

Innleveringsoppgave

Nettverk og sikkerhet/ TSD2090 – 1 25V

01.04.2025

[Salim Zakaria Ahmed/241775]

# Oblig 3 – Implementere en Web Server

# Innholdsfortegnelse

Forside.....	1
Om bruk av KI-verktøy.....	2
Innledning.....	2
Hovedkapitlene.....	2
Kode.....	2
400 Protocol.....	2
Egen filsett.....	3
Publisert filsett.....	4
2 ulike klienter.....	7
Kildekode.....	8
sikkerhetsrisiko og forbedringer .....	10
Konklusjon.....	11
Referanser .....	11
Vedlegg.....	12

# Om bruk av Kl-verktøy

Jeg har ikke bruk nesten noe Kl. Det som er Kl brukt her er at jeg spurte om kilde for en nettside med alle contenttypes så kom den fram til en bra side som var [6] *An Alphabetized List of MIME Types*. Jeg spurte også om struktur på oppgaven.

## Innledning

I denne oppgaven har vi fått i oppdrag å implementere en web server. Jeg har tatt min forrige oblig 1 sin server som inspirasjon. Serveren skal kunne støtte http/1.1 fra en web klient. Den skal kunne finne en metode på å kunne returnere en webserver, som request. Webserveren skal være tilgjengelig for ulike klienter. Ved å skrive inn en URL med riktig IP adresse og port nummer i IRI-laben. Jeg skal skrive en kode i C++ som bruker publiserte og private filsett. Dette skal kunne testes med andre. Serveren skal kunne støtte enkel GET og POST. Senere skal man lytte til datatrafikken med wireshark. Til slutt skal analysere sikkerheten og gi forslag til forbedringer.

## Hovedkapitlene

### Kode:

I denne oppgaven har jeg tatt inspirasjon fra min gamle kode. Jeg har fjernet unødvendige kommentarer og tatt i bruk best practice(Unntatt GETtype og filePOST, sor å spesifisere get og post), som i variabelnavn. Jeg har unngått å bruke globale variable. Som man ser i vedlegget har jeg tatt med oppbygningen til sockaddr\_in, in\_addr og sockaddr for å vise hvordan de hadde vært bygd opp. Deretter lagde jeg en logg funksjon, etterfulgt med en signal\_handler. Så kom det en cleanSocket som er det samme som WSACleanup. Og errorSocket som er som WSAGetLastError, men de kommenterer ut i loggen. Deretter er det en enkel readFile som bruker if tester som filen har blitt lest eller ikke. Dersom den ikke er funnet skal ingenting returneres, men hvis den er der. Blir hele filen lest med en isrdbuf\_iterator. Deretter kommer GETtype. Noe som skal se hvilken type fil det er. Den bruker map for de forskjellige innholdstypene til filene. Den leter etter siste punktum i filbanen, så lese fra punktumet til slutten. For å finne riktig type fil som er oppgitt i mappen. Så kommer filePOST. Den finner hvilken http type det er om det er get eller post. Hvilken filbane og protocol. Man kan se at jeg kan velge om den skal se på hvilken fil og hvilken html fil sin path. Her har jeg brukt 200 serien som er en suksessfull webserver, der post forteller at requesten er sent. Også har jeg brukt 400 serien som er at serveren ikke klarer å fylle request, ønsket til klienten. Jeg har ikke brukt 500 serien siden jeg mener at den ikke er nødvendig her. Den forteller om interne problemer. Videre i koden er vi i main. Der skriver man inn portnummer. Så kommer en signalhandler. Så starter serveren, lage socket, bindet socket, så høre på klient. Til slutt er det en while-løkke som venter på klient sin request, så bruker den filePOST for å håndtere dem. Så lukker serveren og forbindelsen på en ryddig måte. Eneste forskjell mellom oblig3(publisertfilsett) eller EgenFil(egenfilsett) er på filePOST der man nevner hvilken fil og mappe som er brukt.

### 400 Protocol:

Her er hva som kommer dersom den ikke kan ta imot klienten sitt ønske, transfer protocol:



## 404 - ERROR

## Egen filsett:

Jeg har testet med egen fil. Jeg lagde en enkel html fil som skriver ut «Hello World, This Is For Oblig 3».

