HTTP Server

- **HTTP (Hypertext Transfer Protocol** is a protocol used for transferring data on the web.
- It's the fundamental protocol of the internet and is used for accessing web pages, downloading files, and performing other interactions between servers and browsers.
- To create an HTTP server using the C programming language, you can use the C socket library to create and manage network connections. Here are the basic steps to create an HTTP server using C:
    1. **Initialize Socket**: Use the `socket()` function to create a socket for the server.
    2. **Bind Socket**: Use the `bind()` function to bind the socket to an IP address and a port.
    3. **Listen for Connections**: Use the `listen()` function to allow the socket to accept connections from clients.
    4. **Accept Connection**: Use the `accept()` function to accept a connection from a client and create a child socket for data transmission.
    5. **Handle HTTP Requests**: Read data from the client through the child socket and respond with the corresponding HTTP data.

Advance function

1. **Handle Different HTTP Methods**: Extend the server to handle various HTTP methods such as GET, POST, PUT, DELETE, etc. You can parse the incoming requests and dispatch them to appropriate handler functions based on the method.
2. **Serve Static Files**: Implement logic to serve static files (HTML, CSS, JavaScript, images, etc.) requested by clients. This involves reading the requested file from disk and sending it back to the client in the HTTP response.
3. **Error Handling**: Enhance error handling to provide meaningful error messages to clients in case of errors. Handle common HTTP status codes such as 404 Not Found, 500 Internal Server Error, etc.
4. **Concurrency**: Implement concurrency to handle multiple client connections simultaneously. You can achieve this by using threads or asynchronous I/O techniques such as epoll or kqueue.
5. **Security**: Implement security features such as HTTPS (HTTP over SSL/TLS) to encrypt data transmitted between the server and clients. Additionally, implement measures to prevent common security vulnerabilities such as cross-site scripting (XSS), SQL injection, and CSRF (Cross-Site Request Forgery).

6. **Logging**: Add logging functionality to log server activity, client requests, errors, etc. Logging can help in debugging, monitoring server performance, and auditing.
7. **Configuration**: Implement configuration options to allow customization of server settings such as port number, document root directory, etc. This makes the server more flexible and configurable.
8. **Testing**: Write unit tests and integration tests to ensure the server behaves as expected and handles various scenarios correctly. Automated testing helps in identifying and fixing issues early in the development process.
9. **Documentation**: Document the server's API, configuration options, and usage instructions for developers who want to use or contribute to the server. Clear documentation makes it easier for others to understand and use the server.
10. **Performance Optimization**: Profile the server to identify performance bottlenecks and optimize critical sections of the code. Techniques such as caching, connection pooling, and optimizing I/O operations can improve server performance

I. Multiple thread function

- Goal: we will test multiple thread on 3 machine. HTTP server on Window 11 and Client on 2 Linux.

1. On Window Machine

- Download file *http_server.c* from git:

```
https://github.com/ngoducthuan/Malware/tree/main/FundamentalPrograming/http_server
```

- Open in Visual Studio Code
- Compiler file *http_server.c* to *http_server.exe*

```
gcc .\http_server.c -o .\http_server.exe -lws2_32 -lpthread
```

2. On Linux Machine

- Dowload file *client.c* from git

```
https://github.com/ngoducthuan/Malware/blob/main/FundamentalPrograming/tcp
```

```
_client_server/client.c
```

- Open terminal, if you don't have MinGW:

```
sudo apt-get install mingw-w64
```

- Compiler file *client.c* to *client.exe*

```
i686-w64-mingw32-gcc client.c -o client.exe -lws2_32
```

- Now, you need install *wine* to run fil .exe on Linux

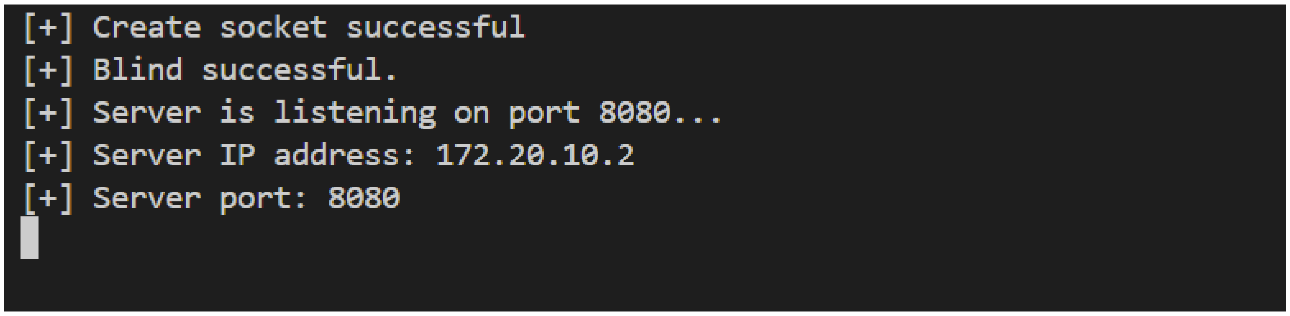
```
sudo apt-get install wine
```

3. Run to see thread function

- On window machine run.

```
 .\http_server.exe
```

- You will see:



```
[+] Create socket successful
[+] Blind successful.
[+] Server is listening on port 8080...
[+] Server IP address: 172.20.10.2
[+] Server port: 8080
```

