# WS1: Introduction to Web Security

Terms and Concepts

- **Hypertext**: formatted text designed for electronic display that contains references ("hyperlinks") to other texts and multimedia content (images, sounds, etc.).
- **World Wide Web (WWW)**: a global network of interlinked hypertext documents; a subset of the Internet.
- **Protocol**: a set of rules governing data transmission between two computing components (hardware or software), e.g., HTTP.
- **Encoding**: a way to represent information.

  Examples of digital encodings:

  - Images: JPG, PNG, etc.
  - Audio: MP3, FLAC, etc.
  - Text: UTF-8, ASCII, ISO-8859-2, etc.
  - Binary data: Base64, etc.
  - Hypertext: HTML, etc.

  **Note**: there are also analog encodings of digital data, such as frequency, amplitude, and/or phase modulation of a signal for transmitting bits in modems or specific voltages (e.g., 0/5V for logical values 0/1) in CMOS logic gates.

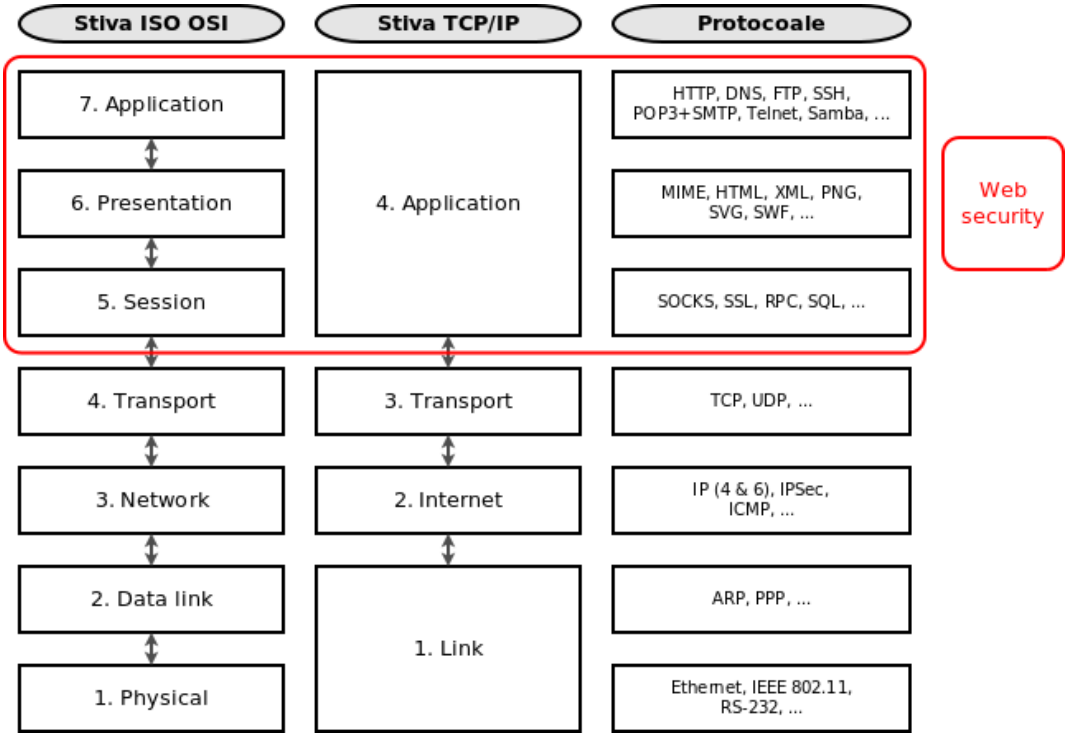
- **Encryption**: encoding that hides information from unauthorized entities (those without the decryption key).

  Important: **encryption ≠ encoding**

- **File Format**: a protocol for encoding data in a file. Typically, files of a certain format have a specific extension, but this is not mandatory (e.g., a file named ana.txt may contain an image in PNG format).

# Web Protocols

## ISO OSI and TCP/IP Stacks




## HTTP Protocol

- Client-server model (browser - web server)
- Request-response mechanism, initiated by the client
- Status codes (numerical encoding of results):
  - 2xx: Success (e.g., 200 = OK)
  - 3xx: Redirection (e.g., 301 = Moved Permanently, 302 = Temporary Redirect)
  - 4xx: Client error (e.g., 400 = Bad Request, 401 = Unauthorized - authentication, 403 = Forbidden - authorization, 404 = Not Found)
  - 5xx: Server error (e.g., 500 = Internal Server Error, 501 = Not Implemented, 503 = Service Unavailable)
  - Standardized statuses: RFC 2616

Request exmple:

```
1.  $ nc example.com 80
2.  GET / HTTP/1.1
3.  Host: example.com
```

Response example:

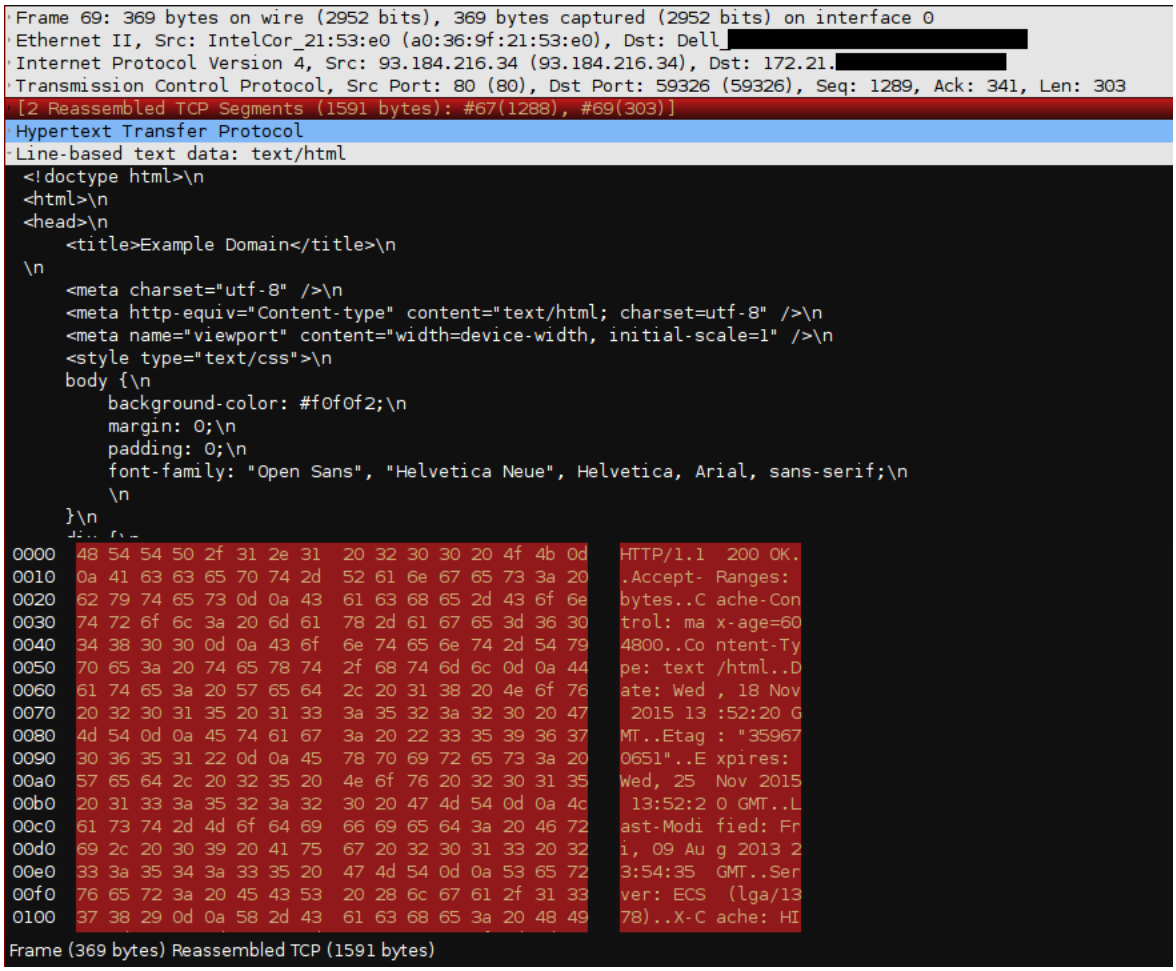
```
HTTP/1.1 200 OK
Accept-Ranges: bytes
Cache-Control: max-age=604800
Content-Type: text/html
Date: Thu, 19 Nov 2015 12:57:34 GMT
Etag: "359670651"
Expires: Thu, 26 Nov 2015 12:57:34 GMT
Last-Modified: Fri, 09 Aug 2013 23:54:35 GMT
Server: ECS (lga/13BA)
X-Cache: HIT
x-ec-custom-error: 1
Content-Length: 1270

<!doctype html>
<html>
<head>
<title>Example Domain</title>
[...]
</html>
```

Example using Curl:

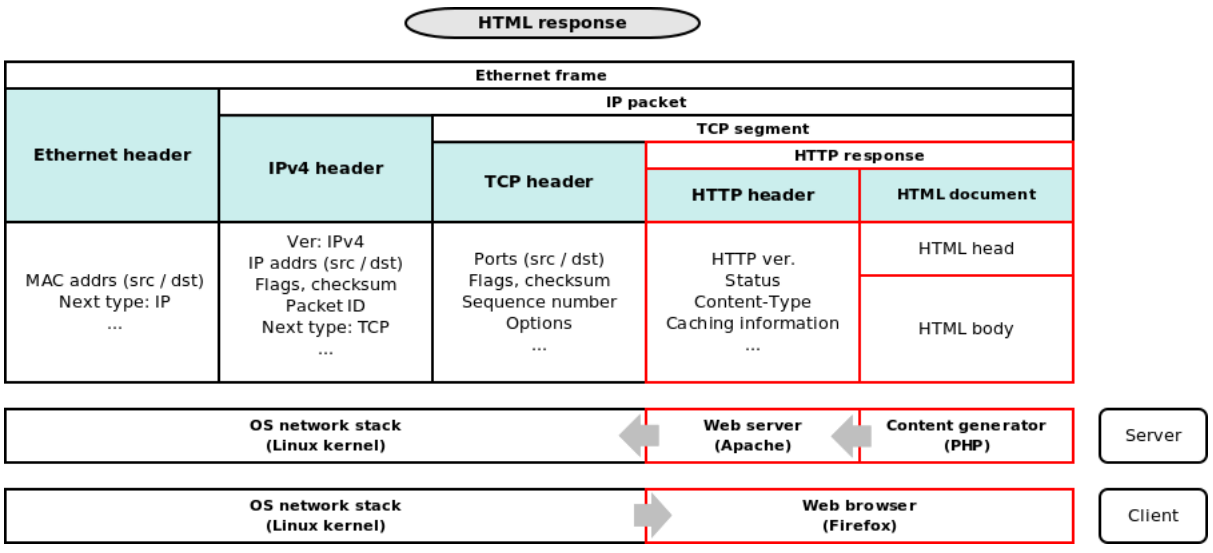
```
$ curl -v example.com
* Rebuilt URL to: example.com/
* Trying 93.184.216.34...
* Connected to example.com (93.184.216.34) port 80 (#0)

GET / HTTP/1.1
Host: example.com
User-Agent: curl/7.43.0
Accept: */*

HTTP/1.1 200 OK
Accept-Ranges: bytes
Cache-Control: max-age=604800
Content-Type: text/html
Date: Thu, 19 Nov 2015 12:57:50 GMT
Etag: "359670651"
Expires: Thu, 26 Nov 2015 12:57:50 GMT
Last-Modified: Fri, 09 Aug 2013 23:54:35 GMT
Server: ECS (ewr/15BD)
Vary: Accept-Encoding
X-Cache: HIT
x-ec-custom-error: 1
Content-Length: 1270

<!doctype html>
<html>
<head>
<title>Example Domain</title>
[...]
</html>
```

# Network Packet Structure

## Example - Wireshark capture




## Diagram: packets structure



## Unauthorized data modification (data tampering)

Tamper Dev is a browser add-on for Chromium-based browsers that intercepts communication between the client and web server, allowing modification of message content before it is sent to or received from the server. A user-initiated request from the browser can be modified using Tamper Dev and then sent to the server or canceled (default action).

Lab Exercises

1. Connect using netcat to **www.utcluj.ro:80**, send a GET request to fetch data from /universitatea/educatie/, and explain the result. Obtain the same content using wget and compare results. Follow the instructions from the page accessed with netcat; what is the result page containing?

   **Hint:** you can use html2text to remove HTML tags and see only the actual text content from the document.

2. Write the following HTML code, save it in /var/www/html as test.html, and open it in a browser:

```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.    <meta charset="utf-8">
5.    <title>Login Form</title>
6.  </head>
7.  <body>
8.    <section class="container">
9.      <div class="login">
10.       <h1>Login to Web App</h1>
11.       <form method="get" action="index.html">
12.         <p><input type="text" name="login" placeholder="Username or Email"></p>
13.         <p><input type="password" name="password" placeholder="Password"></p>
14.         <p class="submit"><input type="submit" name="commit" value="Login"></p>
15.       </form>
16.     </div>
17.   </section>
18. </body>
19. </html>
```

   Access the page using a browser: http://localhost/test.html. Click the Login button, intercept the request using Tamper Dev, and analyze the URL and headers. Modify the request type from GET to POST, intercept the request again, and explain the differences.

   **Hint:** to complete this exercise, you need a locally installed web server. Installing and configuring a complex server (e.g., Apache) is not necessary; instead, you can use built-in extensions from Python or Node. For example, to serve resources from a directory using Python, run the following command in that directory: **python3 -m http.server**. This will start a mini web server on the default port 8000. You can access the web page by navigating to: http://localhost:8000/

3. Authenticate on https://websinu.utcluj.ro and analyze transmitted packets using Wireshark. Observe username and password parameters. Perform the same research using Tamper Dev and explain differences between browser-level parameters and intercepted packets.

## References


- ISO OSI: https://en.wikipedia.org/wiki/OSI_model
- HTTP 1.1 standard: https://datatracker.ietf.org/doc/html/rfc2616
- HTTP Status Codes: https://en.wikipedia.org/wiki/List_of_HTTP_status_codes
- Tamper Dev Tamper Dev