LV Kryptologie WS06/07,
HTWK Leipzig

Matthias Jauernig
12.12.06

# SSH
# Die „Secure Shell“

# Inhalt

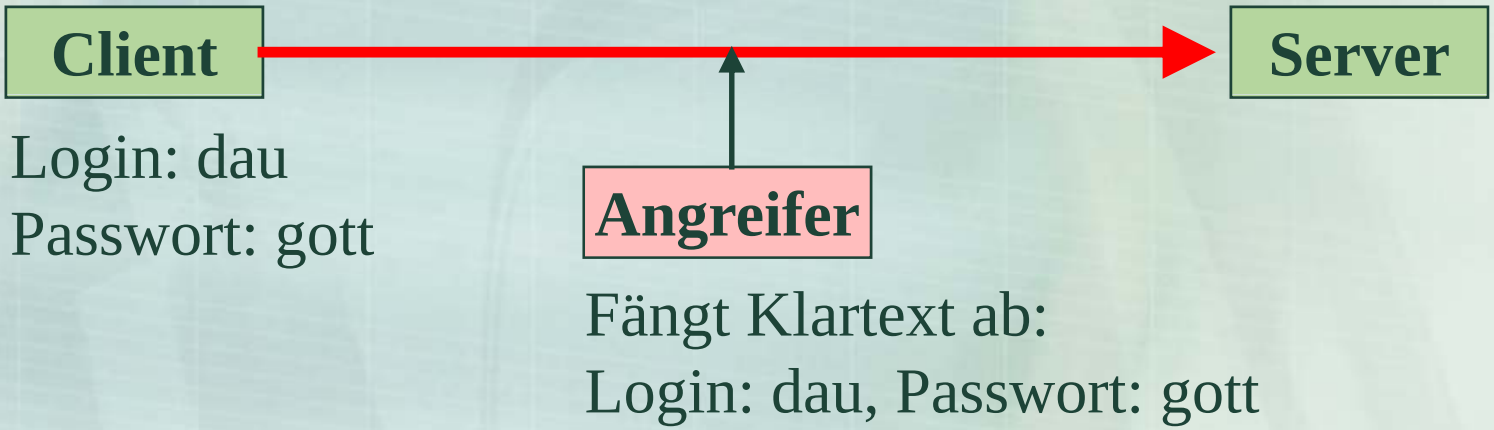
1. Motivation
2. Historie
3. Funktionsweise von SSH-2
4. Das OpenSSH Programmpaket
5. Schlussbemerkungen, Links

# (1) Motivation

# Motivation

- Wunsch: Remote-Login, Programme ausführen, Dateien transferieren
- Klassischer Ansatz: „R-Tools“ (rlogin, rsh, rcp) bzw. telnet und ftp

