# Programowanie Aplikacji Sieciowych - Laboratorium 9

**Protokół HTTP (*Hypertext Transfer Protocol*, RFC 2616 oraz od 7230 do 7235)** - protokół warstwy aplikacji, wykorzystujący na niższej warstwie (zazwyczaj) gniazda TCP/IP oraz 2 domyślne porty: port niezabezpieczony `80` i port zabezpieczony: `443`.

Podstawowe informacje:

- Protokół HTTP jest protokołem wykorzystywanym do przesyłania plików (ogólnie mówiąc: zasobów) w sieci WWW (World Wide Web), bez względu na to, czy zasobem jest plik HTML, plik graficzny, wynik zapytania, czy cokolwiek innego
- Protokół HTTP, do wersji `1.1`, jest protokołem tekstowym, gdzie komendy protokołu, podobnie jak w SMTP, POP3 czy IMAP są komendami tekstowymi, zrozumiałymi dla człowieka
- HTTP to protokół typu zapytanie-odpowiedź. Zapytanie, wysyłane przez klienta, zawiera informację o żądanym zasobie. Odpowiedź, wysyłana przez serwer, zawiera treść zasobu. Jeśli serwer nie jest w stanie zwrócić odpytywanego zasobu, odpowiedź zawiera kod reprezentujący powód, dla którego zasób nie mógł być wysłany (np. zasób nie istnieje)
- Formaty zapytania i odpowiedzi HTTP są do siebie podobne; zarówno zapytanie, jak i odpowiedź HTTP zawierają (linia początkowa i nagłówki powinny się kończyć parą znaków `CRLF`, czyli `\r\n`):
  - linię początkową
  - 0 lub więcej nagłówków
  - pustą linię (`CRLF`, czyli `\r\n`)
  - opcjonalne ciało wiadomości

  **Przykład**:

```
linia poczatkowa, inna dla zadania, inna dla odpowiedzi \r\n
naglowek1: wartosc1 \r\n
naglowek2: wartosc2 \r\n
naglowek3: wartosc3 \r\n
\r\n
cialo wiadomosci, moze sie skladac z 1 lub
wielu linii, lub moze byc puste
```

- Nagłówki HTTP to wszelkie komendy używane do komunikacji między przeglądarką WWW (klientem) a serwerem. Nagłówki są to właściwości żądania i odpowiedzi przesyłane wraz z samą wiadomością. Służą one przede wszystkim do sterowania zachowaniem serwera oraz przeglądarki przez nadawcę wiadomości.
- Jeśli klient wysyła żądanie do serwera HTTP, żądanie powinno zawsze być zakończone parą znaków `CRLF` (czyli `\r\n`)
- Serwer odsyłając odpowiedź HTTP nie określa za pomocą żadnych specjalnych znaków końca odsyłanej odpowiedzi. W przypadku, gdy chcemy mieć pewność, że odebraliśmy całą odpowiedź serwera HTTP, musimy parsować odebrane nagłówki (`Content-Length` lub `Transfer-Encoding`), w których może znajdować się informacja o tym, jaki jest rozmiar odpowiedzi serwera, i użyć tej inforamcji do odebrania całej wiadomości. W przypadku, gdy serwer w odpowiedzi HTTP nie odeśle żadnego z powyższych nagłówków, aby mieć pewność odebrania całej odpowiedzi od serwera, musimy odbierać dane, dopóki serwer nie zakończy / zamknie połącznia. Zgodnie z formatem żądania i odpowiedzi HTTP, nagłówki od ciała oddzielają znaki `CRLF CRLF` (czyli `\r\n \r\n`).

Żądania HTTP

- Ogólny format ***żądania*** HTTP (pola oddzielone spacjami):

```
Method Request-URI HTTP-Version \r\n
HEADER1: VALUE1 \r\n
HEADER2: VALUE2 \r\n
...
HEADERX: VALUEX \r\n
\r\n
BODY
\r\n
```

gdzie:

- `Method` - to metoda żądania, dozwolone metody HTTP:
  - `GET` – pobranie zasobu wskazanego przez `Request-URI`
  - `HEAD` – pobiera informacje o zasobie, stosowane do sprawdzania dostępności zasobu
  - `PUT` – przyjęcie danych przesyłanych od klienta do serwera, najczęściej aby zaktualizować wartość zasobu,
  - `POST` – przyjęcie danych przesyłanych od klienta do serwera (np. wysyłanie zawartości formularzy),
  - `DELETE` – żądanie usunięcia zasobu,
  - `OPTIONS` – informacje o opcjach i wymaganiach dotyczących zasobu,
  - `TRACE` – diagnostyka, analiza kanału komunikacyjnego,
  - `CONNECT` – żądanie przeznaczone dla serwerów pośredniczących pełniących funkcje tunelowania,
  - `PATCH` – aktualizacja części zasobu (np. jednego pola).
- `Request-URI` - to ścieżka do zasobu na serwerze, która może zawierać dodatkowo parametry HTTP oraz fragment (za znakiem #),
- `HTTP-Version` - wersja protokołu HTTP, np. `HTTP/1.0`, `HTTP/1.1`, `HTTP/2.0`
- `HEADER1, HEADER2, ..., HEADERX` - nagłówki HTTP, `VALLUE1, VALUE2, ..., VALUEX` - wartości konkretnych nagłówków
- `BODY` - opcjonalne ciało żądania

