

6 Internet-Dienste

6.1 Internet-Protokolle	2
6.2 TCP	8
6.3 UDP	21
6.4 Gruppenruf	29
6.5 Sicherheitsprobleme	37
Zusammenfassung	50
Quellen	51
Anhang: Sockets mit C	52

6.1 Internet-Protokolle

Zur Erinnerung:

Internet Protocol - IP

Paketvermittlung:

verbindungsloser Vermittlungsdienst zwischen Stationen:
Daten-Pakete werden von Quellstation zu Zielstation
übertragen, i.a. über verschiedene Zwischenstationen.

Geringe Dienstqualität:

keine Reihenfolgetreue, keine Garantie gegen Paketverlust
oder gegen Duplikate.

- Identifizierung einer Station im Netz durch **IP-Adresse** (auch *Internet-Adresse*)
- traditionell 32 Bits, notiert als 4 Bytes in Dezimaldarstellung, z.B. 130.149.27.12 (Übergang nach IPv6)
- IP-Adresse ist *per Software* festgelegt und wird im Lokalnetz auf eine physikalische Netzadresse abgebildet (**MAC-Adresse** - *media access control*)
- **Routing**-Funktionalität des IP-Protokolls besorgt das Weiterleiten von Paketen über Zwischenstationen zur endgültigen Zielstation (mit Hilfe von *routing tables*)

Stationsnamen und DNS:

- mnemonische Identifizierung von Stationen durch Namen, z.B. localhost, lounge, xian (im lokalen Netz)
- weltweiter hierarchischer Namensraum: mit Namen wie lounge.mi.fu-berlin.de für den Rechner 160.45.42.83

```
$ ping lounge.mi.fu-berlin.de
PING lounge.imp.fu-berlin.de (160.45.42.83): ...
```
- **Domain Name System - DNS** - ist zuständig für Zuordnung zwischen Namen und Adressen
- Hilfreich: <http://ping.eu>

```
package java.net;
```

```
class InetAddress: IP-Adressen und Stationsnamen  
in verschiedenen Formaten
```

Beispiel:

```
import java.net.*;  
class IP {  
public static void main(String[] arg) throws Exception {  
    InetAddress a = InetAddress.getByName(args[0]);  
    System.out.println(a.getHostAddress()); }  
}
```

```
$ java IP nawab.mi.fu-berlin.de  
160.45.42.24  
$
```

Zur Erinnerung: Transportdienste

Transportdienst besorgt den Transport von Daten zwischen Kommunikationsendpunkten bei Prozessen - **Ports** genannt. Die Implementierung orientiert sich an der Spezifikation zugehöriger *Transportprotokolle* und benutzt den IP-Dienst.

UDP (*user datagram protocol*) realisiert verbindungslosen, unzuverlässigen, unidirektionalen Transportdienst für Datenpakete.

TCP (*transmission control protocol*) realisiert verbindungsorientierten, zuverlässigen, bidirektionalen Transportdienst für Byteströme.

Portnummern:

- 0..65535
- 0..1023 reserviert für Standard-Dienste
- 1024..49151 weitere vereinbarte Dienste

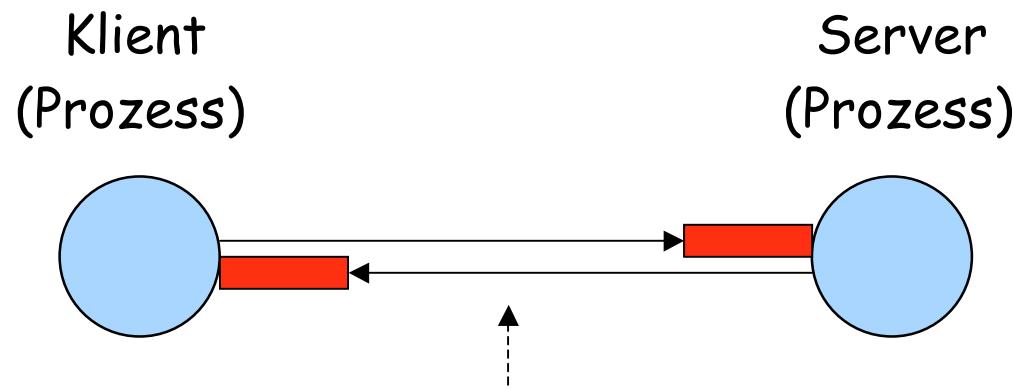
Socket (dt. Steckdose, Stutzen)

ist die technische Realisierung eines Kommunikations-Endpunktes; er beinhaltet die folgenden Daten:

