Netzprogrammierung
HTTP Kommunikation

Prof. Dr.-Ing. Robert Tolksdorf
Freie Universität Berlin
Institut für Informatik
Netzbasierte Informationssysteme
mailto: tolk@inf.fu-berlin.de
http://www.robert-tolksdorf.de

Überblick

1. Authentifizierung in HTTP
2. Anfragen in HTTP
3. Zustand in Web Anwendungen

HTTP Protokoll

Client

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/3.04Gold (Win95; I)
Host: megababe.isdn:80
Accept: image/gif, image/jpeg, image/pjpeg, */*
```

```
HTTP/1.0 200 OK
Last-Modified: Sun, 15 Mar 1998 11:26:50 GMT
MIME-Version: 1.0
Date: Fri, 20 Mar 1998 16:43:11 GMT
Server: Roxen-Challenger/1.2beta1
Content-type: text/html
Content-length: 2990

<HTML><HEAD><TITLE>TU Berlin ---
```

Server

Authentifizierung in HTTP

Interaktion zur Authentifizierung

- Seiten im Web können Zugriffsschutz tragen
- Interaktion zum Abruf
  - Normales GET
  - Antwort 401 und WWW-Authenticate: Header, der Nachweis in unterschiedlichen Schemata anfordert
  - Weiteres GET mit Authorization: Header, der je nach Schema Parameter trägt
  - Antwort 200

Zugangsschutz auf Web-Servern

- Beispiel am Apache Server zum Schutz von
  http://www.inf.fu-berlin.de/inst/ag-nbi/lehre/0304/V_NP/geheim/index.html
- .htaccess:

  AuthUserFile /import/htdocs/inst/ag-nbi/lehre/0304/V_NP/geheim/.htpasswd
  AuthGroupFile /dev/null
  AuthName "Zugang zur geheimen Seite"
  AuthType Basic
  require user Nutzer
- .htpasswd:

  Nutzer:ALwPfINObhEus
  - erzeugt mit /usr/apache/bin/htpasswd -c .htpasswd Nutzer
  - Passwort ist ganzgeheim

ReadProtectedURL/1

```
import java.net.*;
import java.io.*;
public class ReadProtectedURL {
  public static void main(String[] argv) {
    int response=200;
    try {
      // Normal verbinden
      URL page=new URL(argv[0]);
      URLConnection connection=page.openConnection();
      connection.connect();
      System.out.println("Length: "+connection.getContentLength());
      System.out.println("Typ: "+connection.getContentType());
      // Ist es eine HTTP Verbindung?
      if (connection instanceof java.net.HttpURLConnection) {
      // Ja, Response-Code erfragen
      response=((java.net.HttpURLConnection)connection).getResponseCode();
      System.out.println("HTTP Result: "+ response);
```


ReadProtectedURL/2

```
      if (response==401) {   // Ist es ein 401?
        // Herausforderung ermitteln
        String realm=connection.getHeaderField("WWW-Authenticate");
        System.out.println("Realm: "+ realm );
        // Realm: Basic realm="Zugang zur geheimen Seite"
        // Hier eigentlich: Komplexe Behandlung der Herausforderung
        realm=realm.substring(12);
        // Neu anforden aber mit Authorization Header
        URL pageAuth = new URL(argv[0]);
        connection=pageAuth.openConnection();
        byte[] userPass="Nutzer:ganzgeheim".getBytes();
        // in Base64 Kodierung
        String b64UserPass=new sun.misc.BASE64Encoder().encode(userPass);
        connection.setRequestProperty("Authorization", "Basic "+ b64UserPass);
        connection.connect();
      }
    }
```

ReadProtectedURL/3

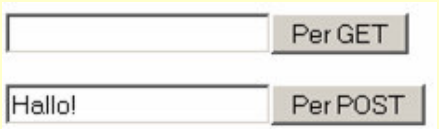
```
    BufferedReader br = new BufferedReader(
                    new InputStreamReader(connection.getInputStream()));
    while (true) {
     String l =br.readLine();
     if (l==null) {
        break;
     } else {
        System.out.println(l);
     }
    }
   } catch (Exception e) {
    System.err.println(e.getMessage());
    return;
   }
  }
}
```


Anfragen in HTTP

Parameter für Web-Skripte

- Zwei Arten der Übermittlung von Parametern an Skripte:
  - GET: Daten werden in URL kodiert
  - POST: Daten werden kodiert über Standardeingabe geliefert
- HTML:

