramme, die in Abbildung 25 vorgestellt werden, gehören der *SNAC-Family* „Messaging“ (0x0004) an.

Sobald sich der Benutzer entschieden hat, dass er eine Nachricht an einen anderen Anwender senden möchte, das entsprechende Fenster öffnet und ein Zeichen eingibt, wird eine Vorankündigung über die Server- an die gewünschten Klienteninstanz geschickt. In diesem Datagramm, das den *SNAC-Subtype* „Mini typing notifications“ (0x0014) besitzt, wird neben der Nutzlast *typing begun sign* (0x0002) ein vom IM-Protokoll abhängiger Identifikator übermittelt. Bei dem Informationsaustausch zwischen dem Sender und der Serverinstanz wird hierbei der Empfänger eingetragen. Zwischen der Serverinstanz und Empfänger wird der Identifikator des Senders übermittelt.

Nach einiger Zeit teilt die Klienteninstanz dem Empfänger über die Serverinstanz mit, dass die Nachricht zur Zeit noch bearbeitet wird. Hierfür übermittelt sie ein „Mini typing notifications“-*SNAC* (0x0014) mit der Nutzlast „*text typing sign*“ (0x0001) und dem entsprechenden Identifi-

kator. Bricht der Sender das Verfassen der Nachricht ab, wird keine weitere Nachricht in diesem Zusammenhang versendet.

Andernfalls übermittelt der Sender dem Empfänger über die Serverinstanz eine Vorankündigung, dass die Nachricht fertig gestellt wurde. Dieses Datagramm unterscheidet sich bis auf die Nutzlast *typing finished sign* (0x0000) von den vorigen Datagrammen kaum.

Nun sendet der Sender die eigentliche Nachricht über ein *Outgoing-SNAC* an den Server. Außerdem wird neben der Nachricht auch der Identifikator des Empfängers übermittelt. Zusätzlich ist das „*message channel ID*“-Feld von zentraler Bedeutung. Beinhaltet es den Wert „0x0001“, handelt es sich bei der Nachricht um eine ASCII-Textnachricht. Ein weiterer Wert wäre „0x0002“, der auf einen multimedialen Inhalt hinweist, wie Grafik, Audiodatei oder ein Dokument mit einem anderen Zeichensatz.

Die Serverinstanz wiederum sendet ein *Incoming-SNAC*, das die Nachricht enthält, an den Empfänger. Zusätzlich werden noch der Identifikator, der Online-Status des Senders und eine Angabe, wie lange dieser schon angemeldet ist, übertragen.

Die eigentliche Nachricht wird dabei wie bei dem *Outgoing-SNAC* in einer *TLV* versendet. Der *Value*-Anteil besteht dabei aus dem Identifikator des Senders (vier Byte), dem *Message Type* (ein Byte), *Message Flags* (ein Byte), der Länge der Nachricht (ein Byte) und der eigentlichen Nachricht. Eine einfache Nachricht besitzt dabei den *Message Type* 0x01 („MTYPE_PLAIN“).

Das Abweisen von Nachrichten unerwünschter Benutzer wird ausschließlich von der Klienteninstanz geregelt. In praktischen Tests wurde keine Möglichkeit gefunden, dass die Nachrichten von Benutzern, die sich nicht auf der Kontaktliste befinden, direkt von der Serverinstanz abgewiesen werden.

Peer-To-Peer Klientenkommunikation Wenn dies vom Klienten unterstützt wird, kann auch eine direkte Verbindung zu dem Empfänger aufgebaut werden. Beeinflusst wird dies sowohl von der Instanz der Klienten als auch von dem Netz (wenn sich der Empfänger zum Beispiel hinter einem NAT-Gateway befindet). Einige Klienten ermöglichen es auch, die direkte Verbindung nur einzelnen Kontakten zu gestatten.

Im Weiteren benötigt der Sender die öffentliche IP-Adresse und die Portnummer des Empfängers, die er aus den vorher übertragenen *Direct Connection Information* des Empfängers extrahieren kann. Anhand dieser erfolgt eine Überprüfung, ob der Empfänger diese Art von Verbindung unterstützt.

Nun baut der Sender mit diesen Informationen eine eigenständige TCP-Verbindung auf. Prinzipiell werden im ersten Schritt die Identifikatoren der Gesprächspartner ausgetauscht. Zusätzlich war zu erkennen, dass die IP-Adressen der Gesprächspartner ebenfalls übermittelt wurden. Der Sinn dieser Maßnahme ist aber nicht ersichtlich, da diese Adressen schon vorher dem Sender bekannt sein musste und der Empfänger diese ohne die Information ermitteln kann.

Aber auch ohne die Kenntnis des verwendeten Protokolls sind durch das geringen Overhead des Datagramms die übertragenen Textinformationen sofort ersichtlich, da diese in Klartext übermittelt werden.

Die Benachrichtigung, dass eine neue Nachricht erstellt wird, verläuft wie im Anhang A.1.5 vorgestellt, ebenfalls über den Server. Im Gegensatz zu der ausschließlichen Kommunikation über den Server wird das Datagramm mit der Nachricht direkt an die Klienteninstanz übermittelt.

Die bestehende TCP-Verbindung bleibt für eine weitere Kommunikation bestehen. Erst nach einem *timeout* oder einer expliziten Anweisung wird diese geschlossen.

A.2. TOC

Das *TOC*-Protokoll wurde 1998 von AOL veröffentlicht. Obwohl es von AOL stammt, wird dieses Protokoll von keinem AOL-eigenen Klienten verwendet. Vielmehr wurde TOC für *Open-Source*-Entwickler veröffentlicht, damit diese auf legalem Wege freie AOL-Klienten schreiben können. Das Ziel war es, ohne die Veröffentlichung der OSCAR-Spezifikationen viele *Open-Source*-Nutzer für den *AOL Instant Messenger* zu gewinnen.

Obwohl OSCAR für lizenzierte AOL-Klienten vorgesehen ist und TOC für freie Software verwendet werden soll, wird TOC nur sehr selten eingesetzt. Viele freie Klienten nutzen OSCAR als IM-Protokoll, da es mehr Vorteile bietet.

Bei einem Informationsaustausch können nur Textnachrichten übertragen werden - das Versenden von Sprache oder Dateien wird nicht unterstützt. Eine Kommunikation zwischen TOC-Klienten und OSCAR-AIM-Klienten ist problemlos möglich. Dagegen können ICQ-Klienten, die ebenfalls OSCAR verwenden, nicht mit Benutzern kommunizieren, die TOC einsetzen. Da sich die beiden OSCAR-Protokolle von ICQ und AIM nur in sehr wenigen Details unterscheiden, hat die Unterdrückung von ICQ-Klienten vermutlich nur einen AOL-politischen Hintergrund. Darauf deutet auch hin, dass eine nachträgliche ICQ-Unterstützung von AOL nicht geplant ist.

Im Gegensatz zu OSCAR finden die Authentisierung und die eigentliche Nutzung auf einem Server statt. Einer der bekanntesten Server ist *toc.oscar.aol.com*. Obwohl in der Grundeinstellung der Klienten der Port 9898 verwendet wird, lauscht die Serverinstanz wie auch bei OSCAR auf einer Vielzahl von TCP-Ports. Die Nutzung eines Proxy-Servers ist möglich. Die *Peer-to-Peer*-Kommunikation wird von TOC nicht unterstützt.

Die offizielle TOC-Spezifikation ist unter [9] zu finden. Außerdem können erlauschte TOC-Mitschnitte sehr leicht ausgewertet werden, da es sich bei TOC um ein ASCII-basiertes Klartextprotokoll handelt. Auf die Funktion der Datagramme kann auf Grund eines aussagekräftigen Bezeichners, der am Beginn der Nutzlast steht, geschlossen werden. Daher wird im Folgenden der Informationsaustausch nur kurz angerissen. Dabei wurde für die Analyse der unter Abschnitt C.2 zu findende Klient genutzt. Für weiterführende Untersuchungen wurde der Tcl/Tk-basierte Klient TiK[20] verwendet.

A.2.1. Datagrammformat

Header Der Header besitzt den gleichen Aufbau wie der aus Anhang A.1.1 bekannte Header des OSCAR-*FLAPs*. Da das *Channel ID*-Feld des TOC-Headers weniger Werte als das des *FLAP*-Header annehmen kann, wird ein TOC-Datagramm auch als *SFLAP* (Vermutlich: *Simple FLAP*) bezeichnet.

Bei dem Feld für die *Channel ID* wird innerhalb eines TOC-Headers nur zwischen den Werten „SIGNON“ (0x01), „DATA“ (0x02) und „KEEP_ALIVE“ (0x05) unterschieden. Das bei TOC hinzugekommene „SIGNON“ wird für den Aufbau der Verbindung benötigt.

[20] **Verweis auf Applikation:** http://tik.sourceforge.net

SIGNON-Datagramme Die Nutzlast der Datagramme, deren *Channel IDs* im TOC-Header den Wert „SIGNON“ beinhalten, wird nicht ASCII-kodiert übertragen. Die Struktur der Nutzlast ist davon abhängig, ob das Datagramm von der Server- oder von der Klienteninstanz versendet wurde.

In beiden Fällen beginnt die Nutzlast mit einem vier Byte großen Wert für die Versionsnummer. Warum diese optimistische Größe gewählt wurde ist unklar, da seit der Veröffentlichung der Version 1 (0x0001) im Jahre 1998 keine weitere Version erschienen ist.

Das vom Server übermittelte Datagramm enthält keine weiteren Informationen. Wird das Datagramm vom Klient versendet, folgt der Versionsnummer ein zwei Byte großer Wert für eine *TLV*-Markierung. Auf die Funktion dieses Eintrags wird in der Spezifikation nicht näher eingegangen. Bei praktischen Tests wurde nur der Wert 0x01 übertragen.

Der *TLV*-Markierung des Datagramms an die Serverinstanz folgen eine zwei Byte große Angabe der Länge des Benutzernamens und der Benutzername selbst.

A.2.2. Anmeldung

Abbildung 26 zeigt ein Protokollablaufdiagramm für eine Anmeldung an einer TOC-Serverinstanz. Wie bei allen TOC-Anweisungen ist hier der Unterschied zwischen Klartextbefehlen, die von der Server- oder von der Klienteninstanz versendet werden, sofort ersichtlich. Die Befehle der Serverinstanz werden grundsätzlich mit Großbuchstaben übertragen, wogegen die der Klienteninstanz bis auf das *SIGNON* klein geschrieben werden. Einzelne Parameter einer Anweisung werden mit einem ASCII-Leerzeichen („0x20“) voneinander getrennt.

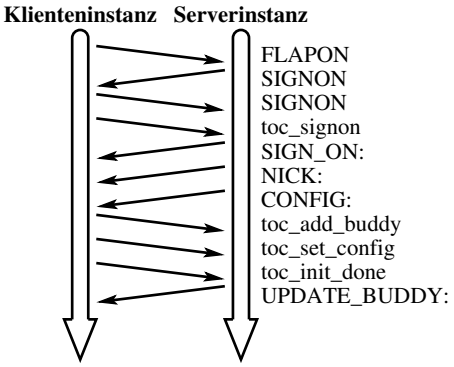

Abbildung 26: TOC - Protokollablauf einer Anmeldung

Wie auch „normale“ *FLAPs* können mehrere *SFLAPs* in einem TCP-Datagramm versendet werden.

FLAPON: Nachdem die TCP-Verbindung zwischen der Server- und der Klienteninstanz aufgebaut wurde, initiiert die Klienteninstanz die TOC-Verbindung durch ein *FLAPON*.

Dieses Datagramm besitzt keinen TOC-Header und besteht insgesamt aus der TCP-Nutzlast „0x 46 4c 41 50 4f 4e 0d 0a 0d 0a“ (FLAPON\r\n\r\n).

SIGNON: Als nächstes übermitteln jeweils die Server- und die Klienteninstanz ein *SIGNON*-Datagramm. Die Struktur dieser Datagramme ist im vorigen Abschnitt A.2.1 beschrieben.

toc_signon In diesem Datagramm überträgt die Klienteninstanz alle für die Authentisierung benötigten Informationen. Die *toc_signon*-Anweisung, die im Klartext übermittelt wird, hat folgenden Aufbau:

```
toc_signon <Netzname des Servers> <Port des Servers> <Benutzername>
           <Passwort> <Sprache> <Klientensoftware>
```

Das Passwort wird als Nutzlast mit führenden „0x“ in hexadezimaler Schreibweise XOR-kodiert übertragen. Wie in der Spezifikation zu TOC zu finden ist, entspricht der Schlüssel der sich wiederholenden Zeichenkette „0x 54 69 63 2f 54 6f 63“ (Tic/Toc).

SIGN_ON: Nachdem die Serverinstanz die bei ihr hinterlegten Authentisierungsdaten mit denen der Klienteninstanz verglichen hat und diese übereinstimmen, sendet die Serverinstanz ein Datagramm mit folgender Struktur:

```
SIGN_ON:<Version des TOC-Protokolls>
```

Schlägt die Authentisierung fehl, übermittelt der Server ein *ERROR*-Datagramm und bricht die Verbindung ab.

NICK: Praktische Tests haben gezeigt, dass während der Anmeldung der Benutzername nochmals separat übertragen wurde. Dies wurde mit folgender Anweisung des Servers umgesetzt:

```
NICK:<Benutzername>
```

CONFIG: Wenn in vorangegangenen Sitzungen Informationen auf dem Server abgelegt wurden, werden diese bei der Anmeldung von der Serverinstanz gesendet.

```
CONFIG:<Einstellungen>
```

Damit kann beispielsweise die Kontaktliste übertragen werden. Im Weiteren können Listen verwaltet werden, die regeln, wer mit dem Benutzer kommunizieren darf und wer ignoriert werden soll. Die detaillierten Beschreibungen sind in der TOC-Spezifikation unter [9] zu finden.

toc_add_buddy Mittels der Anweisung

```
toc_add_buddy <Kontakt 1> [<Kontakt 2> [<Kontakt 3> [...]]]
```

kann der Benutzer dem Server seine Kontakte mitteilen. Die Serverinstanz gibt den Status der Kontakte dieser Liste später zurück.

toc_set_config Möchte der Benutzer seine aktuellen Listen wie beispielsweise die der Kontakte auf dem Server ablegen, kann er dies mit der folgenden Anweisung realisieren:

```
toc_set_config <Einstellungen>
```

Das Format der Einstellungen entspricht dem Format der *CONFIG:*-Anweisung des Servers.

toc_init_done Sind keine Fehler aufgetreten, schließt die Klienteninstanz die Anmeldung mit einer *toc_init_done*-Anweisung ab. Diesem Befehl werden keine Parameter übergeben.

UPDATE_BUDDY: Obwohl diese Anweisung nicht zwingend zu der Anmeldung gehört und jederzeit von dem Klient empfangen werden kann, wird sie an dieser Stelle vorgestellt. Nachdem der Server die Listen mit den Kontakten des Benutzers empfangen hat, muss dieser dem Benutzer mitteilen, wer sich am Server angemeldet hat. Hierfür verwendet dieser für jeden Kontakt ein *UPDATE_BUDDY* mit den folgenden Parametern:

```
UPDATE_BUDDY:<Kontakt>:<Online? T/F>:<Warnung vor Kontakt>:
             <Uhrzeit der Anmeldung>:<Wieviel Minuten Inaktiv?>:
             <UserClass>
```

Ist der Kontakt auf dem Server angemeldet, wird als Onlinestatus „T“ übertragen. Schließt der Kontakt zu einem späteren Zeitpunkt die Verbindung, wird dies dem Benutzer mit dem Onlinestatus „F“ mitgeteilt.

Hat der Kontakt beispielsweise andere Benutzer im Vorfeld belästigt, so können diese eine Warnung vor dem Kontakt aussprechen. Diese Warnungen haben direkten Einfluss auf die Datenrate, mit der dieser Kontakt Nachrichten versenden darf. Weitere Details zu dieser Anweisung enthält die Spezifikation unter [9].

A.2.3. Klientenkommunikation

Der Nachrichtenaustausch zwischen zwei Benutzern geschieht ausschließlich über den TOC-Server. Hierfür übermittelt der Sender die Nachricht an die Serverinstanz, die wiederum die Nachricht an den Empfänger weiterleitet.

Die Nachrichten können beispielsweise mittels HTML optisch gestaltet werden. Diese Gestaltungsmöglichkeiten werden direkt von der Klienteninstanz ausgewertet, wenn diese dies unterstützt. Der Server leitet die Nachrichten in dem vom ihm empfangenen Format weiter.

Versenden Nachrichten werden unter TOC im Klartext mit einem nur sehr geringen Protokolloverhead übermittelt. Für die Übertragung der Nachrichten vom Sender zur Serverinstanz wird die folgende Anweisung in ein TOC-Datagramm verpackt:

```
toc_send_im <Empfänger> <Nachricht> [auto]
```

Das optionale „[auto]“ spezifiziert, ob die Nachricht vom Nutzer selbst oder automatisch erstellt wurde.

Empfangen Für die Nachrichtenübermittlung vom Server zu dem Empfänger wird das folgende *TOC*-Datagramm übermittelt:

```
IM_IN:<Sender>:<Auto Response T/F?>:<Nachricht>
```

Der *Auto Response*-Eintrag gibt an, ob die Nachricht automatisch generiert wurde. Ist dies der Fall, wird der Wert „T“ (True) übertragen, andernfalls „F“ (False).

A.3. MSN Messenger Protocol

Die gesamte Kommunikation findet auf der Verarbeitungsschicht des ISO/OSI- beziehungsweise des TCP/IP-Referenzmodells statt. Zunächst wird eine TCP-Verbindung, die während der gesamten Nutzung bestehen bleibt, zu einem *Notification Server* aufgebaut. Diese Serverinstanz ist unter anderem für die gesamte Verwaltung der *Online Session* zuständig. Die Aufgabe der Identitätsprüfung übernimmt der *Authentification Server*.

Für jede Kommunikation zu anderen Benutzern muss eine eigenständige Verbindung zu einem *Switchboard Server* aufgebaut werden.

A.3.1. Datagrammformat

Alle Steueranweisungen und Nachrichten, die über das *MSN Messenger Protocol* versendet werden, werden in UTF-8 kodiertem Klartext übertragen. In der Regel wird eine Anweisung, die im Folgenden skizziert wird, in einem eigenständigen Datagramm übermittelt.

```
<Anweisung> [<TrID>] <Attribute>
```

Die in dieser Diplomarbeit betrachteten Anweisungen sind in Tabelle 11 zu finden. Jedes Datagramm des *MSN Messenger Protocols* muss mit einer Anweisung beginnen. Diese spezifiziert die Funktion der übertragenen Information.

Viele Datagramme besitzen einen *Transaction Identificator* (TrID), der einen Bezug zwischen einer Anfrage und einer Antwort herstellen soll. Stellt die Klienteninstanz eine Anfrage, antwor-

Anweisung	Bedeutung
ACK	Bestätigung - nur im Zusammenhang mit Nachrichtenversand
ADD	Anweisung oder Bestätigung zum Einfügen eines Kontakts in eine entsprechende Liste
ANS	Informationen zur Authentisierung am *Switchboard Server*
BLP	Umgang mit unbekannten Kontakten
CAL	Mitteilung eines Verbindungswunschs
CHG	Änderung des eigenen Status
CVR	Informationen zu der eingesetzten Klienteninstanz
GTC	Verfahren zu Authentisierung für Eintragung in Kontaktliste
IRO	Abfrage, ob gewünschter Kommunikationspartner noch angemeldet ist
JOI	Benachrichtigung, dass sich der Gesprächspartner angemeldet hat
LSG	Übermittlung der Kontaktliste vom Server zur Klienteninstanz
LST	Übermittlung der Liste mit den Gruppen vom Server zur Klienteninstanz
MSG	Übermittlung einer MIME-Nachricht
NLN	Mitteilung von Status- und Aliasänderung eines Kontakts
REA	Änderung des eigenen Alias
REM	Anweisung oder Bestätigung zum Löschen eines Kontakts von entsprechender Liste
RNG	Benachrichtigung eines Gesprächspartners
SYN	Aktueller Stand der Kontaktlisten. Wird für Abgleich benötigt.
USR	Informationen, die zur Anmeldung benötigt werden (Benutzername, Ticket, ..)
VER	Informationen zum Protokoll
XFR	Übermittlung der Adresse des angegebenen Servers

Tabelle 11: MSN - betrachtete Anweisungen

tet die Serverinstanz mit der gleichen *Transaction ID* (4 Byte).

Die Attribute sind abhängig von der Funktion und somit der Anweisung des Datagramms. Jede Befehlszeile wird mit „$\backslash r \backslash n$“ abgeschlossen.

A.3.2. Anmeldung

In die vollständige Anmeldung am MSN-System sind drei Serverinstanzen integriert. Von zentraler Bedeutung ist der *Notification Server*, der die gesamte Kommunikation steuert.

In der Regel baut der Klient zu Beginn eine Verbindung zu dem *Dispatch Server* auf, um die Adresse des *Notification Server* zu erhalten. Dieser Schritt ist nur notwendig, wenn diese Adresse nicht bekannt ist. Nach dem erfolgreichen Verbindungsaufbau zum *Notification Server* muss der Benutzer seine Identität nachweisen. Dazu wird eine HTTPS-Verbindung zu dem *Authentification Server* aufgebaut. Im Anschluss erfolgt die Synchronisation mit dem *Notification Server*. Ein Protokollablaufdiagramm ist in Abbildung 27 zu finden.

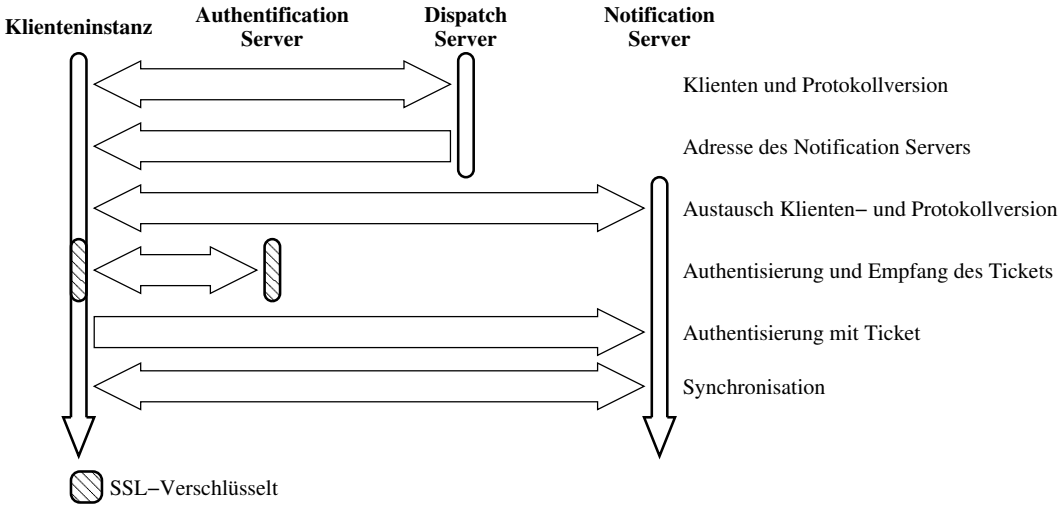

Abbildung 27: MSN - Protokollablauf einer Anmeldung

Informationsaustausch zwischen Klienteninstanz und *Dispatch Server* Im ersten Schritt meldet sich die Klienteninstanz an der Serverinstanz *messenger.hotmail.com* an, um von dieser die Adresse des *Notification Servers* zu erhalten. Praktische Tests haben gezeigt, dass im Gegensatz zu den vorigen IM-Systemen nur der TCP-Port 1863 verwendet werden kann.

Bis auf den Benutzernamen läuft dieser Dialog immer gleich ab. Die Klienten- und die Serverinstanz wechseln sich mit der Übermittlung der Anweisungen ab, wobei die Klienteninstanz beginnt.

```
bash-2.05b$ telnet messenger.hotmail.com 1863
Trying 207.46.104.20...
Connected to messenger.hotmail.com.
Escape character is '^]'.
VER 0 MSNP9 MSNP8 CVR0
VER 0 MSNP9 MSNP8 CVR0
CVR 1 0x0409 winnt 5.1 i386 MSNMSGSR 6.0.0602 MSMSGS im-studie@web.de
CVR 1 6.2.0137 6.2.0137 6.1.0211 http://download.microsoft.com/download/
      0/9/a/09a79e99-f2c8-4933-9797-77e88836bb52/SetupDL.EXE http://mess
      enger.msn.com
USR 2 TWN I im-studie@web.de
XFR 2 NS 207.46.106.110:1863 0 207.46.104.20:1863
Connection closed by foreign host.
```

In den ersten beiden „VER“-Anweisungen (*Version*) folgen auf die *Transaction ID* die Versionsnummern der MSN-Protokolle, die Klient und Server unterstützen (hier: 8 und 9). Die genaue Bedeutung des *CVR0*-Protokolls ist nicht bekannt. Die spätere Verwendung der *Consumer Versioning*-Anweisung (CVR) lässt vermuten, dass dieses Protokoll hierfür benötigt wird.

Die *Consumer Versioning*-Anweisung der Klienteninstanz übermittelt die *Transaction ID*, die hexadezimale *Local ID*[21], das Betriebssystem mit Version, die Hardwarearchitektur, die MSN-Klientensoftware mit Version und einen weiteren *MSMSGS*. Dies ist die Bezeichnung für den offiziellen *Instant Messenger* von Microsoft. Die genaue Funktion dieses Parameters konnte nicht ermittelt werden. Der letzte Parameter entspricht dem Benutzernamen.

Die ersten drei Parameter der Serverantwort stehen im Zusammenhang mit der Klientensoftware. Diesen Werten folgt die Downloadadresse des aktuellsten Klienten und eine Adresse mit allgemeinen Informationen.

In der „USR“-Anweisung folgt der *Transaction ID* das gewünschte Authentisierungsverfahren (hier: *Tweener*). Daran schließen sich ein „I“ (für Initialisierung der Anmeldung) und der Benutzername des zu Authentisierenden an.

