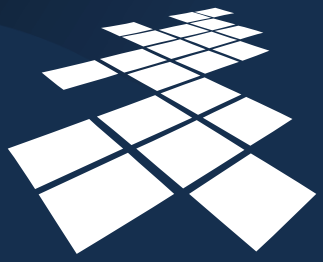
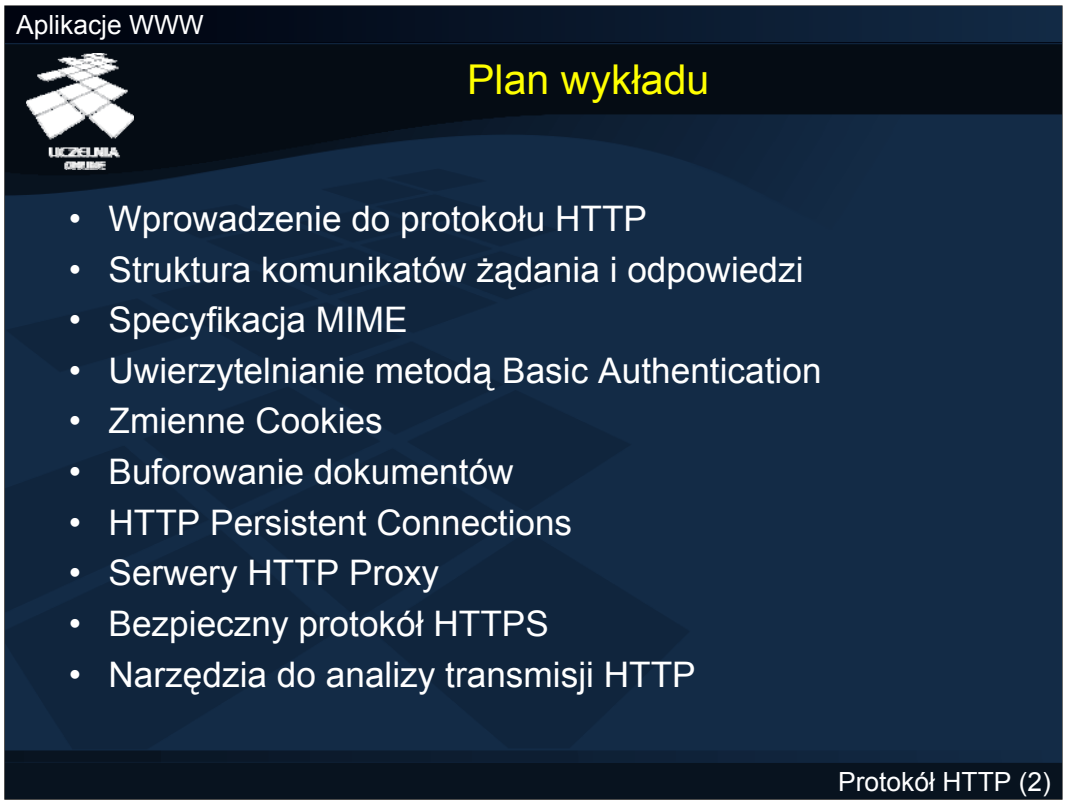
Wykład 4

Protokół HTTP

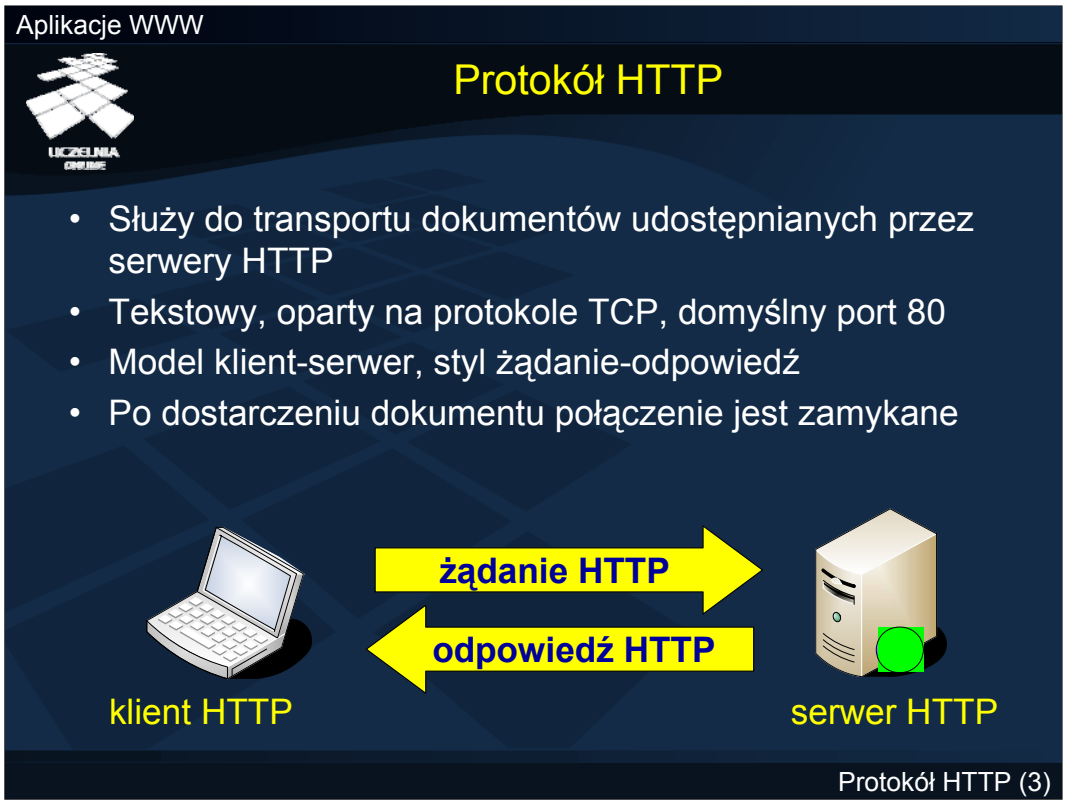
wykład prowadzi: Maciej Zakrzewicz



UCZELNIA
ONLINE

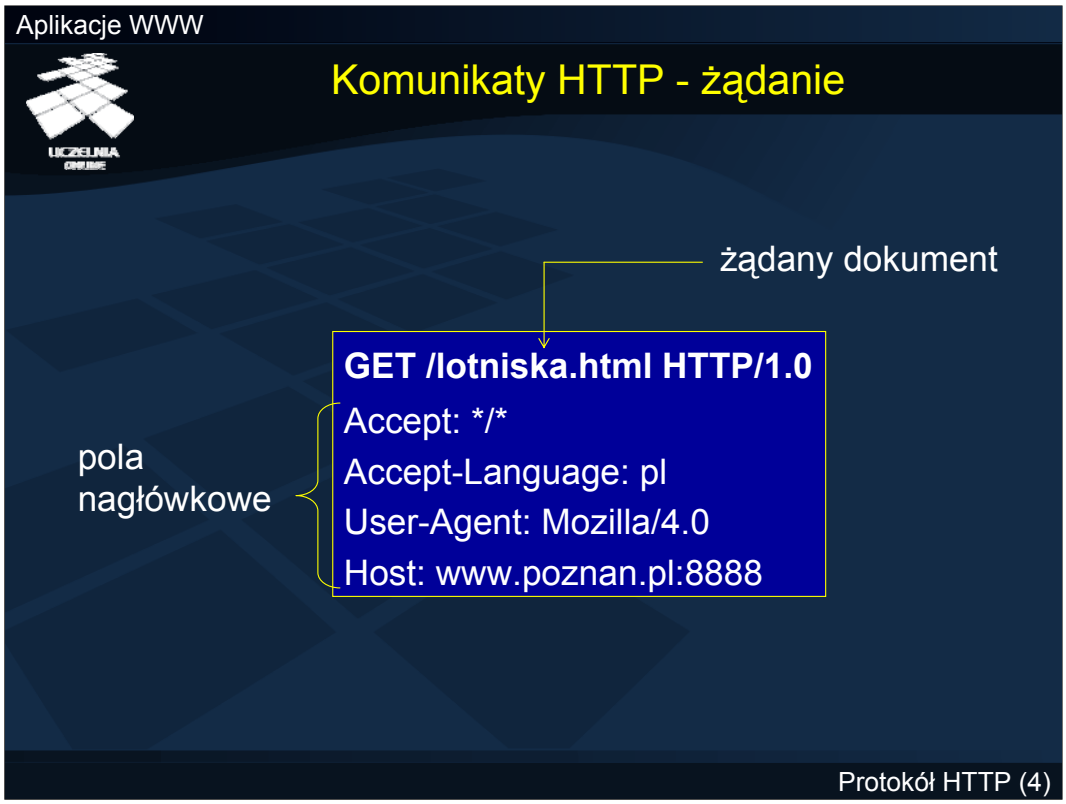


Celem wykładu jest przedstawienie podstawowych własności i mechanizmów protokołu sieciowego HTTP, stanowiącego podstawę architektury WWW. Wykład obejmie omówienie struktury komunikatów protokołu, podstawowe rozkazy, mechanizm uwierzytelniania użytkowników, mechanizm zmiennych Cookies, obsługę buforowania dokumentów, mechanizm połączeń o przedłużonej żywotności, koncepcję serwerów HTTP Proxy oraz bezpieczny protokół HTTPS. Na zakończenie zostaną przedstawione narzędzia do analizy transmisji HTTP, pozwalające na lepsze poznanie szczegółów technicznych protokołu.



HTTP (Hypertext Transfer Protocol) jest protokołem sieciowym służącym do transportu dokumentów (pliki HTML, pliki graficzne, strony generowane dynamicznie, itp.) udostępnianych przez serwery HTTP. HTTP jest protokołem tekstowym, opartym na TCP, zwykle wykorzystującym port komunikacyjny 80.

HTTP wykorzystuje model komunikacyjny typu klient-serwer. Klient HTTP nawiązuje połączenie sieciowe z serwerem HTTP i przekazuje mu komunikat zawierający żądanie HTTP. Serwer HTTP odpowiada na to żądanie wysyłając odpowiedź HTTP, zawierającą żądany dokument. Po dostarczeniu żądanego dokumentu serwer HTTP zamyka połączenie. HTTP jest protokołem bezsesyjnym i bezstanowym, co z jednej strony pozytywnie przekłada się na skalowalność środowisk WWW, a z drugiej stanowi istotne utrudnienie dla projektantów i programistów transakcyjnych aplikacji WWW.



Łączność HTTP ma formę wymiany komunikatów. Formaty zapisu żądania HTTP i odpowiedzi HTTP są podobne, zawierają: wiersz początkowy (np. adres URL dokumentu), zero lub wiele wierszy nagłówkowych (np. nazwę programu klienta HTTP), wolny wiersz (kody CR LF), opcjonalne ciało komunikatu (np. treść dokumentu).

Wiersz początkowy przybiera inną postać dla komunikatu żądania HTTP, a inną dla komunikatu odpowiedzi HTTP. Żądanie HTTP rozpoczyna się zwykle wierszem podobnym do poniższego:

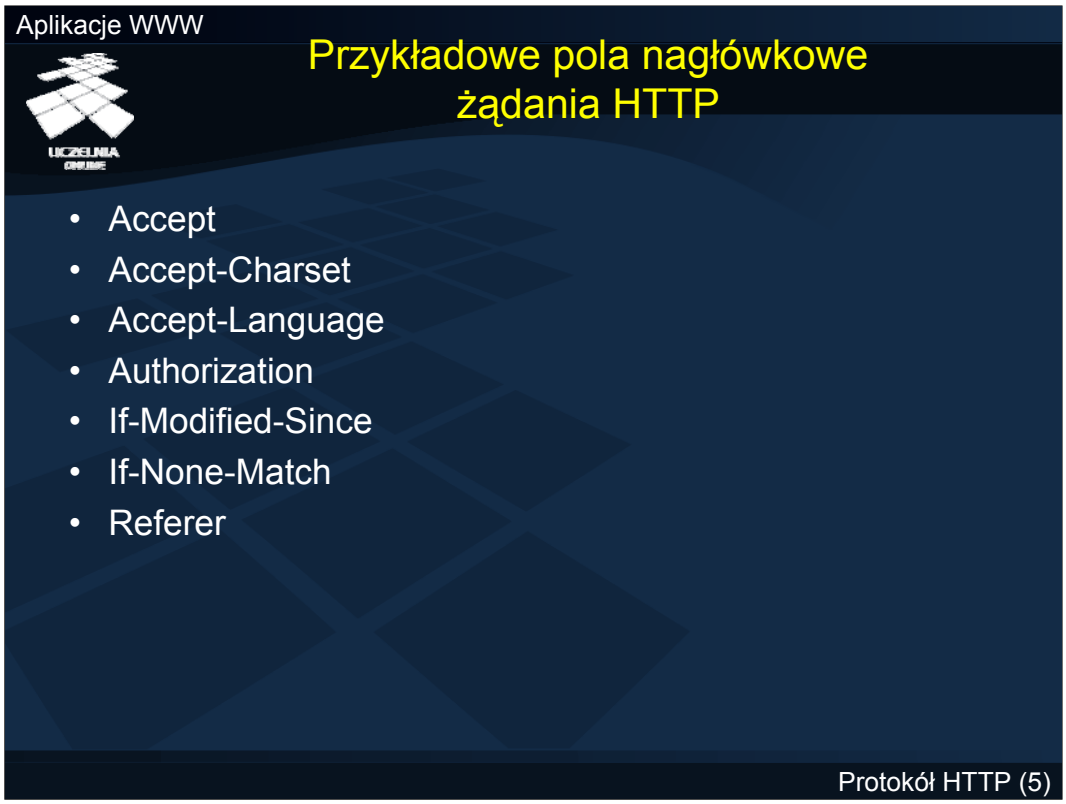
GET /ścieżka/plik.html HTTP/1.0

gdzie „GET" jest rozkazem protokołu HTTP wyrażającym żądanie otrzymania dokumentu, „ścieżka" opisuje lokalizację żądanego dokumentu w wirtualnym systemie plików serwera HTTP, „plik.html" jest nazwą żądanego pliku, „HTTP/1.0" wskazuje wersję protokołu HTTP, którą posługuje się klient HTTP.

