BURSA TEKNİK ÜNİVERSİTESİ

Bilgisayar Ağları Dönem Projesi Final Raporu

Üniversite Adı : Bursa Teknik Üniversitesi

Bölüm : Mühendislik ve Doğa Bilimleri Fakültesi Bilgisayar Mühendisliği

Dersin Adı : Bilgisayar Ağları

Proje Başlığı : Advanced Secure File Transfer System

Öğrenci Adı Soyadı : Hasan Umut Kocatepe

Öğrenci Numarası : 21360859077

Ders Öğretim Üyesi : Doç. Dr. İzzet Fatih Şentürk

Ders Araştırma Görevlisi : Arş. Gör. Şeyma Doğru

Teslim Tarihi : 10.06.2025

İÇİNDEKİLER

1. GİRİŞ

Bu proje, **Secure File Transfer System** (Güvenli Dosya Transfer Sistemi) adı altında geliştirilmiştir. Amacı, ağ ortamında şifreli ve güvenli dosya transferlerini mümkün kılmak, saldırı ve sniffing gibi güvenlik tehditlerine karşı önlem alabilmektir.

1.1. Proje Amacı

Projenin temel amacı, dosya transferlerinde güvenliği artırmak ve **AES-256 şifreleme** gibi güçlü algoritmalarla veri bütünlüğünü korumaktır. Ayrıca, UDP ve TCP gibi farklı iletişim protokollerini kullanarak esnek ve güvenilir bir aktarım ortamı sağlamak hedeflenmiştir.

1.2. Projenin Önemi ve Kullanım Alanı

Ağ üzerinden dosya aktarımı, birçok sistemde hayati bir işlemdir. Bu projeyle, **sniffing ve IP spoofing gibi saldırı türleri** test edilerek sistemin ne kadar güvenli olduğu gözlemlenmiştir. Gerçek dünya uygulamaları; şirket içi veri paylaşımı, IoT cihazlarının güncellenmesi ve hassas verilerin korunması gibi alanlarda geniş yer bulur.

2. TEKNİK DETAYLAR

Bu bölümde, projenin teknik yapısı, uygulama bileşenleri ve temel algoritmalar anlatılmaktadır.

2.1 Proje Yapısı ve Dosyalar

Proje, **4 ana modülden** oluşmaktadır:

- **client.ipynb**: İstemci tarafı, dosya gönderimi ve şifreleme kodlarını içerir.
- **server.ipynb**: Sunucu tarafı, gelen dosyaların çözülmesi ve loglanması kodlarını

içerir.

- **config.py**: Ortak yapılandırma dosyasıdır. AES anahtarı, IP adresleri ve port numaraları gibi parametreler burada tanımlanmıştır.
- **ip-spoofing-simulation.ipynb**: IP spoofing saldırı simülasyonu için deneysel kodları içerir.

Bu modüllerle birlikte **attack_log.txt** gibi çıktı dosyaları da proje dizininde oluşturulur.

2.2 AES Encyrption & Decyrption Uygulaması

Projenin en önemli güvenlik bileşeni, **AES-256** şifreleme algoritmasıdır.

- **AES_KEY**: config.py dosyasında **32 byte** uzunluğunda tanımlanmıştır.
- Dosya, 1 KB'lik fragmentlere bölünerek şifrelenir.
- client.ipynb tarafında dosya gönderimi öncesinde **AES şifreleme** yapılır.
- server.ipynb tarafında ise dosya alındığında **decryption** (çözme) işlemi yapılır.

Bu sayede dosya bütünlüğü korunur ve iletim sırasında veri şifreli olarak kalır.

