Ağ Güvenliği Proje Ödevi

Öğrenci Bilgileri: - **413518** - Berk Çağrı Laçin - **402498** - Enes Ceviz

---

Raw Socket Port Scanner - Program Akışı ve Çalışma Mantığı

Genel Program Akışı

`main() → Parametre Kontrolü → Root Yetki Kontrolü → Hostname Çözümleme → PortScanner Oluşturma → Port Aralığı Parse → Tarama Başlatma → Sonuç Gösterme`

Ana Fonksiyon (main) Akışı

1. Parametre Kontrolü

```
if (argc != 3) {
    // Kullanım bilgilerini göster
    return 1;
}
```

- Program 2 parametre bekler: `<IP_adresi>` ve `<port_aralığı>`
- Eksik parametre durumunda kullanım örnekleri gösterilir

2. Root Yetki Kontrolü

```
if (getuid() != 0) {
    // Root yetkisi uyarısı
    return 1;
}
```

- Raw socket kullanımı için root yetkisi zorunlu
- `getuid() != 0` kontrolü ile root olmayan kullanıcılar uyarılır

3. Hostname Çözümleme

```
struct hostent *host_entry = gethostbyname(target_input.c_str());
if (host_entry != nullptr) {
    target_ip = inet_ntoa(*((struct in_addr *)host_entry->h_addr_list[0]));
}
```

- Domain adları IP adresine çevrilir
- `gethostbyname()` fonksiyonu DNS çözümlemesi yapar

PortScanner Sınıfı Yapısı

Constructor (Yapıcı Fonksiyon)

```
PortScanner(const std::string &ip) : target_ip(ip) {
    // Bilinen servisleri initialize et
    known_services[21] = "FTP";
    known_services[22] = "SSH";
    // ... diğer servisler
}
```

- Hedef IP'yi saklar
- Bilinen servis portlarını harita olarak tanımlar

Port Aralığı Parse İşlemi

parsePortRange() Fonksiyonu

```
void parsePortRange(const std::string &port_range) {
    std::stringstream ss(port_range);
    std::string token;

    while (std::getline(ss, token, ',')) {
        size_t dash_pos = token.find('-');
        if (dash_pos != std::string::npos) {
            // Aralık formatı (1-1024)
            int start = std::stoi(token.substr(0, dash_pos));
            int end = std::stoi(token.substr(dash_pos + 1));
            for (int i = start; i <= end; i++) {
                ports.push_back(i);
            }
        } else {
            // Tek port (80)
            ports.push_back(std::stoi(token));
        }
    }
}
```

Desteklenen Formatlar: - `80` - Tek port - `1-1024` - Port aralığı - `22,80,443` - Çoklu port - `1-100,443,8080-8090` - Karma format

Raw Socket İmplementasyonu

TCP SYN Scan Akışı

1. Raw Socket Oluşturma

```
int sock = socket(AF_INET, SOCK_RAW, IPPROTO_TCP);
if (sock < 0) {
```

```
    return false; // Fallback'e geç
}
```

2. Socket Seçenekleri

```
int one = 1;
setsockopt(sock, IPPROTO_IP, IP_HDRINCL, &one, sizeof(one));
```

- `IP_HDRINCL`: Manuel IP header oluşturma izni

3. Paket Oluşturma

```
char packet[4096];
struct ip_header *ip_hdr = (struct ip_header *)packet;
struct tcp_header *tcp_header = (struct tcp_header *)(packet + sizeof(struct ip_header));
```

4. IP Header Doldurma

```
ip_hdr->ip_vhl = (4 << 4) | 5;  // IPv4, 20 byte header
ip_hdr->ip_tos = 0;             // Type of Service
ip_hdr->ip_len = htons(sizeof(struct ip_header) + sizeof(struct tcp_header));
ip_hdr->ip_id = htons(54321);   // Identification
ip_hdr->ip_off = 0;             // Fragment offset
ip_hdr->ip_ttl = 255;           // Time to Live
ip_hdr->ip_p = IPPROTO_TCP;     // Protocol
```

5. Local IP Tespiti

```
int temp_sock = socket(AF_INET, SOCK_DGRAM, 0);
connect(temp_sock, (struct sockaddr *)&temp_dest, sizeof(temp_dest));
getsockname(temp_sock, (struct sockaddr *)&local_addr, &addr_len);
```

- Geçici UDP socket ile local IP adresi tespit edilir

6. TCP Header Doldurma

```
tcp_header->th_sport = htons(12345);    // Source port
tcp_header->th_dport = htons(port);     // Destination port
tcp_header->th_seq = 0;                 // Sequence number
tcp_header->th_ack = 0;                 // Acknowledgment
tcp_header->th_off = 5 << 4;            // Data offset
tcp_header->th_flags = TH_SYN;          // SYN flag
tcp_header->th_win = htons(5840);       // Window size
```

7. Checksum Hesaplama

```
ip_hdr->ip_sum = checksumCalculator((uint16_t *)ip_hdr, sizeof(struct ip_header));
tcp_header->th_sum = calculateTCPChecksum(ip_hdr, tcp_header);
```

TCP Checksum Algoritması:

```
// Pseudo header oluştur
struct pseudo_header {
    uint32_t source_address;
    uint32_t dest_address;
    uint8_t placeholder;
    uint8_t protocol;
    uint16_t tcp_length;
};

// Checksum hesapla
uint16_t checksumCalculator(uint16_t *ptr, int nbytes) {
    long sum = 0;
    while (nbytes > 1) {
        sum += *ptr++;
        nbytes -= 2;
    }
    if (nbytes == 1) {
        oddbyte = 0;
        *((uint8_t *)&oddbyte) = *(uint8_t *)ptr;
        sum += oddbyte;
    }
    sum = (sum >> 16) + (sum & 0xffff);
    sum = sum + (sum >> 16);
    return (uint16_t)(~sum);
}
```

