# SSI = Sieciowe Systemy Informacyjne

Wykład nr 2

# Gniazdka "bezpieczne" - SSL/TLS

- SSL = Secure Socket Layer, TLS = Transport Layer Security,
  OpenSSL = implementacja SSL/TLS (biblioteka programistyczna i polecenie openssl)
- Skrócony opis pojęć kryptograficznych (patrz link)
- Co zapewnia SSL/TLS? (patrz link)
  szyfrowanie danych, żłośliwe zmiany niemożliwe, uwierzytelnianie serwera, uwierzytelnianie serwera i klienta
- Co jest potrzebne po stronie serwera ?
  certyfikat SSL serwera (z kluczem pub serwera), klucz pryw serwera
- Co jest potrzebne po stronie klienta ?
  jeśli klient chce sprawdzic certyfikat SSL serwera, to musi podac certyfikat SSL CA (który podpisal elektronicznie certyfikat serwera)
  podobnie w drugą stronę...
- Zasada działania ...
  1. serwer wysyła do klienta swój cert+klucz pub
  2. klient wymyśla klucz symetryczny X do szyfrowania danych, i szyfruje X kluczem pub serwera i wysyła do serwera
  3. serwer odszyfrowuje swoim kluczem pryw X; teraz oba końce mają X służący do szyfrowania danych płynących przez połączenie

- przyklad: ssl_kli.tcl i ssl_ser.tcl
- polecenie openssl:

```
# openssl: szyfrowanie/deszyfrowanie metoda Blowfish
# "-e" encode, "-d" decode, "-a" base64, "-bf" Blowfish
echo "tekst do zaszyfrowania" | openssl enc -e -a -bf -k haslo > qqq.txt
cat qqq.txt | openssl enc -d -a -bf -k haslo
```

- bezpieczna odmiana HTTP: HTTPS
  to samo co http, ale używa połączenia TCP nad SSL/TLS ...

# Prot. nad warstwą transportową: FTP

- służy do kopiowania plików File Transfer Protocol
- model klient/serwer
- zasada działania: używa 2 połączeń TCP;
  1. połączenie dla komend,
  przez to połaczenie klient wysyła rozkazy do serwera i otrzymuje odpowiedzi
  (patrz wydruk z sockspy, ftp01.txt)
  2. połączenie dla danych
  służy do kopiowania plików; tworzone gdy to jest potrzebne wiele razy...
- dwa tryby tworzenia połączenia dla danych:
  **aktywne FTP:**
  klient tworzy połączenie dla komend do serwera czekającego na porcie 21,
  port klienta w to N; klient wysyla komendę "PORT N+1" i czeka na połączenie od serwera na porcie N+1,
  serwer tworzy połączenie dla danych do klienta czekającego na porcie N+1
  **pasywne FTP:**
  klient tworzy połączenie dla komend do serwera czekającego na porcie 21,
  klient wysyła komendę "PASV" a serwer odpowiada z nr portu M, na którym będzie oczekiwał na połączenie dla danych (i robi to),
  klient tworzy połączenie dla danych do serwera czekającego na porcie M

# Prot. nad warstwą transportową: HTTP

- przeglądarka ściąga strony z serwera WWW za pomocą prot. HTTP
- ma także inne zastosowania, może być używany przez programy (inne niż przeglądarka)
- model klient/serwer, jedno połączenie TCP (wielokrotnie tworzone)
- opisane w dokumencie RFC 2616 (HTTP/1.1), http://tools.ietf.org/html/rfc2616

- żądanie HTTP:

```
GET /index.html HTTP/1.0 [CRLF]
Accept: image/gif, image/jpeg [CRLF]
User-Agent: Mozilla/4.0 [CRLF]
Host: www.cs.huji.ac.il:80 [CRLF]
Connection: Keep-Alive [CRLF]
[CRLF]
.....................................
```

pierwsza linia zawiera: metode, url, wersje protokołu
nastepne linie to tzw "nagłówki"
potem "pusta linia" i ew. dane (dane metody POST)

- metody w żądaniu HTTP:
**GET** - pobieranie zasobu na który wskazuje URL w żądaniu HTTP
(nie powinno niczego modyfikować na serwerze!!)
**POST** - przyjęcie danych od klienta (np. z formularza HTML)
**HEAD** - jak GET, ale nie pobiera danych zasobu (same nagłówki)
**PUT** - podobne do POST, ale URL oznacza co innego ("obiekt", a nie "metode")
**DELETE** - usuwanie zasobu

