Der Internet-Dienst E-Mail

- Auf UNIX-Rechnern mit (klassischem) Mail-Server hat jeder User ein privates Postfach (*Mailbox*): `/var/mail/`$User$
- meist ein Mail-Server pro Domain
- Empfang von E-Mail unter der Postadresse $Benutzername$`@`$Domainname$
  z.B.: `wolfgang@cs.uni-potsdam.de`
- Der Benutzername muss nicht der Login-Name sein. $\rightsquigarrow$ `/etc/mail/aliases` oder Namensdienst
- Der Mail-Server kann die E-Mails an die privaten Postfächer weitergeben.
- Anwender benutzt einen $Mail\ User\ Agent\ (MUA)$ zum Lesen, Verfassen und Versenden von E-Mail (z.B. `pine`, `xemacs`, `mailtool`).

Dienste des Mail-Servers

- **Mail Transfer Agent (MTA)**, z.B. `sendmail`
  Weiterleitung der E-Mails, die von MUAs abgesendet werden
  (Protokoll: SMTP)

  – Erstellen des Briefumschlags (*Envelope*): Sender, Empänger
  – Erstellen des *Headers*: `From, To, Cc, Subject, Received, ...`
  – Versenden an nächsten Mail-Knotenpunkt (im Klartext!)

  Sender- und Empfängerangaben im Envelope können von den wirklichen Adressen abweichen, z.B. bei

  – Mailverteilern,
  – Spam.

- **Mailfilter**
  Spam-Filter, Blacklists, Whitelists, Schutz vor Viren
- **Mail Delivery Agent (MDA)**, z.B. `procmail`
  Verteilung auf die privaten Postfächer
- **Mail Retrieval Agent (MRA)**
  Ermöglicht Herunterladen der Emails durch MUAs.
  (Protokolle: POP3, IMAP)

  IMAP erlaubt Verwaltung der E-Mails auf dem Server $\rightsquigarrow$ Webmailer

  $\rightsquigarrow$ ständige Verbindung zum Server nötig!

Multipart-Mails

- E-Mails mit Anhängen (Attachments) werden als *eine* Datei übertragen.
- Die Teile dieser Datei müssen in standardisierter Form voneinander getrennt sein, z.B.
  - Text der Email,
  - angehängte ZIP-Datei.
- Dieser Standard teilt der interpretierenden Software mit, um welchen Datentyp es sich beim nächsten Teil der Datei handelt: MIME-Typen
  z.B.: `text/plain`, `application/zip`, `image/jpeg`
- MIME: **M**ultipurpose **I**nternet **M**ail **E**xtensions
- Wird auch bei vielen anderen Anwendungen verwendet, wenn entfernte Prozesse miteinander kommunizieren (z.B. Web-Browser und Web-Server).

WWW — World Wide Web

- Jeder Benutzer kann allen Benutzern Informationen anbieten.
- Client-Server-Kommunikation
- **WWW-Client:** Internet-Browser
  - Kommunikation mit dem Benutzer
  - Kommunikation mit dem Server (Protokoll: HTTP)
    1. TCP-Verbindung zum Web-Server aufbauen
    2. Anfrage an Server abschicken
    3. Antwort vom Server empfangen
    4. Verbindung schließen
  - Interpretation von HTML-Code $\rightsquigarrow$ Darstellung ist browser-abhängig
  - Ansteuerung weiterer Seiten über *Hyperlinks*

- **WWW-Server:**
  - Erwartet Anfragen von WWW-Clients (Standard: am Port 80)
  - sucht und sendet angeforderte Dokumente ($Ressourcen$) an Clients
    - HTML-Dokumente (definieren Web-Seiten)
    - Dateien zum Download oder Einbinden in Web-Seiten
    - Skripte zur Benutzerkommunikation
    - im Browser ausführbarer Code
  - Zugriffskontrolle, Datenverschlüsselung, Fehlermeldungen
- freie Server:
  - Apache (`http://httpd.apache.org`)
  - Jigsaw (`http://www.w3.org/Jigsaw`)

Adressierung von Ressourcen

- meist durch URL (**U**niform **R**esource **L**ocator)
- *protokoll*`://[`*user*`:`*passwort*`@]hostname[:`*port*`]/pfad/datei[#`*marke*`]`

  `http://www.cs.uni-potsdam.de`
  `http://samuel.cs.uni-potsdam.de/index.html`
  `http://www.cs.uni-potsdam.de/ml/teaching/ws08/rnb/f09.pdf`
  `ftp://ftp.uni-potsdam.de`
  `file:///home/wolfgang/Lehre/RNB/HTML/beispiel.html`
- Protokoll `http`: **H**yper**t**ext **T**ransfer **P**rotocol
- Dateiformat `.html` (oder `.htm`): **H**yper**t**ext **M**arkup **L**anguage

Struktur von HTML-Dokumenten

- Strukturierung des Textes durch „Tags“ der Form `<tagname>`
- Viele Tags treten paarweise auf:

  `<tagname>` *Wirkungsbereich* `</tagname>`
- „Beginn-Tags“ können attributiert sein:

  `<tagname attribut="wert" ... attribut="wert">`
- Es gibt eine feste Liste erlaubter Tags.