```
<html><body>
<form action="http://flp.cs.tu-berlin.de/~tolk/echo.cgi" method="get">
   <input name="Eingabe" type="text">
   <input type="submit" value="Per GET">
</form>
<form action="http://flp.cs.tu-berlin.de/~tolk/echo.cgi" method="post">
  <input name="Eingabe" type="text">
   <input type="submit" value="Per POST">
</form>
</body></html>
```



Echo Skript

- Serverseitig:

```
#!/usr/local/bin/perl -w
use strict;
use CGI;
my $cgi = new CGI; // Dekodiert Parameter je nach Methode
print $cgi->header(),
      $cgi->start_html('Echo'),
      $cgi->h1('Echo der Eingabe:'),
      $cgi->pre($cgi->param('Eingabe')),
      $cgi->end_pre(), end_html();
```

Kodierung von Eingabewerten

- Parameter haben bestimmten Übergabeformat
  - $Name_1$=$Wert_1$&$Name_2$=$Wert_2$
- Dieser *Query String* wird
  - bei GET an die URL mit ? getrennt angehängt
    - http://flp.cs.tu-berlin.de/~tolk/echo.cgi?Eingabe=Hallo!
  - bei POST als Inhalt übermittelt und beim Web-Server über stdin einem Skript übergeben
- Query String selber muss kodiert werden
  - Um Transport zu sichern
  - Um bedeutungstragende Zeichen (=, & etc.) übertragen zu können
  - Medientyp der Nachricht ist application/x-www-form-urlencoded

Kodierung von Werten

- Namen und Werte anpassen
  - Leerzeichen -> +
  - Reservierte Zeichen -> %HH
    - CR LF -> %0D%0A
    - & -> %26
    - %&" "+#äß -> %25%26%22+%22%2B%23%E4%DF
- Name-Wert Paare gruppieren
  - $Name_1$=$Wert_1$
- Parameter zu Query-String gruppieren
  - $Name_1$=$Wert_1$&$Name_2$=$Wert_2$
- Dekodierung entsprechend

URLEncoding in Java

- java.net.URLEncoder:
  - public static String encode(String s, String enc) throws UnsupportedEncodingException
  - enc ist registrierte Zeichensatzbenennung wie "ISO-8859-1"
- java.net.URLDecoder:
  - public static String decode(String s, String enc) throws UnsupportedEncodingException

Beispiel: PostURL

```
import java.net.*;
import java.io.*;
public class PostURL {
  public static void main(String[] argv) throws IOException {
      URL url = new URL(argv[0]);
   HttpURLConnection http =  (HttpURLConnection)url.openConnection();
   http.setDoOutput(true);
   PrintWriter out = new PrintWriter(http.getOutputStream());
   out.println(URLEncoder.encode("Eingabe","ISO-8859-1") + "=" +
               URLEncoder.encode("%&\" \"+#äß","ISO-8859-1"));
   out.close();
   BufferedReader in = new BufferedReader(new
      InputStreamReader(http.getInputStream()));
   String line;
   while ((line = in.readLine()) != null)
      System.out.println(line);
   in.close();
  }
}
```

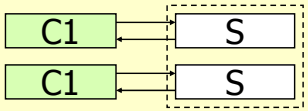
Ausführen

