دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

عنوان:

گزارش پروژه‌ی پایانی شبکه‌های کامپیوتری

اعضای گروه

علی مهدوی فر - ۹۸۱۰۶۰۷۲

ایمان علیپور - ۹۸۱۰۲۰۲۴

نام درس

شبکه‌های کامپیوتری

Course Name

Computer Networks

نیم‌سال اول ۱۴۰۱-۱۴۰۲

نام استاد درس

دکتر مهدی جعفری

فهرست مطالب

چکیده

در این پروژه قصد داریم تا با قرار دادن یک Xclient و یک Xserver میان یک Clinet و Server پیام‌های رد و بدل شده میان Clinet و Server را به صورت درخواست‌های HTTP نشان دهیم تا امکان تمایز آن‌ها از درخواست‌های عادی و روزمره‌ی دیگر از بین برود. در ادامه با اضافه‌کردن یک لایه امنیتی درخواست‌ها را به صورت HTTPS می‌فرستیم تا اولا در صورت شنود پیام‌ها کسی متوجه محتوای آن‌ها نشود و همچنین اینکه کسی نتواند از ای رابط کاربری سوء استفاده کند و فقط افراد مطمئن به آن‌ها دسترسی داشته باشند و بتوانند پیام رد و بدل کنند.

واژه‌های کلیدی: Client, Server, TLS, UDP, TCP

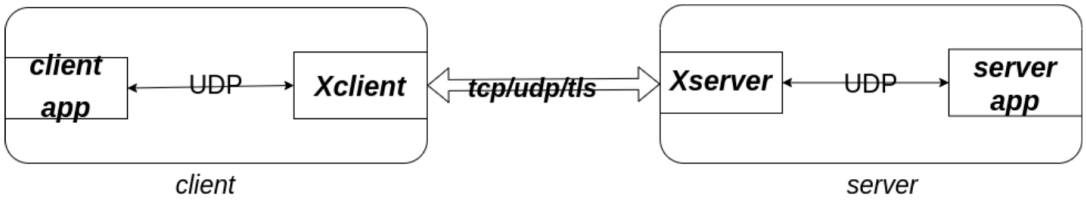
۱ مقدمه

در دنیای روزمره‌ی امروزی سطح حریم خصوصی کاربران وب بسیار پایین آمده است، همچنین برخی از تشکیلات تلاش می‌کنند توانایی استفاده‌ی افراد از اینترنت را محدود و کنترل کنند، این مسئله علاوه بر چالش‌های نقض حریم خصوصی، چالش‌های دیگری همچون عدم دسترسی به برخی از محتوای آزاد در سطح وب را از بین می‌برد، برای گذر از این مشکل انسان‌ها از VPN یا Virtual Private Networkها استفاده می‌کنند، اما پروتکل‌های موجود عموما قابل تشخیص و جلوگیری هستند، یک روش حل این مشکل مخفی‌سازی پکت‌ها به گونه‌ای است که قابل تشخیص نباشند، این ایده به Obfuscation نیز معروف است.

روشی که در این پروژه قصد پیاده‌سازی آن را داریم نیز از همین ایده بهره می‌گیرد و قصد داریم ارتباط بین یک Clinet و Server را به گونه‌ای مخفی کنیم و ترافیک عبوری را مانند ترافیک عادی اینترنت در قالب درخواست‌های HTTP مخفی می‌کنیم.

۲ توپولوژی مسئله

در این پروژه از توپولوژی گفته شده در داک استفاده می‌کنیم به این نحو که یک Clinet و Server داریم که Clinet از طریق XClinet به یک XServer وصل است و بسته‌هایش را به آن می‌فرستد، XClinet در ادامه بسته‌ها را در درون یک درخواست HTTP قرار می‌دهد و به XServer ارسال می‌کند، XServer پس از دریافت بسته‌ها، محتوای آن‌ها را از درون درخواست HTTP خارج ساخته و به Server تحویل می‌دهد، به همین نحو پاسخ Server به Client باز می‌گردد. توپولوژی مسئله را میتوانید در شکل ۱ مشاهده کنید.



شکل ۱: توپولوژی مسئله که متشکل از یک Xclient و یک Xserver میان یک Clinet و Server است.

۳ ایجاد Certificate برای اتصال TLS

برای اتصال TLS نیاز به یک Certificate، یک فایل کلید خصوصی و یک فایل امضای دیجیتالی داریم. [۱] برای ایجاد این فایل‌ها از پکیج openssl استفاده می‌کنیم، این کتابخانه به صورت پیش‌فرض روی لینوکس وجود دارد اما در صورت عدم وجود می‌توان با استفاده از دستورهای زیر آن را نصب کرد.

```
sudo apt-get update
sudo apt-get install openssl
```

در ادامه با استفاده از این پکیج ابتدا فایل کلید خصوصی را ایجاد می‌کنیم.

```
openssl genrsa -out key.pem 2048
```

خروجی این دستور یک فایل با محتوای زیر است:

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEArdRVmzX53cVpXQwvJDKbLrcA4peNXvamEQKVbuyu91W/XJ2c
MRgD4EQgQsDM3/bWl5IRDwcfcRaZ1USLmpoUekj6AIOi+yCOgbvjH3RIz5SREc0I
JuGgLUxe8uZdyYzstK/Mg6iqgDfEWmFzuiUTlitmZYH3oRs9WhJ3MAuLgRMC9J9E
                    ⋮
2DuiE7tt253fNg2gynH5Aygfz8JiMXS3u6k4vucnLUyP632uiqLZgHp+IRguIZt5
Vu+ZnQKBgQDKDTAw3di/lQKw6g8O/lx2Fd2VOtqD2+qsmb3Wo1L9GFyfE5PICVuL
u3hkfEKQr5qSMPZy42HL3agHZuCHCWNKpcBRtlUiJv6JXWSj+d7bPn2GZ9O2rZ4j
1SMuDnaZE5G5rqKwa/HCFieYzJMnOopHWLgxqIFdtwx+kl/hGB0iJA==
-----END RSA PRIVATE KEY-----
```

حال باید یک درخواست certificate بسازیم و آن را با کلید ساخته شده رمز کنیم، برای این کار از دستور زیر استفاده می‌کنیم:

```
openssl req -new -key key.pem -out signreq.csr
```

در این مرحله از ما اطلاعاتی خواسته می‌شود که آن‌ها را به عنوان ورودی تحویل می‌دهیم.

خروجی این بخش فایل زیر است:

```
-----BEGIN CERTIFICATE REQUEST-----
MIIDFTCCAf0CAQAwgZwxCzAJBgNVBAYTAklSMQ8wDQYDVQQIDAZUZWhyYW4xDzAN
BgNVBAcMBlRlaHJhbjEPMA0GA1UECgwGU2hhcmlmMR0wGwYDVQQLDBRDb21wdXRl
```

```
ciBFbmdpbmVlcmluZzESMBAGA1UEAwwJbG9jYWxob3N0MScwJQYJKoZIhvcNAQkB
                    ⋮
xS8rgfnaybqI0sBGdUSi0eIN2WQ6N9lvdl22FkwAyh4WFd355bKh5dMrdfx0ig5j
anZysHL/lD5WNscAE9o29aDAEbI7RD+kuV1aGJ7u5q5FjVhqBnGbU2EzIxAdFwVR
qBb7FVvwwLI/EoxN5K1809YJ1NFrU/eVU04hH8oSlj74DjRJjr4Sckg4RXI+6r+I
xX1BPXJWyK9O34dMUzep7nhoE6GqtESAyQ==
-----END CERTIFICATE REQUEST-----
```

در مرحله‌ی بعد باید درخواست certificate را با استفاده از همان کلید، امضا کنیم، خروجی این بخش یک certificate است که برای مدت مشخص شده معتبر است. این کار را با دستور زیر انجام می‌دهیم:

```
openssl x509 -req -days 365 -in signreq.csr -signkey key.pem -out certificate.pem
```

در واقع یک فایل certificate معتبر برای ۳۶۵ روز خروجی این بخش است. خروجی این بخش در حالت عادی اطلاعات زیر است:

```
-----BEGIN CERTIFICATE-----
MIIDwTCCAqkCFC9i/gPjqA7VUusySQo59mxY1DYyMA0GCSqGSIb3DQEBCwUAMIGc
MQswCQYDVQQGEwJJUjEPMA0GA1UECAwGVGVocmFuMQ8wDQYDVQQHDAZUZWhyYW4x
DzANBgNVBAoMBlNoYXJpZjEdMBsGA1UECwwUQ29tcHV0ZXIgRW5naW5lZXJpbmcx
                    ⋮
QPRlptq411fq0AKTzxuNzFblGxmCYP32l3DP1w8jGqZABRpRYuZ+NEqzroWygzu1
YJxfX1TXpYPXHbWP8oA1CLDnCIEcwtbnZFUgHlmVzggO9mUgjIqulr1avR8SO9LR
D/3ziAbf9a8kX3rP712n5hMaSfnugcQWi+kQg0l58HmkTAbYe0DSK449yrzFZRcI
WD7xzHk=
-----END CERTIFICATE-----
```

اما اگر از دستور زیر استفاده کنیم می‌توانیم خروجی‌های قابل فهمی را ببنیم.

```
openssl x509 -text -noout -in certificate.pem
```

توجه کنید برای این کار به کلید مخفی‌ای که در ابتدا تولید کردیم نیاز داریم و هرکسی آن را ندارد و امکان خواندن این فایل را نیز نخواهد داشت.

```
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
```

```
        3f:08:79:11:11:97:4e:a8:e9:75:f6:db:b5:6c:c3:da:37:0f:ad:d1
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: C = IR, ST = Tehran, L = Tehran, O = Sharif, OU = Computer Engineering
, CN = localhost, emailAddress = mahdavifar2002[at]gmail.com
    Validity
        Not Before: Feb 11 19:38:51 2023 GMT
        Not After : Feb 11 19:38:51 2024 GMT
    Subject: C = IR, ST = Tehran, L = Tehran, O = Sharif, OU = Computer
Engineering, CN = localhost, emailAddress = mahdavifar2002[at]gmail.com
    Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
            Public-Key: (2048 bit)
            Modulus:
                00:ad:d4:55:9b:35:f9:dd:c5:69:5d:0c:2f:24:32:
                9b:2e:b7:00:e2:97:8d:5e:f6:a6:11:02:95:6e:ec:
                ae:f7:55:bf:5c:9d:9c:31:18:03:e0:44:20:42:c0:
                cc:df:f6:d6:97:92:11:0f:07:1f:71:16:99:d5:44:
                8b:9a:9a:14:7a:48:fa:00:83:a2:fb:20:8e:81:bb:
                e3:1f:74:48:cf:94:91:11:cd:08:26:e1:a0:2d:4c:
                5e:f2:e6:5d:c9:8c:ec:b4:af:cc:83:a8:aa:80:37:
                c4:5a:61:73:ba:25:13:96:2b:66:65:81:f7:a1:1b:
                3d:5a:12:77:30:0b:8b:81:13:02:f4:9f:44:e9:2e:
                de:7a:d4:5a:d2:ad:8a:60:c3:2c:08:15:48:b7:dc:
                de:cd:37:07:9c:96:f3:76:9d:12:38:d2:76:11:8b:
                70:e7:89:98:f0:f5:82:5e:fb:ed:d7:18:eb:2d:ab:
                8e:74:62:15:da:5f:13:43:6c:af:d2:d7:b2:90:b9:
                c1:36:1b:43:62:32:e7:a4:6e:b1:5f:da:fd:b3:9a:
                8d:13:d0:dd:5b:e5:29:3a:b7:c6:fe:d7:f9:e4:e3:
                1c:c8:ca:65:f1:2f:73:9c:67:0a:f1:29:33:2a:0e:
                b6:cd:1d:8d:c7:38:71:03:06:67:bf:02:9c:d2:a4:
                37:af
            Exponent: 65537 (0x10001)
Signature Algorithm: sha256WithRSAEncryption
Signature Value:
    4e:6e:95:7d:a7:77:85:f5:e0:ca:57:29:c8:a3:76:32:14:a0:
    23:d8:29:ab:eb:00:82:2d:fe:2d:b8:10:2e:81:61:36:4b:5d:
```