Wiersze nagłówkowe zawierają metadane opisujące żądanie HTTP lub odpowiedź HTTP. Każdy wiersz nagłówkowy przedstawia jedno pole, zapisane w formacie: „Nazwa-pola: wartość", np.:

User-Agent: Mozila/4.0

gdzie „User-Agent" to nazwa pola nagłówkowego, w którym program-klient HTTP umieszcza swoją nazwę. Wielkość znaków nie jest rozróżniana w nazwach pól nagłówkowych. Pełna lista nazw pól nagłówkowych żądań HTTP i odpowiedzi HTTP jest częścią specyfikacji protokołu HTTP (RFC 2068).



Pole Accept jest wykorzystywane przez klienta HTTP do przedstawienia serwerowi HTTP listy obsługiwanych formatów danych. Na podstawie tej listy serwer HTTP dobiera właściwą postać treści dostępnej w wielu formatach. Np.: "Accept: text/html, text/plain; q=0.5" oznacza, że klient HTTP preferuje dokumenty w formacie "text/html", a jeżeli takie nie są dostępne, to w formacie "text/plain", zaznaczając jednocześnie, że będzie to oznaczać 50% utratę jakości. Gdyby serwer HTTP nie mógł dostarczyć treści w żadnym z podanych formatów, wtedy odpowiedź HTTP będzie zawierać kod zwrotny 406 (Not Acceptable).

Pole Accept-Charset jest wykorzystywane przez klienta HTTP do przedstawienia serwerowi HTTP listy preferowanych standardów kodowania znaków narodowych. Składnia pola oraz zachowanie serwera HTTP są analogiczne jak w przypadku pola Accept.

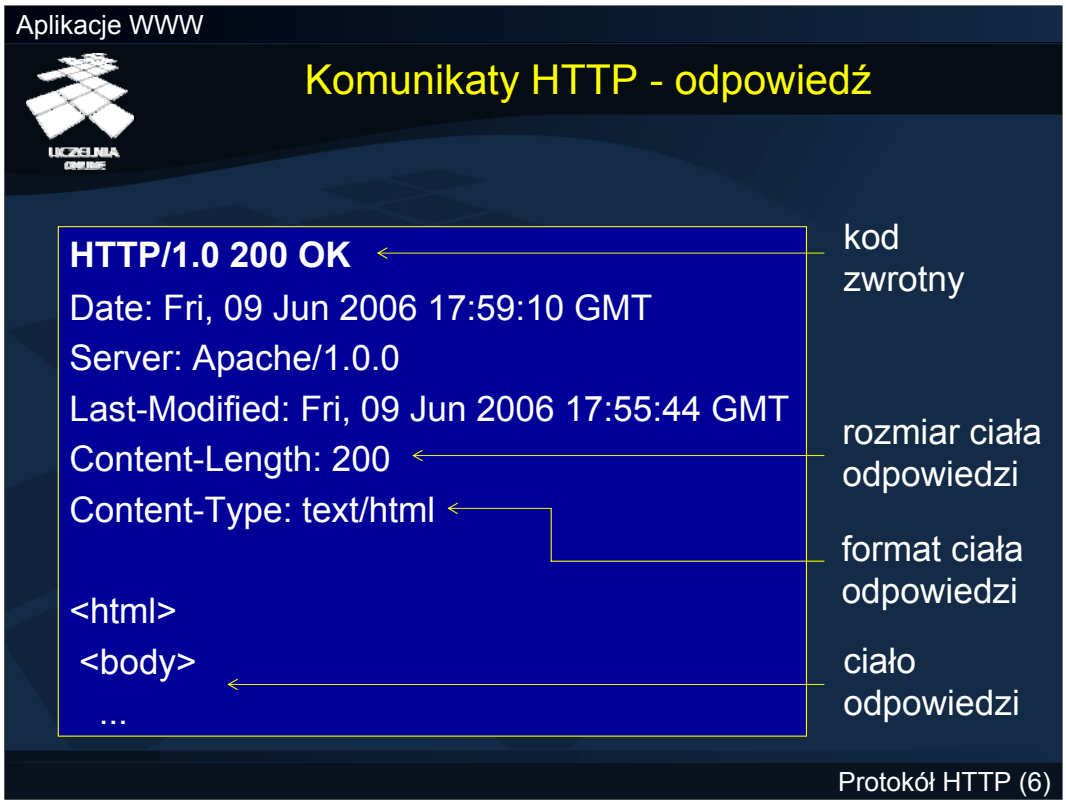
Pole Accept-Language jest wykorzystywane przez klienta HTTP do wskazania preferowanego języka narodowego dla dokumentu udostępnianego przez serwer HTTP. Pole to może być wykorzystywane zarówno przez twórców dokumentów HTML, jak i przez twórców aplikacji WWW do automatycznej selekcji języka narodowego dla użytkownika klienta HTTP.

Pole Authorization zawiera zakodowaną nazwę i hasło użytkownika, wymagane dla potrzeb uwierzytelnienia przez serwer HTTP.

Pole If-Modified-Since jest stosowane wraz z rozkazem GET w celu sformułowania warunkowego żądania HTTP. Warunkowe żądanie HTTP będzie obsłużone przez serwer HTTP tylko wtedy, gdy treść żądanego dokumentu uległa zmianie po czasie wskazanym w polu If-Modified-Since. W przeciwnym razie serwer HTTP zwróci kod 304 (Not Modified). Warunkowe żądania HTTP są wykorzystywane w obecności bufora dokumentów pobranych w przeszłości przez klienta HTTP w celu badania spójności posiadanej kopii z dokumentem oryginalnym.

Pole If-None-Match również umożliwia formułowanie warunkowych żądań HTTP lecz w tym przypadku warunek jest oparty na kodzie wersji dokumentu, tzw. ETag. Serwer HTTP obsłuży takie warunkowe żądanie tylko wtedy, gdy kod wersji dokumentu oryginalnego różni się od podanego, tzn. oryginalny dokument uległ zmianie w stosunku do posiadanej w buforze kopii.

Pole Referer umożliwia klientowi HTTP poinformowanie serwera HTTP o adresie URL dokumentu, w którym znajdował się adres URL przedmiotowego żądania HTTP. Informacja ta może być wykorzystana przez serwer HTTP do śledzenia ścieżek nawigacyjnych użytkowników.



Odpowiedź HTTP rozpoczyna się zwykle wierszem podobnym do poniższego:

HTTP/1.0 200 OK

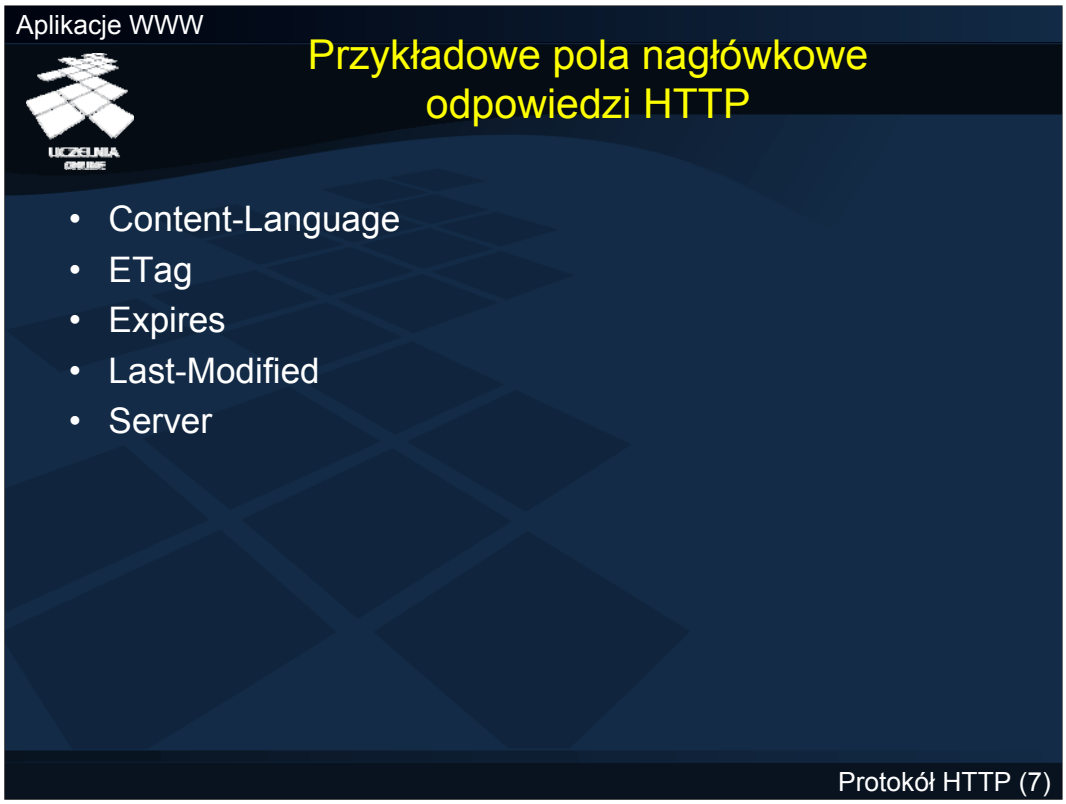
gdzie „HTTP/1.0” wskazuje wersję protokołu HTTP, którą posługuje się serwer, „200” jest kodem zwrotnym (OK) oznaczającym, że serwer HTTP bezbłędnie obsłużył żądanie HTTP. Przykłady innych kodów zwrotnych to: 404 (Not Found), oznaczający, że żądany dokument nie został znaleziony przez serwer HTTP, 500 (Server Error), oznaczający, że nastąpiła awaria serwera, itp. Pełna lista kodów zwrotnych jest częścią specyfikacji protokołu HTTP (RFC 2068).

Ciało komunikatu żądania HTTP lub odpowiedzi HTTP jest umieszczane poniżej wierszy nagłówkowych. W przypadku odpowiedzi HTTP ciałem komunikatu jest żądany dokument. W przypadku żądania HTTP, ciało komunikatu może zawierać wartości lub pliki wprowadzone przez użytkownika do formularza HTML. Jeżeli komunikat HTTP zawiera ciało, to w jego wierszach nagłówkowych znajdują się zwykle następujące pola: „Content-Type”, deklarujące format pliku stanowiącego ciało, „Content-Length”, deklarujące rozmiar tego pliku w bajtach, np.:

Content-Type: text/html

Content-Length: 200

gdzie “text/html” to symbol oznaczający format HTML, a „200” to rozmiar załączonego dokumentu HTML wyrażony w bajtach. Symbole formatów plików są definiowane przez specyfikację MIME (RFC 1521), np. „image/gif” oznacza plik graficzny w formacie GIF, a „application/zip” oznacza plik skompresowany w formacie ZIP. Odpowiedź HTTP może zawierać tylko jeden załączony plik/dokument.