- Protokollfamilie, Dienst, Protokoll
- IP-Adresse, Portnummer
- Verbindungsdaten (im Fall von TCP)

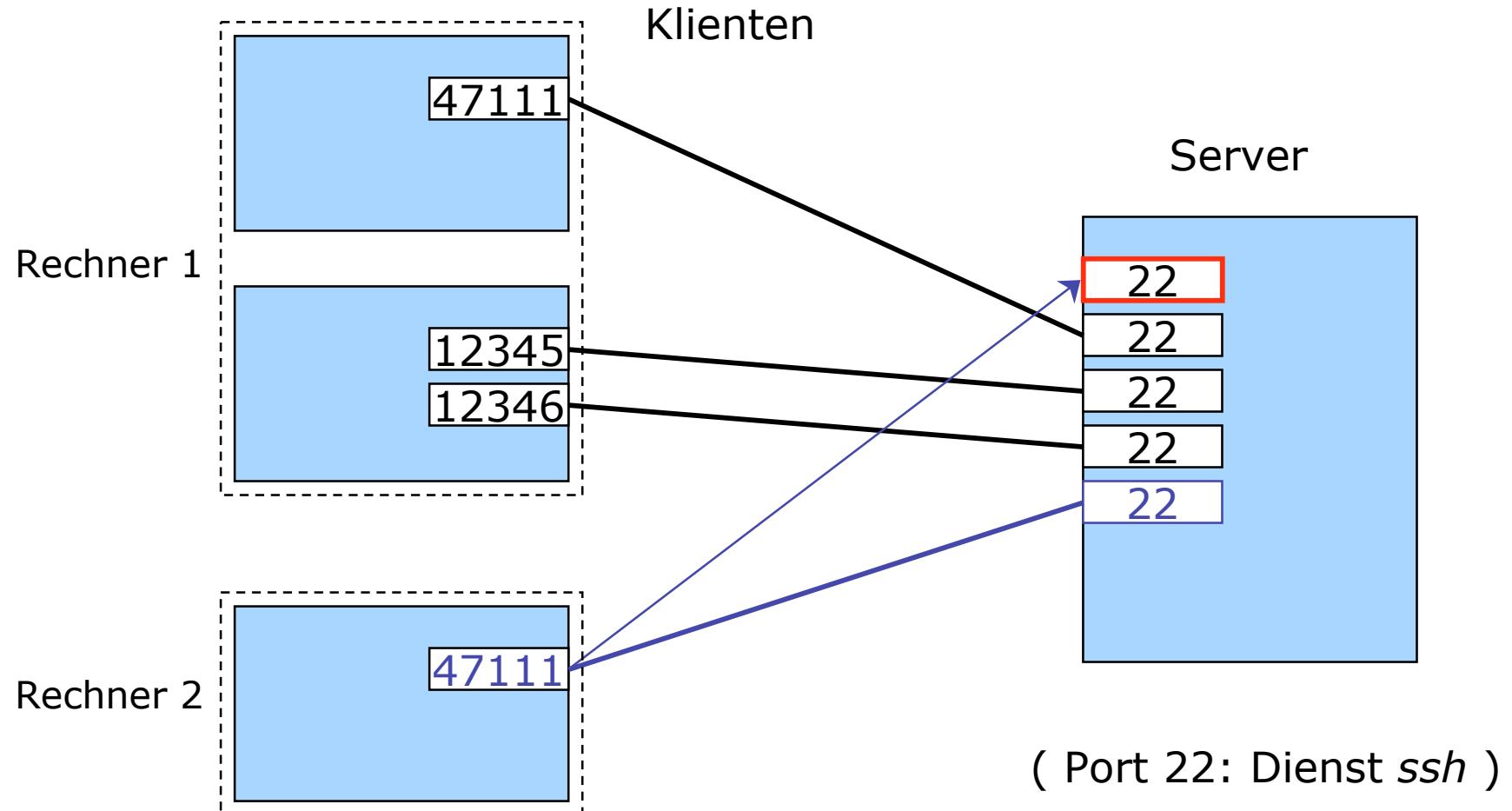
6.2 TCP

TCP ist für dienstorientierte Architekturen konzipiert.