- **Server tarafı decrypt_file() :**

```python
# AES-256 şifreli dosya çözme fonksiyonu
def decrypt_file(input_file, output_file):
    with open(input_file, 'rb') as f:
        nonce, tag, ciphertext = [f.read(x) for x in (16, 16, -1)]
    cipher = AES.new(AES_KEY, AES.MODE_EAX, nonce=nonce)
    plaintext = cipher.decrypt_and_verify(ciphertext, tag)
    with open(output_file, 'wb') as f:
        f.write(plaintext)
```

- **Client tarafı encrypt_file() :**

```python
# AES-256 ile dosyayı şifreler ve nonce, tag, ciphertext şeklinde kaydeder
def encrypt_file(input_file, output_file):
    cipher = AES.new(AES_KEY, AES.MODE_EAX)
    with open(input_file, 'rb') as f:
        plaintext = f.read()
    ciphertext, tag = cipher.encrypt_and_digest(plaintext)
    with open(output_file, 'wb') as f:
        f.write(cipher.nonce)
        f.write(tag)
        f.write(ciphertext)
```

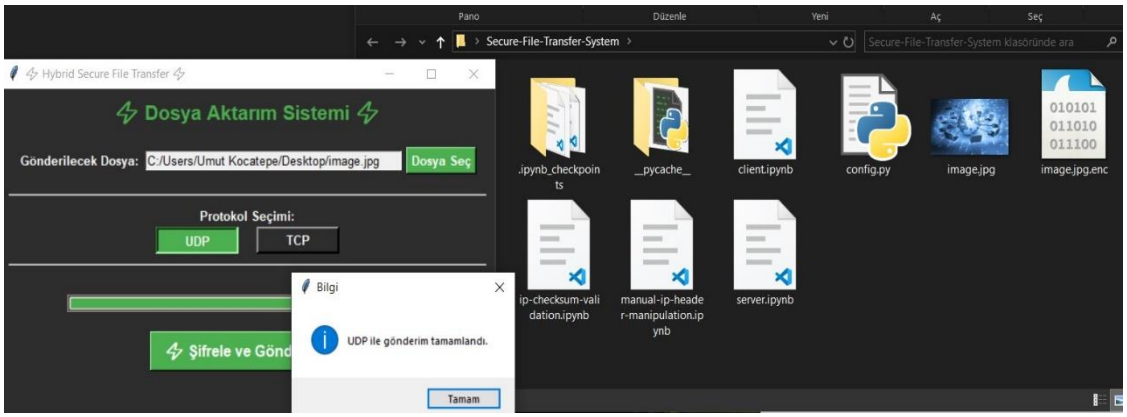
2.3. GUI Kullanımı ve Dosya Gönderimi

Proje, kullanıcı dostu bir arayüz sunar.

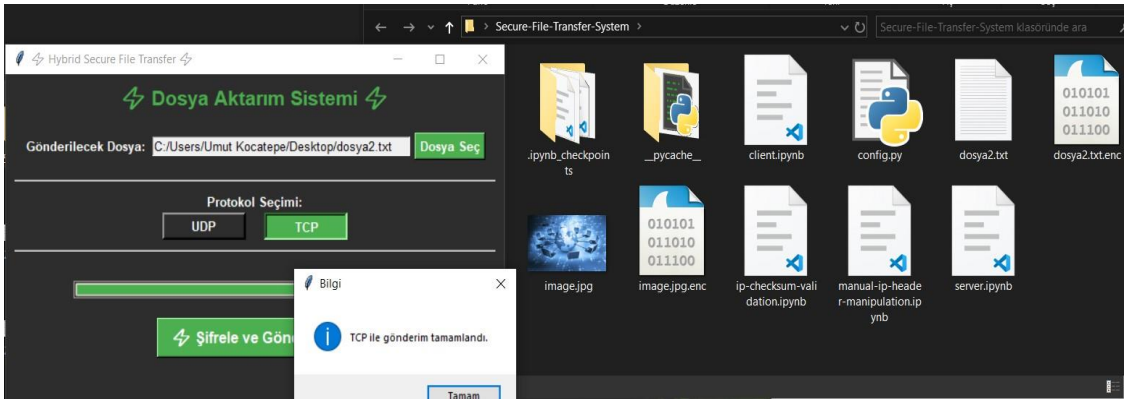
- Kullanıcı, göndermek istediği dosyayı seçer ve hangi protokolü kullanacağını belirler (**UDP** veya **TCP**).
- GUI, Python notebook çıktıları ve **Tkinter** veya ipywidgets gibi araçlarla desteklenmiştir.
- Kullanıcı gönderim sürecini görsel olarak takip edebilir.

