HTTP Protocol Basics

3.2 HTTP Protocol Basics

+ **How does this support my pentesting career?**

  - The ability to exploit web applications and find vulnerabilities in web servers and services
  - Web applications technology is used market-wide also by desktop or mobile applications

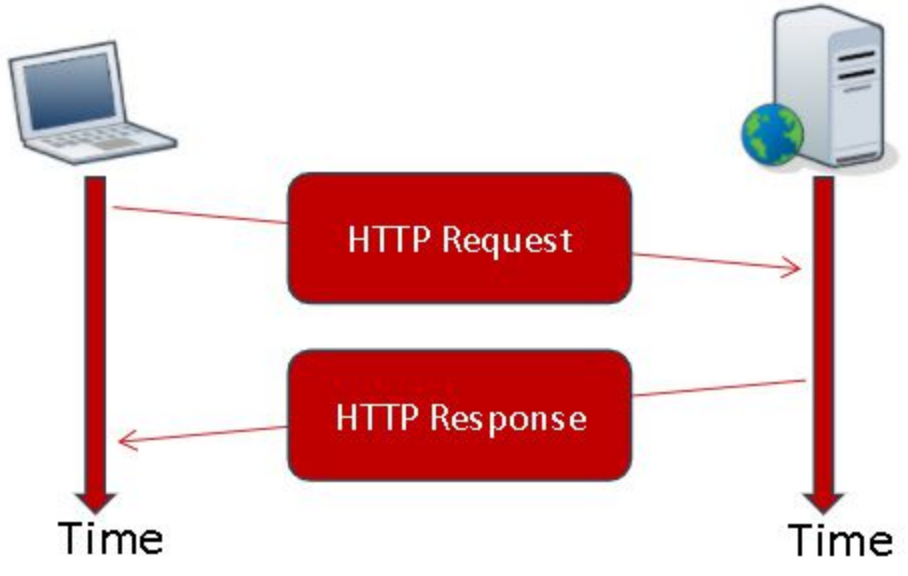
3.2 HTTP Protocol Basics

- **Hypertext Transfer Protocol** (HTTP) is the most used application protocol on the Internet. It is the client-server protocol used to transfer web pages and web application data.

- In HTTP, the client, usually a web browser, connects to a web server such as **MS IIS** or **Apache HTTP Server**. HTTP is also used under the hood by many mobile and modern applications.