Verbindung für zwei zuverlässige Byteströme
- dies sieht symmetrisch aus, aber:

Klient kennt Server und äußert Verbindungswunsch.
Server akzeptiert den Verbindungswunsch



package java.net:

abstract class SocketAddress

Socket-Daten

class InetSocketAddress **extends** SocketAddress

Socket-Daten für Internet-Dienste,
bestehend aus IP-Adresse und Portnummer

```
class Socket {    Klienten-Sockets !
public Socket()    leerer, unverbundener Socket
public Socket(String host, int port) throws ...
                    verbundener Socket, mit port auf host
                    und bislang unbenutztem lokalen Port
public void bind(SocketAddress bindpoint) throws ...
                    belegt den Socket mit Daten
public void connect(SocketAddress endpoint) throws ...
                    verbindet den Socket mit dem
                    Server Socket endpoint

public void close() throws ...
                    macht den Socket unbenutzbar und trennt
                    eine eventuell existierende Verbindung
public OutputStream getOutputStream() throws ...
public InputStream  getInputStream() throws ...
                    für das Senden und Empfangen von Bytes
```

```
class ServerSocket {  Server Sockets!  
public ServerSocket() throws ...  
                      leerer, unverbundener Socket  
public ServerSocket(int port, int backlog) throws ...  
                      empfangsbereiter Socket  
public void bind(SocketAddress endpoint, int backlog)  
                      belegt den Socket mit Daten  
public Socket accept() throws ...  
                      liefert einen neuen Socket, der mit dem  
                      Socket eines Klienten verbunden ist  
public void close() throws ...  
                      macht den Socket unbenutzbar und trennt  
                      eine eventuell existierende Verbindung  
}
```

Vergleich:

lokal



```
Port p = sendarg(arg, server);
Port p = recvarg(argvar);
```

```
sendres(res, p);
recvres(resvar, p);
```

versus

TCP



```
s1 = new ServerSocket(17,5);
s = new Socket();
s.connect(serverport);
s2 = s1.accept();
in = s.getInputStream();
out = s.getOutputStream();
is = s2.getInputStream();
os = s2.getOutputStream();
out.println(arg);
line = in.readLine();

out.println(res);
res = in.readLine();

(serverport = new InetSocketAddress(host,17)
BufferedReader in = ...
printStream out = ... )
```

Komplikationen bei TCP gerechtfertigt ?

① Identifizierung des Servers

Stationsname und Portnummer statt einfach Tid oder Pid.

Stationsname: bedingt durch physische Verteilung ✓

② Portnummer

Portnummer statt Pid identifiziert einen Kanal, über den ein bestimmter Dienst erreicht werden kann.

Statt nur einer „Mailbox“ hat ein Server potentiell mehrere Ports/Dienste! ✓

③ Verbindung

Feste Verbindung statt ad-hoc-Kontaktaufnahme ermöglicht längeren Dialog statt nur einzelner Auftragsabwicklung ✓

Testen von Diensten mit dem „generischem“ Klienten netcat:

`nc` stellt Verbindung mit Dienst her und erlaubt Interaktion über Tastatur/Bildschirm;
Anwendungsprotokoll beachten!

```
$ nc nawab.mi.fu-berlin.de 13          (Port 13: Dienst daytime)  
Sat Oct 18 18:13:11 2008
```

```
$ nc localhost 79          (Port 79: Dienst finger)  
lohr  
Login: lohr  
Directory: /Users/lohr          Name: Klaus-Peter Loehr  
                                Shell: /bin/bash
```

```
$ nc lounge.mi.fu-berlin.de 22          (Port 22: Dienst ssh)  
SSH-2.0-OpenSSH_5.5p1 Debian-6+squeeze2  
hallo!  
Protocol mismatch.
```

Testen von Klienten mit einfachem Dienst Echo:

Echo antwortet auf Textzeile mit ähnlicher Textzeile

Starten des Dienstes:

```
nawab: java Echo 54321 &  
[1] 2011  
nawab:
```

Benutzung:

```
mypc: nc nawab 54321  
ping  
pong  
risibisi  
rosoboso  
^D  
mypc:
```