GUI Görselleri :

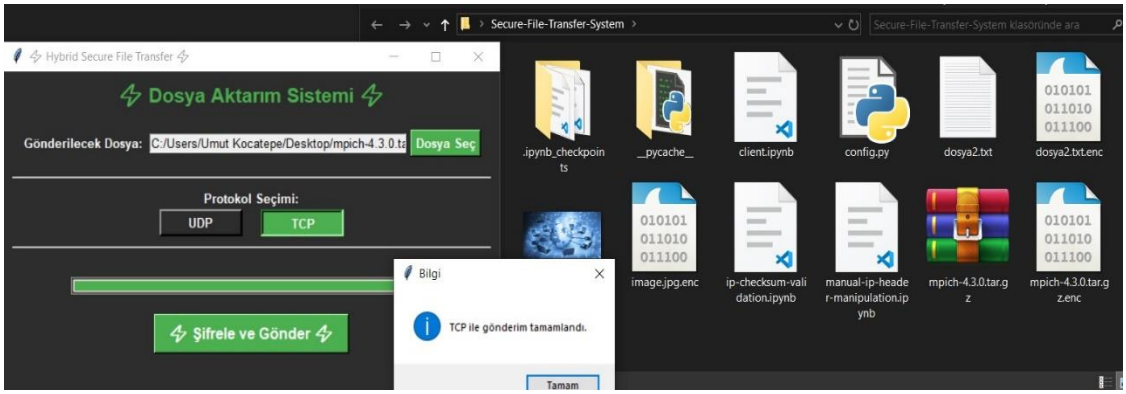
- **UDP gönderimi:**



- **TCP gönderimi (örnek .txt dosyası):**



- **TCP gönderimi (örnek .mpich dosyası):**



2.4. Server Tarafı Bağlantı ve Dinleme İşlemleri

server.ipynb, hem **UDP hem TCP** bağlantılarını dinleyerek dosya alımını sağlar:

- **UDP Dinleme ve Dosya Alma Mantığı** :

```
# UDP veri transferi için sunucu oluşturuluyor
def handle_udp():
    udp_server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    udp_server.bind(("", 5555))
    print("UDP sunucusu 5555 portunda dinleniyor...")

    while True:
        file_name, addr = udp_server.recvfrom(4096)
        file_name = file_name.decode()
        fragments = []

        while True:
            data, _ = udp_server.recvfrom(4096)
            if data == b'EOF':
                break
            fragments.append(data)

        with open(file_name, 'wb') as f:
            for frag in fragments:
                f.write(frag)

        decrypt_file(file_name, file_name.replace('.enc', ''))
        print(f"UDP dosyası çözüldü: {file_name.replace('.enc', '')}")
```

  - socket.socket(AF_INET, SOCK_DGRAM) kullanılır.
  - UDP portu (5555) üzerinde veriler dinlenir ve gelen parçalar **AES**

decryption ile çözülür.

- **TCP Dinleme ve Dosya Alma Mantığı** :

```
# TCP veri transferi için sunucu oluşturuluyor
def handle_tcp():
    tcp_server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    tcp_server.bind(("", 5556))
    tcp_server.listen(5)
    print("TCP sunucusu 5556 portunda dinleniyor...")

    while True:
        conn, addr = tcp_server.accept()
        print(f"TCP bağlantı: {addr}")

        file_name = conn.recv(1024).decode()
        with open(file_name, 'wb') as f:
            while True:
                data = conn.recv(1024)
                if not data:
                    break
                f.write(data)

        decrypt_file(file_name, file_name.replace('.enc', ''))
        print(f"TCP dosyası çözüldü: {file_name.replace('.enc', '')}")
        conn.close()
```

  - socket.socket(AF_INET, SOCK_STREAM) kullanılır.
  - Sunucu, gelen bağlantıyı accept() ile kabul eder ve veri transferini başlatır.
  - TCP portu (5556) üzerinde de bağlantılar dinlenir ve alınan parçalar AES decryption ile çözülür.