3.2 HTTP Protocol Basics

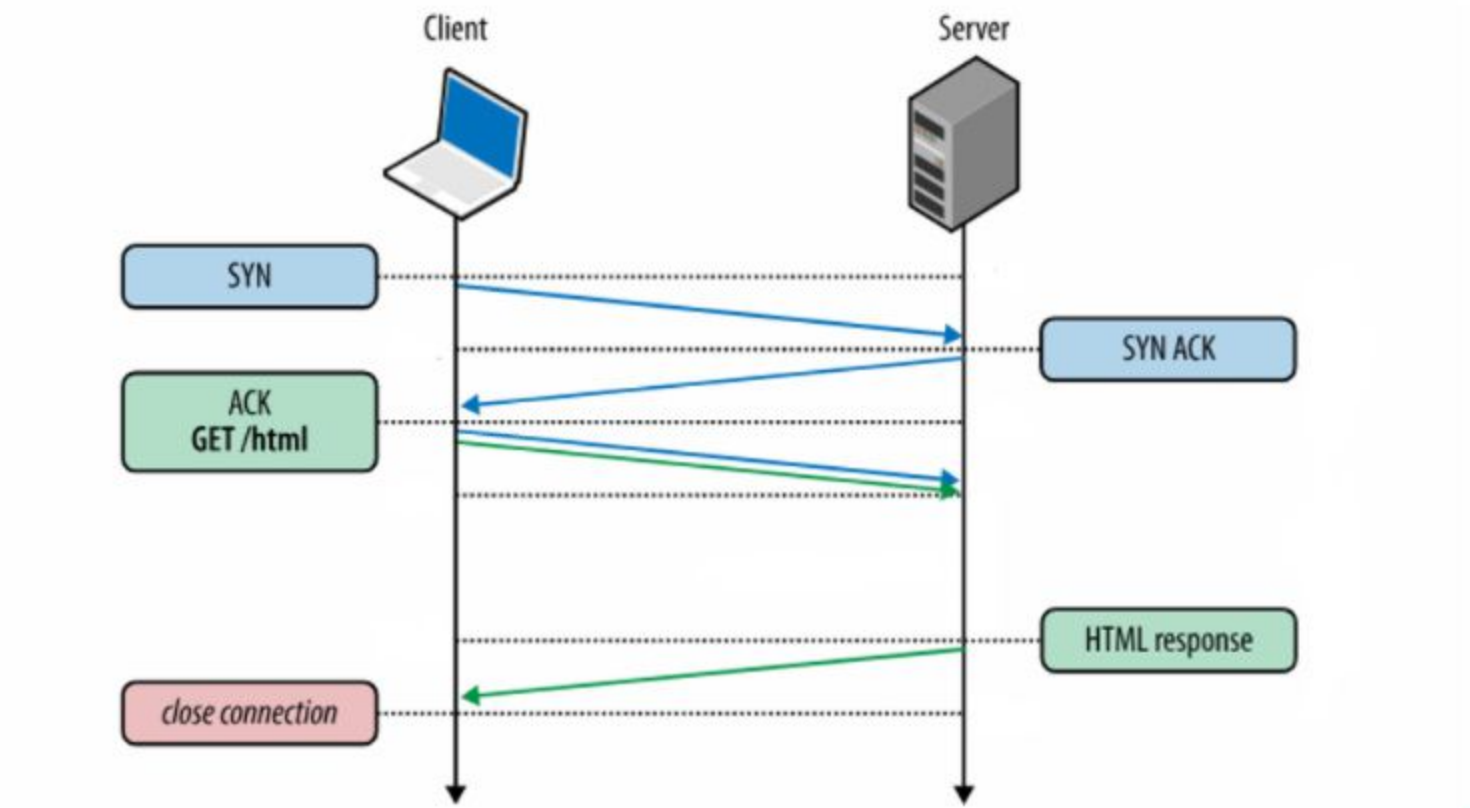
- During an HTTP communication, the client and the server exchange **messages.**
- The client sends **requests** to the server and gets back **responses.**



3.2 HTTP Protocol Basics

- HTTP works on top of TCP protocol.
- That means, first a TCP connection is established, and then the client sends its request, and waits for the answer. The server processes the request and sends back its answer, providing a status code and appropriate data.

3.2 HTTP Protocol Basics



https://hpbn.co/http1x/

3.2 HTTP Protocol Basics

- The format of an HTTP message is:

3.2 HTTP Protocol Basics

- To end lines in HTTP, you have to use the `\r` (carriage return) and the `\n` (newline) characters.

- The header contains a request followed by some header fields. Every header field has the following format:

```
Header-name: header value
```

3.2.1 HTTP Requests

- The following is an HTTP request example.

```
GET / HTTP/1.1
Host: www.elearnsecurity.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:31.0) Gecko/20100101 Firefox/31.0
Iceweasel/31.2.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

3.2.1 HTTP Requests

+ This request has an empty body, as there is nothing after the two empty lines following the headers.

```
GET / HTTP/1.1
Host: www.elearnsecurity.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:31.0) Gecko/20100101 Firefox/31.0
Iceweasel/31.2.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive

```

3.2.1 HTTP Requests

- This is the **HTTP verb** of the request.
- The HTTP verb, or request method, states the type of the request.