  ($\rightsquigarrow$ XML, XHTML flexibler)
- `http://www.selfhtml.org`

Beispiel

```
<html>
<head>
    <title> Ein Beispieldokument</title>
</head>
<body>
    <h1>Eine &Uuml;berschrift</h1>

    <p>Ein Absatz.
    </p>
</body>
</html>
```

Einige HTML-Tags

Tag	Bedeutung
`<html> ... </html>`	Beginn/Ende HTML-Code
`<head> ... </head>`	Beginn/Ende Kopfdaten
`<title> ... </title>`	Beginn/Ende Titel der Webseite
`<body> ... </body>`	Beginn/Ende Inhalt der Webseite
`<p> ... </p>`	Beginn/Ende Absatz
`<h`n`> ... </h`n`>`	Beginn/Ende Überschrift der Kategorie n (1 bis 6)
` `	Zeilenumbruch
`<hr>`	horizontale Linie
`<i> ... </i>`	Beginn/Ende kursiver Text
`<b> ... </b>`	Beginn/Ende fettgedruckter Text
`<!-- ... -->`	Beginn/Ende Kommentar

Cascading Style Sheet (CSS)

- CSS dient der dokumentweiten Definition von Layouts
- Formatierung der HTML-Tags wird neu vereinbart
  $\rightsquigarrow$ Verzicht auf Attribute in den Tags möglich
- CSS-Angaben für ein Dokument im Kopfbereich mittels des `<style>`-Tags:

  ```
  <style type="text/css">          <!-- Attribut: MIME-Type -->
  selektor  { attribut: Wert; ... attribut: Wert; }
  </style>
  ```

- CSS-Angaben können in seperater `.css`-Datei definiert werden:

  ```
  <link rel="STYLESHEET" type="text/css" href="datei.css">
  ```

Hyperlinks

- Verbindung von einem HTML-Dokument zu einem anderen
- `<a href="`$Ressource$`">`$Text$`</a>`
  $\rightsquigarrow$ Hyperlink auf $Ressource$, wobei $Text$ im Broser angezeigt wird
- Maus-Klick auf den Hyperlink-$Text$ lädt die $Ressource$
- Absolute oder relative Pfadangaben von Dateien des lokalen Systems sind als $Ressource$ erlaubt.

Formulare und Skripte

- zum Abfragen von Informationen vom Benutzer
- Formular ist z.B. CGI-Skript (**C**ommon **G**ateway **I**nterface):
  ausführbares Programm (meist Perl), vom Web-Server auf Anforderung
  gestartet
- *Andere* Skripte werden mit der HTML-Seite übertragen und im Browser ausgeführt.
  - Javascript
  - Java
- PHP (*PHP: Hypertext Preprocessor*) wird serverseitig ausgeführt und die Ausgabe (z.B. `html`- oder `pdf`-Dokumente) in die Webseite eingebaut, die erst dann an den Browser übertragen wird.

- Anforderung eines CGI-Skripts mittels HTML-Tag `<form>`, z.B.:

```
<form method="post" action="http://www.example.com/cgi-bin/skript.pl">
<!-- Formularelemente (Text- und Auswahlfelder etc.) --->
<input type="submit" value="Absenden">
</form>
```

  - `method`: Sendemethode
  - `action`: URL des CGI-Skripts, das die Daten verarbeiten soll
  - Sicherheit: Browser lassen oft nur Skripte im Verzeichnis `cgi-bin` des Web-Servers zu.
  - `<input>`: Button „Absenden“
- CGI-Ausgabe wird an Client zurückgeschickt.

Cookies

- ermöglichen das Speichern von Daten auf der Client-Seite (Browser)
- Browser speichert Cookies und sendet sie auf Anfrage an den Web-Server zurück
- Verwendung als statische Variablen, die ihren Inhalt nicht verlieren und für jeden Server verschieden sind
- relativ kleine Sicherheitslücke, da kein ausführbarer Code und kein Zugriff auf Daten des Rechners möglich ist
- Es können aber Daten über das Surfverhalten gesammelt werden.
  („Wann wurde diese Seite zuletzt besucht?“ etc.)