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Hello World, This Is For Oblig 3</title>
</head>
<body>
    <h1>Hello World, This Is For Oblig 3</h1>
</body>
</html>
```

[1] Learn Basic HTML – Hello World – Part 1, (sist besøkt 30.03.2025)

Så kom min html fil fra EgenFil mappe ut slik ved å gå inn i url med port 8080,  
<http://192.168.41.74:8080>



## Hello World, This Is For Oblig 3

Terminalen logget det slikt:

A screenshot of a terminal window titled "C:\Users\Administrator\source/repos\nettverk3\x64\Debug\nettverk3.exe". The window contains the following text:

```
Enter port: 8080
NSAStartup was successfull
OK, server is running8080
file found
type of file found
pathway posted
sent to client
```

I wireshark kan man se frame altså hva datapakkene inneholder. Internet protocol viser også source adressen som er Ip 192.168.41.74 som er min ip til destinasjon adresse som er 192.168.41.255, som er broadcast der de andre kan lytte til den også. Deretter ser man User datagram protocol, som man ser sender jeg til meg selv derfor er source port 137 og destination port 137 samme. Ethernet viser mac adressene og hvilken portocool som blir brukt. Som man ser på bildet er ipv4 bruk så det står version 4 der. Den refererer til neste stack, så hvis det var brukt ipv6 så hadde det stått version 6 der.

Man se hvordan trafikken ser ut i wireshark:

A screenshot of the Wireshark application. The left pane shows a list of network packets with their details and bytes sections. The details section for the first few packets shows:

```
15267 4340.626754 192.168.41.74 192.168.41.255 NBNS 92 Name query NB WPAD<00>
15268 4340.626996 192.168.41.74 192.168.41.255 NBNS 92 Name query NB WPAD<00>
```

The bytes section shows the raw hex and ASCII data for each packet. The right pane shows the raw hex and ASCII data for the selected packet, which is a UDP segment with a length of 58 bytes. The bytes are shown in pairs of four, with the first byte being FF and the last byte being 00.

```

[Time delta from previous displayed frame: 0.000242000 seconds]
[Time since reference or first frame: 4340.626996000 seconds]
Frame Number: 15268
Frame Length: 92 bytes (736 bits)
Capture Length: 92 bytes (736 bits)
[Frame is marked: False]
[Frame is ignored: False]
[Protocols in frame: eth:ethertype:ip:udp:nbsn]
[Coloring Rule Name: SMB]
[Coloring Rule String: smb || nbss || nbsn || netbios]
` Ethernet II, Src: Intel_ac:52:ba (f4:8c:50:ac:52:ba), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  > Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  > Source: Intel_ac:52:ba (f4:8c:50:ac:52:ba)
  Type: IPv4 (0x0800)
  [Stream index: 0]
` Internet Protocol Version 4, Src: 192.168.41.74, Dst: 192.168.41.255
  A10A      = Version: 4

```

Man kan også se om forespørselen er GET eller POST på http laget:

10599	3825.100873	192.168.41.74	192.168.41.60	HTTP	740	POST	/submit	HTTP/1.1	(application/x-www-form-urlencoded)
10611	3825.225345	192.168.41.74	192.168.41.60	HTTP	484	GET	/favicon.ico	HTTP/1.1	

## Publisert filsett:

Jeg brukte de kommenterte linjene på filePOST i filen. Jeg fikk ut samme logg:

```

C:\Users\Administrator\source\repos\nettverk3\x64\Debug\nettverk3.exe
Enter port: 8080
NSAStart was successful
OK, server is running
8080
file found
type of file found
pathway posted
sent to client

```

Etter at jeg satt inn url adressen med port 8080, <http://192.168.41.74:8080> så kom filene opp på en webserver. De inneholdte en about, index og result fil som er html. Nett\_sikkerhet som er webp fil. Css fil som er kalt styles og er et JavaScript som er kalt script. Webserveren ser slik ut:

Welcome to the Test Webserver

About

**Images**

Local Image:

External Image:

## Simple JavaScript Test

[Click me!](#)

JavaScript not executed yet.

## Test Form with POST

Username:   
Email:   
[Submit \(POST\)](#)

Test Webserver Footer

På about ser man at det er en link som sender deg videre til filen, som da legger til/about.html i filstien:

A screenshot of a browser window. The address bar shows the URL "192.168.41.74:8080/about.html". The page content is a dark header with "About This Test" and a paragraph below it stating "This page is used to test internal linking within the webserver." There is a "Back to Home" link. A footer bar at the bottom says "Test Webserver Footer".

This page is used to test internal linking within the webserver.

[Back to Home](#)

Test Webserver Footer

På javascriptet står det "JavaScript not executed yet". Når man klikker på «click me» ser siden slik ut:

## Simple JavaScript Test

[Click me!](#)

JavaScript is working! Button clicked.

Og når man sender inn POST så ser det slik ut før og etter, der etter har med /post\_result i URL adressen sin. POST adressen sender «POST Request is sent» som viser at det er sent (200 serien):

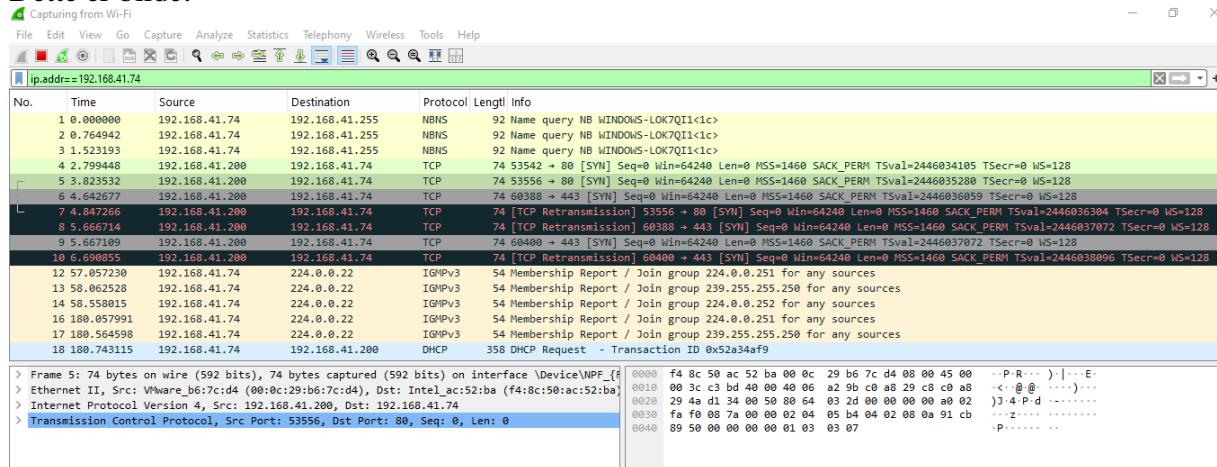
## Test Form with POST

Username:   
Email:   
[Submit \(POST\)](#)

## POST Request is sent

Jeg så videre innpå wireshark for å se trafikken. Den ble sent som TCP. Som forteller kommunikasjon mellom klient og server.

Dette er bilde:



Hvis man ser på Transmission Control Protocol, ser man source port som er 53556 og destinasjons port som er 80. port 80 er default for http port kobling, for å koble seg til internett.

Her ser man det:

Transmission Control Protocol, Src Port: 53556, Dst Port: 80, Seq: 0, Len: 0

- Source Port: 53556
- Destination Port: 80
- [Stream index: 1]
- > [Conversation completeness: Incomplete, SYN\_SENT (1)]
- [TCP Segment Len: 0]
- Sequence Number: 0 (relative sequence number)
- Sequence Number (raw): 2154038061
- [Next Sequence Number: 1 (relative sequence number)]
- Acknowledgment Number: 0
- Acknowledgment number (raw): 0
- 1010 .... = Header Length: 40 bytes (10)
- > Flags: 0x002 (SYN)
- Window: 64240
- [Calculated window size: 64240]
- Checksum: 0x087a [unverified]
- [Checksum Status: Unverified]

Internet protocol, viser hvem som sender og mottaker. Destination(dst) er min IPadresse, mens source er den som sendte det. VMwear er at requesten er fra en virtual maskin (192.168.41.200). Intel står på min, som betyr at datapakken blir sent med Intel som nettverkskort:

```

▼ Internet Protocol Version 4, Src: 192.168.41.200, Dst: 192.168.41.74
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 60
  Identification: 0xc3bd (50109)
> 010. .... = Flags: 0x2, Don't fragment
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 64
  Protocol: TCP (6)
  Header Checksum: 0xa29b [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 192.168.41.200
  Destination Address: 192.168.41.74
  [Stream index: 1]
> Transmission Control Protocol. Src Port: 53556. Dst Port: 80. Seq: 0, Len: 0

```

Ethernet viser igjen mac – adresser og ipv4(version 4):

```

▼ Ethernet II, Src: VMware_b6:7c:d4 (00:0c:29:b6:7c:d4), Dst: Intel_ac:52:ba (f4:8c:50:ac:52:ba)
  > Destination: Intel_ac:52:ba (f4:8c:50:ac:52:ba)
  > Source: VMware_b6:7c:d4 (00:0c:29:b6:7c:d4)
    Type: IPv4 (0x0800)
    [Stream index: 1]
> Internet Protocol Version 4, Src: 192.168.41.200, Dst: 192.168.41.74
> Transmission Control Protocol, Src Port: 53556, Dst Port: 80, Seq: 0, Len: 0

```

Frame, alt med informasjon i datapakken som lende og tid på pakken:

```

▼ Frame 5: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface \Device\NPF
  Section number: 1
> Interface id: 0 (\Device\NPF_{FD7DBED2-54A3-4423-AF8E-D03A4D50F378})
  Encapsulation type: Ethernet (1)
  Arrival Time: Mar 31, 2025 18:44:59.947605000 Sentral-Europa (sommertid)
  UTC Arrival Time: Mar 31, 2025 16:44:59.947605000 UTC
  Epoch Arrival Time: 1743439499.947605000
  [Time shift for this packet: 0.000000000 seconds]
  [Time delta from previous captured frame: 1.024084000 seconds]
  [Time delta from previous displayed frame: 1.024084000 seconds]
  [Time since reference or first frame: 3.823532000 seconds]
  Frame Number: 5
  Frame Length: 74 bytes (592 bits)
  Capture Length: 74 bytes (592 bits)
  [Frame is marked: False]
  [Frame is ignored: False]
  [Protocols in frame: eth:ethertvde:ip:tcp]

```

## 2 ulike klienter:

Jeg har testet med 2 ulike klienter. Den ene klienten brukte en motor som er chromium basert(microsoft edge) og den andre var Mozilla Firefox med Gecko som motor. De testet ved å skrive inn url adressen på samme port nummer.

Klient 1(Microsoft Edge) som koblet seg til brukte ip-adressen 192.168.41.60. Hypertext transfer protocol viser statusen. Origin, for eksempel viser URL adressen. Man kan se at det er chromiumum på User-agent. Der står det AppleWebKit. Som forteller om søkermotoren som er brukt. Bildet under viser avsender og mottaker av datapakken.

```

▼ Hypertext Transfer Protocol
  > POST /post_result HTTP/1.1\r\n
    Host: 192.168.41.74:8080\r\n
    Connection: keep-alive\r\n
  > Content-Length: 31\r\n
    Cache-Control: max-age=0\r\n
    Origin: http://192.168.41.74:8080\r\n
    Content-Type: application/x-www-form-urlencoded\r\n
    Upgrade-Insecure-Requests: 1\r\n
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.36\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7

[Coloring Rule String: http || tcp.port == 80 || http2]
▼ Ethernet II, Src: Intel_85:ca:fa (c4:bd:e5:85:ca:fa), Dst: Intel_ac:52:ba (f4:8c:50:ac:52:ba)
  > Destination: Intel_ac:52:ba (f4:8c:50:ac:52:ba)
  > Source: Intel_85:ca:fa (c4:bd:e5:85:ca:fa)
  Type: IPv4 (0x0800)
  [Stream index: 26]
▼ Internet Protocol Version 4, Src: 192.168.41.60, Dst: 192.168.41.74
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 755

```

Klient 2(Firefox) er IP adressen som kobler seg på 192.168.41. Du kan se de ulike protocolene og hvordan datapakkene oppfører seg. Du kan nå se at det er firefox som er bruk på det andre bilde. Det står på User-agent, Gecko og firefox.

```

[Coloring Rule String: http || tcp.port == 80 || http2]
▼ Ethernet II, Src: Intel_c7:81:70 (3c:58:c2:c7:81:70), Dst: Intel_ac:52:ba (f4:8c:50:ac:52:ba)
  > Destination: Intel_ac:52:ba (f4:8c:50:ac:52:ba)
  > Source: Intel_c7:81:70 (3c:58:c2:c7:81:70)
  Type: IPv4 (0x0800)
  [Stream index: 17]
▼ Internet Protocol Version 4, Src: 192.168.41.26, Dst: 192.168.41.74
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 578
  Identification: 0x5b4c (23372)

```