Für die *Tweener*-Authentisierung muss die Identität einer anderen Serverinstanz nachgewiesen werden. Ein weiteres Authentisierungsverfahren ist *md5*, das in der Praxis aber nicht verwendet wird.

Bevor die TCP-Verbindung zum *Dispatch Server* geschlossen wird, übermittelt dieser der Klienteninstanz eine „XFR“-Anweisung. Diese beinhaltet nach der *Transaction ID* den Servertyp (hier „NS“ für *Notification Server*). Der IP-Adresse und TCP-Portnummer des *Notification Server* folgt immer eine Null. Diese Anweisung wird durch die IP-Adresse und die Portnummer der aktuellen Serverinstanz abgeschlossen.

Austausch von Benutzer-, Klienten- und Protokollinformationen zwischen der Klienteninstanz und dem *Notification Server* Nachdem die Klienteninstanz eine TCP-Verbindung zum *Notification Server* aufgebaut hat, werden wie nach der Anmeldung an dem *Dispatch Server* „VER“- und „CVR“-Anweisungen ausgetauscht. Diese Operationen unterscheiden sich bis auf die fortlaufende *Transaction ID* nicht von den vorherigen.

Dem folgt eine „USR“-Anweisung der Klienteninstanz, die bis auf eine höhere *Transaction ID* mit der „USR“-Anweisung identisch ist, die der Klient an den *Dispatch Server* geschickt hat.

[21] *Local IDs*: http://msdn.microsoft.com/library/en-us/script56/html/vsmsclcid.asp

Diese Anweisung enthält ebenfalls Informationen über das Authentisierungsverfahren.

Die Serverinstanz antwortet hierauf mit einer neuen „USR“-Operation. Diese besitzt neben der *Transaction ID* und dem Authentisierungsverfahren einen Eintrag, dass es sich um eine der Initialisierung folgende Anweisung („S“ für *subsequent*) handelt. Diese Anweisung wird durch eine Zeichenkette abgeschlossen, die für eine spätere Authentisierung benötigt wird.

```
USR <TrID> TWN S <Zeichenkette für Authentisierung>
```

Diese Zeichenkette hat prinzipiell den Aufbau „Bezeichner=Wert“, wobei die Eintragungen mit einem Komma voneinander getrennt werden. Ein Eintrag entspricht der Adresse des Authentisierungssserver.

SSL-Authentisierung zwischen Klienteninstanz und Authentisierungsserver Die Übermittlung des Identitätsnachweises in Form eines Passworts geschieht verschlüsselt über eine *Secure Sockets Layer*-Verbindung (SSL, siehe auch Anhang B.15). Um diesen Nachrichtenstrom im Klartext analysieren zu können, wurde eine *Man-In-The-Middle-Attack* durchgeführt. Die Durchführung dieses Angriffs ist im Kapitel 6.3 beschrieben.

Für die Authentisierung sendet die Klienteninstanz dem Server eine HTTP-*GET*-Anfrage, die dieser entsprechend beantwortet. Alle benötigten Informationen werden im HTTP-Header übertragen.

Für das *GET* des Klienten ist das „Authorization“-Feld von zentraler Bedeutung. Die mit Komma getrennten Eintragungen besitzen die Gestalt „Bezeichner=Wert“. Neben Teilen der Zeichenkette aus der *USR*-Antwort des *Notification Server* werden der Benutzername („sign-in=“) und das Passwort („pwd=“) übertragen.

```
GET /login2.srf HTTP/1.1\r
TE: deflate,gzip;q=0.3\r
Connection: Keep-Alive, TE, close\r
Cache-Control: no-cache\r
Authorization: Passport1.4 OrgVerb=GET,<Zeichenkette mit pwd=<passwort>>
Host: login.passport.com\r
User-Agent: MSMSGS\r
\r
```

Das Passwort wird hierbei im Klartext innerhalb der verschlüsselten SSL-Verbindung übertragen - ein weiterer Schutz ist nicht vorgesehen.

Nachdem die Serverinstanz die Identität des Benutzers überprüft hat, sendet diese die HTTP-Antwort. In dieser Antwort ist besonders das „Authentication-Info“-Feld des Headers von zentraler Bedeutung. Der Bezeichner „t=“ spezifiziert ein Ticket. Dieses Ticket wird von dem Klienten für die Authentisierung gegenüber dem *Notification Server* benötigt.

```
HTTP/1.1 200 OK\r
Connection: close\r
Date: Wed, 08 Dec 2004 13:47:38 GMT\r
Server: Microsoft-IIS/6.0\r
PPServer: PPV: 25 H: BAYPPLOG2A03 V: 1113\r
Content-Type: text/html\r
Expires: Wed, 08 Dec 2004 13:46:38 GMT\r
Cache-Control: no-cache\r
cachecontrol: no-store\r
Pragma: no-cache\r
P3P: CP="DSP CUR OTPi IND OTRi ONL FIN"\r
Set-Cookie: <Zeichenkette mit mehreren Information, die
             für die Sicherheitsanalyse vernachlässigt werden können>
Authentication-Info: Passport1.4 <Zeichenkette u.A. mit t=<Ticket>>
Content-Length: 0\r
```

Nachdem die *HTTP-GET*-Anforderung mit dem Ticket beantwortet wurde, wird die SSL-Verbindung zu dem Authenisierungsserver abgebaut.

Authentisierung gegenüber dem *Notification Server* Nachdem die Klienteninstanz das Ticket von dem Authentisierungsserver erhalten hat, muss sie dieses dem *Notification Server* übermitteln. Erst nachdem der Klient diesem seine Identität nachgewiesen hat, kann die Klienteninstanz das MSN-System nutzen.

Hierfür verwendet der Klient eine „USR“-Anweisung, die über die bestehende Verbindung übermittelt wird. Da im Vorfeld eine *Tweener*-Authentisierung gewählt wurde, folgt der *Transaction ID* das Schlüsselwort „TWN“. Nach einem schon aus dem vorigen Datagramm vom Server bekannten „S“ folgt der Bezeichner „t=“, der als Wert das vorher übertragene Ticket beinhaltet.

```
USR <TrID> TWN S t=<Ticket vom AS>
```

Bei einem korrekten Ticket antwortet die Serverinstanz mit einer „USR“-Anweisung, der das Schlüsselwort „OK“ folgt. Zusätzlich werden der Benutzername und der Alias des Benutzers übertragen.

```
USR <TrID> OK <ID des Benutzers> <Alias des Benutzers>
```

Die eigentliche Anmeldung wird nun mit einer „MSG“-Nachricht der Klienteninstanz mit Informationen über den Benutzer abgeschlossen. Diese *MIME*-Nachricht mit dem *Content-Type* „text/x-msmsgsprofile“ wird mit „MSG“ eingeleitet. Jede zu übertragende Information wird mit „$\backslash r \backslash n$“ abgeschlossen und besitzt den Aufbau „Bezeichner : Inhalt“.

```
MSG Hotmail Hotmail 460
MIME-Version: 1.0
Content-Type: text/x-msmsgsprofile; charset=UTF-8
<letzter und aktueller Anmeldezeitpunkt>
<Informationen zur Kontaktaufnahme>
<Persönliche Informationen>
MSPAuth: <Authenisierungsticket>
ClientIP: <IP-Adresse der Klienteninstanz>
ClientPort: <Portnummer>
```

Synchronisation Wurde die Anmeldung erfolgreich abgeschlossen, müssen Informationen, wie die Kontaktliste, zwischen der Server- und der Klienteninstanz abgeglichen werden. Im ersten Schritt sendet die Klienteninstanz eine „SYN“-Anweisung an die Serverinstanz. Die Anweisung besitzt neben der *Transaction ID* eine *version number*. Diese entspricht einer fortlaufenden Nummer, die die Aktualität der hinterlegten Informationen widerspiegeln soll. Ist der Server auf dem gleichen Stand, muss dieser keine Informationen übertragen. Bei verschiedenen Nummern muss die Serverinstanz die für eine Aktualisierung benötigten Daten bereitstellen.

Da die Klienteninstanz nach der erfolgreichen Anmeldung noch keine aktuellen Informationen besitzt, wird eine „0“ als Versionsnummer übertragen.

```
SYN <TrID> 0
```

Diese „0“-Version beantwortet die Serverinstanz mit einem Tupel aus der aktuellen *version number* des Servers, der Anzahl der Einträge der Kontaktliste und der Anzahl Gruppen, die die Kontaktliste strukturieren.

```
SYN <TrID> <Stand des Servers> <Anzahl Kontakte> <Anzahl Gruppen>
```

Als weitere Antwort sendet die Serverinstanz ein Datagramm mit einer „GTC“- und einer „BLP“-Anweisung. Der „GTC“-Befehl übermittelt das Verhalten bei der Eintragung eines unbekannten Benutzers auf der Kontaktliste.

```
GTC {A|N}
BLP {BL|AL}
```

Diese Information musste bei einer vorangegangenen Sitzung der Serverinstanz mitgeteilt werden. Bei dem Eintrag „A“ wird der Benutzer von der Klienteninstanz in einem Dialog gefragt, wie mit dem neuen Kontakt zu verfahren ist (Beispiel: Blocken oder in die *Allow List* einfügen). Bei der Option „N“ wird der Kontakt, der den Benutzer in seine Kontaktliste eingetragen hat, automatisch in die *Allow List* des Benutzers eingefügt.

Der „BLP“-Befehl steuert das Verhalten bei einer empfangenden Nachricht von einen fremden Benutzer, der sich weder auf der *Allow List* noch auf der *Deny List* befindet. Bei dem Argument „AL“ werden unbekannte Benutzer behandelt, als ob sich sich auf die *Allow List* befinden würden. So können alle fremden Kontakte den Benutzer eine Nachricht senden. Im anderen Fall wird das Argument „BL“ übermittelt, bei dem die Nachrichten von fremden Benutzern geblockt werden.

Nun übermittelt die Serverinstanz als weitere Information die hinterlegte Telefonnummer des Benutzers. Zum Abschluss übermittelt die Serverinstanz die hinterlegte Kontaktliste (Anweisung: „LST“) und eine Liste mit den vom Benutzer definierten Benutzergruppen (Anweisung: „LSG“). Für jeden Eintrag wird eine eigenständige Anweisung übermittelt.

```
LST <ID des Kontakts> <Alias des Kontakts> <Listeneintrag>
LSG <Nummer der Gruppe> <Name der Gruppe> 0
```

Der Listeneintrag gibt an, auf welchen Listen der Kontakt eingetragen ist. Dabei entspricht die *Forward List* dem Wert 1, die *Allow List* der 2, die *Block List* der 4 und die *Reverse List* der 8. Ist der Kontakt auf mehreren Listen eingetragen, werden diese Werte entsprechend addiert.

A.3.3. Einfügen eines Benutzers in die Kontaktliste

Im Gegensatz zu anderen *Instant Messaging*-Systemen wird bei diesem Protokoll keine einzelne Kontaktliste definiert. An dessen Stelle wird die Funktionalität zwischen der *Forward List* (FL), *Reverse List* (RL), *Allow List* (AL) und *Block List* (BL) aufgeteilt.

Diese Listen werden nur auf dem Server verwaltet. Somit ist es jederzeit problemlos möglich, dass der Benutzer sein Klienten wechselt.

In der *Allow List* stehen alle Kontakte, die den Online-Status des Benutzers einsehen können. Im Gegensatz dazu werden alle Kontakte, in die *Block List* eingetragen, die der Benutzer ignorieren möchte. Die *Forward List* entspricht der Kontaktliste, die der Benutzer in seiner Klienteninstanz angezeigt bekommt. In der *Reverse List* stehen alle Kontakte, die den aktuellen Benutzer in ihre *Forward List* integriert haben.

Prinzipiell sind viele der Listen von den anderen *Instant Messaging*-Systemen bekannt, sie sind dort jedoch nur optional. Beim *MSN Messenger* werden diese größtenteils transparent vom Benutzer gepflegt.

Für das Eintragen eines neuen Kontakts, das in Abbildung 28 dargestellt wird, muss dieser im ersten Schritt von der *Block List* entfernt und dann in die *Allow List* und *Forward List* eingetragen werden. Abschließend muss in der *Reverse List* des gewünschten Kontakts der Benutzer eingefügt und der Kontakt hierüber informiert werden.

Die Anweisungen dafür sind ähnlich strukturiert. Es muss aber zwischen Anweisungen von der Klienten- und von der Serverinstanz unterschieden werden. Für das Löschen von der *Block List* und das Einfügen auf die *Allow List* wird diese Information von der Klienteninstanz übertragen, wobei dies die Serverinstanz bestätigten muss.

Eine Anfrage für das Löschen („REM“) und das Einfügen („ADD“) von der Klienteninstanz hat folgenden Aufbau:

```
{REM|ADD} <TrID> {BL|AL} <Kontakt-Identifikator>
```

Der zweite Parameter spezifiziert die Liste, auf der die Operation durchgeführt werden soll. Das Einfügen auf die *Reverse List* und *Forward List* wird in der Regel direkt von der Serverinstanz ausgeführt. Um den Benutzer beziehungsweise Kontakt hierüber zu informieren, sendet die Serverinstanz ebenfalls eine Anweisung, die nicht bestätigt werden muss.

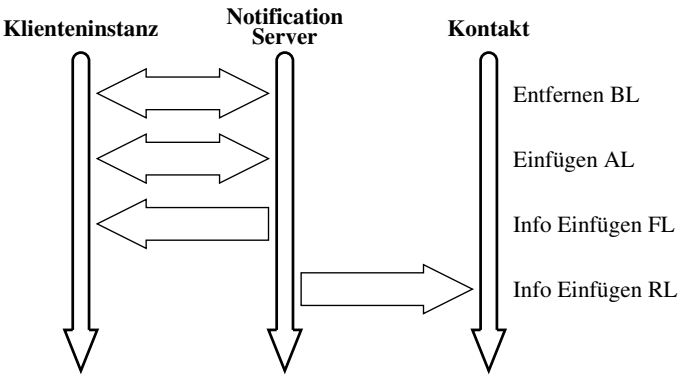

Abbildung 28: MSN - Einfügen eines Kontakts

```
{REM|ADD} <TrID> {BL|AL|FL|RL} <Version der Liste> <Kontakt-ID>
```

Hinzugekommen ist bei den Serveranweisungen die Versionsnummer der aktuellen Liste. Diese wird für die Synchronisation bei der Anmeldung benötigt.

A.3.4. Statusverwaltung

Anhand der *Forward List* übermittelt die Serverinstanz dem Klienten den Status seiner Kontakte. Zusätzlich kann der Benutzer jederzeit seinen eigenen Status ändern, was den Kontakten ebenfalls mitgeteilt werden muss. Es steht den einzelnen Benutzern frei, folgende übergeordnete Zustände zu verwenden:

- Online (NLN), angemeldet und für Kontakte sichtbar
- Offline (FLN), nicht angemeldet und für Kontakte nicht sichtbar
- Hidden (HDN), angemeldet und für Kontakte nicht sichtbar

Der „Online“- Status kann noch durch folgende untergeordnete Zustände verfeinert werden:

- Busy (BSY)
- Idle (IDL)
- Be Right Back (BRB)
- Away (AWY)
- On the Phone (PHN)
- Out to Lunch (LUN)

Änderung des Status eines Kontakts Sobald der Kontakt Statusänderungen durchgeführt hat, werden diese Informationen selbständig von der Serverinstanz zu allen Benutzern übertragen, die den Kontakt auf ihrer *Forward List* stehen haben. Dies kann durch folgende Anweisung realisiert werden:

```
NLN <Status> <Kontakt-ID> <Aktueller Alias des Kontakts>
```

Durch diese Anweisung kann ebenfalls der Wechsel des Alias des Kontakts allen Benutzern mitgeteilt werden, die diesen Kontakt auf ihrer *Forward List* stehen haben. Hierfür überträgt der Kontakt einfach seinen neuen Alias als letzten Parameter dieser Anweisung.

Wechsel des eigenen Status Oft ist es sinnvoll, der Serverinstanz den eigenen Status zu übermitteln, die dies wiederum den Kontakten mitteilt. Dies kann mit folgender Anweisung umgesetzt werden:

```
CHG <TrID> <Status>
```

Jede dieser Anweisungen, die von der Klienteninstanz versendet wird, muss von der Serverinstanz mit einer Anweisung mit dem gleichen Aufbau bestätigt werden.

Wechselt der Benutzer den Alias, der in den Listen seiner Kontakte geführt wird, muss er hierfür folgende Anweisung zum Server übermitteln:

```
REA <TrID> <Identifikator des Benutzers> <Neuer Alias>
```

Diese Anweisung wird mit einer ähnlichen Anweisung durch die Serverinstanz bestätigt:

```
REA <TrID> <Version der Serverliste> <Benutzer-ID> <Neuer Alias>
```

Steuerung der Sichtbarkeit und Privatsphäre Anders als bei AIM oder ICQ gibt es bei dem *MSN Messenger* nicht die Möglichkeit, dass ein anderer Benutzer für das Einfügen auf der Kontaktliste eine Erlaubnis benötigt. Diese Funktionalität wird durch das Zusammenspiel der *Allow List* und der *Block List* umgesetzt.

Diese Listen gelten sowohl für die Kontaktaufnahme als auch zum Einsehen des Online-Status. Neben der expliziten Eintragung einzelner Benutzer kann bestimmt werden, wie mit Benutzern umzugehen ist, die auf keiner der beiden Listen stehen.

Zum Verständnis wird an dieser Stelle der Benutzername „Unbekannt“ einführt, der für alle unbekannten Benutzer verwendet werden soll, die nicht auf einer der Listen stehen. Steht der Kontakt „Unbekannt“ auf der *Block List*, kann niemand unbekanntes mit dem Benutzer in Kontakt treten. Im umgekehrten Fall, in dem „Unbekannt“ auf der *Allow List* steht, kann jeder eine Verbindung mit Statuseinsicht aufbauen.

Dieser virtuelle Benutzer wird in der Praxis durch den kombinierten Einsatz von *Block List* und *Allow List* simuliert. Dadurch kann jeder Benutzer jeden anderen auf seiner Kontaktliste eintragen. Ob aber die Kontaktaufnahme oder die Einsicht in den Status möglich ist, entscheidet der Benutzer ganz alleine.

Trägt ein Benutzer einen anderen in seine Kontaktliste ein, wird auf der *Reverse List* des einzutragenden Benutzers der Benutzer aufgenommen, der die Eintragung vornimmt. Dabei wird der einzutragende Benutzer hierüber informiert, so dass er bei Bedarf den anderen Benutzer in seiner *Allow List* aufnimmt.

Ob nun alle unbekannten Benutzer auf der *Allow List* oder oder *Block List* stehen sollen, muss die Klienten- der Serverinstanz übermitteln. Dies kann sie mit der folgenden Anweisung umsetzen, die in Abhängigkeit mit der gewünschten Liste steht:

```
BLP <TrID> {AL|BL}
```

Diese Nachricht muss von der Serverinstanz bestätigt werden. Diese Antwort hat den identischen Aufbau wie die Nachricht der Klienteninstanz.

A.3.5. Klientenkommunikation

Im Gegensatz zu vielen anderen *Instant Messaging*-Systemen ist bei der Kommunikation über den *MSN Messenger* keine direkte Verbindung zum Gesprächspartner möglich. Um aber den *Notification Server* zu entlasten, wird für die Kommunikation eine Verbindung zu einem *Switchboard Server* aufgebaut, über den die gesamte Kommunikation mit dem Gesprächspartner abgewickelt wird. In Abbildung 29 wird ein Ablaufdiagramm für den benötigten Informationsaustausch dargestellt.

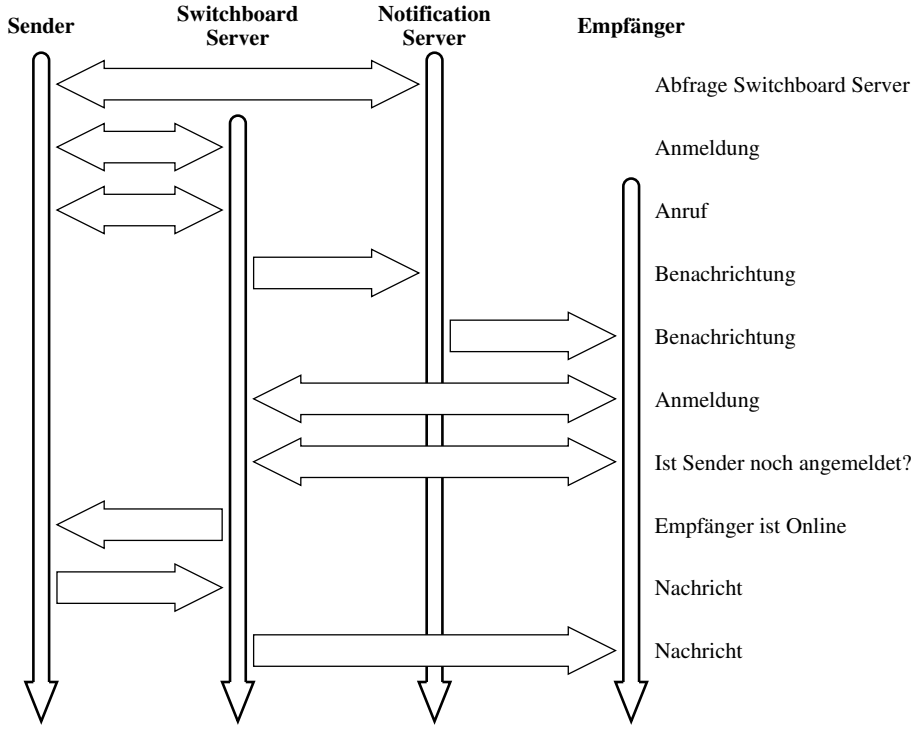

Abbildung 29: MSN - Kommunikation zwischen zwei Benutzern

Abfrage von Informationen über den *Switchboard Server* Im ersten Schritt muss die Klienteninstanz die Verbindungsdaten des *Switchboard Servers* abfragen. Hierfür sendet sie dem *Notification Server* eine kurze „XFR“-Anweisung, die neben der *Transaction ID* das Schlüsselwort „SB“ für *Switchboard Server* enthält:

```
XFR <TrID> SB
```

Hierauf antwortet der *Notification Server* ebenfalls mit einer „XFR“-Anweisung mit entsprechender *Transaction ID*. Neben dem Schlüsselwort „SB“ werden eine Kombination aus Serveradresse und Portnummer sowie ein Schlüssel für die Authentisierung übermittelt. Dieser relativ kurze Schlüssel besteht aus zwanzig bis dreißig Ziffern, die für den Nachweis der Identität gegenüber dem *Switchboard Server* benötigt werden.

```
XFR <TrID> SB <Adresse>:<Port> <Schlüssel>
```

Anmeldung des Senders am *Switchboard Server* Nun wird eine neue TCP-Verbindung zu dem *Switchboard Server* aufgebaut. Dieser „USR“-Anmeldevorgang ist mit dem am *Notification Server* identisch. Für die Authentisierung wird der von dem *Notification Server* erhaltene Schlüssel übermittelt.

Übermittlung des Verbindungswunsches zum *Switchboard Server* Der eigentliche Anruf wird durch die Übermittlung einer „CAL“-Anweisung mit folgender *Transaction ID*

umgesetzt. Als Anfrage übermittelt die Klienteninstanz zusätzlich den eindeutigen Identifikator des Gesprächspartners.

```
CAL <TrID> <ID des Gesprächspartners>
```

Ist dieser gültig, wird die Klienteninstanz mit dem Schlüsselwort „RINGING“ und einer *Session ID* hierüber in Kenntnis gesetzt. Diese *Session ID* soll nach der späteren Anmeldung des Gesprächspartners eine Zuordnung zwischen dem Sender und Empfänger ermöglichen.

```
CAL <TrID> RINGING <Session ID>
```

Kommunikation zwischen *Switchboard Server* und *Notification Server* des Gesprächspartners Über den genauen Inhalt dieser Kommunikation kann keine Aussage getroffen werden. Aber prinzipiell muss der *Notification Server* informiert werden, welcher Benutzer mit welchem anderen Benutzer kommunizieren möchte. Außerdem muss die *Session ID*, die vorher der Sender vom *Notification Server* empfangen hat, übermittelt werden.

Benachrichtigung des Gesprächspartners Nachdem der *Switchboard Server* den *Notification Server* über die anstehende Kommunikation unterrichtet hat, wird nun der Empfänger über den Anruf in Kenntnis gesetzt. Hierfür versendet der *Notification Server* eine „RNG“-Anweisung. Als Parameter werden die vorher übertragene *Session ID*, die Adresse und die Portnummer des *Switchboard Server*, ein Schlüssel für die Anmeldung am *Switchboard Server* und der Identifikator des Senders übertragen.

```
RNG <Session ID> <Adresse>:<Port> CKI <Schlüssel> <Sender-ID> <Alias>
```

Anmeldung des Gesprächspartners am *Switchboard Server* Nachdem der vorige Schritt erfolgreich ausgeführt wurde, kann sich nun der Empfänger am *Switchboard Server* anmelden.

Im Gegensatz zu der aus vorigen Beispielen bekannten Authentisierung meldet sich hier der Empfänger mit einer „ANS“-Anweisung am *Switchboard Server* an. Als Parameter folgen die *Transaction ID*, der Benutzername, der übermittelte Schlüssel für die Authentisierung und die *Session ID*. Diese *Session ID* soll den Empfänger der Kommunikation dem Sender zuordnen.

```
ANS <TrID> <ID des Benutzers> <Schlüssel> <Session ID>
```

Die erfolgreiche Authentisierung wird durch den *Switchboard Server* ebenfalls mit einer „ANS“-Anweisung mit der entsprechenden *Transaction ID* und dem Schlüsselwort „OK“ erwidert.

