



**BURSA TEKNİK  
ÜNİVERSİTESİ**

**Bilgisayar Ağları  
Dönem Projesi Final Raporu**

**Mustafa AYKUT**

**22360859028**

# INTRODUCTION

Bu proje, cihazlar arasında güvenli şekilde veri iletimini sağlayan bir dosya gönderim sistemidir. Soket bağlantısı ile cihazlar arasında TCP veya UDP protokollerini kullanılarak paketler gönderilebilir. Scapy kütüphanesi ile ağ paketlerinin detaylı kontrolü sayesinde paketlerin başlık bilgileri istenilen şekilde değiştirilebilir. Crypto kütüphanesi sayesinde veriler AES ve RSA metodları ile şifrelenir. SHA256 gibi algoritmalar ile de parolalar şifrelerek gönderilir.

Proje aşağıdaki dosya düzeneğine sahiptir:

- /keys
  - private.pem
  - public.pem
- /socket
  - /iperf3
    - iperf3.exe
  - client.py
  - server.py
  - generator.py
  - network\_analysis.py
  - mitm\_proxy.py
  - packet\_injection.py
  - secure\_transfer\_gui.py
- /scapy
  - receiver\_scapy.py
  - sender\_scapy.py

Raporun ilerleyen kısımlarında sırasıyla bu dosyalar ve içerikleri sırayla inceleneciktir. Proje kullanılırken secure\_transfer\_gui.py dosyasının çalıştırılması önerilir. GUI sayesinde programın kullanımı oldukça kolay ve anlaşılır hale gelmektedir. Arayüzde bir dosya seçiliğinde socket klasörünün altında received\_files adında bir klasör oluşturulmaktadır. Gönderilen dosyalar bu klasörün içinde gönderilme tarihi ile isimlendirilmiş şekilde bulunabilir.

## TECHNICAL DETAILS

Bu bölümde projenin genel işleyişi, bu işleyişteki fonksiyonlar ve kodlar detaylı olarak açıklanacaktır. Ayrıca projede kullanılan python kütüphanelerine de değinilecektir.

### 1. Kullanılan Kütüphaneler

Standart kütüphaneler:

- **socket** - Ağ bağlantıları için
- **threading** - Çoklu thread işlemleri için
- **time** - Zaman ölçümleri ve bekleme işlemleri için
- **sys** - Sistem parametreleri ve çıkış kodları için
- **os** - Dosya ve klasör işlemleri için
- **datetime** - Tarih ve zaman işlemleri için
- **pathlib (Path)** - Dosya yolu işlemleri için
- **zlib** - CRC32 checksum hesaplama için
- **hashlib** - SHA256 hash hesaplama için
- **statistics** - İstatistiksel hesaplamalar için
- **platform** - İşletim sistemi bilgileri için
- **subprocess** - Dış komutları çalıştırmak için
- **re** - Düzenli ifadeler (regex) için
- **signal** - Sinyal yakalama için
- **shutil** - Dosya ve sistem araç kontrolü için

GUI Kütüphaneleri:

- **tkinter** - Ana GUI framework'ü
  - **ttk** - Gelişmiş GUI bileşenleri
  - **filedialog** - Dosya seçme dialogları
  - **messagebox** - Mesaj kutuları

- **scrolledtext** - Kaydırılabilir metin alanları

### Kriptografi Kütüphaneleri (PyCrypto/PyCryptodome)

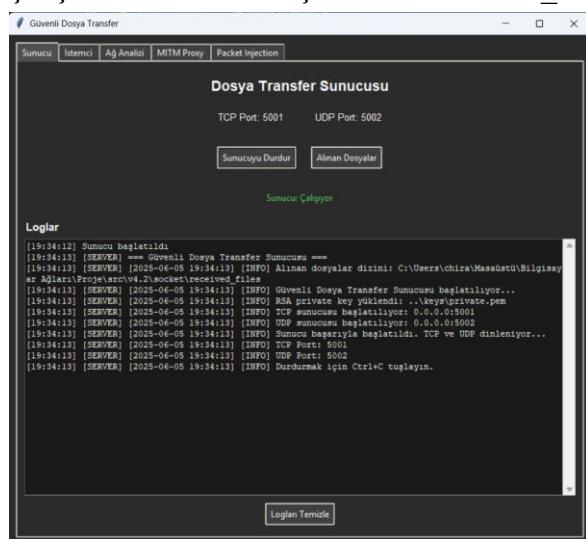
- **Crypto.PublicKey.RSA** - RSA anahtar işlemleri
- **Crypto.Cipher.AES** - AES şifreleme
- **Crypto.Cipher.PKCS1\_OAEP** - RSA şifreleme
- **Crypto.Util.Padding** - AES padding işlemleri
- **Crypto.Random** - Güvenli rastgele sayı üretimi

### Ağ Analizi Kütüphaneleri

- **scapy.all** - Ağ paketleri oluşturma ve analizi için
  - **IP** - IP paketleri için
  - **UDP** - UDP paketleri için
  - **Raw** - Ham veri paketleri için
  - **fragment** - Paket parçalama için
  - **send** - Paket gönderme için
  - **sniff** - Paket yakalama için

## 2. GUI (Grafiksel Kullanıcı Arayüzü) Kullanımı

Projeyi python dosyalarıyla uğraşmadan kolayca kullanabilmenin en iyi yolu GUI programını çalıştırmaktır. Bunun için “/socket/secure\_transfer\_gui.py” dosyasını python yorumlayıcısı ile



çalıştırmamız yeterlidir. Program çalıştırıktan sonra karşımıza aşağıdaki ekran gelecektir:

Bu programda seküler arasında gezip istenilen işlem yapılmaktadır. Anlık olarak kayıtlar da programın aşağıdaki ekranda gözükmektedir.

### 3. “/keys” Klasörü

Bu klasör, socket klasöründeki dosyaların güvenli veri iletimi gerçekleştirebilmesi için gerekli olan şifreleme anahtarlarını içermektedir. Sistemde hibrit şifreleme yaklaşımı kullanılmakta olup, AES anahtarı RSA anahtarı ile şifrelenmektedir.

Şifrelerin çözümü için public ve private anahtar çiftine ihtiyaç duyulmaktadır. Bu anahtar çiftinin henüz mevcut olmaması durumunda, socket klasörü altında bulunan generator.py dosyası çalıştırılarak gerekli anahtarlar oluşturulabilir.

- **generator.py**

Bu python dosyası iki ana kısımdan oluşmaktadır: main() fonksiyonu programın başlangıç noktasıdır. Diğer fonksiyonu çağırır. create\_rsa\_key\_pair(...) fonksiyonu da girilen anahtar boyutunda iki tane anahtar oluşturur ve ilgili klasöre kaydeder. Public key herkesle paylaşılabilir ve veri şifrelemek için kullanılır. Private key ise sadece yetkili kişilerce bilinmelidir. Şifrelenmiş veriyi çözmek için kullanılır.

```
1 import os # dosya ve klasör işlemleri için
2 from Crypto.PublicKey import RSA # RSA anahtar oluşturma için
3
4 # =====
5 # RSA ANAHTAR ÇİFTİ OLUŞTURMA
6 # =====
7 def create_rsa_key_pair(key_size=2048, output_dir="keys"):
8     # Çıktı Klasörünün varlığını kontrol et ve gerekirse oluştur
9     if not os.path.exists(output_dir):
10         os.makedirs(output_dir) # klasör oluştur (alt klasörler dahil)
11         print(f"'{output_dir}' klasörü oluşturuldu.")
12
13     try:
14         # RSA anahtar çifti oluşturma işlemini başlat
15         print(f"[key_size] bit RSA anahtar çifti oluşturuluyor...")
16         key = RSA.generate(key_size) # belirtilen uzunlukta RSA anahtarı oluştur
17
18         # Anahtarların kaydedileceği dosya yollarını belirle
19         private_key_path = os.path.join(output_dir, "private.pem") # özel anahtar dosya yolu
20         public_key_path = os.path.join(output_dir, "public.pem") # genel anahtar dosya yolu
21
22         # Özel anahtarı PEM formatında dosyaya kaydet
23         with open(private_key_path, "wb") as private_file:
24             private_file.write(key.export_key('PEM')) # PEM formatında dışa aktar ve yaz
25
26         # Genel anahtarı PEM formatında dosyaya kaydet
27         with open(public_key_path, "wb") as public_file:
28             public_file.write(key.publickey().export_key('PEM')) # genel anahtarı al, PEM formatında dışa aktar
29
30         # Başarılı oluşturma mesajını göster
31         print(f"RSA anahtar çifti başarıyla oluşturuldu:")
32         print(f" - Özel anahtar: {private_key_path}")
33         print(f" - Genel anahtar: {public_key_path}")
34
35         # Dosya yollarını tuple olarak döndür
36         return private_key_path, public_key_path
37
38     except Exception as e:
39         # Hata durumunda hata mesajını göster ve None değerleri döndür
40         print(f"!Hata oluştu: {e}")
41         return None, None
42
43 # =====
44 # ANA FONKSİYON
45 # =====
46 def main():
47     # Varsayılan ayarlarla anahtar çiftini oluştur
48     private_path, public_path = create_rsa_key_pair()
49
50     # Anahtar oluşturma başarılı oldusaya güvenlik uyarılarını göster
51     if private_path and public_path:
52         print("\nGÜVENLİK UYARIŞI:")
53         print("- Özel anahtarınızı (private.pem) güvenli bir yerde saklayın")
54         print("- Özel anahtarınızı asla paylaşmayın")
55         print("- Genel anahtarları (public.pem) güvenle paylaşabilirsiniz")
56
57 # =====
58 # PROGRAMIN BAŞLANGIÇ NOKTASI
59 # =====
60 if __name__ == "__main__":
61     main()
```

## 4. “/socket” Klasörü

Bu klasör içerisindeki kodlar socket bağlantıları yapar ve server.py dosyasıyla TCP/UDP bağlantıları kabul eder, client.py dosyasıyla sunucuya bağlanıp dosya gönderir. Geri kalan python dosyaları ile de gerekli ağ analizlerini ve saldırısı simülasyonlarını gerçekleştirir. Sırasıyla bu kodlar şu şekilde çalışmaktadır:

- **server.py**

Sunucu kodu oldukça uzun ve karmaşık olduğu için bölümler halinde anlatılacaktır.

Sunucu başlatılırken kullanılacak ayarlar aşağıdaki DEFAULT\_CONFIG değişkeninde saklanmaktadır.

```
1 # =====
2 # VARSAYILAN AYARLAR
3 # =====
4 DEFAULT_CONFIG = {
5     'HOST': '0.0.0.0',                      # tüm ağ arayüzlerini dinle
6     'TCP_PORT': 5001,                        # TCP portu
7     'UDP_PORT': 5002,                        # UDP portu
8     'PASSWORD': b"gizli_sifre",              # kimlik doğrulama parolası (ürütim ortamında güçlü parola kullanın)
9     'PRIVATE_KEY_PATH': "../keys/private.pem", # RSA private key dosya yolu
10    'RECEIVED_FILES_DIR': "./received_files", # alınan dosyaların kaydedileceği dizin
11    'SESSION_TIMEOUT': 30.0,                  # UDP oturum timeout (saniye)
12    'MAX_PACKET_SIZE': 2048,                 # maksimum paket boyutu
13    'MAX_CONNECTIONS': 10,                   # maksimum eşzamanlı bağlantı sayısı
14    'CLEANUP_INTERVAL': 5.0,                 # temizleme döngüsü (saniye)
15    'LOG_LEVEL': 'INFO'                    # log seviyesi (DEBUG, INFO, WARNING, ERROR)
16 }
```

**main()** fonksiyonu da varsayılan ayarları kullanarak SecureFileTransferServer sınıfından bir sunucu oluşturur ve server.start() fonksiyonu ile sunucuyu başlatır.

```
1 # =====
2 # ANA PROGRAM
3 # =====
4 def main():
5     # Program başlık mesajı
6     print("== Güvenli Dosya Transfer Sunucusu ==")
7
8     # Konfigürasyonu özelleştir (varsayılan değerleri kopyala)
9     config = DEFAULT_CONFIG.copy()
10
11    # Komut satırı argümanlarını kontrol et
12    if len(sys.argv) > 1:
13        # Eğer parametre verilmişse kullanım bilgisini göster
14        print("Kullanım: python server.py")
15        print("Tüm ayarlar kod içinde DEFAULT_CONFIG'de tanımlanmıştır.")
16        sys.exit(1) # hata kodu ile çıkış
17
18    # Sunucu nesnesini oluştur ve başlat
19    server = SecureFileTransferServer(config)
20    server.start()
21
22 # =====
23 # PROGRAMIN BAŞLANGIÇ NOKTASI
24 # =====
25 if __name__ == "__main__":
26     main()
```

SecureFileTransferServer sınıfı kendi içerisinde pek çok fonksiyon bulundurmaktadır. Sırasıyla bunları da incelemek gerekirse:

`__init__(...)` fonksiyonu yapıçı bir fonksiyondur ve değişken tanımlamalarını üstlenir.

```
1 def __init__(self, config=None):
2     # Konfigürasyon ayarlarını yükle
3     self.config = config or DEFAULT_CONFIG.copy()
4
5     # RSA private key'i saklamak için
6     self.rsa_private_key = None
7
8     # UDP oturum yönetimi için sözlük
9     self.udp_sessions = {}
10
11    # TCP bağlantı yönetimi için sözlük
12    self.tcp_connections = {}
13
14    # Sunucu çalışma durumu
15    self.running = True
16
17    # İstatistik verileri
18    self.stats = {
19        'tcp_connections': 0,          # toplam TCP bağlantı sayısı
20        'udp_sessions': 0,           # toplam UDP oturum sayısı
21        'files_received': 0,         # alınan dosya sayısı
22        'bytes_received': 0,         # alınan toplam byte sayısı
23        'errors': 0                 # hata sayısı
24    }
25
26    # Temizleme thread'i için thread lock
27    self.sessions_lock = threading.Lock()
28
29    # Alınan dosyalar için dizin oluştur
30    self.setup_directories()
31
32    # Sinyal işleyicilerini ayarla (sadece ana thread'de)
33    if threading.current_thread() == threading.main_thread():
34        self.setup_signal_handlers()
```