mit eigenem Klienten:

```
mypc: java Netcat nawab 54321  
ping  
pong  
risibisi  
rosoboso  
^D  
mypc:
```

```
import java.io.*;
import java.net.*;
public class Echo {
public static void main (String arg[]) throws IOException {
    int port = Integer.parseInt(arg[0]);
    ServerSocket listen = new ServerSocket(port);
    while(true) { // this is a non-terminating server
        Socket socket = listen.accept();
        InputStream is = socket.getInputStream();
        OutputStream os = socket.getOutputStream();
        BufferedReader in = new BufferedReader(
                            new InputStreamReader(is));
        PrintStream out = new PrintStream(os);
        while(true) { // begin dialogue
            String request = in.readLine();
            if(request==null) break;
            String answer = request.replace('i','o');
            out.println(answer);
        } // end of input stream will end dialogue
        in.close(); out.close(); socket.close();
        // ready to accept new client
    }
}
```

```
import java.io.*;
import java.net.*;
public class Netcat {
    public static void main (String arg[]) throws IOException {
        String host = arg[0];
        int      port = Integer.parseInt(args[1]);
        Socket socket = new Socket(host, port);
        InputStream is   = socket.getInputStream();
        OutputStream os  = socket.getOutputStream();
        BufferedReader in = new BufferedReader(
                            new InputStreamReader(is));
        PrintStream out   = new PrintStream(os);
        BufferedReader keyboard = new BufferedReader(
                                new InputStreamReader(System.in));
        while(true) { // get user input; ^D will close dialogue
            String request = keyboard.readLine();
            if(request==null) break;
            out.println(request);
            String answer = in.readLine();
            System.out.println(answer);    }
        in.close(); out.close(); socket.close();
    }
}
```

Nichtsequentieller Server

kann mehrere Dialoge gleichzeitig führen (vgl. schwacher Monitor!):

```
import java.io.*;
import java.net.*;
public class AnswerQuestion {
    public static void main (String arg[]) throws IOException {
        int port = Integer.parseInt(arg[0]);
        ServerSocket socket = new ServerSocket(port);
        while(true) {
            Socket s = socket.accept(); // new client ...
            new Service(s).start(); } // ... gets new thread
    } // end main
    static class Service extends Thread {
        Service(Socket s) { socket = s; }
        Socket socket;
        public void run() .....
    }
}
```

Alternativ, z.B. bei sequentiellen Sprachen:
neuen schwergewichtigen Prozess erzeugen (Unix: `fork`)

Merke:

In der Regel gehört zu einem Standard-Dienst im Internet eine vereinbarte Portnummer und ein Server-Programm, ferner ein Klienten-Programm, das das Anwendungsprotokoll des Servers versteht und eine textuelle oder graphische Schnittstelle für den menschlichen Benutzer hat. Beispiele:

Port	Dienst	Server	Klient
13	daytime	daytimed	daytime
22	Secure Shell	sshd	ssh, scp
25	E-mail (smtp)	sendmail	Mailtool
80	World-Wide Web	httpd	Browser
.....			

6.3 UDP

Zur Erinnerung:

Bei *UDP* wird ein Paket („Datagramm“) auf den Weg von einem Port zu einem anderen Port geschickt.

Dass das Datagramm am Ziel eintrifft ist *nicht garantiert*, Reihenfolgetreue (FIFO) auch *nicht*. Auch *Doppelungen* können vorkommen.

Vergleich:

lokal

versus

UDP



(nicht dienstorientiert!)

send(msg, peer);

recv(msgvar);

DatagramSocket socket =
new DatagramSocket(port);

DatagramSocket socket =
new DatagramSocket();

DatagramPacket out =
new DatagramPacket(
 msg, msg.length, host, port);
socket.send(out);

DatagramPacket in =
new DatagramPacket(
 msgbuf, msgbuf.length);
socket.receive(in);

(blau - Sender, rot - Empfänger)

Komplikationen bei UDP gerechtfertigt ?

① Identifizierung des Empfängers

Stationsname und Portnummer statt einfach Tid oder Pid.

Stationsname: bedingt durch physische Verteilung ✓

② Portnummer

Portnummer statt Pid: identifiziert einen Kanal, hinter dem der Empfänger steht. Anstelle nur einer „Mailbox“ hat der Empfänger potentiell mehrere Ports! ✓

③ Socket beim Sender

... ist eigentlich entbehrlich - aber: häufig für Rückmeldungen bei UDP-basierten Diensten benutzt ! ✓

Variante des *Echo-Dienstes* von S. 16 - Starten:

```
$ java EchoUDP 54321 &
[1] 2011
$
```