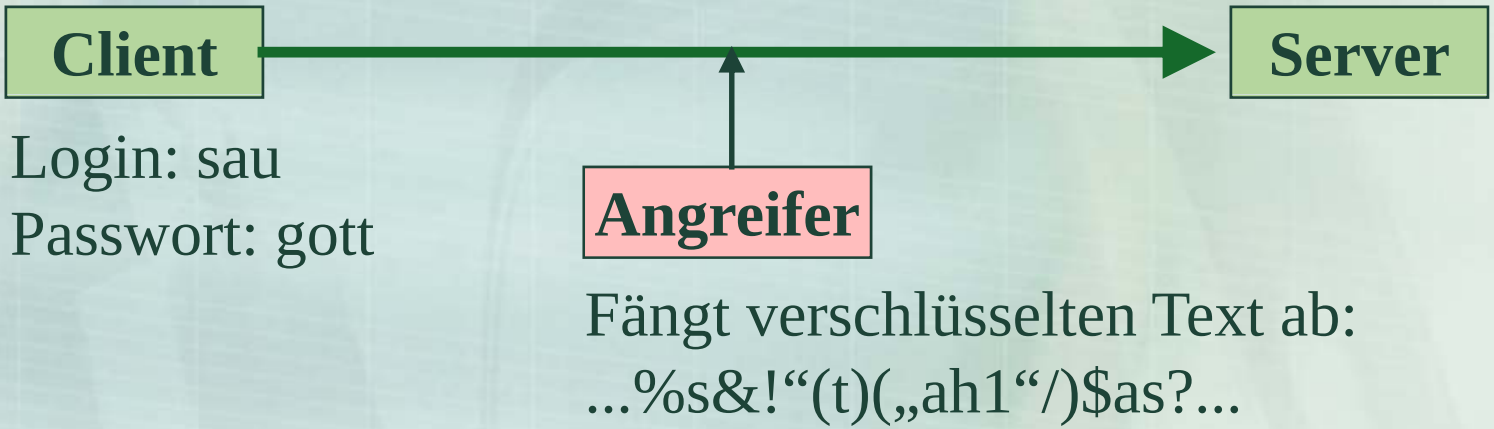
  → **unverschlüsselte Übertragung!**



# Motivation

- Unkalkulierbares Sicherheitsrisiko!: Offenlegung von Geheimnissen, Manipulation von Daten, Einbruch in Systeme, …
- Ausweg: Benutzung von **ssh (secure shell)**

  → **verschlüsselte Übertragung!**



# (2) Historie

# Historie

- **1995**: Entwicklung von SSH1 als Freeware durch Tatu Ylönen an der TU Helsinki; Reaktion auf Passwortdiebstähle
- **Dezember 1995**: Gründung der Firma „SSH Communications Security“, Kommerzialisierung
- **1998**: SSH-2 veröffentlicht, inkompatibel zu SSH-1

# Historie

- **1999**: Entwicklung von OpenSSH als freie Implementierung von SSH2
- **2005**: kommerzielles SSH G3 veröffentlicht, von Kryptoexperten verpöhnt (security through obscurity)
- **2006**: SSH-2 von IETF als Internetstandard vorgeschlagen

# (3) Funktionsweise von SSH-2

# Funktionsweise von SSH-2

- **Sicherheitsprinzipien** von SSH:
  1. Authentifizierung
  2. Verschlüsselung
  3. Integrität

# Funktionsweise von SSH-2

- Rückblick → 1995: **SSH-1**:
  - Monolithische Architektur, alle Funktionen in einem Protokoll vereint
  - Nur Verschlüsselungsalgorithmen sind verhandelbar
  - Neue Algorithmen kaum integrierbar
  - Sicherheitslücke: CRC32 zur Integritätsprüfung, erlaubt Insertion-Angriff und Ausspähen von Verbindungen (1998 publiziert)

# Funktionsweise von SSH-2

- 1998: **SSH-2**:
  - Modularer Aufbau des Protokolls
  - 3 Schichten: Transport-, Verbindungs-, Authentifizierungsprotokoll
  - Implementierung und Verwendung weiterer Verschlüsselungsalgorithmen und Authentifizierungsmethoden möglich
  - Integritätsprüfung mit MAC-Algorithmen (msg auth code, hmac-sha1 von RFC gefordert)
  - Gilt derzeit als sicher

# Funktionsweise von SSH-2

- **Schichtenmodell:**

| Anwendungssoftware (ssh, sftp, scp etc.) | |
|---|---|
| **SSH User Authentication Protocol (SSH-USERAUTH)**<br>• Client-Authentifizierung<br>- Passwort<br>- Public Key<br>- Host based | **SSH Connection Protocol (SSH-CONNECT)**<br>• Ausführung entfernter Programme<br>• Unterstützung interaktiver Sessions<br>• TCP-Port- und X11-Weiterleitung<br>• Datenkomprimierung |
| **SSH Transport Protocol (SSH-TRANS)**<br>• Aushandlung von Algorithmen<br>• Austausch von Sitzungsschlüsseln<br>• Server-Authentifizierung<br>• Datenintegrität | |
| TCP-Protokoll | |

# Authentifizierung

- **Server-Authentifizierung:**
  - Identifizierung über RSA-Zertifikat (auch zentrale Zertifizierungsstelle denkbar, vgl. SSL)
  - Bei 1. Verbindung speichert Client den öffentlichen Server-Key in ~/.ssh/known_hosts
  - Prüfung des Keys auf Übereinstimmung bei jedem neuen Login auf dem Server
  - Methode: diffie-hellman-group1-sha1, Server signiert bestimmten Hash mit private Key

# Authentifizierung

- **Client-Authentifizierung:**
  - Passwort
  - Public Key
  - Host Based
  - Beliebige weitere, z.B. Kerberos, Smart Card, SecurID etc.