Überprüfung, ob der Sender noch angemeldet ist Um zu überprüfen, ob der Sender noch am *Switchboard Server* angemeldet ist, übermittelt der Empfänger eine „IRO“-Anweisung an den Server. Dies ist für den Fall relevant, bei dem der Empfänger über einen längeren Zeitraum nicht antworten konnte. Falls mehrere Sender innerhalb einer Gruppendiskussion mit dem Empfänger kommunizieren wollen, wird für jeden Gesprächspartner eine einzelne „IRO“-Anweisung übermittelt.

```
IRO <TrID> <aktuelles IRO> <Anzahl aller IRO's> <Senders-ID> <Alias>
```

Der Anweisung folgt eine *Transaction ID*. Da mehrere Empfänger berücksichtigt werden können, wird die Gesamtanzahl der zu übertragenden Anweisungen übermittelt. Jedes dieser Datagramme beinhaltet eine Zahl, die angibt, um welche Anweisung es sich dabei handelt und den Identifikator, der den jeweiligen Sender bestimmt.

Sind noch alle gewünschten Gesprächspartner am *Switchboard Server* angemeldet, bestätigt dies der Server mit einer „ANS"-Anweisung mit einer *Transaction ID* und dem Schlüsselwort „OK".

```
ANS <TrID> OK
```

Benachrichtigung, dass sich der gewünschte Gesprächspartner angemeldet hat Hat sich der fehlende Gesprächspartner erfolgreich am *Switchboard Server* angemeldet, wird der ursprüngliche Sender hierüber in Kenntnis gesetzt.

```
JOI <ID des Empfängers> <Alias des Empfängers>
```

Für die Übermittlung wird eine „JOI"-Anweisung genutzt, die als Parameter den Identifikator des Gegenübers übermittelt. Eine *Transaction ID* wird für diese Mitteilung nicht benötigt.

Übermittlung der Nachricht Wurden die bisherigen Schritte erfolgreich ausgeführt, können an dieser Stelle die eigentlichen Informationen wie Textnachrichten übermittelt werden. Möchte der Empfänger die Nachrichten beantworten, so müssen die vorigen Schritte nicht noch einmal durchgeführt werden, sondern die bestehende Verbindung kann genutzt werden.

Zu unterscheiden ist zwischen Nachrichten, die vom Benutzer empfangen und solchen, die vom Benutzer gesendet werden sollen. Zu sendende Nachrichten, die von der Klienten- zu der Serverinstanz übertragen werden sollen, besitzen folgende Gestalt:

```
MSG <TrID> <Benachrichtigungmodus> <Umfang in Bytes>
<MIME-Nachricht>
```

Bei dem Benachrichtungsmodus „U" muss keine Nachricht mit einem „ACK" bestätigt werden. Soll jede Nachricht von der Serverinstanz bestätigt werden, wird ein „A" übertragen. Soll der Benutzer nur darüber informiert werden, wenn die Nachricht dem Empfänger nicht zugestellt werden kann, wird hier ein „N" eingetragen.

Da die aktuelle TCP-Verbindung nur für eine Gesprächssitzung aufgebaut wurde und nur die gewünschten Empfänger diese Nachricht erhalten können, wird die explizite Angabe des Empfängers nicht benötigt.

Nachrichten, die von der Server- zu der Klienteninstanz übermittelt werden, besitzen den folgenden Aufbau:

```
MSG <ID des Senders> <Alias des Senders> <Umfang in Bytes>
<MIME-Nachricht>
```

Da es möglich ist, dass diese Nachricht mehrere Benutzer innerhalb einer Gruppendiskussion erhalten, muss der Sender explizit genannt werden.

Alle Nachrichten sind *MIME*-kodiert. Daher ist es für die Analyse irrelevant, ob es sich bei diesem Nachrichtenstrom um Textnachrichten, Sprache, Videos oder andere Daten handelt.

Der Umfang einer *MIME*-Nachricht ist auf 1664 Bytes beschränkt. Da die Serverinstanz bei einer größeren Byteanzahl die Verbindung ohne Übermittlung abbricht, ist die Klienteninstanz für das Teilen und Zusammenfügen der Nachrichten zuständig. Dazu können manche Klienten bei Texten die Anzahl der Zeichen pro Nachricht beschränken.

Eine *MIME*-Nachricht besitzt folgenden Aufbau, wobei die zwingend benötigten Eintragungen fett dargestellt werden:

```
MIME-Version: 1.0
Content-Type: text/plain; charset=UTF-8
TypingUser: im-studie@gmx.de
X-MMS-IM-Format: FN=Arial; EF=I; CO=0; CS=0; PF=22

Hier könnte eine Textnachricht stehen
```

A.4. XMPP

Das ***Ex**tensible **M**essaging and **P**resence **P**rotocol* (XMPP) wurde hauptsächlich für das *Instant Messaging*-System *Jabber* entwickelt. Die Spezifikationen wurden von der IETF[22] als Internetstandard in den RFCs 3920 (*XMPP: Core*), 3921 (*XMPP: Instant Messaging and Presence*) und 3922 (*Mapping the XMPP to Common Presence and Instant Messaging (CPIM)*) [15] veröffentlicht.

Die gesamte Kommunikation findet über eine TCP-Verbindung statt, die optional mittels TLS geschützt werden kann. Sowohl für die unverschlüsselte als auch die verschlüsselte Kommunikation zwischen einer Klienten- und Serverinstanz wurde der TCP-Port 5222 reserviert. Soll eine Verbindung zwischen Servern aufgebaut werden, geschieht dies sehr oft über den TCP-Port 5269.

Als Format für die zu übertragenden Anweisungen wurde XML gewählt. Da alle Informationen ASCII-kodiert sind, kann die gesamte Kommunikation auch sehr gut von Menschen gelesen und die damit verbundenen Funktionalitäten erkannt werden.

Die folgende Protokollanalyse stützt sich hauptsächlich auf die oben genannten RFCs. Wenn dies möglich war, wurden die Ergebnisse durch einen für diese Diplomarbeit entwickelten Klienten (siehe Anhang C.4) in Verbindung mit einem aus *Jabberd*-Servern in der Version 2.0.6 bestehenden Netz nachvollzogen.

A.4.1. Unterschiede zu früheren Protokollversionen

Bevor XMPP als *Jabber*-Protokoll eingesetzt wurde, existierte ein anderes XML-basiertes Protokoll für den Informationsaustausch. Dieses Protokoll wurde wie auch das *Instant Messaging*-System und das dazugehörige Netz als „*Jabber*“ bezeichnet. Um im Folgenden Verwechselungen zu vermeiden, wird dieses Protokoll „klassisches *Jabber*-Protokoll“ genannt.

Ein Großteil der Klienten und Server, die zum Zeitpunkt der Erstellung der Diplomarbeit vorlagen, nutzen noch das klassische *Jabber*-Protokoll. Da aber in naher Zukunft nur XMPP eingesetzt werden soll, konzentriert sich die Analyse darauf. Um Probleme mit Klienten auszu-

[22] **Vertiefung:** Internet Engineering Task Force http://www.ietf.org

schließen, die vollständig oder teilweise das klassische *Jabber*-Protokoll verwenden, wurde ein Klient entwickelt, der nur XMPP unterstützt (Quellcode: siehe Anhang C.4).

Bei der Entwicklung von XMPP wurden vom klassischen *Jabber*-Protokoll viele XML-Namensräume sowie deren Syntax und Semantik übernommen. Teile des klassischen *Jabber*-Protokolls, die durch andere Funktionen eleganter implementiert werden konnten, wurden entfernt. Dies macht XMPP viel übersichtlicher. Einige wesentliche Unterschiede werden in den folgenden Abschnitten vorgestellt.

Verschlüsselung der Nutzlast Das klassische *Jabber*-Protokoll verwendet SSL (siehe Anhang B.15). Dieser Informationsaustausch findet über eine eigenständige TCP-Verbindung statt - in der Regel wird Port 5223 für eine Kommunikation zwischen Klienten- und Serverinstanzen und Port 5270 für den Informationsaustausch zwischen Serverinstanzen verwendet.

XMPP nutzt die registrierten Jabber-Ports 5222 (Klient-Server) und Port 5269 (Server-Server) für eine verschlüsselte Kommunikation. Hierfür wird der kryptographische Schutz über TLS (siehe Kapitel 10.2.1) aktiviert.

Authentisierungsverfahren Im klassischen Protokoll werden die Informationen für eine Authentisierung gegenüber einem Server in einem eigenen XML-Namensraum (*jabber:iq:auth*) versendet. Informationen wie Passwörter werden dabei mittels Prüfsummenverfahren während der Übertragung geschützt. Zwischen den Servern findet keine Überprüfung der Identität statt. Durch das Fälschen von Netzadressen kann sich ein Angreifer als eine Serverinstanz ausgeben.

Für die Authentisierung in XMPP wird SASL verwendet (Siehe Kapitel 10.2.2). SASL ermöglicht den sicheren und geschützten Nachweis der Identität des Benutzers. Serverinstanzen können mittels *Server Dialbacks* (Siehe Kapitel 10.2.3) anderen Servern ihre Identität auf Grundlage ihres DNS-Hostnamens nachweisen.

Ressourcenbindung Das Binden von Ressourcen an eine Verbindung ist bei dem klassischen *Jabber*-Protokoll Bestandteil der Authentisierung.

XMPP sieht eine Trennung zwischen der Authentisierung und dem Binden der Ressourcen vor. Dies wird in einen eigenen Dialog mittels „<bind>“-Anweisungen umgesetzt.

Sonderzeichen in JIDs Die Definition von *Jabber IDs* (*JID*) im klassischen Protokoll ist sehr unpräzise. Damit ist die Gültigkeit einer *JID* von der verwendeten Software abhängig.

Unter XMPP wurde das Aussehen der JIDs detailliert festgelegt. Im Gegensatz zum klassischen Protokoll sind einige Sonderzeichen wie $;/?:@\&=+\$,[]$ im Benutzernamen explizit ausgeschlossen.

Fehlerbehandlung Fehler bei der Initialisierung der Verbindung („<stream:stream>“) werden von beiden Protokollen durch spezielle *error*-Elemente beantwortet. Diese unterscheiden sich jedoch in ihrem Aufbau.

Tritt ein Fehler erst während der späteren Kommunikation auf, so beantwortet das klassische *Jabber*-Protokoll diese mit HTML-ähnlichen Fehlercodes. XMPP dagegen verwendet spezielle *error*-Elemente, die den gleichen Aufbau wie die Anfrage aufweisen.

Internationalisierung Bei beiden Protokollen werden die XML-Anweisungen UTF-8 kodiert übertragen. In XMPP-Anweisungen kann über das *xml:lang*-Attribut eine Sprache zugeordnet werden. Dieses Attribut vereinfacht bei einer Analyse der übertragenen Steueranweisungen die Zuordnung der teilweise auf den ersten Blick für einen menschlichen Betrachter schwer zu „übersetzenden“ Zeichen einer Sprache - für die Kommunikation der Klienten- und Serverinstanzen hat es keine Bedeutung.

Bei dem klassischen *Jabber*-Protokoll müsste der Betrachter die Zeichen komplett dekodieren, um den Sinn zu interpretieren.

Versionsnummer Eine Möglichkeit, um zweifelsfrei das klassische *Jabber*-Protokoll von XMPP zu unterscheiden, ist die Betrachtung der Argumente des initialisierenden XML-Streams („<stream:stream>“). Besitzt es als Attribut eine Versionsnummer, dann findet die Kommunikation über XMPP ab. Es ist zu beachten, dass unabhängig vom Protokoll als erste Information noch vor dem „<stream:stream>“ eine XML-Versionsnummer („<?xml version='1.0'?> “) übertragen wird.

A.4.2. Grundlagen für den Informationsaustausch mittels XML

Befehle und Nachrichten werden in XMPP als ASCII-kodierte XML-Anweisungen übermittelt. Der logische Aufbau eines XML-Dokumentes ist mit einem hierarchisch strukturierten Baum vergleichbar. Jedes Element (Knoten) besitzt einen Start- und ein End-*Tag* (Beispiel: „<knoten> weitere Elemente </knoten>“). Soll ein Element keine Unterelemente besitzen, kann dieses *Tag* mit „<knoten />“ ohne ein End-*Tag* geschlossen werden. Eine XML-Struktur ist wohlgeformt, wenn jedes *Tag* geschlossen wird und keine Kollisionen bei Verschachtelungen auftreten.

Vielen Elementen können Attribute zugeordnet werden. Dies kann mit „<*knoten attribut-name=attribut-wert*>“ realisiert werden.

Da an dieser Stelle nur schematisch auf XML eingegangen werden kann, wird für vertiefende Informationen auf die W3C-Spezifikationen[23] verwiesen. Die hier vorgestellten Eigenschaften sind aber für das Verständnis von XMPP ausreichend.

XMPP-Streams Eine XMPP-Kommunikation über eine TCP-Verbindung muss mit einem *XMPP-Stream* eingeleitet werden. Dieser Rahmen kann einen oder mehrere Blöcke (*Stanzas*) umfassen, die die zu übertragenden Informationen beinhalten. Ein schematischer Aufbau eines XML-Stroms ist in Abbildung 30 zu finden.

Zu jeder TCP-Verbindung werden mindestens ein *XMPP-Stream* zwischen dem Sender und dem Empfänger und zwischen dem Empfänger und dem Sender initialisiert. Der prinzipielle Aufbau einer solchen Anweisung, die jede XMPP-Kommunikation einleitet, ist im folgenden Quelltextsegment zu finden:

```
<?xml version='1.0'?>
<stream:stream ...>
  ...
</stream:stream>
```

[23] **Vertiefung:** http://www.edition-w3c.de

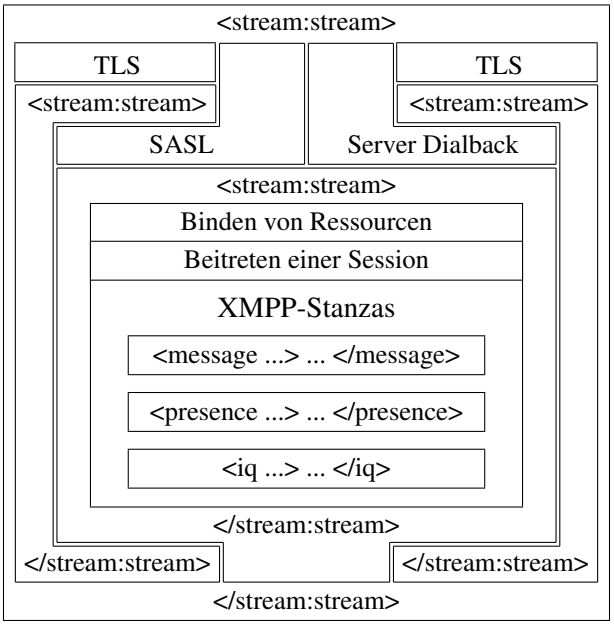

Abbildung 30: XMPP - Struktur eines XMPP-Streams

Nachdem die TCP-Verbindung aufgebaut wurde, übermittelt der Sender die XML-Version (hier: 1.0) an den Empfänger. Der Versionsnummer folgt der *Tag* „<stream:stream>“, dem wiederum zahlreiche Argumente folgen.

Mögliche Argumente für das „<stream:stream>“-*Tag* sind in Tabelle 12 zu finden. Zusätzlich wird als Argument eine nicht-*Jabber*-spezifische XML-Definition eines eindeutigen Namensraums („*xmlns*“) übergeben. Relevant in diesem Zusammenhang ist die Zuordnung des Namensraum „jabber:client“ für Kommunikationsabläufe zwischen Klienten- und Serverinstanzen und „jabber:server“ für Verbindungen zwischen Serverinstanzen.

Wert	**Sender zum Empfänger**	**Empfänger zum Sender**
to	DNS-Name des Empfängers	optional, wird aber sehr oft verwendet
from	optional, wird aber sehr oft verwendet	DNS-Name des Empfängers
id	wird nicht benötigt	*Session Key*
xml:lang	XML-Universalattribut, verwendete Sprache, optional	
version	Versionsnummer unterscheidet XMPP von anderen Jabberprotokollen	

Tabelle 12: XMPP - Argumente des „<stream:stream>“-*Tags*

Dem einleitenden „<stream:stream>“-Tag folgen alle Anweisungen, die während der gesamten Verbindung übermittelt werden.

Vor dem Abbau der TCP-Verbindung sollte der Anweisungsblock mit dem Tag „</stream: stream>“ geschlossen werden. Viele Klienten ignorieren diese aber, so dass der Anweisungsblock nicht mehr wohlgeformt ist.

Innerhalb dieses Rahmens können weitere „<stream:stream>“-Blöcke übertragen werden. Sehr oft wird nach einer SASL-Authentisierung, die durch ein „<stream:stream>“-*Tag* eingelei-

tet wurde, die folgende Bindung an das *Instant Messaging*-System ebenfalls in einem „<stream:stream>“ gekapselt. Ein weiteres Beispiel ist „<stream:error>“ für Fehlermeldungen.

Für die weitere Betrachtung des XMPP-Protokolls wird das über die gesamte Kommunikation geöffnete „<stream:stream>“-*Tag* nicht weiter berücksichtigt.

XMPP-Stanzas Dem „<stream:stream>“-*Tag* folgt die Nutzlast. Dabei kann zwischen Steueranweisungen („<iq>“), Anweisungen für die Nachrichtenübermittlung („<message>“) und Steueranweisungen für den Status („<presence>“) unterschieden werden. Diese XMPP-Anweisungen werden auch als *Stanzas* bezeichnet.

Abbildung 30 auf Seite 137 beinhaltet ebenfalls die Anordnung von XMPP-*Stanzas* in einer *Jabber*-Sitzung.

Steueranweisungen Alle Anweisungen, die nicht für Statusänderungen oder für die Übermittlung von Nachrichten genutzt werden, werden innerhalb von „<iq>“ (*Info/Query*)-Blöcken übertragen. Diese Anweisungen können zwischen Klienten- und Serverinstanzen oder zwischen Serverinstanzen untereinander ausgetauscht werden. Bei der Client-Server-Kommunikation sendet im Normalfall die Klienteninstanz eine Anfrage, die die Serverinstanz beantwortet.

Die möglichen Argumente einer „<iq>“Anweisung sind in Tabelle 13 zu finden. Sollen Informationen ausgetauscht werden, so muss das Argument **type** angegeben werden. Mit dem Wert „get“ kann eine Information abgefragt, mit „set“ eine Einstellung gesetzt und mit „result“ bestätigt beziehungsweise beantwortet werden. Zusätzlich kann mit einem „error“ dem Kommunikationspartner das Scheitern einer Aktion angezeigt werden.

Neben dem *type*-Argument ist die Angabe eines **id**-Arguments zwingend erforderlich. In der Antwort auf „get“ oder „set“-Anfragen muss der Empfänger das gleiche *id*-Argument verwenden wie der Sender in der zugehörigen Anfrage. Obwohl der Sender den Inhalt des Wertes frei wählen kann, werden in der Praxis häufig aussagekräftige Werte wie „auth_1“ für den ersten Authentisierungsschritt gewählt.

Attribut	**Beschreibung**
type	Bezeichner für Nachrichtentyp (Anfrage oder Antwort), Werte: *set*/*get*/*response*/*error*
id	*Session ID*, ordnet einer Anfrage eine Antwort zu
to	eindeutiger Identifikator des Empfängers, wird nur für einige *Stanza* benötigt
from	eindeutiger Identifikator des Senders, wird nur für einige *Stanza* benötigt
xmlns	Namensraum der XML-Anweisung, beschreibt Funktion der *Stanza*

Tabelle 13: XMPP - Mögliche Attribute einer „<iq>“-*Stanza*

Zusätzlich können noch die eindeutigen Identifikatoren des Senders (**from**) und des Empfängers (**to**) übermittelt werden. Das *from*-Argument wird hauptsächlich für die Kommunikation zwischen Serverinstanzen benötigt. Dies ist erforderlich, da bei einer weitergeleiteten Nachricht der Empfangsserver nicht auf die Informationen der TCP-Verbindung des Senders und seiner Serverinstanz zurückgreifen kann, um den Sender zu bestimmen.

Die Funktion der „<iq>“-Anweisung wird als Namensraum im Attribut *xmlns* definiert. Mögliche Werte hierfür werden in späteren Beispielen vorgestellt.

Nachdem alle Argumente gesetzt wurden, können zwischen dem Start- und dem End-*Tag* alle benötigten Informationen übertragen werden. Hierzu gehören auch Fehlermeldungen, die mit dem *type*-Argument „error“ eingeleitet werden. Folgend wird der Aufbau einer Fehlermeldung grob skizziert:

```
<iq type="error" ... >
  <!-- oft Wiederholung der problematischen Anweisung des Senders -->
    <error ...>
      <!-- <Tag mit "Condition" /> -->
      <text ...>        <!-- Anfang Optional -->
        ...
      </text>           <!-- Ende Optional -->
    </error>
</iq>
```

Zusätzlich zum gleichen **id**-Wert, der schon den Bezug zu der Anfrage herstellt, wird sehr oft die komplette Nachricht des Senders wiederholt.

Dieser optionalen Nachricht folgt das „<error>“-*Tag*, das Argumente besitzt. Wichtig ist das *type*-Argument des „<error>“-*Tags*, dessen Werte in Tabelle 14 zu finden sind.

Wert	Beschreibung
cancel	Abbruch - Fehler ist nicht zu beheben
continue	Aktion wird fortgesetzt - nur Warnung
modify	nachdem Informationen geändert wurden kann Anfrage wiederholt werden
auth	nachdem Berechtigung nachgewiesen wurde kann Anfrage wiederholt werden
wait	temporärer Fehler - Nach einer bestimmten Zeitspanne kann Anfrage wiederholt werden

Tabelle 14: XMPP - Werte des *type*-Arguments für Fehlermeldungen

Dem „<error>“-*Tag* folgt ein *Tag* für **Conditions**, das die Ursache der Fehlermeldung beschreibt.

Zusätzlich besteht die Möglichkeit, durch eine Zeichenkette den aufgetretenen Fehler auch für den Menschen verständlich zu beschreiben. Diese Fehlermeldung kann zwischen den „<text>“-*Tags* definiert werden.

Als nicht offizielle Variante kann die Fehlermeldung auch numerisch kodiert werden. Diese Möglichkeit, die in dieser Arbeit nicht weiter vorgestellt wird, hat folgende Gestalt:

```
<iq type="error" ... >
  <!-- oft Wiederholung der problematischen Anweisung des Senders -->
    <error code='...'>
      <!-- Optionale Fehlerbeschreibung -->
    </error>
</iq>
```

Nachrichtenübermittlung Für die textbasierte Kommunikation zwischen Benutzern werden „<message>“-Anweisungen benötigt. Die Zustellung erfolgt anhand des *to*-Attributs, das die eindeutige *JID* des Empfängers beinhaltet. Die Sprache, in der die Nachricht verfasst wurde, kann mit einem optionalen *xml:lang*-Attribut übergeben werden.

Der genaue Aufbau einer „<message>“-Anweisung ist im Anhang A.4.7 zu finden. Dort wird auch detaillierter auf den Informationsfluss eingegangen.

Benutzerverwaltung Anweisungen, die für die Verwaltung der Benutzerinformationen benötigt werden, werden innerhalb eines „<presence>“-Blocks übertragen. Hierzu gehören auch Anweisungen, die die Verwaltung der Kontaktliste betreffen als auch solche für Änderungen des Status.

Der „<presence>“-*Tag* kann auch Attribute besitzen. Neben den **from**, **to** und **xml:lang**-Attributen, die die gleiche Funktion wie die in den vorigen Abschnitten besitzen, kann auch ein spezifisches **type**-Attribut dem „<presence>“-*Tag* übergeben werden. Die Werte und die Bedeutung der möglichen Werte des *type*-Attributs sind in Tabelle 15 zu finden.

Wert	Beschreibung
unavailable	Informiert, dass Kontakt nicht mehr verfügbar ist
subscribe	Sender möchte Status des Empfängers einsehen
subscribed	Sender darf Status einsehen
unsubscribe	Einsicht in Status abmelden
unsubscribed	Die Einsicht in den Status ist abgemeldet
probe	Abfrage der Statusinformationen, wird nur vom Server aufgerufen
error	Fehler ist aufgetreten

Tabelle 15: XMPP - Werte des *type*-Arguments in dem „<presence>“-*Tag*

A.4.3. Registrierung

Sehr viele *Jabber*-Server unterstützen nur die Registrierung direkt über den Klienten - eine webbasierte Anmeldung, wie sie von vielen proprietären IM-Anbietern angeboten wird, gibt es nur selten.

Nachdem die Klienteninstanz eine TCP-Verbindung zu der gewünschten Serverinstanz aufgebaut hat (im Folgenden: „jabber.org“) und beide die Verbindung mit dem „<stream:stream>“-*Tag* angenommen haben, wird sehr oft eine SASL-Authentisierung begonnen.

Da der Klient noch kein gültiges Passwort und keinen gültigen Benutzernamen besitzt, wird die SASL-Authentisierung abgebrochen und die Registrierung gestartet.

Der Klient kann vor der eigentlichen Registrierung Informationen zu dieser anfordern. Dafür sendet er ein *<iq>-get* an den Server:

```
<!-- Klient -> Server -->
<iq id='netjabber-0' type='get'>
    <query xmlns='jabber:iq:register'/>
</iq>
```

Diese Anfrage könnte der Server mit dem folgenden *result* beantworten:

```
<!-- Server -> Klient -->
<iq xmlns='jabber:client' id='netjabber-0' type='result'>
    <query xmlns='jabber:iq:register'>
      <username/>
```

```
      <password/>
      <instructions>
        Enter a username and password to register with this server.
      </instructions>
    </query>
</iq>
```

Nun leitet der Klient durch folgende „<iq>“-Anweisung die Registrierung ein:

```
<!-- Klient -> Server -->
<iq id='netjabber-1' type='set'>
  <query xmlns="jabber:iq:register">
    <username>Benutzername</username>
    <password>Passwort</password>
  </query>
</iq>
```

Der im „<iq>“-*Tag* definierte Namensraum für eine Klient-Server-Verbindung enthält einen Bezeichner für die Registrierung. Diese wird mit dem Argument „jabber:iq:register“ übergeben. Die dem „<iq>“-*Tag* folgenden *Tags* sind selbsterklärend. Bei dem Benutzernamen handelt es sich nicht um die komplette *JID* mit dem Hostnamen sondern nur um den eigentlichen Benutzernamen.

Das Passwort wird an dieser Stelle ungeschützt übertragen. Daher sollte im Vorfeld eine *TLS*-Verschlüsselung aktiviert werden.