**setup\_directories(...)** fonksiyonuyla alınan dosyaların kaydedileceği `received_files` klasörü oluşturulur.

```
1 # Gerekli dizinleri oluşturur
2 def setup_directories(self):
3     try:
4         # Alınan dosyalar dizinini oluştur
5         received_dir = Path(self.config['RECEIVED_FILES_DIR'])
6         received_dir.mkdir(parents=True, exist_ok=True) # Üst dizinleri de oluştur
7         self.log(f"Alınan dosyalar dizini: {received_dir.absolute()}", 'INFO')
8     except Exception as e:
9         self.log(f"Dizin oluşturma hatası: {e}", 'ERROR')
10        sys.exit(1) # kritik hata, programı sonlandır
```

**setup\_signal\_handlers(...)** fonksiyonu sunucunun Ctrl + C gibi komutlar ile kapanmasını sağlar.

```
1 # Sinyal yakalama ve işleme fonksiyonlarını ayarlar
2 def setup_signal_handlers(self):
3     def signal_handler(signum, frame):
4         """Sinyal yakalandığında çalışacak fonksiyon"""
5         self.log(f"Sinyal alındı: {signum}. Sunucu kapatılıyor...", 'INFO')
6         self.shutdown()
7         sys.exit(0)
8
9     # SIGINT (Ctrl+C) sinyalini yakala
10    signal.signal(signal.SIGINT, signal_handler)
11    # SIGTERM sinyalini yakala
12    signal.signal(signal.SIGTERM, signal_handler)
```

**log(...)** fonksiyonu aldığı mesajı zaman damgasıyla birlikte terminale yazdırır.

```
1 # Log mesajlarını formatlar ve yazdırır
2 def log(self, message, level='INFO'):
3     # Zaman damgası oluştur
4     timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
5     # Formatlanmış log mesajını yazdır
6     print(f"[{timestamp}] [{level}] {message}")
```

**load\_rsa\_private\_key(...)** fonksiyonuyla beraber raporun başında generator.py programı sonucu oluşan private key yüklenir ve değişkene atanır.

```
1 # RSA private key'i dosyadan yükler
2 def load_rsa_private_key(self):
3     try:
4         # Key dosyasının yolunu al
5         key_path = Path(self.config['PRIVATE_KEY_PATH'])
6
7         # Dosyanın var olup olmadığını kontrol et
8         if not key_path.exists():
9             raise FileNotFoundError(f"Private key dosyası bulunamadı: {key_path}")
10
11        # Dosyayı oku ve RSA anahtarını yükle
12        with open(key_path, "rb") as f:
13            self.rsa_private_key = RSA.import_key(f.read())
14
15        self.log(f"RSA private key yüklendi: {key_path}", 'INFO')
16
17    except Exception as e:
18        self.log(f"RSA key yükleme hatası: {e}", 'ERROR')
19        raise # hatayı yukarı aktar
```

**decrypt\_and\_save\_file(...)** fonksiyonu şifrelenmiş veriyi çözer ve dosyaya kaydeder.

```
1 # Sifrelenmiş veriyi çözer ve dosyaya kaydeden
2 def decrypt_and_save_file(self, encrypted_data, client_info="unknown"):
3     # Boş veri kontrolü
4     if not encrypted_data:
5         self.log(f"[{client_info}] Çözülecek veri alınmadı", 'WARNING')
6         return False
7
8     try:
9         # AES padding'ini kaldır
10        plaintext = unpad(encrypted_data, AES.block_size)
11
12        # Veri formatını parse et: hash(32) + filename_length(2) + filename + file_content
13        if len(plaintext) < 34: # minimum boyut kontrolü (32+2)
14            raise ValueError("Veri formatı geçersiz - çok kısa")
15
16        # İlk 32 byte SHA-256 hash
17        received_hash = plaintext[:32]
18
19        # Sonraki 2 byte dosya adı uzunluğu
20        filename_length = int.from_bytes(plaintext[32:34], 'big')
21
22        # Dosya adı uzunluğu güvenlik kontrolü
23        if filename_length > 255 or filename_length == 0:
24            raise ValueError(f"Geçersiz dosya adı uzunluğu: {filename_length}")
25
26        # Toplam veri uzunluğu kontrolü
27        if len(plaintext) < 34 + filename_length:
28            raise ValueError("Veri formatı geçersiz - dosya adı eksik")
29
30        # Dosya adını çıkar
31        filename = plaintext[34:34+filename_length].decode('utf-8')
32
33        # Dosya içeriğini çıkar
34        file_content = plaintext[34+filename_length:]
35
36        # Dosya adı güvenlik kontrolü
37        if not self.is_safe_filename(filename):
38            raise ValueError(f"Güvenli olmayan dosya adı: {filename}")
39
40        # SHA-256 hash doğrulaması
41        calculated_hash = hashlib.sha256(file_content).digest()
42
43        if calculated_hash != received_hash:
44            raise ValueError("SHA-256 hash doğrulaması başarısız")
45
46        # Dosyayı kaydet
47        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S") # zaman damgası
48        safe_filename = f"{timestamp}_{filename}" # güvenli dosya adı
49        save_path = Path(self.config['RECEIVED_FILES_DIR']) / safe_filename
50
51        # Dosyayı diske yaz
52        with open(save_path, "wb") as f:
53            f.write(file_content)
54
55        # İstatistikleri güncelle
56        self.stats['files_received'] += 1
57        self.stats['bytes_received'] += len(file_content)
58
59        self.log(f"[{client_info}] Dosya başarıyla kaydedildi: {save_path} ({len(file_content)} bytes)", 'INFO')
60        return True
61
62    except Exception as e:
63        self.log(f"[{client_info}] Veri çözme hatası: {e}", 'ERROR')
64        self.stats['errors'] += 1
65        return False
```

Burada önemli kısım dosyayı kaydetmeden önce isim kontrolü yapmasıdır. Şüpheli bir ismi varsa kaydetmez bu şekilde güvenlik sağlanır.

`is_safe_filename(...)` fonksiyonu da argüman olarak aldığı dosya adının güvenli olup olmadığını kontrol eder. Özel karakterli dosya adları ile sistem için tehlikeli işlemler yapılabilmektedir.

```
1 # Güvenli dosya adlarını kontrol eder
2 def is_safe_filename(self, filename):
3     # Boş veya çok uzun dosya adı kontrolü
4     if not filename or len(filename) > 255:
5         return False
6
7     # Tehlikeli karakterleri kontrol et
8     dangerous_chars = ['/', '\\', '..', '<', '>', ':', '"', '|', '?', '*']
9     for char in dangerous_chars:
10        if char in filename:
11            return False
12
13     # Windows sistem dosya adlarını kontrol et
14     system_names = ['CON', 'PRN', 'AUX', 'NUL'] + [f'COM{i}' for i in range(1, 10)] + [f'LPT{i}' for i in range(1, 10)]
15     if filename.upper() in system_names:
16         return False
17
18     return True
```

`authenticate_tcp_client(...)` TCP bağlantısı kurulduğunda bu bağlantının güvenliliğinin garanti edilmesi gereklidir. Bu yüzden istemci tarafından yollanan parola ile sunucudaki parola karşılaştırılır. Parola eşleşirse istemciye “OK” mesajı, eşleşmezse “NO” mesajı gönderilir.

```
1 # TCP istemcisi kimlik doğrulama işlemi
2 def authenticate_tcp_client(self, client_socket, client_addr):
3     try:
4         # İstemciden parola hash'ini al (32 byte SHA-256)
5         received_hash = client_socket.recv(32)
6         if len(received_hash) != 32:
7             self.log(f"[TCP-{client_addr}] Geçersiz hash boyutu", 'WARNING')
8             return False
9
10        # Beklenen parola hash'ini hesapla
11        expected_hash = hashlib.sha256(self.config['PASSWORD']).digest()
12
13        # Hash'leri karşılaştır
14        if received_hash != expected_hash:
15            self.log(f"[TCP-{client_addr}] Kimlik doğrulama başarısız", 'WARNING')
16            client_socket.sendall(b"NO") # başarısız yanıtını gönder
17            return False
18
19        # Başarılı yanıt gönder
20        client_socket.sendall(b"OK")
21        self.log(f"[TCP-{client_addr}] Kimlik doğrulama başarılı", 'INFO')
22        return True
23
24    except Exception as e:
25        self.log(f"[TCP-{client_addr}] Kimlik doğrulama hatası: {e}", 'ERROR')
26        return False
```

```

1 # TCP istemci bağlantısını işler
2 def handle_tcp_client(self, client_socket, client_addr):
3     client_info = f"TCP-{client_addr}"
4     decrypted_data = b"" # çözülmüş veriyi biriktirmek için
5
6     try:
7         # Önce kimlik doğrulaması yap
8         if not self.authenticate_tcp_client(client_socket, client_addr):
9             return
10
11         # RSA ile şifrelenmiş AES anahtarını al (256 byte)
12         encrypted_aes_key = client_socket.recv(256)
13         if len(encrypted_aes_key) != 256:
14             raise ValueError("Geçersiz AES anahtar boyutu")
15
16         # AES IV'sini al (16 byte)
17         iv = client_socket.recv(16)
18         if len(iv) != 16:
19             raise ValueError("Geçersiz IV boyutu")
20
21         # RSA ile AES anahtarını çöz
22         rsa_cipher = PKCS1_OAEP.new(self.rsa_private_key)
23         aes_key = rsa_cipher.decrypt(encrypted_aes_key)
24
25         # AES cipher'i başlat (CBC modu)
26         aes_cipher = AES.new(aes_key, AES.MODE_CBC, iv)
27
28         self.log(f"[{client_info}] Şifreleme anahtarları alındı", 'INFO')
29
30         # Veri paketlerini sürekli al
31         packet_count = 0
32         while self.running:
33             try:
34                 # Paket boyutunu al (4 byte big-endian)
35                 size_data = client_socket.recv(4)
36                 if not size_data or len(size_data) != 4:
37                     break # bağlantı kapatıldı
38
39                 packet_size = int.from_bytes(size_data, 'big')
40
41                 # Paket boyutu güvenlik kontrolü
42                 if packet_size > self.config['MAX_PACKET_SIZE'] or packet_size <= 0:
43                     self.log(f"[{client_info}] Geçersiz paket boyutu: ({packet_size})", 'WARNING')
44                     client_socket.sendall(b"RETRY") # tekrar gönder talebi
45                     continue
46
47                 # CRC değerini al (4 byte)
48                 crc_data = client_socket.recv(4)
49                 if len(crc_data) != 4:
50                     break
51
52                 expected_crc = int.from_bytes(crc_data, 'big')
53
54                 # Paketi tamamen al
55                 packet = b""
56                 while len(packet) < packet_size:
57                     chunk = client_socket.recv(packet_size - len(packet))
58                     if not chunk:
59                         break
60                     packet += chunk
61
62                 # Paket tam alınamadıysa
63                 if len(packet) != packet_size:
64                     self.log(f"[{client_info}] Eksik paket alındı", 'WARNING')
65                     client_socket.sendall(b"RETRY")
66                     continue
67
68                 # CRC kontrolü yap
69                 calculated_crc = zlib.crc32(packet) & 0xffffffff
70                 if calculated_crc != expected_crc:
71                     self.log(f"[{client_info}] CRC hatası (beklenen: {expected_crc}, " +
72                             "hesaplanan: {calculated_crc})", 'WARNING')
73                     client_socket.sendall(b"RETRY")
74                     continue
75
76                 # Paketi AES ile çöz ve biriktir
77                 decrypted_packet = aes_cipher.decrypt(packet)
78                 decrypted_data += decrypted_packet
79
80                 packet_count += 1
81                 client_socket.sendall(b"OK") # başarılı yanıt gönder
82
83                 # Her 10 pakette bir ilerleme logu
84                 if packet_count % 10 == 0:
85                     self.log(f"[{client_info}] (packet_count) paket alındı", 'DEBUG')
86
87                 except socket.timeout:
88                     self.log(f"[{client_info}] Paket alma timeout", 'WARNING')
89                     break
90                 except Exception as e:
91                     self.log(f"[{client_info}] Paket işleme hatası: {e}", 'ERROR')
92                     break
93
94             self.log(f"[{client_info}] Toplam {packet_count} paket alındı", 'INFO')
95
96         except Exception as e:
97             self.log(f"[{client_info}] İstemci işleme hatası: {e}", 'ERROR')
98             self.stats['errors'] += 1
99
100     finally:
101         # Socket'i kapat
102         try:
103             client_socket.close()
104         except:
105             pass
106
107         # Bağlantı listesinden çıkar
108         with self.sessions_lock:
109             if client_addr in self.tcp_connections:
110                 del self.tcp_connections[client_addr]
111
112         # Toplanan veriyi çöz ve kaydet
113         if decrypted_data:
114             self.decrypt_and_save_file(decrypted_data, client_info)