# Authentifizierung

- Client-Authentifizierung:
  - Zu **Public Key**:
    - RSA oder DSA Zertifikate
    - Client besitzt Schlüsselpaar in Dateien ~/.ssh/id_rsa und ~/.ssh/id_rsa.pub (Erzeugung mit ssh-keygen)
    - Server hält öffentliche Client-Schlüssel in ~/.ssh/authorized_keys
    - Client erzeugt Signatur (mehrere Werte) mit seinem privaten Schlüssel, Server verifiziert diese

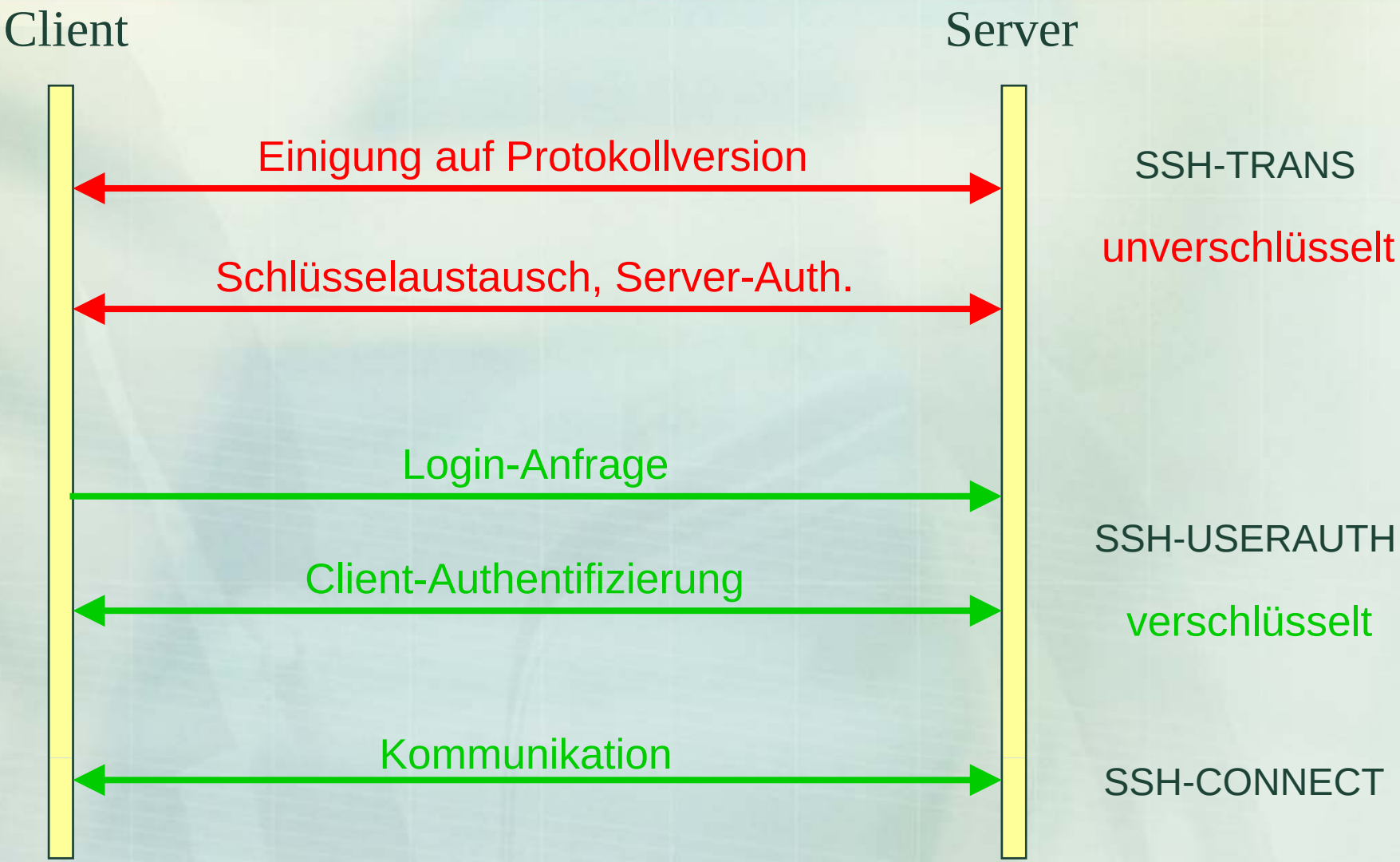
# Verschlüsselung

- **Algorithmen:** 3DES (default), DES, Twofish, Blowfish, AES, Serpent, ARCFOUR, IDEA, CAST-128 oder selbst definierte
- **Session Keys:**
  - Für jede SSH-Sitzung wird neuer Session Key generiert → geschieht zusammen mit Server-Authentifizierung nach diffie-hellman-group1-sha1
  - Session Key wird nie übertragen, sondern auf Client und Server separat berechnet
  - Erneuter Schlüsselaustausch nach $2^{32}$ Bytes empfohlen

# Sicherheit von SSH-2

- SSH-2 hilft beim **Schutz vor**:
  - IP spoofing
  - IP source routing
  - DNS spoofing
  - Abfangen von Login-Daten und Nutzdaten
  - Manipulation von Daten durch man-in-the-middle-Attacken
- ***Abgrenzung***:
  - Kein Allheilmittel zur IT-Sicherheit, kann nur Bestandteil eines integrierten Konzepts sein

# Kommunikation in SSH-2



# (4) Das OpenSSH-Programmpaket

# OpenSSH - Allgemein

- Seit 1999 entwickelt