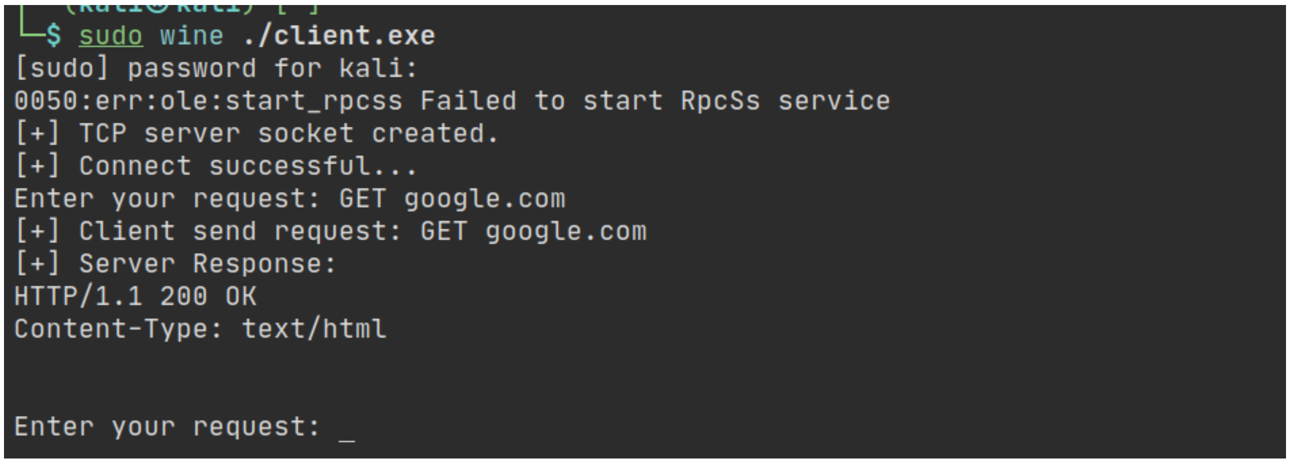
- On linux machine run:

```
sudo wine client.exe
```

- If it not work:

```
 WINEPREFIX=$HOME/.wine_new wine client.exe
```

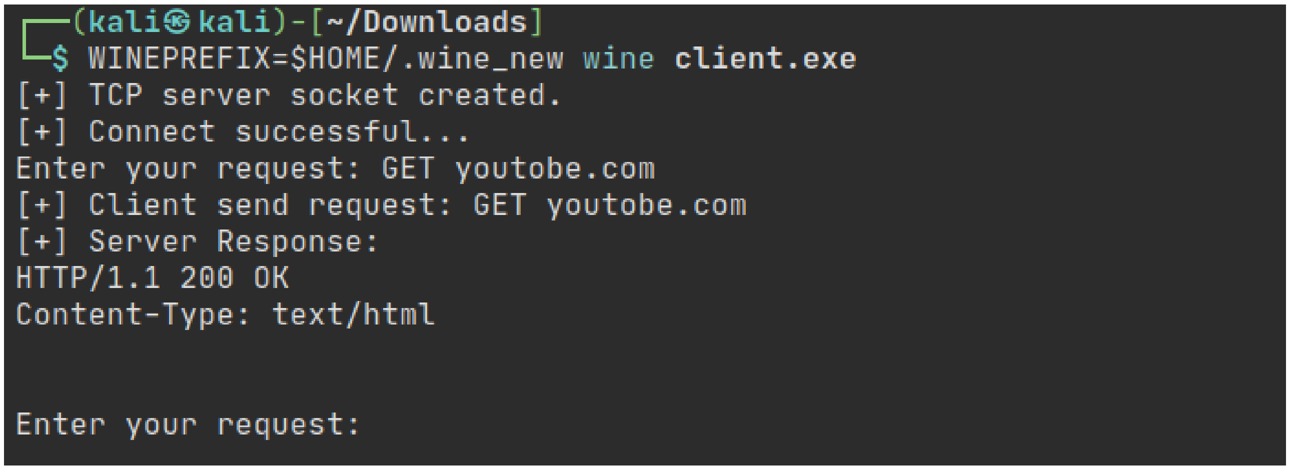
- You will send a requestion(GET google.com), then received:



```
└─$ sudo wine ./client.exe
[sudo] password for kali:
0050:err:ole:start_rpcss Failed to start RpcSs service
[+] TCP server socket created.
[+] Connect successful...
Enter your request: GET google.com
[+] Client send request: GET google.com
[+] Server Response:
HTTP/1.1 200 OK
Content-Type: text/html


Enter your request: _
```