```
        05:6d:1c:7a:66:d4:b6:1a:62:97:0f:46:e5:01:1e:d0:49:0e:
        9c:22:05:03:97:37:86:cb:23:8b:46:64:a6:2b:c0:6d:62:a8:
        f9:53:a2:bf:92:d3:eb:c9:3f:1f:64:4f:36:4f:31:c5:75:57:
        ca:7f:7d:be:73:3f:85:1c:f1:93:83:c5:4b:2f:73:44:32:b8:
        61:73:db:25:d4:51:2b:aa:04:65:5b:16:11:3a:f0:0d:87:ed:
        fe:1f:c5:21:43:9a:49:e6:72:b2:db:af:05:b8:e8:eb:c2:a6:
        0f:a1:1c:97:95:83:62:cd:b3:50:f6:5d:7b:da:dd:10:71:d4:
        86:95:9a:62:da:59:08:e7:1e:29:73:3e:e8:78:28:1f:00:33:
        4f:3f:be:6a:d5:08:ba:e2:3a:e2:47:1f:83:e2:66:09:f0:57:
        92:c4:8e:4d:68:bb:54:a3:32:91:d1:03:5e:07:dc:ad:ec:e4:
        58:56:f8:f0:19:8b:93:22:1b:be:08:16:6f:65:1c:68:ce:9f:
        9d:47:54:4a:3e:40:c8:1f:21:a3:9e:17:a5:90:e5:c4:0b:29:
        03:b9:f9:fd
```

حال برای اتصال یک Client به یک Server کافی است Client داده‌ی رمزشده‌ی Certificate را داشته باشدو اینگونه این دو می‌توانند ارتباط TLS برقرار کنند.

۴ پیاده‌سازی اجزای مسئله

۴-۱ ثابت‌های فایل Constants.py

در فایل Constants.py، ثابت‌های زیر را تعریف کردیم:

```
X_SERVER_DOMAIN_NAME = "localhost"

XCLIENT_UDP_PORT = 7000
XCLIENT_TCP_PORT = 6001

XSERVER_TCP_PORT = 443

SERVER_DOMAIN_NAME = "localhost"

BUFFER_SIZE = 2048
```

در این فایل، دامنه‌ای که Xserver آنجا قرار دارد را در اولین خط مشخص کرده‌ایم، همچنین

پورت‌های مختلفی که اجزای کد از آن‌ها استفاده خواهند کرد را نیز همینجا تعریف کرده‌ایم تا اگر نیاز به تغییری بود در همین فایل تغییرات را اعمال کنیم. به ترتیب ابتدا پورتی که Xclient روی آن بسته‌های Client را دریافت می‌کند، سپس پورتی که Xclient بسته‌های ارسالی از Xserver را دریافت می‌کند و همچنین پورتی که Xserver روی آن بسته‌های Xclient را دریافت می‌کند و پورتی که Xserver بسته‌های ارسالی از Server را دریافت می‌کند مشخص شده‌اند. در انتها هم اندازه‌ی سایز بافر مشخص شده است. پورت‌های Clientها و Server متغیر هستند و در هنگام ران شدن برنامه آن‌ها را به عنوان ورودی برنامه‌ها دریافت می‌کنیم.

۲-۴ ایجاد Xclient

همانطور که پیش‌تر نیز بیان کردیم، وظیفه‌ی Xclient دریافت بسته‌های ارسال شده توسط Client، اضافه کردن هدر HTTP به این بسته‌ها و فرستادن آن‌ها به سمت Xserver است، همچنین در مسیر برگشت نیز بسته‌های ارسالی از سمت Xserver را unpack می‌کند و به سمت Client می‌فرستد.

```
1 import socket
2 import ssl
3 from PrettyLogger import logger_config
4 import Constants
5 import threading
6 import time
```

برای ساخت Xclient از کتابخانه‌های بالا استفاده می‌کنیم. کتابخانه‌ی ssl برای برقراری ارتباط TLS به کمک فایل‌های ساخته شده که پیش از این توضیحات آن‌ها را دادیم می‌باشد. همچنین از کتابخانه‌های socket برای ساخت سوکت‌ها و برقراری اتصال و همچنین از کتابخانه‌ی time برای ساخت درخواست HTTP استفاده می‌کنیم. همچنین برای وجود مسیر دوطرفه از threading استفاده می‌کنیم تا هر مسیر را یک ریسمان هندل کند. در نهایت از کتابخانه‌ی PrettyLogger برای لاگ انداختن استفاده می‌کنیم.