```

**handle\_tcp\_client(...)** fonksiyonu sunucunun bütün TCP işlemlerini üstlenir. İstemci tarafından gelen istekler burada değerlendirilir. TCP protokolü olduğu için güvenlik işlemleri yapılmaktadır. Ayrıca paket传递 garantisinin sağlanması için hatalı bir paket alındığında istemciye “RETRY” kodu yollar ve bu sayede paketi tekrar ister. Bütün paketler eksiksiz bir şekilde alınana kadar bağlantı devam eder. Bütün paketler alındıktan sonra raporun önceki kısmında anlatılan verinin şifresini çözen ve kaydeden fonksiyon ile alınan paketleri dosyaya kaydeder.

```

1 # UDP paketlerini işler
2 def handle_udp_packet(self, data, client_addr, udp_socket):
3     client_info = f"UDP-{client_addr}"
4
5     try:
6         # PING paketini yanıtla (bağlantı testi için)
7         if data == b"PING":
8             udp_socket.sendto(b"PONG", client_addr)
9             return
10
11         with self.sessions_lock:
12             # Yeni oturum kurulumu
13             if client_addr not in self.udp_sessions:
14                 # Minimum paket boyutu kontrolü (256 RSA + 16 IV)
15                 if len(data) < 272:
16                     self.log(f"[{client_info}] Geçersiz kurulum paketi boyutu", 'WARNING')
17                     return
18
19             self.log(f"[{client_info}] Yeni UDP oturumu başlatılıyor", 'INFO')
20
21             # RSA ile şifrelenmiş AES anahtarını ve IV'yi ayırt
22             encrypted_aes_key = data[:256] # ilk 256 byte
23             iv = data[256:272] # sonraki 16 byte
24
25             # RSA ile AES anahtarını çöz
26             rsa_cipher = PKCS1_OAEP.new(self.rsa_private_key)
27             aes_key = rsa_cipher.decrypt(encrypted_aes_key)
28
29             # Yeni oturum oluştur
30             self.udp_sessions[client_addr] = {
31                 "cipher": AES.new(aes_key, AES.MODE_CBC, iv), # AES cipher objesi
32                 "data": b"",
33                 "last_seen": time.time(), # son görülmeye zamanı
34                 "packet_count": 0 # alınan paket sayısı
35             }
36
37             self.stats['udp_sessions'] += 1
38             return
39
40         # Mevcut oturum işlemi
41         session = self.udp_sessions[client_addr]
42         session["last_seen"] = time.time() # son görülmeye zamanını güncelle
43
44         # Oturum bitirme paketi
45         if data == b"FIN":
46             self.log(f"[{client_info}] UDP oturumu tamamlandı ({session['packet_count']} paket)", 'INFO')
47             # Biriktirilen veriyi çöz ve kaydet
48             self.decrypt_and_save_file(session["data"], client_info)
49             del self.udp_sessions[client_addr] # oturumu sil
50             return
51
52         # Veri paketi işlemi
53         if len(data) < 8: # minimum header boyutu (4+4)
54             self.log(f"[{client_info}] Geçersiz paket boyutu", 'WARNING')
55             return
56
57         # Paket başlığını parse et
58         packet_size = int.from_bytes(data[0:4], 'big') # paket boyutu
59         expected_crc = int.from_bytes(data[4:8], 'big') # beklenen CRC
60
61         # Paket boyutu güvenlik kontrolü
62         if packet_size > self.config['MAX_PACKET_SIZE'] or packet_size <= 0:
63             self.log(f"[{client_info}] Geçersiz paket boyutu: {packet_size}", 'WARNING')
64             return
65
66         # Paket verisi tam mı?
67         if len(data) < 8 + packet_size:
68             self.log(f"[{client_info}] Eksik paket verisi", 'WARNING')
69             return
70
71         # Paket verisini çıkar
72         packet = data[8:8+packet_size]
73
74         # CRC kontrolü yap
75         calculated_crc = zlib.crc32(packet) & 0xffffffff
76         if calculated_crc != expected_crc:
77             self.log(f"[{client_info}] CRC hatası, paket atlandı", 'WARNING')
78             return
79
80         # Paketi AES ile çöz ve biriktir
81         decrypted_packet = session["cipher"].decrypt(packet)
82         session["data"] += decrypted_packet
83         session["packet_count"] += 1
84
85         # Her 10 pakette bir ilerleme logu
86         if session["packet_count"] % 10 == 0:
87             self.log(f"[{client_info}] (session['packet_count']) paket alındı", 'DEBUG')
88
89     except Exception as e:
90         self.log(f"[{client_info}] UDP paket işleme hatası: {e}", 'ERROR')
91         self.stats['errors'] += 1

```

**handle\_udp\_packet(...)** fonksiyonu da sunucunun bütün UDP bağlantılarını üstlenir. TCP versiyonundan farklı olan birkaç yanı vardır. Bağlantı testi için “ping” mesajına “pong” mesajıyla cevap verir. TCP fonksiyonunun aksine parola kontrolü yapmaz ve paketler eksik gelirse geri istemez. Ancak içeriği “FIN” olan bir paket gelirse UDP paketlerinin tamamlandığı anlaşılır ve o ana kadar alınan paketler birleştirilip veri kaydedilir.

**cleanup\_expired\_sessions(...)** UDP bağlantısı sağlandığında eğer kapatılmazsa sorunlar ortaya çıkabilmektedir. Bu fonksiyon ile oturumlar taranır ve süresi dolmuşlar tespit edilir ardından bu oturumlar temizlenir.

```
1 # Süresi dolmuş UDP oturumlarını temizler
2 def cleanup_expired_sessions(self):
3     while self.running:
4         try:
5             # Temizleme aralığı kadar bekle
6             time.sleep(self.config['CLEANUP_INTERVAL'])
7
8             current_time = time.time()
9             expired_sessions = [] # süresi dolmuş oturumlar
10
11            # Süresi dolmuş oturumları tespit et
12            with self.sessions_lock:
13                for addr, session in self.udp_sessions.items():
14                    # Son görülmeye zamanından bu yana geçen süre
15                    if current_time - session["last_seen"] > self.config['SESSION_TIMEOUT']:
16                        expired_sessions.append(addr)
17
18            # Süresi dolmuş oturumları işley
19            for addr in expired_sessions:
20                with self.sessions_lock:
21                    if addr in self.udp_sessions:
22                        session = self.udp_sessions[addr]
23                        self.log(f"[UDP-{addr}] Oturum zaman aşımına uğradı ({session['packet_count']} paket)", 'WARNING')
24
25                        # Kisiyi varsa kaydetmeyi dene
26                        if session["data"]:
27                            self.decrypt_and_save_file(session["data"], f"UDP-{addr}")
28
29                        del self.udp_sessions[addr] # oturumu sil
30
31            # Temizleme istatistikleri
32            if expired_sessions:
33                self.log(f"Temizlenen oturum sayısı: {len(expired_sessions)}", 'INFO')
34
35        except Exception as e:
36            self.log(f"Oturum temizleme hatası: {e}", 'ERROR')
```

**tcp\_server(...)** fonksiyonu sunucunun TCP isteklerini aktif olarak dinleyeceği sunucuyu

```
1 # TCP sunucusunu başlatır ve bağlantıları kabul eder
2 def tcp_server(self):
3     try:
4         self.log(f"TCP sunucusu başlatılıyor: {self.config['HOST']}:{self.config['TCP_PORT']}", 'INFO')
5
6         # TCP socket oluştur
7         tcp_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
8         tcp_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1) # adres yeniden kullanımı
9         tcp_socket.bind((self.config['HOST'], self.config['TCP_PORT'])) # adres ve porta bağla
10        tcp_socket.listen(self.config['MAX_CONNECTIONS']) # dinlemeye başla
11        tcp_socket.settimeout(1.0) # timeout ayarla
12
13        # Ana bağlantı kabul döngüsü
14        while self.running:
15            try:
16                # Yeni bağlantı kabul et
17                client_socket, client_addr = tcp_socket.accept()
18
19                # Bağlantı limiti kontrolü
20                with self.sessions_lock:
21                    if len(self.tcp_connections) >= self.config['MAX_CONNECTIONS']:
22                        self.log(f"TCP bağlantı limiti aşıldı, bağlantı reddedildi: {client_addr}", 'WARNING')
23                        client_socket.close()
24                        continue
25
26                    # Yeni bağlantıyı kaydet
27                    self.tcp_connections[client_addr] = {
28                        'socket': client_socket,
29                        'start_time': time.time()
30                    }
31
32                    self.log(f"Yeni TCP bağlantısı: {client_addr}", 'INFO')
33                    self.stats['tcp_connections'] += 1
34
35                    # İstemciyi ayrı thread'de işle
36                    client_thread = threading.Thread(
37                        target=self.handle_tcp_client,
38                        args=(client_socket, client_addr),
39                        daemon=True # ana program kapanlığında thread'i de kapat
40                    )
41                    client_thread.start()
42
43                    except socket.timeout:
44                        continue # timeout olursa devam et
45                    except Exception as e:
46                        if self.running:
47                            self.log(f"TCP sunucusu hatası: {e}", 'ERROR')
48
49                    except Exception as e:
50                        self.log(f"TCP sunucusu başlatma hatası: {e}", 'ERROR')
51
52                    # Socket'i kapat
53                    try:
54                        tcp_socket.close()
55                    except:
56                        pass
```

açar. Bu TCP sunucusu soket ile açılır ve istemciden bağlantı gelmesini bekler. Herhangi bir bağlantı gelirse thread açarak eş zamanlı olarak işlemeye başlar.

**udp\_server(...)** fonksiyonu **tcp\_server(...)** fonksiyonu gibi sunucu için bir UDP sunucusu açar. Bu işlemleri soket üzerinden yapar. Sunucunun TCP ve UDP sunucuları eş zamanlı olarak thread bazlı çalışır. Bu sayede birden çok bağlantı eş zamanlı olarak yapılabilmektedir.

```

1 # UDP sunucusunu başlatır ve paketleri alır
2 def udp_server(self):
3     try:
4         # Sunucu başlatma mesajını logla
5         self.log(f"UDP sunucusu başlatılıyor: {self.config['HOST']}:{self.config['UDP_PORT']}", 'INFO')
6
7         # UDP socket oluştur (SOCK_DGRAM = UDP protokolü)
8         udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
9         udp_socket.bind((self.config['HOST'], self.config['UDP_PORT'])) # adres ve porta bağla
10        udp_socket.settimeout(1.0) # timeout ayarla (1 saniye)
11
12        # Ana paket alma döngüsü - sunucu çalıştığı sürece devam eder
13        while self.running:
14            try:
15                # UDP paketini al (bloklanır, timeout ile sınırlı)
16                data, client_addr = udp_socket.recvfrom(self.config['MAX_PACKET_SIZE'])
17
18                # Paketi ayrı thread'de işle (ana dönemin bloklanması önlemek için)
19                packet_thread = threading.Thread(
20                    target=self.handle_udp_packet, # çalıştırılacak fonksiyon
21                    args=(data, client_addr, udp_socket), # fonksiyona gönderilecek parametreler
22                    daemon=True # ana program kapandığında bu thread'i de otomatik kapat
23                )
24                packet_thread.start() # thread'i başlat
25
26            except socket.timeout:
27                # Timeout durumunda döngüye devam et (normal davranış)
28                continue
29            except Exception as e:
30                # Diğer hatalar için log kaydet (eğer sunucu hala çalışıyorsa)
31                if self.running:
32                    self.log(f"UDP paket alma hatası: {e}", 'ERROR')
33
34            except Exception as e:
35                # Sunucu başlatma hatalarını logla
36                self.log(f"UDP sunucu başlatma hatası: {e}", 'ERROR')
37        finally:
38            # Her durumda socket'i güvenli şekilde kapat
39            try:
40                udp_socket.close()
41            except:
42                pass # kapatma hatalarını yoksay

```

**print\_stats(...)** fonksiyonu genel sunucudaki anlık TCP ve UDP bağlantılarını, eskiden

```

1 # İstatistikleri yazdırır (aktif bağlantı sayıları, alınan dosyalar vs.)
2 def print_stats(self):
3     # Sunucu çalıştığı sürece istatistik yazdırılmaya devam et
4     while self.running:
5         try:
6             time.sleep(30) # 30 saniyede bir istatistik yazdır
7
8             # Thread-safe şekilde aktif bağlantı sayılarını al
9             with self.sessions_lock:
10                 active_tcp = len(self.tcp_connections) # aktif TCP bağlantı sayısı
11                 active_udp = len(self.udp_sessions) # aktif UDP oturum sayısı
12
13                 # İstatistikleri logla
14                 self.log(f"--- İSTATİSTİKLER ---", 'INFO')
15                 self.log(f"Aktif TCP bağlantıları: {active_tcp}", 'INFO')
16                 self.log(f"Aktif UDP oturumları: {active_udp}", 'INFO')
17                 self.log(f"Toplam TCP bağlantıları: {self.stats['tcp_connections']}", 'INFO')
18                 self.log(f"Toplam UDP oturumları: {self.stats['udp_sessions']}", 'INFO')
19                 self.log(f"Alınan dosya sayısı: {self.stats['files_received']}", 'INFO')
20                 self.log(f"Alınan toplam veri: {self.stats['bytes_received']} bytes", 'INFO')
21                 self.log(f"Hata sayısı: {self.stats['errors']}", 'INFO')
22                 self.log(f"--- İSTATİSTİKLER ---", 'INFO')
23
24             except Exception as e:
25                 # İstatistik yazdırma hatalarını logla
26                 self.log(f"İstatistik yazdırma hatası: {e}", 'ERROR')

```

yapılmış bağlantıları, alınan dosyalar gibi bilgileri her 30 saniyede bir ekrana yazdırır.