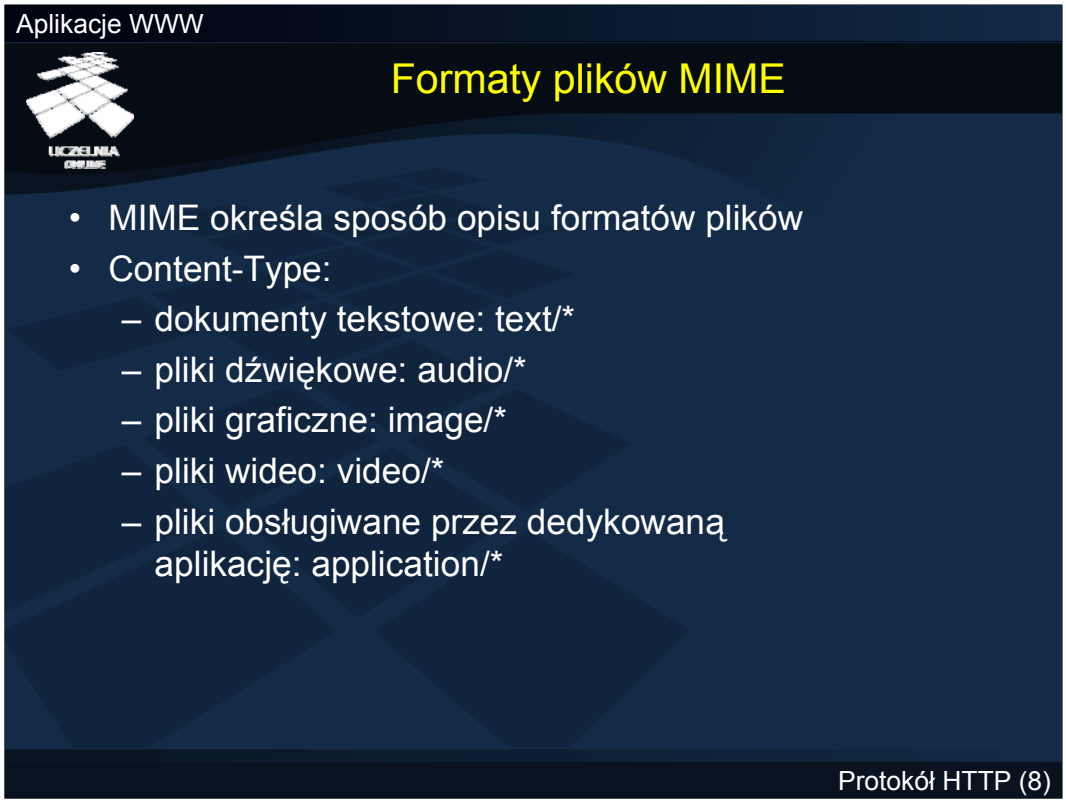
Pole Content-Language zawiera symbol języka narodowego, w którym zapisana jest treść przesyłanego dokumentu.

Pole ETag zawiera identyfikator wersji przesyłanego dokumentu. Jego wartość jest dowolnym łańcuchem znakowym cechującym się tym, że gdy zmianie ulega treść oryginalnego dokumentu, zmienia się również jego identyfikator wersji. Pole to służy klientowi HTTP do porównywania wersji kopii dokumentów znajdujących się w buforze z ich oryginałami umieszczonymi po stronie serwera HTTP.

Pole Expires definiuje moment czasowy, do którego klient HTTP może przechowywać otrzymany dokument w buforze i wykorzystywać do lokalnej obsługi powtórnych żądań HTTP, np. "Expires: Thu, 22 Jun 2006 16:00:00 GMT" . Po przekroczeniu zadeklarowanego czasu dokument powinien zostać usunięty z bufora.

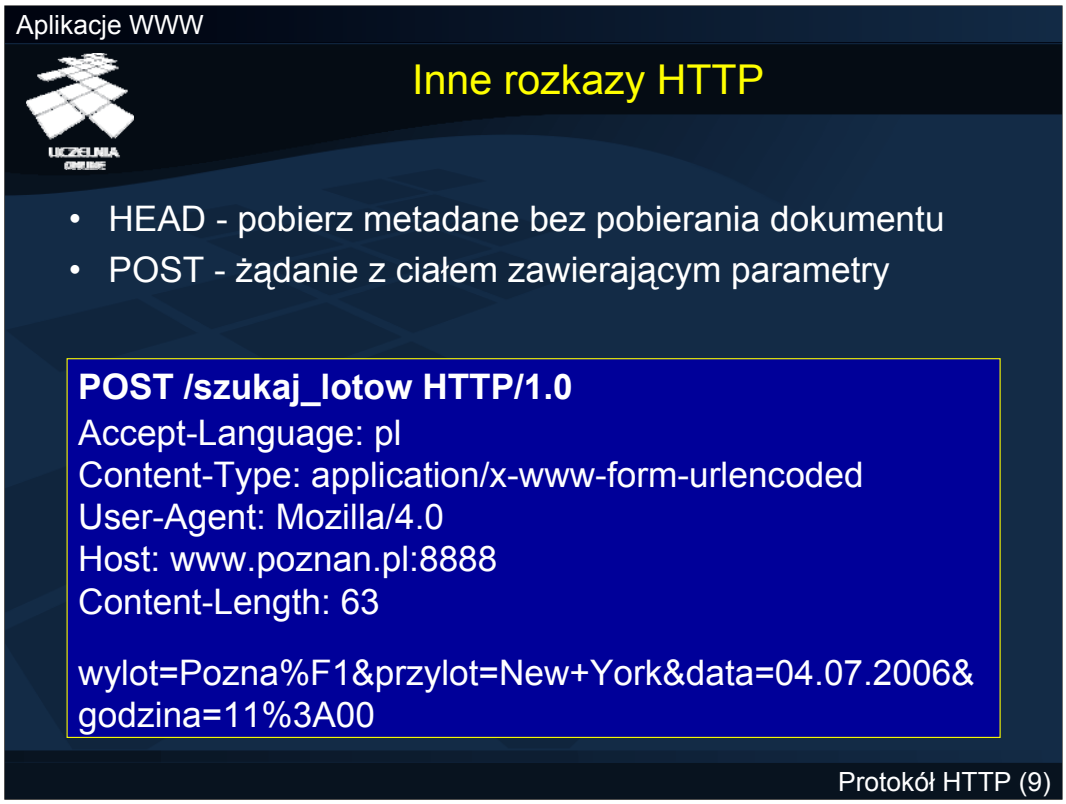
Pole Last-Modified wskazuje czas ostatniej modyfikacji treści dokumentu przesyłanego przez serwer HTTP. Pole to może być wykorzystywane na potrzeby zarządzania buforem.

Pole Server zawiera nazwę oprogramowania serwera HTTP, który obsłużył żądanie.



Specyfikacja MIME (RFC1521) definiuje sposób opisu formatów plików przesyłanych m.in. za pomocą protokołów SMTP, POP3, IMAP, HTTP. Każdy format pliku jest opisany jednoznacznym symbolem, składającym się z dwóch części: nazwy typu i nazwy podtypu, rozdzielonych znakiem "/". Poniżej przedstawiono najczęściej wykorzystywane formaty MIME:

text/html - dokument hipertekstowy w formacie HTML
text/css - arkusz stylistyczny CSS
text/plain - dokument tekstowy
audio/midi - plik muzyczny w formacie MIDI
audio/mpeg - plik dźwiękowy w formacie MP3
audio/x-realaudio - strumień dźwiękowy w formacie Real Audio
audio/x-wav - plik dźwiękowy w formacie WAV
image/gif - plik graficzny w formacie GIF
image/jpeg - plik graficzny w formacie JPEG
image/png - plik graficzny w formacie PNG
video/mpeg - plik wideo w formacie MPEG
video/quicktime - plik wideo w formacie QuickTime
application/pdf - dokument w formacie PDF
application/zip - plik skompresowany w formacie ZIP



Oprócz rozkazu GET, powszechnie wykorzystuje się też rozkazy HEAD i POST.

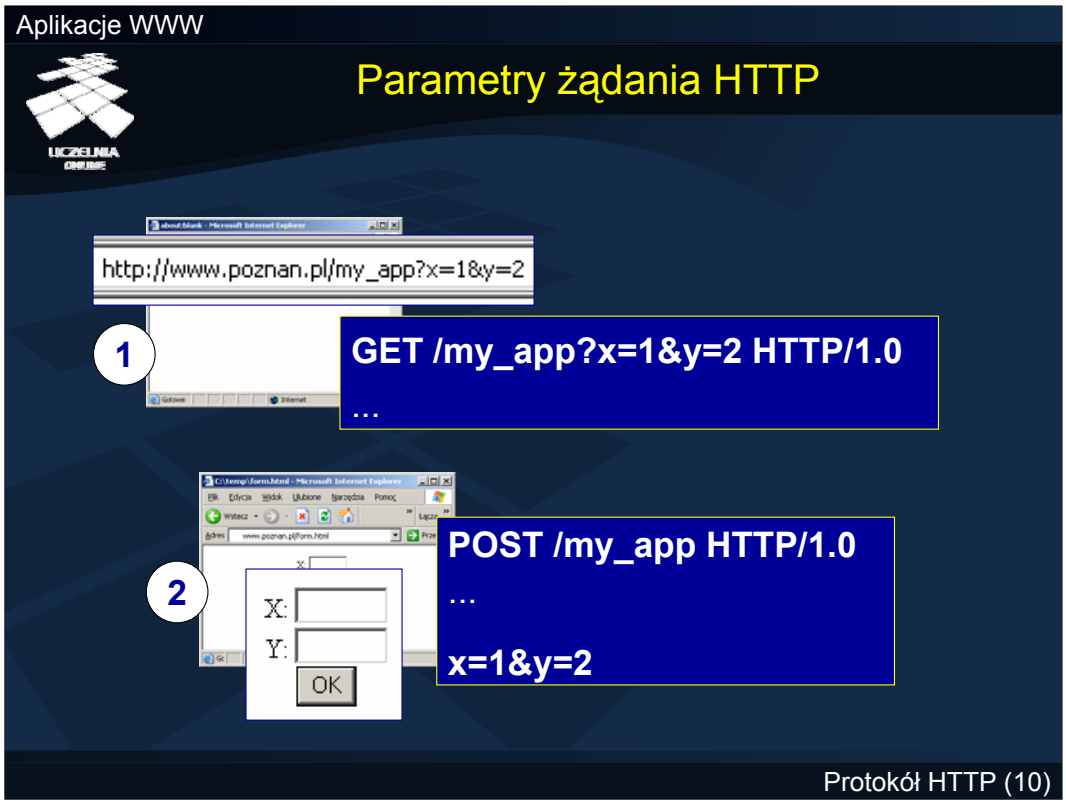
Rozkaz HEAD

Rozkaz HEAD jest podobny do rozkazu GET – reprezentuje żądanie klienta HTTP, jednak z tą różnicą, że w jego wyniku klient otrzymuje wyłącznie wiersze nagłówkowe odpowiedzi HTTP, bez załączonego ciała (dokumentu wynikowego). Rozkaz ten umożliwia przeglądarce pobranie metadanych opisujących dokument, bez konieczności pobierania pełnej treści dokumentu. Główne zastosowania obejmują weryfikowanie poprawności adresów i łączy URL.

Rozkaz POST

Rozkaz POST jest podobny do rozkazu GET – reprezentuje żądanie klienta HTTP, lecz do żądania tego dołączone jest ciało, które reprezentuje dane wysyłane przez klienta HTTP do serwera HTTP (np. parametry, plik). W związku z obecnością ciała żądania, komunikat sieciowy zawiera pola nagłówkowe „Content-Type" i „Content-Length", a najpopularniejszym formatem przekazywanej treści jest „Content-Type: application/x-www-form-urlencoded", oznaczający dane formularza HTML przekształcone do postaci ASCII metodą „URL-encoding" (RFC 1738)

W przedstawionym przykładzie ciało komunikatu zawiera cztery parametry przekazane serwerowi HTTP przez klienta HTTP: parametr „wylot" o wartości „Poznań", w której litera „ń", jako niekompatybilna z zestawem znaków ASCII została zastąpiona kodem heksadecymalnym; parametr „przylot" o wartości „New York", w której znak spacji został zastąpiony znakiem „+"; parametr „data" o wartości „04.07.2006" oraz parametr „godzina" o wartości „11:30", w którym znak „:" został zastąpiony kodem heksadecymalnym wg ASCII.