```
GET / HTTP/1.1
Host: www.elearnsecurity.com
User-Agent: Mozilla/5.0 (X11; Linux
x86_64; rv:31.0) Gecko/20100101
Firefox/31.0 Iceweasel/31.2.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

3.2.1 HTTP Requests

- GET is used when opening web resources.
- If you open a browser and type www.elearnsecurity.com in the address bar, your browser will send this very same request to the server.

```
GET / HTTP/1.1
Host: www.elearnsecurity.com
User-Agent: Mozilla/5.0 (X11; Linux
x86_64; rv:31.0) Gecko/20100101
Firefox/31.0 Iceweasel/31.2.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

3.2.1 HTTP Requests

- After the HTTP VERB you can see the **path** (/) and the **protocol version** (HTTP 1.1).
- The path tells the server which resource the browser is asking for. The protocol version tells the server how to communicate with the browser.

```
GET / HTTP/1.1
Host: www.elearnsecurity.com
User-Agent: Mozilla/5.0 (X11; Linux
x86_64; rv:31.0) Gecko/20100101
Firefox/31.0 Iceweasel/31.2.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

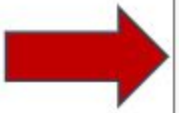
3.2.1 HTTP Requests

- There are many HTTP methods, like:
  - PUT
  - TRACE
  - HEAD
  - POST
- These are only a few but know that there are many more out there.

```
GET / HTTP/1.1
Host: www.elearnsecurity.com
User-Agent: Mozilla/5.0 (X11; Linux
x86_64; rv:31.0) Gecko/20100101
Firefox/31.0 Iceweasel/31.2.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

3.2.1 HTTP Requests

- The **Host** header field specifies the Internet hostname and port number of the resource being requested.
- A web server can host multiple websites. This header field tells the server which site the client is asking for.



```
GET / HTTP/1.1
Host: www.elearnsecurity.com
User-Agent: Mozilla/5.0 (X11; Linux
x86_64; rv:31.0) Gecko/20100101
Firefox/31.0 Iceweasel/31.2.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

3.2.1 HTTP Requests

- The host value is obtained from the URI of the resource.
- In this case:

`www.elearnsecurity.com`

```
GET / HTTP/1.1
Host: www.elearnsecurity.com
User-Agent: Mozilla/5.0 (X11; Linux
x86_64; rv:31.0) Gecko/20100101
Firefox/31.0 Iceweasel/31.2.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

http://www.w3.org/TR/uri-clarification/

3.2.1 HTTP Requests

- **User-Agent** tells the server what client software is issuing the request.
- A client could be:
  - **Firefox**
  - **Internet Explorer**
  - **Safari**
  - **Opera**
  - **Chrome**
  - **A mobile app…**

```
GET / HTTP/1.1
Host: www.elearnsecurity.com
User-Agent: Mozilla/5.0 (X11; Linux
x86_64; rv:31.0) Gecko/20100101
Firefox/31.0 Iceweasel/31.2.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

3.2.1 HTTP Requests

- It also reveals to the server the operating system version.

```
GET / HTTP/1.1
Host: www.elearnsecurity.com
User-Agent: Mozilla/5.0 (X11; Linux
x86_64; rv:31.0) Gecko/20100101
Firefox/31.0 Iceweasel/31.2.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

3.2.1 HTTP Requests

- The browser sends the Accept header field to specify which document type it is expecting in the response.

```
GET / HTTP/1.1
Host: www.elearnsecurity.com
User-Agent: Mozilla/5.0 (X11; Linux
x86_64; rv:31.0) Gecko/20100101
Firefox/31.0 Iceweasel/31.2.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

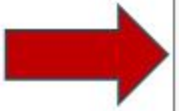
3.2.1 HTTP Requests

- Similarly, with **Accept-Language**, the browser can ask for a specific (human) language in the response.

```
GET / HTTP/1.1
Host: www.elearnsecurity.com
User-Agent: Mozilla/5.0 (X11; Linux
x86_64; rv:31.0) Gecko/20100101
Firefox/31.0 Iceweasel/31.2.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

3.2.1 HTTP Requests

- **Accept-Encoding** works similarly to Accept but restricts the content encoding, not the content itself.
- In this case, the browser accepts two types of compression:
  - gzip
  - deflate