**start(...)** fonksiyonu hem UDP sunucusunu hem de TCP sunucusunu thread açarak eş zamanlı olarak başlatır.

```
1 # Sunucuyu başlatır ve gerekli thread'leri oluşturur
2 def start(self):
3     try:
4         # Başlatma mesajını logla
5         self.log("Güvenli Dosya Transfer Sunucusu başlatılıyor...", 'INFO')
6
7         # RSA private key'i dosyadan yükle (şifre çözme için gerekli)
8         self.load_rsa_private_key()
9
10        # Sunucu thread'lerini oluştur (her biri farklı görev için)
11        tcp_thread = threading.Thread(target=self.tcp_server, daemon=True)           # TCP bağlantıları için
12        udp_thread = threading.Thread(target=self.udp_server, daemon=True)           # UDP paketleri için
13        cleanup_thread = threading.Thread(target=self.cleanup_expired_sessions, daemon=True) # süresi geçmiş oturumları temizle
14        stats_thread = threading.Thread(target=self.print_stats, daemon=True)          # istatistik yazdırma için
15
16        # Tüm thread'leri başlat
17        tcp_thread.start()
18        udp_thread.start()
19        cleanup_thread.start()
20        stats_thread.start()
21
22        # Başarılı başlatma mesajları
23        self.log("Sunucu başarıyla başlatıldı. TCP ve UDP dinleniyor...", 'INFO')
24        self.log(f"TCP Port: {self.config['TCP_PORT']}", 'INFO')
25        self.log(f"UDP Port: {self.config['UDP_PORT']}", 'INFO')
26        self.log("Durdurmak için Ctrl+C tuşlayın.", 'INFO')
27
28        # Ana thread'i canlı tut (diğer thread'ler daemon olduğu için gerekli)
29        while self.running:
30            time.sleep(1) # CPU kullanımını azaltmak için kısa bekleme
31
32    except Exception as e:
33        # Sunucu başlatma hatalarını logla ve kapat
34        self.log(f"Sunucu başlatma hatası: {e}", 'ERROR')
35        self.shutdown()
```

**shutdown(...)** fonksiyonu da TCP ve UDP oturumlarını kapatır.

```
1 # Sunucuyu güvenli şekilde kapatır
2 def shutdown(self):
3     # Kapatma mesajını logla
4     self.log("Sunucu kapatılıyor...", 'INFO')
5     self.running = False # ana döngüleri durdurmak için flag'i false yap
6
7     # Aktif bağlantıları güvenli şekilde kapat
8     with self.sessions_lock:
9         # Tüm TCP bağlantılarını kapat
10        for addr, conn_info in self.tcp_connections.items():
11            try:
12                conn_info['socket'].close() # socket'i kapat
13            except:
14                pass # kapatma hatalarını yoksay
15
16        # Kalan UDP oturumlarındaki verileri işle (veri kaybını önlemek için)
17        for addr, session in self.udp_sessions.items():
18            if session["data"]:
19                # Veriyi şifre çöz ve dosya olarak kaydet
20                self.decrypt_and_save_file(session["data"], f"UDP-{addr}")
21
22        # Başarılı kapatma mesajını logla
23        self.log("Sunucu başarıyla kapatıldı.", 'INFO')
```

- **client.py**

İstemci SecureFileTransferClient sınıfından oluşmaktadır. Bu sınıf içerisinde pek çok fonksiyon vardır. Bu fonksiyonlar sırasıyla bölüm bölüm inceleneciktir.

DEFAULT\_CONFIG değişkeni ile istemcinin herhangi bir özelleştirme olmadan başlatılması durumunda kullanacağı varsayılan bilgiler tutulur. İstemci başlatılırken aşağıdaki bilgiler değiştirilebilir.

```
1 # =====
2 # VARSAYILAN AYARLAR
3 # =====
4 DEFAULT_CONFIG = {
5     'HOST': '127.0.0.1',                      # varsayılan hedef IP adresi (localhost)
6     'TCP_PORT': 5001,                          # TCP portu
7     'UDP_PORT': 5002,                          # UDP portu
8     'PASSWORD': b"gizli_sifre",                # kimlik doğrulama parolası
9     'FILENAME': "deneme.txt",                  # varsayılan gönderilecek dosya
10    'PACKET_SIZE': 1024,                      # paket boyutu (byte)
11    'SEND_COUNT': 1,                          # dosyanın kaç kez gönderileceği
12    'TIMEOUT': 5.0,                           # bağlantı timeout süresi (saniye)
13    'LATENCY_THRESHOLD': 50.0,                 # hibrit mod için gecikme eşiği (ms)
14    'PUBLIC_KEY_PATH': "../keys/public.pem"  # RSA public key dosya yolu
15 }
```

`print_usage()` fonksiyonu çağrıldığında ekrana istemcinin nasıl kullanılması gerektiğine dair bilgileri gösterir. Örneğin DEFAULT\_CONFIG ayarları yerine özelleştirilmiş ayarlar ile istemcinin başlatılması isteniyorsa buradaki bilgilere bakılarak başlatılabilir.

```
1 # =====
2 # KULLANIM KILAVUZU FONKSİYONU
3 # =====
4 def print_usage():
5     print("\n==> Güvenli Dosya Transfer İstemcisi ==")
6     print("Kullanım:")
7     print("  python client.py [mod] [host] [port] [paket_boyutu] [gönderim_sayısı] [dosya_adı]")
8     print("\nParametreler:")
9     print("  mod          : TCP, UDP veya HYBRID")
10    print("  host         : Hedef IP adresi")
11    print("  port         : Hedef port numarası")
12    print("  paket_boyutu : Paket boyutu (bytes)")
13    print("  gönderim_sayı : Dosyanın kaç kez gönderileceği")
14    print("  dosya_adı   : Gönderilecek dosyanın adı")
15    print("\nÖrnekler:")
16    print("  python client.py TCP 192.168.1.100 5001 1024 1 test.txt")
17    print("  python client.py UDP 127.0.0.1 5002 512 3 document.pdf")
18    print("  python client.py HYBRID 10.0.0.1 5001 2048 1 video.mp4")
19    print("\nVarsayılan değerler kullanmak için parametresiz çalıştırın:")
20    print("  python client.py")
```

**main()** fonksiyonu SecureFileTransferClient sınıfından bir nesne oluşturur ve istemciyi başlatır. Ayrıca istemci başlatılırken özelleştirmeler de yapıldıysa burada argüman yakalama ile alınır ve o şekilde başlatılır.

```
1 # =====
2 # ANA PROGRAM
3 # =====
4 def main():
5     # Komut satırı argümanlarının sayısını kontrol et
6     if len(sys.argv) == 1:
7         # Parametre verilmemiş - varsayılan değerlerle çalıştır
8         mode = "TCP"
9         host = DEFAULT_CONFIG['HOST']
10        port = DEFAULT_CONFIG['TCP_PORT']
11        packet_size = DEFAULT_CONFIG['PACKET_SIZE']
12        send_count = DEFAULT_CONFIG['SEND_COUNT']
13        filename = DEFAULT_CONFIG['FILENAME']
14
15        print("[*] Varsayılan ayarlarla çalıştırılıyor...")
16
17    elif len(sys.argv) == 7:
18        # Tüm parametreler verilmiş - parse et ve doğrula
19        try:
20            mode = sys.argv[1].upper()           # mod adını büyük harfe çevir
21            host = sys.argv[2]                # hedef IP adresi
22            port = int(sys.argv[3])          # port numarasını integer'a çevir
23            packet_size = int(sys.argv[4])    # paket boyutunu integer'a çevir
24            send_count = int(sys.argv[5])     # gönderim sayısını integer'a çevir
25            filename = sys.argv[6]           # dosya adı
26
27            # Parametrelerin geçerliliğini kontrol et
28            if mode not in ["TCP", "UDP", "HYBRID"]:
29                raise ValueError(f"Geçersiz mod: {mode}")
30            if port < 1 or port > 65535:
31                raise ValueError(f"Geçersiz port: {port}")
32            if packet_size < 64 or packet_size > 65507:
33                raise ValueError(f"Geçersiz paket boyutu: {packet_size}")
34            if send_count < 1:
35                raise ValueError(f"Geçersiz gönderim sayısı: {send_count}")
36
37        except (ValueError, IndexError) as e:
38            print(f"[!] Parametre hatası: {e}")
39            print_usage()
40            sys.exit(1) # hata kodu ile çıkış
41        else:
42            # Yanlış parametre sayısı
43            print("[!] Hatalı parametre sayısı")
44            print_usage()
45            sys.exit(1)
46
47        # İstemci nesnesini oluştur ve çalıştır
48        client = SecurefileTransferClient()
49        success = client.run(mode, host, port, packet_size, send_count, filename)
50
51        # Sonuca göre çıkış kodu belirle
52        if success:
53            print(f"\n[+] Transfer başarıyla tamamlandı!")
54            sys.exit(0) # başarı kodu
55        else:
56            print(f"\n[!] Transfer başarısız!")
57            sys.exit(1) # hata kodu
58
59 # =====
60 # PROGRAMIN BAŞLANGIÇ NOKTASI
61 # =====
62 if __name__ == "__main__":
63     main()
```

SecureFileTransferClient sınıfı kendi içerisinde pek çok fonksiyon içermektedir. Bu fonksiyonlar raporun ilerleyen kısımlarında sırasıyla inceleneciktir.

**run(...)** fonksiyonu ile SecureFileTransferClient sınıfından oluşturulan nesne yani istemci alınan argümanlarla başlatılır. Kullanıcının girdiği veya DEFAULT\_CONFIG değişkeninden alınan TCP, UDP veya HYBRID bilgisine göre istemci oluşturulmuş olunur. HYBRID mod sayesinde ağ durumuna göre UDP veya TCP protokolü ile dosya gönderimi yapılır.

```
1 # Ana çalışma fonksiyonu - mod, host, port, paket boyutu, gönderim sayısı ve dosya adını alır
2 def run(self, mode, host, port, packet_size, send_count, filename):
3     print(f"[*] Güvenli Dosya Transfer İstemcisi Başlatılıyor...")
4     print(f"[*] Mod: {mode}, Hedef: {host}:{port}")
5     print(f"[*] Paket boyutu: {packet_size}, Gönderim sayısı: {send_count}")
6
7     try:
8         # RSA public key'i yükle (şifreleme için gerekli)
9         self.load_rsa_public_key()
10
11        # Seçilen moda göre ilgili metodu çağır
12        if mode == "TCP":
13            return self.send_file_tcp(host, port, packet_size, send_count, filename)
14        elif mode == "UDP":
15            return self.send_file_udp(host, port, packet_size, send_count, filename)
16        elif mode == "HYBRID":
17            # Hibrit mod için hem TCP hem UDP portları gerekli
18            tcp_port = self.config['TCP_PORT']
19            udp_port = self.config['UDP_PORT']
20            return self.send_file_hybrid(host, tcp_port, udp_port, packet_size, send_count, filename)
21        else:
22            print(f"[!] Geçersiz mod: {mode}")
23            print("[*] Geçerli modlar: TCP, UDP, HYBRID")
24            return False
25
26    except Exception as e:
27        print(f"[!] Kritik hata: {e}")
28        return False
```

**\_\_init\_\_(...)** fonksiyonu SecureFileTransferClient sınıfının yapıcı fonksiyonudur. Değişkenleri tanımlar.

```
1 def __init__(self, config=None):
2     # Konfigürasyon ayarlarını yükle (özel ayar yoksa varsayılanları kullan)
3     self.config = config or DEFAULT_CONFIG.copy()
4     self.rsa_public_key = None # RSA public key objesi
```

`load_rsa_public_key(...)` fonksiyonu ile gönderilecek verileri şifrelerken kullanılacak anahtar yüklenir ve değişkende tutulur.

```
1 # RSA public key'i dosyadan yükleyen fonksiyon
2 def load_rsa_public_key(self):
3     try:
4         # Key dosyasının yolunu al
5         key_path = Path(self.config['PUBLIC_KEY_PATH'])
6
7         # Dosya varlığını kontrol et
8         if not key_path.exists():
9             raise FileNotFoundError(f"Public key dosyası bulunamadı: {key_path}")
10
11        # RSA key'i dosyadan yükle
12        with open(key_path, "rb") as f:
13            self.rsa_public_key = RSA.import_key(f.read())
14
15        print(f"[+] RSA public key yüklandı: {key_path}")
16
17    except Exception as e:
18        raise Exception(f"RSA key yükleme hatası: {e}")
```

`encrypt_file_data(...)` fonksiyonu rastgele oluşturulan AES ile veriyi şifreler `load_rsa_public_key(...)` ile yüklenen RSA anahtarı ile de bu AES anahtarını şifreler. Ardından şifrelenmiş veriyi ve şifrelenmiş AES anahtarını döndürür.

```
1 # Dosya verisini şifreleyen fonksiyon
2 def encrypt_file_data(self, file_data, filename):
3     # Güvenli rastgele anahtar ve IV oluştur
4     aes_key = get_random_bytes(32) # 256-bit AES anahtarı
5     iv = get_random_bytes(16)      # 128-bit başlangıç vektörü
6
7     # Dosya hash'i hesapla (büyük olasılık kontrolü için)
8     file_hash = hashlib.sha256(file_data).digest()
9
10    # Dosya adını bytes formatına çevir
11    filename_bytes = filename.encode('utf-8')
12    filename_length = len(filename_bytes).to_bytes(2, 'big') # big-endian format
13
14    # Veri paketini oluştur: hash + dosya_adi_uzunluğu + dosya_adi + dosya_verisi
15    data_package = file_hash + filename_length + filename_bytes + file_data
16
17    # AES ile veriyi şifrele (CBC modu)
18    aes_cipher = AES.new(aes_key, AES.MODE_CBC, iv=iv)
19    encrypted_data = aes_cipher.encrypt(pad(data_package, AES.block_size))
20
21    # AES anahtarını RSA ile şifrele (OAEP padding ile)
22    rsa_cipher = PKCS1_OAEP.new(self.rsa_public_key)
23    encrypted_aes_key = rsa_cipher.encrypt(aes_key)
24
25    return encrypted_data, encrypted_aes_key, iv
```

**measure\_network\_latency(...)** fonksiyonu ile ağ gecikmesi ölçülür. Eğer ki istemci başlatma modu HYBRID seçilmişse istemci sunucuya bu fonksiyon ile bir ping atar ve bu sayede gecikmeyi ölçümiş olur.

```
1 # Ağ gecikmesini ölçen fonksiyon
2 def measure_network_latency(self):
3     try:
4         # Zaman ölçümü başlat
5         start_time = time.perf_counter()
6
7         # UDP ping paketi gönder
8         with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as sock:
9             sock.settimeout(2.0) # 2 saniye timeout
10            sock.sendto(b"PING", (self.config['HOST'], self.config['UDP_PORT']))
11
12        try:
13            sock.recv(64) # sunucudan pong yanıtını bekle
14        except socket.timeout:
15            print("![] Ping yanıtı alınamadı, varsayılan gecikme kullanılıyor")
16            return 100.0 # timeout durumunda yüksek gecikme döndür
17
18        # Zaman ölçümü bitir
19        end_time = time.perf_counter()
20        latency = (end_time - start_time) * 1000 # milisaniyeye çevir
21
22    return latency
23
24 except Exception as e:
25     print(f"![] Gecikme ölçüm hatası: {e}")
26     return 100.0 # hata durumunda varsayılan yüksek gecikme
```