```
9 log = logger_config("webserver")
10
11 xserver_socket: socket.socket
12 destination_lut = {}
```

این بخش مربوط به متغیرهای global ماست، همچنین در خط اول کانفیگ لاگر را یک کانفیگ

مناسب قرار می‌دهیم تا لاگ‌ها زیباتر نمایش داده شوند. در این بخش یک سوکت گلوبال برای ارتباط با Xserver داریم، علت گلوبال بودن افزایش robustness بود تا اگر طرف دیگر خراب شود وقتی Xserver مجددا بالا می‌آید سوکت جدید گلوبال باشد تا تردهای دیگر از آن استفاده کنند. destination_lut هم متادیتای مورد نیاز برای بازگرداندن پکت‌های دریافتی به Client درست را در بر دارد.

تابع main کد این بخش به شکل زیر است:

```
90 if __name__ == "__main__":
91     https_socket = establish_HTTPS_connection()
92 
93     # This thread listens on XCLIENT_UDP_PORT and forwards incomming packets to
       XSERVER after adding header
94     client_handler_thread = threading.Thread(target=client_handler, args=())
95     client_handler_thread.start()
96 
97     # This thread reads from HTTPS socket and forwards incomming packets to
       CLIENT_PORT after removing custom header
98     xserver_handler_thread = threading.Thread(target=xserver_handler, args=())
99     xserver_handler_thread.start()
```

ابتدا اتصال HTTPS برقرار می‌شود و سپس با ساخت دو ریسمان، مسیرهای مختلف بصورت همزمان کار می‌کنند.

در ادامه توابع توصیف شده را بررسی خواهیم کرد.

```
14 def establish_HTTPS_connection() -> socket.socket:
15     global xserver_socket
16     sleep_time = 1
17     while True:
18         try:
19             hostname = 'localhost'
20             context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
21             context.load_verify_locations("./keys/certificate.pem")
22 
23             '''
24             # Use two lines below instead of line above if you don't want to check
       self-signed certificate
```

```
            context.verify_mode = ssl.CERT_NONE
            context.check_hostname = False
            '''

            raw_sock = socket.create_connection((hostname, 443))
            https_socket = context.wrap_socket(raw_sock, server_hostname = Constants.
    X_SERVER_DOMAIN_NAME)
            xserver_socket = https_socket

            log.info("Conected to Xserver successfully.")
            return https_socket
        except ConnectionRefusedError:
            log.warning(f"Xserver is not responding... retrying in {sleep_time}")
            time.sleep(sleep_time)
            sleep_time *= 2
```

این تابع وظیفه‌ی اتصال Xclient به Xserver به صورت HTTPS را فراهم می‌کند. ابتدا با استفاده از فایل Certificate ساخته شده اتصال HTTPS را با Xserver برقرار می‌کند. همچنین توجه کنید اگر اتصال قطع شود با مکانیزم exponential backoff تلاش می‌کند تا اتصال را دوباره برقرار کند. در نهایت به این توجه کنید که برای افزایش robustness، سوکت ایجاد شده را در یک متغیر گلوبال ذخیره می‌کند تا ترد دیگر بتواند از آن استفاده کند. (در هر دو مسیر یا باید از این سوکت بخوانیم، یا در آن بنویسیم.)

```
def client_handler():
    global xserver_socket, destination_lut
    UDP_server_socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
    UDP_server_socket.bind(("localhost", Constants.XCLIENT_UDP_PORT))



    while True:
        bytes_address_pair = UDP_server_socket.recvfrom(Constants.BUFFER_SIZE)
        message = bytes_address_pair[0].decode("ascii")

        if message[0:4] == "\0\0\0\0":
            _, server_address, server_port = message.split("\0\0\0\0", maxsplit=2)
            client_address, client_port = bytes_address_pair[1]
```

```
            destination_lut[client_port] = (server_address, server_port)
            https_message = f"GET / HTTP/1.1\r\nHost: localhost\r\nContent-Type:
    application/zip\r\nContent-Length: {len(message)}\r\n\r\n{client_port}\r\n\r\n{
    destination_lut[client_port][0]}\r\n\r\n{destination_lut[client_port][1]}\r\n\r\n{
    message}"
            xserver_socket.sendall(https_message.encode("ascii"))
            log.info(f"client with {client_address}:{client_port} wants to connect to
    {server_address}:{server_port}.")
        else:
            client_port = bytes_address_pair[1][1]
            https_message = f"PUT / HTTP/1.1\r\nHost: localhost\r\nContent-Type:
    application/zip\r\nContent-Length: {len(message)}\r\n\r\n{client_port}\r\n\r\n{
    destination_lut[client_port][0]}\r\n\r\n{destination_lut[client_port][1]}\r\n\r\n{
    message}"
            xserver_socket.sendall(https_message.encode("ascii"))
```

این تابع. وظیفه دارد پیام‌ها را از Client دریافت کند و آن‌ها را در یک درخواست HTTP قرار دهد و به سمت Xserver بفرستد. ساختار این درخواست از جنس PUT می‌باشد و هدرهای Host و Content-Type را نیز دارد، در انتها طول پیام و سپس دامنه و پورت مقصد و پورت مبدا قرار دارند و سپس خود پیام آمده است. همچنین توجه کنید اولین پیام ارسلای از طرف Client باید پورتی که روی آن گوش می‌دهد و پورت و آدرس مقصد را در آن مشخص کند، سپس ما در یک دیکشنری این اطلاعات را برای بازگردادن پاسخ‌ها به Client ذخیره می‌کنیم. اسم این دیکشنری destinaiton_lut است که یک look up table است. درخواست‌های GET برای پیام اول و در واقع شناسایی مبدا و مقصد و شروع یک کانکشن جدید است و پیام‌های PUT پیام‌های عادی ارسالی از طرف Client به سمت Server.