- **Sniffer Fonksiyonu Dinleme ve Log Kaydetme :**

```python
# Sniffer anomaly detection (tekrar eden UDP flood kontrolü)
recent_packets = deque(maxlen=200) # Flood kontrolü için artan buffer boyutu

# Sniffer fonksiyonu: Ağdaki UDP flood saldırılarını ve anormal trafiği algılar, 100 tekrar sonrası IP'yi kara listeye ekler
def sniff_packets():
    def packet_callback(packet):
        if packet.haslayer(IP):
            ip_layer = packet[IP]
            src_ip = ip_layer.src
            dst_ip = ip_layer.dst

            # Bloklanan IP'yi kontrol et: loglamayı veya işlemeyi durdur
            if src_ip in blocked_ips:
                return  # Bu IP engellendiği için artık işlem yapmıyoruz!

            length = len(packet)
            print(f"[Sniffer] {src_ip} -> {dst_ip} | {length} byte")

            if packet.haslayer(UDP):
                payload = bytes(packet[Raw]) if packet.haslayer(Raw) else b""
                payload_hash = hashlib.md5(payload).hexdigest()

                recent_packets.append((src_ip, length, payload_hash))
                repeat_count = sum(1 for p in recent_packets if p == (src_ip, length, payload_hash))

                # Flood için aynı paket tekrarını kontrol et 100 tekrar varsa IP'yi kara listeye al
                if repeat_count >= 100:
                    warning = f"! [Sniffer] {src_ip} -> {dst_ip} - Tehlikeli Flood Detected: {repeat_count} kez tekrar! IP bloklandı."
                    print(warning)
                    with open("attack_log.txt", "a") as log:
                        log.write(warning + "\n")
                    blocked_ips.add(src_ip)  # 100'den fazlaysa IP'yi kara listeye ekle
                    return

                if length > 4096:
                    warning = f"! [Sniffer] {src_ip} -> {dst_ip} - Çok büyük UDP paketi ({length} byte)!"
                    print(warning)
                    with open("attack_log.txt", "a") as log:
                        log.write(warning + "\n")
                elif length < 3:
                    warning = f"! [Sniffer] {src_ip} -> {dst_ip} - Çok küçük UDP paketi ({length} byte)!"
                    print(warning)
                    with open("attack_log.txt", "a") as log:
                        log.write(warning + "\n")

    print("[Sniffer] Raw packet dinleme başladı...")
    sniff(prn=packet_callback, store=0)
```

Gelen veriler **attack_log.txt** gibi dosyalara loglanır ve AES decryption ile çözülür.

- **Server tarafı konsol çıktısı :**

```
UDP sunucusu 5555 portunda dinleniyor...
TCP sunucusu 5556 portunda dinleniyor...
UDP dosya adı: image.jpg.enc
UDP dosyası alındı: image.jpg.enc
UDP dosyası çözüldü: image.jpg
TCP bağlantı: ('127.0.0.1', 60711)
TCP dosyası alındı: dosya2.txt.enc
TCP dosyası çözüldü: dosya2.txt
TCP bağlantı: ('127.0.0.1', 60712)
TCP dosyası alındı: mpich-4.3.0.tar.gz.enc
TCP dosyası çözüldü: mpich-4.3.0.tar.gz
```

2.5. Client Tarafı Dosya Gönderimi

client.ipynb, kullanıcının seçtiği dosyayı **AES şifreleme** sonrası UDP veya TCP üzerinden göndermek için yapılandırılmıştır:

- Dosya, **1 KB fragment**'lere ayrılır.
- Her fragment encrypt_file fonksiyonunda şifrelenir ve socket bağlantısı üzerinden gönderilir.
- UDP için: Bağlantı kurulmadan doğrudan gönderim yapılır.
- TCP için: Bağlantı connect() komutu ile kurulur ve ardından fragmentler sırayla yollanır.