8. Paket Gönderme

```
sendto(sock, packet, ntohs(ip_hdr->ip_len), 0, (struct sockaddr *)&dest, sizeof(dest));
```

9. Yanıt Dinleme

```
fd_set readfds;
struct timeval timeout;
timeout.tv_sec = 2;
timeout.tv_usec = 0;

FD_ZERO(&readfds);
FD_SET(sock, &readfds);

if (select(sock + 1, &readfds, NULL, NULL, &timeout) > 0) {
    if (recvfrom(sock, buffer, 4096, 0, (struct sockaddr *)&from, &fromlen) > 0) {
        // Yanıt analizi
    }
}
```

10. Yanıt Analizi

```
struct ip_header *recv_ip = (struct ip_header *)buffer;
struct tcp_header *recv_tcp = (struct tcp_header *)(buffer + ((recv_ip->ip_vhl & 0x0F) * 4))

if (recv_tcp->th_dport == htons(12345) && recv_tcp->th_sport == htons(port)) {
    ttl = recv_ip->ip_ttl;
    window_size = ntohs(recv_tcp->th_win);

    if ((recv_tcp->th_flags & TH_SYN) && (recv_tcp->th_flags & TH_ACK)) {
        port_open = true; // SYN+ACK alındı
    }
}
```

Fallback Mekanizması

TCP Connect Scan

```
bool tcpConnectScan(int port) {
    int sock = socket(AF_INET, SOCK_STREAM, 0);

    struct timeval timeout;
    timeout.tv_sec = 2;
    timeout.tv_usec = 0;
    setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, &timeout, sizeof(timeout));

    struct sockaddr_in target;
    target.sin_family = AF_INET;
    target.sin_port = htons(port);
    target.sin_addr.s_addr = inet_addr(target_ip.c_str());

    bool result = (connect(sock, (struct sockaddr *)&target, sizeof(target)) == 0);
    close(sock);
    return result;
}
```

UDP Scan (Seçili Portlar)

```
bool udpScan(int port) {
    // Sadece DNS(53), SNMP(161), NTP(123), TFTP(69) için
    if (port == 53 || port == 161 || port == 123 || port == 69) {
        // UDP socket oluştur ve test paketi gönder
    }
}
```

İşletim Sistemi Tespiti

TTL Analizi

```
void detectOS(int ttl, int window_size) {
    int original_ttl = 0;
    int hops = 0;

    if (ttl <= 64) {
        original_ttl = 64;
        hops = 64 - ttl;
        detected_os = "Linux/Unix";
    } else if (ttl <= 128) {
        original_ttl = 128;
        hops = 128 - ttl;
        detected_os = "Windows";
    } else if (ttl <= 255) {
        original_ttl = 255;
        hops = 255 - ttl;
        detected_os = "Cisco/Network Device";
    }
}
```

OS Tespit Mantığı: - **TTL 64:** Linux/Unix sistemler - **TTL 128:** Windows sistemler
- **TTL 255:** Cisco/Network cihazları - **Hop Count:** Network mesafesi hesaplama

Window Size Analizi

```
if (window_size == 65535) {
    detected_os += " (Max Window - Linux/BSD)";
} else if (window_size == 8192) {
    detected_os += " (8K Window - Windows)";
} else if (window_size == 5840) {
    detected_os += " (5840 Window - Linux)";
}
```

Servis Tespiti ve Banner Grabbing

Banner Grabbing Süreci

```
std::string grabBanner(int port) {
    int sock = socket(AF_INET, SOCK_STREAM, 0);

    // HTTP portları için özel istek
    if (port == 80 || port == 443 || port == 8080) {
        send(sock, "HEAD / HTTP/1.0\r\n\r\n", 18, 0);
```

```
    }

    char buffer[1024];
    recv(sock, buffer, sizeof(buffer) - 1, 0);

    // İlk satırı al
    std::string banner(buffer);
    size_t newline = banner.find('\n');
    if (newline != std::string::npos) {
        banner = banner.substr(0, newline);
    }

    return banner;
}
```

Multi-Threading Yapısı

Ana Tarama Döngüsü

```
void scan() {
    std::vector<std::thread> threads;
    const int max_threads = 50;

    for (size_t i = 0; i < ports.size(); i += max_threads) {
        threads.clear();

        // 50'şer port grupları halinde thread oluştur
        for (int j = 0; j < max_threads && (i + j) < ports.size(); j++) {
            threads.emplace_back(&PortScanner::scanPort, this, ports[i + j]);
        }

        // Tüm thread'lerin bitmesini bekle
        for (auto &t : threads) {
            t.join();
        }

        // İlerleme göster
        std::cout << "\rTarama ilerlemesi: " << std::min(i + max_threads, ports.size())
                  << "/" << ports.size() << " port" << std::flush;
    }
}
```

Thread-Safe Sonuç Toplama

```
void scanPort(int port) {
    // ... tarama işlemleri ...
```

```
    if (is_open) {
        std::lock_guard<std::mutex> lock(result_mutex);
        open_ports[port] = scan_method;
        services[port] = service;

        std::cout << "Port " << port << " açık: " << services[port]
                  << " [" << scan_method << "]" << std::endl;
    }
}
```

Sonuç Gösterme

Gerçek Zamanlı Bildirimler

- Her açık port bulunduğunda anında konsola yazdırılır
- Format: `Port 80 açık: HTTP [TCP-SYN]`

Final Rapor Tablosu

```
void displayResults() {
    // ASCII tablo formatında sonuçları göster
    // Port, Durum, Protokol, Servis bilgileri
}
```