Im Erfolgsfall bestätigt der Server die Registrierung mit einer

```
<!-- Server -> Client -->
<iq xmlns='jabber:client' id='netjabber-1' type='result'/>
```

Anweisung. Ist der Benutzername schon anderweitig vergeben, antwortet der Server mit einer „<conflict>“-Fehlermeldung, die zum sofortigen Abbruch führt (*cancel*). Ist die eigenständige Registrierung der Benutzer nicht gestattet, wird eine „<not-allowed>“-Fehlermeldung übertragen.

Nachdem die Registrierung erfolgreich durchgeführt wurde, kann sich der Klient nohne eine neue Verbindung aufzubauenn über *SASL* authentisieren.

A.4.4. Anmeldung

Um *Jabber* nutzen zu können, muss sich der Benutzer an dem Server, der dem Hostteil seiner *JID* entspricht, anmelden. Im Gegensatz zu den im Vorfeld vorgestellten *Instant Messaging*-Systemen kann somit der kontaktierte Server nicht problemlos gewechselt werden.

Wie in der Abbildung 30 auf Seite 137 bereits skizziert, wird nach der Initialisierung der TCP-Verbindung jeweils von Sender und Empfänger ein *XMPP-Stream* aufgebaut, dem eine optionale *TLS*-Verschlüsselung folgen. Nachdem diese Schritte erfolgreich ausgeführt wurden, beginnt die eigentliche Anmeldung am *Instant Messaging*-System.

Für eine vollständige Anmeldung müssen folgende Schritte ausgeführt werden:

1. Authentisierung über SASL (siehe Kapitel 10.2.2)

2. Initialisieren eines neuen <*stream:stream*>-Blocks
3. Binden von Ressourcen
4. Sessionbeitritt
5. Anforderung der Kontaktliste
6. Übermittlung des eigenen Status
7. Empfang des Status der Kontaktlistenmitglieder

Initialisierung Die einleitenden „<stream:stream>“-*Tags* und die möglichen Attribute des Senders und des Empfänger sind schon aus den vorigen Abschnitten bekannt. Bei der Initialisierung hängt der Empfänger im einfachen Fall (für Alternativen siehe SASL, TLS und *Server Dialback* im Kapitel 10.2) folgende Aufforderung zum Binden der Ressourcen und zum Beitritt einer Session an, auf die der Sender in späteren Anweisungen reagieren muss:

```
<!-- Server -> Klient
  <stream:stream mit Argumenten des Klienten -->
  <stream:features xmlns:stream='http://etherx.jabber.org/streams'>
    <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'/>
    <session xmlns='urn:ietf:params:xml:ns:xmpp-session'/>
  </stream:features>
```

Ressourcenbindung Dem initialisierten *XMPP-Stream* muss in der Regel eine Ressource zugewiesen werden. Die Ressource bezieht sich auf den Identifikator des Klienten. Dies kann zum Beispiel der Name der Klienteninstanz, der Standort, von dem die Klienteninstanz aufgerufen wurde, oder eine Bezeichnung für eine bestimmte Funktionalität sein. Diesen Bezeichner übermittelt die Klienteninstanz mit der folgenden Anweisung an den Server:

```
<!--Klient -> Server -->
<iq id='netjabber-0' type='set'>
   <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
     <resource>Ressourcenname</resource>
   </bind>
</iq>
```

Nachdem der Server diese Information erhalten hat, kann er die vollständige *JID* bilden und diese der Klienteninstanz übermitteln:

```
<!-- Server -> Klient -->
<iq xmlns='jabber:client' id='netjabber-0' type='result'>
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
    <jid>Vollstaendige_JID</jid>
  </bind>
</iq>
```

Sessionbeitritt Nun erfolgt die eigentliche Anmeldung am *Instant Messaging*-System mit der folgenden Nachricht.

```
<!-- Klient -> Server -->
<iq id='netjabber-1' type='set'>
  <session xmlns='urn:ietf:params:xml:ns:xmpp-session'/>
</iq>
```

Der Server bestätigt dies mit der folgenden Anweisung. Damit wurden alle zwingend benötigten Schritte der Anmeldung ausgeführt.

```
<!-- Server -> Klient -->
<iq xmlns='jabber:client' id='netjabber-1' type='result'/>
```

Kontaktlistenübermittlung Die Listen mit den Kontakten werden bei *Jabber* direkt auf dem Server abgelegt. Um den Klienten über die Kontakte zu informieren, die er in vorigen Sitzungen eingetragen hat, müssen diese von der Serverinstanz übertragen werden. Hierfür sendet die Klienteninstanz bei der Anmeldung eine entsprechende Anfrage an den Server:

```
<!-- Klient -> Server -->
<iq from='JID_des_Benutzers' type='get' id='roster_1'>
  <query xmlns='jabber:iq:roster>
</iq>
```

Im folgenden Schritt übermittelt die Serverinstanz die gewünschten Informationen.

```
<!-- Server -> Klient -->
<iq to='JID_des_Benutzers' type='result' id='roster_1'>
  <query xmlns='jabber:iq:roster'>
    <item jid='JID_des_ersten_Kontakts'
          name='Angezeigter_Name'
          subscription='both'
      <group>Gruppenname</group>
    </item>
    <item jid='JID_des_zweiten_Kontakts'
          name='Angezeigter_Name'
          subscription='both'
      <group>Gruppenname</group>
    </item>
    <!-- weitere Kontakte -->
  </query>
</iq>
```

Innerhalb des „<query>“-Blocks wird für jeden Kontakt jeweils ein „<item>“ übertragen. Neben der JID, dem in der Kontaktliste angezeigten Bezeichner und der Gruppe ist der Eintrag *subscription* von zentraler Bedeutung. Dieser Eintrag zeigt an, ob der Kontakt in der eigenen Liste geführt wird und ob der Benutzer auf der Liste des Kontakts steht. Die vier möglichen Kombinationen mit dem entsprechen Wert für *subscription* sind in Tabelle 16 zu finden

Statusübermittlung Nun sollte die Klienteninstanz ihren Zustand übermitteln, damit dieser an alle, die den Benutzer in Kontaktlisten führen, übermittelt werden kann. Diese Mitteilung geschieht durch die folgende Anweisung, die nicht von der Serverinstanz bestätigt werden muss.

Wert	Eingetragen in eigener Liste	Von Kontakt eingetragen
none	nein	nein
to	ja	nein
from	nein	ja
both	ja	ja

Tabelle 16: XMPP - Werte für *subscription* in der Kontaktliste

```
<!-- Klient -> Server -->
<presence/>
```

Diese Anweisung kann ohne Argumente gesendet werden - in der Standardvorgabe wird dies als *available* interpretiert. Je nach Konfiguration des Klienten kann aber ein anderer Status während der Anmeldung übermittelt werden. In diesem Fall wird der gewünschte Status im *type*-Attribut und „<show>“ -*Tag* übertragen (Siehe Anhang A.4.6).

Diese Information muss den Kontakten, die eventuell andere Server nutzen, mitgeteilt werden. Die Verbindung zu dem Server des Kontakts kann wie zwischen Klienten und Server verschlüsselt aufgebaut werden. Je nach Konfiguration des Servers kann die Verbindung nach einem Timeout beendet werden oder für weitere Aktivitäten bestehen bleiben.

Für die eigentliche Nachrichtenübermittlung muss der Server zwischen Kontakten, die der Benutzer auf seiner Kontaktliste führt und Anwender, die den Benutzer führen, unterscheiden.

Alle Kontakte, die den Benutzer in ihrer Kontaktliste führen (also *subscription=„both“* oder *subscription=„from“*) müssen über die Verfügbarkeit informiert werden. Hierfür sendet die Serverinstanz folgende Mitteilung an die Serverinstanz des Kontakts, die wiederum die Nachricht weiterleitet:

```
<!-- Server des Benutzers -> Server des Kontakts -->
<presence xmlns='jabber:client' from='SenderJID' to='EmpfaengerJID'/>
```

Im Weiteren muss der Server den Status der Kontakte, die der Benutzer auf seiner Kontaktliste eingetragen hat (*subscription=„both“* oder *subscription=„to“*), von den entsprechenden Servern anfordern:

```
<!-- Server des Benutzers -> Server des Kontakts -->
<presence xmlns='jabber:client' from='SenderJID'
          to='EmpfaengerJID' type='probe'/>
```

Ab diesem Zeitpunkt erscheint der Benutzer allen Kontakten, die ihn auf der Kontaktliste führen, als angemeldet und somit für eine Kommunikation verfügbar.

Kontaktstatus Zu diesem Zeitpunkt kennt der Klient zwar seine Kontaktliste, aber der Status der einzelnen Mitglieder ist nur seinem Server bekannt. Wie bei einem späteren Statuswechsel eines Kontakts übermittelt der Server des Kontakts diese Informationen, ohne das eine entsprechende Anfrage gesendet wurde. Dies ist möglich, da der Server des Kontakts zuvor den Status des angemeldeten Benutzers empfangen hat.

Da der einzige Unterschied zu einer späteren Statusübermittlung darin besteht, dass bei der Anmeldung Informationen zu allen Benutzern zur gleichen Zeit übertragen werden, wird an

dieser Stelle auf Anhang A.4.6 verwiesen, wo dies detaillierter behandelt wird.

A.4.5. Einfügen eines Benutzers in die Kontaktliste

In XMPP werden für das Einfügen von neuen Benutzern „<presence>“-Anweisungen verwendet. Zum Schutz der Statusinformationen muss bei dem Einfügen die Erlaubnis des Kontakts eingeholt werden.

Bei den im Folgenden vorgestellten Anweisungen wird auf eine explizite Angabe des *from*-Attributs verzichtet. Bei der Übermittlung zwischen Serverinstanzen und von einer Server- zu einer Klienteninstanz muss dieses Attribut eingefügt werden.

Um einen entfernten Benutzer in die Kontaktliste einfügen zu können muss diesem die folgende Nachricht übermittelt werden:

```
<!-- Benutzer -> Kontakt -->
<presence to='JID_des_gewuenschten_Kontakts' type='subscribe'/>
```

Nachdem der Kontakt die *subscribe*-Anfrage empfangen hat, kann er entscheiden, ob der Benutzer Einsicht in den Status erhalten darf. Ist dies der Fall, antwortet der Kontakt mit einem *Subscribed*:

```
<!-- Kontakt -> Benutzer -->
<presence to='JID_des_authentisierten_Benutzers' type='subscribed'/>
```

Die Eintragung dieser Informationen geschieht direkt auf den Server des Kontakts und des Benutzers. Hierbei wird dem Kontakt der Zustand *from*, *to*, *none* oder *both* (siehe Tabelle 16 auf Seite 144) zugewiesen. Bei Änderungen des Zustands des Benutzers sendet der Server auf Grund dieser Eintragung die benötigten Informationen direkt an den Kontakt.

Möchte der Kontakt diese Anfrage abweisen, kann er eine *unsubscribed*-Anweisung übermitteln:

```
<!-- Kontakt -> Benutzer -->
<presence to='JID_des_abgewiesenen_Benutzers' type='unsubscribed'/>
```

Diese *unsubscribed*-Anweisung kann auch in dieser Form verwendet werden um einem Benutzer die Einsicht in den Status, die zuvor erteilt wurde, wieder zu entziehen. Um einen Kontakt wieder aus der Kontaktliste zu entfernen kann eine *unsubscribe*-Anweisung übermittelt werden:

```
<!-- Benutzer -> Kontakt -->
<presence to='JID_des_Kontakts' type='unsubscribe'/>
```

Ab diesem Zeitpunkt erhält der Benutzer keine weiteren Informationen über den Status des Kontakts.

A.4.6. Statusverwaltung

Wie bei vielen *Instant Messaging*-Systemen sind auch in *Jabber* Statusmöglichkeiten vorgesehen. Um den aktuellen Status des Kontakts einsehen zu dürfen, wird die Erlaubnis des Kontakts benötigt (siehe Anhang A.4.5).

Die Klienteninstanz, die eine Statusänderung durchführen will, sendet die Information an die angeschlossene Serverinstanz. Diese leitet diese an den Server des Kontakts weiter. Die Benachrichtigung der Benutzer, die den Status des Kontakts einsehen dürfen, geschieht automatisch. Eine explizite Anforderung des Status ist aber trotzdem möglich, was besonders direkt nach der Anmeldung von Bedeutung ist (siehe Anhang A.4.4, Abschnitt über Statusübermittlung, insbesondere *type=probe*).

Eine vollständige Struktur einer hierfür verwendeten Mitteilung ist im folgenden Beispiel zu finden:

```
<presence xmlns:lang='en'
          from='JID_des_Kontakts'
          to='JID_des_zu_informierenden_Benutzers'
          type='Statustyp'/>
  <status>Optionale, Frei waehlbare, informierende Zeichenkette</status>
  <show>Zustand, wenn nicht angegeben: Online</show>
  <priority>numerische Prioritaet der Ressource</priority>
</presence>
```

Als Statustyp sind alle Zustände aus der Tabelle 15 auf Seite 140 möglich. In dem Zusammenhang der Statusübermittlung sind insbesondere die Werte *available* und *unavailable* interessant. Für die Anzeige des Zustands im Klienten ist das „<show>“-*Tag* von Bedeutung. Die in der RFC3921[15] definierten Werte sind in Tabelle 17 zu finden. Wird kein „<show>“ verwendet, interpretiert dies der Klient als *online*. Nicht unter XMPP definierte Zustände können ebenfalls übertragen werden. Der Zustand *invisible* ist besonders für die Nutzung von Gateways zu anderen *Instant Messaging*-Systemen sinnvoll.

Wert	**Bedeutung**
away	Zur Zeit nicht anwesend
chat	Interesse an Kommunikation
dnd	Bitte nicht stören (*Do Not Disturb*)
xa	Benutzer ist längerfristig abwesend (*eXtended Away*)

Tabelle 17: XMPP - Definierte Zustände des Status

Wie schon die Anmeldung gezeigt hat, bei der nur die Anweisung „<presence/>“ übermittelt wurde, sind alle Angaben optional. Die Praxis hat auch gezeigt, dass viele sinnlose Kombinationen wie *type='unavailable'* und „<status>chat</status>“ ebenfalls möglich sind und nicht abgefangen werden. Letztendlich ist es die Aufgabe der Klienteninstanz, die übertragenen Informationen sinngemäß zu deuten.

Wenn die Klienteninstanz die Argumente *xmlns*, *to* und *from* nicht angegeben hat, fügt der Server diese hinzu. Aber insbesondere das *to*-Element ist von zentraler Bedeutung: Wird dieses nicht angegeben, wird diese Statusänderung an alle Kontakte, die den Status einsehen dürfen (also *subscription=„both“* oder *subscription=„from“*), übermittelt. Dies wird als **Presence Broadcast** bezeichnet.

Im Weiteren besteht die Möglichkeit, nur an ausgewählte Kontakte eine Statusinformation zu senden. Ein Beispiel hierfür ist die Erteilung der Erlaubnis für die Eintragung in die Kontaktliste, bei der ebenfalls „<presence>“-Anweisungen übermittelt wurden (siehe Anhang A.4.5). Diese **Direct Presence** ermöglicht auch die Übermittlung eines ausgewählten Zustands

an einen Benutzer. Ein Beispiel hierfür ist bei Desinteresse an einer Kommunikation mit einem bestimmten Kontakt die Zusendung eines *away*-Zustand, ohne dass dieser einen Unterschied zu einem *Presence Broadcast* erkennt. Ohne diesen Kontakt von der Kontaktliste zu entfernen steht der Benutzer weiteren Kontakten beispielsweise als „interessiert an einer Kommunikation“ (<*show*>*chat*</*show*>) zur Verfügung.

A.4.7. Klientenkommunikation

Der Empfang von Nachrichten kann nur direkt vom Empfänger verhindert werden. Die mit der Kontaktliste verbundene Authentisierung bezieht sich nur auf die Übermittlung des Status. Auch hier übermittelt der Sender die Nachricht seinem Server, der wiederum über den Server des Empfängers diese zustellt. In der Regel kann die Nachricht auch auf dem Server des Empfängers zwischengespeichert werden, wenn dieser zur Zeit nicht angemeldet ist.

Prinzipiell haben die Nachrichten auf den einzelnen Teilstrecken (Sender-Server, Server-Server und Server-Empfänger) die gleiche Gestalt. Als einer der wenigen Unterschiede braucht der Sender das *from*-Attribut nicht zu belegen - dies kann nachträglich durch den Server geschehen.

Eine Angabe eines *to*-Arguments ist hingegen zwingend erforderlich - wird dieses Attribut nicht gesetzt, dann wird die Nachricht nicht vom Server weitergeleitet. Als *to* muss auch nicht unbedingt die komplette *JID* angegeben werden. Die Angabe des Benutzer- und des Rechnernamens ohne Ressource ist ausreichend. Bei einer Antwort auf eine Nachricht wird aber in der Regel die Ressource durch den Klient ergänzt. Als *type*-Argument wird eine Nachrichtenklasse angegeben. Beispiele dafür sind in Tabelle 18 zu finden.

Jede Nachricht sollte aus der eigentlichen Nachricht zwischen den „<body>“-*Tags* und einer optionalen Betreffzeile bestehen.

Im Folgenden wird eine Beispielnachricht skizziert:

```
<message xmlns='jabber:client'
         xml:lang='en'
         from='JID_des_Senders'
         to='JID_des_Empfaengers'
         type='Nachrichtenklasse'>
  <subject>Betreffszeile</subject>
  <body>Und hier steht die eigentliche Nachricht</body>
</message>
```

Wert	Bedeutung
normal	Normale Nachrichten, in der Regel pro Nachricht ein Fenster
chat	Diskussion, Textnachrichtenaustausch in einem Fenster
groupchat	Diskussion mit mehreren Benutzern
headline	Nachrichtenticker, Systemnachrichten ...
error	Fehlermeldungen

Tabelle 18: XMPP - Nachrichtenklassen

A.4.8. Regelungen zur Server-Server-Kommunikation

Wie in Kapitel 10.1 vorgestellt, besteht ein Jabber-Netz aus mehreren Serverinstanzen, an denen Klienteninstanzen angemeldet sind. Bei dieser Architektur werden die Informationen von den Klienteninstanzen zu einer Serverinstanz weitergeleitet, die die Informationen zu dem Server des Kontakts übermittelt, der diese Information wiederum zum Ziel sendet. Dies wird schematisch in Abbildung 31 vorgestellt.

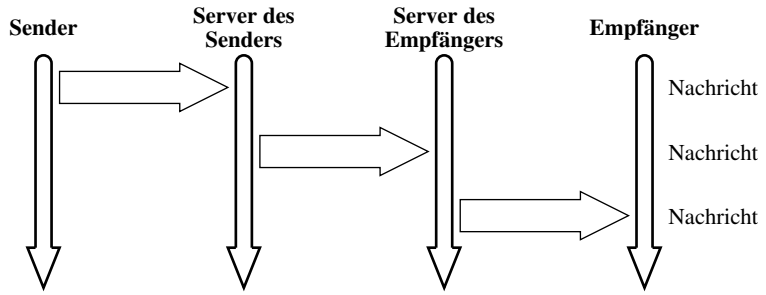

Abbildung 31: XMPP - Informationsübermittlung von Klienten über Server

Wird dieser Informationsstrom detaillierter betrachtet, fällt auf, dass sich die einzelnen übermittelten Informationen (Klient-Server, Server-Server und Server-Klient) nur wenig unterscheiden.

Die einzige Ausnahme hierbei ist der erste Abschnitt zwischen Klienten und Server. In der Praxis übermittelt die Klienteninstanz nur so wenige Informationen wie nötig - das *from* und *xml:lang*-Attribut ist schon aus dem „<stream:stream>“ bekannt und das *to*-Argument wird nicht immer benötigt.

Diese unvollständige Nachricht nimmt die Serverinstanz an und ergänzt die fehlenden Attribute. Sie versucht, die Nachricht nun anhand des *to*-Arguments zu bearbeiten. Entspricht der Hostteil der *JID* einem ihrer Servernamen, muss sie diese Nachricht anhand des Benutzernames zu dem Kontakt weiterleiten.

Bei einem fremden Servernamen versucht sie, die Nachricht an dieses System weiterzuleiten. Ist dies nicht möglich, muss sie den Sender mit einer Fehlermeldung hierüber unterrichten. Die Serverinstanz des Empfängers sendet die Nachricht in der Regel unverändert zum Ziel weiter. Im Folgenden werden einige Fälle in diesem Zusammenhang detaillierter vorgestellt.

Fehlende To-Angabe Fehlt bei einer empfangenen Nachricht die *To*-Angabe, so versucht die Serverinstanz die Nachricht im Sinne des Senders zu bearbeiten. Da Nachrichten von anderen Serverninstanzen immer ein *To*-Feld besitzen, ist dies nur bei der Klient-Server-Kommunikation relevant.

„<presence>“-Anweisungen ohne *to*-Argument werden als *Broadcast*-Mitteilung behandelt - diese Statusmitteilung muss an alle Kontakte, die den Status des an dieser Ressource angeschlossenen Benutzers einsehen dürfen, weitergeleitet werden.

Weniger einfach ist das Problem eines fehlendes *To*-Arguments in einem „<iq>“-*get* oder -*set* zu lösen. Hier muss die Serverinstanz versuchen, die Zieladresse anhand des angegebenen XML-

Namensraum zu bestimmen. Ist dies nicht möglich, wird eine entsprechende Fehlermeldung versendet.

Fehlende From-Adresse Untersuchungen haben gezeigt, dass angemeldete Klienteninstanzen nur sehr selten ein *From*-Argument übermitteln. Soll diese Anweisung an eine andere Instanz (Klient oder Server) weitergeleitet werden, muss die Serverinstanz diese Information in die Anweisung eintragen, bevor sie sie weiterleitet.

Diesen Eintrag kann die Serverinstanz sehr leicht aus dem „<stream:stream>“-Block extrahieren, der die Anweisung umschließt.

Fremde Domains Besteht das *to*-Argument einer Anweisung aus einer JID, deren Hostteil nicht dem des Servers entspricht, muss er diese Anweisung an dem betreffenden Server weiterleiten. Hierfür wird eine TCP-Verbindung zu dem entfernten Server benötigt, wobei zwei Möglichkeiten berücksichtigt werden können:

- Zu einem früheren Zeitpunkt wurde schon eine TCP-Verbindung mit einem einleitenden „<stream:stream>“ aufgebaut. Diese Verbindung kann für die Übermittlung verwendet werden.
- Es besteht noch keine TCP Verbindung zwischen den beiden Servern. In diesem Fall muss eine Verbindung aufgebaut werden, die kryptographisch geschützt werden kann. Diese Kommunikation erfordert einen „<stream:stream>“-Austausch.

Konnte keine Verbindung zu dem Server hergestellt werden, muss der Sender hierüber informiert werden.

Benutzer in lokaler Domain Empfängt die Serverinstanz eine Nachricht mit einer JID, für die sie zuständig ist, muss zwischen folgenden Möglichkeiten unterschieden werden:

- Wurde eine komplette *JID* (*benutzer@hostteil/ressource*) als Empfänger angegeben und ist ein Benutzer mit genau dieser *JID* (also auch mit diesem Ressourcennamen) angemeldet, muss die Serverinstanz die Nachricht an diesen Benutzer weiterleiten.
- Existert der adressierte Benutzername nicht auf dem Server, beantwortet dies die Serverinstanz je nach Anweisung mit einer Fehlermeldung oder ignoriert diese.
- Existiert bei der Anfrage einer kompletten *JID* zwar der angemeldete Benutzername auf diesem Server, ist aber keine Klienteninstanz mit diesem Ressourcennamen angemeldet, ignoriert die Serverinstanz „<presence>“-Anweisungen. Auf „<iq>“-Anweisungen muss mit „<service-unavailable/>“ geantwortet werden und „<message>“-Anweisungen werden an eine angemeldete Ressource des Benutzers weitergeleitet.
- Wurde die Ziel-*JID* in der Form *benutzer@hostteil* angegeben, wird die Nachricht dem Benutzer übermittelt, der mit einer beliebigen Ressource angemeldet sein kann. Sind mehrere Ressourcen von diesem Benutzer angemeldet, entscheidet die Serverinstanz anhand

der der Ressource zugewiesenen Priorität, wer die Nachricht zugestellt bekommt. Sind mehrere Ressourcen mit der gleichen Priorität angemeldet, wird die Nachricht an die zuletzt angemeldete Ressource gesendet.

- Beinhaltet die Ziel-*JID* einen Benutzernamen des Servers, der zwar registriert, aber zur Zeit nicht angemeldet ist, speichert die Serverinstanz die Nachricht ab und sendet diese dem Benutzer, nachdem er sich wieder angemeldet hat.

A.5. Silc

A.5.1. Datagrammformat

Silc nutzt, wie die bisher vorgestellten Kommunikationssysteme, ebenfalls die Verarbeitungsschicht des ISO/OSI- beziehungsweise des TCP/IP-Referenzmodells. Abbildung 32 zeigt den Aufbau eines Datagramms, der im Folgenden näher vorgestellt wird.

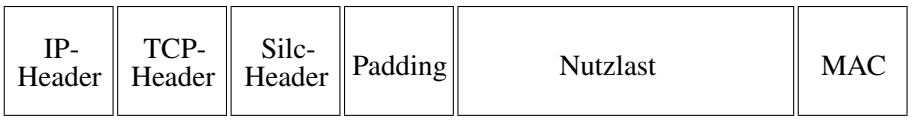

Abbildung 32: Silc - Format eines Datagramms

Das **Padding**-Feld beinhaltet keine relevante Nutzlast und dient primär dazu, das Gesamtdatagramm auf eine volle Byteanzahl oder Blockgröße für die Verschlüsselung aufzufüllen. Das *Padding*-Feld kann eine maximale Größe von 128 Byte annehmen und wird mit Zufallszahlen gefüllt.

An das *Padding*-Feld schließt sich die **Nutzlast** an. An letzter Stelle folgt der *message authentication code* (**MAC**), der die Integrität des Datagramms mit Hilfe einer Prüfsumme sichern soll.