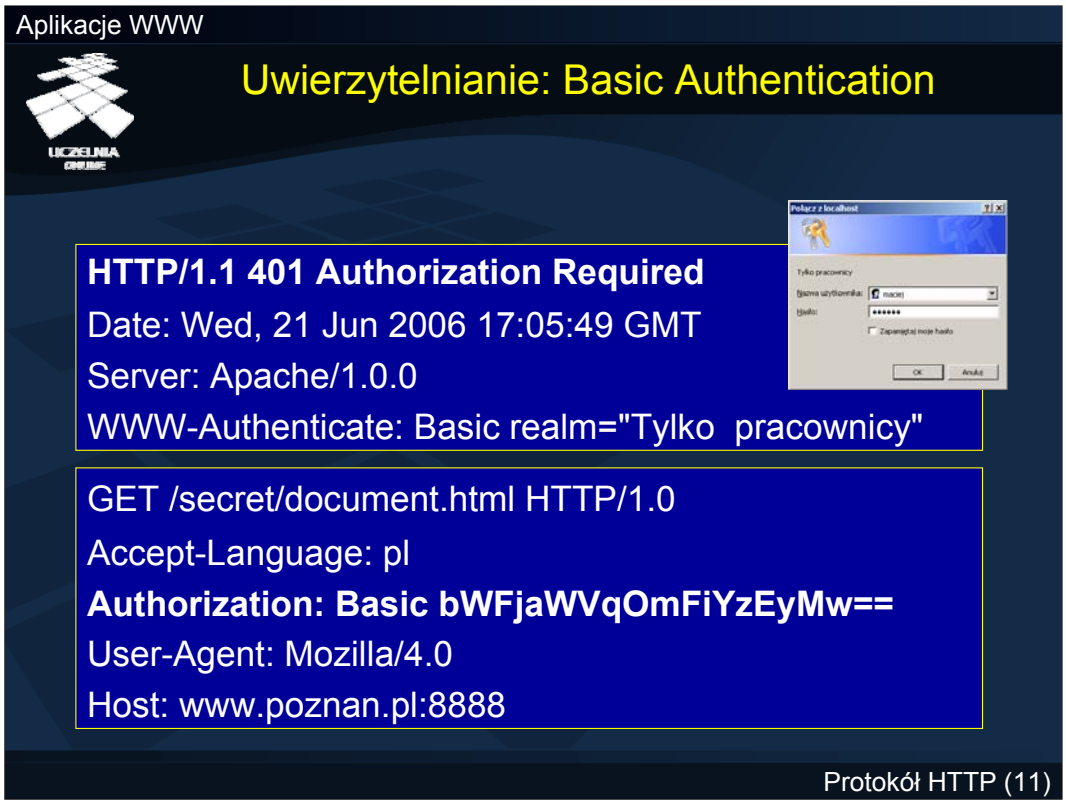
Często w przypadku gdy żądanie klienta HTTP dotyczy uruchomienia aplikacji znajdującej się po stronie serwera HTTP, klient HTTP przekazuje parametry wywołania takiej aplikacji. Parametry wywołania mogą być przekazane za pośrednictwem protokołu HTTP na dwa sposoby:

1. Rozkaz GET, parametry dołączone do adresu URL skojarzonego z aplikacją

2. Rozkaz POST, parametry w ciele żądania HTTP

Na slajdzie przedstawiono obie metody przekazywania parametrów wywołania aplikacji znajdującej się po stronie serwera HTTP.

W pierwszym przypadku zwykle istnieje ograniczenie na łączny rozmiar łańcucha znakowego zawierającego parametry żądania HTTP - dla Microsoft Explorera 6 wynosi ono 2083 znaki na cały adres URL. W drugim przypadku rozmiary parametrów nie są ograniczone.

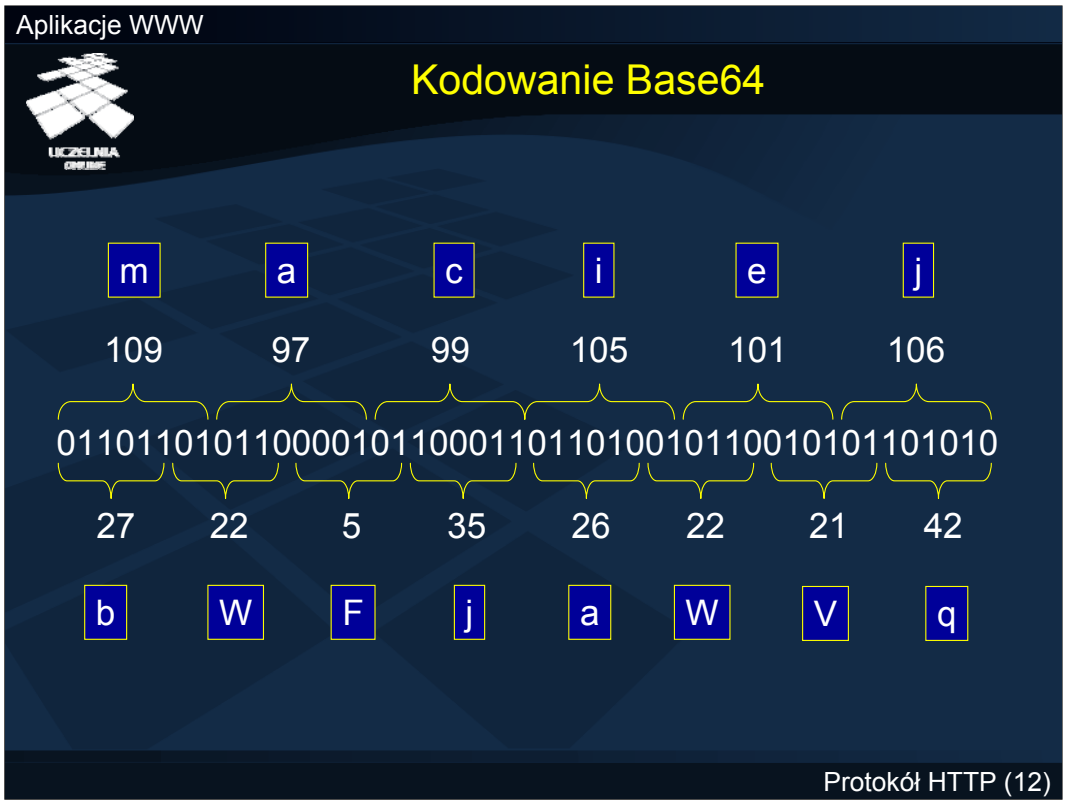


Nie wszystkie dokumenty zgromadzone po stronie serwera HTTP powinny być udostępniane każdemu użytkownikowi. Jedną z najprostszych metod ochrony dostępu jest uwierzytelnianie użytkowników za pomocą nazwy i hasła.

Protokół HTTP obsługuje prostą metodę uwierzytelniania użytkowników nazwaną Basic Authentication. Jeżeli serwer HTTP wymaga uwierzytelnienia użytkownika, wtedy odpowiedź na żądanie HTTP zawiera kod zwrotny 401 (Authorization Required), oznaczający, że klient HTTP powinien zażądać od użytkownika wprowadzenia nazwy i hasła. W takiej sytuacji klient HTTP pobiera od użytkownika jego nazwę i hasło, łączy je w jeden łańcuch znakowy, koduje za pomocą algorytmu Base64, a następnie zapisuje w polu nagłówka „Authorization” ponowionego żądania HTTP. Jeżeli wynik weryfikacji nazwy/hasła użytkownika przez serwer HTTP jest negatywny, serwer HTTP ponownie przesyła kod zwrotny 401 (Authorization Required) – aż do skutku.

Wprowadzone przez użytkownika dane uwierzytelniające są przez klienta HTTP automatycznie dołączane do kolejnych żądań HTTP.

Należy zauważyć, że metoda HTTP Basic Authentication nie zapewnia ochrony przekazywanych danych uwierzytelniających. Kodowanie Base64 jest odwracalne, dzięki czemu ewentualny podsłuch połączenia HTTP umożliwia intruzowi przejęcie nazwy i hasła użytkownika. Pewniejsze metody uwierzytelniania użytkowników opierają się na formularzach HTML, których dane są przekazywane za pośrednictwem szyfrowanego protokołu HTTPS.

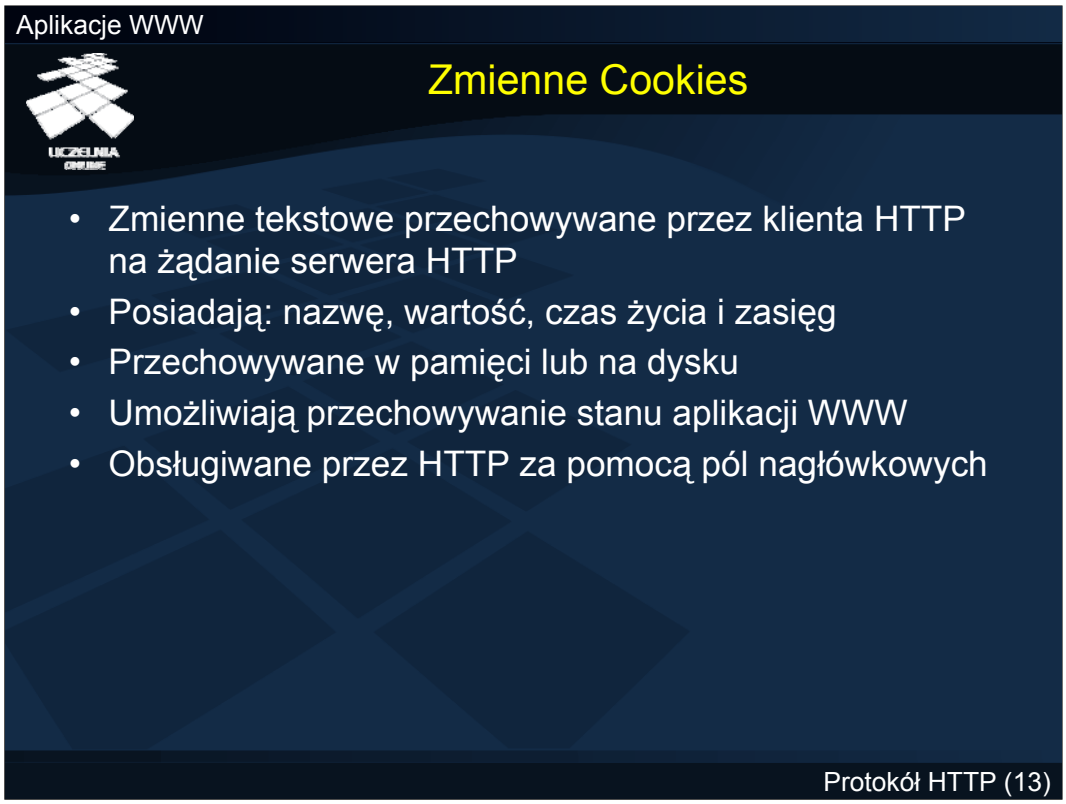


Kodowanie Base64 zostało opracowane na potrzeby poczty elektronicznej - umożliwia zapisanie ciągu bajtów w formie łańcucha znakowego, dzięki czemu możliwa jest jego transmisja za pośrednictwem znakowej infrastruktury komunikacyjnej (np. SMTP). Ze źródłowego ciągu bajtów wybierane są 6-bitowe odcinki, które po konwersji do wartości dziesiętnych służą jako indeksy w specjalnej tablicy kodowej Base64. Wskazywane znaki są umieszczane w wynikowym łańcuchu znakowym.

Powyższy slajd ilustruje przebieg kodowania nazwy użytkownika "maciej" na potrzeby uwierzytelniania Basic Authentication. Każdy znak nazwy użytkownika jest opisywany przez swój kod ASCII, przekształcany do postaci binarnej. Następnie z ciągu bitów wycinane są 6-bitowe odcinki. Każdy 6-bitowy odcinek jest przekształcany do postaci dziesiętnej, a następnie zastępowany odpowiednim znakiem z tablicy Base64. Wynikowy ciąg znaków to "bWFjaWVq".