Odpowiedzi HTTP

- Ogólny format ***odpowiedzi*** HTTP (pola oddzielone spacjami):

```
HTTP-Version Status-Code Reason-Phrase \r\n
HEADER1: VALUE1 \r\n
HEADER2: VALUE2 \r\n
...
HEADERX: VALUEX \r\n
\r\n
BODY
\r\n
```

gdzie:

- `HTTP-Version` - wersja protokołu HTTP, np. `HTTP/1.0`, `HTTP/1.1`, `HTTP/2.0`
- `Status-Code` - ***kod odpowiedzi***, który informuje klienta, w jaki sposób żądanie zostało lub nie zostało obsłużone, kody odpowiedzi to liczby trzycyfrowe, gdzie pierwsza z nich określa grupę odpowiedzi:
  - `1xx` - to kody informacyjne
  - `2xx` - to kody powodzenia
  - `3xx` - to kody przekierowania
  - `4xx` - to kody błędu aplikacji klienta
  - `5xx` - to kody błędu serwera

- `Reason-Phrase` - wiadomość powiązana z danym kodem odpowiedzi
- `HEADER1, HEADER2, ..., HEADERX` - nagłówki HTTP, `VALLUE1, VALUE2, ..., VALUEX` - wartości konkretnych nagłówków
- `BODY` - opcjonalne ciało żądania

Dozwolone nagłówki HTTP:

- ***General Header Fields*** - are a few header fields which have general applicability for both request and response messages, but which do not apply to the entity being transferred.
- ***Entity Header Fields*** - define metainformation about the entity-body or, if no body is present, about the resource identified by the request.
- ***Request Header Fields*** - allow the client to pass additional information about the request, and about the client itself, to the server.
- ***Response Header Fields*** - allow the server to pass additional information about the response which cannot be placed in the Status- Line. These header fields give information about the server and about further access to the resource identified by the Request-URI.

Przykłady żądań i odpowiedzi HTTP (można przetesotwać również samodzielnie: `telnet 212.182.24.27 8080`):

- Żądanie:

```
GET /index.html HTTP/1.1
HOST: 212.182.24.27
```

- Odpowiedź:

```
HTTP/1.1 200 OK
Date: Thu, 13 Apr 2017 14:25:38 GMT
Server: Apache/2.4.18 (Ubuntu)
Last-Modified: Thu, 13 Apr 2017 13:57:13 GMT
ETag: "2c39-54d0cb3af4405"
Accept-Ranges: bytes
Content-Length: 11321
Vary: Accept-Encoding
Content-Type: text/html


<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Apache2 Ubuntu Default Page: It works</title>
  </head>
  <body>
   ...
  </body>
</html>
```

- Żądanie:

```
TRACE / HTTP/1.1
HOST: 212.182.24.27
```

- Odpowiedź:

```
HTTP/1.1 405 Method Not Allowed
Date: Thu, 13 Apr 2017 14:31:22 GMT
Server: Apache/2.4.18 (Ubuntu)
Allow:
Content-Length: 302
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>405 Method Not Allowed</title>
</head><body>
<h1>Method Not Allowed</h1>
<p>The requested method TRACE is not allowed for the URL /.</p>
<hr>
<address>Apache/2.4.18 (Ubuntu) Server at 212.182.24.27 Port 80</address>
</body></html>
```

- Żądanie:

```
OPTIONS /index.html HTTP/1.1
HOST: 212.182.24.27
```

- Odpowiedź:

```
HTTP/1.1 200 OK
Date: Thu, 13 Apr 2017 14:52:31 GMT
Server: Apache/2.4.18 (Ubuntu)
Allow: OPTIONS,GET,HEAD,POST
Content-Length: 0
Content-Type: text/html
```

- Żądanie:

```
HEAD /index.html HTTP/1.1
HOST: 212.182.24.27
```

- Odpowiedź:

```
HTTP/1.1 200 OK
Date: Thu, 13 Apr 2017 14:53:06 GMT
Server: Apache/2.4.18 (Ubuntu)
Last-Modified: Thu, 13 Apr 2017 13:57:13 GMT
ETag: "2c39-54d0cb3af4405"
Accept-Ranges: bytes
Content-Length: 11321
Vary: Accept-Encoding
Content-Type: text/html
```

**Uwaga** W poniższych zadaniach zakładamy, iż serwer powinien obsługiwać tylko jednego klienta w danej chwili.

1. Pod adresem `httpbin.org` na porcie TCP o numerze `80` działa serwer obsługujący protokół HTTP w wersji `1.1`. Pod odnośnikiem `/html` udostępnia prostą stronę HTML. Napisz program klienta, który połączy się z serwerem, a następnie pobierze treść strony i zapisze ją na dysku jako plik z rozszerzeniem `*.html`. Spreparuj żądanie HTTP tak, aby serwer myślał, że żądanie przyszło od przeglądarki `Safari 7.0.3`. Jakich nagłówków HTTP należy użyć?