Testen des Dienstes:

```
$ nc -u localhost 54321      (Option -u steht für UDP)
ping
pong
singing
songong
^D
$
```

```
import java.net.*;
import java.io.*;

public class EchoUDP {
    public static void main(String arg[]) throws IOException {
        int port = Integer.parseInt(arg[0]);
        byte[] inData = new byte[1024];
        byte[] outData = new byte[1024];
        DatagramPacket in = new DatagramPacket(inData,inData.length);
        DatagramSocket socket = new DatagramSocket(port);

        while(true) {                                // non-terminating server
            socket.receive(in);                      // wait for message and get it
            InetAddress senderAdress = in.getAddress();
            int senderPort = in.getPort();
            String message = new String(in.getData(),0,in.getLength());
                                            // build answer:
```

```
        // build answer:  
        outData = message.replace('i','o').getBytes();  
        DatagramPacket out = new DatagramPacket(  
            outData, outData.length,  
            senderAddress, senderPort);  
        socket.send(out);    // send answer  
    } // end loop  
}
```

```
import java.io.*;
import java.net.*;

public class NetcatUDP {
    public static void main (String arg[]) throws IOException {
        InetAddress server = InetAddress.getByName(arg[0]);
        int port = Integer.parseInt(args[1]);
        byte[] inData = new byte[1024];
        byte[] outData = new byte[1024];
        DatagramPacket in = new DatagramPacket(inData,inData.length);
        DatagramSocket socket = new DatagramSocket();
                                // gets any available local port
        BufferedReader keyboard = new BufferedReader(
                                new InputStreamReader(System.in));
        while(true) {           // dialogue starts here:
```

```
while(true) {                                // ^D will close dialogue
    String message = keyboard.readLine();
    if(message==null) break;
    outData = message.getBytes();
    DatagramPacket out = new DatagramPacket(outData,
                                              outData.length, server, port);
    socket.send(out);                         // send request
    socket.receive(in);                      // receive reply
    String answer = new String(in.getData(),0,in.getLength());
    System.out.println(answer);
}
socket.close();
}
}
```

6.4 Gruppenruf (*multicast*)

- IP Multicast ist ein verbindungsloser Vermittlungsdienst, bei dem die Zieladresse eines Datagramms nicht einen bestimmten Rechner, sondern eine Gruppe von Rechnern identifiziert.
- Für solche Gruppenadressen ist folgender Bereich reserviert:
Empfohlen für „private“ Zwecke:

 - 224.0.0.1 bis 239.255.255.255
 - 225.0.0.0 bis 238.255.255.255

- Rechner können mit Nennung einer solchen Adresse einer bestimmten Gruppe beitreten.

- **UDP Multicast** ist UDP mit einer Gruppenadresse:
Das Multicast-Datagramm geht an den gewählten Ziel-Port
bei allen Gruppenmitgliedern.
- *Internet Group Management Protocol (IGMP)*
ist zuständig für die Gruppenverwaltung.
- Gruppenruf wird üblicherweise auf einen bestimmten
Lokalitätsbereich beschränkt: die Datagramme enthalten
kleine Werte für die „*Time to Live*“, z.B. 0 für einen
rechnerinternen Gruppenruf, 1 für Beschränkung auf das
Lokalnetz, dem der Ruf entstammt.
- Empfehlenswert: <http://tldp.org/HOWTO/Multicast-HOWTO.html>

```
public class MulticastSocket extends DatagramSocket {  
    public MulticastSocket()...  
    public MulticastSocket(int port)...  
        // Socket erzeugen, Portnummer implizit oder explizit gewählt  
    public void setTimeToLive(int ttl) ...  
        // time_to_live setzen - Voreinstellung ist 1!  
    public void joinGroup(InetAddress mcastaddr) throws ...  
        // Gruppe unter der Adresse mcastaddr beitreten;  
    public void leaveGroup(InetAddress mcastaddr) throws ...  
        // Gruppe verlassen  
}
```

`send, receive, ...` werden von Oberklasse `DatagramSocket` geerbt!

Beispiel:

Prozess mit Programm `Name` tritt einer Gruppe bei und antwortet auf Anfrage mit einem vom Benutzer gewählten Namen.

Programm `Who` sendet Rundmeldung "who?" an diese Gruppe.