**authenticate\_tcp\_connection(...)** fonksiyonu sunucuya hem istemcide hem sunucuda var olan parolayı gönderir. Sunucu istemcinin gönderdiği parolayı onaylarsa “OK” mesajı ile yanıt verir, “OK” mesajı gelmezse parola uyuşmazlığından TCP bağlantısı yapılamaz.

```
1 # TCP bağlantısı için kimlik doğrulama işlemi
2 def authenticate_tcp_connection(self, sock):
3     try:
4         # Parolanın SHA256 hash'ini hesapla
5         password_hash = hashlib.sha256(self.config['PASSWORD']).digest()
6
7         # Hash'i sunucuya gönder
8         sock.sendall(password_hash)
9
10        # Sunucudan yanıt bekle
11        response = sock.recv(2)
12
13        if response == b"OK":
14            print("[+] Kimlik doğrulama başarılı")
15            return True
16        else:
17            print(f"![] Kimlik doğrulama başarısız: {response}")
18            return False
19
20    except Exception as e:
21        print(f"![] Kimlik doğrulama hatası: {e}")
22        return False
```

```

1 # TCP protokoli ile güvenli dosya gönderimi
2 def send_file_tcp(self, host, port, packet_size, send_count, filename):
3     print(f"[MODE] TCP - Dosya: {filename}")
4
5     # Dosya varlığını kontrol et
6     if not Path(filename).exists():
7         print(f"[!] Dosya bulunamadı: {filename}")
8         return False
9
10    client_socket = None
11    try:
12        # TCP socket oluştur ve bağlan
13        client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
14        client_socket.settimeout(self.config['TIMEOUT']) # timeout ayarla
15        client_socket.connect((host, port))
16        print(f"[+] {host}:{port} adresine bağlandı")
17
18        # Kimlik doğrulama işlemi
19        if not self.authenticate_tcp_connection(client_socket):
20            return False
21
22        # Dosyayı binary moda oku
23        with open(filename, "rb") as f:
24            file_data = f.read()
25
26        print(f"[+] Dosya okundu: {len(file_data)} bytes")
27
28        # Dosyayı şifrele
29        encrypted_data, encrypted_aes_key, iv = self.encrypt_file_data(
30            file_data, Path(filename).name
31        )
32
33        # Şifrelenmiş AES anahtarını ve IV'yi gönder
34        client_socket.sendall(encrypted_aes_key) # RSA ile şifrelenmiş AES key
35        client_socket.sendall(iv) # AES başlangıç vektörü
36
37        print(f"[+] Şifreleme anahtarları gönderildi")
38
39        # Toplam paket sayısını hesapla
40        total_packets = len(encrypted_data) // packet_size + (1 if len(encrypted_data) % packet_size else 0)
41
42        # Belirlenen sayıda gönderim turu yap
43        for send_round in range(send_count):
44            print(f"[*] Gönderim turu: {send_round + 1}/{send_count}")
45
46            # Şifrelenmiş veriyi paketler halinde gönder
47            for i in range(0, len(encrypted_data), packet_size):
48                packet_data = encrypted_data[i:i + packet_size] # mevcut paket verisi
49                packet_number = i // packet_size + 1 # paket numarası
50
51                # CRC32 checksum hesapla (veri bütünlüğü için)
52                crc = zlib.crc32(packet_data) & 0xffffffff
53
54                # Paket başlığı oluştur: boyut (4 byte) + CRC (4 byte)
55                header = len(packet_data).to_bytes(4, 'big') + crc.to_bytes(4, 'big')
56
57                # Paket gönderimi ve yeniden deneme mantığı
58                max_retries = 3 # maksimum deneme sayısı
59                retry_count = 0 # mevcut deneme sayacı
56
59
60                # Paket başarılı gönderilene kadar dene
61                while retry_count < max_retries:
62                    try:
63                        # Paket başlığı + veriyi gönder
64                        client_socket.sendall(header + packet_data)
65
66                        # Sunucudan yanıt bekle
67                        response = client_socket.recv(5)
68
69                        if response == b'RETRY':
70                            # Paket bozulmuş, yeniden gönder
71                            retry_count += 1
72                            print(f"[!] Paket {packet_number} yeniden gönderiliyor ({retry_count}/{max_retries})")
73                            time.sleep(0.1) # kısa bekleme
74                            continue
75
76                        elif response == b'OK':
77                            # Paket başarıyla alındı
78                            break
79
80                        else:
81                            print(f"[!] Beklenmeyen yanıt: {response}")
82                            break
83
84                    except socket.timeout:
85                        retry_count += 1
86                        print(f"[!] Timeout - Paket {packet_number} yeniden gönderiliyor")
87                        continue
88
89                # Maksimum deneme sayısı aşıldığsa hata ver
90                if retry_count >= max_retries:
91                    print(f"[!] Paket {packet_number} gönderilemedi, maksimum deneme sayısı aşıldı")
92                    return False
93
94                # İllerleme göstergesi (her 10 pakette bir veya son pakette)
95                if packet_number % 10 == 0 or packet_number == total_packets:
96                    print(f"[*] İllerleme: {packet_number}/{total_packets} paket gönderildi")
97
98        print(f"[+] TCP ile {filename} dosyası başarıyla gönderildi")
99        return True
100
101    except Exception as e:
102        print(f"[!] TCP gönderim hatası: {e}")
103        return False
104
105    finally:
106        # Socket'i kapat
107        if client_socket:
108            client_socket.close()

```

## send\_file\_tcp(...)

fonksiyonu TCP protokolü ile güvenli dosya transferi yapar. Önce dosyayı AES ile şifreler ve RSA ile anahtarı korur. Şifrelenmiş dosyayı küçük paketler halinde karşı tarafa gönderir. Her paket için onay bekler, başarısız paketleri tekrar gönderir. Böylece dosyalar güvenli ve kayıpsız şekilde aktarılır.

**send\_file\_udp(...)** fonksiyonu UDP protokolü ile güvenli dosya transferi yapar. Dosyayı AES ile şifreler ve kurulum paketini gönderir. Şifrelenmiş veriyi küçük paketlere böler. Belirtilen sayıda gönderim turu yaparak paketleri karşı tarafa gönderir. Son olarak "FIN" sinyali göndererek transfer tamamlandığını bildirir.

```
1 # UDP protokolü ile güvenli dosya gönderimi
2 def send_file_udp(self, host, port, packet_size, send_count, filename):
3     print(f"[MODE] UDP - Dosya: {filename}")
4
5     # Dosya varlığını kontrol et
6     if not Path(filename).exists():
7         print(f"[!] Dosya bulunamadı: {filename}")
8         return False
9
10    client_socket = None
11    try:
12        # UDP socket oluştur
13        client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
14
15        # Dosyayı binary moda oku
16        with open(filename, "rb") as f:
17            file_data = f.read()
18
19        print(f"[+] Dosya okundu: {len(file_data)} bytes")
20
21        # Dosyayı şifrele
22        encrypted_data, encrypted_aes_key, iv = self.encrypt_file_data(
23            file_data, Path(filename).name
24        )
25
26        # Kurulum paketini oluştur (şifrelenmiş AES key + IV)
27        setup_packet = encrypted_aes_key + iv
28        client_socket.sendto(setup_packet, (host, port))
29        time.sleep(0.1) # sunucunun kurulum paketini işlemesi için bekle
30
31        print(f"[+] Kurulum paketi gönderildi")
32
33        # Toplam paket sayısını hesapla
34        total_packets = len(encrypted_data) // packet_size + (1 if len(encrypted_data) % packet_size else 0)
35
36        # Belirtilen sayıda gönderim turu yap
37        for send_round in range(send_count):
38            print(f"[*] Gönderim turu: {(send_round + 1)}/{send_count}")
39
40            # Şifrelenmiş veriyi paketler halinde gönder
41            for i in range(0, len(encrypted_data), packet_size):
42                packet_data = encrypted_data[i:i + packet_size] # mevcut paket verisi
43                packet_number = i // packet_size + 1 # paket numarası
44
45                # CRC32 checksum hesapla (veri bütünlüğü için)
46                crc = zlib.crc32(packet_data) & 0xffffffff
47
48                # Paket başlığı oluştur: boyut (4 byte) + CRC (4 byte)
49                header = len(packet_data).to_bytes(4, 'big') + crc.to_bytes(4, 'big')
50
51                # Paket gönder (UDP connectionless)
52                client_socket.sendto(header + packet_data, (host, port))
53
54                # UDP için küçük bekleme (ağ yoğunluğunu önlemek için)
55                time.sleep(0.001) # 1 milisaniye
56
57                # İlerleme göstergesi (her 10 pakette bir veya son pakette)
58                if packet_number % 10 == 0 or packet_number == total_packets:
59                    print(f"[*] İlerleme: {packet_number}/{total_packets} paket gönderildi")
60
61            # Bitirme sinyali gönder (dosya gönderiminin tamamlandığını belirt)
62            time.sleep(0.1)
63            client_socket.sendto(b'FIN', (host, port))
64
65            print(f"[+] UDP ile {filename} dosyası başarıyla gönderildi")
66            return True
67
68        except Exception as e:
69            print(f"[!] UDP gönderim hatası: {e}")
70            return False
71
72    finally:
73        # Socket'i kapat
74        if client_socket:
75            client_socket.close()
```

**send\_file\_hybrid(...)** bu fonksiyon önce ağ gecikmesini ölçer ve belirlenen eşik değerle karşılaştırır. Gecikme düşükse UDP protokolünü tercih eder çünkü daha hızlıdır. Gecikme yüksekse TCP protokolünü kullanır çünkü daha güvenilirdir. Böylece ağ koşullarına göre otomatik olarak en uygun protokolü seçer.

```
1 # Hibrit mod - ağ koşullarına göre TCP veya UDP kullanarak dosya gönderir
2 def send_file_hybrid(self, host, tcp_port, udp_port, packet_size, send_count, filename):
3     print("[MODE] HYBRID - Ağ koşulları analiz ediliyor...")
4
5     # Ağ gecikmesini ölç
6     latency = self.measure_network_latency()
7     print(f"[*] Ölçülen gecikme: {latency:.2f} ms")
8
9     # Gecikme eşigine göre protokol seç
10    if latency < self.config['LATENCY_THRESHOLD']:
11        # Düşük gecikme - UDP daha verimli
12        print(f"[*] Düşük gecikme ({latency:.2f} ms < {self.config['LATENCY_THRESHOLD']} ms) - UDP kullanılıyor")
13        return self.send_file_udp(host, udp_port, packet_size, send_count, filename)
14    else:
15        # Yüksek gecikme - TCP daha güvenilir
16        print(f"[*] Yüksek gecikme ({latency:.2f} ms >= {self.config['LATENCY_THRESHOLD']} ms) - TCP kullanılıyor")
17        return self.send_file_tcp(host, tcp_port, packet_size, send_count, filename)
```

- **network\_analysis.py**

Bu python kodu sayesinde derinlemesine bir ağ analizi yapılmaktadır. Sunucu ve istemci arasında çeşitli ağ testleri yaparak bağlantı kalitesini ölçer ve detaylı bir rapor sunar. Bahsedilen işlemleri yapan çok çeşitli fonksiyonlar vardır. Bu ağ analizi programının işleyişini incelemek gerekirse:

Ağ analizi için kullanılacak varsayılan ayarlar aşağıdaki gibidir.

```
1 # =====
2 # VARSAYILAN AYARLAR
3 # =====
4 HOST = "google.com"                      # ping testi için varsayılan hedef
5 IPERF_SERVER = "127.0.0.1"                # iPerf sunucu adresi (localhost)
6 SOCKET_SERVER_HOST = "127.0.0.1"           # socket sunucu IP adresi
7 SOCKET_SERVER_PORT = 9999                  # socket sunucu port numarası
8 IPERF_CMD = None                          # iPerf komut yolu (dinamik olarak belirlenir)
```

Ağ analizlerini gerçekleştirmek için gerekli yardımcı fonksiyonlar aşağıdaki gibidir.

```
1 # Başlık yazdırma fonksiyonu
2 def print_header(title):
3     print("\n'*50")
4     print(f"[{title}]")
5     print("*50")
```

**print\_header(...)** fonksiyonu ile argüman olarak verilen metin başlık bilgisi olarak yazdırılır.

**check\_dependencies()** fonksiyonuyla beraber ağ analizi sırasında kullanılacak dış kaynakların varlığı kontrol edilir. Örneğin daha önceden anlatılan “server.py” ve “client.py” dosyalarının varlığı veya test için kullanılan iPerf3 dosyası bulunmaya çalışılır. Bulunamazsa o dosyayla ilgili analiz atlanır.

```
1 # Bağımlılıkları kontrol etme fonksiyonu
2 def check_dependencies():
3     print("Bağımlılıklar kontrol ediliyor...")
4
5     # Socket test dosyalarının varlığını kontrol et
6     if not os.path.exists("server.py"):
7         print("server.py bulunamadı!")
8         return False
9     if not os.path.exists("client.py"):
10        print("client.py bulunamadı!")
11        return False
12
13     # iPerf3 aracını farklı konumlarda ara - sistem PATH'i ve yerel dizin
14     global IPERF_CMD
15     iperf_paths = ["./iperf3/iperf3", "iperf3", "iperf"] # olası yollar listesi
16     iperf_found = False
17
18     # Her olası yolu kontrol et
19     for path in iperf_paths:
20         if shutil.which(path) or os.path.exists(path): # sistem PATH'inde veya dosya olarak mevcut mu?
21             IPERF_CMD = path
22             iperf_found = True
23             break
24
25     # iPerf bulunamazsa uyarı ver ama devam et
26     if not iperf_found:
27         print("iPerf3 bulunamadı, bant genişliği testi atlanacak")
28     else:
29         print(f"iPerf3 bulundu: {IPERF_CMD}")
30
31     print("Socket dosyaları bulundu")
32     return True
```