Alle übertragenen Byte sind *UTF-8* und *Big Endian* (das *Most Significant Byte* wird als erstes übertragen) codiert.

A.5.2. Header

Der Silc-Header, der in Abbildung 32 vorgestellt wird, schließt sich an den TCP-Header an. Er beinhaltet unter anderem, welcher Silc-Benutzer das Datagramm versendet hat und an wen es adressiert ist. Außerdem können die Silc-Systeme, die dieses Datagramm empfangen, anhand des Headers erkennen, welche Funktion das Datagramm erfüllt.

Die Architektur des Headers, der eine variable Größe besitzt, wird in den folgenden Abschnitten näher erläutert.

Payload Length Die Länge des gesamten Silc-Datagramms wird in diesem Feld eingetragen. Hierfür wird eine Größe von zwei Byte reserviert. Dabei werden jedoch die *Padding*-Bits nicht berücksichtigt.

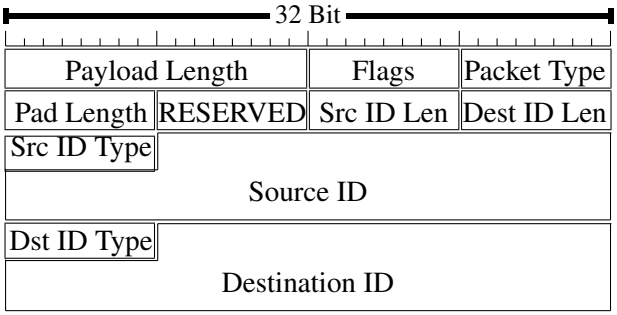

Abbildung 33: Silc - Aufbau eines Headers

Flags Neben TCP-Flags können auch Flags für die Kommunikation unter Silc gesetzt werden. Obwohl acht Bits für die Flags reserviert werden, werden zum Zeitpunkt dieser Diplomarbeit nur vier ausgewertet. Die möglichen Flags, die gesetzt werden können, sind Tabelle 19 zu entnehmen.

Wert	Name	Bedeutung
0001 (0x01)	Private Message Key	Das Datagramm beinhaltet Informationen, die mit einer *Client-to-Client*-Verschlüsselung geschützt sind. Die zwischen den beiden Klienteninstanzen gelegenen Router und Server können dieses Datagramm nicht entschlüsseln.
0010 (0x02)	List	Zeigt an, dass das Datagramm weitere Datagramme enthält, deren Typ durch das *Packet Type* Feld angegeben wird. Die Nutzlasten der Datagramme werden aneinander gehängt. Die Datagramme werden durch Berechnung der Länge der einzelnen Datagramme und unter Betrachtung der einzelne Datagramme bearbeitet.
0100 (0x04)	Broadcast	Dieses Flag wird von nur von Silc-Routern gesetzt und ausgewertet. Empfängt eines dieser Systeme ein Datagramm mit diesem Flag, muss es weitergeleitet werden.
1000 (0x08)	Compressed	Dieses Flag signalisiert dem Empfänger, dass die Nutzlast dieses Datagramms mit einem Kompressionsverfahren verkleinert wurde und beim Empfangen dekomprimiert werden muss.

Tabelle 19: Silc - Flags des Silc-Headers

Packet Type Dieses Feld beschreibt, welche Funktion durch das Datagramm erfüllt wird. Eine Übersicht der unterstützten Datagrammarten ist Tabelle 20 zu entnehmen.

Um die Tabelle 20 übersichtlich zu halten, wurde vor jedem Datagrammnamen das Schlüsselwort *SILC_PACKET_* entfernt. Beispiel: *DISCONNECT* (01) statt *SILC_PACKET_DISCONNECT*.

Wert	Name	Bedeutung
00	NONE	Dieser Datagrammtyp wird nicht genutzt und ist für eine spätere Verwendung reserviert.

Fortsetzung auf der nächsten Seite

Wert	Name	Bedeutung
01	DISCONNECT	Schließt einen Kommunikationskanal. Die Gründe hierfür werden in der Nutzlast angegeben.
02	SUCCESS	Die Aktion wurde erfolgreich ausgeführt. Der Status der Aktion wird in der Nutzlast angegeben.
03	FAILURE	Gibt an, dass eine Aktion fehlgeschlagen ist. Die Art des Fehlers wird in der Nutzlast angegeben.
04	REJECT	Kann gesendet werden, wenn die Ausführung einer Aktion abgelehnt wurde. Der Grund hierfür ist in der Nutzlast zu finden.
05	NOTIFY	Wird für Benachrichtigungen verwendet. Es kann sowohl von Server- als auch von Routerinstanzen versendet werden.
06	ERROR	Wird von der Serverinstanz gesendet, wenn ein Fehler aufgetreten ist.
07	CHANNEL_MESSAGE	Diese Art von Datagrammen werden verwendet, um Nachrichten an einen Channel zu schicken. Das Datagramm beinhaltet die *Channel ID* und wird mit dem Schlüssel geschützt, der für diesen Channel generiert wurde.
08	CHANNEL_KEY	Dieses Datagramm wird von der Serverinstanz verschickt und beinhaltet einen neuen *Channel Key*. Dieser wird in regelmäßigen Abständen und wenn ein Benutzer den Channel betritt oder verlässt gewechselt.
09	PRIVATE_MESSAGE	Dieses Datagramm wird versendet, um eine Nachricht zu einer anderen Klienteninstanz zu schicken. Diese Nachricht kann zusätzlich mit einem privaten Schlüssel geschützt werden, so dass sie nur von der Zielklienteninstanz entschlüsselt werden kann.
0A	PRIVATE_MESSAGE_KEY	Wird benutzt, um einen Schlüssel für eine private Kommunikation zu einer anderen Klienteninstanz zu schicken.
0B	COMMAND	Wird verwendet, um Steuerbefehle zu senden.
0C	COMMAND_REPLY	Dieses Datagramm wird als Antwort zu einem *SILC_PACKET_COMMAND*-Datagramm versendet.
0D	KEY_EXCHANGE	Dieses Datagramm initiiert den Schlüsselaustauschs nach dem *Silc Key Exchange*-Protokoll. In der Nutzlast übermittelt der Initiator, welche Verschlüsselungsverfahren von seiner Seite unterstützt werden. Aus den übermittelten Verfahren wählt der Empfänger ein Verfahren aus und teilt diese Entscheidung den Initiator unter Verwendung von *KEY_EXCHANGE* mit.
0E	KEY_EXCHANGE_1	Dieses Datagramm ist ein Bestandteil des Schlüsselaustauschs nach dem *Silc Key Exchange*-Protokoll. Dieses Datagramm beinhaltet die Schlüssel und wird vom Initiator des Schlüsselaustauschs versendet.
0F	KEY_EXCHANGE_2	Dieses Datagramm ist ein Bestandteil des Schlüsselaustauschs nach dem *Silc Key Exchange*-Protokoll. Dieses Datagramm beinhaltet die Schlüssel und wird vom Empfänger des Schlüsselaustauschs versendet.
10	CONNECTION_AUTH_REQUEST	Dieses Datagramm wird zu Beginn des *SILC Connection Authentication*-Protokolls versendet. Es wird verwendet, um die Authentisierungsmethode auszuhandeln. Möglichkeiten hierfür sind die Authentisierung durch eine Passphrase oder mit Hilfe eines öffentlichen Schlüssels.

Fortsetzung auf der nächsten Seite

Wert	Name	Bedeutung
11	CONNECTION_AUTH	Dieses Datagramm leitet die Authentisierung nach dem *Silc Connection Authentication*-Protokoll ein. Das *CONNECTION_AUTH*- Datagramm wird von dem Initiator des Informationsaustauschs gesendet und beinhaltet als Nutzlast die Authentisierungsinformationen. Dies kann eine Passphrase oder der öffentliche Schlüssel sein.
12	NEW_ID	Wird von Servern und Routern genutzt, um anderen Routern die Anmeldung neuer, lokaler Klienten mitzuteilen.
13	NEW_CLIENT	Wird von der Klienteninstanz verwendet, um sich an dem Silc-Netz anzumelden.
14	NEW_SERVER	Wird zur Bekanntgabe eines neuen Servers im Silc-Netz genutzt.
15	NEW_CHANNEL	Wenn ein neuer Channel generiert wird, wird dieser durch ein Datagramm dieses Typs allen weiteren Routern bekannt gegeben.
16	REKEY	Fordert einen neuen *Session-Key* an. Dieses Datagramm besitzt keine Nutzlast.
17	REKEY_DONE	Signalisiert dem Empfänger, dass die Neuerstellung des *Session-Keys* abgeschlossen ist. Ab diesem Zeitpunkt muss der neue Schlüssel verwendet werden.
18	HEARTBEAT	Soll feststellen, ob die Verbindung noch aktiv ist.
19	KEY_AGREEMENT	Wird verwendet, um einen Schlüsselaustausch mit einer anderen Klienteninstanz zu initiieren. Dieser kann für eine *Client-to-Client*-Verschlüsselung genutzt werden.
1A	RESUME_ROUTER	Wird gesendet, wenn der Backup-Router nach einem Ausfall wieder abgelöst werden soll.
1B	FTP	Wird für die Übertragung von Dateien verwendet.
1C	RESUME_CLIENT	Nachdem die Verbindung zu einer Klienteninstanz abgebrochen wurde, kann diese die Verbindung damit wieder herstellen.
FF	MAX	Für spätere Erweiterungen des Protokolls reserviert.

Tabelle 20: Silc - Im Header definierbare Datagrammarten

Pad Length Dieses Feld gibt an, mit wievielen Byte das Datagramm aufgefüllt wurde. Obwohl das Feld ein Byte groß ist, kann das Datagramm statt mit den möglichen 256 Byte nur mit 128 Byte aufgefüllt werden.

RESERVED Das ein Byte große Feld ist für eine spätere Verwendung reserviert und wird mit Null belegt.

Src ID Len Das *Source ID Len*-Feld legt die Größe der später folgenden *Source ID* fest. Hierfür wird ein Byte reserviert.

Dest ID Len Das *Dest ID Len*-Feld legt die Größe der später folgenden *Destination ID* fest. Hierfür wird ein Byte reserviert.

Src ID Type Dieses ein Byte große Feld beschreibt den Typ des *Source ID*-Feldes. In Tabelle 21 werden die möglichen Einträge charakterisiert.

Wert	Bedeutung
0	No ID
1	Server ID
2	Client ID
3	Channel ID

Tabelle 21: Silc - Mögliche Identifikatoren im Silc-Header

Source ID Identifiziert den Sender des Datagramms anhand seiner ID. Um welche Art von ID es sich dabei handelt beschreibt das Feld *Src ID Type*. Die Länge des Feldes wird durch das Header-Feld *Src ID Len* bestimmt. Das Format des Identifikators ist von dem Typ abhängig. In den Kapiteln 12.1.1, 12.1.2 und 12.1.3 werden die möglichen Formate wie *Client ID*, *Server ID* und *Router ID* vorgestellt.

Dst ID Type Dieses ein Byte große Feld beschreibt den Typ der ID aus dem *Destination ID*-Feld. In Tabelle 21 werden die möglichen Einträge charakterisiert.

Destination ID Bestimmt, an welchen Empfänger das Datagramm geschickt werden soll. An Hand dieses Feldes erkennt die Serverinstanz, dass das Datagramm an eine bestimmte Klienteninstanz weitergeleitet werden muss. Um welche Art von ID es sich dabei handelt, beschreibt das Feld *Dst ID Type*. Die Länge des Feldes wird durch das Header-Feld *Dest ID Len* bestimmt. Das Format des Identifikators ist von dem Typ abhängig. In den Kapiteln 12.1.1, 12.1.2 und 12.1.3 werden die möglichen Formate vorgestellt.

A.5.3. Datagramme für den Schlüsselaustausch über das SKE-Protokoll

Die Sicherheit von Silc beruht unter anderem auf einem sicheren Schlüsselaustausch. Eine Möglichkeit dazu bietet das SKE-Protokoll, das in Kapitel 12.3.2 beschrieben wird.

Für das SKE-Protokoll werden zwei Datagrammenarten benötigt. Die Initiierung und das Aushandeln der unterstützten Kryptographieverfahren finden über *SILC_PACKET_KEY_EXCHANGE*-Datagramme statt. Die Parameter, die für den Austausch des symmetrischen Schlüssels gesendet werden müssen, übermittelt das *SILC_PACKET_ KEY_EXCHANGE_1/2*-Datagramm. In diesem wird als Nutzlast unter anderem der öffentliche Schlüssel übertragen.

Struktur des *SILC_PACKET_KEY_EXCHANGE*-Datagramms Wie jedes Datagramm, das für eine Kommunikation über Silc verwendet wird, wird das *SILC_PACKET_KEY_EXCHANGE*-Datagramm mit dem allgemeinen Silc-Header versehen. Dieser wurde im Anhang A.5.2 vorgestellt. Diesem schließt sich die Nutzlast an, die in Abbildung 34 dargestellt und deren Felder im Anschluss näher erläutert werden.

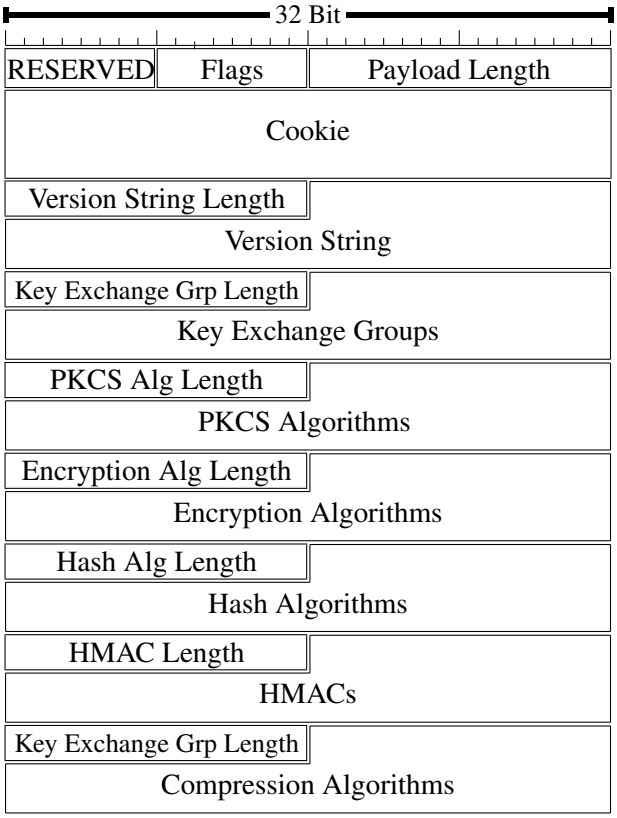

Abbildung 34: Silc - Aufbau eines *SILC_PACKET_KEY_EXCHANGE*-Datagramms

RESERVED Dieses ein Byte große Feld ist reserviert und wird vom Sender mit Nullen gefüllt.

Flags Ein Byte ist für Flags reserviert. Durch das Setzen der Flags kann zum Beispiel *Perfect Forward Secrecy* (PFS, siehe Anhang B.9) aktiviert werden. Zusätzlich kann dem Empfänger verdeutlicht werden, dass sich der Initialisierungsvektor (IV, siehe Anhang B.6) verschlüsselt in dem Datagramm befindet.

Payload Length Das zwei Byte große *Payload Length* gibt die Gesamtlänge der Nutzlast in Byte an.

Cookie Der *Cookie* ist eine 16 Byte große Zufallszahl, die verhindern soll, dass (gefälschte) Antworten, die vor der Anfrage gesendet wurden, berücksichtigt werden. Daher muss in jedem Antwortdatagramm der gleiche Cookie wie in dem Ausgangsdatagramm eingefügt werden.

Version String Length Durch dieses zwei Byte große Feld wird festgelegt, wie groß (in Byte) das *Version String*-Feld maximal werden kann.

Version String In dem *Version String* werden die unterstützten Protokoll- und Softwareversionen eingetragen. Die Größe dieses Feldes ist flexibel und wird im *Version String Length*-Feld festgelegt.

Key Exchange Grp Length Anzahl der benötigten Byte im *Key Exchange Groups*-Feld. Zur Angabe dieser Größe stehen zwei Byte zur Verfügung.

Key Exchange Groups In diesem Feld wird eingetragen, wie der Schlüsselaustausch erfolgen kann, da es drei verschiedene Möglichkeiten des Diffie-Hellman-Merkle-Verfahrens gibt. Die maximale Größe dieses Feldes wurde vorher im *Key Exchange Grp Length*-Feld festgelegt.

PKCS Alg Length Anzahl der Byte, die für das *PKCS Algorithms*-Feld reserviert werden sollen. Die Größe des *PKCS Alg Length*-Feldes ist auf zwei Byte beschränkt.

PKCS Algorithms Das *PKCS Algorithms*-Feld, dessen Größe im *PKCS Alg Length*-Feld festgelegt wird, beinhaltet eine Auflistung der unterstützten PKCS-Algorithmen (siehe Anhang B.12). Bezeichner hierfür sind im Kapitel 12.3.1 auf Seite 83 zu finden.

Encryption Alg Length Anzahl der Byte, die für das *Encryption Algorithms*-Feld reserviert werden sollen. Für das *Encryption Alg Length*-Feld ist ein Platzbedarf von zwei Byte vorgesehen.

Encryption Algorithms Dieses Feld enthält die unterstützten Verschlüsselungsalgorithmen. Das Feld beansprucht die Anzahl der Byte, die in dem Feld *Encryption Alg Length* festgelegt wurden. Bezeichner hierfür sind im Kapitel 12.3.1 auf Seite 83 zu finden.

Hash Alg Length Das zwei Byte große Feld *Hash Alg Length* gibt die Länge des *Hash Algorithms*-Feldes in Byte an.

Hash Algorithms Das Feld beinhaltet eine Auflistung der unterstützten Hash-Algorithmen. Die Größe ist variabel und wird durch das *Hash Alg Length*-Feld bestimmt. Bezeichner hierfür sind im Kapitel 12.3.1 auf Seite 83 zu finden.

HMAC Length Der Inhalt des *HMAC Length*-Feldes ist auf zwei Byte beschränkt und entspricht der Anzahl der für das *HMACs*-Feld benötigten Byte.

HMACs In diesem Feld werden die von dem Absender unterstützten *Hashed Message Authentication Code*-Algorithmen (siehe Anhang B.5) eingetragen. Die Größe ist auf den im *HMAC Length*-Feld angegebenen Wert beschränkt. Bezeichner hierfür sind im Kapitel 12.3.1 auf Seite 83 zu finden.

Compression Alg Length Für das *Compression Alg Length*-Feld sind zwei Byte im Datagramm vorgesehen. Der Inhalt beschreibt die Anzahl der benötigten Byte für das *Compression Algorithms*-Feld.

Compression Algorithms Die zu übertragende Nutzlast kann mit den hier eingetragenen Algorithmen komprimiert werden. Die Anzahl der Byte, die für die Angabe der Algorithmen notwendig sind, werden durch das Datagrammfeld *Compression Alg Length* festgelegt.

Struktur der *SILC_PACKET_KEY_EXCHANGE_1/2*-Datagramme Zum Übertragen der Schlüsselparameter werden die *SILC_PACKET_KEY_EXCHANGE_1/2*-Datagramme verwendet. Wie alle Silc-Datagramme beginnen diese mit dem Silc-Header (siehe Anhang A.5.2), der in diesem Abschnitt nicht berücksichtigt wird. Anhand des *Packet Type*-Feldes ist erkennbar, von wem das Datagramm stammt. Versendet der Initiator ein *SILC_PACKET_KEY_EXCHANGE_1*-Datagramm, beinhaltetet das *Packet Type*-Feld den Wert **0x0E**. Folgt auf das Datagramm des Initiators die Antwort des Empfängers mit einem *SILC_KEY_EXCHANGE_2*-Datagramm, wird in dem Header der Wert **0x0F** eingetragen.

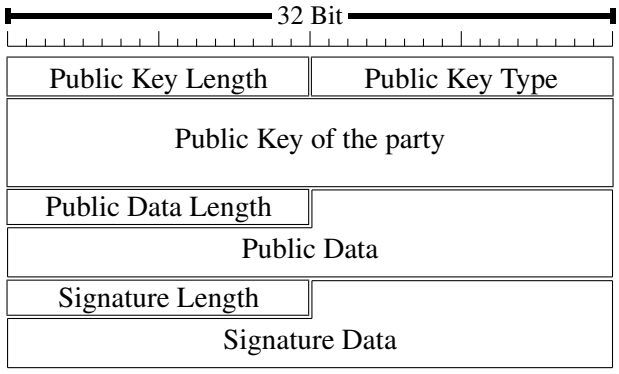

Abbildung 35: Silc - Aufbau eines *SILC_PACKET_KEY_EXCHANGE_1/2*-Datagramms

Public Key Length Das Feld *Public Key Length* spezifiziert die Länge des öffentlichen Schlüssels im Feld *Public Key of the party*. Dafür werden zwei Byte benötigt.

Public Key Type In diesem Feld wird die Art des öffentlichen Schlüssels aus dem *Public Key of the party*-Feld definiert. Der Schlüssel könnte als *SILC style*, *SSH2 style*, *X.509 Version 3 certificate*, *OpenPGP certificate* oder *SPKI certificate* kodiert werden. Für diese Angabe stehen zwei Byte zur Verfügung.

Public Key of the party In dem *Public Key of the party*-Feld wird der öffentliche Schlüssel eingetragen, mit dem die Nachrichten signiert wurde. Das asymmetrische Verschlüsselungsverfahren wird benötigt, um den Sender eindeutig und fälschungssicher zu bestimmen. Das Format

dieses Eintrags, dessen Größe im Feld *Public Key Length* festgelegt wurde, ist am Ende des Abschnitts A.5.3 zu finden.

Public Data Length Dieser Eintrag im Datagramm beinhaltet die Größe des *Public Data*-Feldes in Byte. Das Feld *Public Data Length* ist auf zwei Byte beschränkt.

Public Data Hier wird die Nutzlast des Schlüsselaustauschs eingetragen. Dies können die ausgetauschten Informationen für das Diffie-Hellmann-Merkle-Verfahren (siehe Anhang B.3) sein. Die Größe dieses Feldes ist auf den Eintrag des *Public Data Length*-Feldes beschränkt.

Signature Length Das *Signature Length*-Feld gibt die Länge der Signatur an. Der Eintrag ist auf zwei Byte beschränkt.

Signature Data Als letztes Feld des *SILC_PACKET_KEY_EXCHANGE_1/2*-Datagramms wird das *Signature Data*-Feld verwendet, dessen Größe im *Signature Length*-Feld festgelegt wurde.

Über die Nutzlast des Datagramms wird eine Prüfsumme gebildet und hier eingetragen. Sobald mindestens ein Bit des gesamten Datagramms verändert wird, entspricht die eingetragene Prüfsumme nicht mehr der des Datagramms. Dies verhindert, dass die Modifikation durch einen Angreifer oder Fehler in der Übertragung unentdeckt bleiben.

Die Prüfsumme ist abhängig von dem privaten Schlüssel des Versenders und kann nur mit diesem erstellt werden. Dadurch kann ein Angreifer keine neue Prüfsumme ohne den privaten Schlüssel generieren, nachdem er das Datagramm modifiziert hat.

Nachdem das Datagramm empfangen wurde, muss von dem Empfänger die Prüfsumme mit Hilfe des zu dem Versender gehörigen öffentlichen Schlüssels überprüft werden.

Struktur eines *Public Keys*, der in einem *SILC_PACKET_KEY_EXCHANGE_1/2*-Datagramm übertragen wird In der Nutzlast des *SILC_PACKET_KEY_EXCHANGE_1/2*-Datagramms wird der öffentliche Schlüssel (*Silc Public Key*) übertragen.

In Abbildung 36 ist der Aufbau der Nutzlast des *SILC_PACKET_KEY_EXCHANGE*- Datagramms dargestellt. Der darin übermittelte *Silc Public Key* ist hervorgehoben. Der öffentliche Schlüssel besteht aus folgenden Feldern:

Public Key Length Das Feld *Public Key Length* gibt die Gesamtgröße des öffentlichen Schlüssels an. Hierfür werden 4 Byte benötigt.

Algorithm Name Length Dieses zwei Byte große Feld wird benötigt, um die Gesamtlänge des Feldes *Algorithm Name* festzulegen.

Algorithm Name Hier wird der Name des asymmetrischen Verschlüsselungsalgorithmus eingetragen. Die Größe des Feldes ist vom Algorithmus abhängig, daher wurde die Größe im Feld *Algorithm Name Length* festgelegt. Bezeichner hierfür sind im Kapitel 12.3.1 auf Seite 83 zu finden.

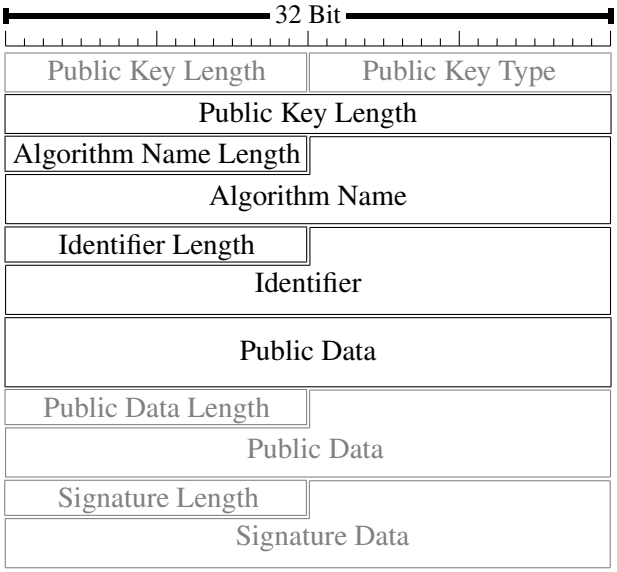

Abbildung 36: Silc - Aufbau eines öffentlichen Schlüssels

Identifier Length Im *Identifier Length*-Feld wird die Größe des folgenden *Identifier*-Feldes angegeben. Dazu werden zwei Byte Speicher benötigt.

Identifier Der Platzbedarf des *Identifier*-Feldes wurde im Feld *Identifier Length* festgelegt. Es beinhaltet einige Informationen über den Besitzer des Schlüssels.

