Bursa Teknik Üniversitesi

Bilgisayar Ağları

Low-Level IP İşleme ve Ağ Performans Analizi ile Gelişmiş Güvenli Dosya Transfer Sistemi

Proje Ara Raporu

Ad: Eren

Soyad: KÖSE

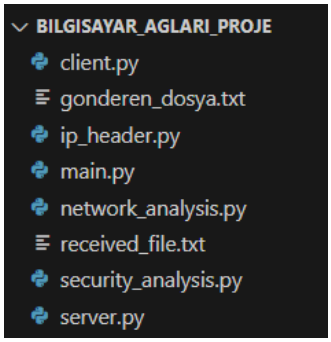
Öğrenci Numarası: 22360859075

Introduction - Tanıtım

- Bu projede, şifreleme, kimlik doğrulama ve bütünlük kontrolü sağlayan gelişmiş bir dosya transfer sistemi geliştirilmekte. Sistem, IP başlıklarının düşük seviyeli işlenmesi (TTL, bayraklar, parçalara ayırma, checksum) yoluyla ağ protokollerine derinlemesine bir bakış sunmayı hedeflemektedir. Aynı zamanda, ağ performansı (gecikme, bant genişliği, paket kaybı) çeşitli senaryolar altında analiz edilmekte ve güvenlik testleri (MITM saldırıları, paket enjeksiyonu) gerçekleştirilmektedir. Python (Scapy) tabanlı bu sistem, güvenli iletişim ve performans analizi konuları deneyimi amaçlamaktadır.

Technical Details – Teknik Detaylar

- Yazılım Dili → Python
- Şifreleme Kütüphaneleri → cryptography
- Diğer Kütüphaneler → Scapy, socket, secrets, subprocess
- Ağ Analiz Araçları → ping, iperf3, Wireshark
- Paket Manipülasyonu → scapy
- Dosya Sistemi →



- Sunucuyu başlatmak için "python main.py server" komutunu terminale giriyoruz. Bu bizim TCP[1] sunucumuzu başlatıyor, gelen bağlantılara RSA[2] public anahtarını gönderiyor, AES[3] anahtarını çözüyor.

```
1  def start_server():
2      server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
3      server_socket.bind(('0.0.0.0', 9999))
4      server_socket.listen(5)
5      print("Sunucu başlatıldı, bağlantı bekleniyor...")
```

  - start_server fonksiyonu sunucuyu başlatır, bağlantı bekler, istemciden veri alır ve dosyayı deşifre ederek kaydeder. TCP socket açılır ve 9999 numaralı port üzerinden dinlenir, 0.0.0.0 ile her IP'yi kabul eder. Çok sayıda bağlantı durumu yaşanmaması amacıyla da 5 adet IP dinlemesine izin vermesi sağlanır. Ayrıca diğer anlatılacak olan fonksiyonlar da bu fonksiyonun içinde çağırılmaktadır.

-
```python
# İstemciye RSA public key gönderme
pem = public_key.public_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PublicFormat.SubjectPublicKeyInfo
)
client_socket.send(pem)
```

Bu kısım start_server fonksiyonu içinde RSA public keyini client tarafına gönderdiğimiz kısımdır.

-
```python
# Şifrelenmiş AES anahtarını alma
encrypted_aes_key = client_socket.recv(256)
aes_key = decrypt_aes_key_with_rsa(encrypted_aes_key, private_key)
```

Bu kısım start_server fonksiyonu içinde şifrelenmiş olan AES anahtarının şifresinin çözüldüğü kısımdır.

-
```python
# Şifrelenmiş dosyayı alma
encrypted_data = b''
while True:
    chunk = client_socket.recv(4096)
    if not chunk:
        break
    encrypted_data += chunk

print("Dosya alındı, şifre çözülüyor...")

# Şifre çözme
iv = encrypted_data[:16]
ciphertext = encrypted_data[16:]

decryptor = Cipher(
    algorithms.AES(aes_key),
    modes.CBC(iv),
    backend=default_backend()
).decryptor()

unpadder = padding.PKCS7(128).unpadder()

decrypted_padded = decryptor.update(ciphertext) + decryptor.finalize()
decrypted_data = unpadder.update(decrypted_padded) + unpadder.finalize()

# Dosyayı kaydetme
with open("received_file.txt", "wb") as f:
    f.write(decrypted_data)

print("Dosya başarıyla kaydedildi: received_file.txt")
client_socket.close()
```