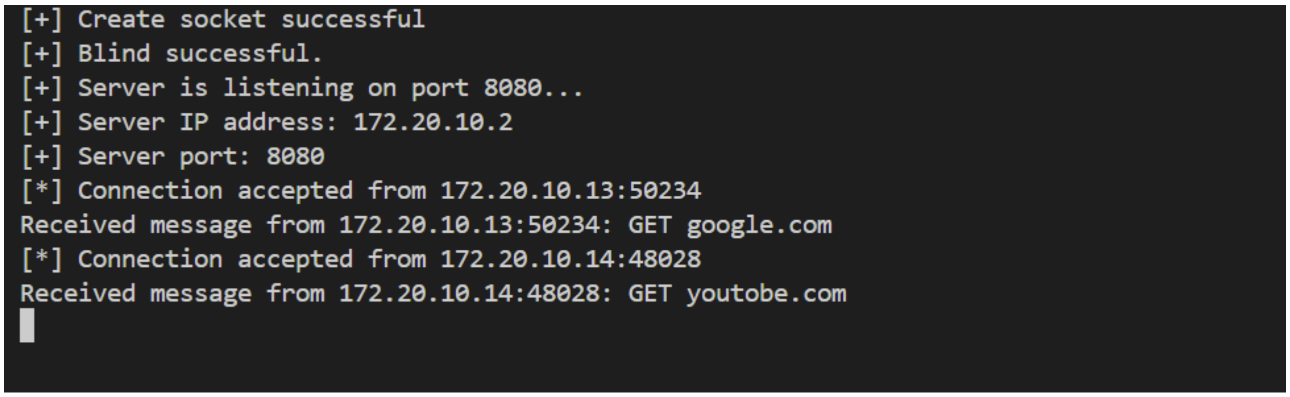
- Work same with another Linux machine:



```
┌──(kali㉿kali)-[~/Downloads]
└─$ WINEPREFIX=$HOME/.wine_new wine client.exe
[+] TCP server socket created.
[+] Connect successful...
Enter your request: GET youtobe.com
[+] Client send request: GET youtobe.com
[+] Server Response:
HTTP/1.1 200 OK
Content-Type: text/html


Enter your request:
```

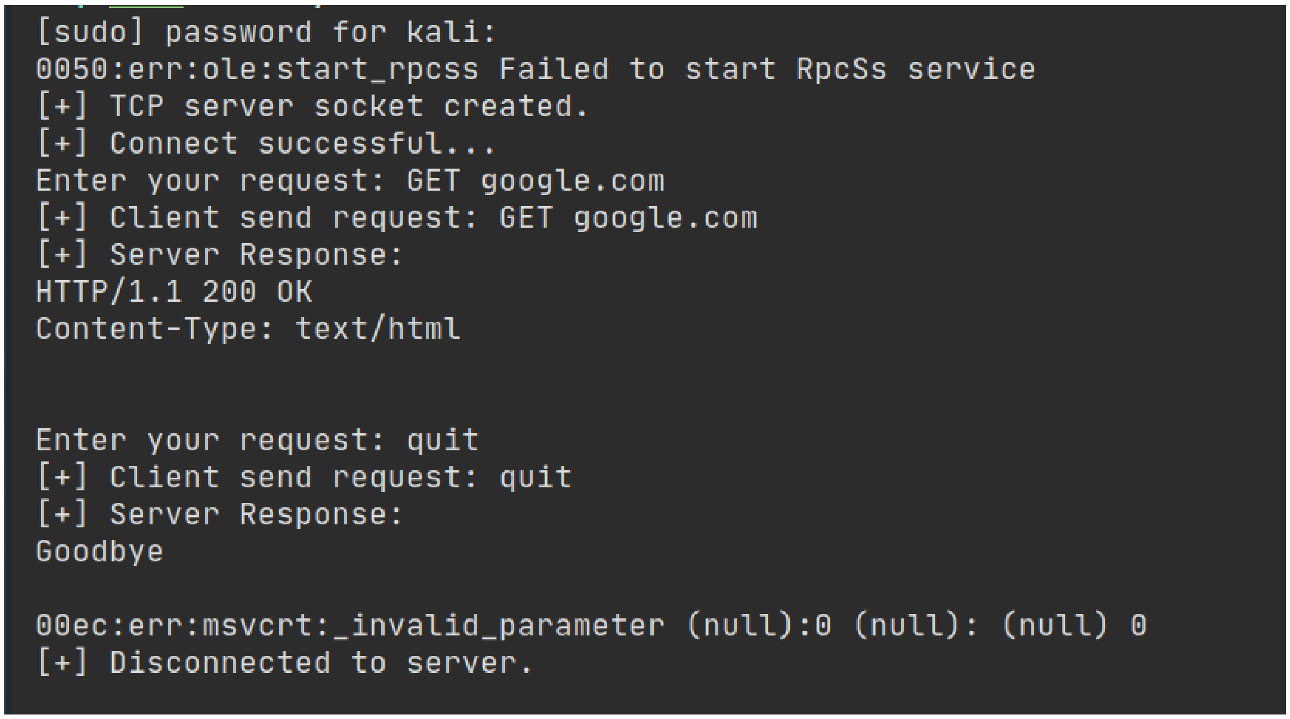
- Come back window machine, you can see:



```
[+] Create socket successful
[+] Blind successful.
[+] Server is listening on port 8080...
[+] Server IP address: 172.20.10.2
[+] Server port: 8080
[*] Connection accepted from 172.20.10.13:50234
Received message from 172.20.10.13:50234: GET google.com
[*] Connection accepted from 172.20.10.14:48028
Received message from 172.20.10.14:48028: GET youtobe.com
```