- Implementierung des SSH-2 Protokolls
- Open-Source
- Für viele Plattformen und Betriebssysteme verfügbar

# OpenSSH - Dateien

- **Konfigurationsdateien**:
  - Client:
    - /etc/ssh/ssh_config
    - ~/.ssh/config
  - Server:
    - /etc/ssh/sshd_config

# OpenSSH - Funktionen

- Umfangreiche **Funktionalität**:
  - Remote Login (**`ssh`** und **`sshd`**)
  - Dateitransfer (**`scp`**, **`sftp`**)
  - X11 Weiterleitungen (**`ssh -X`**)
  - Weiterleitung beliebiger TCP-Verbindungen (**`ssh -L`** bzw. **`ssh -R`**)

# OpenSSH - Basisbefehle

- Einfache Remote Logins:

```
ssh host | ssh -l user host | ssh user@host
```

- Direkte Befehlsausführung:

```
ssh user@host befehl
```

- Dateitransfers:

```
scp lokaldatei user@host:/pfad/remotedatei
sftp user@host
```

- Komprimierung einschalten (bis 50% weniger Last):

```
ssh -C ...
```

- Verschlüsselung auswählen:

```
ssh -c blowfish ...
```

# OpenSSH - Public-Key Auth.

- Befehle auf Server ohne Passworteingabe ausführen (über **Public Key** Auth.):
  1. Erzeugung eines SSH-Schlüsselpaars zur Client-Authentifizierung (public-key):

     **`ssh-keygen –t rsa|dsa|rsa1`**

     → Dateien: ~/.ssh/id_rsa und ~/.ssh/id_rsa.pub
  2. Public key auf Server kopieren (in Datei ~/.ssh/authorized_keys):

