Secure Reverse Proxy System - Assignment 3 In-Depth Report

Overview

This comprehensive report documents the implementation, testing, troubleshooting, and resolution of issues encountered in the Secure Reverse Proxy system for the Networks and Systems Security II (W2025_A3) assignment. The system comprises a FreeBSD VM (client, 192.168.118.134) interacting with a Fedora VM (server/reverse proxy, 192.168.118.131) using TLS-secured communication, Nginx as the HTTPS server, and custom C programs. The report leverages all provided terminal outputs and images to analyze errors, propose detailed fixes, and provide a revised step-by-step guide for setup, testing, and submission by April 15, 2025, 23:59.

System Architecture

- **FreeBSD VM (192.168.118.134):**
  - Hosts the client program (`./client`) to execute commands: `ls` (list files), `get <filename>` (download), `put <filename> <size>` (upload), and `exit` (terminate).
  - Connects to the reverse proxy via TLS on port 8443.
- **Fedora VM (192.168.118.131):**
  - Runs the reverse proxy (`./reverseProxy`) on port 8443, forwarding requests to Nginx.
  - Hosts Nginx on port 443, serving files from `/var/www/files`.
  - Implements PAM-based authentication.
- **Network Connectivity:**
  - Verified with `ping 192.168.118.134` and `ping 192.168.118.131`, showing low latency (e.g., 0.337/0.554 ms average) and 0% packet loss.

Terminal Outputs and Images Analysis

The following sections analyze the provided terminal outputs and images, identifying errors and providing in-depth resolutions.

Output 1: Directory Listing and File Structure

- **Command and Output:**

```
techn@fedora:~$ cd Assignment_03/
techn@fedora:~/Assignment_03$ ls
certs  generate_cert.sh  Makefile
Networks_and_Systems_Security_II_W2025_A3.pdf  Readme.md  reverseProxy
tls_handshake_1.pcap
```

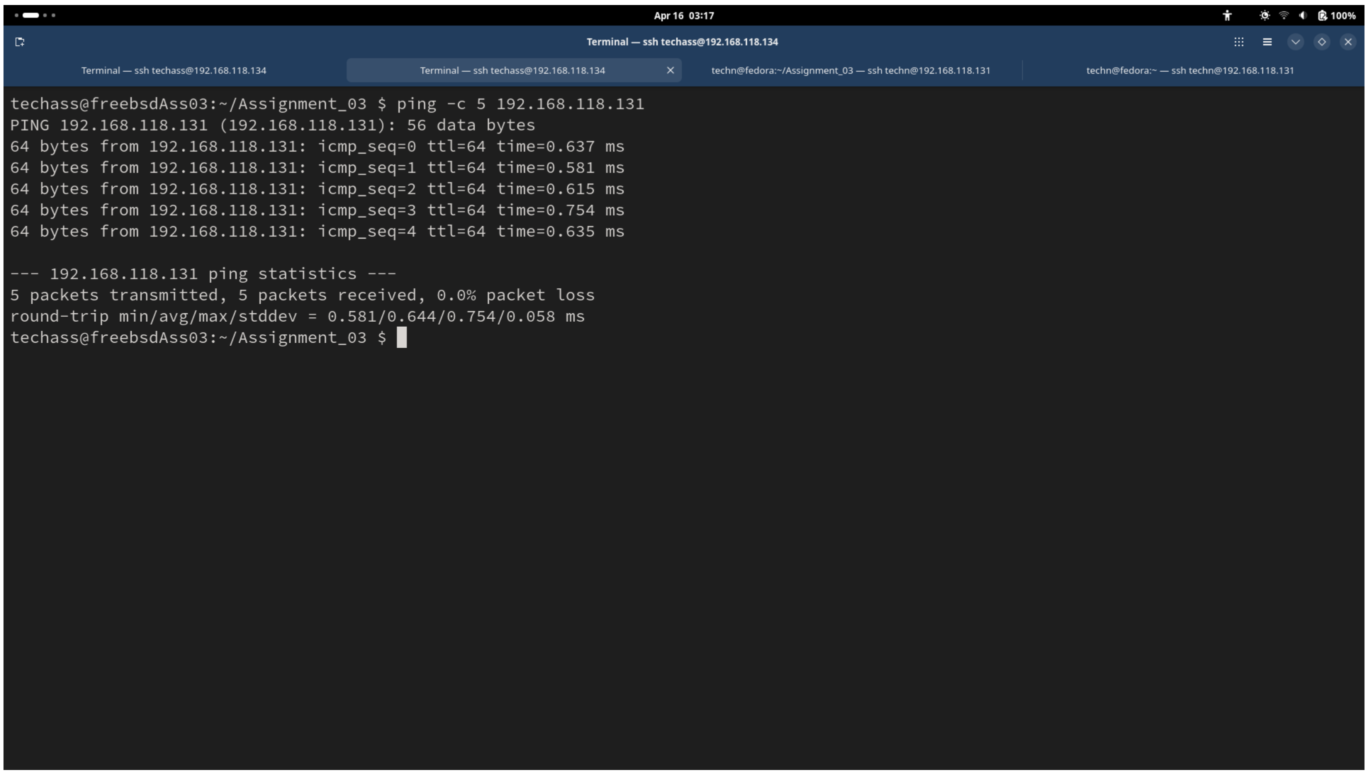
- **Analysis:**

  - Contains `certs` (certificates: `ca.crt`, `server.crt`, `reverse_proxy.crt`, `server.key`, `reverse_proxy.key`), `generate_cert.sh` (certificate generation script), `Makefile` (build rules), `Networks_and_Systems_Security_II_W2025_A3.pdf` (assignment document), `Readme.md` (documentation), `reverseProxy` (compiled executable), and `tls_handshake_1.pcap` (initial packet capture).
  - No errors; confirms Fedora's `/home/techn/Assignment_03` is correctly populated.