- ważne nagłówki w żądaniu HTTP:
  "Host: ???" - umożliwia tworzenie "wirtualnych hostów" (1 adres IP, wiele adresów domenowych), obowiązkowy w HTTP/1.1
  "Connection: Keep-Alive" - jedno połączenie używane do wielu zapytań HTTP
  "Authorization: Basic cXFxOnFxcQ==" - uwierzytelnianie klienta typu "basic" (jest też "digest"; pokazać przykład z sockspy...)
  "Cookie: ???" - ciasteczka wysyłane przez przeglądarkę do serwera

- odpowiedź HTTP:

```
HTTP/1.0 200 OK [CRLF]
Date: Fri, 31 Dec 1999 23:59:59 GMT [CRLF]
Content-Type: text/html[CRLF]
Content-Length: 1354[CRLF]
[CRLF]
<html>[CRLF]
<body>[CRLF]
<h1>Hello World</h1> [CRLF]
.....................................
```

  pierwsza linia zawiera: wersję prot, kod odpowiedzi i jej słowny opis
  następne linie zawierają nagłówki odpowiedzi HTTP
  potem "pusta linia" i dane odpowiedzi, np. HTML lub coś innego...

- kody w odpowiedzi HTTP:
  200 OK - prawidłowa odpowiedź
  302 Found - tzw "redirekt", przeglądarka powinna przełączyć się na inny URL
  401 Unauthorized - strona wymaga, aby użytkownik się uwierzytelnił
  404 Not Found - serwer nie znalazł zasobu

- ważne nagłówki w odpowiedzi HTTP:
  "Content-Type: text/html" - typ odpowiedzi jako mime
  "Content-Type: text/html; charset=utf-8"
  "Content-Length: 1354" - długość odpowiedzi HTTP
  "Set-Cookie" - serwer zmusza przeglądarke żeby utworzyła ciasteczko

- przekazywanie dodatkowych parametrów do żądania HTTP:

  1. zmienne w url-u, met. GET

     ```
     http://localhost:8001/np02/plik1.tcl?x=1234&y=4321
       # kodowanie znaków przy pomocy %kod, tzw  "x-url-encoding"
     ```

  2. "dane POST" za pusta linia, met. POST
     żadania http typu POST (lub GET) są tworzone przez formularz
     w pliku HTML, w przeglądarce, guzik submit
     lub przez biblioteki http, pokazać przykład http02.tcl ...
     kwestia kodowania znaków (utf-8 ? iso8859-2 ?)

     ```
     POST /np02/plik1.tcl HTTP/1.0[CRLF]
     Accept: */*[CRLF]
     Host: localhost:8001[CRLF]
     User-Agent: Tcl http client package 2.5.2[CRLF]
     Content-Type: application/x-www-form-urlencoded[CRLF]
     Content-Length: 15[CRLF]
     [CRLF]
     x=12345&y=54321[CRLF]
     ```

- ciasteczka czyli Cookies, session_id ...
  - w żądaniu HTTP:

    ```
    Cookie: nazwa1=wartość1; nazwa2=wartość2; ...
    ```

  - w odpowiedzi HTTP:

    ```
    Set-Cookie: nazwa=wartość; expires=DATA; path=ŚCIEŻKA; secure
      # expires -  czas życia ciasteczka u klienta
      # path - jaki url-i na hoscie to ciasteczko dotyczy
      # secure - tylko dla HTTPS
    ```

  - jak działają ciasteczka?
    + tworzone przez odpowiedź http serwera www (Set-Cookie:)
    + dopóki się nie przeterminują, wysyłane przez przeglądarke do serwera www (Cookie:) w każdym żądaniu http
    + fizycznie ciasteczka są przechowywanymi w plikach u klienta (przez przeglądarkę)

  - **zastosowanie ciasteczek:** umożliwiają przechowywanie "zmiennych sesyjnych" na serwerze www;
    + co to są "zmienne sesyjne" ? zmienne związane z sesją użytkownika
    + co to jest "sesja użytkownika" ? ciąg kliknięć (w przeglądarce), przez danego użytkownika, które nie są zbytnio oddzielone w czasie...
    + dlaczego ciasteczka są niezbędne? bo serwer http jest "bezstanowy"
    + identyfikator sesji jest przechowywany w ciasteczku u klienta ...

- obsługa sesji użytkownika na przykładzie frameworka webowego "OpenACS":
  ciasteczko z identyfikatorem sesji: *ad_session_id*
  parametry obsługi sesji:
  *SessionRenew* = 5min (czas po którym "odnawia się" ciasteczko sesji)
  *SessionTimeout* = 20min (bez odnawiania sesja znika po tym czasie)
  *SessionLifetime* = 7dni (sesja znika)
  pokazać przykład oacs_session.tcl ...

- ad_session_id/ pytanie 1: jak jest minimalny czas międz kliknięciami, po którym sesja może zniknąć???

- ad_session_id/ pytanie 2: dlaczego SessionRenew >0 ? wskazówka: strona www z 1000 obrazków ...