     **`ssh-copy-id –i ~/.ssh/id_rsa.pub user@host`**

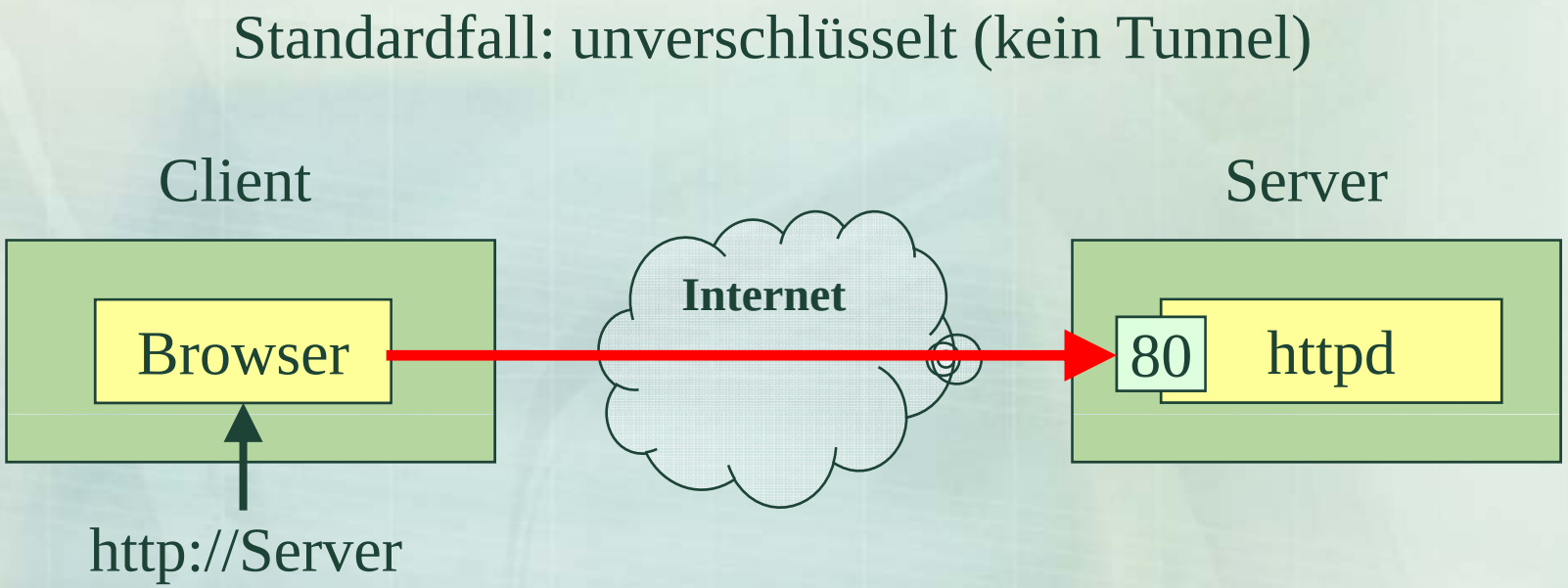
➔ Verwaltung von Public Keys auf Client mittels **`ssh-agent`** und **`ssh-add`** möglich

# OpenSSH - Tunneling

- Weiterleitung beliebiger TCP-Verbindungen über einen sicheren **SSH-Tunnel**
- Vorteile:
  - Umgehen von Firewalls
  - Verschlüsselung traditionell unsicherer Protokolle, ohne deren Anwendungen zu ändern
  - Komprimierung des Datenstroms möglich