Tablicę kodową Base64 przedstawiono poniżej.

indeks	znak	indeks	znak	indeks	znak	indeks	znak
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		



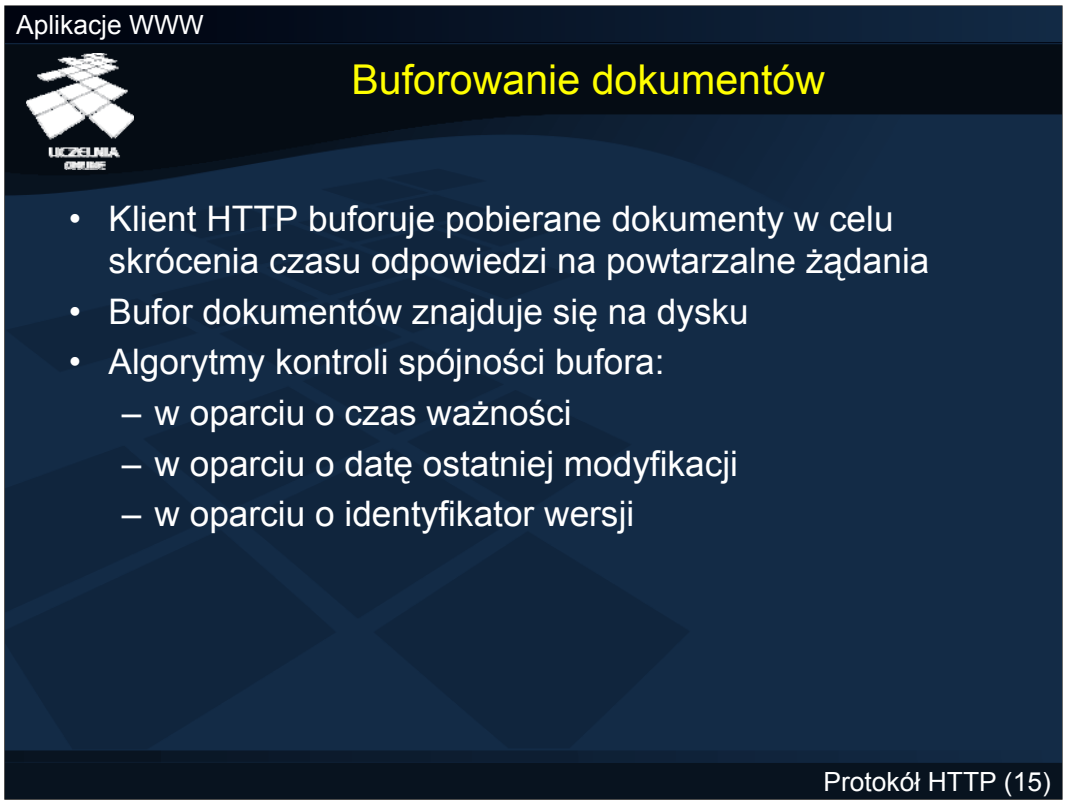
Protokół HTTP został zaprojektowany jako bezstanowy, co oznacza, że serwer HTTP nie potrafi rozpoznać czy określone żądania HTTP pochodzą od tego samego użytkownika końcowego, czy od użytkowników niezależnych. Ponieważ wiele zastosowań aplikacyjnych wymaga korelowania żądań HTTP, dlatego zaproponowano rozszerzenie funkcjonalności protokołu o obsługę niewielkich zmiennych o czasie życia dłuższym od czasu trwania połączenia HTTP. Zmienne te nazwano Cookies.

Zmienna Cookie jest powoływana do życia przez aplikację pracującą po stronie serwera HTTP, a następnie przesyłana do klienta HTTP. Zadaniem klienta HTTP jest przechować tę zmienną, a następnie wysyłać ją do serwera HTTP wraz z kolejnymi żądaniami HTTP. Dzięki temu serwer HTTP ma możliwość tymczasowego przechowywania zmiennych w pamięci klienta HTTP, a co się z tym wiąże, podtrzymywać stan aplikacji obsługującej klienta.

Każda zmienna Cookie posiada nazwę, wartość będącą łańcuchem znakowym, czas życia określający jak długo klient HTTP powinien ją przechowywać, oraz zasięg rozumiany jako dziedzinę adresów URL, którym zmienna ta będzie udostępniana. Jeżeli czas życia zmiennej Cookie wynosi "-1", to zmienna jest przechowywana w pamięci operacyjnej. W przeciwnym przypadku jest zapisywana na dysku przez klienta HTTP.

Przesyłanie zmiennych Cookies pomiędzy klientem HTTP a serwerem HTTP przebiega według następującego schematu:

1. Klient HTTP wysyła do serwera HTTP żądanie uruchomienia aplikacji.
2. Serwer HTTP wysyła do klienta HTTP odpowiedź wygenerowaną przez aplikację. W nagłówku odpowiedzi HTTP znajduje się pole Set-Cookie, zawierające m.in. nazwę i wartość zmiennej Cookie: v1=ABC.
3. Klient HTTP pobiera zmienną Cookie z otrzymanego nagłówka odpowiedzi HTTP i zapisuje na dysku lokalnym.
4. Podczas wysyłania każdego następnego żądania HTTP klient umieszcza w jego nagłówku pole Cookie, zawierające nazwę i wartość zmiennej Cookie. W ten sposób zmienna powraca do serwera HTTP i może być wykorzystywana przez jego aplikacje. Jeżeli klient HTTP otrzymał wiele zmiennych Cookies, to wszystkie umieszcza w polu nagłówkowym Cookie, separując średnikiem.



W celu skrócenia czasu odpowiedzi na żądanie użytkownika, a także w celu redukcji obciążenia serwerów HTTP, klient HTTP buforuje pobierane w przeszłości dokumenty i wykorzystuje je do uproszczonej obsługi nowych żądań.

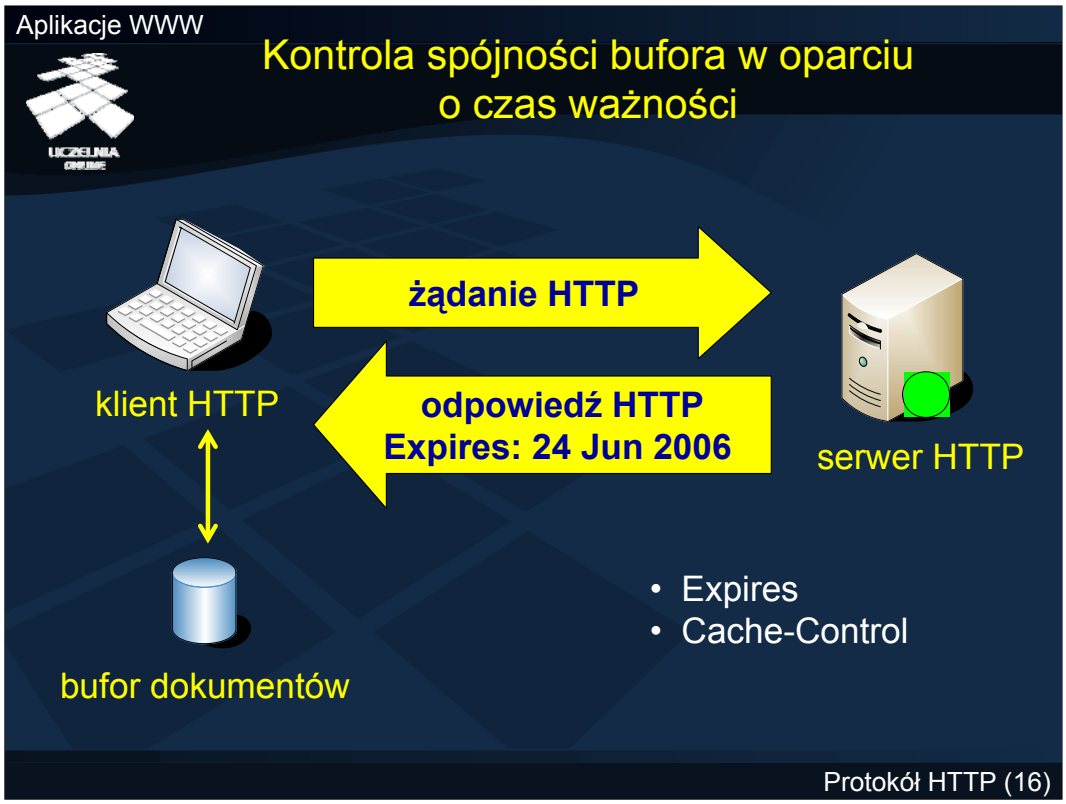
Bufor dokumentów jest najczęściej umieszczony w lokalnym systemie plików klienta HTTP. Gdy użytkownik formułuje nowe żądanie, bufor ten jest przeszukiwany w celu znalezienia kopii żądanego dokumentu. Jeżeli taka kopia zostanie znaleziona, klient HTTP rezygnuje z połączenia z serwerem HTTP i natychmiast przedstawia posiadaną kopię użytkownikowi.

Ten rodzaj buforowania stwarza zagrożenie, iż kopia dokumentu znajdująca się w buforze nie jest już identyczna z oryginalnym dokumentem znajdującym się po stronie serwera HTTP, który mógł ulec modyfikacjom. Aby uniknąć ryzyka niespójności, klient HTTP stosuje jeden z algorytmów kontroli spójności bufora:

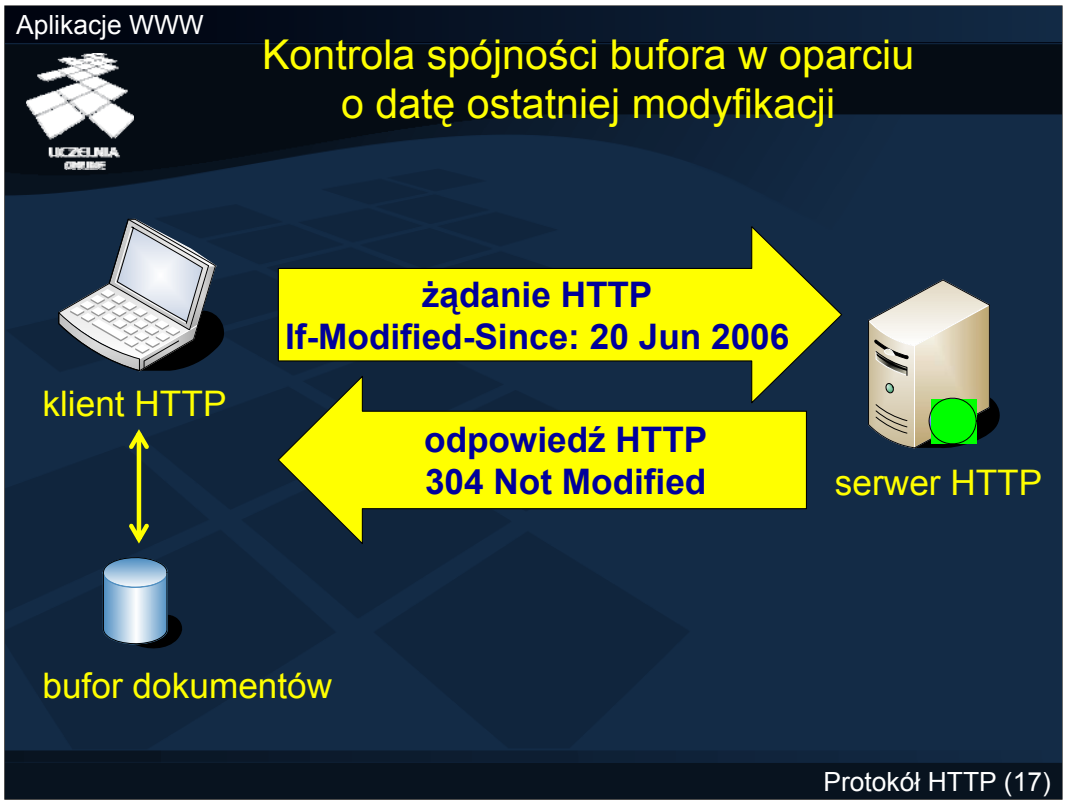
1. Kopie dokumentów są przechowywane w buforze wyłącznie przez czas ważności zadeklarowany przez serwer HTTP.

2. Klient HTTP każdorazowo odpytuje serwer HTTP, czy kopia dokumentu utworzona w podanym momencie czasowym jest nadal identyczna z oryginałem.