**cleanup\_processes(...)** fonksiyonu test süresince başlatılan tüm sunucu süreçlerini güvenli bir şekilde sonlandırır. Kaynak sizıntısını önlemek ve sistem temizliği için kritik bir fonksiyondur.

```
1 # =====
2 # SÜREÇ TEMİZLEME FONKSİYONU
3 # =====
4 def cleanup_processes(processes):
5     print("\nSüreçler temizleniyor...")
6     for process in processes:
7         if process and process.poll() is None: # süreç hala çalışıyorsa
8             try:
9                 # Önce nazikçe sonlandırmayı dene
10                process.terminate()
11                process.wait(timeout=5) # 5 saniye bekle
12                print("Süreç temizlendi")
13            except subprocess.TimeoutExpired:
14                try:
15                    # Nazik sonlandırma işe yaramazsa zorla öldür
16                    process.kill()
17                    process.wait(timeout=2) # 2 saniye bekle
18                    print("Süreç zorla sonlandırıldı")
19                except:
20                    print("Süreç temizlenemedi")
21            except:
22                print("Süreç temizleme hatası")
```

**print\_summary(...)** fonksiyonu tüm testlerin sonuçlarını tek bir yerde toplar ve detaylı bir şekilde ekrana yazdırır. Ping gecikmeleri, socket test sonuçları ve bant genişliği verilerini organize eder. Test tamamlanma zamanını da ekleyerek kapsamlı bir özet rapor oluşturur.

```

1 # =====
2 # ÖZET RAPOR FONKSIYONU
3 # =====
4 def print_summary(ping_result, socket_result, bandwidth_result):
5     print_header("TEST ÖZETİ")
6
7     # Ping testi sonuçları
8     if ping_result:
9         print(f"Ping Gecikme: {ping_result['avg']:.2f} ms (min: {ping_result['min']:.2f}, max: {ping_result['max']:.2f})")
10        print(f" Paket Kaybı: %{ping_result['packet_loss']:.1f}")
11    else:
12        print("Ping Gecikme: Test başarısız")
13
14     # Socket testi sonuçları
15     if socket_result:
16         print(f"Socket Testleri:")
17         for size, data in socket_result.items():
18             print(f"  {size} byte: {data['avg']:.3f} ms ortalama ({data['count']}/{data.get('count', 0)} başarılı)")
19             print(f"          Min/Max: {data['min']:.3f}/{data['max']:.3f} ms")
20     else:
21         print("Socket Testleri: Test başarısız")
22
23     # Bant genişliği testi sonuçları
24     if bandwidth_result:
25         print(f"Bant Genişliği Testi:")
26         print(f"{bandwidth_result[0]}")
27         print(f"{bandwidth_result[1]}")
28     else:
29         print("Bant Genişliği: Test başarısız veya atlandı")
30
31     # Test tamamlanma zamanı
32     print(f"\nTest tamamlanma zamanı: {time.strftime('%Y-%m-%d %H:%M:%S')}")

```

Ağ analizlerini gerçekleştirecek ana fonksiyonlar aşağıdaki gibidir.

```

1 # =====
2 # ANA PROGRAM FONKSIYONU
3 # =====
4
5 def main():
6     print("Gelişmiş Ağ Performans Analizi Başlatılıyor...")
7     print(f"Sistem: {platform.system()} {platform.release()}")
8
9     # Gerekli dosya ve araçları kontrol et
10    if not check_dependencies():
11        print("Gerekli dosyalar bulunamadı, çıkış...")
12        return 1 # hata kodu ile çıkış
13
14    running_processes = [] # çalışan süreçleri takip et (temizlik için)
15
16    try:
17        # 1. Ağ gecikmesi testi (internet bağlantısı gereklidir)
18        ping_result = measure_latency()
19
20        # 2. Socket sunucusunu başlat ve paket zamanlaması test et
21        socket_server = start_socket_server()
22        if socket_server:
23            running_processes.append(socket_server) # temizlik listesine ekle
24            socket_result = measure_packet_timing()
25        else:
26            socket_result = None
27
28        # 3. iperf bant genişliği testi (localhost üzerinde)
29        iperf_server = start_iperf_server()
30        if iperf_server:
31            running_processes.append(iperf_server) # temizlik listesine ekle
32            bandwidth_result = measure_bandwidth()
33        else:
34            bandwidth_result = None
35
36        # 4. Paket kaybı simülasyonu (su anda kapalı - Linux gerektirir)
37        # simulate_packet_loss()
38
39        # 5. Tüm test sonuçlarının özet raporu
40        print_summary(ping_result, socket_result, bandwidth_result)
41
42    return 0 # başarılı çıkış kodu
43
44 except KeyboardInterrupt:
45     # Kullanıcı Ctrl+C ile durdurulursa
46     print("\nTest kullanıcının tarafından durduruldu")
47     return 1 # hata kodu ile çıkış
48 except Exception as e:
49     # Beklenmeyen hatalar için genel yakalama
50     print(f"\nBeklenmeyen hata: {e}")
51     return 1 # hata kodu ile çıkış
52 finally:
53     # Her durumda çalışan süreçleri temizle (kaynak szintisini önde)
54     cleanup_processes(running_processes)
55     print("\nAğ analizi tamamlandı")
56
57 # =====
58 # PROGRAM GİRİŞ NOKTASI
59 #
60 if __name__ == "__main__":
61     # Ana fonksiyona çalıştırır ve çıkış kodunu al
62     exit_code = main()
63
64     # Sistem çıkış kodunu ayarla (işletim sistemi için)
65     sys.exit(exit_code)

```

**main()** fonksiyonu geri kalan tüm fonksiyonların sırasıyla çağrıldığı ana fonksiyondur. Gerekli ağ analizlerini yapar ve değişkenlerde tutar. Klavyeden “Ctrl + C” girdisi alındığında analizi yarıda keser. Analiz başarıyla tamamlandığında **print\_summary(...)** fonksiyonu ile analiz sonuçlarını ekrana yazdırır.

```

1 # =====
2 # PING GECİKME ÖLÇÜM FONKSİYONU
3 # =====
4 def measure_latency(host=HOST, count=10):
5     print_header("PING GECİKME ÖLÇÜMÜ")
6     print(f"Hedef: {host}")
7     print(f"Paket sayısı: {count}")
8
9     try:
10         # İşletim sistemine göre ping komutunu ayarla
11         if platform.system().lower() == "windows":
12             cmd = ["ping", "-n", str(count), host]    # Windows: -n parametresi
13         else:
14             cmd = ["ping", "-c", str(count), host]    # Linux/macOS: -c parametresi
15
16         print(f"Ping başlatılıyor...")
17         start_time = time.time()                  # toplam süre ölçümü için
18         output = subprocess.check_output(cmd, universal_newlines=True, timeout=30)
19         total_time = time.time() - start_time
20
21         # Ping çıktısından zaman değerlerini çıkar - platform bazlı regex
22         if platform.system().lower() == "windows":
23             times = re.findall(r"time[<](\d{0,3}\.\d{3})ms", output)      # Windows formatı
24             loss_match = re.search(r"\((\d{0,3}\.\d{3})% loss\)", output)    # paket kaybı
25         else:
26             times = re.findall(r"time=(\d{0,3}\.\d{3}) ms", output)       # Unix formatı
27             loss_match = re.search(r"(\d{0,3}\.\d{3})% packet loss", output)
28
29         # String değerleri sayısal değerlere çevir
30         times = [float(t) for t in times]
31         packet_loss = float(loss_match.group(1)) if loss_match else 0
32
33         if times:
34             # İstatistiksel değerleri hesapla
35             min_rtt = min(times)                                # minimum gecikme
36             max_rtt = max(times)                                # maksimum gecikme
37             avg_rtt = statistics.mean(times)                   # ortalama gecikme
38             median_rtt = statistics.median(times)            # medyan gecikme
39             stddev_rtt = statistics.stdev(times) if len(times) > 1 else 0 # standart sapma
40
41             # Sonuçları detaylı şekilde yazdır
42             print(f"\nPING SONUCLARI:")
43             print(f"  Gönderilen: {count} paket")
44             print(f"  Alınan: {len(times)} paket")
45             print(f"  Kayıp: %{packet_loss:.1f}")
46             print(f"  Min RTT: {min_rtt:.2f} ms")
47             print(f"  Max RTT: {max_rtt:.2f} ms")
48             print(f"  Ortalama RTT: {avg_rtt:.2f} ms")
49             print(f"  Medyan RTT: {median_rtt:.2f} ms")
50             print(f"  Standart Sapma: {stddev_rtt:.2f} ms")
51             print(f"  Toplam süre: {total_time:.2f} saniye")
52
53             # Sonuçları dictionary olarak döndür (diğer fonksiyonlarda kullanım için)
54             return {
55                 'min': min_rtt, 'max': max_rtt, 'avg': avg_rtt,
56                 'median': median_rtt, 'stddev': stddev_rtt,
57                 'packet_loss': packet_loss, 'total_time': total_time
58             }
59         else:
60             print("Ping sonuçlarında zaman bilgisi bulunamadı")
61             return None
62
63     except subprocess.TimeoutExpired:
64         # Timeout durumu - hedef ulaşılamaz veya çok yavaş
65         print("Ping timeout - hedef yanıt vermiyor")
66         return None
67     except subprocess.CalledProcessError as e:
68         # Ping komutu başarısız - ağ hatası veya geçersiz hedef
69         print(f"Ping komutu başarısız: {e}")
70         return None
71     except Exception as e:
72         # Beklenmeyen hatalar için genel yakalama
73         print(f"Gecikme ölçümünde hata: {e}")
74         return None

```

## measure\_latency(...)

Ping komutuyla belirtilen hedef adres'e paket göndererek gecikme süresini ölçer. İşletim sistemine göre uygun ping parametrelerini kullanır ve sonuçları regex ile parse eder. Min, max, ortalama, medyan RTT değerlerini hesaplayarak paket kaybı oranıyla birlikte detaylı istatistik sunar.

**start\_socket\_server()** fonksiyonu yerel bilgisayarda socket sunucusunu başlatmak için server.py dosyasını ayrı bir Python süreci olarak çalıştırır. Sunucunun düzgün başlayıp başlamadığını kontrol eder ve process handle'ını döndürür. Bu sunucu, paket zamanlama testleri için gerekli olan TCP bağlantısını sağlar.

```

1 # =====
2 # SOCKET SUNUCU YÖNETİM FONKSİYONU
3 # =====
4 def start_socket_server():
5     print_header("SOCKET SUNUCUSU")
6     try:
7         print("Socket sunucusu başlatılıyor...")
8         # Ayrı bir Python süreci olarak sunucuyu başlat
9         process = subprocess.Popen([
10             sys.executable, "server.py"           # mevcut Python interpreter'i kullan
11         ], stdout=subprocess.PIPE, stderr=subprocess.PIPE)
12
13         time.sleep(2) # Sunucunun tamamen başlaması için bekleme süresi
14
15         # Sunucunun düzgün çalışıp çalışmadığını kontrol et
16         if process.poll() is None: # None = hala çalışıyor
17             print("Socket sunucusu başarıyla başlatıldı")
18             return process # process handle'ını döndür (sonra kapatmak için)
19         else:
20             # Sunucu başlatılamadıysa hata mesajını al
21             _, stderr = process.communicate()
22             print(f"Socket sunucusu başlatılamadı: {stderr.decode()}")
23             return None
24
25     except Exception as e:
26         print(f"Socket sunucusu hatası: {e}")
27         return None

```

```

1 # =====
2 # PAKET ZAMANLAMA TEST FONKSİYONU
3 # =====
4 def measure_packet_timing():
5     print_header("PAKET ZAMANLAMA TESTİ")
6
7     packet_sizes = [64, 512, 1024] # Farklı paket boyutları (byte) - küçük/orta/büyük
8     results = {} # test sonuçlarını sakla
9     test_count = 5 # Her boyut için tekrar sayısı (güvenilirlik için)
10
11     # Her paket boyutu için ayrı test döngüsü
12     for size in packet_sizes:
13         print(f"\n{size} byte paket testi ({test_count} test):")
14         times = [] # bu boyut için ölçülen süreler
15
16         try:
17             # Belirtilen sayıda test tekrarı yap
18             for i in range(test_count):
19                 # Her paket gönderimini için hassas zamanlama
20                 start_time = time.time()
21
22                 # Client'lı tek paket göndermek için çalıştır
23                 result = subprocess.run([
24                     sys.executable, "client.py"
25                 ], capture_output=True, text=True, timeout=10)
26
27                 end_time = time.time()
28
29                 # Client başarılı olursa süreyi kaydet
30                 if result.returncode == 0:
31                     # Paket gönderme süresini milisaniye cinsinden hesapla
32                     packet_time = (end_time - start_time) * 1000
33                     times.append(packet_time)
34                     print(f" Test {i+1}: {packet_time:.3f} ms")
35                 else:
36                     print(f" Test {i+1}: Başarısız - {result.stderr.strip()}")
37
38             # Başarılı testlerin istatistiklerini hesapla ve kaydet
39             if times:
40                 results[size] = {
41                     'times': times, # ham veriler
42                     'min': min(times), # minimum süre
43                     'max': max(times), # maksimum süre
44                     'avg': statistics.mean(times), # ortalama süre
45                     'median': statistics.median(times), # median süre
46                     'stdev': statistics.stdev(times) if len(times) > 1 else 0, # standart sapma
47                     'count': len(times), # başarılı test sayısı
48                     'success_rate': len(times) / test_count * 100 # başarılı oranı
49                 }
50
51             # Bu boyut için özet istatistikleri yazdır
52             print("\n Sonuçlar:")
53             print(f" Min/Avg/Max: {(min(times):.3f)}/{(statistics.mean(times):.3f)}/{(max(times):.3f)} ms")
54             print(f" Medyan: {(statistics.median(times):.3f)} ms")
55             print(f" Std Sapma: {(statistics.stdev(times) if len(times) > 1 else 0:.3f)} ms")
56             print(f" Başarı Oranı: %{(len(times) / test_count) * 100:.1f}%")
57
58         else:
59             print(" Hiçbir test başarılı olmadı")
60
61     except Exception as e:
62         print(f"{size} byte test hatası: {e}")
63
64     return results # tüm boyutlar için sonuçları döndür

```

## measure\_packet\_timing()

fonksiyonu farklı boyutlarda paket göndererek ağ zamanlamasını test eder. Her paket boyutu için 5 test yaparak hassas zamanlama ölçümleri alır. Her boyut için min, max, ortalama değerleri hesaplayarak başarı oranını da raporlar.