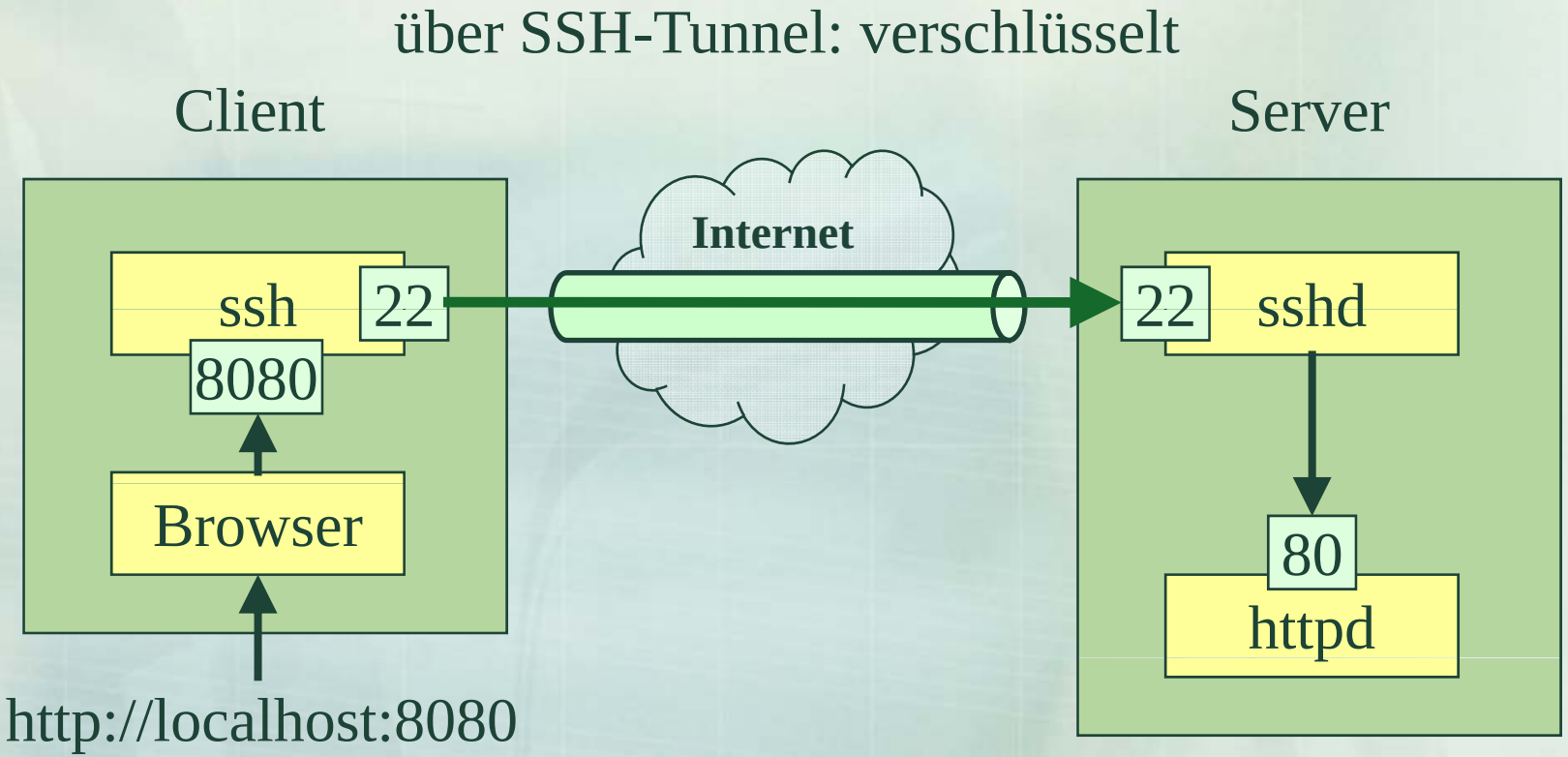
# OpenSSH - Tunneling

- **Beispiel**: HTTP tunneln (local forwarding -L)