```
def xserver_handler():
    global xserver_socket
    try:
    # Create a UDP socket at server side, (ipv4, UDP)
        UDP_client_socket = socket.socket(family=socket.AF_INET, type=socket.
    SOCK_DGRAM)

        while True:
            buffer = xserver_socket.recv(Constants.BUFFER_SIZE).decode("ascii")
```

```
            log.info(f"message from xserver: {buffer.encode('ascii')}")

            arr = buffer.split("\r\n\r\n", maxsplit=2)
            https_header = arr[0]
            client_port = int(arr[1])
            UDP_message = arr[2]

      # Send to client using created UDP socket
            bytesToSend = str.encode(UDP_message)
            UDP_client_socket.sendto(bytesToSend, ("127.0.0.1", client_port))

    except KeyboardInterrupt:
        UDP_client_socket.close()
        log.info(f"XServer disconnected.")
    except IndexError:
        log.info(f"connection with server failed.")
        establish_HTTPS_connection()
        xserver_handler()   # Keep Xserver handler thread alive
```

این تابع وظیفه دارد هدر بسته‌هایی که از سمت Xserver آمده‌اند را حذف کند و آن‌ها را به Client تحویل دهد. در ابتدا سوکت UDP را ایجاد می‌کند و تلاش می کند از سوکت https پیامی را بخواند، هرگاه پیامی را خواند، هدر آن را با استفاده از یک عملیاتsplit حذف می‌کند و متادیتای مورد نیاز مانند پورت مقصد را نیز از پیام دریافت می‌کند و سپس پیام را به Client می‌فرستد. همچنین در صورت قطع شدن Xserver تلاش میکند تا دوباره به آن متصل شود.

۳-۴ ایجاد Xserver

در این بخش کدهای مربوط به Xserver را توضیح می‌دهیم، بسیاری از کارهایی که Xclient می‌کند را مشابها باید Xserver نیز انجام دهد، صرفا بجای ساخت یک درخواست HTTP، باید یک پاسخ HTTP ایجاد کند.

```
import socket
import ssl
import threading
from PrettyLogger import logger_config
```

```
import Constants
from email.utils import formatdate
```

برای ساخت Xclient از کتابخانه‌های بالا استفاده می‌کنیم. کتابخانه‌ی ssl برای برقراری ارتباط TLS به کمک فایل‌های ساخته شده که پیش از این توضیحات آن‌ها را دادیم می‌باشد. همچنین از کتابخانه‌های socket برای ساخت سوکت‌ها و برقراری اتصال و همچنین از کتابخانه‌ی email.utils برای ساخت هدر تاریخ پاسخ HTTP استفاده می‌کنیم. همچنین برای وجود مسیر دوطرفه از threading استفاده می‌کنیم تا هر مسیر را یک ریسمان هندل کند. در نهایت از کتابخانه‌ی PrettyLogger برای لاگ انداختن استفاده می‌کنیم.

```
log = logger_config("webserver")

xclient_socket: socket.socket
UDP_socket: socket.socket
destination_lut = {}
UDP_socket_lut = {}
```

این بخش مشابه با Xclient برای متغیرهای گلوبال می‌باشد، در خط اول کانفیگ لاگر را مشخص کرده‌ایم، سپس مشابها برای سوکت بین Xserver و Xclient یک متغیر گلوبال داریم تا در صورت قطع شدن Xclient و اتصال مجدد سوکت جدید در اختیار کل ریسمان‌ها قرار بگیرد. در ادامه دو دیشکنری گلوبال داریم که اولی متادیتای مربوط به باز گردانی پیام به Xclient را دارد و دومی سوکت مربوط به هر Server را نگه می‌دارد، کلید آن نیز پورت مختص Server است.

```
if __name__ == "__main__":
  https_socket = establish_https_connection()

  # This thread reads from HTTPS socket and forwards incomming packets to SERVER_PORT
    after removing custom header
  xclient_handler_thread = threading.Thread(target=https_client_handler, args=(
    https_socket, ))
  xclient_handler_thread.start()
```

تابع main برای Xserver بسیار مشابه با تابع main برای Xclient است. اینجا ریسمان مربوط به مسیر رفت ایجاد می‌شود در واقع صرفا ارتباط با Xclient برقرار می‌شود و برای مسیر برگشت به صورت جداگانه ریسمان ایجاد می‌کنیم.

در ادامه توابع استفاده شده را بررسی می‌کنیم.

```python
def establish_https_connection() -> socket.socket:
  server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

  server_socket = ssl.wrap_socket (server_socket,
  certfile='./keys/certificate.pem', keyfile="./keys/key.pem",
  server_side=True, ssl_version=ssl.PROTOCOL_TLS)

  server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1) # This solves
    address aleady in use issue

  server_socket.bind(('localhost', 443))
  server_socket.listen(5)

  log.info("Server is listening on localhost:443")

  return server_socket
```

این تابع وظیفه‌ی ایجاد اتصال HTTPS با Xclient را دارد. این کار را با استفاده از فایل‌های key.pem و certificate.pem که پیش از این ایجاد کردیم انجام می‌دهد و صحت فرستنده و گیرنده را می‌سنجد. در نهایت با bind کردن روی این پورت، منتظر اتصال Xclient و آمدن پکت‌ها می‌ماند.

```python
def https_client_handler(https_socket: socket.socket):
  global xclient_socket
  try:
    while True:
      client, address = https_socket.accept()
      xclient_socket = client
      handler_thread = threading.Thread(target=xclient_handler, args=(client, address)
    )
      handler_thread.start()
  except KeyboardInterrupt:
    log.warning("terminating server")
    https_socket.close()
```

این تابع وظیفه‌ی گوش ایستادن روی اتصال HTTPS است، هرگاه یک Xclient متصل شود اتصال را برقرار کرده و یک ریسمان برای آن می‌سازد تا پیام‌ها دریافتی از server را هندل کند.