- Now, our test is successfully.

- Go to Linux machine type quit to disconnected.

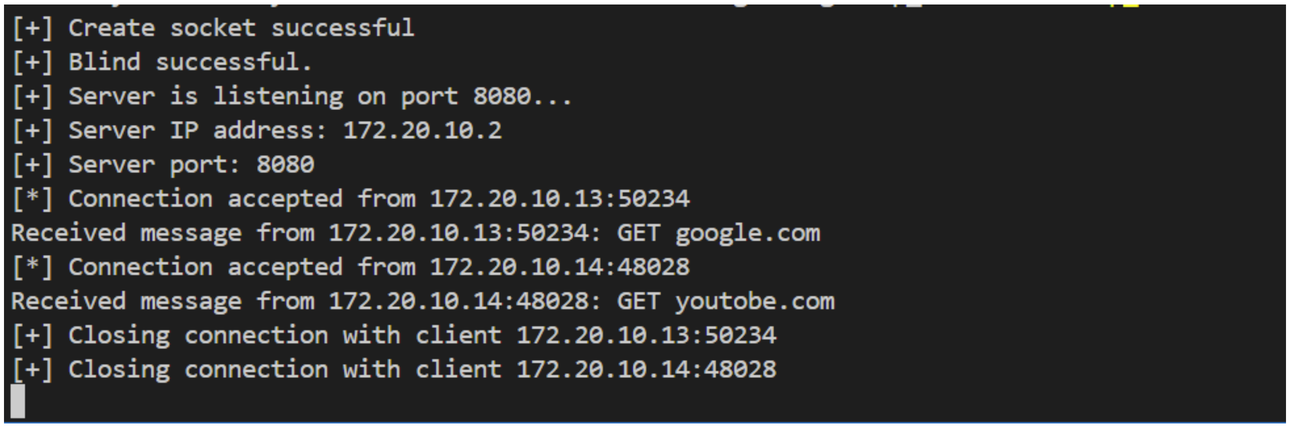


```
[sudo] password for kali:
0050:err:ole:start_rpcss Failed to start RpcSs service
[+] TCP server socket created.
[+] Connect successful...
Enter your request: GET google.com
[+] Client send request: GET google.com
[+] Server Response:
HTTP/1.1 200 OK
Content-Type: text/html


Enter your request: quit
[+] Client send request: quit
[+] Server Response:
Goodbye

00ec:err:msvcrt:_invalid_parameter (null):0 (null): (null) 0
[+] Disconnected to server.
```

- We also message disconnection on Window.



```
[+] Create socket successful
[+] Blind successful.
[+] Server is listening on port 8080...
[+] Server IP address: 172.20.10.2
[+] Server port: 8080
[*] Connection accepted from 172.20.10.13:50234
Received message from 172.20.10.13:50234: GET google.com
[*] Connection accepted from 172.20.10.14:48028
Received message from 172.20.10.14:48028: GET youtobe.com
[+] Closing connection with client 172.20.10.13:50234
[+] Closing connection with client 172.20.10.14:48028
```