Die Zeichenkette kann folgende Gestalt haben, wobei nur die Angabe des *UN*- und des *HN*-Feldes zwingend erforderlich sind:

UN=<*Benutzername*>, HN=<*Hostname*>[, RN=<*vollständiger Name*>, O=<*Firma*>, C=<*Land*>]

Public Data Das *Public Data*-Feld beinhaltet den eigentlichen Schüssel. Die Größe des Feldes wird explizit angegeben. So wird bei einer RSA-Verschlüsselung vor dem eigentlichen Schlüsselparametern *e* und *n* die jeweilige Länge dieser angegeben.

Mit gleichen Struktur (abwechselnd Platzbedarf und Schlüsselparameter) kann auch ein *DSS*-Schlüssel (*Digital Signature Standard*, siehe Anhang B.2) dargestellt werden. Hierfür werden die Variablen *p*, *q*, *g* und *y* benötigt.

A.5.4. SILC Connection Authentication Protocol

Wenn es vom Serverbetreiber gefordert wird, können sich nur autorisierte Nutzer an der Serverinstanz anmelden. Eine solchen Authentisierung kann über das *SILC Connection Authentication*-Protokoll durchgeführt werden.

Generell wird die Authentisierung durch den Initiator der Verbindung unter der Verwendung eines *SILC_PACKET_CONNECTION_AUTH*-Datagramms eingeleitet. Wenn sich der Initiator

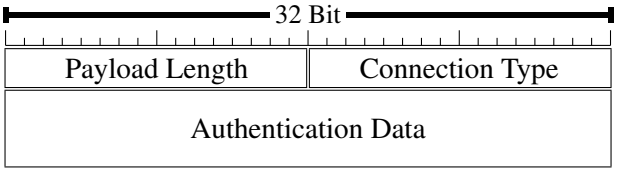

Abbildung 37: Silc - Aufbau der Authentisierungsnutzlast

erfolgreich authentisiert hat, sendet der Empfänger ein *SILC_PACKET_SUCCESS*-Datagramm als Antwort. Andernfalls antwortet er mit einem *SILC_PACKET_FAILURE*-Datagramm.

Datagrammdesign Die Anordnung der Nutzlast, die mit Hilfe des *SILC_PACKET_CONNECTION_AUTH*-Datagramms übertragen werden kann, ist in Abbildung 37 skizziert.

Das zwei Byte große **Payload Length**-Feld legt die Größe der übertragenen Nutzlast fest. Das **Connection Type**-Feld bestimmt, ob für die Authentisierung eine Passphrase oder der öffentliche Schlüssel verwendet wird.

In dem **Authentication Data**-Feld werden die eigentlichen Informationen übertragen. Die Größe ist abhängig von der verwendeten Authentisierungsmethode. Im Weiteren können die Länge der Passphrase und des Schlüssels variieren, womit eine feste Größenangabe dieses Feldes unmöglich ist.

A.5.5. Einsatz eines Backup-Routers

Der Einsatz und die Notwendigkeit eines Backup-Routers wurde ausführlich auf Seite 80 diskutiert. Ziel dieses Abschnitts ist die Betrachtung der Datagramme, die für eine Übergabe der Routingaufgabe an das Backupsystem und wieder zurück benötigt werden.

Für eine Übergabe der Routing-Funktionalitäten an ein anderes Silc-System wird das *SILC_PACKET_RESUME_ROUTER*-Datagramm verwendet. Die mit diesem Datagramm versendete, zwei Byte große Nutzlast ist in der Abbildung 38 skizziert.

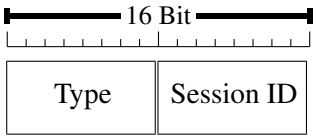

Abbildung 38: Silc - Nutzlast eines *SILC_PACKET_RESUME_ROUTER*-Datagramms

Wie in Abbildung 38 erkennbar, ist ein Byte für den *Type* der Nachricht und ein Byte für die *Session ID* reserviert. Bei der *Session ID* handelt es sich um einen numerischen Wert, der von dem Sender gesetzt wird. Der Empfänger muss in seiner Antwort diesen Wert übernehmen.

Die Werte, die das *Type*-Feld beinhalten kann, sind in Tabelle 22 zu finden. Die Funktionen sind aus der Bezeichnung der Inhalte ersichtlich und werden, wenn nötig, bei der späteren Verwendung erläutert.

Wert	Bezeichnung
1	SILC_SERVER_BACKUP_START
2	SILC_SERVER_BACKUP_START_CONNECTED
3	SILC_SERVER_BACKUP_START_ENDING
4	SILC_SERVER_BACKUP_START_RESUMED
20	SILC_SERVER_BACKUP_START_REPLACED
21	SILC_SERVER_BACKUP_START_USE

Tabelle 22: Silc - Type-Felder des *SILC_PACKET_RESUME_ROUTER*-Datagramms

Übergabe an den Backup-Router Über die Übergabe an den Backup-Router müssen alle Systeme, die den ausgefallenen Router als primären Router eingetragen haben, informiert werden. Bei diesen Systemen kann es sich um normale Server in der *Cell* oder um den Router einer anderen *Cell* handeln.

Wenn die Verbindung von den Systemen zu dem primären Router abbricht, müssen sie im ersten Schritt in ihren Konfigurationsdateien den Backuprouter nachschlagen. Hiernach folgt die Änderung der lokalen Datenstrukturen. In allen Einträgen muss die Adresse des primären Router durch die des Backup-Routers ersetzt werden.

Oft ist es aber so, dass der Backup-Router den Ausfall des primären Routers noch nicht bemerkt hat. Daher senden die Serverinstanzen, die den Backup-Router als primären Router nutzen wollen, ein *SILC_PACKET_RESUME_ROUTER*-Datagramm an den Backup-Router. In der Nutzlast wird dem *type*-Feld der Wert 21 zugeordnet (*SILC_SERVER_BACKUP_START_USE*).

Daraufhin überprüft der Backup-Server, der in der Regel den ausgefallenen Router ebenfalls als primären eingetragen hat, ob dieser wirklich ausgefallen ist. Hierfür verwendet der Backup-Router eine Ping-Anweisung, die in einem *SILC_PACKET_COMMAND*-Datagramm verpackt ist. Ist der primäre Router nicht ausgefallen, sendet der Backup-Router an die Serverinstanz, die die Nachricht geschickt hat, ein allgemeines *PACKET_FAILURE* mit der Nutzlast 21 (4 Byte, Big Endian). Andernfalls übernimmt dieser jetzt die Aufgaben des primären Routers.

Rückgabe an den primärem Router Der Ausfall des primären Routers ist meist zeitlich begrenzt. Ist dessen Funktion wieder hergestellt, sollte er seine Aufgaben wieder übernehmen, da der Backup-Router noch andere Aufgaben zu erfüllen hat und nicht genügend Kapazitäten für einen dauerhaften Router-Betrieb zur Verfügung stellen kann.

Anders als vielleicht erwartet, ist es die Aufgabe des Backup-Routers diese Rückgabe anzustoßen. In dem Protokollablaufdiagramm 39 werden alle *SILC_PACKET_RESUME_ROUTER*-Datagramme zum leichteren Verständnis skizziert.

In der Regel war der ausgefallene Router der primäre Router des Backup-Systems und dieser versucht sich in regelmäßigen Abständen (Voreinstellung: 30 Sekunden), wieder mit diesem zu verbinden. Ist der primäre Router wieder online, sendet der Backup-Router ein *SILC_PACKET_RESUME_ROUTER*-Datagramm mit dem Inhalt *SILC_SERVER_BACKUP_START* (Wert: 1) an den ausgefallenen, primären Server.

Dieser kann einfach überprüfen, ob er wirklich in der näheren Vergangenheit ausgefallen ist und der Backup-Server wirklich für die Aufgabenübernahme vorgesehen war. Ist dies nicht der Fall, antwortet der primäre Server mit einem *SILC_PACKET_FAILURE* mit der Nutzlast 1.

Nachdem dieser Datagrammaustausch erfolgreich ausgeführt wurde, beginnt der Backup-Server, seine Datenstrukturen an den primären Server zu übertragen. Dies soll eine reibungslose Kommunikation der restlichen Systeme untereinander ermöglichen, wenn der primäre Server später das Routing wieder übernimmt.

Zusätzlich sendet der Backup-Router ein *SILC_PACKET_RESUME_ROUTER*-Datagramm mit dem Inhalt *SILC_SERVER_BACKUP_START* (Wert: 1) an die beiden Nachbar-Router. Der erste der beiden verwendet den Backup-Router inzwischen als primären Router und sendet alle Datagramme, die nicht an seine *Cell* adressiert sind, an den Backup-Router. Der zweite empfängt alle Datagramme des Backup-Routers, die dieser nicht direkt zustellen kann. Somit erfahren diese, dass der ursprüngliche primäre Router wieder seine Aufgaben übernehmen wird.

Das *SILC_PACKET_RESUME_ROUTER*-Datagramm wird ebenfalls an alle Server der lokalen *Cell* geschickt, damit diese über den anstehenden Router-Wechsel informiert werden.

Nun bauen alle Systeme eine Verbindung zu dem primären Router auf. Ist dies geschehen, senden alle Systeme ein *SILC_ PACKET_RESUME_ROUTER*-Datagramm mit dem Inhalt *SILC_-SERVER_BACKUP_START_CONNECTED* (Wert: 2) an den Backup-Router. Dadurch erkennt der Backup-Router, dass alle Schritte erfolgreich ausgeführt wurden.

Der Backup-Router kann erst die folgenden Schritten ausführen, wenn er von allen Systemen das *SILC_SERVER_BACKUP_START_CONNECTED* empfangen hat. Dies kann er anhand der

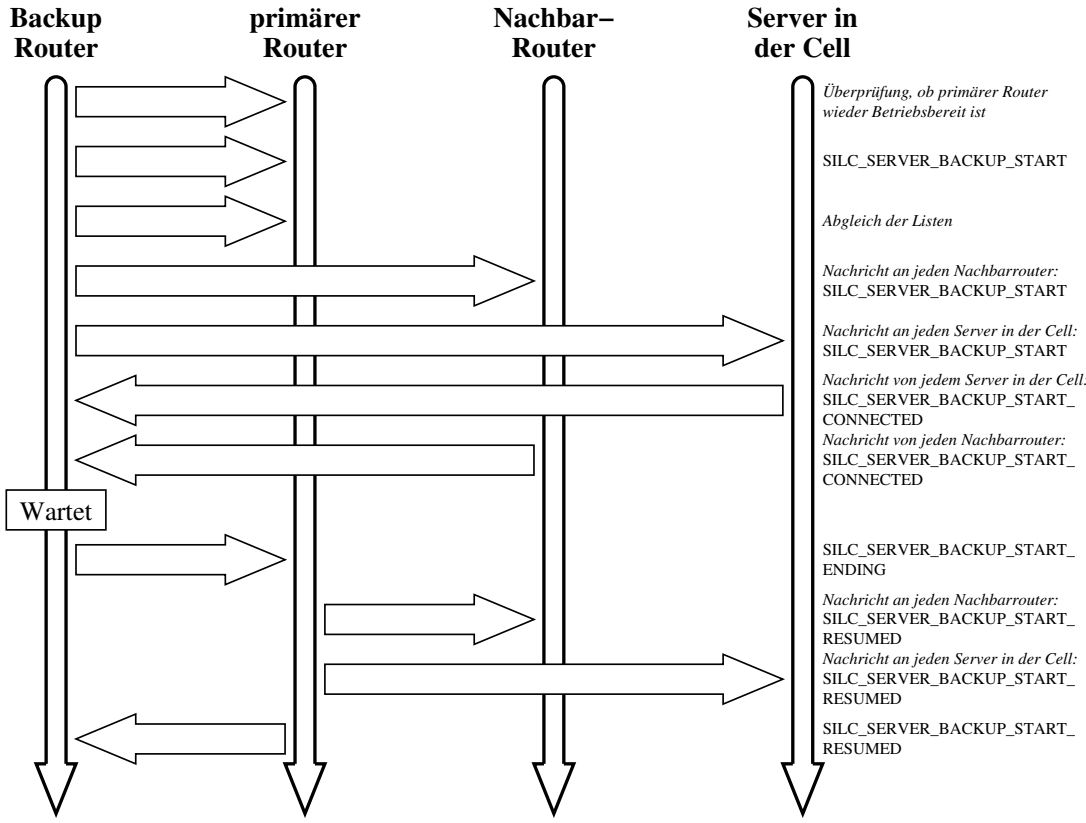

Abbildung 39: Silc - Protokollablauf für Rückgabe am primären Router

versendeten *Session ID* feststellen. Nachdem alle Systeme ihm das Datagramm gesendet haben, schickt der Backup-Server ein *SILC_PACKET_RESUME_ROUTER* mit dem Inhalt *SILC_SERVER_BACKUP_START_ENDING* (Wert: 3) an den primären Router.

Der primäre Router schickt allen Systemen die Nutzlast *SILC_SERVER_BACKUP_START_RESUMED* (Wert: 4) in einem *RESUME_ROUTER*-Datagramm. Dieses zeigt an, dass der Informationsaustausch zwischen Backup-Router und primärem Router beendet ist. Alle Systeme, die dieses Datagramm empfangen, dürfen nun nur noch den primären Router verwenden. Auf allen Systemen werden die Datenstrukturen dahingehend geändert.

Wurde alles erfolgreich ausgeführt, erledigt der Backup-Server wieder seine Server-Tätigkeit.

B. Begriffserklärungen

B.1. Advanced Encryption Standard (AES)

Das symmetrische Kryptographieverfahren wurde von Joan Daemen und Vincent Rijmen entwickelt und ist daher auch als Rijndael-Algorithmus bekannt.

AES ist als Sieger aus einer Ausschreibung des US-amerikanischen *National Institute of Standards and Technology* (NIST) als Nachfolger des veralteten und unsicheren *Data Encryption Standard* (DES) hervorgegangen.

AES kann mit unterschiedlichen Block- und Schlüssellängen arbeiten. Als Werte sind dabei 128, 192 und 256 Bit möglich. Jeder Block wird dabei in eine Tabelle mit vier Zeilen geschrieben, wobei die einzelnen Zellen jeweils ein Byte groß sind. Die Anzahl der Spalten ist von der Blockgröße anhängig und variiert zwischen vier (128 Bit) und acht (256 Bit).

Jeder Block wird nun nacheinander bestimmten Transformationen unterzogen. Aber anstatt jeden Block einmal mit dem Schlüssel zu verschlüsseln, wendet AES verschiedene Teile des Schlüssels nacheinander auf den Klartextblock an. Die Anzahl dieser Runden ist anhängig von der Schlüssellänge und Blockgröße und kann Tabelle 23 entnommen werden (Quelle: [19]).

	Blockgröße 128 Bit	**Blockgröße 192 Bit**	**Blockgröße 256 Bit**
Schlüssellänge 128 Bit	10 Runden	12 Runden	14 Runden
Schlüssellänge 192 Bit	12 Runden	12 Runden	14 Runden
Schlüssellänge 256 Bit	14 Runden	14 Runden	14 Runden

Tabelle 23: Anzahl der AES-Verschlüsselungs-Runden in Abhängigkeit von der Schlüssellänge und der Blockgröße

B.2. Digital Signature Standard (DSS)

Der *Digital Signature Standard* (DSS) wurde am 19. Mail 1994 vom *National Institute of Standards and Technology* (NIST) freigegeben.

DSS ist ein Signaturverfahren. Damit ist es möglich, zu einer Nachricht eine Prüfsumme zu bilden, die den Absender der Nachricht eindeutig bestimmt. Die Prüfsumme wird mit Hilfe eines privaten Schlüssels erzeugt und kann durch den öffentlichen Schlüssel überprüft werden.

DSS nutzt ähnlich wie RSA (siehe B.13) ein zahlentheoretisches Problem um den Signierschlüssel zu schützen. Es ist mit heute bekannten Methoden nicht in vernünftiger Zeit realisierbar, den diskreten Logarithmus einer großen Zahl zu berechnen. Dies macht sich DSS zu Nutze.

Weitere Informationen zu DSS sind unter den *Federal Information Processing Standards Publications*[24] zu finden.

B.3. Diffie-Hellmann-Merkle-Verfahren

Lange Zeit war in der Kryptographie der Austausch von Schlüsseln sehr problematisch, da diese abgehört werden können, wenn sie nicht persönlichen übergeben werden. Die drei Kryptographen Whitfielt Diffie, Martin Hellmann und Ralph Merkle fanden mit Hilfe von Einwegfunktionen eine Möglichkeit, geheime Schlüssel über einen unsicheren Kommunikationskanal auszutauschen.

Zu Veranschaulichung des Verfahrens soll ein einfaches Beispiel dienen. Das Mischen von Farben soll als Einwegfunktion angenommen werden. Es ist sehr einfach, mehrere Farben zusammen zu rühren, aber es ist so gut wie unmöglich, aus der fertigen Farbe die ursprünglichen Farben genau zu bestimmen.

Angenommen, Alice und Bob einigen sich auf eine gemeinsame Farbe wie Grün, die sie vor niemanden verheimlichen müssen. Nun rührt jeder der beide zu der ausgewählten Farbe eine weitere Farbe, die nur jeweils dieser Person bekannt ist. Alice wählt Rosa und Bob verwendet Hellblau. Da die Bestimmung der verwendeten, geheimen Farbe nur auf Grundlage der gemischten, fertigen Farbe unmöglich ist, können beide Ihre verschiedenen Farbtöpfe veröffentlichen und austauschen. Gibt nun Alice ihre geheime Farbe in den von Bob gemischten Farbtopf und

[24] **Vertiefung:** http://www.itl.nist.gov/fipspubs/fip186.htm

	Alice	**Bob**
Schritt 1:	Bob und Alice einigen sich auf ein gemeinsamen α und β, wobei α kleiner als β sein muss. Dieser Austausch kann über eine unsichere Verbindung geschehen. Beispiel: $\alpha = 5$ und $\beta = 7$.	
Schritt 2:	Beide überlegen sich jeweils ein geheimes x (Alice) und y (Bob), das sie nicht weitergeben dürfen und berechnen die Werte für $\alpha^x \ (mod\ \beta)$ beziehungsweise $\alpha^y \ (mod\ \beta)$. Beispiel:	
	$x = 3$	$y = 8$
	$a = \alpha^x \ (mod\ \beta)$	$b = \alpha^y \ (mod\ \beta)$
	$a = 5^3 \ (mod\ 7) = 125 \ (mod\ 7)$	$b = 5^8 \ (mod\ 7) = 390625 \ (mod\ 7)$
	$\mathbf{a = 6}$	$\mathbf{b = 4}$
Schritt 3:	Beide tauschen Ihre Ergebnisse aus, wobei Alice das Ergebnis a und Bob das Ergebnis b übermittelt. Dieser Austausch kann über einen unsicheren Kommunikationsweg stattfinden.	
Schritt 4:	Beide berechnen folgende Gleichung und das ermittelte, an beiden Stellen identische Ergebnis dient als Schlüssel	
	$k = b^x (mod\ \beta)$	$k = a^y \ (mod\ \beta)$
	$k = 4^3 \ (mod\ 7) = 64 \ (mod\ 7)$	$k = 6^8 \ (mod\ 7) = 1679616 \ (mod\ 7)$
	$\mathbf{k = 1}$	$\mathbf{k = 1}$

Tabelle 24: Schlüsselaustausch nach Diffie-Hellmann-Merkle

umgekehrt, erhalten beide genau den gleichen Farbton. Aus den ausgetauschten Informationen konnte ein Angreifer aber nicht auf den fertigen Farbton schließen.

Da mit diesem Gedankenbeispiel keine Nachrichten verschlüsselt werden können, muss eine mathematische, nicht umkehrbare Funktion gefunden werden. Hierfür eignet sich die modulo-Rechnung. Beispielsweise kann bei einer Addition von Minuten eine volle Stunde überschritten werden. Wenn die Stundenanzahl nicht berücksichtigt wird, kann bei dem Addieren von einer bekannten mit einer unbekannten Minutenanzahl mit Hilfe des Ergebnisses die unbekannte Zahl nicht genau bestimmt werden.

Mit Hilfe der modulo-Rechnung erfolgt der Schlüsselaustausch wie in Tabelle 24 beschrieben. Wenn hinreichend große Zahlen gewählt werden, kann auch durch einen „Brute-Force“-Angriff der geheime Schlüssel (k), der in der Tabelle 24 „1“ lautet, nur mit einem sehr hohen Zeitbedarf ermittelt werden (Quelle: [20]).

B.4. Flooding

Flooding (Fluten) ist eine weit verbreitete Angriffstechnik gegen die Verfügbarkeit des Opfers. Im Gegensatz zu anderen Methoden wird hier nicht direkt das Opfer sondern eher die Anbindung des Opfers an das Netz angegriffen. *Flooding* wird der Gruppe der *Denial of Service*-Angriffe zugeordnet.

Ziel des *Flooding* ist es, die Anbindung des Opfers zu seinem Netz soweit zu überlasten, dass ein Abbruch der Verbindung nicht mehr verhindert werden kann.

Am verbreitetsten ist das Fluten mit Datagrammen (*Packetfloods*), die einer tieferen Schicht des TCP/IP-Protokollstacks zugeordnet sind. Das Fluten mit Textmitteilungen ist hauptsächlich ein Problem von *Instant Messaging*- und Konferenzsystemen. Da sich diese Diplomarbeit primär mit diesen Systemen befasst, stehen daher *Textfloods* im Vordergrund.

Für beide Ansätze sollte im allgemeinen die Anbindung des Angreifers an das Netz eine größer Bandbreite besitzen als die des Opfers, damit dieses mehr Datagramme empfängt als es verarbeiten kann. Ist dies nicht möglich, besteht für den Angreifer die Möglichkeit, mit einem *Distributed Denial of Service*-Angriff zum Ziel zu gelangen.

Packetfloods

Diese Art des Angriffs ist sehr effizient und ein weit verbreiteter Ansatz für einen *Denial of Service*-Angriff. Im Rahmen dieser Diplomarbeit spielen *Packetfloods* nur eine Rolle, um einen *Net Split* in einem IRC-Netz zu verursachen.

Packetfloods greifen in ihrer Umsetzung tiefere Schichten des TCP/IP-Protokollstacks an. Da dieses Problem nicht kommunikationssystemsspezifisch ist, wird hierauf nur kurz eingegangen.

Eine weit verbreitete Möglichkeit eines *Packetfloods* ist das **SYN-Flooding**. Ein TCP-Verbindungsaufbau besteht aus einem *SYN*-, *SYN/ACK* und *ACK*-Datagramm. Verwendet der Angreifer nun eine gefälschte, inaktive Absenderadresse für das *SYN*-Datagramm und schickt dieses zu dem Opfer, sendet dieses ein *SYN/ACK*-Datagramm zu der gefälschten Adresse. Nun wartet das Opfer auf das *ACK*-Datagramm. Während des Wartens werden auf dem Rechner des Opfers Ressourcen für diese „halboffene“ Verbindung reserviert.

Das Ziel des *SYN-Floodings* ist es, das Opfer mit so vielen gefälschten Verbindungsanfrage zu belegen, dass es keine Ressourcen mehr für echte Anfragen zur Verfügung hat und diese ablehnt. Bietet das Betriebssystem oder ein möglicher Paketfilter keinen Schutz vor diese Art des Angriffs, kann das Opfer seinen Dienst praktisch nicht mehr anbieten.

Textfloods

Textfloods werden in der Praxis des öfteren verwendet, um im IRC die Verbindung einer anderen Klienten- zu einer Serverinstanz zu unterbrechen. Dies wird von einem Angreifer angestrebt, wenn dieser den eindeutigen *Nickname* übernehmen will (Siehe auch *Nickname War*, Abschnitt B.7). Neben anderen Klienten können aber auch Channel das Ziel eines *Textfloods* werden.

Die Durchführung dieses Angriffs ist sehr simpel und erfordert wenig Hintergrundwissen. Eine riesige Menge von mehr oder weniger sinnvollen Textnachrichten wird mit Hilfe des Kommunikationsprotokolls an das Opfer gesendet. Wenn dieses die Informationsflut nicht mehr verarbeiten kann, bricht seine Verbindung ab. Ein Bot oder eine andere Instanz hilft dabei dem Angreifer, die Textnachrichten zu generieren und zu verschicken.

Neben IRC sind natürlich alle Kommunikationssysteme von dieser Gefahr betroffen. Jedoch ist *Instant Messaging* diesbezüglich weniger verletzlich, da der Klient sowieso nicht ständig mit dem Server verbunden sein muss. Hält der Angriff nur eine kurze Zeitspanne an, kann der Klient die Nachrichten langsam herunterladen. Zusätzlich steht es dem Opfer frei, alle Nachrichten von dem Angreifer zu ignorieren oder zu verwerfen.

Theoretisch kann *Flooding* im Silc-Netz ebenfalls zu Ausfällen führen. Aber die Motivation, den *Nickname* des Opfers oder einen Channel zu übernehmen, existiert durch das Systemdesign von Silc gar nicht.

Den effizientesten Schutz vor *Flooding* bieten Systeme, die bei dem Empfang vieler Informationen innerhalb einer kurzen Zeit von einem Absender diese einfach ignorieren. Dies kann direkt auf dem Server implementiert werden, damit die Informationen gar nicht erst an den Klienten übertragen werden. Ein weiterer Ansatz für den Schutz eines Channels vor *Flooding* ist ein Bot, der die Mitglieder, die dem Channel innerhalb einer vordefinierten Zeitspanne zu viele Nachrichten schicken, aus dem Channel kickt.

B.5. Hashed Message Authentication Code (HMAC)

Das Problem bei der Verwendung von normalen Prüfsummen wie *md5* ist, dass die aktuelle Prüfsumme nur mit der Vergleichssumme verglichen werden kann. In der Praxis wird die Prüfsumme der Information von dem Absender erstellt und an einer vorher definierten Position angefügt. Der Empfänger extrahiert die Prüfsumme, die der Versender erstellt hat, und vergleicht diese mit der Prüfsumme der übertragenen Information.