```
GET / HTTP/1.1
Host: www.elearnsecurity.com
User-Agent: Mozilla/5.0 (X11; Linux
x86_64; rv:31.0) Gecko/20100101
Firefox/31.0 Iceweasel/31.2.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

3.2.1 HTTP Requests

- The **Connection** header field allows the sender to specify options that are desired for that particular connection.
- Future communications with the server will reuse the current connection.

```
GET / HTTP/1.1
Host: www.elearnsecurity.com
User-Agent: Mozilla/5.0 (X11; Linux
x86_64; rv:31.0) Gecko/20100101
Firefox/31.0 Iceweasel/31.2.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

3.2.2 HTTP Responses

+ When the server receives a request, it processes it and then sends an **HTTP response** to the client. The response has its own header format.

```
HTTP/1.1 200 OK
Date: Wed, 19 Nov 2014 10:06:45 GMT
Cache-Control: private, max-age=0
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip
Server: Apache/2.2.15 (CentOS)
Content-Length: 99043


< PAGE CONTENT > ...
```

3.2.2 HTTP Responses

- As you can see, this response has a message body (< PAGE CONTENT >). The header and message body are separated by two empty lines (\r\n\r\n).

```
HTTP/1.1 200 OK
Date: Wed, 19 Nov 2014 10:06:45 GMT
Cache-Control: private, max-age=0
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip
Server: Apache/2.2.15 (CentOS)
Content-Length: 99043

< PAGE CONTENT > ...
```

3.2.2 HTTP Responses

- The first line of a Response message is the **Status-Line**, which consists of the **protocol version** (HTTP 1.1) followed by a numeric **status code** (200) and its relative **textual meaning** (OK).

```
HTTP/1.1 200 OK
Date: Wed, 19 Nov 2014 10:06:45 GMT
Cache-Control: private, max-age=0
Content-Type: text/html;
charset=UTF-8
Content-Encoding: gzip
Server: Apache/2.2.15 (CentOS)
Content-Length: 99043


< PAGE CONTENT > ...
...
```

3.2.2 HTTP Responses

The more common status codes are:

- **200 OK**: the resource is found.
- **301 Moved Permanently**: the requested resource has been assigned a new permanent URI.
- **302 Found**: the resource is temporarily under another URI.

```
HTTP/1.1 200 OK
Date: Wed, 19 Nov 2014 10:06:45 GMT
Cache-Control: private, max-age=0
Content-Type: text/html;
charset=UTF-8
Content-Encoding: gzip
Server: Apache/2.2.15 (CentOS)
Content-Length: 99043


< PAGE CONTENT > ...
...
```

3.2.2 HTTP Responses

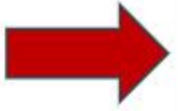
- **403 Forbidden:** the client does not have enough privileges, and the server refuses to fulfill the request.
- **404 Not Found:** the server cannot find a resource matching the request.
- **500 Internal Server Error:** the server does not support the functionality required to fulfill the request.

```
HTTP/1.1 200 OK
Date: Wed, 19 Nov 2014 10:06:45 GMT
Cache-Control: private, max-age=0
Content-Type: text/html;
charset=UTF-8
Content-Encoding: gzip
Server: Apache/2.2.15 (CentOS)
Content-Length: 99043


< PAGE CONTENT > ...
...
```

3.2.2 HTTP Responses

- **Date** represents the date and time at which the message was originated.



```
HTTP/1.1 200 OK
Date: Wed, 19 Nov 2014 10:06:45 GMT
Cache-Control: private, max-age=0
Content-Type: text/html;
charset=UTF-8
Content-Encoding: gzip
Server: Apache/2.2.15 (CentOS)
Content-Length: 99043


< PAGE CONTENT > ...
...
```

3.2.2 HTTP Responses

- With the **Cache-Control** header, the server informs the client about cached content.
- Using cached content saves bandwidth, as it prevents the client from re-requesting unmodified content.