- **Client Tarafı UDP Gönderim Fonksiyonu (Kısa Versiyon – Progress Bar'sız) :**

```
# UDP protokolüyle dosyayı parçalara ayırarak gönderir
def send_udp(encrypted_filename, progress_bar):
    client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    file_name = os.path.basename(encrypted_filename).encode()
    client.sendto(file_name, (SERVER_IP, UDP_PORT))

    with open(encrypted_filename, 'rb') as f:
        while True:
            fragment = f.read(FRAGMENT_SIZE)
            if not fragment:
                break
            client.sendto(fragment, (SERVER_IP, UDP_PORT))
            # Progress bar güncelleme ve delay
            time.sleep(0.001)
    client.sendto(b'EOF', (SERVER_IP, UDP_PORT))
    client.close()
```

- **Client Tarafı TCP Gönderim Fonksiyonu (Tam Versiyon) :**

```python
# TCP protokolüyle dosyayı gönderme
def send_tcp(encrypted_filename, progress_bar):
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # Bağlantı kur ve ilk paket olarak dosya adını gönder
    client.connect((SERVER_IP, TCP_PORT))
    client.send(os.path.basename(encrypted_filename).encode())

    filesize = os.path.getsize(encrypted_filename)
    sent_bytes = 0

    # Dosya parçalarını tek tek gönder (TCP akışı)
    with open(encrypted_filename, 'rb') as f:
        progress_console = tqdm(total=filesize, unit='B', unit_scale=True, desc="Gönderiliyor") # outputtaki progress_bar
        while True:
            chunk = f.read(1024)
            if not chunk:
                break
            client.sendall(chunk)
            sent_bytes += len(chunk)
            progress_gui = int((sent_bytes / filesize) * 100) #gui'deki progress bar
            progress_bar['value'] = progress_gui
            progress_bar.update()
            progress_console.update(len(chunk)) # console'daki progress bar
        progress_console.close()

    client.close()
    messagebox.showinfo("Bilgi", "TCP ile gönderim tamamlandı.")
    print("TCP ile gönderim tamamlandı.")
```

- **Client Tarafı Dosya Seçimi ve Gönderimi Fonksiyonu (GUI'de Opsiyonel UDP/TCP Seçimi) :**

```python
# Kullanıcıdan alınan dosyayı şifrele ve protokole göre gönder
def start_transfer(entry, protocol_var, progress_bar):
    filename = entry.get()
    if not filename:
        messagebox.showwarning("Uyarı", "Lütfen bir dosya seçin.")
        return
    encrypted_filename = filename + '.enc'
    encrypt_file(filename, encrypted_filename)

    protocol = protocol_var.get()
    if protocol == "UDP":
        send_udp(encrypted_filename, progress_bar)
    else:
        send_tcp(encrypted_filename, progress_bar)
```

- **Client Tarafı Konsol Çıktısı :**

```
Gönderiliyor: 100%|████████████████████████████████████████████████| 151k/151k [00:01<00:00, 111kB/s]
UDP ile gönderim tamamlandı.
Gönderiliyor: 100%|████████████████████████████████████████████████| 53.0/53.0 [00:00<00:00, 17.7kB/s]
TCP ile gönderim tamamlandı.
Gönderiliyor: 100%|████████████████████████████████████████████████| 37.5M/37.5M [00:40<00:00, 933kB/s]
TCP ile gönderim tamamlandı.
```