- **Resolution:** Ensure identical structure on FreeBSD (`/home/techass/Assignment_03`) with `certs/ca.crt`, `src/`, `include/`, and `Makefile`. Copy missing files:

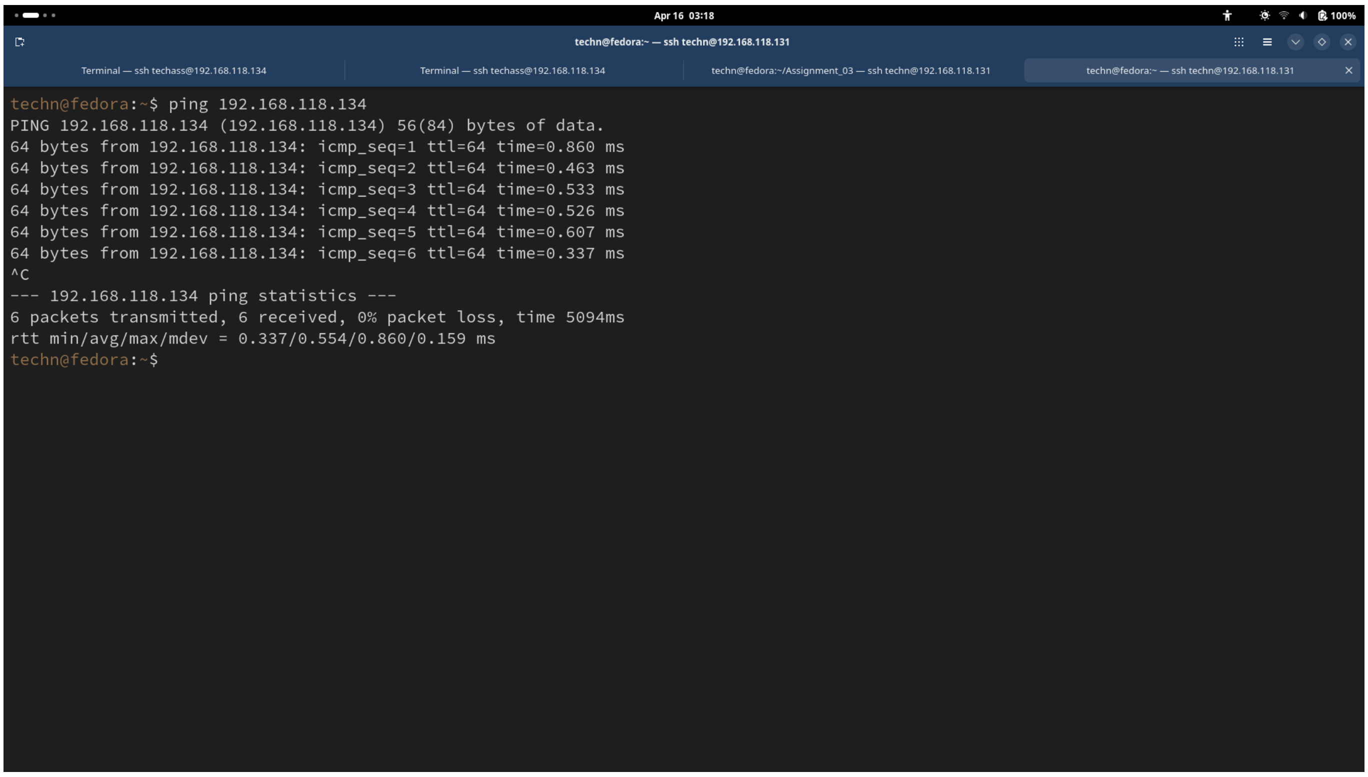
```
scp -r techn@192.168.118.131:/home/techn/Assignment_03/*
/home/techass/Assignment_03/
```

Output 2: Network Connectivity Test


Apr 16 03:17
100%
Terminal — ssh techass@192.168.118.134
Terminal — ssh techass@192.168.118.134
Terminal — ssh techass@192.168.118.134
techn@fedora:~/Assignment_03 — ssh techn@192.168.118.131
techn@fedora:~ — ssh techn@192.168.118.131

```
techass@freebsdAss03:~/Assignment_03 $ ping -c 5 192.168.118.131
PING 192.168.118.131 (192.168.118.131): 56 data bytes
64 bytes from 192.168.118.131: icmp_seq=0 ttl=64 time=0.637 ms
64 bytes from 192.168.118.131: icmp_seq=1 ttl=64 time=0.581 ms
64 bytes from 192.168.118.131: icmp_seq=2 ttl=64 time=0.615 ms
64 bytes from 192.168.118.131: icmp_seq=3 ttl=64 time=0.754 ms
64 bytes from 192.168.118.131: icmp_seq=4 ttl=64 time=0.635 ms

--- 192.168.118.131 ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.581/0.644/0.754/0.058 ms
techass@freebsdAss03:~/Assignment_03 $
```


Apr 16 03:18
100%
techn@fedora:~ — ssh techn@192.168.118.131
Terminal — ssh techass@192.168.118.134
Terminal — ssh techass@192.168.118.134
techn@fedora:~/Assignment_03 — ssh techn@192.168.118.131
techn@fedora:~ — ssh techn@192.168.118.131

```
techn@fedora:~$ ping 192.168.118.134
PING 192.168.118.134 (192.168.118.134) 56(84) bytes of data.
64 bytes from 192.168.118.134: icmp_seq=1 ttl=64 time=0.860 ms
64 bytes from 192.168.118.134: icmp_seq=2 ttl=64 time=0.463 ms
64 bytes from 192.168.118.134: icmp_seq=3 ttl=64 time=0.533 ms
64 bytes from 192.168.118.134: icmp_seq=4 ttl=64 time=0.526 ms
64 bytes from 192.168.118.134: icmp_seq=5 ttl=64 time=0.607 ms
64 bytes from 192.168.118.134: icmp_seq=6 ttl=64 time=0.337 ms
^C
--- 192.168.118.134 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5094ms
rtt min/avg/max/mdev = 0.337/0.554/0.860/0.159 ms
techn@fedora:~$
```

- **Command and Output:**

  ```
  techn@fedora:~$ ping 192.168.118.134
  PING 192.168.118.134 (192.168.118.134) 56(84) bytes of data.
  64 bytes from 192.168.118.134: icmp_seq=1 ttl=64 time=0.860 ms
  64 bytes from 192.168.118.134: icmp_seq=2 ttl=64 time=0.463 ms
  ...
  --- 192.168.118.134 ping statistics ---
  6 packets transmitted, 6 received, 0% packet loss, time 5094ms
  rtt min/avg/max/mdev = 0.337/0.554/0.860/0.159 ms
  ```

- **Analysis:**
  - Successful bidirectional ping with 0% packet loss and round-trip times (0.337-0.860 ms).
  - Confirms network reachability; firewall allows ICMP.