```
HTTP/1.1 200 OK
Date: Wed, 19 Nov 2014 10:06:45 GMT
Cache-Control: private, max-age=0
Content-Type: text/html;
charset=UTF-8
Content-Encoding: gzip
Server: Apache/2.2.15 (CentOS)
Content-Length: 99043


< PAGE CONTENT > ...
...
```

3.2.2 HTTP Responses

- **Content-Type** lets the client know how to interpret the body of the message.

```
HTTP/1.1 200 OK
Date: Wed, 19 Nov 2014 10:06:45 GMT
Cache-Control: private, max-age=0
Content-Type: text/html;
charset=UTF-8
Content-Encoding: gzip
Server: Apache/2.2.15 (CentOS)
Content-Length: 99043


< PAGE CONTENT > ...
...
```

3.2.2 HTTP Responses

- **Content-Encoding** extends Content-Type.
- In this case, the message body is compressed with gzip.

```
HTTP/1.1 200 OK
Date: Wed, 19 Nov 2014 10:06:45 GMT
Cache-Control: private, max-age=0
Content-Type: text/html;
charset=UTF-8
Content-Encoding: gzip
Server: Apache/2.2.15 (CentOS)
Content-Length: 99043


< PAGE CONTENT > ...
...
```

3.2.2 HTTP Responses

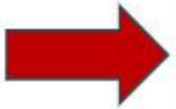
- The **Server** header field simply contains the header of the server that generated the content.
- This (optional) field is very useful during a pentest to identify the software running on a server.

```
HTTP/1.1 200 OK
Date: Wed, 19 Nov 2014 10:06:45 GMT
Cache-Control: private, max-age=0
Content-Type: text/html;
charset=UTF-8
Content-Encoding: gzip
Server: Apache/2.2.15 (CentOS)
Content-Length: 99043


< PAGE CONTENT > ...
...
```

3.2.2 HTTP Responses

- **Content-Length** indicates the length, in bytes, of the message body.



```
HTTP/1.1 200 OK
Date: Wed, 19 Nov 2014 10:06:45 GMT
Cache-Control: private, max-age=0
Content-Type: text/html;
charset=UTF-8
Content-Encoding: gzip
Server: Apache/2.2.15 (CentOS)
Content-Length: 99043