Bu kısım start_server fonksiyonu içinde şifrelenmiş dosyanın clienttan alınıp şifresinin çözüldüğü kısımdır. Sonrasında ise alınan dosya kaydedilir ve o clienta olan erişim kapatılır.

```python
def generate_keys():
    # RSA anahtar çifti oluşturma
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048,
        backend=default_backend()
    )
    public_key = private_key.public_key()
    return private_key, public_key
```

- generate_keys fonksiyonu sunucunun RSA anahtar çifti üretmesini sağlar. Public key sunucuya gönderilir, private key ise sunucuda gizli tutulur. rsa fonksiyonu ile yapılır.

```python
def decrypt_aes_key_with_rsa(encrypted_key, private_key):
    # AES anahtarını RSA ile çözme
    decrypted_key = private_key.decrypt(
        encrypted_key,
        asymmetric_padding.OAEP(
            mgf=asymmetric_padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )
    return decrypted_key
```

- decrypt_aes_key_with_rsa fonksiyonu client tarafından gelen AES anahtarını, RSA-OAEP (Optimal Asymetric Encryption Padding) kullanılarak şifre çözülür. Koruma ise SHA-256 hash algoritması kullanılarak sağlanır.

```
PS C:\Users\erenk\OneDrive\Desktop\bilgisayar_aglari_proje> python main.py server
Sunucu başlatıldı, bağlantı bekleniyor...
```

- Sunucu başlatıldıktan sonra şekildeki ekran çıktısını verir ve 9999 portunu dinlemeye başlar.

- Başka bir terminale gelip “python main.py client --file example.txt --ip <SERVER_IP>” komutunu girdiğimizde ise client tarafından IP’si girilen servera dosya aktarımı yapabiliyoruz. Burada sunucuda RSA public anahtarı alınıyor, AES anahtarı oluşturulur ve şifreleme yapılır, SHA256 hash oluşturulur ve dosya paylaşılır.

```python
def encrypt_file_with_aes(file_path, key):
    # Dosyayı AES ile şifreleme
    iv = secrets.token_bytes(16)
    encryptor = Cipher(
        algorithms.AES(key),
        modes.CBC(iv),
        backend=default_backend()
    ).encryptor()

    padder = padding.PKCS7(128).padder()

    with open(file_path, 'rb') as f:
        file_data = f.read()

    padded_data = padder.update(file_data) + padder.finalize()
    encrypted_data = encryptor.update(padded_data) + encryptor.finalize()

    return iv + encrypted_data
```

- encrypt_file_with_aes fonksiyonu ile gönderilecek veri AES algoritması ile şifrelenir. Rastgele 16 bit uzunluğunda bir vektör üretilir. Dosya verisine gerekli padding yapılarak bloklara tamamlanılır. Bu sayede veri şifrelenmiş olur.

```python
def calculate_sha256(data):
    # SHA-256 özeti hesaplama
    return hashlib.sha256(data).digest()
```

- calculate_sha256 fonksiyonu ile gönderilen dosyanın büyüklüğünü doğrulama amaçlı bir SHA-256 hash değeri üretilir.

```python
def start_client(server_address, file_path):
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect(server_address)
    print(f"Sunucuya bağlandı: {server_address}")
```

- start_client fonksiyonu ile ise TCP sunucuya bağlanılır, RSA public key alınır, rastgele bir AES anahtarı üretilir, AES anahtarı RSA ile şifrelenip sunucuya gönderilir, dosya da AES ile şifrelenir ve dosya da sunucuya gönderilir.

```python
# Sunucudan RSA public key al
server_public_key_pem = client_socket.recv(2048)
server_public_key = serialization.load_pem_public_key(
    server_public_key_pem,
    backend=default_backend()
)
```