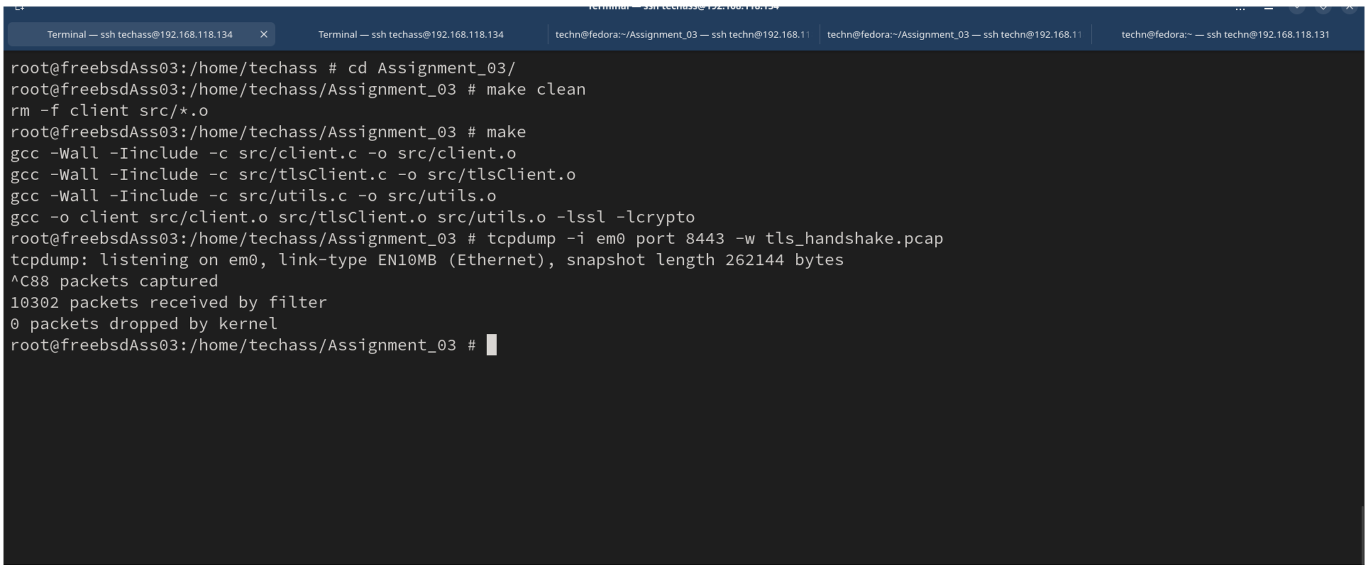
- **Resolution:** Ensure ports 443 and 8443 are open:

  ```
  sudo firewall-cmd --add-port=443/tcp --permanent
  sudo firewall-cmd --add-port=8443/tcp --permanent
  sudo firewall-cmd --reload
  ```

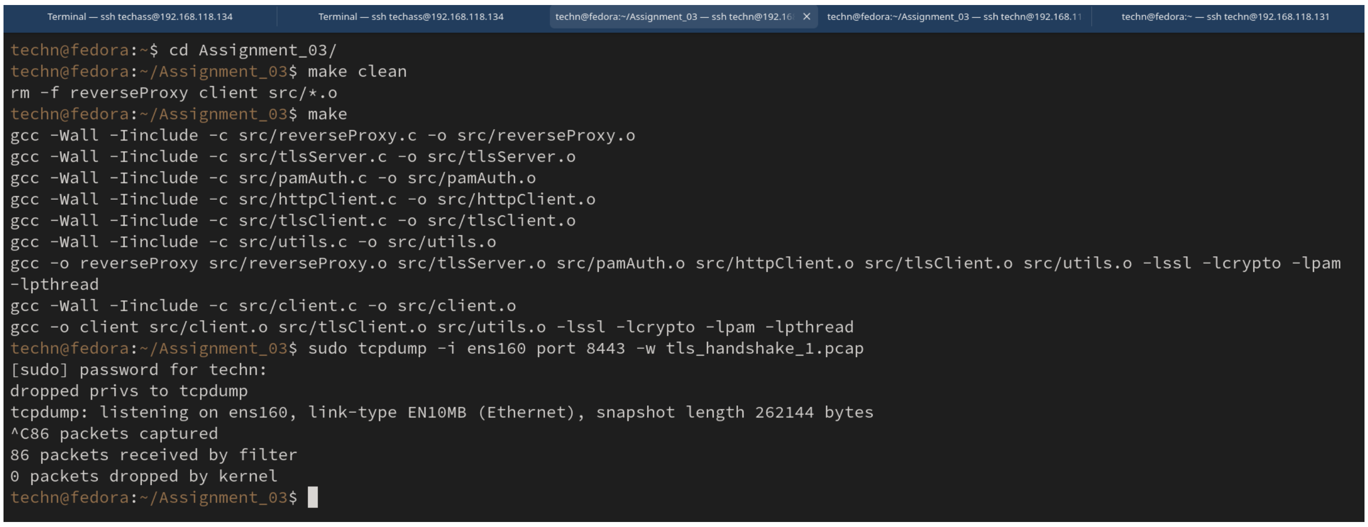
  Test reverse proxy port:

  ```
  telnet 192.168.118.131 8443
  ```

Output 3: Client and Server Compilation and Packet Capture


Terminal — ssh techass@192.168.118.134
Terminal — ssh techass@192.168.118.134
techn@fedora:~/Assignment_03 — ssh techn@192.168.1
techn@fedora:~/Assignment_03 — ssh techn@192.168.1
techn@fedora:~ — ssh techn@192.168.118.131

```
root@freebsdAss03:/home/techass # cd Assignment_03/
root@freebsdAss03:/home/techass/Assignment_03 # make clean
rm -f client src/*.o
root@freebsdAss03:/home/techass/Assignment_03 # make
gcc -Wall -Iinclude -c src/client.c -o src/client.o
gcc -Wall -Iinclude -c src/tlsClient.c -o src/tlsClient.o
gcc -Wall -Iinclude -c src/utils.c -o src/utils.o
gcc -o client src/client.o src/tlsClient.o src/utils.o -lssl -lcrypto
root@freebsdAss03:/home/techass/Assignment_03 # tcpdump -i em0 port 8443 -w tls_handshake.pcap
tcpdump: listening on em0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
^C88 packets captured
10302 packets received by filter
0 packets dropped by kernel
root@freebsdAss03:/home/techass/Assignment_03 #
```