2. Pod adresem `httpbin.org` na porcie TCP o numerze `80` działa serwer obsługujący protokół HTTP w wersji `1.1`. Pod odnośnikiem `/image/png` udostępnia obrazek. Napisz program klienta, który połączy się z serwerem, a następnie pobierze obrazek i zapisze go na dysku. Jakich nagłówków HTTP należy użyć?

3. Pod adresem `212.182.24.27` na porcie TCP o numerze `8080` działa serwer obsługujący protokół HTTP w wersji `1.1`. Pod odnośnikiem `/image.jpg` udostępnia obrazek. Napisz program klienta, który połączy się z serwerem, a następnie pobierze z serwera obrazek w 3 częściach i po odebraniu wszystkich części złoży go w całość. Jakich nagłówków HTTP należy użyć?

4. Pod adresem `httpbin.org` na porcie TCP o numerze `80` działa serwer obsługujący protokół HTTP w wersji `1.1`. Pod odnośnikiem `/post` udostępnia formularz z polami do wypełnienia.

   Napisz program klienta, który połączy się z serwerem, a następnie uzupełni formularz danymi pobranymi od użytkownika, a następnie prześle go do serwera i odbierze odpowiedź.

   Aby sprawdzić, jak wygląda żądanie HTTP potrzebne do wypełnienia i wysłania formularza:

   - jakie *nagłówki HTTP* są wykorzystywane,
   - jak wygląda ciało zapytania,

   podsłuchaj komunikację z serwerem za pomocą Wiresharka, tj. uruchom przeglądarkę oraz Wiresharka; uzupełnij i zatwierdź formularz ręcznie za pomocą przeglądarki, a następnie sprawdź pakiety podsłuchane podczas komunikacji z serwerem `httpbin.org`. Możesz użyć filtrów Wiresharka: `http.request` oraz `http.response` (`http.request || http.response`).

5. **Slowloris, czyli Slow HTTP Headers DoS** Attak o nazwie Slowloris, dzięki wykorzystaniu pewnych koncepcji protokołu HTTP oraz sposobu obsługi żądań serwerów WWW, potrafi całkowicie je sparaliżować w przeciągu kilku sekund. Atak polega na utworzeniu dużej liczby gniazd, a następnie dosyłania w powolny sposób danych częściowych żądań HTTP, co w końcu skutkuje wyczerpaniem puli wolnych wątków obsługujących żądania HTTP.

   W klasycznym żądaniu, np. wykorzystującym metodę HTTP GET, do serwera wysyłana jest linia żądania, nagłówki oraz pusta linia CRLF oznaczająca koniec nagłówków. Atak Slowloris polega na wysyłaniu dużej liczby dodatkowych nagłówków, przykładowo X-a: b, które będą sukcesywnie dochodzić do atakowanego serwera dopiero po pewnym czasie. Podsumowując, atak działa następująco:

   1. Budowane są gniazda TCP (im więcej, tym lepiej, domyślnie 1000)
      `sock = socket(AF_INET, SOCK_STREAM)`
   2. Następuje podłączenie do serwera i wysyłanie podstawowych nagłówków
      `sock.connect(server)`, `sock.send('...')`,
   3. Wysyłany jest nagłówek `X-a: b \r\n`
      `sock.send('...')`
   4. Odczekujemy pewien czas (domyślnie 100 sekund)
      `time.sleep(100)`
   5. Wysyłamy ponownie nagłówek `X-a: b \r\n`
      `sock.send('...')`
   6. Powtarzamy do skutku kroki 4. i 5. dla każdego połączenia, ewentualnie dobudowujemy gniazda do zamkniętych połączeń

Znając założenia ataku Slowloris, napisz program klienta - atakującego, który wykona atak Slowloris na serwer WWW działający pod adresem `212.182.24.27` na porcie TCP `8080`. Jakich nagłówków HTTP należy użyć?

6. Zmodyfikuj program numer 3 z laboratorium numer 9 w taki sposób, aby program pobierał z serwera obrazek tylko wtedy, gdy nie zmienił się on od ostatniego pobrania. Jakich nagłówków HTTP należy użyć?

7. Napisz program serwera, który działając pod adresem 127.0.0.1 oraz na określonym porcie TCP, będzie serwerem HTTP. Obsłuż wybrane nagłówki i co najmniej jeden kod błędu (np. `404`). Jako przykładowe pliki serwera (stronę główną i stronę błędu) możesz wykorzystać pliki dostępne na stronie przedmiotu.