```

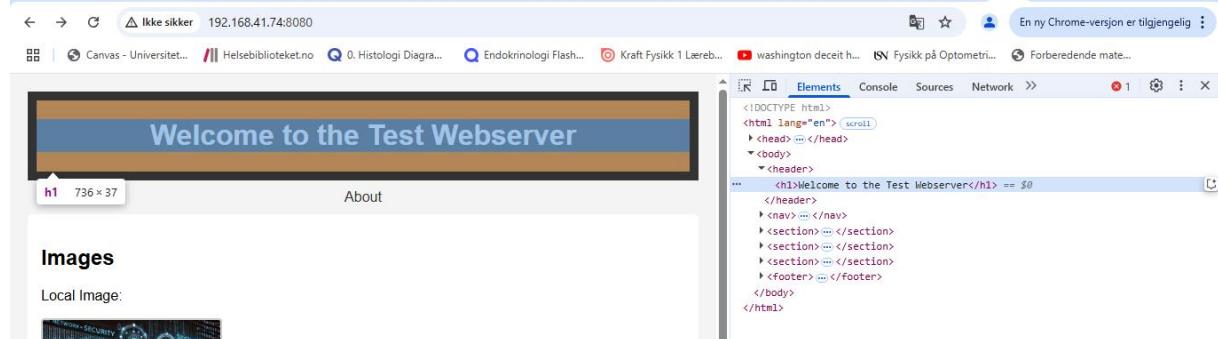
▼ Hypertext Transfer Protocol
  > POST /post_result HTTP/1.1\r\n
    Host: 192.168.41.74:8080\r\n
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:136.0) Gecko/20100101 Firefox/136.0\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    Accept-Language: en-US,en;q=0.5\r\n
    Accept-Encoding: gzip, deflate\r\n
    Content-Type: application/x-www-form-urlencoded\r\n
  > Content-Length: 37\r\n
  Origin: http://192.168.41.74:8080\r\n
  Connection: keep-alive\r\n
  Referer: http://192.168.41.74:8080/

```

## Kildekode:

Jeg har valgt å inspirere kildekoden til den publiserte filen.

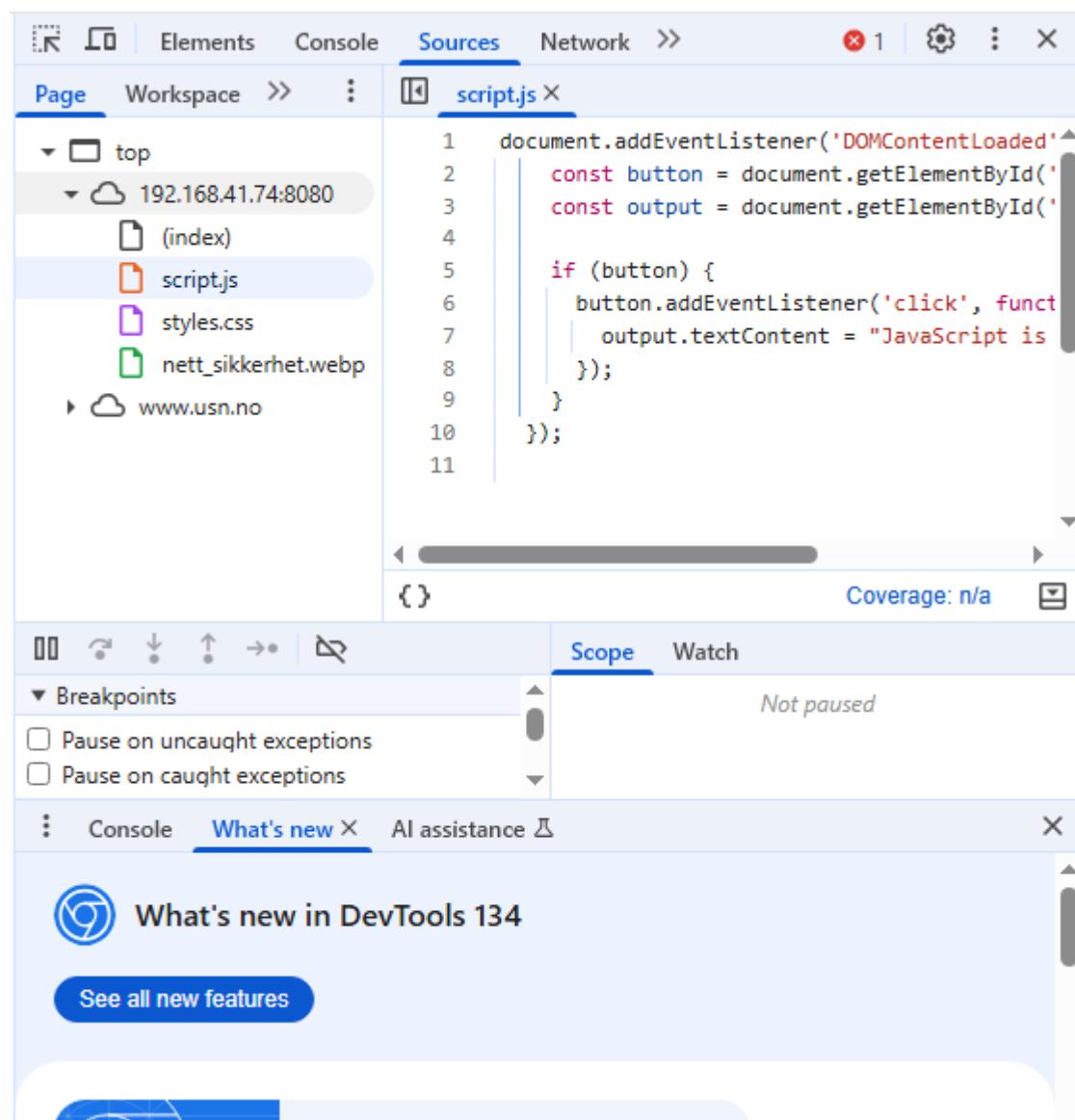
Elements kan vise og endre kilde koden til de ulike elementene som er brukt, her er det about.html filen:



The screenshot shows the Chrome DevTools interface with the 'Elements' tab selected. On the left, the page content is displayed with a header 'Welcome to the Test Webserver'. On the right, the DOM tree is shown, starting with the root <html> tag. The 'About' section is highlighted in the tree, and its source code is visible in the main pane:

```
<!DOCTYPE html>
<html lang="en"> <scroll>
  <head> ... </head>
  <body>
    <header>
      <h1>Welcome to the Test Webserver</h1> == $0
    </header>
    <nav> ... </nav>
    <section> ... </section>
    <section> ... </section>
    <section> ... </section>
    <footer> ... </footer>
  </body>
</html>
```

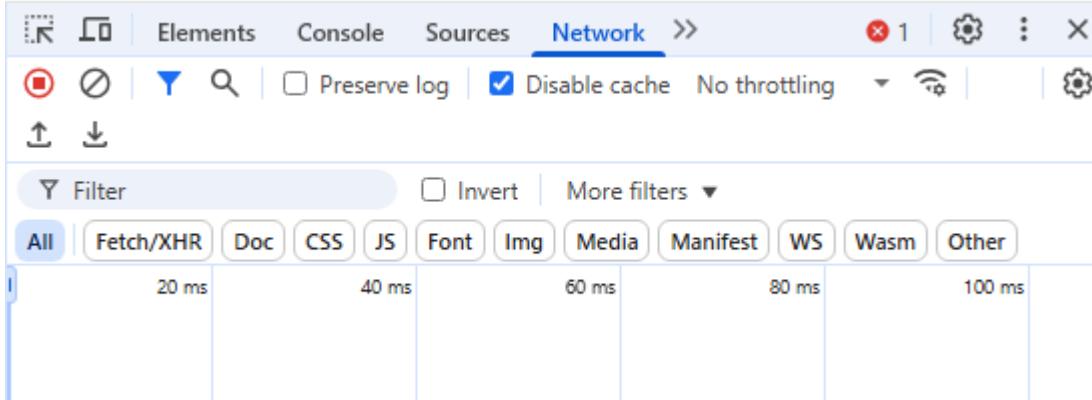
Feilmeldinger vises i konsollen. Der kan man se om det er en ugyldig fil:



The screenshot shows the Chrome DevTools interface with the 'Sources' tab selected. On the left, the file structure is shown with 'script.js' selected. The code in 'script.js' contains several syntax errors, indicated by red squiggly lines under the words 'DOMContentLoaded', 'button', 'output', 'click', and 'function'. The code is as follows:

```
1  document.addEventListener('DOMContentLoaded'
2  const button = document.getElementById('button')
3  const output = document.getElementById('output')
4
5  if (button) {
6    button.addEventListener('click', function() {
7      output.textContent = "JavaScript is cool!";
8    });
9  }
10 });
11
```

Nettverk kan se trafikken:



## Sikkerhetsrisiko og forbedringer:

Det er flere tiltak man kan ta for å få det mye sikrere. Den er en http fil i port 80 som standard så man kan bruke https port 443 for å ha det mer sikkert. På bildet står det ikke sikkert, man blir advart:

⚠ Ikke sikker 192.168.41.74:8080

Dette fungerer kun på IRI-laben der brannmurene er nede. Så det sier seg selv, det må ha en form for beskyttelse. Eksterne trusler kan prøve å ta sensitiv data, altså cross-site scripting(XXS). Andre eksterne trussler kan sitte og lytte til wireshark der de kan se hvem og hvor datapakkene går til (ulovlig sporing). De kan gjøre en form for «phishing» [3] «9. Data sikkerhet – Datakriminalitet.pdf» som er gjort på POST når den er testet med andre klienter. Det er ikke noe form for sikkerhet som autorisering så man kan se det de legger ut. Som på bildet, uavhengig av nettleser:

http.request.method == "POST"					
No.	Time	Source	Destination	Protocol	Lengt! Info
617	338.276610	192.168.41.26	192.168.41.74	HTTP	742 POST /post_result HTTP/1.1 (application/x-www-form-urlencoded)
622	338.885338	192.168.41.26	192.168.41.74	HTTP	742 POST /post_result HTTP/1.1 (application/x-www-form-urlencoded)
727	537.136840	192.168.41.26	192.168.41.74	HTTP	742 POST /post_result HTTP/1.1 (application/x-www-form-urlencoded)
+ 1303	951.666006	192.168.41.60	192.168.41.74	HTTP	769 POST /post_result HTTP/1.1 (application/x-www-form-urlencoded)

```
Cache-Control: max-age=0\r\n
Origin: http://192.168.41.74:8080\r\n
Content-Type: application/x-www-form-urlencoded\r\n
Upgrade-Insecure-Requests: 1\r\n
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like G
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,ima
Referer: http://192.168.41.74:8080/\r\n
Accept-Encoding: gzip, deflate\r\n
Accept-Language: nb-NO,nb;q=0.9,en-NO;q=0.8,en;q=0.7,ta-NO;q=0.6,ta;q=0.5,no;q=0.4,nn;q
\r\n
[Response in frame: 1306]
[Full request URI: http://192.168.41.74:8080/post_result]
File Data: 31 bytes
HTML Form URL Encoded: application/x-www-form-urlencoded
> Form item: "username" = "Red"
> Form item: "email" = "red@usn.no"
```