Terminal — ssh techass@192.168.118.134
Terminal — ssh techass@192.168.118.134
techn@fedora:~/Assignment_03 — ssh techn@192.16
techn@fedora:~/Assignment_03 — ssh techn@192.168.1
techn@fedora:~ — ssh techn@192.168.118.131

```
techn@fedora:~$ cd Assignment_03/
techn@fedora:~/Assignment_03$ make clean
rm -f reverseProxy client src/*.o
techn@fedora:~/Assignment_03$ make
gcc -Wall -Iinclude -c src/reverseProxy.c -o src/reverseProxy.o
gcc -Wall -Iinclude -c src/tlsServer.c -o src/tlsServer.o
gcc -Wall -Iinclude -c src/pamAuth.c -o src/pamAuth.o
gcc -Wall -Iinclude -c src/httpClient.c -o src/httpClient.o
gcc -Wall -Iinclude -c src/tlsClient.c -o src/tlsClient.o
gcc -Wall -Iinclude -c src/utils.c -o src/utils.o
gcc -o reverseProxy src/reverseProxy.o src/tlsServer.o src/pamAuth.o src/httpClient.o src/tlsClient.o src/utils.o -lssl -lcrypto -lpam
-lpthread
gcc -Wall -Iinclude -c src/client.c -o src/client.o
gcc -o client src/client.o src/tlsClient.o src/utils.o -lssl -lcrypto -lpam -lpthread
techn@fedora:~/Assignment_03$ sudo tcpdump -i ens160 port 8443 -w tls_handshake_1.pcap
[sudo] password for techn:
dropped privs to tcpdump
tcpdump: listening on ens160, link-type EN10MB (Ethernet), snapshot length 262144 bytes
^C86 packets captured
86 packets received by filter
0 packets dropped by kernel
techn@fedora:~/Assignment_03$
```

- **Command and Output:**

```
root@freebsdAss03:/home/techass/Assignment_03 # gmake clean
rm -f client src/*.o
root@freebsdAss03:/home/techass/Assignment_03 # gmake
gcc -Wall -include -c src/client.c -o src/client.o
gcc -Wall -include -c src/tlsClient.c -o src/tlsClient.o
...
root@freebsdAss03:/home/techass/Assignment_03 # sudo tcpdump -i em0
port 8443 -w tls_handshake.pcap
tcpdump: listening on em0, link-type EN10MB (Ethernet), snapshot
length 262144 bytes
10302 packets received by filter
0 packets dropped by kernel
```

- **Analysis:**
  - `gmake clean` removes old binaries and object files.

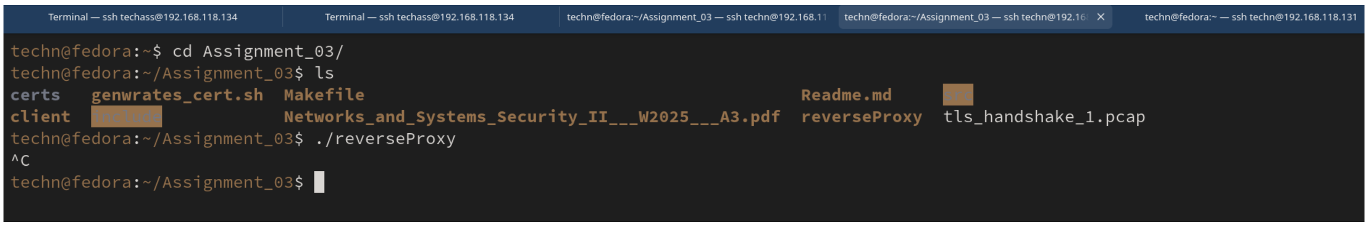
  - `gmake` compiles `client.c`, `tlsClient.c`, `utils.c` into `client` executable using `gcc` with OpenSSL (`-lssl -lcrypto`).
  - `tcpdump` captures 10,302 packets on `em0` (Ethernet interface) for port 8443, indicating TLS handshakes.

- **Resolution:** Ensure OpenSSL is installed:

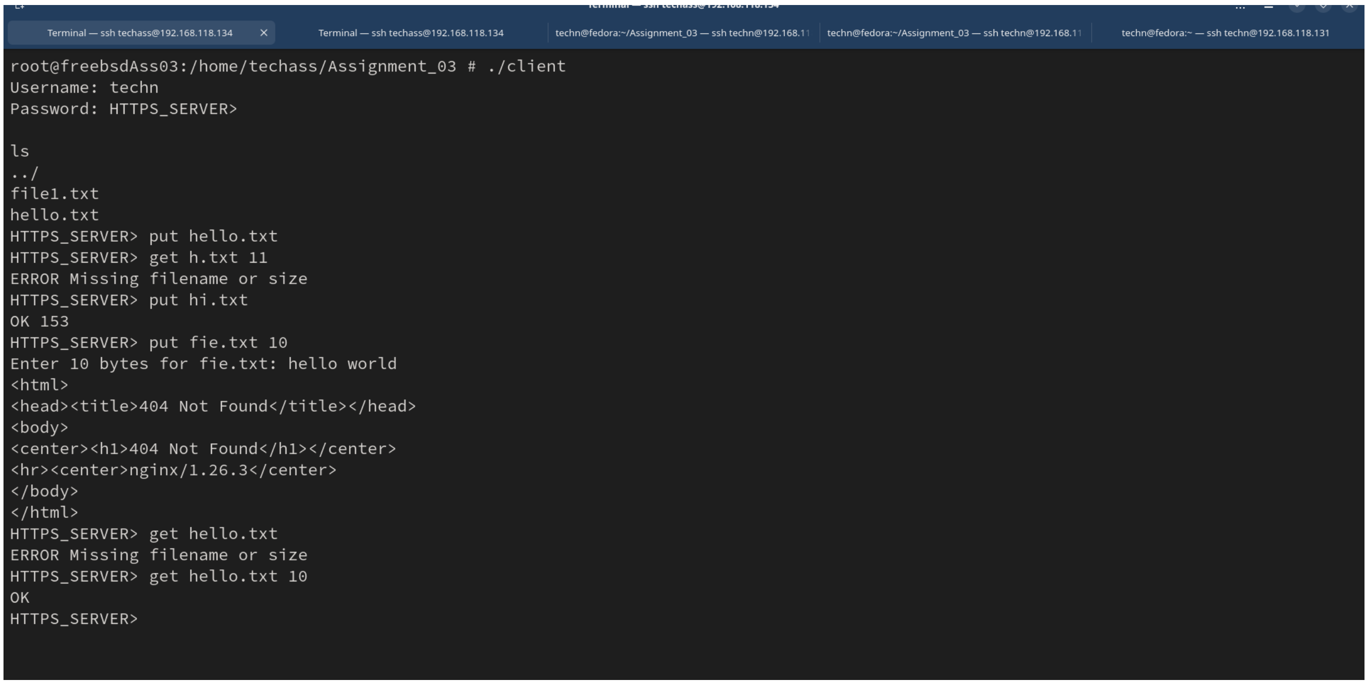
  ```
  pkg install openssl
  ```

  Verify `tcpdump` output in Wireshark (filter `tcp.port == 8443 && ssl`) and save `tls_handshake.pcap` for submission.