start_client fonksiyonu içindeki RSA public key alma kısmıdır. Bu sayede AES keyi şifrelenebilir.

-

```python
# AES anahtarı oluştur
aes_key = secrets.token_bytes(32)  # 256-bit AES anahtarı

# AES anahtarını RSA ile şifrele
encrypted_aes_key = server_public_key.encrypt(
    aes_key,
    asymmetric_padding.OAEP(
        mgf=asymmetric_padding.MGF1(algorithm=hashes.SHA256()),
        algorithm=hashes.SHA256(),
        label=None
    )
)
```

  start_client fonksiyonu içindeki AES anahtarı oluşturma ve bu anahtarı RSA ile şifreleme kısmıdır.

-

```python
# Şifrelenmiş AES anahtarını gönder
client_socket.send(encrypted_aes_key)

# Dosyayı AES ile şifrele
encrypted_file = encrypt_file_with_aes(file_path, aes_key)

# Dosya hash'i hesapla ve gönder
file_hash = calculate_sha256(open(file_path, 'rb').read())

print("Şifreli dosya gönderiliyor...")

# Dosya gönder
client_socket.sendall(encrypted_file)

print("Dosya gönderimi tamamlandı.")
client_socket.close()
```

  start_client fonksiyonu içindeki AES anahtarını servera gönderme, dosyayı AES anahtarı ile şifreleme, dosya hashini hesaplama ve dosyayı gönderip clientı kapatma kısımlarıdır.

-

```
PS C:\Users\erenk\OneDrive\Desktop\bilgisayar_aglari_proje> python main.py client --file gonderen_dosya.txt --ip 192.168.56.1
Sunucuya bağlandı: ('192.168.56.1', 9999)
Şifreli dosya gönderiliyor...
Dosya gönderimi tamamlandı.
```

  Client kodu çalıştırıldığında ekran çıktısı client kısmında bu şekildedir.

-

```
PS C:\Users\erenk\OneDrive\Desktop\bilgisayar_aglari_proje> python main.py server
Sunucu başlatıldı, bağlantı bekleniyor...
Bağlantı kabul edildi: ('192.168.56.1', 52808)
Dosya alındı, şifre çözülüyor...
Dosya başarıyla kaydedildi: received_file.txt
```

  Client kodu çalıştırıldığında ekran çıktısı server kısmında bu şekildedir.

- Latency ölçümü için “python main.py analyze --analyze latency --ip 8.8.8.8” komutunu terminale girdiğimizde mesela Google.com’a olan gecikmeyi ölçebilmekteyiz. IP adresini değiştirip istediğimiz servera olan latencymizi ölçebiliriz.

```python
def measure_latency(target_ip, count=5):
    """Ping kullanarak gecikmeyi ölçme"""
    import platform

    if platform.system() == "Windows":
        import subprocess
        import re

        try:
            output = subprocess.check_output(
                ["ping", "-n", str(count), target_ip],
                universal_newlines=True,
                stderr=subprocess.STDOUT
            )

            # Ping istatistiklerinden doğrudan değerleri alalım
            # "Minimum = 0ms, Maximum = 0ms, Average = 0ms" satırını arıyoruz
            stats_pattern = r"Minimum = (\d+)ms, Maximum = (\d+)ms, Average = (\d+)ms"
            stats_match = re.search(stats_pattern, output)

            if stats_match:
                min_latency = float(stats_match.group(1))
                max_latency = float(stats_match.group(2))
                avg_latency = float(stats_match.group(3))

                # Örnekleme verileri için her yanıtı ayrı ayrı kontrol edelim
                samples_pattern = r"time[=<](\d+)ms"
                samples = [float(match) for match in re.findall(samples_pattern, output)]

                # Eğer "<1ms" formatı varsa, bunları 0.5ms olarak kabul edelim
                if not samples:
                    less_than_pattern = r"time<1ms"
                    less_than_count = len(re.findall(less_than_pattern, output))
                    samples = [0.5] * less_than_count

                return {
                    'min': min_latency,
                    'avg': avg_latency,
                    'max': max_latency,
                    'samples': samples
                }
            else:
                print("Ping istatistikleri bulunamadı")
        except subprocess.CalledProcessError as e:
            print(f"Ping hatası: {e}")

        return None
```

- measure_latency fonksiyonu ile belirtilen hedef IP'ye ping komutu ile ping atılır ve gecikme ölçülmeye çalışılır. Ping komutu sonrası alınan cevaptan ihtiyacımız olan kısımları filtreliyoruz. Sonrasında çok küçük olan gecikmeleri belirli bir değere sabitliyoruz. Ping komutunun herhangi bir şekilde işlememesi sonucunu ise try-except durumu ile kontrol ediyoruz.