\$ java Name Alice &

\$ java Name Bob &

\$ java Name Carl &

\$ java Who

Bob

Alice

Carl

\$

```
import java.net.*;  
  
public class Who {  
    public static void main(String[] arg) throws Exception {  
        String whoGroup = "229.1.2.3";  
        InetAddress group = InetAddress.getByName(whoGroup);  
        DatagramSocket socket = new DatagramSocket();  
        String who = "who?";  
        DatagramPacket out = new DatagramPacket(who.getBytes(),  
                                              who.length(), group, 54321);  
  
        byte[] inData = new byte[1024];  
        DatagramPacket in = new DatagramPacket(inData, inData.length);  
        socket.setSoTimeout(2000); // answers not arriving  
                               // within 2 sec will be ignored  
        socket.send(out);      // send "who?"  
        ....  
    }  
}
```

```
while(true) {  
    try{ socket.receive(in);  
        String name =  
            new String(in.getData(), 0, in.getLength());  
        System.out.println(name);  
    } catch (SocketTimeoutException e) { break; }  
}  
socket.close();  
}  
}
```

```
import java.net.*;
public class Name {
public static void main(String[] arg) throws Exception {
    byte[] inData = new byte[1024];
    byte[] outData = args[0].getBytes(); // chosen name
    String whoGroup = "229.1.2.3";
    InetAddress group = InetAddress.getByName(whoGroup);
    MulticastSocket socket = new MulticastSocket(54321);
    socket.joinGroup(group);
    DatagramPacket in = new DatagramPacket(inData, inData.length);
    while(true) ...                                // wait for inquiries:
```

```
while(true) {  
    socket.receive(in);           // get message, e.g., "who?"  
    InetAddress senderAddress = in.getAddress();  
    int senderPort = in.getPort();  
    DatagramPacket out = new DatagramPacket(outData,  
                                              outData.length(),  
                                              senderAddress, senderPort);  
    socket.send(out);            // reply with user name  
}  
}  
}
```

Beachte: Anders als bei einem Unicast Port erhält
bei einem Multicast Port jeder teilnehmende
Prozess eine eigene Kopie der Nachricht !

6.5 Sicherheitsprobleme

IT-Sicherheit (IT security) = Systemsicherheit + Netzsicherheit

Im Netz Schwachstellen (*vulnerabilities*) bei Angriffen auf

- Transportdienst
- Server
- Klienten

Verteidigung:

- technisch (Hardware, Software)
- organisatorisch (System/Netzverwaltung)
- pragmatisch (Benutzer)

Typische Angriffe im Netz:

- **Abhören (eavesdropping)** übertragener Daten
 - **Beschädigen/Zerstören (corruption)** übertragener Daten
 - **Verfälschen (tampering)** übertragener Daten
 - Umlenken der Verbindung zum Angreifer (*session hijacking*)
 - **Eindringen (intrusion)** in fremdes System
 - **Maskerade (spoofing)**: Täuschung über die Identität eines Partners
 - ... und natürlich **Bauernfang** bei E-mail, Web, ... !
- *sichere Verbindungen helfen ein Stück weit:*
- Secure Socket Layer (SSL)** oberhalb von TCP → `javax.net.ssl`
(z.B. für *https* - sichere Web-Verbindungen)



Ein angebotener Dienst kann von **beliebigen Klienten**,
die **nicht registrierte Benutzer** sein müssen,
in Anspruch genommen werden.

Eine Zugangskontrolle ist zunächst nicht vorgesehen.

Konsequenz:

Klient kann **bösartige Daten** an Server schicken - und umgekehrt.
Port ist Einfallstor für Angriffe - und auch Ausfallstor.

Def.: Eine Programmiersprache heißt **unsicher (unsafe)**, wenn Programmfehler die Speicherverwaltung stören können.

Beispiele: ungültige Verweise in C, Pascal, Modula, ...; zusätzlich Feldgrenzenüberschreitung in C/C++; bei Eingabedaten: „Pufferüberlauf“ (buffer overflow)

Folgen: chaotisches Programmverhalten,
Absturz mit „Bus Error“, „Segmentation Fault“ u.ä.

oder **gezieltes Fehlverhalten** aufgrund **bösartiger Eingabedaten**
(Angreifer nutzt durch Programmfehler bedingte Schwachstelle)

SYNOPSIS

```
#include <stdio.h>
char *gets(char *s);
char *fgets(char *s, int n, FILE *stream);
```

DESCRIPTION

The gets() function **reads characters from the standard input stream** (see intro(3)), stdin, into the array pointed to by s, **until a newline character is read** or an end-of-file condition is encountered. The newline character is discarded and the string is terminated with a null character.

The fgets() function reads characters from the stream into the array pointed to by s, until n-1 characters are read, or a newline character is read and transferred to s, or an end-of-file condition is encountered. The string is then terminated with a null character.

When using gets(), if the length of an input line exceeds the size of s, indeterminate behavior may result. For this reason, it is strongly recommended that gets() be avoided in favor of fgets().