```
Accept-Language: en-US,en;q=0.5\r\n
Accept-Encoding: gzip, deflate\r\n
Content-Type: application/x-www-form-urlencoded\r\n
> Content-Length: 37\r\n
Origin: http://192.168.41.74:8080\r\n
Connection: keep-alive\r\n
Referer: http://192.168.41.74:8080/\r\n
Upgrade-Insecure-Requests: 1\r\n
Priority: u=0, i\r\n
\r\n
\[Response in frame: 2531\]
\[Full request URI: http://192.168.41.74:8080/post\_result\]
File Data: 37 bytes
▼ HTML Form URL Encoded: application/x-www-form-urlencoded
  > Form item: "username" = "oreo42"
  > Form item: "email" = "261688@usn.no"
```

## Konklusjon

Jeg har i denne oppgaven skrevet webserver i C++ kode. Jeg har laget egen html fil og testet meg egenfil og med publisert filsett også. Jeg har logget det og sett på trafikken, egenskapene og innholdet på datapakkene i wireshark. Jeg har testet med 2 ulike pc er og klienter og gjort noen vurderinger rundt sårbarheten. Jeg har analysert post sin oppførsel. Jeg har også tatt en inspeksjon av webserveren.

## Referanser

- [1] What Make Art, *Learn Basic HTML – Hello World – Part 1.*  
<https://www.youtube.com/watch?v=6vJOOhjt3aw>, (sist besøkt 30.03.2025).
- [2] Ingår Foldvik Pedersen., *tutorial* 24.03.2025, USN kongsberg,(sist besøkt 01.04.2025).
- [3] Olaf Hallan Graven., *9. Data sikkerhet – Datakriminalitet.pdf*, USN kongsberg,(sist besøkt 01.04.2025).
- [4] Olaf Hallan Graven., *8. Data sikkerhet – Intro & kryptering.pdf*, USN kongsberg,(sist besøkt 01.04.2025).
- [5] Olaf Hallan Graven., *10. Data sikkerhet – Personvern.pdf*, USN kongsberg,(sist besøkt 31.04.2025).
- [6] Dianne Pena, *An Alphabetized List of MIME Types*, [https://www.sitepoint.com/mime-types-complete-list/?utm\\_source=chatgpt.com](https://www.sitepoint.com/mime-types-complete-list/?utm_source=chatgpt.com) (sist besøkt 29.03.2025).

- Vedlegg

```
// DETTE ER MIN SERVER

#include <winsock2.h>
#include <ws2tcpip.h>
#include <fstream>
#include <csignal>
#include <string>
#include <iostream>
#pragma comment(lib,"Ws2_32.lib")

#include <map>
#include <sstream>
using namespace std;

/*
typedef struct sockaddr_in {
    short sin_family;
    u_short sin_port;
    struct in_addr sin_addr;
    char sin_zero[8];
} SOCKADDR_IN, * PSOCKADDR_IN;

typedef struct in_addr {
    union {
        struct {
            u_char s_b1;
            u_char s_b2;
            u_char s_b3;
            u_char s_b4;
        } S_un_b;
        struct {
            u_short s_w1;
            u_short s_w2;
        } S_un_w;
    } S_un;
} IN_ADDR, * PIN_ADDR;
*/
```

```

        u_short s_w2;
    } S_un_w;
    u_long S_addr;
} S_un;
} IN_ADDR,*PIN_ADDR,FAR*LPIN_ADDR;

//lage struct for sockaddr i bind og accept
typedef struct sockaddr {
    u_short sa_family;
    CHAR sa_data[14];
}SOCKADDR, * PSOCKADDR, * LPSOCKADDR;
*/

void logg(const string& filename, const string& message) {
    std::ofstream logName(filename, ios::app);
    if (logName.is_open()) {
        logName << message << std::endl;
    }
    else {
        std::cout << "failed logg " << filename << std::endl;
    }
}

void signal_handler(int signal_number) {
    string fileName = "clientFile.txt";

    if (signal_number == SIGINT) {
        logg(fileName, "SIGINT (ctl + c)");
        std::cout << "sigint" << std::endl;
    }
}

```

```
        logg(fileName, "SIGINT (ctl + c)");
        std::cout << "sigterm" << std::endl;
    }
    exit(signal_number);
    std::cout << "exit session" << std::endl;
}

✓ void cleanSocket() {
    string fileName = "clientFile.txt";
    logg(fileName, "Socket is cleaned up");
    WSACleanup();
    std::cout << "Socket is clean" << std::endl;
}

✓ void errorSocket() {
    string fileName = "clientFile.txt";
    logg(fileName, "error is detected in socket");
    int errorSocket = WSAGetLastError();
    std::cout << "Socket error: " << errorSocket << std::endl;
}

✓ string readFile(const string& path) {
    string fileName = "clientFile.txt";
    std::ifstream fil(path, std::ios::binary);
    if (!fil) {
        return "";
    }
    std::cout << "file found " << std::endl;
    return string((std::istreambuf_iterator<char>(fil)), std::istreambuf_iterator<char>());
}

✓ string GETtype(const string& path) {
    string fileName = "clientFile.txt";
    std::map<string, string> content = {
        {".html", "text/html"}, {"css", "text/css"}, {"js", "application/javascript"}  
}
```