```
>java PostURL http://flp.cs.tu-berlin.de/~tolk/echo.cgi
Eingabe=%25%26%22+%22%2B%23%E4%DF
<!DOCTYPE HTML
   PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<HTML LANG="en-US"><HEAD><TITLE>Echo</TITLE>
</HEAD><BODY><H1>Echo der Eingabe:</H1><PRE>%&"
   "+#äß
</PRE></PRE></BODY></HTML>
```

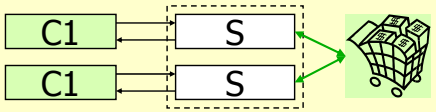

Zustand in Web Anwendungen

Zustand in HTTP

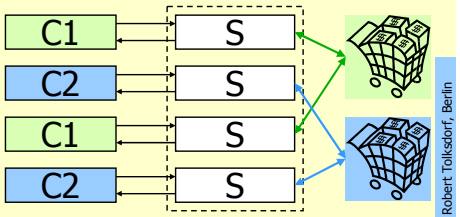
- HTTP ist zustandslos
  - Zwei Interaktionen sind unabhängig voneinander



- Zustand aber oft benötigt
  - Transaktionen auf Datensatz beim Server (z.B. Warenkorb)

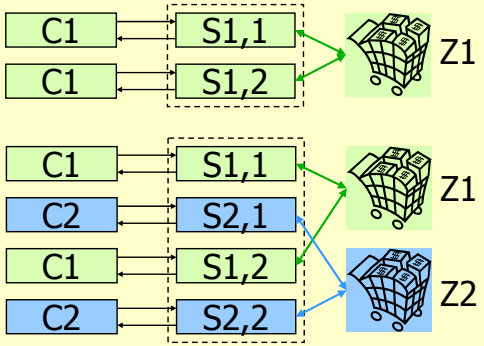


  - Unterscheidung von Klienten zur
    - Personalisierung
    - Authentifizierung
    - ...



Sessions

- Einführung von Sitzungen (Sessions)
- Sitzung: Folge von Interaktionen, die einen gemeinsamen Zustand haben
- Identifikation in der Interaktion durch eindeutige Sitzungsnummer (Session-ID)
- Ermittlung des Zustand auf Basis der Session-ID



Zustand in HTTP

- Client aus HTTP- und Socket-Informationen eindeutig identifizieren?
- Session-ID=
  (Browsername x User x Betriebssystem x IP-Adresse)
- *Nicht* eindeutig, weil:
  - Informationen bis auf IP-Adresse nicht immer vorhanden
  - IP-Adresse nicht eindeutig
    - Mehrere Nutzer auf einem Rechner
    - Proxy/Firewall/NAT Problematik: Keine individuellen IP-Adressen nach aussen
    - Mehrere Browser-Sessions des gleichen Nutzers
- => Session-ID muss in der Interaktion immer zwischen Klient und Server ausgetauscht werden

Zustand in HTTP

- 1. Versteckte Formularfelder enthalten Session-ID
- Formularfelder in HTML:
  - <input type=typ name="name" …> z.B.:
    - <input type="text" name="PLZ"> Texteingabe
    - <input type="password" name="pw">Passwordeingabe
    - Weitere Auswahlen und Textfelder
    - <input type="hidden" name="SessionID"
      value="977e5d8ae8500c456ab1fca6cbaa12af">
- Bei Submit wird ein Query-String

  PLZ=14195&pw=geheim&SessionID=977e5d8ae8500c456ab1fca6cbaa12af

  erzeugt und an den Server übermittelt
- Server wertet Session-ID aus und baut sie in Formulare auf Ergebnisseite ein

Woher Session IDs nehmen?

- Beispiel: http://ws.apache.org/axis
- String sessionID=
  org.apache.axis.utils.SessionUtils.generateSessionId();
- java -cp axis.jar;commons-logging.jar;
  commons-discovery.jar;. SessionID
- 43B04BB7719E14AC1105FCDB20F0CBD2
- Andere Bibliotheken natürlich auch nutzbar

Zustand in HTTP

- 2: Session-ID in URLs in Verweisen (URL Rewriting)
- www.amazon.de: Einstieg per HTTP auf URL mit schon kodierter Session-ID geleitet:
  http://www.amazon.de/exec/obidos/tg/browse/-/301128/ref=cs_nav_tab_1/028-1096689-7395702

http://www.amazon.de/exec/obidos/tg/browse/-/299956/ref=cs_nav_tab_3/028-1096689-7395702

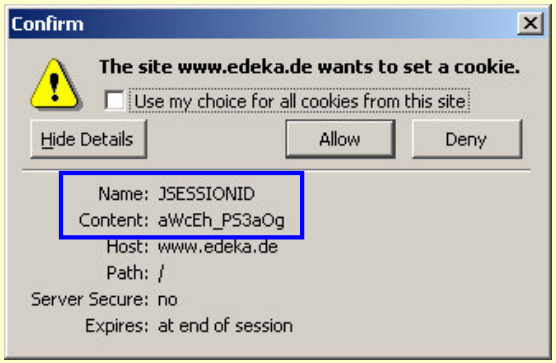
http://www.amazon.de/exec/obidos/tg/stores/static/-/general/international-gateway/ref=cs_nav_sn_1_new/028-1096689-7395702

URLs

- Session-ID kann unterschiedlich kodiert werden
- Abbildung von Pfad auf Informationen ist Server-Sache
  - Im Pfad:
    http://www.amazon.de/exec/obidos/tg/browse/-/301128/ref=cs_nav_tab_1/028-1096689-7395702
  - Im Query-String:
    http://www.cyberport.de/webshop/cyberportShop.omeco?ORDER=&PHPSESSID=8823f90c85597aedc87100cd91a4c7fd&FINANZING=
- Vorteil:
  - Zustand kann auch ausserhalb von Formulareingaben gehalten werden
  - Portabel
- Nachteile:
  - Alle Verweise müssen entsprechend markiert werden
  - Alte Session-ID kann in Bookmark sein
  - Gültige Session-ID kann einfach an andere Nutzer gelangen

Zustand in HTTP

- 3. Cookie Mechanismus zum Speicher der Session-ID



- Cookie ist kleiner Datensatz, der bei Klienten gespeichert ist
- Server kann ihn setzen
- Klient schickt ihn bei jeder weiteren Interaktion mit
- Implementiert mit zusätzlichen HTTP-Headern

Cookies

- Server schickt Set-Cookie Header in HTTP
  Set-Cookie: *Name*=*Wert*; expires=*Datum*; path=*Pfad*; domain=*Internet-Domäne*; secure
- Wenn der Klient will, speichert er den Cookie
- Beispiele:
  - www.edeka.de
    Set-Cookie: JSESSIONID=a3nwR5on0lhe; path=/
  - www.bmw.de
    Set-Cookie: WEBTRENDS_ID=160.45.114.204-1073465686.192079; path=/
  - www.amazon.de
    Set-Cookie: obidos_path_continue-shopping=continue-shopping-url=/tg/browse/-/301128%3Fsite-redirect%3Dde&continue-shopping-post-data=&continue-shopping-description=browse.gateway.301128; path=/; domain=.amazon.de

Cookie-Felder

- *Name*=*Wert*;
  Zeichenkette ohne Leerzeichen, Semikolon, Komma -> Ähnlich Query String kodieren
- domain=*IP-Domäne*
  Der Klient soll den Cookie bei einem GET an Rechner mitschicken, zu denen die IP-Domäne passt
  - mindestens drei Punkte erforderlich
  - bei Toplevel-Domänen (com, edu, net, org, gov, mil,…) nur zwei
- path=*Pfad*;
  Der Klient soll den Cookie bei einem GET an Rechner mitschicken, zu denen die IP-Domäne und der Pfad darin passt
  - / für alle Seiten
  - /foo für /foobar aber auch für /foo/bar.html

Cookie-Felder

- expires=*Datum*
  Altersgrenze ab der der Cookie nicht mehr gespeichert oder herausgegeben werden soll
  - *Datum:* „based on RFC 822, RFC 850, RFC 1036, and RFC 1123, with the variations that the only legal time zone is GMT and the separators between the elements of the date must be dashes."
  - Fehlt expires verfällt der Cookie am Ende der Nutzung des Klienten
  - Ist das Datum in der Vergangenheit soll der Klient den Cookie löschen
- secure
  Cookie nur an Server schicken, wenn dieser mit SSL (also https-Protokoll) kontaktiert wird

Cookies

- Wenn bei einer Anfrage nach einer URL
  - der Klient will,
  - die Domäne zur URL passt,
  - der Pfad zur URL passt,
  - bei mit secure markierten Cookies https benutzt wird
  - der Cookie nicht veraltet ist

  schickt der Klient das Name-Wert-Paar als Header mit
- Cookie: $Name_1 = Wert_1$; $Name_2 = Wert_2$; …
- Server kann mehrere Cookies senden (mehrere Set-Cookie Header)
- Klient kann mehrere Cookies antworten (mehrere pro Header oder mehrere Header)

Cookies

- Vorteile:
  - Einfacher Mechanismus ohne Inhaltsänderung
  - Seiten müssen nicht generiert werden
  - Cookies sind (wahrscheinlich) persistent über Klientennutzungen
- Nachteile:
  - Schlechtes Image von Cookies
  - Cookies für „fremde" Domains
  - Datenschutzaspekt / Nutzerprofile
  - Klientenverhalten heute frei konfigurierbar