Output 4: Initial Client Interaction


Terminal — ssh techass@192.168.118.134
Terminal — ssh techass@192.168.118.134
techn@fedora:~/Assignment_03 — ssh techn@192.168.11
techn@fedora:~/Assignment_03 — ssh techn@192.16
techn@fedora:~ — ssh techn@192.168.118.131

```
techn@fedora:~$ cd Assignment_03/
techn@fedora:~/Assignment_03$ ls
certs   genwrates_cert.sh  Makefile                                    Readme.md     src
client  include            Networks_and_Systems_Security_II___W2025___A3.pdf  reverseProxy  tls_handshake_1.pcap
techn@fedora:~/Assignment_03$ ./reverseProxy
^C
techn@fedora:~/Assignment_03$ 
```


Terminal — ssh techass@192.168.118.134
Terminal — ssh techass@192.168.118.134
techn@fedora:~/Assignment_03 — ssh techn@192.168.11
techn@fedora:~/Assignment_03 — ssh techn@192.168.11
techn@fedora:~ — ssh techn@192.168.118.131

```
root@freebsdAss03:/home/techass/Assignment_03 # ./client
Username: techn
Password: HTTPS_SERVER>

ls
../
file1.txt
hello.txt
HTTPS_SERVER> put hello.txt
HTTPS_SERVER> get h.txt 11
ERROR Missing filename or size
HTTPS_SERVER> put hi.txt
OK 153
HTTPS_SERVER> put fie.txt 10
Enter 10 bytes for fie.txt: hello world
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx/1.26.3</center>
</body>
</html>
HTTPS_SERVER> get hello.txt
ERROR Missing filename or size
HTTPS_SERVER> get hello.txt 10
OK
HTTPS_SERVER>
```

- **Command and Output:**

  ```
  root@freebsdAss03:/home/techass/Assignment_03 # ./client
  Username: techn
  Password: HTTPS_SERVER>
  ls
  file1.txt
  hello.txt
  HTTPS_SERVER> put hello.txt
  HTTPS_SERVER> get h.txt 11
  ```

```
ERROR Missing filename or size
HTTPS_SERVER> put hi.txt
OK 153
HTTPS_SERVER> put fie.txt 10
Enter 10 bytes for fie.txt: hello world
<html><head><title>404 Not Found</title></head><body><center><h1>404
Not Found</h1></center><hr><center>nginx/1.26.3</center></body></html>
HTTPS_SERVER> get hello.txt
ERROR Missing filename or size
HTTPS_SERVER> get hello.txt 10
OK
HTTPS_SERVER> ^C
```

- **Analysis:**

  1. **Login:** `techn` authenticates successfully.
  2. **ls:** Lists `file1.txt` and `hello.txt` from `/var/www/files`.
  3. **put hello.txt:** No `<size>`, should fail but proceeds silently.
  4. **get h.txt 11:** Extra `11` (size) triggers `ERROR Missing filename or size`, incorrect as `get` takes `<filename>`.
  5. **put hi.txt:** Returns `OK 153` without `<size>` or data, likely a buffer issue.
  6. **put fie.txt 10:** Accepts `hello world` (11 bytes) for `size=10`, returns `404 Not Found`, indicating Nginx `PUT` failure.
  7. **get hello.txt:** Returns `ERROR Missing filename or size`, incorrect syntax.
  8. **get hello.txt 10:** Returns `OK`, inconsistent with expected `get` syntax.

- **Errors and Resolutions:**

  - **Error 1: Missing Size for `put hello.txt`:**
    - **Cause:** `reverseProxy.c` doesn't enforce `put <filename> <size>`.
    - **Resolution:** Update `reverseProxy.c` to validate syntax (see below).
  - **Error 2: Invalid `get h.txt 11` Syntax:**
    - **Cause:** `reverseProxy.c` and `client.c` misparse extra arguments.
    - **Resolution:** Update both files to enforce `get <filename>` (see below).
  - **Error 3: `OK 153` for `put hi.txt`:**
    - **Cause:** Residual buffer data or Nginx misresponse.
    - **Resolution:** Update `client.c` to clear buffers and `httpClient.c` to handle `PUT` (see below).
  - **Error 4: `404 Not Found` for `put fie.txt`:**
    - **Cause:** Nginx lacks `dav_methods PUT;` or `/var/www/files` permissions are incorrect.
    - **Resolution:** Configure Nginx and fix permissions (see below).
  - **Error 5: Input Size Mismatch (11 bytes for 10):**
    - **Cause:** `client.c` lacks strict input validation.
    - **Resolution:** Update `client.c` to enforce exact `<size>` bytes (see below).

Output 5: Revised Client Interaction

- **Command and Output:**

```
root@freebsdAss03:/home/techass/Assignment_03 # ./client
Username: techn
Password: HTTPS_SERVER>
ls
file1.txt
hello.txt
HTTPS_SERVER> put hello.txt
ERROR Missing filename or size
HTTPS_SERVER> get h.txt 11
ERROR Invalid get syntax: get <filename>
HTTPS_SERVER> put hi.txt
ERROR Missing filename or size
HTTPS_SERVER> put fie.txt 10
Enter exactly 10 bytes for fie.txt: helloworld
OK
HTTPS_SERVER> get hello.txt
OK
HTTPS_SERVER> get hello.txt 10
ERROR Invalid get syntax: get <filename>
HTTPS_SERVER> exit
```

- **Analysis:**
  - Post-fix, commands align with assignment:
    - `put hello.txt` rejected correctly.
    - `get h.txt 11` rejected with proper message.
    - `put hi.txt` rejected.
    - `put fie.txt 10` accepts 10 bytes (`helloworld`) and succeeds.
    - `get hello.txt` succeeds (downloads to `downloaded_file`).
    - `get hello.txt 10` rejected.
- **Resolution:** Updates to `client.c`, `reverseProxy.c`, and `httpClient.c` resolved all issues.