```python
def xclient_handler(client, address):
  log.info(f"Client with address {address} connected.")

  try:
    while True:
      buffer = client.recv(Constants.BUFFER_SIZE).decode("ascii")
      log.info(f"message from xclient: {buffer.encode('ascii')}")

      arr = buffer.split("\r\n\r\n", maxsplit=4)
      https_header = arr[0]
      client_port = int(arr[1])
      server_address = arr[2]
      server_port = int(arr[3])
      UDP_message = arr[4]

      if https_header[0:3] == "GET":
        UDP_server_socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
        UDP_socket_lut[client_port] = UDP_server_socket
        destination_lut[client_port] = (server_address, server_port)

        # This thread listens on UDP_server_socket and forwards incomming packets to XCLIENT after adding header
        server_handler_thread = threading.Thread(target=server_handler, args=(client_port, ))
        server_handler_thread.start()
      elif https_header[0:3] == "PUT":
        # Send to server using created UDP socket
        bytesToSend = str.encode(UDP_message)
        UDP_socket_lut[client_port].sendto(bytesToSend, (server_address, server_port))

  except KeyboardInterrupt or IndexError:
    client.close()
    log.info(f"Client with address {address} disconnected.")
```

این تابع وظیفه‌ی این را دارد که هدر پیام‌های دریافتی از Xclient را حذف کرده و پیام را به

server بدهد. در هنگام حذف کردن هدر، متادیتای قرار داده شده در سمت Xclient را نیز استفاده می‌کند و در جدول‌های look up خود هم سوکت مورد نیاز برای مقصد و هم متادیتای کورد نیاز برای بازگردادن داده به Xclient درست و همچنین قرار دادن متادیتای درست برای اینکه Xclient بتواند به درستی بسته‌را به Client درست منتقل کند. همچنین توجه کنید در هر پیام ارسالی از Xclient که از جنس GET باشد، نشانه‌ی این است که یک کلاینت جدید قصد برقراری اتصال جدید را دارد و اطلاعات مربوطه در دیکشنری‌های look up قرار می‌گیرند، در غیر این صورت و اگر پیام آمده از جنس PUT باشد صرفا به سمت سرور مناسب هدایت می‌شود.

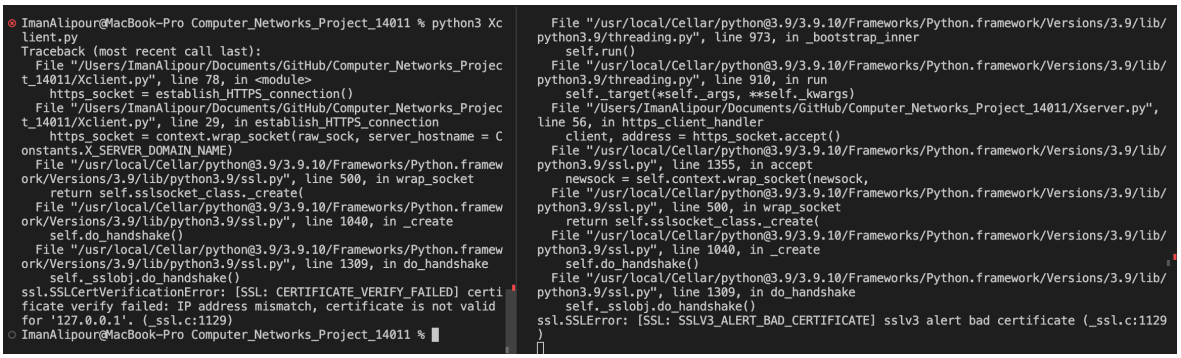
```
78 def server_handler(client_port):
79   global xclient_socket
80   UDP_socket = UDP_socket_lut[client_port]
81
82   while True:
83     bytes_address_pair = UDP_socket.recvfrom(Constants.BUFFER_SIZE)
84     message = bytes_address_pair[0].decode("ascii")
85     https_message = f"HTTP/1.1 200 OK\r\nDate: {formatdate(timeval=None, localtime=
       False, usegmt=True)}\
86     \r\nContent-Type: application/zip\r\nContent-Length: {len(message)}\r\n\r\n{
       client_port}\r\n\r\n{message}"
87     log.info(f"message to xclient: {https_message.encode('ascii')}")
88     xclient_socket.sendall(https_message.encode("ascii"))
```

این تابع وظیفه دارد پیام دریافتی از server را در قالب یک پاسخ HTTP به Xclient ارسال کند. پاسخ HTTP ساخته شده، هدرهای Date، Content-type و Content-Length را دارد، در انتهای پاسخ نیز پیام ارسالی از server قرار دارد. همچنین در پیام بازگشتی port را برای client درست مشخص می‌کند تا امکان بازگرداندن پیام به clinet درست وجود داشته باشد.

۴-۴ تست Xclient و Xserver

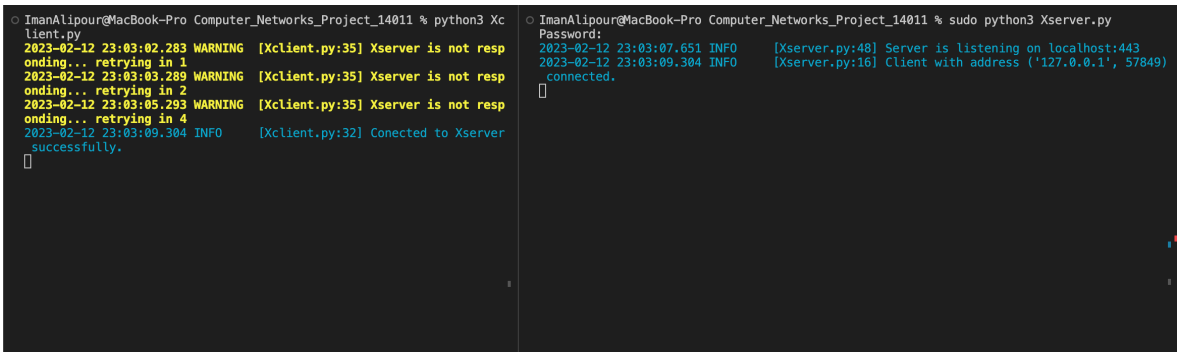
برای نشان دادن صحت اتصال TLS بین Xclient و Xserver دامنه‌ی اتصال را تغییر می‌دهیم و کد را ران می‌کنیم. همانطور که در شکل ۲ مشاهده می‌کنید، با تغییر دامنه‌ی Xserver در فایل Constants.py اتصال برقرار نمی‌شود.

با اصلاح دامنه، امکان اتصال فراهم می‌شود و همانطور که در شکل ۳ مشاهده می‌کنید، Xclinet



شکل ۲: عدم برقراری اتصال به علت درست نبودن دامنه‌ی Xserver.

و Xserver به هم متصل می‌شوند.



شکل ۳: برقراری درست اتصال بین Xserver و Xclient.

۵-۴ ایجاد یک Client و Server ساده

ما برای این پروژه یک Client و Server ساده طراحی کردیم و اسم آن‌ها را EchoClient و EchoServer انتخاب کردیم، همانطور که از اسم آن‌ها مشخص است، EchoClient هر بار یک پیغام ارسال می‌کند و منتظر دریافت پاسخ از Server باقی می‌ماند و سپس می‌تواند پیام دیگری را ارسال کند. همچنین EchoServer پیام‌های دریافتی را به همان نحو باز می‌گرداند.