Dies ist sinnvoll, da bei einem Fehler in der Übertragung die Prüfsumme, die in dem Datagramm eingetragen wurde und die Prüfsumme, die über das Datagramm gebildet wurde, nicht übereinstimmen.

Daher eignen sich Prüfsummen sehr gut, um technisch Probleme festzustellen. Anders sieht es aber aus, wenn ein Angreifer Veränderungen an den Informationen vornimmt. Da die Nachricht

verändert wird, ändert sich somit auch die Prüfsumme. Es ist aber für einen Angreifer sehr leicht die Prüfsumme neu zu bilden und in dem hierfür reservierten Feld einzutragen.

Eine Lösung dieses Problems ist es, die Prüfsumme nicht nur abhängig vom Inhalt sondern auch abhängig vom Wissen des Generierers zu erzeugen. Hierfür haben sich zwei Lösungen durchgesetzt.

Die bekanntere Lösung sind so genannte Public-Key- oder asymmetrische Verschlüsselungsverfahren. Dabei wird eine Prüfsumme in Abhängigkeit eines privaten Schlüssels erstellt, den nur der Generierer kennt. Mit Hilfe des öffentlichen Schlüssels kann der Empfänger bestätigen, dass die übermittelte Nachricht mit dem privaten Schlüssel des Senders erstellt wurde.

Asymmetrische Verfahren sind nicht immer die beste Wahl, da sie hohe Rechenleistung erfordern und aufwändig sind. Einfacher ist es dagegen, ein *Hashed Message Authentication Code* (HMAC) zu erstellen. Das dazugehörige Verfahren unterscheidet sich von einer gewöhnlichen Prüfsummenbildung durch die Einbeziehung eines Geheimnisses. Alle sicheren Prüfsummenverfahren besitzen die Eigenschaft, dass aus gleichen Informationen immer die gleiche Prüfsumme erstellt wird. Dagegen kann auf Grund der Prüfsumme nicht auf den Inhalt der ursprünglichen Nachricht geschlossen werden.

B.6. Initialisierungsvektor

Viele Verschlüsselungsverfahren berücksichtigen vorhergehende Teile des Chiffretextblocks bei der Verschlüsselung eines Klartextblocks (Beispiel: der *cipher block chaining* (CBC)-Modus verknüpft XOR mit dem vorigen Chiffretextblock).

Für den ersten Klartextblock ist dies aber problematisch, da kein Chiffretextblock vorausgeht. Daher wird stattdessen der Initialisierungsvektor (IV) verwendet. Besitzen zwei identische zu verschlüsselnde Nachrichten, die mit dem identischen Schlüssel geschützt werden sollen, verschiedene IVs, unterscheiden sich die gewonnenen Chiffrate. Analog hierzu besitzen zwei identische Klartexte mit identischen IV ein identisches Chiffrat.

Der Initialisierungsvektor kann im Gegensatz zu dem Schlüssel veröffentlicht werden. In der Regel ist diese *Dummy*-Zeichenkette genauso lang wie der Schlüssel.

B.7. Nickname War

Möchte ein Benutzer mit einem anderen über IRC direkt kommunizieren, muss der Sender eine Verbindung zu dem gewünschten Gesprächspartner aufbauen. Hierfür benötigt dieser den eindeutigen Identifikator des Partners.

Dies ist relativ leicht zu implementieren, da es dem Server die Arbeit erleichtert, Nachrichten genau an diesen Benutzer weiterzuleiten. Es wird auch nur eine Datenstruktur benötigt, die dem *Nickname* einen Klienten im Netz zuordnet.

Die Verwendung eines eindeutigen *Nickname* hat auch einen gravierenden Nachteil: Viele Benutzer wünschen sich aussagekräftige *Nicknames*. Da *Nicknames* wie Vor-, Nach- oder Spitznamen, sehr beliebt sind, treten sehr oft Interessenkonflikte zwischen den Benutzern auf.

Wenn sich nun mehrere Benutzer mit einem begehrten *Nickname* im IRC anmelden wollen, entsteht ein Problem: Nur der erste bekommt den gewünschten *Nickname*. Ein Großteil der zu kurz gekommenen Benutzer wird dann einen anderen *Nickname* auswählen. Aber ein kleiner Teil

wird sich damit nicht zu Frieden geben und versuchen durch *Nickname Wars* an den gewünschten *Nickname* zu gelangen.

Der erste Benutzer, der sich mit dem gewünschten *Nickname* anmeldet, bekommt diesen zugewiesen. Wenn berücksichtigt wird, dass der gewünschte, aber schon belegte *Nickname* nur solange nicht zur Verfügung steht, wie der Klient, der ihn besitzt, mit dem IRC-Netz verbunden ist, wird schnell ein Ansatz für die Übernahme des *Nickname* deutlich. Gelingt es dem Angreifer, die Verbindung zwischen Klient und Server zu unterbrechen, ist der Benutzer nicht mehr angemeldet und verliert seinen Anspruch auf den *Nickname*.

Dies kann mit *Denial of Service*-Angriffen wie *Flooding* umgesetzt werden (siehe Kapitel B.4).

B.8. Passphrase

Eine *Passphrase* dient ähnlich wie ein Passwort der Authentisierung. Beispielsweise schützt eine *Passphrase* vor der unautorisierten Nutzung eines privaten Schlüssels. Diese *Passphrase* darf nur dem berechtigten Benutzer bekannt sein.

Im Gegensatz zu einem Passwort besteht eine *Passphrase* aus mehreren Wörtern, die unter Verwendung eines Leerzeiches voneinander getrennt werden können. In der Regel sollten *Passphrasen* auch länger als Passwörter gewählt werden.

Passphrases sollten genauso wie Passwörter regelmäßig gewechselt werden, um den höchsten Schutz zu gewährleisten.

B.9. Perfect Forward Secrecy (PFS)

Wenn das PFS-Verfahren aktiviert ist, beeinflusst es die Erstellung von neuen Schlüsseln. Prinzipiell werden kryptographische Charakteristiken von geheimen Werten abgeleitet. PFS verhindert, dass ein neuer Schlüssel, der einen alten ersetzen soll, von dem vorigen abgeleitet wird.

Dies ist sinnvoll, weil aus einen erfolgreich angegriffenen Schlüssel der neue Schlüssel nicht abgeleitet werden kann.

B.10. Phishing

Phishing ist eine der vielen Möglichkeiten eines *Social Engineerers*, um ein Opfer ohne technische Maßnahmen anzugreifen. Der Begriff „*Phishing*“ besitzt vermutlich seinen Ursprung in ***p**assword **h**arvesting **fishing***.

In der Regel fälscht der Angreifer beim *Phishing* E-Mails, in denen er sich als eine Bank oder andere Institution ausgibt. In dieser E-Mail wird das Opfer zu einem Besuch einer Webseite verleitet, die der der Institution sehr ähnlich sieht. Bei einem erfolgreichen Angriff trägt das Opfer persönliche Informationen in Formulare auf der Webseite ein.

Ein Beispiel für *Phishing* ist eine gefälschte E-Mail einer Bank, bei der das Opfer seine Kontonummer und seine PIN für die Änderung von Kontodaten in Formulare auf einer Website eingeben muss. Der Angreifer, der diese Website pflegt, kann nun die eingegebenen Informationen auswerten.

B.11. Pretty Good Privacy (PGP)

Pretty Good Privacy ist eine Implementierung einer asymmetrischen Verschlüsselung. Bei einer asymmetrischen Verschlüsselung gibt es das Problem des Schlüsselaustauschs nicht, da die Ver- und Entschlüsselung mit verschiedenen Schlüsseln durchgeführt wird. Mit dem öffentlichen Schlüssel kann so eine Nachricht geschützt werden, die nur der Besitzer des entsprechenden privaten Schlüssels lesen kann.

Die asymmetrische Verschlüsselung kann auch zur Signierung genutzt werden: durch das Unterschreiben einer Nachricht mit dem privaten Schlüssel kann der Verfasser der Nachricht mit Hilfe des öffentlichen Schlüssels eindeutig bestimmt werden.

PGP ermöglicht den effizienten Einsatz der Nachrichtenverschlüsselung und -signatur indem es dem Benutzer hilfreiche Routinen zur Verfügung stellt. Somit kann es auch sehr leicht von Benutzern eingesetzt werden, die nur geringes Hintergrundwissen zur Kryptographie besitzen.

Neben dem klassischen, kommerziellen PGP gibt es *OpenSource*-Lösungen wie **GnuPG** oder Softwarebibliotheken, die eine unkomplizierte Integration in eigene Programme ermöglichen.

B.12. Public Key Cryptographic Standard (PKCS)

PKCS ist eine Spezifikation zu einer Reihe von asymmetrische Verschlüsselungsverfahren. Im Jahre 1991 entwickelten die *RSA Laboratories*[25] diesen Standard, um die Verteilung der Public Key Verschlüsselung zu beschleunigen.

Die PKCS-Dokumente bestehen aus insgesamt 15 einzelnen Dokumenten, die im Folgenden aufgelistet sind. Die Nummern zwei und vier wurden zu dem *RSA Cryptography Standard* zusammengefasst und entfernt (Quelle: [19]).

PKCS #1:	RSA Cryptography Standard
PKCS #3:	Diffie-Hellman Key Agreement Standard
PKCS #5:	Password-Based Cryptography Standard
PKCS #6:	Extended-Certificate Syntax Standard
PKCS #7:	Cryptographic Message Syntax Standard
PKCS #8:	Private-Key Information Syntax Standard
PKCS #9:	Selected Attribute Types
PKCS #10:	Certification Request Syntax Standard
PKCS #11:	Cryptographic Token Interface Standard
PKCS #12:	Personal Information Exchange Syntax Standard
PKCS #13:	Elliptic Curve Cryptography Standard
PKCS #15:	Cryptographic Token Information Format Standard

B.13. RSA

Bei RSA (benannt nach seinen Erfindern Ron Rivest, Adi Shamir und Leonard Adleman) handelt es sich um einen *Public Key*-Algorithmus, der sowohl zum Verschlüsseln als auch zum Signieren verwendet werden kann.

[25] **Vertiefung:** http://www.rsasecurity.com/rsalabs/node.asp?id=2124

Die Sicherheit von RSA beruht auf der Schwierigkeit, große Zahlen zu faktorisieren. Der öffentliche und der private Schlüssel hängen von einem Paar großer Primzahlen ab (100 bis 200 Stellen und mehr). Es wird vermutet, dass die Wiederherstellung des Klartexts aus dem öffentlichen Schlüssel und dem Chiffretext äquivalent zur Faktorisierung des Produkts der beiden Primzahlen ist.

Als Quelle diente „Angewandte Kryptographie“ von Bruce Schneier [21] wo vertiefende Informationen zu finden sind.

Erstellung der Schlüssel

Um den öffentlichen Schlüssel n zu erzeugen, müssen zwei zufällige Primzahlen p und q gesucht werden. Diese beiden Zahlen sollten ungefähr gleich lang und sehr groß gewählt werden. Der öffentliche Schlüssel n wird mittels

$$n \quad = \quad pq$$

errechnet. Wird für $\mathbf{p = 47}$ und für $\mathbf{q = 71}$ gewählt, ergibt $\mathbf{n = 3337}$. Nun wird der zufällige Chiffrierschlüssel e so gewählt, dass e und $(p-1)(q-1)$ relativ prim zueinander sind. Aus dem zufälligen Chiffrierschlüssel e kann nun der Dechiffrierschlüssel errechnet werden:

$$d \quad = \quad e^{-1} \; mod \; ((p-1)(q-1))$$

Wird für $\mathbf{e = 79}$ gewählt, kann der Dechiffrierschlüssel mit $\mathbf{d = 79^{-1} \; mod (46 * 70) = 1019}$ errechnet werden. Nun können n (hier: 3337) und e (hier: 79) veröffentlicht werden.

Verschlüsselung einer Nachricht

Nachdem dem Sender die öffentlichen Schlüssel n und e vorliegen, kann die Nachricht m mittels

$$c \quad = \quad m^e \; mod \; n$$

verschlüsselt werden. Das Resultat ist das Chiffrat c. Soll die Nachricht $\mathbf{m = 688}$ verschlüsselt werden, kann das Chiffrat mittels $\mathbf{688^{79} \; mod \; 3337 = 1579}$ erzeugt werden.

Entschlüsselung eines Chiffrats

Wurde dem Besitzer des privaten Schlüssels das Chiffrat $c = 1579$ übermittelt, kann er dieses nun mit

$$m \quad = \quad c^d \; mod \; n$$

entschlüsseln. Für das vorher gewählte Beispiel kann nun die ursprüngliche Nachricht mittels $\mathbf{m = 1570^{1019} \; mod \; 3337 = 688}$ zurückgewonnen werden.

B.14. Schlüssel

Ein Schlüssel in der Kryptographie bezeichnet das Geheimnis, durch das einer autorisierten Person der Zugriff auf die geschützte Information gestattet wird.

Schon Kerckhoff von Nieuwenhof, niederländischer Philologe (1835 bis 1903), postulierte in *Kerckhoffs Prinzip*, dass die Sicherheit eines kryptographischen Systems ausschließlich von der Geheimhaltung des verwendeten Schlüssels abhängen darf. Dagegen bietet die Geheimhaltung des verwendeten Verschlüsselungsverfahrens nur einen sehr geringen Schutz.

Bei einer symmetrischen Verschlüsselung haben sowohl der Sender als auch der Empfänger Zugang zu dem Schlüssel, der für die Ver- und Entschlüsselung benötigt wird. Hier tritt das Problem des Schlüsselaustauschs auf.

Bei einem asymmetrischen Verschlüsselungsverfahren existieren zwei Arten von Schlüsseln: ein privater Schlüssel und ein öffentlicher, der aus dem privaten Schlüssel generiert wurde. Die Nachrichten, die mit einem öffentlichen Schlüssel verschlüsselt wurden, können nur mit dem privaten Schlüssel entschlüsselt werden.

Zusätzlich kann eine Signatur, die mit einem privaten Schlüssel erstellt wurde, mit dem öffentlichen Schlüssel überprüft werden. Dies kann genutzt werden, um zu prüfen, ob die übermittelte Nachricht wirklich von dem vermuteten Absender stammt.

B.15. Secure Sockets Layer (SSL)

Bei *Secure Sockets Layer* (SSL) handelt es sich um ein von *Netscape* entwickeltes Protokoll, das eine geschützte Kommunikation ermöglicht. Prinzipiell werden dabei die Datagramme höherer Protokolle wie HTTP innerhalb einer verschlüsselten SSL-Verbindung übertragen. Dementsprechend baut SSL auf untergeordnete Transportprotokolle wie TCP auf.

Für die Verschlüsselung der Nutzlast werden symmetrische Verschlüsselungsverfahren eingesetzt. Hierfür wird ein Schlüsselaustausch durch SSL angeregt.

Soll eine geschützte Kommunikation zwischen einer Klienten- und Serverinstanz aufgebaut werden, findet mittels des *SSL Handshake Protocols* eine Authentisierung statt.

1. Die Klienteninstanz übermittelt der Serverinstanz eine Verbindungsanfrage.
2. Die Serverinstanz beantwortet diese und übermittelt unter Umständen ein Zertifikat.
3. Die Klienteninstanz versucht, das Zertifikat zu authentisieren - bei Misserfolg wird die Kommunikation abgebrochen. Dieses Zertifikat beinhaltet den öffentlichen Schlüssel des Servers.
4. Nach einer erfolgreichen Authentisierung erstellt die Klienteninstanz das *pre-master secret*. Dieses wird mit dem öffentlichen Schlüssels des Servers verschlüsselt und an diesen übertragen. Zusätzlich erzeugt der Klient hieraus das *master secret*.
5. Die Serverinstanz entschlüsselt das *pre-master secret* mit seinem privaten Schlüssel und erstellt ein *master secret*.
6. Beide erstellen aus dem *master secret* den *session key*, der als symmetrischer Schlüssel ab diesem Zeitpunkt genutzt werden kann.

7. Beide tauschen mit diesem *session key* verschlüsselte Nachrichten aus und signalisieren damit ihre Kommunikationsbereitschaft.

(Quelle: [19])

B.16. Sniffen

Sniffen ist ein Begriff für das Protokollieren von Netzverkehr. Dieser kann dann für eine weitere Verwendung ausgewertet werden. Werden kritische Informationen wie Passwörter im Klartext übertragen, kann der Angreifer diese aus den empfangenen Informationen extrahieren.

Für das klassische Sniffen wird die lokale Netzkarte in den so genannten *Promiscous Mode* geschaltet, in dem alle empfangenen Datagramme mitgelesen werden. Befindet sich die Netzkarte nicht darin, werden nur die an die Karte adressierten Datagramme an das System weitergereicht.

Diese Art des Sniffens ist nur sinnvoll, wenn die Datagramme von dem lokale System empfangen werden können. Dies ist der Fall, wenn es sich um ein Netz mit einem Hub oder bei dem System um eine zentrale Stelle wie einen Gateway oder Server handelt. Für einen Angreifer ist es so gut wie unmöglichen, in Netzen zu sniffen, auf die er keinen direkten Zugriff hat.

Das Sniffen in geswitchten Netzen ist nur mit Aufwand möglich. In diesen werden die Datagramme von dem Switch direkt an die Zieladressen verteilt. Somit sollten die Datagramme, die nicht an das System des Angreifers adressiert sind, diesem auch nicht zugestellt werden. In diesem Fall kann der Angreifer aber durch Techniken wie *ARP-Spoofing* (siehe Anhang B.17.2) an sein Ziel gelangen.

Bei vielen Switches kann ein Port für Protokollierungsszwecke verwendet werden. Dieser Port, an den von jedem Datagramm eine Kopie geschickt wird, erleichtert dem Administrator die Suche nach Fehlern. Hat ein Angreifer direkten Zugriff auf den Switch, kann er diesen Port für seine Angriffe missbrauchen.

B.17. Spoofing

Als *Spoofing* wird das Fälschen der Identität zur Täuschung eines anderen Systems bezeichnet. In IP-Netzen wird *Spoofing* von einem Angreifer durchgeführt, um ein bestimmtes System als ein anderes System auszugeben. Dies kann auch für eine *Man-In-The-Middle-Attack* ausgenutzt werden.

Auf vielen Schichten des ISO/OSI-Referenzmodells lassen sich *Spoofing*-Angriffe realisieren. Im Folgenden werden einige *Spoofing*-Techniken vorgestellt.

B.17.1. MAC-Spoofing

Für die Adressierung von *Ethernetdatagrammen* innerhalb der Sicherungsschicht des ISO/OSI-Referenzmodells werden *Media Access Control* (MAC)-Adressen benötigt. Jedes System, das an ein Netz angeschlossen wird, sollte eine eindeutige MAC-Adresse besitzen, die von dem Hersteller fest mit der Netzkomponente verknüpft wird.

MAC-Adressen können auch zur Authentisierung verwendet werden - Beispiele hierfür sind *Switches* und WLAN-*Access Points*, die nur Datagramme bearbeiten, die von Systemen mit vorher definierten MAC-Adressen stammen.

Obwohl die MAC-Adresse fest in die Netzkomponente „eingebrannt“ ist, kann sie sehr leicht verändert werden. Der Hersteller bietet unter Umständen Schutz vor einer dauerhafter Änderung der MAC-Adresse, indem die Speicherstelle, die die MAC-Adresse beinhaltet, mit einem Schreibschutz versehen ist - aber eine temporäre Änderung kann nicht verhindert werden.

Viele gängige Betriebssysteme liefern für die temporäre Änderung der MAC-Adresse die benötigten Werkzeuge mit. Somit kann ein Angreifer, der bei dem Großteil der verbreiteten Betriebssysteme Administratorenrechte hierfür benötigt, sehr leicht die MAC-Adresse bis zum nächsten Neustart ändern.

B.17.2. ARP-Spoofing

Bei der Adressierung von Datagrammen spielt die MAC-Adresse eine wichtige Rolle. Da aber in der Regel eine IP-Adresse als Zieladresse angegeben wird, müssen die Rechner in der Lage sein, den Rechnern in ihrer Domain eine MAC-Adresse zuzuordnen.

Zur Zuordnung von IP- zur MAC-Adressen dient das *Address Resolution Protocol* (ARP). Möchte ein System eine Verbindung zu einem anderen System aufbauen, von dem es nur die IP-Adresse kennt, sendet es einen ARP-*Request*. Dieser *Request* ist an die ARP-*Broadcast*-Adresse (*ff:ff:ff:ff:ff:ff*) adressiert und beinhaltet neben der gesuchten IP-Adresse die MAC-Adresse des Senders.

Empfängt der Rechner mit der gesuchten IP-Adresse diesen *Request*, sendet er ein ARP-*Reply* an den Rechner mit der empfangenen MAC-Adresse. Aus dieser Antwort kann der ursprüngliche Sender nun der IP-Adresse eine MAC-Adresse zuordnen.

In der Regel besitzt jedes ARP-fähige Gerät einen Speicher (*Cache*), in dem die MAC-/IP-Adresspaare abgelegt werden können. Aus diesem Grund muss nicht für jedes Datagramm ein neuer ARP-*Request* versendet werden.

Empfängt ein Angreifer, der einen Angriff durch *ARP-Spoofing* ausführen will, einen *Request*, so kann er behaupten, er hätte die gesuchte IP-Adresse (was aber nicht der Fall ist) indem er ein ARP-*Reply* versendet. Empfängt der ursprüngliche Sender nun diesen ARP-*Reply* vor dem des wahren Empfängers, werden alle Datagramme, die an die IP-Adresse des Empfängers gesendet werden sollen, an die MAC-Adresse des Angreifers übermittelt.

Dieser Angriff ist sehr effizient, da der ursprüngliche Sender nur die erste Antwort auf den ARP-*Request* berücksichtigt und die folgenden verwirft. Prinzipiell kann durch folgende Varianten, die sich durch die gesendeten Adressen unterscheiden, ein Angriff durch *ARP Spoofing* erfolgen:

- Der Angreifer versendet *ARP-Replys*, die seine eigene MAC-Adresse beinhalten. Alle beteiligten Systeme ordnen nun der IP-Adresse des anzugreifenden Systems die MAC-Adresse des Angreifers zu. Die Datagramme werden von dem Switch direkt an den Angreifer übermittelt und dieser leitet sie zum korrekten Empfänger um.

- Als zweite Variante überflutet der Angreifer das Netz mit MAC-Adressen, die sonst nicht in diesem Netz vorkommen. Die Tabelle des Switchs kann nicht aktuell gehalten werden und somit sendet der Switch alle Datagramme an alle Ports. Alle Systeme bis auf das des Angreifers verwerfen diese Datagramme und der Angreifer kann diese bearbeiten und an den tatsächlichen Empfänger weiterleiten.

B.17.3. IP-Spoofing

Die Zustellung von Datagrammen über mehrere Subnetze geschieht anhand der *Internet Protocol* (IP)-Adresse. IP ist der Vermittlungsschicht des ISO/OSI-Referenzmodells zuzuordnen.

IP-Adressen müssen eindeutig sein. Eine Ausnahme bilden die privaten IP-Adressbereiche. Daher nutzen Paketfilter und netzfähige Applikationen die IP-Adresse für eine Authentisierung und Zuordnung von Privilegien.

IP-Adressen besitzen einen *Hostteil*, der den Rechner identifiziert und einen *Netzteil*, der einem Subnetz entspricht. Alle Rechner, die an einem Subnetz angeschlossen sind, müssen den gleichen Netzteil aufweisen. Da bei einem Umzug in ein anderes Netz die IP-Adresse diesem angepasst werden muss, stellen viele Betriebssysteme dazu Tools bereit.

B.17.4. DNS-Spoofing

Die Adressierung von Datagrammen im Internet erfolgt nach IP-Adressen, die in der Regel als 32-Bit-Zahl dargestellt werden. Da für den menschlichen Betrachter diese Notation nicht sehr gut zu merken und verarbeiten ist, kann der IP-Adresse ein Rechnername zugeordnet werden.

In der Regel möchte ein Benutzer eine Verbindung zu einem Rechner aufbauen, dessen Namen er kennt. Nachdem er diesen Namen seiner Klienteninstanz übergeben hat, versucht diese, den angegebenen Rechnernamen in eine IP-Adresse zu transformieren. Anhand dieser IP-Adresse wird dann die eigentliche Verbindung aufgebaut.

Diese Transformationen übernehmen *Domain Name Service* (DNS)-Server. Stellt der Klient eine Anfrage mit einem Rechnernamen an diesen Server, antwortet dieser mit der gesuchten IP-Adresse. Auch *Reverse*-DNS-Anfragen sind möglich - in diesem Fall beinhaltet die Anfrage die IP-Adresse und die Antwort den Rechnernamen.

Für *DNS-Spoofing* sind folgende DNS-Grundlagen interessant:

- Es ist unmöglich, dass ein einzelner DNS-Server alle Rechnernamen und IP-Adressen des Internets führen kann. Daher kennt er in der Regel nur IP-Adressen und Rechnernamen von Systemen in seiner Nähe - bei Anfragen nach unbekannten Systemen stellt er eine Anfrage an das hierfür zuständige System.

- Da sich Anfragen jederzeit wiederholen können, werden die Ergebnisse eine Zeit lang gespeichert. So muss für eine Auflösung von Rechnernamen nicht für jede Anfrage eine Verbindung zu der entsprechenden Serverinstanz aufgebaut werden.

- Um den Protokolloverhead zu minimieren, werden bei einer Anfrage nicht nur die gewünschte IP-Adresse/Rechnernamen-Kombination übertragen, sondern auch weitere oft angefragte Kombinationen.

Diese Eigenschaften lassen sich für den folgenden, in Lehrbüchern sehr verbreiteten DNS-*Spoof* ausnutzen:

- Ein Klient möchte die Webseite des Unternehmens „A“ aufrufen und stellt die hierfür benötigte Anfrage an seinen zuständigen DNS-Server.