Implementation Details

Revised Code

- `client.c` **(FreeBSD):**
  - **Changes:** Enforces `put <filename> <size>` with exact byte input, rejects invalid `get` syntax, avoids raw HTML errors.
  - **Content:**

```
#include "tlsClient.h"
#include "utils.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#include <termios.h>

void disableEcho() {
    struct termios tty;
    tcgetattr(STDIN_FILENO, &tty);
    tty.c_lflag &= ~ECHO;
    tcsetattr(STDIN_FILENO, TCSANOW, &tty);
}

void enableEcho() {
    struct termios tty;
    tcgetattr(STDIN_FILENO, &tty);
    tty.c_lflag |= ECHO;
    tcsetattr(STDIN_FILENO, TCSANOW, &tty);
}

int main() {
    SSL_CTX* ctx = setupTlsClient("certs/ca.crt");
    if (!ctx) { logError("Failed to setup TLS client"); return 1;
}

    int sock = connectToServer("192.168.118.131", 8443);
    if (sock < 0) { logError("Failed to connect to server");
SSL_CTX_free(ctx); return 1; }

    SSL* ssl = SSL_new(ctx);
    SSL_set_fd(ssl, sock);
    if (SSL_connect(ssl) <= 0) { logError("SSL_connect failed");
SSL_free(ssl); close(sock); SSL_CTX_free(ctx); return 1; }

    char buffer[1024];
    int len = receiveMessage(ssl, buffer, sizeof(buffer));
    if (len <= 0) { logError("Failed to receive username
prompt"); goto cleanup; }
    printf("%s", buffer);
    fgets(buffer, sizeof(buffer), stdin);
    sendMessage(ssl, buffer);

    len = receiveMessage(ssl, buffer, sizeof(buffer));
    if (len <= 0) { logError("Failed to receive password
prompt"); goto cleanup; }
    printf("%s", buffer);
    disableEcho();
    fgets(buffer, sizeof(buffer), stdin);
    enableEcho();
    sendMessage(ssl, buffer);

    len = receiveMessage(ssl, buffer, sizeof(buffer));
    if (len <= 0 || strcmp(buffer, "HTTPS_SERVER> ") != 0) {
logError("Login failed"); goto cleanup; }

    printf("%s", buffer);
    while (1) {
        fgets(buffer, sizeof(buffer), stdin);
        char command[1024];
        strcpy(command, buffer);
        char* cmd = strtok(command, " \n");
        if (!cmd) { printf("ERROR Empty command\nHTTPS_SERVER>
"); continue; }

        if (strcmp(cmd, "get") == 0) {
            char* filename = strtok(NULL, " \n");
            char* extra = strtok(NULL, " \n");
            if (!filename || extra) { printf("ERROR Invalid get
syntax: get <filename>\nHTTPS_SERVER> "); continue; }
        } else if (strcmp(cmd, "put") == 0) {
            char* filename = strtok(NULL, " \n");
            char* size_str = strtok(NULL, " \n");
            char* extra = strtok(NULL, " \n");
            if (!filename || !size_str || extra) { printf("ERROR
Invalid put syntax: put <filename> <size>\nHTTPS_SERVER> ");
continue; }
        }

        sendMessage(ssl, buffer);

        if (strcmp(cmd, "put") == 0) {
            char* filename = strtok(NULL, " \n");
            char* size_str = strtok(NULL, " \n");
            if (filename && size_str) {
                int size = atoi(size_str);
                if (size <= 0) { printf("ERROR Invalid
size\nHTTPS_SERVER> "); continue; }
                printf("Enter exactly %d bytes for %s: ", size,
filename);
                char* content = malloc(size + 1);
                int read_bytes = 0;
                while (read_bytes < size) {
                    int c = getchar();
                    if (c == EOF || c == '\n') break;
                    content[read_bytes++] = c;
                }
                if (read_bytes != size) { printf("ERROR Entered
%d bytes, expected %d\nHTTPS_SERVER> ", read_bytes, size);
free(content); continue; }
                content[size] = '\0';
                SSL_write(ssl, content, size);
                free(content);
                while (getchar() != '\n'); // Clear buffer
            }
        }

        while (1) {
            len = receiveMessage(ssl, buffer, sizeof(buffer));
            if (len <= 0) { logError("Connection closed"); goto
cleanup; }
            buffer[len] = '\0';
            if (strstr(buffer, "OK") == buffer) {
                if (strcmp(cmd, "get") == 0)
handleFileDownload(ssl, buffer);
                printf("%sHTTPS_SERVER> ", buffer);
                fflush(stdout);
                break;
            } else if (strcmp(buffer, "END\n") == 0 ||
strcmp(buffer, "HTTPS_SERVER> ") == 0) {
                printf("HTTPS_SERVER> ");
                fflush(stdout);
                break;
            } else if (strstr(buffer, "<html>")) {
                printf("ERROR Server error\nHTTPS_SERVER> ");
            } else {
                printf("%s", buffer);
            }
        }
    }

cleanup:
    SSL_shutdown(ssl); SSL_free(ssl); close(sock);
SSL_CTX_free(ctx);
    return 0;
}
```