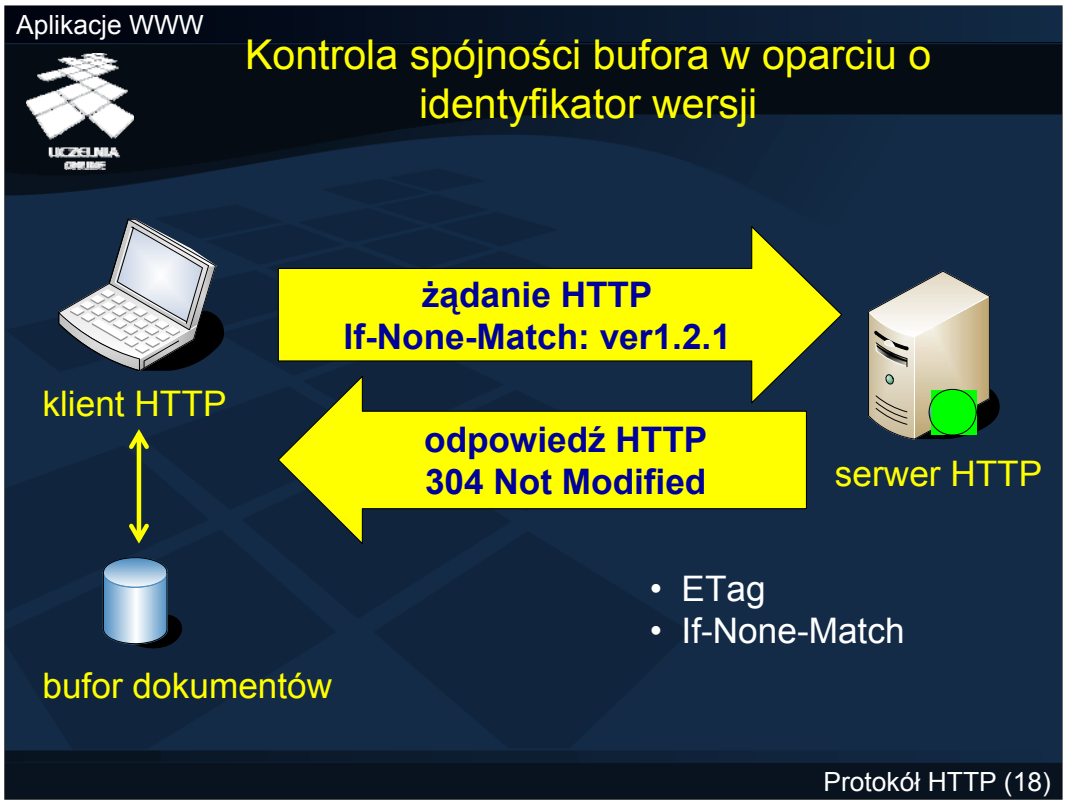
3. Klient HTTP każdorazowo odpytuje serwer HTTP, czy identyfikator wersji kopii dokumentu jest nadal identyczny z identyfikatorem wersji oryginału.



Buforowanie w oparciu o czas ważności dokumentu jest możliwe, jeżeli serwer HTTP umieszcza w nagłówkach odpowiedzi pole Expires lub Cache-Control, zawierające deklarowany czas, przez który kopię dokumentu wolno buforować i wykorzystywać do uproszczonej obsługi nowych żądań użytkowników. Gdy czas ważności dokumentu zostanie przekroczony, klient HTTP powinien usunąć kopię dokumentu z bufora lub pobrać aktualną wersję. Różnica pomiędzy użyciem pól nagłówkowych Expires a Cache-Control polega na tym, że Expires wskazuje bezwzględny punkt czasowy, natomiast Cache-Control umożliwia zliczanie czasu życia od momentu umieszczenia kopii w buforze.

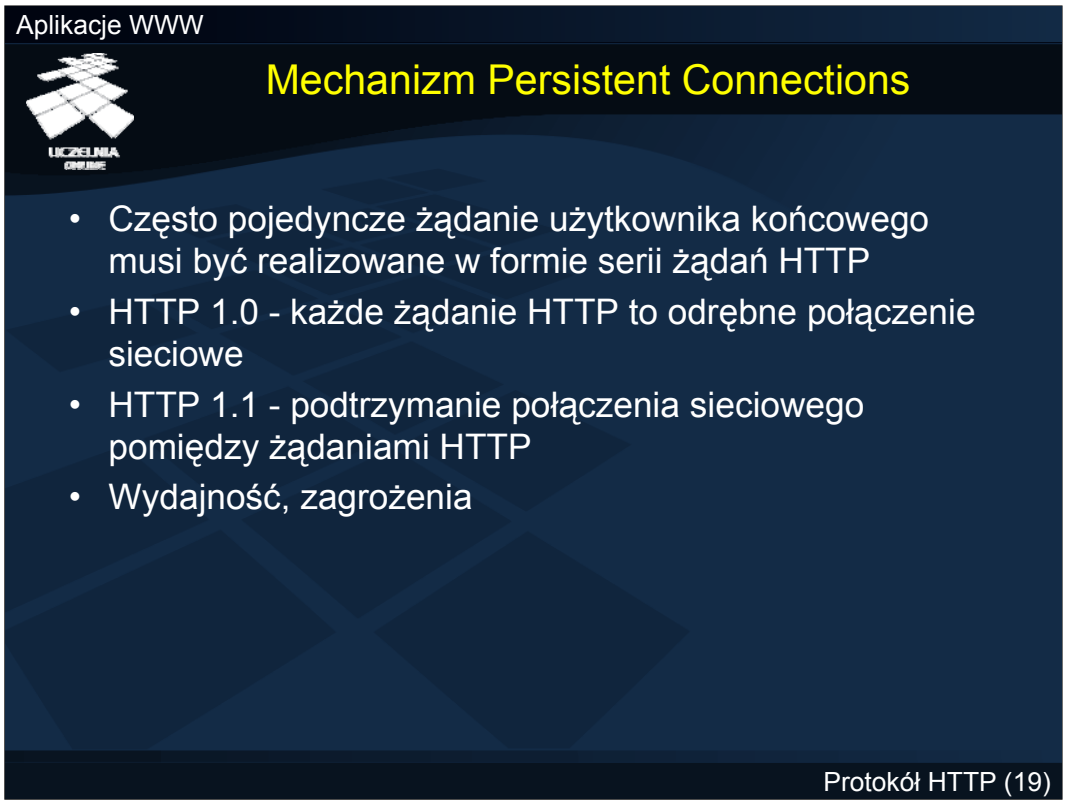


Buforowanie w oparciu o datę ostatniej modyfikacji opiera się na warunkowych żądaniach HTTP. Klient HTTP, który posiada w buforze kopię dokumentu żądanego przez użytkownika, zwraca się do serwera HTTP z żądaniem HTTP zawierającym znacznik czasowy opisujący moment utworzenia kopii w buforze. Serwer HTTP porównuje ten znacznik czasowy z czasem ostatniej modyfikacji oryginalnego dokumentu i bądź przesyła nową postać dokumentu (jeżeli był on modyfikowany), bądź też odpowiada kodem zwrotnym 304 (Not Modified), zezwalając klientowi HTTP na użycie kopii z bufora.



Buforowanie w oparciu o identyfikator wersji opiera się na założeniu, że z każdym dokumentem znajdującym się po stronie serwera HTTP związano łańcuch znakowy nazwany identyfikatorem wersji (entity tag, ETag). Najważniejszą własnością identyfikatora wersji jest to, że ulega zmianie zawsze wtedy, gdy zmieniana jest treść oryginalnego dokumentu. W wielu implementacjach serwerów HTTP identyfikatory wersji są generowane na podstawie sumy kontrolnej dokumentu. Serwer HTTP umieszcza identyfikator wersji przesyłanego dokumentu w polu nagłówkowym ETag.

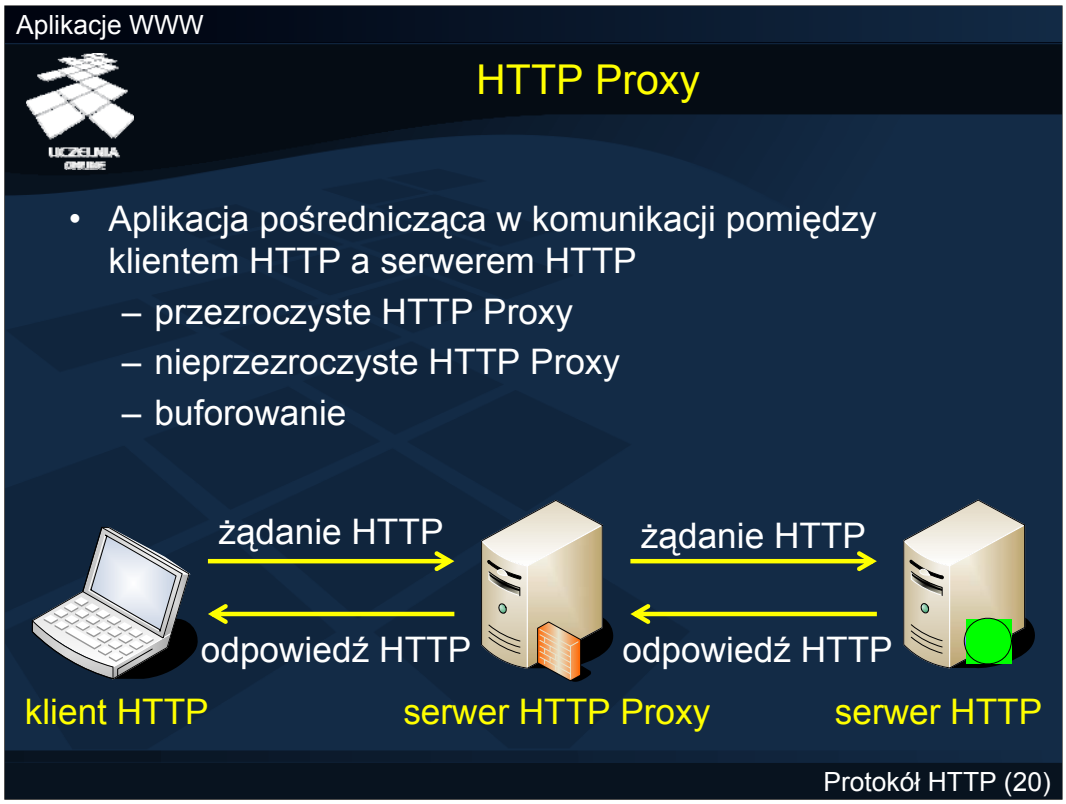
Wraz z kopiami dokumentów klient HTTP zapisuje w buforze ich identyfikatory wersji, a następnie formułuje warunkowe żądania HTTP, informując serwer HTTP o posiadanej wersji dokumentu. Identyfikator wersji dokumentu jest umieszczany w polu nagłówkowym If-None-Match. Serwer HTTP porównuje identyfikator wersji otrzymany od klienta HTTP z aktualnym identyfikatorem wersji oryginalnego dokumentu i bądź przesyła nową postać dokumentu (jeżeli był on modyfikowany), bądź też odpowiada kodem zwrotnym 304 (Not Modified), zezwalając klientowi HTTP na użycie kopii z bufora.



Ze względu na specyfikę języka HTML często zdarza się, że w celu obsługi pojedynczego żądania użytkownika końcowego konieczne jest wykonanie przez klienta HTTP całej serii żądań HTTP. Przykładowo, gdy użytkownik pobiera stronę HTML zawierającą dziesięć fotografii, to klient HTTP musi łącznie zażądać jedenastu oddzielnych plików. Ponieważ podstawowym założeniem protokołu HTTP jest realizowanie jednego żądania HTTP w ramach jednego połączenia sieciowego, to klient HTTP musiałby jedenastokrotnie nawiązywać połączenie sieciowe z serwerem HTTP. Powoduje to duże narzuty czasowe i w konsekwencji wydłużenie czasu odpowiedzi na żądanie użytkownika końcowego.

W specyfikacji 1.1 protokołu HTTP zaproponowano mechanizm połączeń sieciowych o przedłużonej żywotności (Persistent Connections, Keep-Alive). Polega on na nie zamykaniu połączenia sieciowego bezpośrednio po odebraniu odpowiedzi HTTP, lecz na umożliwieniu klientowi HTTP wysłania kolejnego żądania HTTP poprzez zestawione już połączenie. Dzięki temu czas pobrania wszystkich składników skomplikowanego dokumentu ulega skróceniu.

Ponieważ mechanizm Persistent Connections stwarza zagrożenie atakami typu Denial of Service, to zwykle serwery HTTP ograniczają zarówno liczbę żądań HTTP, jakie wolno przesłać w ramach raz zestawionego połączenia, jak i czas bezczynności połączenia otwartego przez klienta HTTP.