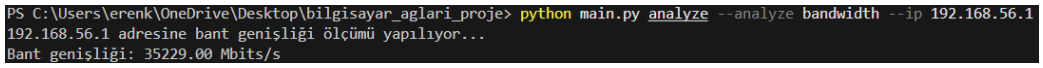
```
PS C:\Users\erenk\OneDrive\Desktop\bilgisayar_aglari_proje> python main.py analyze --analyze latency --ip google.com
google.com adresine gecikme ölçümü yapılıyor...
Minimum gecikme: 19.00 ms
Ortalama gecikme: 22.00 ms
Maksimum gecikme: 35.00 ms
```

Bu şekilde örneğin Google.com sitesine olan latencyi ölçebiliriz. Ekran çıktısı bu şekildedir.

- Bandwith ölçümü için öncelikle terminale gelip "iperf -s" komutu ile server olarak iperf'i[3] çalıştırıyoruz. Sonrasında ise farklı bir terminalde ise client olarak "python main.py analyze --analyze bandwidth --ip <SERVER_IP>" komutunu çalıştırıyoruz ve bu sayede ağın bant genişliğini ölçebiliyoruz.

```python
def run_iperf_client(server_ip, duration=10, port=5201):
    """
    iPerf kullanarak bant genişliğini ölçme
    """

    iperf_path = "C:\\iperf\\iperf3.exe"

    cmd = f"{iperf_path} -c {server_ip} -p {port} -t {duration} -f m"

    try:
        process = subprocess.Popen(cmd, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
        output, error = process.communicate()
        output = output.decode('utf-8')

        # Hata çıktısını kontrol edin
        if process.returncode != 0:
            print(f"iPerf hatası: {error.decode('utf-8')}")
            return None

        # iPerf çıktısından bant genişliğini çıkarma
        bandwidth_pattern = r"(\d+\.?\d*)\s+Mbits/sec"
        match = re.search(bandwidth_pattern, output)

        if match:
            bandwidth = float(match.group(1))
            return bandwidth
        else:
            print("iPerf çıktısından bant genişliği değeri çıkarılamadı")
            print(f"iPerf çıktısı: {output}")
            return None
    except Exception as e:
        print(f"Hata oluştu: {str(e)}")
        return None
```

- run_iperf_client fonksiyonu ile hedef IP üzerinde çalışan bir iPerf sunucusuna bağlanarak bant genişliği ölçebiliriz. iPerf üzerinden gelen çıktıyı algılayıp istenen sonuç yine filtrelenir, kullanıcıya geri dönüş sağlanır.

    - 

      iPerf sunucusuna bağlanıp bant genişliği ölçümü sonrası elde edilen ekran çıktısı bu şekildedir.
- Paket kaybı ve ağ güvenlik analizi kısımları şu anda yapım aşamasındadır. Teknolojileri ve nasıl yapabileceğimi araştırmaya devam ediyorum.

Limitations and Improvements – Kısıtlamalar ve Geliştirmeler

- Şu anda geliştirmemiş olmama rağmen bu proje üzerine bir Graphical User Interface (GUI) tasarlayıp kodlamayı da planlamaktayım.
- Ağ analizinde eksik paket yollama denemesinde tc kullanmayı deneyecektim ancak yaptığım araştırmalar sonucunda tc'nin yalnızca Linux'ta kullanıldığını öğrendim. Bu yüzden onun yerine Clumsy[5] kullanmayı planlıyorum.
- Wireshark[6] üzerinden iletimleri nasıl izleyeceğimi biraz çözdüm fakat tam anlayamadım, bundan sonraki kısımlarda oraya odaklanıp anlamaya çalışacağım.