2.6. IP Spoofing Saldırı Simülasyonu

Proje kapsamında **ip-spoof-simulation.ipynb** dosyası kullanılarak bir **UDP flood spoofing saldırısı** simüle edilmiştir:

- **IP ve UDP başlıkları** elle ayarlanır ve sahte (spoofed) IP adresiyle paket gönderilir.
- Bu saldırı için config.py dosyasında sunucu IP'si **192.168.1.10** olarak güncellenir.
- Bu saldırı için server.ipynb dosyası ana makinede çalışırken , **ip-spoof-simulation.ipynb** dosyası , **farklı bir cihazdan veya sanal makineden** çalıştırılmalıdır.
- Saldırı 1000 kez tekrar edilerek sunucunun anomaliyi algılaması sağlanır. 100. tekrarda zaten anomali olarak algılanarak IP'si bloklanır.
- **IP Spoofed UDP Flood Saldırı Kod Görseli (ip-spoof-simulation.ipynb) :**

```python
# ip-spoof-simulation.py #Bu spoofed UDP saldırı testi farklı bir cihazdan veya sanal makineden yapılmal
from scapy.all import IP, UDP, send

# Sahte (spoofed) IP adresi
source_ip = "192.168.1.100"  # sahte IP
target_ip = "192.168.1.10"   # hedef IP (Bu saldırı simulasyonu için local değil gerçek bir target-ip gerekir)

# IP header elle ayarla
ip_layer = IP(src=source_ip, dst=target_ip)

# UDP header ayarla
udp_layer = UDP(sport=12345, dport=5555)

# Veri kısmı
data = "Spoofed UDP Flood Attack".encode()

# Paket oluştur
packet = ip_layer / udp_layer / data

# Paketi çok sayıda gönder (ör. 1000 kere)
for i in range(1000):
    send(packet, verbose=0)  # verbose=0 → her gönderimde mesaj yazmasın

print("Spoofed UDP paketleri gönderildi.")
```

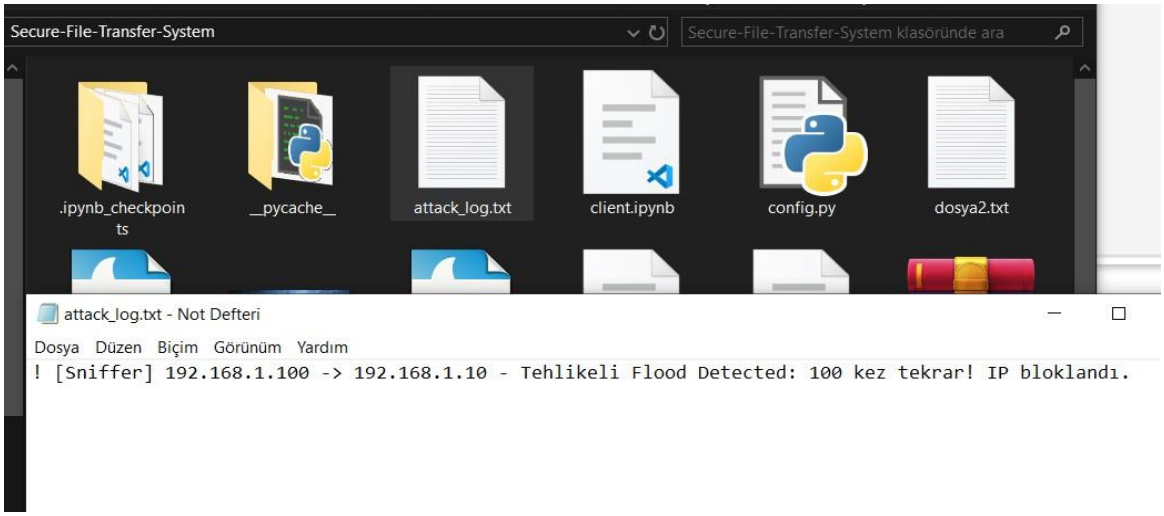
Saldırı Simulasyon Akışı :