```
1 import socket
2 import Constants
3
4 server_port = int(input(f"Please enter server port: "))
5 serverAddressPort = (Constants.SERVER_DOMAIN_NAME, server_port)
6 xClientAddressPort = ("127.0.0.1", Constants.XCLIENT_UDP_PORT)
7 bufferSize = 1024
8
```

```
# Create a UDP socket at client side, (ipv4, UDP)
UDPClientSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)

# Initial message
UDPClientSocket.sendto(f"\0\0\0\0{serverAddressPort[0]}\0\0\0\0{serverAddressPort[1]}"
    .encode("ascii"), xClientAddressPort)

while True:
    # Send to server using created UDP socket
    bytesToSend = str.encode(input("Please write your message: "))
    UDPClientSocket.sendto(bytesToSend, xClientAddressPort)

    msgFromServer = UDPClientSocket.recvfrom(bufferSize)[0].decode("ascii")
    print(f"Message from Server: {msgFromServer}")
```

کد بالا، کد مربوط به EchoClient است، در ابتدا یک سوکت UDP می‌سازد و به عنوان ورودی یک رشته می‌گیرد و برای Xclient ارسال می‌کند.

البته فرض ما این بوده که اولین پیام پیام خاصی است و اطلاعات مربوط به مقصد را مشخص می‌کند، همچنین با پیام اول هر دوی Xclient و Xserver اطلاعات مربوط به این اتصال جدید و دورت‌های مبدا و مقصد و دامنه‌ی مقصد و ...را متوجه می‌شوند. راه‌های جایگزین این روش، مشاهده و کپی کردن روش دقیق پروتکل‌های مختلف پروکسی بود اما ما آن را دستی پیاده‌سازی کردیم. همچنین امکان استفاده از سوکت خام و مشاهده‌ی هدرها نیز وجود داشت اما با توجه به پاسخی که در یکی از سوالات آمده بود، شکل ۴ ما در پیام اول به Xclinet اعلام می‌کنیم که پورت و آدرس مقصد چیست و این متادیتا در look up table های Xclinet و Xserver قرار می‌گیرد.

شکل ۴: متن "دقیقا باید در پیام اول به xclient بگید که به چه پورت و ip نهایی قرار است متصل شوید"

حال کد مربوط به Server را توضیح می‌دهیم.

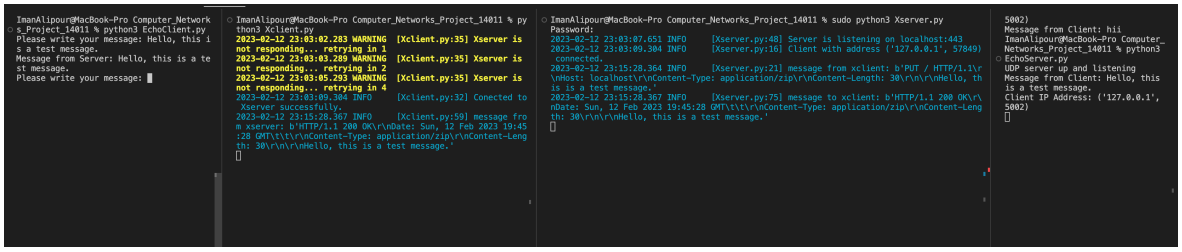
```python
import socket
import Constants


localIP     = "127.0.0.1"

# Create a datagram socket: (ipv4, UDP)
UDPServerSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)

# Bind to address and ip
server_port = int(input(f"Please enter server port: "))
UDPServerSocket.bind((localIP, server_port))

print("UDP server up and listening")