# OpenSSH - Tunneling

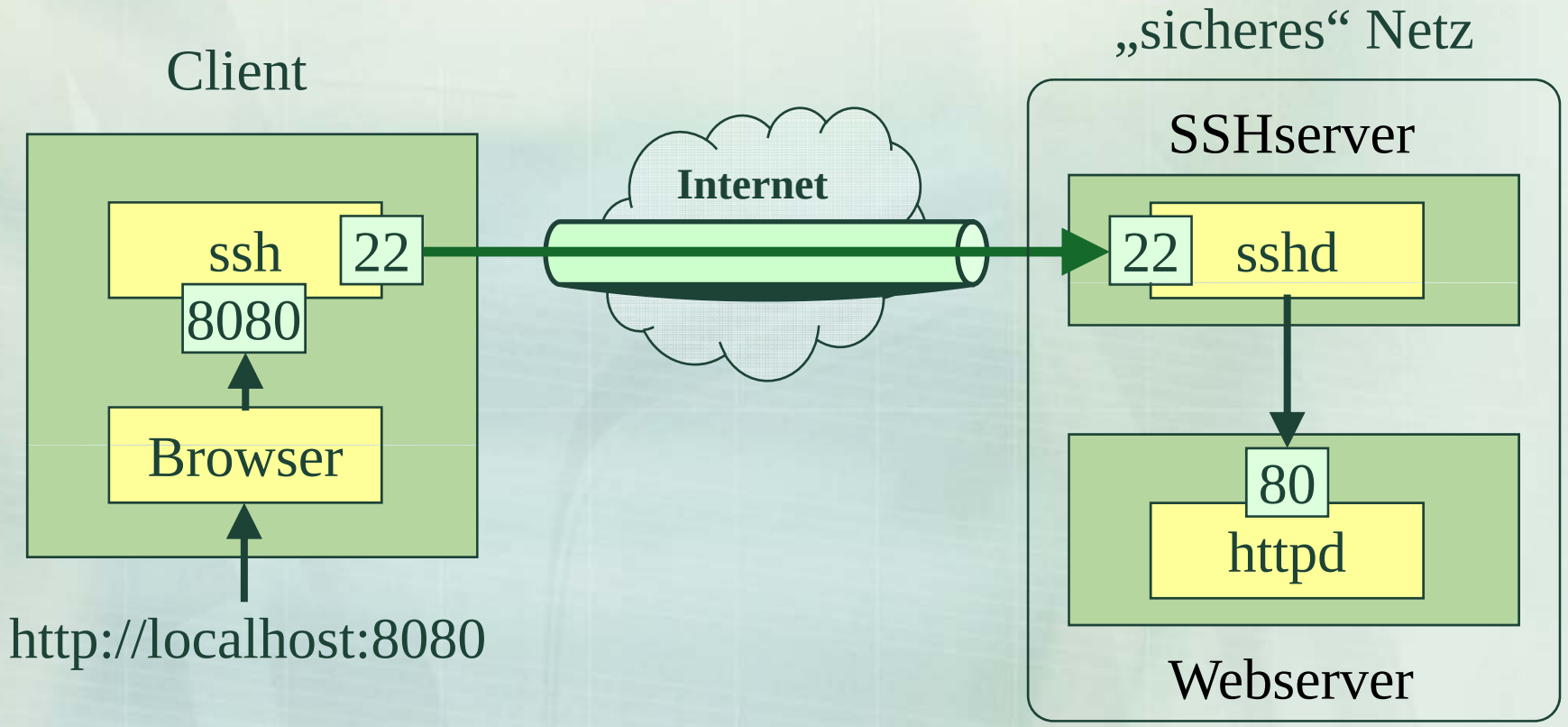
- **`ssh -L 8080:Server:80 user@Server`** oder
- **`ssh -L 8080:localhost:80 user@Server`**



# OpenSSH - Tunneling

- allgemeiner:

```
ssh -L 8080:Webserver:80 user@SSHserver
```



# (5) Schluss-bemerkungen, Links

# Links

- http://de.wikipedia.org/wiki/Ssh
- http://de.wikipedia.org/wiki/OpenSSH
- http://www.openssh.com
- RFC 4250-4254

# Schlussbemerkungen

- SSH ist kein Sicherheits-Allheilmittel
- Gilt jedoch als sicher und ist vielfältig einsetzbar:
  - Remote – Systemadministration
  - Sichere Dateitransfers
  - Sichere TCP-Verbindungen
- Ist in Version 2 unersetzbar für Systemadministratoren

# Ende