- Der DNS-Server kennt die IP-Adresse der Webseite des Unternehmens „A“ nicht und stellt eine Anfrage an den für das Unternehmen „A“ zuständigen DNS-Server.
- Der DNS-Server des Unternehmens übermittelt neben der angeforderten Information auch die DNS-Daten des Konkurrenzunternehmen „B“. Dabei gibt er allerdings nicht dessen wahre Adresse, sondern eine gefälschte zurück.
- Diese beiden Einträge werden auf dem DNS-Server zwischengespeichert. Der Klient erhält die angeforderten Daten.
- Möchte nun der Klient die Internetpräsenz von Unternehmen „B“ anschauen, stellt er wieder eine Anfrage an seinen DNS-Server nach der IP-Adresse. Da der DNS-Server diesem Rechnernamen eine Adresse zuordnen kann, übermittelt er die vorher übertragene Adresse dem Klienten. Da diese Adresse zu dem Server des Unternehmens „A“ zeigt, ist es dem Klienten unmöglich, die Webseite des Unternehmens „B“ aufzurufen.

B.17.5. URL-Spoofing

Besonders die Adressierung eines der verbreitetsten Protokolle im Internet, HTTP, bietet einige Ansätze für das Fälschen des gewünschten Rechnernamen. Das Ziel des Angreifers ist hier ebenfalls, dass der Benutzer auf die Webseite des Angreifers gelangt, obwohl er eine andere betrachten möchte.

Soll das Opfer durch *Phishing* (siehe Anhang B.10) auf die Seite des Angreifers gelockt werden, sollte der Angreifer dafür sorgen, dass die angegebene URL (*Uniform Resource Locator*) des Angreifers der Originalen sehr ähnlich sieht. Neben einer URL, die bis auf einzelne Zeichen mit der Originalen identisch ist (Beispiel: `http://www.google.de` und `http://www.goole.de`) existieren weitere zahlreiche Möglichkeiten.

Beispielsweise sieht HTTP eine optionale Angabe des Benutzernamens vor, der in der URL mit einem „@“ beendet wird. Versucht ein unachtsamer Benutzer die Seite `http://www.paypal.com@192.168.0.1` zu betrachten, so erhält er seine Antworten nicht wie vermutet von Firma *paypal*, sondern wird als Benutzer *www.paypal.com* auf dem Server *192.168.0.1* angemeldet.

Auch die Möglichkeit, Umlaute in einer URL zu kodieren, bietet einem Angreifer zahlreiche Angriffsmöglichkeiten wie *International Domain Name (IDN)-Spoofing*. Neben den in Deutschland gängigen Umlauten wie „ü“, „ä“, und „ö“ und lateinischen Schriftzeichen können auch Kyrillische verwendet werden. Da das lateinische „a“ dem kyrillischen „a“ (*#1072;*) sehr ähnlich sieht, ist in der Adresszeile des Browsers kein Unterschied zwischen den URLs `http://www.p\аypal.com` und `http://www.paypal.com` zu erkennen. Diese Beispielrechnernamen sind Standardbeispiele in zahlreichen Publikationen für diese Schwachstelle.

B.18. TCP Hijacking

Wie in der Luftfahrt ist das Ziel von *Hijacking* die (gewaltsame) Übernahme von Ressourcen. Im Zusammenhang mit der IT-Sicherheit versucht ein Angreifer, in einen bestehenden TCP-Datagrammstrom Daten einzufügen.

Der Angreifer muss hierfür direkten Zugriff auf die Verbindung besitzen. Besonders einfach ist die Durchführung eines solchen Angriffs auf dem System des Senders, Empfängers oder eines dazwischenliegenden Routers. Ist das System aber nicht direkt an der Kommunikation beteiligt, kann durch *Spoofing* (siehe Anhang B.17) der Datagrammstrom über das System des Angreifers geleitet werden.

Prinzipiell empfängt der Angreifer wie bei vielen Angriffen die Daten und leitet sie am Ende zu dem gewünschten Empfänger weiter. Bei einem Angriff durch *Hijacking* wird in diese Kette ein weiteres Glied eingefügt: die Manipulation.

Ein blindes Manipulieren des Datenstroms führt gerade in Verbindung mit TCP nicht zum gewünschten Erfolg: Sehr oft müssen dynamische Größen wie Sequenznummern geändert werden. Daher sollten die kritischen Informationen erst eine Weile abgehört werden, um diese Einträge vorherzusagen.

Eine Applikation, mit der sehr einfach eine Verbindung übernommen werden kann, ist *hunt*. Ein Beispiel für einen Angriff mit *hunt* wird im Kapitel 6.1 gezeigt.

B.19. Transport Layer Security (TLS)

Transport Layer Security (TLS) ist eine Weiterentwicklung vom *Secure Sockets Layer* (SSL) (siehe B.15), die die Integrität und Vertraulichkeit eines Datenflusses garantieren soll.

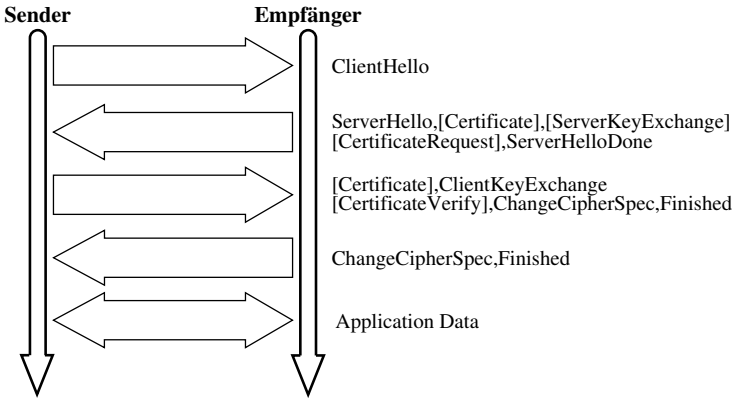

Abbildung 40: Protokollablauf von TLS

TLS arbeitet mit einer Vielzahl von Applikationsprotokollen zusammen - HTTP, SMTP und XMPP sind nur einige Beispiele. Dadurch können Protokolle, die keine eigenen Schutzmechanismen bereitstellen, abgesichert werden.

Dabei kann für den verschlüsselten Informationsstrom eine eigene Verbindung aufgebaut oder dieser in die bestehende ungeschützte Verbindung integriert werden. Im letzteren Fall wird TLS durch das Schlüsselwort *STARTTLS* ausgelöst.

Prinzipiell besteht TLS aus zwei Protokollschichten: dem **TLS Record Protocol** und dem **TLS Handshake Protocol**. Über das *TLS Handshake Protocol* werden zunächst die unterstützten Algorithmen ausgehandelt. Im nächsten Schritt authentisieren sich die Kommunikationspartner gegenseitig mit Hilfe von Public-Key-Verschlüsselung und X.509 Zertifikaten. Im letzten Schritt des *TLS Handshake Protocols* wird ein Sitzungsschlüssel ausgehandelt.

Nach dem Aushandeln des Sitzungsschlüssels kann das *TLS Record Protocol* die Nachrichten verschlüsseln und kryptographische Prüfsummen bilden. In Abbildung 40 wird der Kommunikationsablauf schematisch skizziert (Quelle: [19]).

C. Entwickelte Software

Um die Funktionsweise der vorgestellten Protokolle zu verdeutlichen, wurde die Analyse mit eigenen Klienten durchgeführt. Die meisten verbreiteten Klienten versenden in der Regel mehr Datagramme als notwendig, und erschweren damit die Untersuchung. Ziel dieser Klienten war es, den Datagrammfluß nur auf die wesentlichen Datagramme zu beschränken.

Diese Klienten bieten nur einen sehr geringen Funktionsumfang. Um die Quelltexte so übersichtlich wie möglich zu halten, wurde auf das Abfangen von fehlerhaften Benutzereingaben verzichtet.

Neben einer Perl-Distribution werden für die Verwendung der Software die entsprechenden Bibliotheken benötigt.

C.1. OSCAR-Klient

```
#!/usr/bin/perl
#siehe http://search.cpan.org/~matthewg/Net-OSCAR-1.907

use Net::OSCAR qw(:standard);
use LWP::UserAgent qw(:standard);
my $host = "login.oscar.aol.com";
my $port = "5190";

if ($#ARGV == 1){
        $screenname = $ARGV[0];
        $password = $ARGV[1]
   } else { die "Kein Benutzernamen und Passwort eingegeben\n"}

print "Erstellen eines Objektes\n";
$oscar = Net::OSCAR->new();

print "Definition von Callbacks, um auf Serverinformationen zu reagieren\n";
#Ausgabe von Fehlern
$oscar->set_callback_error(
  sub {
    my($osc, $connection, $error, $description, $fatal) = @_;
    die "Error $error: $description\n";
    }
);

#Ausgabe empfangener Nachrichten
$oscar->set_callback_im_in(
  sub {
    my ($osc, $sender, $message, $away) = @_;
    print "$sender: $message\n";
    #und gleich antworten
```

```
    $oscar->send_im($sender,"Ack\n");
  }
);

#wird aufgerufen, wenn ein Kontakt sich anmeldet
$oscar->set_callback_buddy_in(
  sub{
    my ($osc, $buddy, $group, $data) = @_;
    print "ONLINE: ";
    print "$buddy / $group / $data\n";
    }
);

#wird aufgerufen, wenn anmeldung abgeschlossen ist
$oscar->set_callback_signon_done(
  sub {
    print "Anmeldung abgeschlossen\n";
    #einfuegen neuer Buddies
    $oscar->add_buddy("Buddies", "imstudie");
    print "done\n";
    }
);

print "Anmelden am OSCAR-Server\n";
$oscar->signon(screenname=>$screenname, password=>$password,
               host=>$host, port=>$port);

print "Warten auf Ereignisse\n";
while(1){
  $oscar->do_one_loop();
}
```

C.1.1. Klienten mit getrennter Authentisierung und Anmeldung

Der folgende Klient ermöglicht nur die Anmeldung am OSCAR-Server - *Instant Messaging*-Funktionalitäten sind nicht implementiert. Dabei ist eine Trennung zwischen der Kontaktaufnahme zum *Authentification Server* und zum *BOS-Server* vorgesehen. Im Detail sendet das Script *oscar-auth.pl* eine Zeichenkette an den *Authentification Server*, der mit dem *Authorization Cookie* antwortet.

Dieses *Cookie* wird über eine TCP-Verbindung an das Script *oscar-bos.pl* gesendet, das wiederum eine Anmeldung an dem BOS-Server vornimmt.

Durch diese Trennung kann die Abhängigkeit des *Authorization Cookie* von der IP-Adresse nachgewiesen werden (siehe Demonstration 6.2).

Beide Scripte erwarten als Eingabeparameter jeweils die Nutzlast der „*New Connection*“-Datagramme, die mit verschieden *Accounts* erzeugt wurden. Diese Informationen können beispielsweise von dem OSCAR-Klienten C.1 abgehört werden.

oscar-auth.pl

```
#!/usr/bin/perl -w
use IO::Socket;

#verbindungsinfos zum Anmelde-System (ueber ein VPN),
my $logonaddr = "192.168.1.192";
my $logonport = "2401";

#UEBERGABE BEIM AUFRUF: ERSNIFFTES NEW-CONNECTION
#also: ./oscar-auth.pl "\x2a\x01\x41\xf2\x00\x84\x00\x00\x00\x01\x00\x01
    ....."
my $authpacket = $ARGV[0];

#VERBINDUNGSAUSBAU ZU LOGIN.OSCAL.AOL.COM
my $login = IO::Socket::INET->new( PeerAddr => "login.oscar.aol.com",
        PeerPort => "5190",         Proto    => "tcp",
        Type     => SOCK_STREAM);
setsockopt($login, &Socket::IPPROTO_TCP, &Socket::TCP_NODELAY, 1);
$login->send($authpacket,0);
my $antwort;
#unwichtiges "new Connection" vom Server
$login->recv(my $antwort1, 1024, 0);
#benoetigten infos
$login->recv(my $antwort2, 1024, 0);

print "Verbindungsaufbau zum Klienten, der sich am BOS anmeldet\n";
my $agent = IO::Socket::INET->new( PeerAddr => $logonaddr,
        PeerPort => $logonport,     Proto    => "tcp",
        Type     => SOCK_STREAM);
$agent->send($antwort2,0);
close($agent);
```

oscar-bos.pl

```
#!/usr/bin/perl -w
use IO::Socket;
#UEBERGABE BEIM AUFRUF: ERSNIFFTES NEW-CONNECTION
#also: ./oscar-bos.pl "\x2a\x01\x41\xf2\x00\x84\x00\x00\x00\x01\x00\x01....."
my $fakelogin = $ARGV[0];

#header für Anmeldung zum BOS. Nur auth-cookie muss angefuegt werden
my $header = "\x2a\x01\x17\x79\x01\x08\x00\x00\x00\x01\x00\x06\x01\x00";

#WARTEN AUF INFORMATIONEN VOM AUTHSERVER
$auth = IO::Socket::INET->new( LocalPort => "2401",
        Type => SOCK_STREAM, Reuse => 1, Listen => 2);
setsockopt($auth, &Socket::IPPROTO_TCP, &Socket::TCP_NODELAY, 1);
my $client = $auth->accept();
$client->recv(my $AuthAntwort, 1024,0);
close($auth);
my ($ScreenName, $BOSIP, $BOSPORT, $AUTHCOOKIE)=AuthdatagrammToData(
    $AuthAntwort);
print "Abgefangener BOS-Server $BOSIP : $BOSPORT\n";
```

```
#DAMIT BOS NICHT VON VORNHEREIN IP ABWEIST, ANMELDUNG MIT WEITEREN
    BENUTZERNAMEN
my $login = IO::Socket::INET->new( PeerAddr => "login.oscar.aol.com",
        PeerPort => "5190",         Proto    => "tcp",
        Type     => SOCK_STREAM);
setsockopt($login, &Socket::IPPROTO_TCP, &Socket::TCP_NODELAY, 1);
$login->send($fakelogin,0);
$login->recv(my $fakeantwort, 1024, 0);
#benoetigten infos
$login->recv($fakeantwort, 1024, 0);
my ($FakeName, $FakeIP, $FakePORT, $FakeCOOKIE)=AuthdatagrammToData(
    $fakeantwort);
print "Fake BOS-Server $FakeIP : $FakePORT \n\n";

if ($BOSIP eq $FakeIP){
  print "Verbindungsaufbau zum BOS:\n";
  my $bos = IO::Socket::INET->new(PeerAddr => $BOSIP,
        PeerPort => $BOSPORT,     Proto     => "tcp",
        Type     => SOCK_STREAM)
        or die "Fehler: $!\n";
    my $authdata = $header.$AUTHCOOKIE;
    $bos->recv(my $newcon, 1024, 0);
    print "\"New Connection\" des Servers :".unpack("H*",$newcon)."\n\n";
    $bos->send($authdata,0);
    $bos->recv(my $serverready, 1024, 0);
    print "\"Server Ready\":".unpack("H*",$serverready)."\n\n";
}else {print "Abbruch: Verschiedene BOS-Adressen\n\n"};


sub AuthdatagrammToData
#extrahiert die benoetigten Informationen aus dem vom Auth-Server
    empfangenden Datagramm
#Eingabe: Nutzlast der Antwort, Ausgabe: ScreenName, IP & Port des BOS, Auth-
    Cookie
{
  my $antwort = $_[0];
  #ScreenNamsSize. offset=6header+2valueID
  my $sns = hex(unpack("H*", substr($antwort,8,2)));
  #ScreenName offset=10newconnection+6header+2valueID+2$sns
  my $sn = substr($antwort,10,$sns);
  #ServerStringSize offset=18+2SNS+SNS+2valueID
  my $sss = hex(unpack("H*", substr($antwort,12+$sns,2)));
  #BOS -Adresse und Port
  my ($bosIP, $bosPort) = split(/:/,substr($antwort,14+$sns,$sss));
  #extrahieren des auth-coockie
  my $authcookie =  substr($antwort,-256, 256);
  return($sn,$bosIP,$bosPort,$authcookie);
}
```

C.2. TOC-Klient

```
#!/usr/bin/perl
#siehe http://search.cpan.org/search?dist=Net-AOLIM

use Net::AOLIM;
my $login_server = "toc.oscar.aol.com";
my $port = "9898";

if ($#ARGV == 1){
        $user = $ARGV[0];
        $password = $ARGV[1]
   } else { die "Kein Benutzernamen und Passwort eingegeben\n"}


sub handler {
  my ($type, $buddy_name, $something, $message) = @_;
  print "$type -Nachricht von $buddy_name: \n";
  print $message;
  }

$toc = Net::AOLIM->new('username' => $user, 'password' => $password,
                       'server' => $login_server, 'port' => $port,
                        'callback' => \&handler,
                        'aim_agent'=> "SHOL's AIM-Testklient",
                        'allow_srv_settings' => 0,
                        'login_timeout' => 0
                        ) or die;

$toc->im_permit_all();
$toc->add_buddies("Buddies", "imstudie");
$toc->signon();

print $Net::AOLIM::ERROR_MSGS{$IM_ERR};

while (1) {
    last unless defined($toc->ui_dataget(undef));
}
```

C.3. MSN-Klient

```
#!/usr/bin/perl -w
#siehe http://search.cpan.org/~DJR/Net-MSN/
use Net::MSN;
use Crypt::SSLeay; #wird fuer https-authentisierung benoetigt

if ($#ARGV == 1){
        $screenname = $ARGV[0];
        $password = $ARGV[1]
   } else { die "Kein Benutzernamen und Passwort eingegeben\n"}

my $msn = new Net::MSN(
   Debug            =>  1,
   Debug_Lvl        =>  3,
```

```
    Debug_STDERR     =>  1,
    Debug_LogCaller =>  1,
    Debug_LogTime    =>  1,
    Debug_LogLvl     =>  1,
    Debug_Log        =>  "/tmp/msn.log"
    );

$msn->set_event(
  on_connect => sub{
                  print "Anmeldung abgeschlossen\n";
                },
  on_message => sub{
                  my ($sb, $contact, $friendly, $message) = @_;
                  print "$contact : $message\n";
                  $msn->sendmsg($contact, "Ack\n");
                },
  on_join    => sub{
                  my ($sb, $chandle, $friendly) = @_;
                  print "$chandle / $friendly";
                }
);

$msn->connect($screenname, $password);

while (1) {
  $msn->check_event();
}
```

C.4. XMPP-Klient

```
#!/usr/bin/perl -w
#siehe http://cpan.uwinnipeg.ca/htdocs/Net-XMPP/
use strict;
use diagnostics;
use Net::XMPP qw(Client);
my $jabber = new Net::XMPP::Client(debuglevel=>0);

$SIG{INT}  = sub {
                  print "Abbruch\n";
                  $jabber->Disconnect();
                  exit(0);
                };

if ($#ARGV == 1){
        $screenname = $ARGV[0];
        $password = $ARGV[1]
   } else { die "Kein Benutzernamen und Passwort eingegeben\n"}

$jabber->SetCallBacks(
  message =>  sub { my ($sid,$message) = @_;
                    my $fromJID = $message->GetFrom("jid");
                    my $from = $fromJID->GetUserID();
                    my $subject = $message->GetSubject();
                    my $body = $message->GetBody();
```

```
                    print "$from:\t$subject\n$body\n\n";
                    $jabber->MessageSend(
                      to       => $fromJID,
                      subject  => "Ack",
                      body     => $body)},
  presence => sub { my ($sid,$presence) = @_;
                    print $presence->GetXML(),"\n"},
  iq =>       sub { my ($sid,$iq) = @_;
                    print $iq->GetXML(),"\n"}
);

$jabber->Connect('hostname' => "jabber.org", 'tls' => '1');

my @cnres = $jabber->AuthSend(
        hostname => "jabber.org",
        username => $screenname,
        password => $password,
        resource => "perlklient"
);

print "Sende Anforderung zur Übermittlung der Kontaktliste\n";
$jabber->RosterGet();

print "Sende Kontaktbereitschaft\n";
$jabber->PresenceSend(  status=>"Bin beim Essen",
                        type=>"unavailable",
                        show=>"online",
                        priority=>6
                        );

#print "Warte auf Ereignisse (Abbruch durch CTRL+c)\n";
$jabber->MessageSend(
                to => 'bla@jabber2.clowntown.priv/perlklient',
                 subject => "Betreffszeile",
                 body    => "Und hier steht die eigentliche Nachricht");

$jabber->Disconnect();

sub Abbruch{
        $jabber->Disconnect();
        exit(0);
}
```

C.5. Algorithmus zur Bildung des SASL-Responses

```
#!/usr/bin/perl -w
#Rechenvorschrift aus: http://www.ietf.org/rfc/rfc2831.txt
#Werte der Variablen durch Sniffen gewonnen
#zum dekodieren von base64: decode_base64(string)

use MIME::Base64;
use Digest::MD5 qw(md5 md5_hex);

#1. Challenge Server -> Client
```

```
my $realm="jabber.clowntown.priv";
my $nonce="fddce3a7b07153103c9912d13135634b8cac930d";
my $qop="auth";
my $charset="utf-8";
my $algorithm="md5-sess";

#1. Response Client -> Server
my $passwort = "ein_passwort";
my $cnonce = "f213dd122aefc19c2b5b1c680be1ebe0";
my $digest_uri="/";
my $nc="00000001";
my $username="beispiel";
my $rA1 = md5($username.":".$realm.":".$passwort).":".$nonce.":".$cnonce;
my $rA2 = "AUTHENTICATE:".$digest_uri;
my $response =  md5_hex(md5_hex($rA1).":".$nonce.":".$nc.":".$cnonce.":".
                $qop.":".md5_hex($rA2));
print "1. Response:\tresponse=$response\n";

#2. Challenge Server -> Client
my $cA1 = md5($username.":".$realm.":".$passwort).":".$nonce.":".$cnonce;
my $cA2 = ":".$digest_uri;
my $rspauth =  md5_hex(md5_hex($cA1).":".$nonce.":".$nc.":".$cnonce.":".
               $qop.":".md5_hex($cA2));
print "2. Challenge:\trspauth=$rspauth\n";
```

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[1] Studie: Instant Messenger häufig zum Klönen während der Arbeit genutzt
http://www.heise.de/newsticker/meldung/53198

[2] heise.de: AOL will von offenem Instant Messaging nichts mehr wissen
http://www.heise.de/newsticker/meldung/29346

[3] freiheit.com: 150 Millionen User können nicht irren!
Einsatz von Instant Messaging im Unternehmen.
(http://www.freiheit.com/technologies/download)

[4] Nielsen//NetRatings: Instant Messaging fesselt die Surfer
http://www.nielsen-netratings.com/pr/pr_040322_gr.pdf

[5] CERT Coordination Center: Denial of Service Attacks
(http://www.cert.org/tech_tips/denial_of_service.html)

[6] ICQ Terms Of Service
(http://www.icq.com/legal)

[7] Alexandr Shutko, OSCAR (ICQ v7/v8/v9) protocol
(http://iserverd1.khstu.ru/oscar)

[8] AIM Java Developer Pages: Terminology
(http://www.aim.aol.com/javadev/terminology.html)

[9] TOC specification
(http://www.fruhead.com/joshw/unused/PROTOCOL)

[10] Nutzungsbedingungen, Lizenzen und Hinweise für .NET Messenger
(http://messenger.msn.com/Help/Terms.aspx?mkt=de)

[11] Offizielle Spezifikation des *MSN Messenger*-Protokolls, Version 1
(http://www.hypothetic.org/docs/msn/ietf_draft.txt)

[12] Mike Mintz, MSN Messenger Protocol Version 8
(http://www.hypothetic.org/docs/msn/)

[13] Gespräch mit Torsten Werner, Referent für IT-Strategie, Auswärtiges Amt am 5. März 2005

[14] Gespräch mit Bert Radke, Softwareentwickler bei „jabber-fanatix“ am 5. März 2005

[15] RFC's zum *Extensible Messaging and Presence Protocol (XMPP)*
RFC 3920 (http://www.ietf.org/rfc/rfc3920.txt)
RFC 3921 (http://www.ietf.org/rfc/rfc3921.txt)
RFC 3922 (http://www.ietf.org/rfc/rfc3922.txt)
RFC 3923 (http://www.ietf.org/rfc/rfc3923.txt)

[16] RFCs zum *Internet Relay Chat* (IRC)
RFC 1459: (http://www.irchelp.org/irchelp/text/rfc1459.txt)
RFC 2810: (http://www.irchelp.org/irchelp/rfc/rfc2810.txt)
RFC 2811: (http://www.irchelp.org/irchelp/rfc/rfc2811.txt)
RFC 2812: (http://www.irchelp.org/irchelp/rfc/rfc2812.txt)
RFC 2813: (http://www.irchelp.org/irchelp/rfc/rfc2813.txt)

[17] SILC-Spezifikationen - alle sind bei einer Silc-Installation lokal verfügbar:
SILC Protocol Specification: draft-riikonen-silc-spec-07.txt
SILC Packet Protocol: draft-riikonen-silc-pp-07.txt
SILC Key Exchange and Authentication Protocol:draft-riikonen-silc-ke-auth-07.txt

[18] Microsoft Office Online: Live Communications Server home page
(http://www.microsoft.com/office/livecomm)

[19] Wikipedia - die freie Enzyklopädie
AES (http://de.wikipedia.org/wiki/Advanced_Encryption_Standard)
PKCS (http://de.wikipedia.org/wiki/PKCS)
SSL (http://de.wikipedia.org/wiki/Secure_Sockets_Layer)
TLS (http://de.wikipedia.org/wiki/Transport_Layer_Security)

[20] Geheime Botschaften von Simon Singh
ISBN: 3423330716

[21] Angewandte Kryptographie von Bruce Schneier
Addision-Wesley, 1996, ISBN: 3893198547

Erklärungen

Haftungsausschluss

Der Autor, die Technische Universität Chemnitz und das Bundesamt für Sicherheit in der Informationstechnik übernehmen keine Haftung für Schäden, die aus der Benutzung dieser Diplomarbeit entstehen.

Schutzrechte

Eingetragene Waren- und Markenzeichen sind in diesem Text nicht als solche gekennzeichnet. Das Fehlen der Kennzeichen bedeutet nicht, dass diese Zeichen frei verwendbar sind.

Selbstständigkeitserklärung

Hiermit erkläre ich, Holger Schildt, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur angefertigt habe.

Chemnitz, den 10. Juli 2005

Holger Schildt