```

string GETtype(const string& path) {
    string fileName = "clientFile.txt";
    std::map<string, string> content = {
        {"html", "text/html"}, {"css", "text/css"}, {"js", "application/javascript"},
        {"jpg", "image/jpeg"}, {"png", "image/png"}, {"webp", "image/webp"}
    };

    size_t dot = path.rfind('.');
    if (dot != string::npos) {
        string ext = path.substr(dot);
        if (content.count(ext)) {
            return content[ext];
        }
    }
}

log(fileName, "type of file found");
std::cout << "type of file found" << std::endl;
return "application/octet-stream";
}

void filePOST(SOCKET clientSocket, const string& data) {
    string fileName = "clientFile.txt";
    std::istringstream request(data);
    string HttpType, pathway, protocol;
    request >> HttpType >> pathway >> protocol;

    if (pathway == "/") pathway = "/helloworld1.html";
    //if (pathway == "/") pathway = "/index.html";
    //string filePath = "oblig3" + pathway;
    string filePath = "EgenFil" + pathway;
    string contentType = readFile(filePath);
    string callBack;
    log(fileName, "request from client found");
    std::cout << "request from client found" << std::endl;

    std::cout << "request from client found" << std::endl;

    if (HttpType == "POST") {
        callBack = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n<h1>POST Request is sent</h1>";
        log(fileName, "request from client sent");
        std::cout << "request from client sent" << std::endl;
    }
    else if (contentType.empty()) {
        callBack = "HTTP/1.1 404 Not Found\r\nContent-Type: text/html\r\n\r\n<h1>404 - ERROR</h1>";
        log(fileName, "request from client not found");
        std::cout << "request from client not found" << std::endl;
    }
    else {
        callBack = "HTTP/1.1 200 OK\r\nContent-Type: " + GETtype(filePath) + "\r\n\r\n" + contentType;
        log(fileName, "pathway posted");
        std::cout << "pathway posted" << std::endl;
    }
    send(clientSocket, callBack.c_str(), callBack.size(), 0);
    log(fileName, "sent to client");
    std::cout << "sent to client" << std::endl;
    closesocket(clientSocket);
    cleanSocket();
}

int main() {
    int port;
    std::cout << "Enter port: " << std::endl;
    std::cin >> port;

    signal(SIGABRT, signal_handler);
    signal(SIGINT, signal_handler);

    string fileName = "clientFile.txt";
    log(fileName, "start program");
}

```

```
string fileName = "clientFile.txt";
logg(fileName, "start program");

WSADATA wsaData;
if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
    logg(fileName, "WSAStartup has not started up");
    std::cout << "WSAStartup failed" << std::endl;
    errorSocket();
    cleanSocket();
    return 1;
}
std::cout << "WSAStartup was successful " << std::endl;

SOCKET serverSocket = socket(AF_INET, SOCK_STREAM, 0);
if (serverSocket == INVALID_SOCKET) {
    logg(fileName, "Server socket creation failed");
    std::cout << "Server socket creation failed" << std::endl;
    cleanSocket();
    return 1;
}
else {
    std::cout << "Server socket creation" << std::endl;
}

sockaddr_in serverAddr = {};
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(port);
serverAddr.sin_addr.s_addr = INADDR_ANY;

if (bind(serverSocket, (SOCKADDR*)&serverAddr, sizeof(serverAddr)) == SOCKET_ERROR) {
    logg(fileName, "Server bind failed");
    std::cout << "Server bind failed" << std::endl;
    closesocket(serverSocket);
    cleanSocket();
    return 1;
}
```

```

        log(fileName, "Server bind failed");
        std::cout << "Server bind failed" << std::endl;
        closesocket(serverSocket);
        cleanSocket();
        return 1;
    }

    else {
        std::cout << "server bind creation" << std::endl;
    }

    if (listen(serverSocket, 1) == SOCKET_ERROR) {
        log(fileName, "Server listen failed");
        std::cout << "Server listen failed" << std::endl;
        closesocket(serverSocket);
        cleanSocket();
        return 1;
    }

    else {
        std::cout << "Server listen creation" << std::endl;
    }

    std::cout << "OK, server is running" << std::endl;

    while (true) {
        sockaddr_in client_addr;
        int clientSize = sizeof(client_addr);
        SOCKET clientSocket = accept(serverSocket, (SOCKADDR*)&client_addr, &clientSize);
        if (clientSocket == INVALID_SOCKET) continue;

        char buffer[1024];
        int revSocket = recv(clientSocket, buffer, sizeof(buffer), 1024, 0);
        if (revSocket > 0) {
            buffer[revSocket] = '\0';
            filePOST(clientSocket, buffer);

            filePOST(clientSocket, buffer);
        }
        closesocket(clientSocket);
    }

    closesocket(serverSocket);
    cleanSocket();
    return 0;
}

```