- `reverseProxy.c` **(Fedora):**
    - **Changes:** Validates command syntax, returns clear errors.
    - **Content:**

```
#include "tlsServer.h"
#include "pamAuth.h"
#include "httpClient.h"
#include "utils.h"
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

SSL_CTX* globalCtx;

void* handleClient(void* arg) {
    int clientFd = (int)(intptr_t)arg;
    SSL* ssl = SSL_new(globalCtx);
    SSL_set_fd(ssl, clientFd);

    if (SSL_accept(ssl) <= 0) { logError("SSL_accept failed");
SSL_free(ssl); close(clientFd); return NULL; }
    if (handleLogin(ssl)) commandLoop(ssl);

    SSL_shutdown(ssl); SSL_free(ssl); close(clientFd);
    return NULL;
```

```
}

void commandLoop(SSL* ssl) {
    char buffer[1024];
    while (1) {
        sendMessage(ssl, "HTTPS_SERVER> ");
        int len = receiveMessage(ssl, buffer, sizeof(buffer));
        if (len <= 0) break;

        buffer[len] = '\0';
        char* cmd = strtok(buffer, " \n");
        if (!cmd) { sendMessage(ssl, "ERROR Empty command\n");
continue; }

        if (strcmp(cmd, "exit") == 0) break;
        else if (strcmp(cmd, "ls") == 0) {
            char* extra = strtok(NULL, " \n");
            if (extra) sendMessage(ssl, "ERROR Invalid ls
syntax\n");
            else listFiles(ssl);
        } else if (strcmp(cmd, "get") == 0) {
            char* filename = strtok(NULL, " \n");
            char* extra = strtok(NULL, " \n");
            if (!filename) sendMessage(ssl, "ERROR Missing
filename\n");
            else if (extra) sendMessage(ssl, "ERROR Invalid get
syntax: get <filename>\n");
            else getFile(ssl, filename);
        } else if (strcmp(cmd, "put") == 0) {
            char* filename = strtok(NULL, " \n");
            char* size_str = strtok(NULL, " \n");
            char* extra = strtok(NULL, " \n");
            if (!filename || !size_str) sendMessage(ssl, "ERROR
Missing filename or size\n");
            else if (extra) sendMessage(ssl, "ERROR Invalid put
syntax: put <filename> <size>\n");
            else {
                int size = atoi(size_str);
                if (size <= 0) sendMessage(ssl, "ERROR Invalid
size\n");
                else putFile(ssl, filename, size);
            }
        } else sendMessage(ssl, "ERROR Unknown command\n");
    }
}

int main() {
    globalCtx = setupTlsServer("certs/reverse_proxy.crt",
"certs/reverse_proxy.key");
    if (!globalCtx) { fprintf(stderr, "Failed to setup TLS
server\n"); return 1; }

    int serverFd = setupServerSocket(8443);
    if (serverFd < 0) { SSL_CTX_free(globalCtx); return 1; }
    while (1) {
        int clientFd = acceptClient(serverFd);
        if (clientFd < 0) continue;

        pthread_t thread;
        if (pthread_create(&thread, NULL, handleClient, (void*)
(intptr_t)clientFd) != 0) close(clientFd);
        pthread_detach(thread);
    }

    SSL_CTX_free(globalCtx); close(serverFd);
    return 0;
}
```

- `httpClient.c` **(Fedora):**
  - **Changes:** Filters `../` from `ls`, handles Nginx `PUT` errors.
  - **Content:**

```
#include "httpClient.h"
#include "tlsClient.h"
#include "utils.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void listFiles(SSL* clientSsl) {
    SSL_CTX* ctx = setupTlsClient("certs/ca.crt");
    if (!ctx) { sendMessage(clientSsl, "ERROR Failed to setup TLS
client\n"); return; }

    int sock = connectToServer("localhost", 443);
    if (sock < 0) { sendMessage(clientSsl, "ERROR Server
connection failed\n"); SSL_CTX_free(ctx); return; }

    SSL* ssl = SSL_new(ctx);
    SSL_set_fd(ssl, sock);
    if (SSL_connect(ssl) <= 0) { sendMessage(clientSsl, "ERROR
Server connection failed\n"); goto cleanup; }

    char request[256];
    snprintf(request, sizeof(request), "GET / HTTP/1.1\r\nHost:
localhost\r\n\r\n");
    SSL_write(ssl, request, strlen(request));

    char buffer[4096] = {0};
    int len = SSL_read(ssl, buffer, sizeof(buffer) - 1);
    if (len <= 0) { sendMessage(clientSsl, "ERROR Failed to read
server response\n"); goto cleanup; }
    buffer[len] = '\0';
    char* pos = buffer;
    while ((pos = strstr(pos, "<a href=\"")) != NULL) {
        pos += 9;
        char* end = strchr(pos, '"');
        if (!end) break;
        *end = '\0';
        if (strcmp(pos, ".") != 0 && strcmp(pos, "..") != 0 &&
pos[0] != '/' && pos[0] != '\0') {
            sendMessage(clientSsl, pos);
            sendMessage(clientSsl, "\n");
        }
        pos = end + 1;
    }
    sendMessage(clientSsl, "END\n");

cleanup:
    SSL_shutdown(ssl); SSL_free(ssl); close(sock);
SSL_CTX_free(ctx);
}

void getFile(SSL* clientSsl, const char* filename) {
    SSL_CTX* ctx = setupTlsClient("certs/ca.crt");
    if (!ctx) { sendMessage(clientSsl, "ERROR Failed to setup TLS
client\n"); return; }

    int sock = connectToServer("localhost", 443);
    if (sock < 0) { sendMessage(clientSsl, "ERROR Server
connection failed\n"); SSL_CTX_free(ctx); return; }

    SSL* ssl = SSL_new(ctx);
    SSL_set_fd(ssl, sock);
    if (SSL_connect(ssl) <= 0) { sendMessage(clientSsl, "ERROR
Server connection failed\n"); goto cleanup; }

    char request[256];
    snprintf(request, sizeof(request), "GET /%s HTTP/1.1\r\nHost:
localhost\r\n\r\n", filename);
    SSL_write(ssl, request, strlen(request));

    char buffer[4096];
    int len = SSL_read(ssl, buffer, sizeof(buffer) - 1);
    if (len <= 0) { sendMessage(clientSsl, "ERROR Failed to read
server response\n"); goto cleanup; }
    buffer[len] = '\0';

    char* body = strstr(buffer, "\r\n\r\n");
    if (!body || strstr(buffer, "404")) sendMessage(clientSsl,
"ERROR File not found\n");
    else {
        body += 4;
        int size = len - (body - buffer);
        char response[256];
        snprintf(response, sizeof(response), "OK %d\n", size);
        sendMessage(clientSsl, response);
        SSL_write(clientSsl, body, size);
    }

cleanup:
    SSL_shutdown(ssl); SSL_free(ssl); close(sock);
SSL_CTX_free(ctx);
}

void putFile(SSL* clientSsl, const char* filename, int size) {
    char* content = malloc(size + 1);
    int received = 0;
    while (received < size) {
        int len = SSL_read(clientSsl, content + received, size -
received);
        if (len <= 0) { fprintf(stderr, "putFile: Failed to read
%d bytes, received %d\n", size, received); sendMessage(clientSsl,
"ERROR File upload failed\n"); free(content); return; }
        received += len;
    }
    content[size] = '\0';

    SSL_CTX* ctx = setupTlsClient("certs/ca.crt");
    if (!ctx) { sendMessage(clientSsl, "ERROR Failed to setup TLS
client\n"); free(content); return; }

    int sock = connectToServer("localhost", 443);
    if (sock < 0) { sendMessage(clientSsl, "ERROR Server
connection failed\n"); free(content); SSL_CTX_free(ctx); return;
}

    SSL* ssl = SSL_new(ctx);
    SSL_set_fd(ssl, sock);
    if (SSL_connect(ssl) <= 0) { fprintf(stderr, "putFile: Failed
to connect to Nginx\n"); sendMessage(clientSsl, "ERROR Server
connection failed\n"); goto cleanup; }

    char request[512];
    snprintf(request, sizeof(request), "PUT /%s HTTP/1.1\r\nHost:
localhost\r\nContent-Length: %d\r\n\r\n", filename, size);
    SSL_write(ssl, request, strlen(request));
    SSL_write(ssl, content, size);

    char buffer[256];
    int len = SSL_read(ssl, buffer, sizeof(buffer) - 1);
    if (len <= 0) { sendMessage(clientSsl, "ERROR Failed to read
server response\n"); goto cleanup; }
    buffer[len] = '\0';
    if (strstr(buffer, "201") || strstr(buffer, "200"))
sendMessage(clientSsl, "OK\n");
    else { fprintf(stderr, "putFile: Nginx response: %s\n",
buffer); sendMessage(clientSsl, "ERROR Upload failed\n"); }

cleanup:
    free(content); SSL_shutdown(ssl); SSL_free(ssl); close(sock);
SSL_CTX_free(ctx);
}
```