Sicherheit im Netz

- Problem: Viele Protokolle übertragen Daten (auch Passwörter) im Klartext.
  - SMTP
  - HTTP
  - TELNET
  - FTP
- Unbefugter Zugang zu fremden Systemen und fremden Daten möglich.
- Lösungsansatz: Einmal-Passwörter, Kryptographie

Mögliche Angriffe

- **Packet Sniffing**: Auslesen von in der Netz-Topologie vorbekommenden Datenpaketen mit der Netzwerkkarte
  $\rightsquigarrow$ Einmal-Passwörter
- **Hijacking**: Angreifer ersetzt einen der Teilnehmer bei laufender Verbindung (also nach Passwort-Eingabe)
- **DNS-Spoofing**: Manipulation eines Nameservers; Nachahmung des Login-Vorgangs anstelle des angewählten Servers
  $\rightsquigarrow$ zunächst falsches Passwort eingeben
- **Man-in-the-middle-Attack**: wie DNS-Spoofing, aber Angreifer leitet Daten unverändert an echten Teilnehmer weiter

Kryptographische Verschlüsselungen

1. Was kann man erreichen?
   - Schutz von Daten vor nicht-autorisiertem Zugriff
   - Schutz von Daten bei der Übertragung zwischen zwei Hosts
   - Erkennen von Datenmanipulationen
   - Verifikation des Autors von Nachrichten (z.B. am Schlüssel)
2. ... und was nicht?
   - Verhindern, dass ein Angreifer Daten löscht oder verändert
   - Verhindern, dass ein Angreifer das benutzte Programm modifiziert
   - Verhindern, dass ein Angreifer eine neue Entschlüsselungsmethode findet

Arten von Verschlüsselungsalgorithmen

- **symmetrisch**: Sender und Empfänger benutzen denselben Schlüssel
  - mono- oder polyalphabetische Verschlüsselung
  - UNIX-Programm `crypt` (basiert auf Enigma-Maschine)
  - DES/3DES/AES (**D**ata/**A**dvanced **E**ncryption **S**tandard), z.B. EC-PIN
  - RC2/4/5/6 (Blockchiffre von Ronald Rivest), genutzt z.B. bei Lotus, Netscape, Oracle, Microsoft, Adobe
- **asymmetrisch**: Jeder Teilnehmer hat ein eigenes Schlüsselpaar:
  - Verschlüsseln mit „öffentlichem Schlüssel“ (*public key*),
  - Entschlüsseln mit zugehörigem „privaten Schlüssel“ (*private key*)
  - Beispiel: RSA-Algorithmus nach (R. Rivest, A. Shamir, L. Adleman)
  - ca. 1000 mal langsamer als symmetrische Verfahren

Verschlüsselung bei SSH

Jeder Server besitzt zwei Schlüsselpaare:

1. **Host-Key-Paar**: unveränderliches RSA-Schlüsselpaar

   public host key - Verteilung an alle Kommunikationspartner
   - dient als „Ausweis“ gegenüber den anderen Hosts und zur Verschlüsselung durch die Clients

   private host key: - bleibt geheim, geht nie über das Netz
   - dient zum Entschlüsseln

2. **Server-Key-Paar**: i.a. stündlich vom Server-Prozess neu erzeugtes RSA-Schlüsselpaar, wird nur im Primärspeicher gehalten (Schutz bei Hacking)

   public server key - Verteilung an alle Kommunikationspartner
   - dient zur zusätzlichen Verschlüsselung durch die Clients

   private server key: - bleibt geheim, geht nie über das Netz
   - dient zum Entschlüsseln

Verbindungsaufbau bei SSH

- Server sendet sowohl public host key als auch public server key an Client
- Client überprüft, ob er den public host key kennt
  $\rightsquigarrow$ ggf. Warnung, Eintrag in `~/.ssh/known_hosts` (falls vom Nutzer bestätigt)
- Prüfen der Identität durch Senden von Testnachrichten
  $\rightsquigarrow$ Kann der Server diese entschlüsseln?
- Client generiert Zufallszahl als *session key* und sendet diese, mit beiden public keys des Servers verschlüsselt, an den Server
  $\rightsquigarrow$ von nun an symmetrische Verschlüsselung
- Vorbereitung der Sitzung (Terminaltyp, Shelltyp etc.), Start einer Shell

HTTP over SSL (HTTPS)

- SSL: **S**ecure **S**ocket **L**ayer
- URL mit Protokoll `https://...`: Anforderung eines Zertifikats vom Server
- Zertifikat geben Zertifizierungsstellen (z.B. VeriSign) heraus
- Server sendet Zertifikat an Client (Browser), Zertifizierungsstelle bestätigt Gültigkeit des Zertifikats ($\rightsquigarrow$ Identitätsprüfung)
- „Vorhängeschloss“ des Browsers wird geschlossen
- Einrichten der Verschlüsselung ähnlich wie bei SSH