< PAGE CONTENT > ...
...
```

3.2.3 HTTPS

- Now that you know how HTTP works, let's see how to **protect it**!

- HTTP content, as in every clear-text protocol, can be easily intercepted or mangled by an attacker on the path. Moreover, HTTP does not provide strong authentication between the parties.

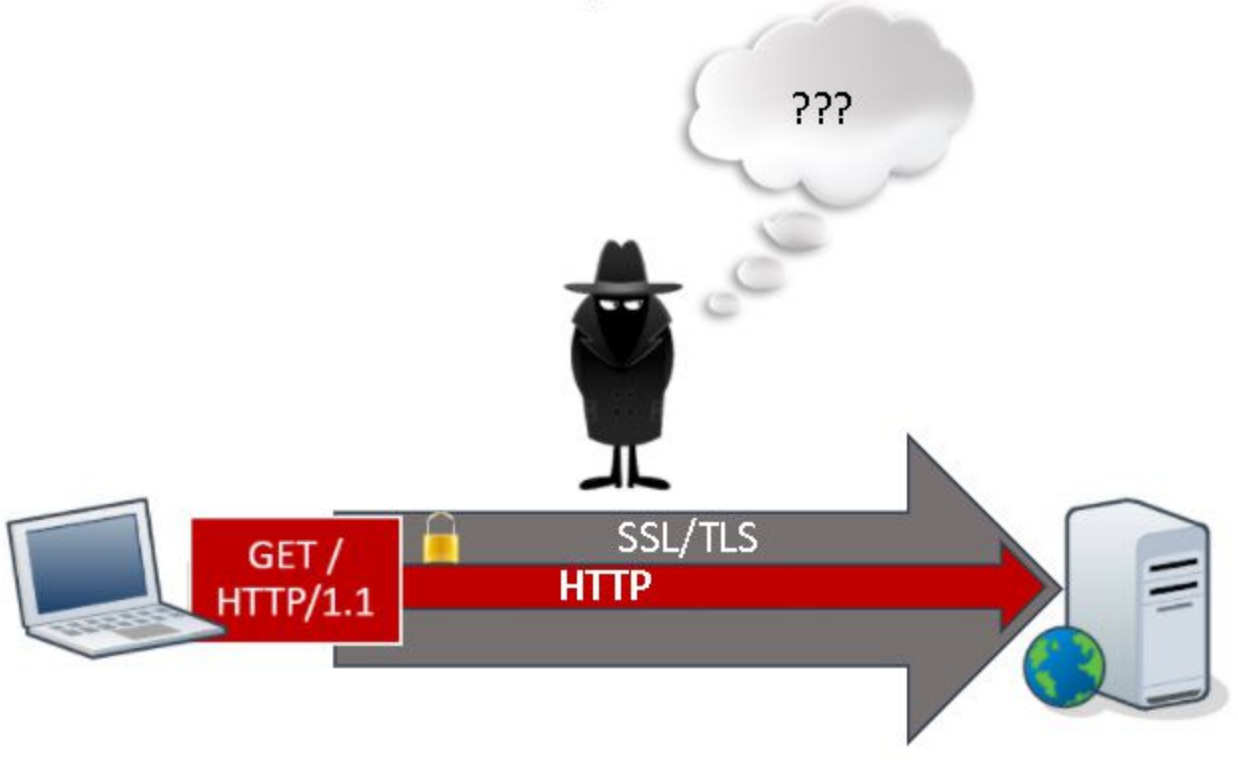
3.2.3 HTTPS

- In the following slides, you will see how to **protect HTTP** using an **encryption layer**.

- **HTTP Secure** (HTTPS), or HTTP over SSL/TLS, is a method to run HTTP which is a clear-text protocol over SSL/TLS, a cryptographic protocol.

3.2.3 HTTPS

+ This layering technique provides confidentiality, integrity protection and authentication to the HTTP protocol.



3.2.3 HTTPS

+ In other words, when using HTTPS:
  - An attacker on the path cannot sniff the application layer communication.
  - An attacker on the path cannot alter the application layer data.
  - The client can tell the real identity of the server and, sometimes, vice-versa.

3.2.3 HTTPS

- HTTPS offers encryption, which means that a network adjacent user is able to sniff the traffic, but he will not know:
  - HTTP Request headers, body, target domain
  - HTTP Response headers, body

- On the other hand, when inspecting HTTPS, one cannot know what domain is contacted and what data is exchanged.

3.2.3 HTTPS

+ A network adjacent user might recognize:
  - Target IP address
  - Target port
  - DNS or similar protocols may disclose which domain user tries to resolve

3.2.3 HTTPS

- HTTPS **does not protect against web application flaws**! All the attacks against an application happen regardless of SSL/TLS.

- The extra encryption layer just protects data exchanged between the client and the server. It does not protect from an attack against the application itself.

3.2.3 HTTPS

- Attacks such as XSS and SQL injections will still work.
- Understanding how HTTP and web applications work is fundamental to mount stealthy and effective attacks!

References

- HTTP Status Code Reference: a document referencing HTTP status codes and their meaning.
- SSL/TLS Strong Encryption: An Introduction: Apache's introduction on protecting HTTP by means of SSL/TLS.
- HTTP Overview, History, Versions and Standards: History and evolution of HTTP.

http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html
http://httpd.apache.org/docs/2.2/ssl/ssl_intro.html
http://www.tcpipguide.com/free/t_HTTPOverviewHistoryVersionsandStandards.htm

References

+ HTTP/1.X: https://hpbn.co/http1x/

+ URI: http://www.w3.org/TR/uri-clarification/