- **1-Spoofed Paket Gönderimi (Linux Sanal Makineden) :**

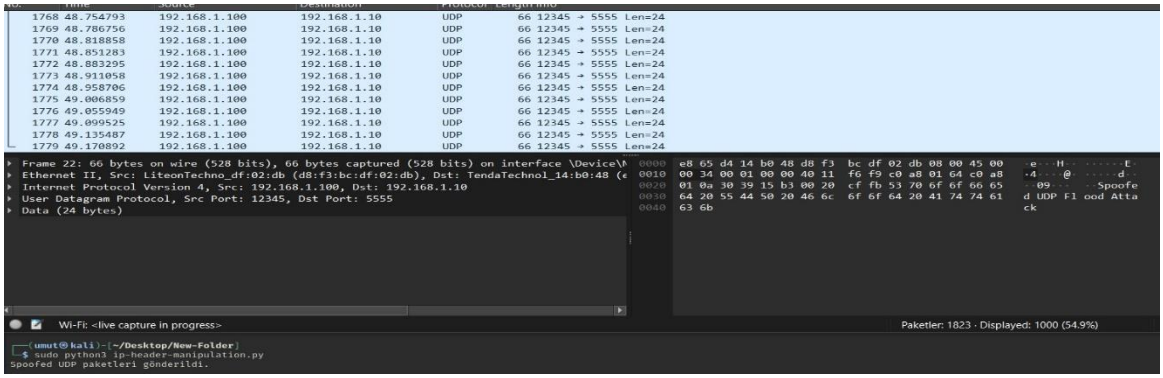
- **2-Server Tarafı (Ana Makine) Konsol Çıktısı ve Bloklanan IP'nin kanıtı :**

```
[Sniffer] 192.168.1.100 -> 192.168.1.10 | 66 byte
[Sniffer] 192.168.1.100 -> 192.168.1.10 | 66 byte
[Sniffer] 192.168.1.100 -> 192.168.1.10 | 66 byte
[Sniffer] 192.168.1.100 -> 192.168.1.10 | 66 byte
[Sniffer] 192.168.1.100 -> 192.168.1.10 | 66 byte
[Sniffer] 192.168.1.100 -> 192.168.1.10 | 66 byte
! [Sniffer] 192.168.1.100 -> 192.168.1.10 - Tehlikeli Flood Detected: 100 kez tekrar! IP bloklandı.
[Sniffer] 216.58.213.106 -> 192.168.0.102 | 140 byte
[Sniffer] 192.168.0.102 -> 216.58.213.106 | 54 byte
```

- **3-attack-log.txt dosyası proje dizininde kaydedilir :**



- **4-Wireshark'tan da saldırıyı analiz edebiliriz :**



2.7. Loglama ve Anomali Tespiti

Sunucu tarafında, saldırı ve anormal trafik aktiviteleri **attack_log.txt** dosyasına kaydedilerek loglanır:

- **Normal veri transferi**: 3 Bayt ile 4096 bayt arasındaki düzenli, şifreli ve TCP/UDP üzerinden aktarılan veri parçaları.
- **Anormal trafik**: 1000 paketlik UDP flood saldırısı gibi 100 tekrarın üzerinde olan durumlar **anormal** olarak kaydedilir.
- Server tarafındaki konsolda da anormallikler ekrana yansır ve loglar kaydedilecektir.

Bu loglama mekanizması, sistemin saldırılara karşı **temel anomali tespiti** özelliğini kanıtlar ve gerçek zamanlı savunma senaryoları için temel oluşturur.

3. SINIRLAMALAR VE GELİŞTİRME ÖNERİLERİ

Bu bölümde projenin mevcut sınırlandırmaları ve gelecekte yapılabilecek iyileştirme önerileri sunulmaktadır.