`start_iperf_server()` fonksiyonu bant genişliği testleri için iPerf3 sunucusunu başlatır.

```
1 # =====
2 # IPERF SUNUCU YÖNETİM FONKSİYONU
3 # =====
4 def start_iperf_server():
5     if IPERF_CMD is None: # iPerf bulunamadıysa işlem yapma
6         return None
7
8     print_header("IPERF3 SUNUCUSU")
9     try:
10         print("iPerf3 sunucusu başlatılıyor...")
11         # iPerf sunucusunu başlat (-s = server modu, -p = port)
12         process = subprocess.Popen([
13             IPERF_CMD, "-s", "-p", "5201"
14         ], stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL) # çıktıları gizle
15
16         time.sleep(3) # Sunucunun tamamen başlaması için yeterli bekleme
17         print("iPerf3 sunucusu başlatıldı")
18         return process # process handle'ını döndür
19
20     except Exception as e:
21         print(f"iPerf3 sunucusu hatası: {e}")
22         return None
```

`measure_bandwidth()` fonksiyonu iPerf3 client modunda çalışarak localhost'a 10 saniyelik

```
1 # =====
2 # BANT GENİŞLİĞİ ÖLÇÜM FONKSİYONU
3 # =====
4 def measure_bandwidth():
5     if IPERF_CMD is None: # iPerf yoksa test yapılamaz
6         print("iPerf3 bulunamadı, bant genişliği testi atlandı")
7         return None
8
9     print_header("BANT GENİŞLİĞİ TESTİ")
10
11    try:
12        print("Bant genişliği ölçülüyor (10 saniye)...")
13        # iPerf client modunda çalıştır (-c = client, -t = süre, -f = format)
14        result = subprocess.run([
15            IPERF_CMD, "-c", IPERF_SERVER, "-t", "10", "-f", "M" # 10 saniye, MBytes formatı
16        ], capture_output=True, text=True, timeout=30)
17
18        if result.returncode == 0:
19            print("Bant genişliği testi tamamlandı")
20            print("\nSONUÇLAR:")
21
22            # iPerf çıktısını filtrele ve önemli satırları göster
23            lines = result.stdout.split("\n")
24            for line in lines:
25                # Sadece bant genişliği bilgisi içeren satırları göster
26                if ('Mbps/sec' in line or 'sender' in line or 'receiver' in line) and line.strip().startswith('['):
27                    print(f" {line.strip()}")
28
29            # Özeti bilgiyi parse et ve daha okunabilir hale getir
30            summary_lines = [line for line in lines if 'sender' in line or 'receiver' in line]
31            results = [] # Sonuçları saklamaK için liste
32
33            if summary_lines:
34                print(f"\nBant genişliği sonucu:")
35                # Son iki satır genellikle sender/receiver özetleridir
36                for line in summary_lines[-2:]:
37                    if 'Mbps/sec' in line:
38                        parts = line.split()
39                        # Hız değerini bul ve ayıkla
40                        for i, part in enumerate(parts):
41                            if 'Mbps/sec' in part and i > 0:
42                                try:
43                                    speed = parts[i-1]
44                                    direction = 'Gönderme' if 'sender' in line else 'Alma'
45                                    result_text = f"(direction): {speed} Mbps/sn"
46                                    print(f" {result_text}")
47                                    results.append(result_text) # Sonucu listeye ekle
48                                break
49                            except IndexError:
50                                continue
51
52            return results # sonuçları döndür
53        else:
54            print(f"Bant genişliği testi başarısız:")
55            print(result.stderr)
56            return None
57
58    except subprocess.TimeoutExpired:
59        # Test çok uzun sürerse timeout
60        print("Bant genişliği testi timeout")
61        return None
62
63    except Exception as e:
64        print(f"Bant genişliği testi hatası: {e}")
65        return None
```

bant genişliği testi yapar. Sonuçları MBytes/saniye formatında alır ve gönderme/alma hızlarını ayrı ayrı raporlar. iPerf çiktısını ekranaya yazdırır.

**simulate\_packet\_loss()** fonksiyonu tc (traffic control) komutuyla %10 paket kaybı simüle eder. Localhost üzerinde paket kaybının etkilerini test eder ve sonrasında ağ ayarlarını eski haline getirir.

```
# =====
# PAKET KAYBI SİMÜLASYON FONKSİYONU
# =====
def simulate_packet_loss():
    if platform.system().lower() != "linux":
        print("\nPaket kaybı simülasyonu sadece Linux'ta desteklenir")
        return

    print_header("PAKET KAYBI SİMÜLASYONU")

    try:
        print("⚡ %10 paket kaybı simüle ediliyor...")

        # Linux tc (traffic control) komutuyla paket kaybı ekle
        subprocess.run([
            "sudo", "tc", "qdisc", "add", "dev", "lo", "root", "netem", "loss", "10%"
        ], check=True)

        print("Paket kaybı simülasyonu aktif")
        print("5 saniye test süresi...")

        # Paket kaybı ile localhost'a ping testi yap
        test_result = measure_latency(host="127.0.0.1", count=5)

        time.sleep(5) # Simülasyonun etkisini gözlemlemek için bekle

        # tc kuralını temizle (ağ ayarlarını eski haline getir)
        subprocess.run([
            "sudo", "tc", "qdisc", "del", "dev", "lo", "root"
        ], check=True)

        print("Paket kaybı simülasyonu temizlendi")
        return test_result

    except subprocess.CalledProcessError as e:
        print(f"tc komutu başarısız: {e}")
        print("sudo yetkisi gereklidir veya tc kurulu olmayıabilir")
        return None
    except Exception as e:
        print(f"Simülasyon hatası: {e}")
        return None
```

- **mitm\_proxy.py**

Bu kod, MITM (Man-in-the-Middle) Proxy sunucusu açar. İstemci ile hedef sunucu arasındaki tüm trafiği yakalar, loglar ve ileter. Bu MITM programı MITMProxy sınıfından oluşmaktadır. Bu sınıf içerisinde pek çok fonksiyon bulunmaktadır. Bunlar sırasıyla:

`__init__(...)` fonksiyonu sınıfın yapıcı fonksiyonudur ve sınıf içindeki değişkenleri tanımlar ve ayarlamaları yapar.

```
1 def __init__(self, listen_port=8080, target_host='127.0.0.1', target_port=5001):
2     self.listen_port = listen_port          # proxy'nin dinleyeceği port
3     self.target_host = target_host          # hedef sunucunun IP adresi
4     self.target_port = target_port          # hedef sunucunun port numarası
5     self.intercepted_data = []             # yakalanan veri listesi (opsiyonel)
```

`log_traffic(...)` fonksiyonu proxy sunucu üzerinden geçen tüm ağ trafiğini detaylı şekilde loglar ve analiz eder.

```
1 # Ağ trafiğini loglamak için yardımcı fonksiyon
2 def log_traffic(self, direction, data, client_addr=None):
3     # Zaman damgası oluştur (millisaniye hassasiyetinde)
4     timestamp = datetime.now().strftime("%H:%M:%S.%f")[:-3]
5
6     # Client adresi varsa dahil et, yoksa sadece yön ve veri boyutu göster
7     if client_addr:
8         print(f"[{timestamp}] {direction} from {client_addr}: {len(data)} bytes")
9     else:
10        print(f"[{timestamp}] {direction}: {len(data)} bytes")
11
12    # Verinin ilk 100 byte'ını hexadecimal formatında göster
13    hex_data = data[:100].hex()
14    print(f"        Data (hex): {hex_data}")
15
16    # Veriyi UTF-8 text olarak decode etmeye çalış
17    try:
18        text_data = data.decode('utf-8', errors='ignore')[:100] # ilk 100 karakter
19        if text_data.strip(): # boş değilse göster
20            print(f"        Data (text): {text_data}")
21    except:
22        # Decode hatası durumunda sessizce geç
23        pass
24
25    # Ayırıcı çizgi ekle (okunabilirlik için)
26    print("-" * 60)
```

`handle_client(...)` fonksiyonu her client bağlantısı için ayrı bir thread'de çalışan ana bağlantı

yönetim fonksiyonudur.  
MITM köprüsünü kurar.

```
1 # Client bağlantısını isleyen ana fonksiyon
2 def handle_client(self, client_socket, client_addr):
3     server_socket = None
4     try:
5         # Hedef sunucuya bağlantı oluştur
6         server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
7         server_socket.connect((self.target_host, self.target_port))
8
9         # Başarılı MITM bağlantısı mesajı
10        print("[+] MITM bağlantısı kuruldu: {client_addr} <-> Proxy <-> {self.target_host}:{self.target_port}")
11
12        # İki yönlü veri aktarımı için thread'ler oluştur
13        client_to_server = threading.Thread(
14            target=self.relay_data,                                # çalıştırılacak fonksiyon
15            args=(client_socket, server_socket, "CLIENT->SERVER", client_addr) # fonksiyon parametreleri
16        )
17        server_to_client = threading.Thread(
18            target=self.relay_data,                                # çalıştırılacak fonksiyon
19            args=(server_socket, client_socket, "SERVER->CLIENT", client_addr) # fonksiyon parametreleri
20        )
21
22        # Thread'leri daemon olarak ayarla (ana program kapandığında otomatik kapanın)
23        client_to_server.daemon = True
24        server_to_client.daemon = True
25
26        # Her iki thread'i de başlat
27        client_to_server.start()
28        server_to_client.start()
29
30        # Thread'lerin bitmesini bekle (bağlantı kesilene kadar)
31        client_to_server.join()
32        server_to_client.join()
33
34    except Exception as e:
35        # Bağlantı hataları için log kaydet
36        print(f"[!] MITM proxy hatası: {e}")
37    finally:
38        # Her durumda socket'leri güvenli şekilde kapat
39        try:
40            if client_socket:
41                client_socket.close()
42            if server_socket:
43                server_socket.close()
44        except:
45            pass # kapatma hatalarını yoksay
```

**relay\_data(...)** fonksiyonu iki socket arasında sürekli veri aktarımı yapar. Veriyi aktarırken de aynı anda kaydeder.

```
1 # İki socket arasında veri aktarımı yapan fonksiyon
2 def relay_data(self, source, destination, direction, client_addr):
3     try:
4         # Sürekli veri aktarım döngüsü
5         while True:
6             # Kaynaktan veri al (maksimum 4096 byte)
7             data = source.recv(4096)
8
9             # Veri yoksa bağlantı kesilmiş demektir
10            if not data:
11                break
12
13            # Yakalanan trafiği logla
14            self.log_traffic(direction, data, client_addr)
15
16            # Değiştirilmiş (veya orijinal) veriyi hedef tarafa gönder
17            destination.send(data)
18
19    except Exception as e:
20        # Veri aktarım hatalarını logla
21        print(f"[!] Relay hatası ({direction}): {e}")
```

**start\_proxy(...)** fonksiyonu MITM Proxy sunucusunu başlatır ve TCP protokolünde bir soket oluşturur. Her bir istemci bağlantısında bir thread açar ve bu sayede birden çok istemci bağlanabilir.

```
1 # MITM Proxy'yi başlatan ana fonksiyon
2 def start_proxy(self):
3     # TCP socket oluştur
4     proxy_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
5
6     # Socket ayarları: adresi yeniden kullanmaya izin ver
7     proxy_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
8
9     # Tüm network interface'lerden gelen bağlantıları kabul et
10    proxy_socket.bind(('0.0.0.0', self.listen_port))
11
12    # Maksimum 5 bekleyen bağlantıya izin ver
13    proxy_socket.listen(5)
14
15    # Başlatma mesajları
16    print(f"[+] MITM Proxy başlatıldı: 0.0.0.0:{self.listen_port}")
17    print(f"[*] Hedef sunucu: {self.target_host}:{self.target_port}")
18    print(f"[*] Client'ları {self.listen_port} portuna yönlendirin")
19    print("-" * 60)
20
21    try:
22        # Ana bağlantı kabul döngüsü
23        while True:
24            # Yeni client bağlantısını kabul et
25            client_socket, client_addr = proxy_socket.accept()
26            print(f"[+] Yeni bağlantı: {client_addr}")
27
28            # Her client için ayrı thread oluştur (paralel işlem için)
29            client_thread = threading.Thread(
30                target=self.handle_client,           # çalıştırılacak fonksiyon
31                args=(client_socket, client_addr)   # fonksiyon parametreleri
32            )
33            client_thread.daemon = True          # daemon thread olarak ayarla
34            client_thread.start()               # thread'i başlat
35
36    except KeyboardInterrupt:
37        # Ctrl+C ile durdurulduğunda temizlik yap
38        print("\n[!] MITM Proxy durduruldu.")
39    finally:
40        # Proxy socket'ini kapat
41        proxy_socket.close()
```

**main()** fonksiyonu programın çalıştırıldığı yerdir. Proxy sunucusu gerekli ayarlar ile

başlatılır.

```
1 # =====
2 # ANA PROGRAM FONKSİYONU
3 # =====
4 if __name__ == "__main__":
5     # Program tanıtım mesajları
6     print("== MITM PROXY ==")
7
8     # MITM proxy nesnesini oluştur
9     mitm = MITMProxy(
10         listen_port=8080,           # Client'ların bağlanacağı proxy portu
11         target_host='127.0.0.1',   # Gerçek sunucunun IP adresi (localhost)
12         target_port=5001          # Gerçek sunucunun port numarası
13     )
14
15     # Proxy'yi başlat
16     mitm.start_proxy()
```

- **packet\_injection.py**

Bu kod, RSA ve AES şifrelemeli sahte dosya enjeksiyon programıdır. UDP sunucusuna şifreli sahte dosya göndererek güvenlik testleri yapmaktadır.