... und viele andere:

NAME

string, strcasecmp, strncasecmp, strcat, strncat, strlcat,
strchr, strrchr, strcmp, strncmp, strcpy, strncpy, strlcpy,
strcspn, strspn, strdup, strlen, strpbrk, strstr, strtok,
strtok_r - string operations

DESCRIPTION

The arguments s, s1, and s2 point to strings (arrays of characters terminated by a null character). The strcat(), strncat(), strlcat(), strcpy(), strncpy(), strlcpy(), strtok(), and strtok_r() functions all alter their first argument. **These functions do not check for overflow of the array pointed to by the first argument.**

Anwendung von `gets` z.B. so:

```
char line[100]; . . . . . gets(line);
```

wobei der Programmierer vergessen hat, längere Zeilen
(länger als 99 Zeichen) als fehlerhaft abzufangen!

Gefahr: Wenn eine längere Zeile eingegeben wird,
wird der Keller beschädigt, und das Programm
arbeitet fehlerhaft oder stürzt ab.

Dies ist eine **sicherheitsrelevante Schwachstelle**,
wenn das Programm mit anderer Autorisierung
als der des Benutzers/Klienten ausgeführt wird!

(typisch: setuid-Programm, Server-Programm!)

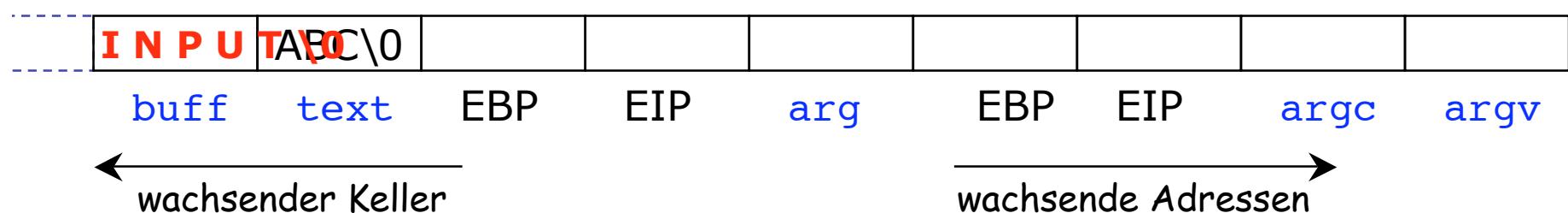
```

void getarg (char *arg ){
    char text[4] = "ABC";
    char buff[4];
    strcpy(buff, arg);
    .....
}
int main( int argc, char *argv[] ) {
    if (argc >1) getarg(argv[1]);
    .....
}

```

\$ a.out INPUT

Keller (wächst von oben nach unten):



Trivialer Echo-Dienst - in C :

```
#include <sys/types.h>
#include <sys/socket.h>          // vgl. Anhang - S. 52
#include <netinet/in.h>
int main(int argc, char *argv[ ]) {
    int max = 100;
    char line[max];
    .... // initialization, port 54321
    for(;;) {
        int connfd =
            accept(listenfd,
                    (struct sockaddr *) &clientaddr, &len);
        readln(connfd, line, max);
        print(line);
        line[5] = '\n'; write(connfd,line,6);
        close(connfd);    }
}
```

```
void print(char *str) {  
    char buf[10];      // eine 0 vergessen!  
    strcpy(buf, str);  
    printf("got %s", buf);  
}
```

server: a.out

got first test

got fuzzfuzzfuzzfu...
Segmentation fault

server:

badguy: nc localhost 54321
first test

first

badguy: !!

fuzzfuzzfuzzfuzzfuzzfuzzfuzz

badguy:

Server abgeschossen !

- Was ist passiert?
- Rücksprungadresse aus `print` im Keller wurde durch `strcpy` mit Daten überschrieben.
- Rücksprung aus `print` erfolgte an eine ungültige Adresse
→ *Segmentation fault*

- Effekt: Abschuss des Servers → „denial of service“
- Beachte: mit der gleichen Technik kann auch ein bösartiger Server einen schwachstellenbehafteten Klienten zum Absturz bringen!

Noch viel gefährlicher ist das **Einschleusen von Schadcode**:

- Keller mit Daten *und Code (!)* überschreiben und dabei
- die Rücksprungadresse so setzen, dass der eingeschleuste Code aktiviert wird,
- der eventuell weiteren Code nachziehen kann.

Effekt:

- Ursprünglicher Dienst ist mit **Schadcode** des Angreifers infiziert,
- d.h. der Schadcode läuft mit der Autorisierung des Angriffenen. Falls das (bei Unix) der Super-User **root** ist, ist das System **unterwandert (compromised)**.