Conclusion - Sonuç

Bu proje, güvenli dosya iletimi, düşük seviyeli IP paket işleme ve ağ performansı analizi konularını birleştiren, ileri düzeyde bir bilgisayar ağları uygulamasıdır. Projenin temel amacı, ağ üzerinden dosya gönderimini şifreli ve bütünlüğü korunmuş bir biçimde gerçekleştirmek, aynı zamanda bu aktarımı düşük seviyeli IP düzeyinde manuel olarak kontrol etmektir. Dosyalar AES algoritması kullanılarak simetrik olarak şifrelenmiş ve bu anahtar RSA ile asimetrik olarak güvenli biçimde sunucuya aktarılmıştır. Şifreleme işlemleri sırasında cryptography ve hashlib kütüphaneleri kullanılarak hem gizlilik (confidentiality) hem de bütünlük (integrity) sağlanmıştır. Gönderilen dosyanın SHA-256 özeti, alıcıda doğrulama için kullanılmıştır.

Dosya aktarımı sırasında düşük seviyeli IP paketleri Scapy kütüphanesi kullanılarak doğrudan yapılandırılmıştır. IP başlıklarında TTL, fragment offset, flags ve checksum alanları manuel olarak değiştirilmiş ve büyük veriler paketlere bölünerek fragmentasyon gerçekleştirilmiştir. Bu sayede, hem IP düzeyinde veri iletimi hem de yeniden birleştirme (reassembly) süreçleri uygulamalı olarak test edilmiştir. Ek olarak checksum hesaplamaları, olası iletim hatalarını tespit etmek üzere özel olarak kodlanmıştır.

Ağ performans analizi modülünde, gecikme ölçümü için sistem bağımlı olarak ping komutu kullanılmış, bant genişliği ölçümleri ise iperf aracı ile yapılmıştır. Clumsy adlı harici bir araç ile paket kaybı ve ağ gecikmesi simüle edilmesi planlanmaktadır. Böylece sistem, farklı ağ koşulları (Wi-Fi, Ethernet, sanal ağlar vb.) altında test edilerek analiz edebilecektir.

Projenin güvenlik analiz bileşeninde ise, ağ trafiği dinlenecek ve analiz edilecektir. Scapy ile gerçek zamanlı olarak filtreli paket yakalama işlemleri yapılacak, şifreli veri içeren paketler entropi analizine tabi tutulacaktır. Bu analizle, yakalanan verinin

rastgeleliğe dayalı olarak şifreli olup olmadığı değerlendirilecektir. Ayrıca Man-in-the-Middle (MITM) saldırısı bir ARP spoofing simülasyonu ile gerçekleştirilecek ve sahte HTTP istekleri gönderilerek paket enjeksiyonu yapılacaktır. Bu işlemler sonucunda, sistemin veri güvenliği açıklarına karşı dayanıklılığı da test edilecektir.

Genel olarak, bu proje şifreleme tekniklerinden düşük seviyeli IP işleme yapılarına, ağ performans analizinden güvenlik testlerine kadar pek çok ağ teknolojisini bütüncül bir yapıda bir araya getirecektir. Hem istemci hem de sunucu tarafında platforma özel koşullar gözetilerek uyumlu çalışma sağlanacak, analiz ve simülasyon modülleri sayesinde kullanıcıya çeşitli test senaryoları sunulacaktır. Proje, ağ katmanlarının yapısını, güvenlik protokollerinin uygulamasını ve performans testlerini derinlemesine öğrenmek için kapsamlı ve teknik açıdan güçlü bir örnek teşkil etmektedir.

References - Kaynakça

- [1] Comer, D. E. (2018). *Internetworking with TCP/IP: Principles, Protocols, and Architecture* (6th ed.). Pearson.
- [2] Milanov, E. (2009). The RSA algorithm. *RSA laboratories*, *1*(11).
- [3] Daemen, J., & Rijmen, V. (2002). *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer Science & Business Media.
- [4] Taso, K., & Tirumala, A. (2005). *iPerf: Bandwidth Measurement Tool*. https://iperf.fr
- [5] Jagt. (2016). *Clumsy* [Computer software]. GitHub. https://github.com/jagt/clumsy
- [6] The Wireshark Team. (n.d.). *Wireshark user documentation*. Wireshark. https://www.wireshark.org/docs/