UDP sunucusunun bilgileri aşağıdaki sekildedir.

```
1 # =====
2 # SUNUCU BİLGİLERİ
3 # =====
4 IP = '127.0.0.1'
5 PORT = 5002
```

Paket enjeksiyonunda kullanılması için anahtarın yüklenmesi gereklidir. Bu anahtarın kötücül yollarla ele geçirildiği varsayılmaktadır.

```
1 # =====
2 # RSA GENEL ANAHTARINI YÜKLEME
3 # =====
4 def load_server_pubkey():
5     with open("../keys/public.pem", "rb") as f:
6         return RSA.import_key(f.read())
```

**main()** fonksiyonu programın başlangıç noktasıdır. Burada `send_fake_session()` fonksiyonu çağırılmaktadır.

```
1 # =====
2 # PROGRAMIN BAŞLANGIÇ NOKTASI
3 # =====
4 if __name__ == "__main__":
5     # Sahte oturumu başlat
6     send_fake_session()
```

`send_fake_session()` fonksiyonu sahte dosya gönderme işleminin tüm aşamalarını koordine eden ana saldırısı fonksiyonudur. UDP soketi oluşturur, şifreleme anahtarları üretir ve çok aşamalı bir protokol izleyerek sunucuya sahte dosya enjekte eder. Kurulum, veri gönderimi ve oturum sonlandırma olmak üzere üç ana aşamada çalışır.

```
1  # =====
2 # SAHTE OTURUM GÖNDERME FONKSİYONU
3 # =====
4 def send_fake_session():
5     # UDP soketi oluştur
6     sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
7
8     # Rastgele AES anahtarı ve IV (Initialization Vector) oluştur
9     aes_key = get_random_bytes(16)
10    iv = get_random_bytes(16)
11
12    # RSA genel anahtarını yükle ve AES anahtarını şifrele
13    rsa_key = load_server_pubkey()
14    rsa_cipher = PKCS1_OAEP.new(rsa_key)
15    encrypted_aes_key = rsa_cipher.encrypt(aes_key)
16
17    # Kurulum paketi: encrypted_aes_key (256 byte) + iv (16 byte)
18    setup_packet = encrypted_aes_key + iv
19    sock.sendto(setup_packet, (IP, PORT))
20    print("[*] Kurulum paketi gönderildi.")
21
22    # Sunucunun kurulum paketini işlemesi için bekleme
23    time.sleep(0.05)
24
25    # Sahte dosya içeriği ve bilgileri (güçellenmiş formata uygun)
26    fake_file_content = b"ENJEKTE_EDILMIS_DOSYA_ICERIGI"
27    fake_filename = "injected_packet.txt"
28
29    # Dosya hash'ini hesapla
30    file_hash = hashlib.sha256(fake_file_content).digest() # 32 byte
31
32    # Dosya adı bilgilerini hazırla
33    filename_bytes = fake_filename.encode('utf-8')
34    filename_length = len(filename_bytes).to_bytes(2, 'big') # 2 byte uzunluk
35
36    # Güncellenmiş format: hash(32) + filename_length(2) + filename + file_content
37    data_with_info = file_hash + filename_length + filename_bytes + fake_file_content
38
39    # Veriyi şifrele
40    padded_payload = pad(data_with_info, AES.block_size)
41    encrypted_data = AES.new(aes_key, AES.MODE_CBC, iv).encrypt(padded_payload)
42
43    # Veri paketini hazırla: size (4 byte) + crc (4 byte) + encrypted_data
44    size = len(encrypted_data).to_bytes(4, 'big')
45    crc = zlib.crc32(encrypted_data).to_bytes(4, 'big')
46    packet = size + crc + encrypted_data
47
48    # Veri paketini gönder
49    sock.sendto(packet, (IP, PORT))
50    print(f"[*] Sahte dosya paketi gönderildi: {fake_filename}")
51
52    # Oturumu sonlandır
53    time.sleep(0.1)
54    sock.sendto(b'FIN', (IP, PORT))
55    print("[*] Sahte oturum sonlandırıldı.")
56
57    sock.close()
```

## 5. “/scapy” Klasörü

Bu klasörün içindeki python kodları, IP paket fragmentasyonu (bölünme) ve yeniden birleştirme işlemlerini test eder. `sender_scapy.py` dosyası bir UDP mesajını küçük IP fragmentlarına bölgerek yerel ağ üzerinden gönderir. `receiver_scapy.py` dosyası ise gelen IP fragmentlarını yakalar, belirli bir ID'ye sahip olanları filtreler ve bu fragmentları doğru sırada birleştirerek orijinal mesajı yeniden oluşturur. Scapy kütüphanesi kullanılarak düşük seviyeli ağ paket manipülasyonu bu kısımda yapılır.

- `receiver.py`

Bu kod scapy ile `sender_scapy.py` kodu tarafından yollanan paketi yakalar ve işler. İçerisinde çeşitli fonksiyonlar barındırmaktadır. Bunlar sırasıyla:

`extract_text_from_combined(...)` bu fonksiyon argüman olarak aldığı veriyi işler ve içerisindeki metni aşağı çıkarır.

```
1 def extract_text_from_combined(data):
2     for i in range(len(data)):
3         try:
4             return data[i:].decode("utf-8")
5         except UnicodeDecodeError:
6             continue
7     return "[!] Metin çözülemedi."
```

`process_packet(...)` fonksiyonu argüman olarak aldığı paketin ID'si beklediği ID ile aynı mı diye kontrol eder. Aynı değilse paketi işlemez. Aynıysa paketin bilgilerini ekrana yazdırır.

```
1 # Her paketi işle
2 def process_packet(pkt):
3     if pkt.haslayer(IP):
4         ip = pkt[IP]
5         if ip.id != TARGET_ID:
6             return # Başka paketleri geç
7         ident = ip.id
8         offset = ip.frag
9         mf_flag = ip.flags.MF
10        length = ip.len
11        print(f"[+] IP Packet: ID={ident}, Offset={offset}, MF={mf_flag}, Len={length}, Checksum={ip.chksum}")
12        if ident not in fragment_data:
13            fragment_data[ident] = []
14            fragment_data[ident].append((offset, bytes(ip.payload)))
```

**analyze.fragments(...)** fonksiyonu spesifik bir ID'yi dinler. Bu ID'ye sahip paketler geldiğinde bunları birleştirir ve tek bir veri elde eder. Ardından bu veriyi **extract\_from\_data\_combined(...)** fonksiyonuyla metin haline getirip ekrana yazdırır.

```
1 # Sniff işlemini başlat ve fragment'lari birleştir
2 def analyze_fragments(timeout=5):
3     print("[*] Sadece ID=12345 olan IP fragmentlar dinleniyor...")
4     try:
5         sniff(prn=process_packet, timeout=timeout, store=False, iface="\\Device\\NPF_Loopback")
6     except PermissionError:
7         print("[] Yönetici olarak çalıştırılmalıdır!")
8
9     print("\n[*] Fragment birleştirme sonucu:")
10    for ident, frags in fragment_data.items():
11        sorted frags = sorted(frags, key=lambda x: x[0])
12        combined = b"".join(frag[1] for frag in sorted frags)
13        print(f"[√] ID={ident} - {len(frags)} fragment birleştirildi - Toplam boyut: {len(combined)} byte")
14
15    print("[!] Mesaj içeriği:", extract_text_from_combined(combined))
```

- **sender\_scapy.py**

Bu kod ile scapy üzerinden aşağıdaki bilgilere sahip alıcıya paketler gönderilir.

```
1 DST_IP = "127.0.0.1" # Alıcı IP
2 DST_PORT = 5001          # Hedef port
```

**create\_and\_fragment\_packet()** fonksiyonuyla gönderilecek verinin içeriği ve başlık bilgileri hazırlanır. Ardından veri paketlere bölünür.

```
1 # IP/UDP paketi oluştur ve fragment'lara ayı
2 def create_and_fragment_packet():
3     data = b"Bu, IP fragment testi için gönderilen bir mesajdır." * 2
4     ip = IP(dst=DST_IP, id=12345, flags=1, ttl=64) # TTL, ID ve MF flag ayarlandı
5     udp = UDP(sport=40000, dport=DST_PORT)
6     pkt = ip / udp / Raw(load=data)
7     pkt = IP(bytes(pkt)) # checksum hesapla
8     fragments = fragment(pkt, fragsize=24) # 24 baytlik fragment'lara böl
9     return fragments
```

**send\_fragments()** fonksiyonuyla da **create\_and\_fragment\_packet()** fonksiyonu kullanılarak paketler oluşturulup ilgili alıcıya gönderilir.

```
1 # Fragment'lari gönder
2 def send_fragments():
3     frags = create_and_fragment_packet()
4     print(f"[*] {len(frags)} fragment gönderiliyor...")
5     for i, frag in enumerate(frags):
6         print(f"> [!] Fragment {i+1}: Offset={frag[IP].frag}, MF={frag[IP].flags.MF}, TTL={frag[IP].ttl}, Checksum={frag[IP].chksum}")
7         send(frag, verbose=0)
8         time.sleep(0.05)
```

## LIMITATIONS AND IMPROVEMENTS

### 1. Geliştirmeler

Proje geliştirmesinde zorunlu olmayan “Hybrid TCP/UDP Switching” ve “Graphical User Interface (GUI)” özelliği bu projede bulunmaktadır. Ağ durumuna bağlı olarak protokol seçilmektedir. Aynı zamanda proje kullanıcı dostu tasarıma sahip arayüz ile kolayca kullanılabilir hale getirilmiştir.

### 2. Kısıtlamalar

Projedeki en önemli eksikliklerden biri Scapy kütüphanesiyle yapılan başlık manipülasyonu özelliğinin programın soket ile çalışan kısmına entegre olmamasıdır. Yani scapy ve soket kısımları birbirinden bağımsız çalışmaktadır. Ayrıca hibrit modda protokol seçimi için yapay zeka temelli ağ durumu analizi entegre edilebilir. Bunun dışında program oldukça stabil ve sorunsuz şekilde çalışmaktadır.

## CONCLUSION

Bu proje sadece teorik anlamda güvenli veri aktarımı değil, aynı zamanda pratik saldırılara karşı nasıl bir direnç sağladığını göstermektedir. Özellikle MITM ve sahte paket saldırularına karşı sistemin başarısı, kullanılan şifreleme algoritmalarının güvenilirliğini desteklemektedir.

- [Projenin kaynak kodlarının bulunduğu GitHub deposu](#)
- [Projenin anlatıldığı YouTube videosu](#)

Bu proje şu özellikleri içermektedir:

- Dosya gönderimini ve alımını; manuel paket parçalamayı ve birleştirmeyi; hata tespiti ve hataları düzeltmeyi içeren bir güvenli dosya transfer sistemini
- AES/RSA şifrelemeyi kullanan, istemcinin bilmesi gereken parola koruması olan, dosya bütünlüğünü SHA-256 ile garanti eden güvenlik sistemini
- Elle IP başlığı değiştirmeyi, veri gönderiminden önce IP checksum bilgisinin hesaplanması, alıcı tarafında paket birleştirmeyi analiz etmeyi sağlayan düşük seviyeli IP başlık manipülasyonunu (Bağımsız bir Scapy klasörünün altında)
- Ağ gecikmesini, ağ genişliğini, paket kaybı ve ağ tıkanıklığını test eden ağ performans ölçümünü

- Wireshark ile paketleri yakalayıp analiz eden ve Man-in-the-middle (MITM) ve paket enjeksiyonu saldırısını simüle edip şifreleme ile yakalanan paketlerin okunamayacağını garanti eden güvenlik analizi ve saldırı simülasyonlarını

## REFERENCES

Python Software Foundation. (t.y.). *Python socket kütüphanesi dokümantasyonu*. Python. 27 Nisan 2025 tarihinde <https://docs.python.org/3/library/socket.html> adresinden erişildi.

Python Software Foundation. (t.y.). *Python zlib kütüphanesi dokümantasyonu*. Python. 27 Nisan 2025 tarihinde <https://docs.python.org/3/library/zlib.html> adresinden erişildi.

PyPI. (t.y.). *PyCrypto*. PyPI. 27 Nisan 2025 tarihinde <https://pypi.org/project/pycrypto/> adresinden erişildi.

GeeksforGeeks. (t.y.). *AES ve RSA şifreleme arasındaki farklar*. GeeksforGeeks. 27 Nisan 2025 tarihinde <https://www.geeksforgeeks.org/difference-between-aes-and-rsa-encryption/> adresinden erişildi.

Wikipedia. (t.y.). *Başlatma vektörü*. Wikipedia. 27 Nisan 2025 tarihinde [https://en.wikipedia.org/wiki/Initialization\\_vector](https://en.wikipedia.org/wiki/Initialization_vector) adresinden erişildi.

Wikipedia. (t.y.). *Dönüştümlü fazlalık kontrolü (CRC)*. Wikipedia. 27 Nisan 2025 tarihinde [https://en.wikipedia.org/wiki/Cyclic\\_redundancy\\_check](https://en.wikipedia.org/wiki/Cyclic_redundancy_check) adresinden erişildi.

GeeksforGeeks. (t.y.). *How to Capture UDP Packets in Python*. 5 Haziran 2025 tarihinde <https://www.geeksforgeeks.org/how-to-capture-udp-packets-in-python/> adresinden erişildi.

GeeksforGeeks. (t.y.). *Socket Programming in Python*. 5 Haziran 2025 tarihinde <https://www.geeksforgeeks.org/socket-programming-python/> adresinden erişildi.

Scapy. (t.y.). 5 Haziran 2025 tarihinde <https://scapy.readthedocs.io/en/latest/> adresinden erişildi.

Wireshark. (t.y.). *Documentation*. 5 Haziran 2025 tarihinde <https://www.wireshark.org/docs/> adresinden erişildi.