Gegenmaßnahmen:

- Sichere Programmiersprachen!
- Keller-Segment ohne "execute"-Rechte
(Solaris; OpenBSD 3.2; Linux: OpenWall, PaX, ...)
(verhindert allerdings nicht den Pufferüberlauf selbst)
- Andere Keller-Organisation, Hüllen für die unsichereren
Funktionen, alle möglichen Tricks ...
- restriktive Autorisierung der bedrohten Prozesse
- restriktive Öffnung von Ports
- Einsatz von Firewalls
- Nicht vergessen: *patches einspielen bzw. sicherheits-relevante Updates beachten!*

Zusammenfassung

- Sockets tragen Portnummern und werden vom Betriebssystem für das Senden und Empfangen von Nachrichten bereitgestellt.
- Die über Sockets vermittelte Interprozesskommunikation kann sowohl lokal als auch über Stationsgrenzen hinweg erfolgen.
- TCP ist eine Plattform für Dienstorientierung im Internet.
- UDP ist für manche Dienste ebenfalls einsetzbar.
- UDP Multicast ist für unzuverlässige Rundrufe geeignet, auch mit Antworten.
- Un/sicherheit !

Quellen

Internet-Protokolle: *IETF RFCs.* www.ietf.org/rfc.html

Multicast: tldp.org/HOWTO/Multicast-HOWTO.html

W.R. Stevens et al.: *Unix Network Programming, vol. 1: Sockets Networking.*
ISBN 0-13-141155-1, Addison-Wesley 2005 (3. ed.)

S. Walton: *Linux Socket Programming.* Pearson 2001

E.R. Harold: *Java Network Programming.* O'Reilly 2004 (3. ed.)

A. Lockhart: *Network Security Hacks.* O'Reilly 2006

Anhang: Sockets mit C

(siehe auch [Stevens et al. 2005])

- IP-Adressen und Host-Namen
- TCP Sockets
- Datenrepräsentation

IP-Adressen und Host-Namen

```
#include <netdb.h>
struct hostent *gethostbyname(char *hostname)
    liefert den Verweis auf einen Verbund, der u.a. die
    IP-Adresse(n) des genannten Rechners enthält
struct hostent *gethostbyaddr
    (char *addr, int len, int type)
    arbeitet umgekehrt
    (DNS wird befragt - oder /etc/hosts oder ... )
```

```
#include <netinet/in.h>
unsigned long inet_addr(char *ptr)
    wandelt eine IP-Adresse in die 32-Bit-Darstellung
char *inet_ntoa(struct in_addr addr)
    arbeitet umgekehrt
```

TCP Sockets

Erzeugung eines Socket:

```
#include <sys/socket.h>
int socket(int family, int type, int protocol)
```

↑ ↑ ↑ ↑
 prozesslokale Protokoll- Dienst: Protokoll:
 Socket- Familie: TCP 0
 Nummer AF_INET

Netzweit gültige Benennung eines Socket

- mit Reservierung eines freien Ports:

```
int bind(int socket,
          struct sockaddr *address, ← wählt Portnummer
          int addrlen)                                und IP-Adresse
```

Festlegung der akzeptierten Warteschlangenlänge:

```
int listen(int socket, int backlog)
```

Annahme des Verbindungswunschs eines Klienten:

```
int accept(int socket,  
          struct sockaddr *client,  
          int *addrulen)
```



neuer Socket

□ Was tut der Klient?

- Socket erzeugen: wie auf S. 54
- (wahlweise:) Socket benennen: wie auf S. 54;
falls die Benennung unterbleibt, wählt das System
im nächsten Schritt irgendeine freie Portnummer.
- Verbindungswunsch an den Server richten:

```
#include <sys/socket.h>
int connect(int socket,
            struct sockaddr *server,
            int addrlen);
```

- Nach Rückkehr von `connect` Benutzung der Verbindung auf beiden Seiten mit `read/write` auf den jeweiligen Sockets
- Abbau der Verbindung mit `int close(int socket)` (einseitig oder beidseitig)

- **Datenrepräsentation** nicht vergessen bei unterschiedlicher Rechner-Hardware - auch wenn nur Zeichen übertragen werden (Byte-Ordnung!)
- Typische Hilfsfunktionen dafür: *byte ordering routines*

```
#include <netinet/in.h>
u_long htonl(u_long hostlong)
u_short htons(u_short hostshort)
u_long ntohl(u_long netlong)
u_short ntohs(u_short netshort)
```

- (Bei Java bleibt uns dies verborgen!)