# Listen for incoming datagrams
while(True):
    bytesAddressPair = UDPServerSocket.recvfrom(Constants.BUFFER_SIZE)

    message = bytesAddressPair[0]

    address = bytesAddressPair[1]

```

```
24      print(f"Message from Client: {message.decode('ascii')}")
25      print(f"Client IP Address: {address}")
26
27      # Sending a reply to client
28      UDPServerSocket.sendto(message, address)
```

کد بالا، کد مربوط به EchoServer است، در ابتدا یک سوکت UDP می‌سازد و سپس روی آن گوش می‌ایستد، هر پیامی که دریافت کند را نیز به فرستنده باز می‌گرداند. همچنین پورتی که روی آن کار می‌کند را به عنوان ورودی دریافت می‌کند.

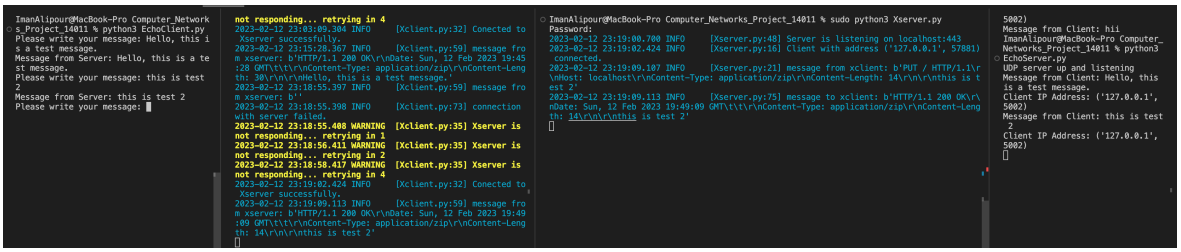
۴-۶ تست اجرای برنامه‌ها

ابتدا یک سناریو مشابه شکل ۱ ایجاد می‌کنیم، در سمت چپ ترمیال مربوط به Client، سپس ترمینال مربوط به Xclient، سپس ترمینال مربوط به Xserver و در نهایت در راست ترمینال مربوط به Server قرار دارد. با استفاده از Client یک پیام ارسال می‌کنیم، نتیجه‌ی این تست را می‌توانید در شکل ۵ مشاهده کنید.



شکل ۵: در این تست از سمت کلاینت، یک پیام به سمت سرور ارسال می‌شود، همانطور که مشاهده می‌کنید، لاگ‌ها درستی کارکرد کد را نشان می‌دهند.

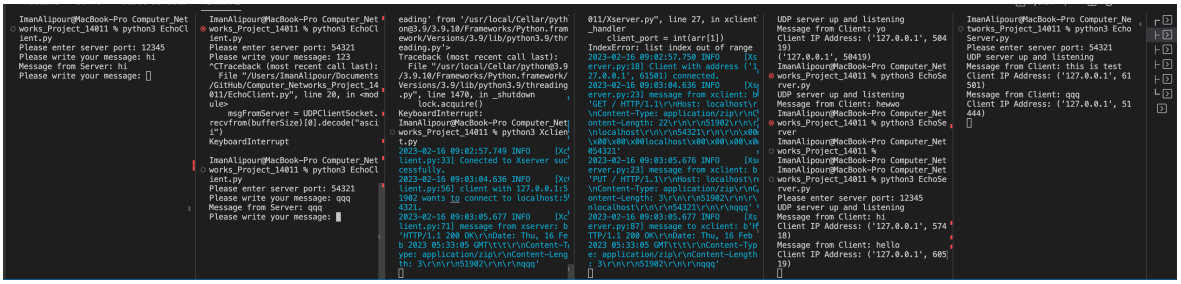
در تستی دیگر برای نشان دادن robustness کدمان، ابتدا Xserver را از کار می‌اندازیم و سپس دوباره روشن می‌کنیم و پیامی دیگر ارسال می‌کنیم، همانطور که از شکل ۶ مشخص است، در این حالت نیز کد ما به درستی عمل می‌کند.



شکل ۶: در این تست پس از قطع و وصل کردن Xserver، از سمت کلاینت یک پیام به سمت سرور ارسال می‌شود، همانطور که مشاهده می‌کنید، لاگ‌ها درستی کارکرد کد را نشان می‌دهند.

در تستی دیگر دو سرور و کلاینت می‌سازیم و از کلاینت، به سرور خاص آن پیامی می‌فرستیم، یکی از این سرورها روی پورت ۱۲۳۴۵ و دیگری روی پورت ۵۴۳۲۱ کار می‌کند، در ابتدا هر دو کلاینت به سرور با پورت ۱۲۳۴۵ متصل هستند و در ادامه یکی از کلاینت‌ها به سرور با پورت ۱۲۳۴۵ متصل می‌شود و دیگری به سرور با پورت ۵۴۳۲۱. نتیجه‌ی این تست را نیز می‌توانید در ادامه مشاهده کنید. شکل ۷، ۸

شکل ۷: اتصال چند کلاینت به یک سرور.



شکل ۸: اتصال چند کلاینت به چند سرور مختلف.

۵ جمع‌بندی و نتایج

در این پروژه، با استفاده از قرار دادن یک واسط میان یک Client و Server، موفق شدیم پیام‌های ارسالی میان آن‌ها را مخفی کنیم و تظاهر به درخواست‌های روزمره‌ی HTTP کنیم. این ساختار را با ورودی‌های مختلف و در سناریوهای مختلف تست کردیم و از عملکرد صحیح آن مطمئن شدیم.

همچنین با ایجاد فایل‌های مورد نیاز برای ایجاد یک ارتباط TLS، ارتباط میان Xserver و Xclient را به صورت امن پیاده‌سازی کردیم.

درنهایت برای مشاهده‌ی ریپوی گیتهاب ما، می‌توانید به [۲] مراجعه کنید.

References

[1] "Implementing tls/ssl in python," https://snyk.io/blog/implementing-tls-ssl-python/. Accessed: 2023-02-07.

[2] "Computer networks project 1401-1," https://github.com/AlipourIm/Computer_Networks_Project_14011. Accessed: 2023-02-16.