3.1. Sınırlamalar

- **Yerel test ortamı**: Proje, yalnızca yerel (localhost veya aynı ağ segmentindeki sanal makineler) test ortamında uygulanmıştır. Gerçek dünya ağ ortamlarında daha geniş bir saldırı vektörü ve farklı senaryolarla test edilmemiştir.
- **Saldırı tespiti**: Şu anki sistem sadece basit bir eşik değeri ile saldırı tespiti yapabilmektedir (örn. 100 paket üzeri UDP flood algılama). Daha karmaşık saldırı türleri veya DDoS gibi geniş ölçekli ataklar için kapsamlı analiz eksiktir.
- **Kullanıcı dostu arayüz**: GUI tarafı basit tutulmuştur. Kullanıcı deneyimini daha fazla zenginleştirecek özellikler (örneğin: dosya transfer ilerleme çubuğu, saldırı bildirim pop-up'ı) eklenmemiştir.

3.2. Gelecek Geliştirme Önerileri

- **IDS/IPS entegrasyonu**: Sunucu tarafında bir **Intrusion Detection System (IDS)** veya **Intrusion Prevention System (IPS)** modülü eklenerek daha karmaşık saldırı türlerinin (örneğin: TCP SYN flood, slowloris gibi) algılanması mümkün kılınabilir.
- **Gerçek zamanlı log analizi**: attack_log.txt gibi log dosyaları, bir web paneli veya gerçek zamanlı analiz aracıyla anlık olarak incelenebilir. Böylece saldırılar daha hızlı yanıtlanabilir.
- **Dosya bütünlüğü kontrolü**: Dosya transferi sırasında **SHA256 veya MD5 gibi hash algoritmaları** kullanılarak dosya bütünlüğü doğrulanabilir. Böylece şifreleme dışında **veri bozulmasına** karşı ek bir koruma katmanı sağlanır.

- **Gelişmiş kullanıcı arayüzü**: Proje arayüzü, kullanıcı deneyimini artırmak için modern UI/UX prensiplerine göre yeniden tasarlanabilir (örneğin: web tabanlı frontend veya masaüstü uygulaması).

4. SONUÇ

Bu proje kapsamında, ağ ortamında güvenli dosya transferlerini sağlamak ve olası saldırılara karşı koruma sağlamak amacıyla **Secure File Transfer System** başarıyla geliştirilmiştir. AES-256 şifreleme algoritması kullanılarak dosya bütünlüğü korunmuş ve hem **UDP hem de TCP protokolleri** üzerinden güvenli veri transferi gerçekleştirilmiştir.

Ayrıca, IP spoofing ve UDP flood gibi saldırı simülasyonları yapılmış ve sistemin anomali tespiti yetenekleri test edilmiştir. **attack_log.txt** gibi loglama mekanizmaları, saldırıların izlenebilirliğini sağlayarak projenin temel hedeflerini desteklemiştir.

Sonuç olarak, proje **hedeflerine ulaşmış ve kullanıcı tarafında şifreli dosya transferi sağlanarak** başarıyla tamamlanmıştır. Bu çalışma, ilerleyen süreçte gerçek zamanlı saldırı tespiti (IDS/IPS) veya gelişmiş kullanıcı deneyimi odaklı iyileştirmelerle daha da ileri taşınabilir.

5. KAYNAKLAR

- Daemen, J., & Rijmen, V. (2002). *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer.
- Python Software Foundation. (n.d.). *Socket Programming HOWTO*. Retrieved from https://docs.python.org/3/howto/sockets.html
- Scapy Project Documentation. (n.d.). Retrieved from https://scapy.readthedocs.io/en/latest/introduction.html
- Comer, D. E. (2018). *Computer Networks and Internets* (6th ed.). Pearson.
- Sanders, C. (2017). *Practical Packet Analysis: Using Wireshark to Solve Real-World Network Problems* (3rd ed.). No Starch Press.
- Proje Tanıtım Video: https://youtu.be/0f_NkozS7w0
- Proje Kaynak Kodu Bağlantısı : https://github.com/UmutKocatepe/Secure-File-Transfer-System