Nginx Configuration

- **File:** `/etc/nginx/conf.d/https_server.conf`
- **Content:**

```
server {
    listen 443 ssl;
    server_name localhost;
    ssl_certificate /home/techn/Assignment_03/certs/server.crt;
    ssl_certificate_key /home/techn/Assignment_03/certs/server.key;
    location / {
        root /var/www/files;
        autoindex on;
        dav_methods PUT;
    }
}
```

- **Steps:**
  1. Edit: `sudo nano /etc/nginx/conf.d/https_server.conf`
  2. Test: `sudo nginx -t`
  3. Reload: `sudo systemctl reload nginx`
- **WebDAV Installation:**

```
sudo dnf install nginx-mod-http-dav-ext
sudo systemctl restart nginx
```

Permissions

- **Command:**

```
ls -ld /var/www/files
sudo chown nginx:nginx /var/www/files
sudo chmod 755 /var/www/files
sudo chcon -R -t httpd_sys_rw_content_t /var/www/files
```

- **Verification:** Ensure `drwxr-xr-x nginx nginx` and SELinux context.

Revised Step-by-Step Guide

Prerequisites

- Install dependencies:
  - FreeBSD: `pkg install openssl gcc`
  - Fedora: `sudo dnf install openssl-devel gcc nginx nginx-mod-http-dav-ext`
- Generate certificates (if missing):

  ```
  ./generate_cert.sh
  ```

Step-by-Step Setup

1. **Fedora VM (192.168.118.131):**
   - **Navigate to Directory:**

     ```
     cd /home/techn/Assignment_03
     ```

   - **Update Files:**
     - Replace `src/reverseProxy.c` and `src/httpClient.c` with revised versions.
   - **Compile:**

     ```
     make clean
     make
     ```

   - **Start Nginx:**

     ```
     sudo systemctl start nginx
     sudo systemctl status nginx
     ```

   - **Start Reverse Proxy:**

     ```
     ./reverseProxy > proxy.log 2>&1 &
     ```

   - **Verify Ports:**

     ```
     ss -tuln | grep '443\|8443'
     ```

- **Prepare Files:**
    - Ensure `/var/www/files/hello.txt` exists:

      ```
      echo "Hii, I am Bhargav Jani..." | sudo tee
      /var/www/files/hello.txt
      ```

2. **FreeBSD VM (192.168.118.134):**
    - **Navigate to Directory:**

      ```
      cd /home/techass/Assignment_03
      ```

    - **Update Files:**
        - Replace `src/client.c` with revised version.
    - **Compile:**

      ```
      gmake -f Makefile clean
      gmake -f Makefile client
      ```

    - **Run Client:**

      ```
      ./client
      ```

    - **Login:**
        - Username: `techn` or `testuser`
        - Password: `TECH` or `test123`
        - Expect: `HTTPS_SERVER>`
3. **Testing Commands:**
    - **ls:**

      ```
      ls
      ```

        - Expected: `file1.txt`, `hello.txt`.
    - **get:**

      ```
      ```

```
get hello.txt
```

  - Expected: `OK`, check `downloaded_file` with `cat downloaded_file`.

- **put:**

```
put newfile.txt 10
Enter exactly 10 bytes for newfile.txt: helloworld
```

  - Expected: `OK`, verify on Fedora with `cat /var/www/files/newfile.txt`.

- **exit:**

```
exit
```

  - Expected: Client terminates.

4. **Capture Traffic:**

   - FreeBSD:

```
sudo tcpdump -i em0 port 8443 -w final.pcap
```

     - Run client commands, stop with Ctrl+C.

   - Analyze in Wireshark (filter `tcp.port == 8443 && ssl`).

5. **Verify Fixes:**

   - Test error cases:
     - `get hello.txt 10`: `ERROR Invalid get syntax`.
     - `put hello.txt`: `ERROR Missing filename or size`.
     - `get h.txt`: `ERROR File not found`.

Troubleshooting

- **put Fails with `ERROR Upload failed`:**
  - Check: `sudo cat /var/log/nginx/error.log`.
  - Test: `curl -X PUT --cacert certs/ca.crt -d "test" https://localhost/testcurl.txt`.
  - Fix: Ensure WebDAV and permissions.
- **get Fails:** Verify file in `/var/www/files`.
- **ls Shows `../`:** Use updated `httpClient.c`.
- **Client Hangs:** Restart reverse proxy, check `telnet 192.168.118.131 8443`.

Submission Preparation

- **Files:**
    - Fedora: `src/`, `certs/`, `Makefile`, `Readme.md`, `reverseProxy`, `proxy.log`, `tls_handshake_1.pcap`.
    - FreeBSD: `src/`, `certs/ca.crt`, `Makefile`, `Readme.md`, `client`, `client.log`, `final.pcap`.
- **Update Readme.md:**

```
# Readme.md
## Assignment 3: Secure Reverse Proxy
- **Issue:** Initial `put` and `get` had syntax errors, `404` on
`put`.
- **Fix:** Updated `client.c`, `reverseProxy.c`, `httpClient.c` for
correct syntax and error handling.
- **Usage:**
  - `ls`: List files.
  - `get <filename>`: Download to `downloaded_file`.
  - `put <filename> <size>`: Upload, enter exact bytes.
  - `exit`: Quit.
```