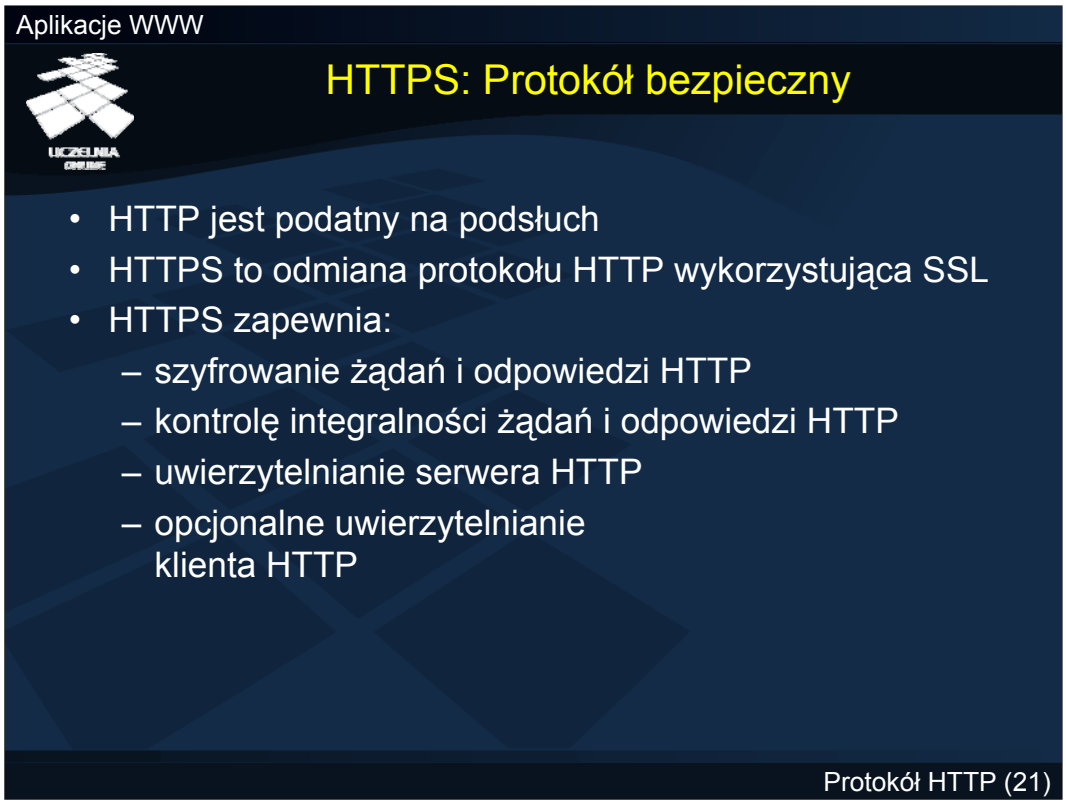
Serwer HTTP Proxy to opcjonalna aplikacja pośrednicząca w komunikacji pomiędzy klientem HTTP a serwerem HTTP. Żądania HTTP klienta są przekazywane do serwera HTTP Proxy, który następnie przekazuje je do zdalnego serwera HTTP. Odpowiedzi HTTP przybywają ze zdalnego serwera HTTP do serwera HTTP Proxy, a następnie są przekazywane do klienta HTTP. Serwer HTTP Proxy może pełnić rolę zapory ogniowej (Firewall), blokującej połączenia z wybranymi serwerami HTTP, lub bufora dokumentów, umożliwiającego skrócenie czasu odpowiedzi na żądanie użytkownika. Często też serwery HTTP Proxy są wykorzystywane do umożliwienia korzystania z zasobów sieci Internet użytkownikom pracującym w sieci lokalnej.

Serwery HTTP dzieli się na dwie główne kategorie:

1. Przezroczyste (transparent HTTP proxy) - nie modyfikują komunikatów żądań HTTP ani odpowiedzi HTTP przekazywanych im przez klientów HTTP i serwery HTTP.

2. Nieprzezroczyste (non-transparent HTTP proxy) - dokonują modyfikacji pól nagłówków lub odpowiedzi HTTP, modyfikacji adresów żądań lub modyfikacji ciała żądania/odpowiedzi HTTP (np. automatyczne tłumaczenie treści pobieranego dokumentu z języka obcego)

Zwykle korzystanie z serwera HTTP Proxy odbywa się w wyniku świadomej decyzji użytkownika końcowego, który dokonuje odpowiedniej konfiguracji aplikacji klienta HTTP. Przykładowo, w aplikacji Microsoft Internet Explorer 6, definicja wykorzystywanego serwera HTTP Proxy znajduje się w menu "Narzędzia->Opcje internetowe->Połączenia->Ustawienia sieci LAN->Serwer proxy".

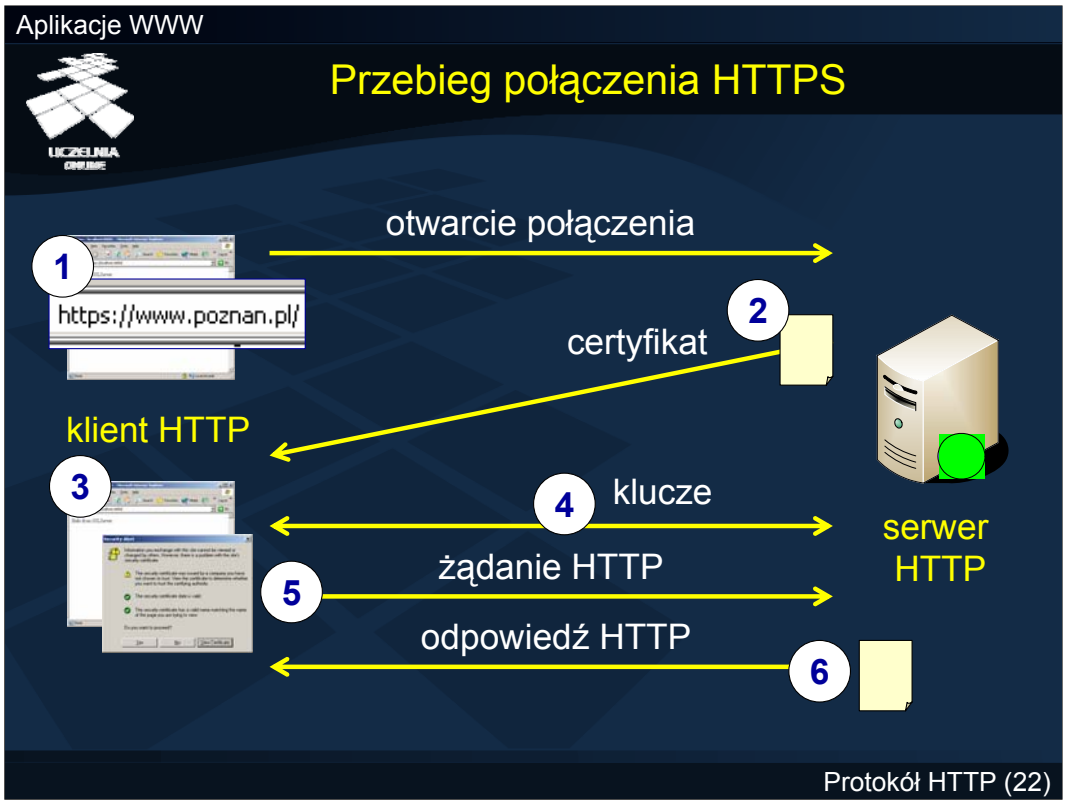


Wadą podstawowego protokołu HTTP jest brak zabezpieczenia poufności komunikacji pomiędzy klientem HTTP a serwerem HTTP. Komendy, parametry, zmienne Cookie i dokumenty są przesyłane w postaci czytelnego tekstu. Hasła uwierzytelniające są kodowane w prymitywny sposób.

W celu zapewnienia bezpieczeństwa komunikacji w środowisku WWW powszechnie stosuje się protokół HTTPS, który w przeciwieństwie do HTTP, opartego bezpośrednio na TCP, wykorzystuje warstwę komunikacyjną SSL. SSL obsługuje szyfrowanie transmisji, kontrolę jej integralności oraz uwierzytelnianie serwera, a opcjonalnie także klienta. Dzięki temu komunikaty żądań HTTP i odpowiedzi HTTP nie są narażone na łatwy podsłuch lub ingerencję osób trzecich.

Podczas nawiązywania połączenia HTTPS klient HTTP otrzymuje od serwera HTTP certyfikat klucza publicznego, na podstawie którego potwierdza się tożsamość serwera HTTP. Aby certyfikat ten został uznany przez klienta HTTP, wystawiający go urząd certyfikacyjny powinien zostać uprzednio zarejestrowany w aplikacji klienta HTTP. W przeciwnym razie certyfikat serwera HTTP będzie uznany za nielegalny. Przykładowo, w aplikacji Microsoft Internet Explorer 6, definicje uznawanych urzędów certyfikacyjnych znajdują się w menu "Narzędzia->Opcje internetowe->Zawartość->Certyfikaty->Zaufane główne urzędy certyfikacji".

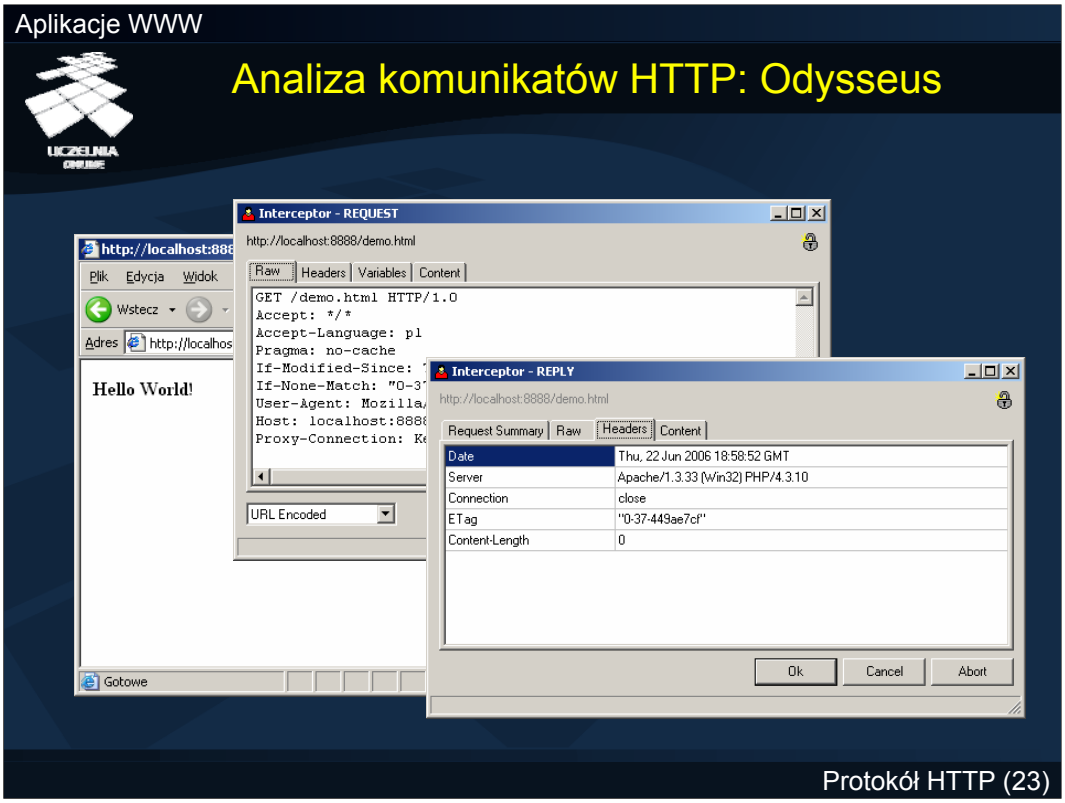
Protokół HTTPS stwarza również możliwość uwierzytelniania użytkownika końcowego za pomocą jego certyfikatu klucza publicznego. W takim przypadku serwer HTTP żąda od klienta HTTP przesłania certyfikatu użytkownika. Certifikat użytkownika musi zostac uprzednio zarajestrowany w aplikacji klienta HTTP. Przykładowo, w aplikacji Microsoft Internet Explorer 6, definicje uznawanych urzędów certyfikacyjnych znajdują się w menu "Narzędzia->Opcje internetowe->Zawartość->Certyfikaty->Osobisty".



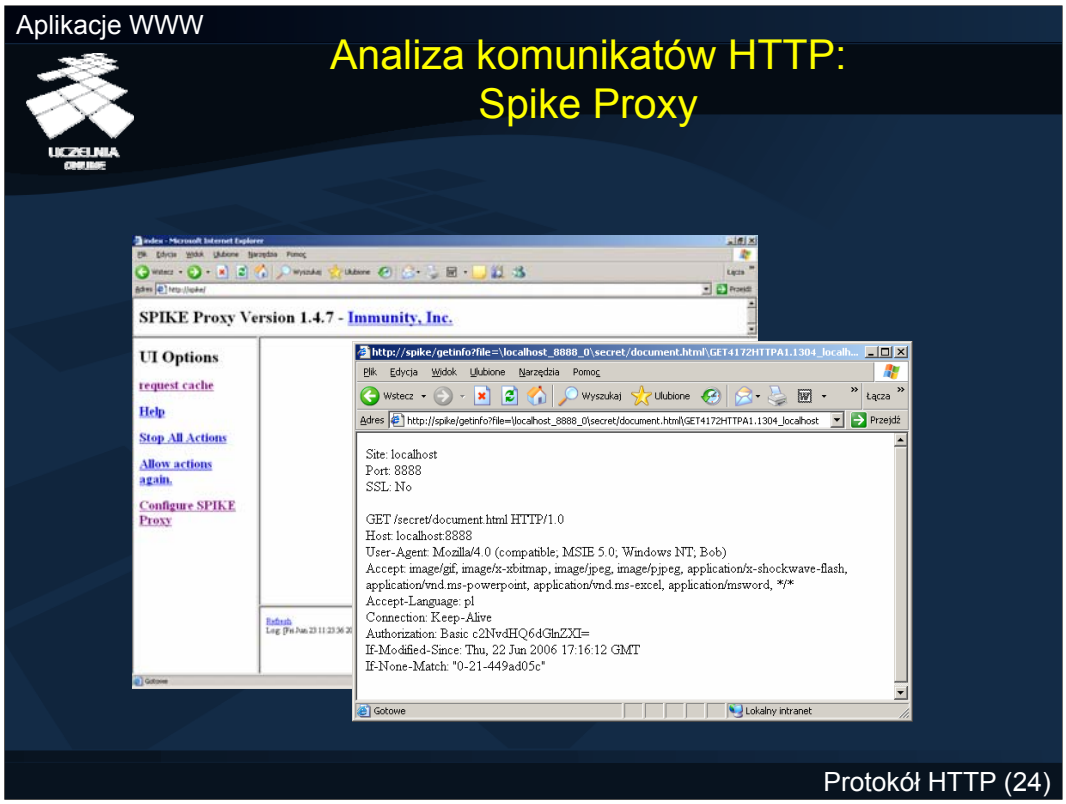
W pewnym uproszczeniu, połączenie HTTPS przebiega w następujących krokach:

1. Klient HTTP wysyła do serwera HTTP sygnał otwarcia połączenia SSL. W tym celu użytkownik końcowy posługuje się symbolem "https" (zamiast "http") w treści adresu URL.
2. Serwer HTTP wysyła do klienta HTTP swój certyfikat klucza publicznego, stanowiący poświadczenie tożsamości serwera. Nazwa właściciela certyfikatu powinna być identyczna z nazwą DNS serwera HTTP.
3. Klient HTTP dokonuje weryfikacji otrzymanego certyfikatu:
   - czy nazwa właściciela zgadza się z nazwą DNS serwera HTTP?
   - czy wystawca certyfikatu należy do zbioru zaufanych urzędów certyfikacyjnych (konfiguracja aplikacji klienta HTTP)?
   - czy data ważności certyfikatu nie wygasła?

   Jeżeli wynik weryfikacji certyfikatu jest negatywny, aplikacja klienta HTTP zwraca się do użytkownika końcowego z pytaniem o kontynuowanie połączenia.
4. Klient HTTP wymienia z serwerem HTTP klucz szyfrujący.
5. Klient HTTP szyfruje komunikat żądania HTTP i przesyła go do serwera HTTP.
6. Serwer HTTP szyfruje komunikat odpowiedzi HTTP i przesyła go do klienta HTTP. Połączenie jest zamykane.

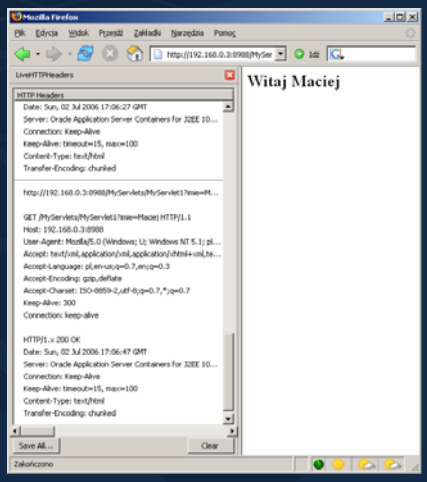


W celu praktycznego zapoznania się z wewnętrznymi mechanizmami funkcjonowania protokołu HTTP warto skorzystać z oprogramowania monitorującego klasy Local Proxy, służącego do monitorowania transmisji HTTP. Przykładem takiego produktu jest dostępny bezpłatnie Odysseus firmy Wastelands Technologies (www.wastelands.gen.nz). Odysseus funkcjonuje jako serwer HTTP Proxy, przechwytując zarówno żądania HTTP wysyłane przez lokalną przeglądarkę internetową, jak i przybywające z serwera odpowiedzi HTTP. Przechwycone komunikaty są wyświetlane na ekranie w postaci surowej oraz ustrukturalizowanej. Użytkownik ma możliwość nie tylko obejrzenia komunikatów, ale także ingerowania w ich treści.

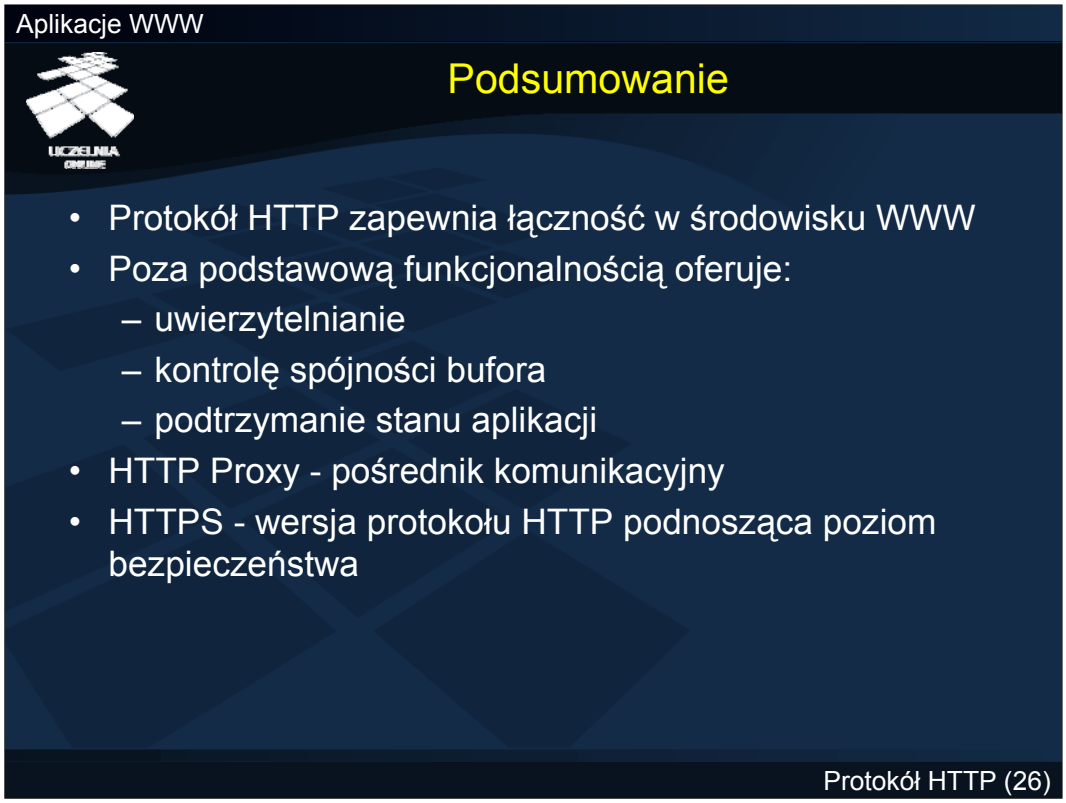


Podobnym narzędziem służącym do monitorowania transmisji HTTP jest bezpłatny Spike Proxy (www.immunitysec.com). Spike Proxy jest zaimplementowany w języku Python, funkcjonuje jako serwer HTTP Proxy i rejestruje przesłania HTTP. Umożliwia późniejszy odczyt zarejestrowanych żądań i odpowiedzi HTTP, a także ich modyfikację i ponowne wykonanie. Spike Proxy może również służyć do badania serwerów HTTP pod kątem ich podatności na włamania.

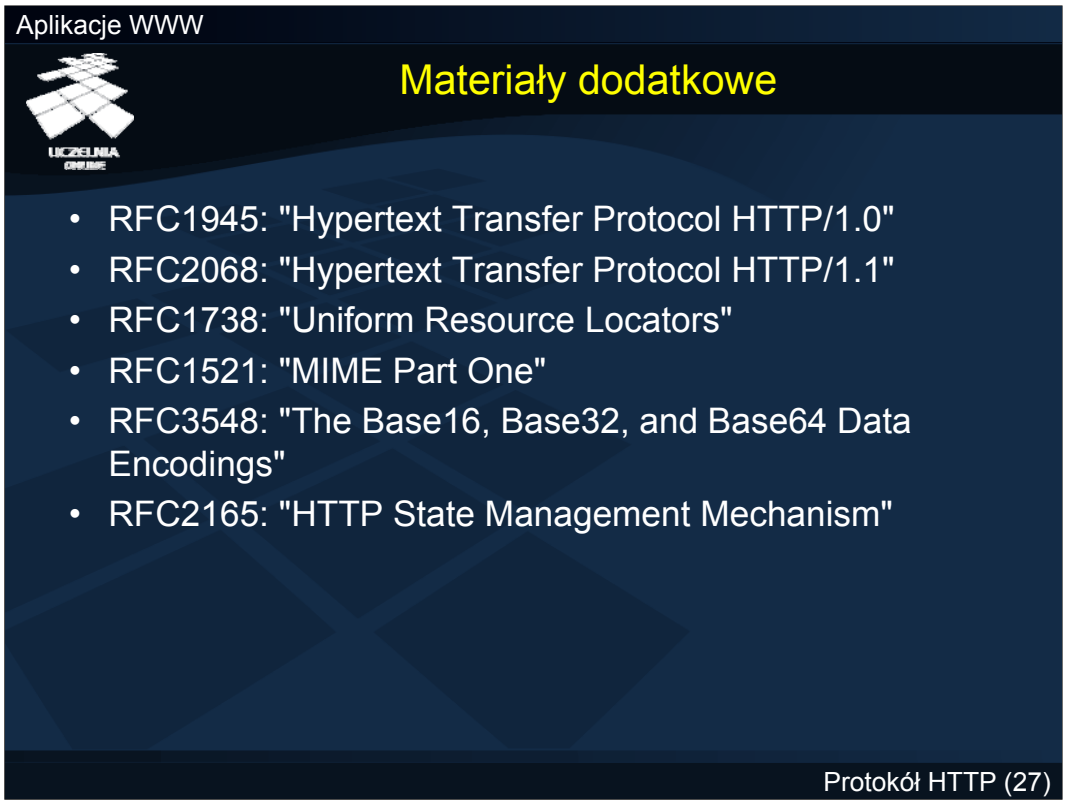
Analiza komunikatów HTTP: Mozilla LiveHTTPHeaders



Podgląd komunikatów HTTP jest również możliwy za pomocą nakładki LiveHTTPHeaders przeznaczonej dla przeglądarki Mozilla Firefox (http://livehttpheaders.mozdev.org/). Po uaktywnieniu tego narzędzia komunikaty żądań i odpowiedzi HTTP są wyświetlane w panelu bocznym przeglądarki. Nie jest możliwa ich modyfikacja.



Protokół HTTP jest zasadniczym składnikiem architektury WWW. Jego podstawowa funkcjonalność dotyczy obsługi dwufazowej komunikacji, w ramach której klient HTTP wysyła żądanie HTTP, a otrzymuje odpowiedź HTTP. Oprócz swojej podstawowej funkcjonalności, protokół HTTP oferuje również mechanizmy wspomagające realizację uwierzytelniania użytkowników końcowych za pomocą nazwy i hasła, mechanizmy kontroli spójności kopii dokumentów buforowanych przez klienta HTTP oraz mechanizmy podtrzymywania stanu aplikacji za pomocą zmiennych Cookies. Architektura komunikacyjna HTTP może być wzbogacona przez pośredniczące serwery HTTP Proxy. W celu podniesienia poziomu bezpieczeństwa komunikacji stosuje się rozszerzoną wersję protokołu HTTP nazywaną HTTPS.



•RFC1945: "Hypertext Transfer Protocol HTTP/1.0"
•RFC2068: "Hypertext Transfer Protocol HTTP/1.1"
•RFC1738: "Uniform Resource Locators"
•RFC1521: "MIME Part One"
•RFC3548: "The Base16, Base32, and Base64 Data Encodings"
•RFC2165: "HTTP State Management Mechanism"