



1차 프로젝트 보고서

태그

1. 기획

1.1 기획 보고서

1.2 요구사항 명세서

1.3 flow chart

2. 실행

2.1 사용 기술 스택들

2.2 포트 스캐너 프로젝트 일정

2.3 단계 별 프로세스

2.4 포트별 스캔 방법

2.6 통합 포트 스캐너 코드

2.7 Flask 구현 코드

2.8 테스트 환경

Test Server

Flask Portscanner

(구동 환경 : goorm IDE Flask 컨테이너)

2.9 최종 결과

2.9.1 스크린샷

2.9.2 최종 Release

3. 회고

3.1 프로젝트 자체 평가 및 느낀 점

홍영창

구현모

김동진

장다솜

조운지

최승희

4. 참고 문헌

1. 기획

1.1 기획 보고서

▼ 서비스 포트 스캐너 구현 기획 보고서

프로젝트 명: 서비스 포트 스캐너 구현

팀명: 해커잡조

1. 개요

1.1 팀 구성원 및 담당 업무

이름	역할	담당 업무
공통진행		포트별 서비스 스캔 코딩
홍영창	팀장	프로젝트 총괄, 노션 및 발표
구현모	팀원	프론트엔드와 서비스 배포
김동진	팀원	테스트 서버 환경 구축
장다솜	팀원	WBS 및 개발 문서 작성
조운지	팀원	프레임과 백엔드
최승희	팀원	포트별 스캐너 통합

1.2 목적

본 프로젝트의 목적은 네트워크 상의 서비스 포트를 탐색하고, 해당 정보를 사용자에게 제공하는 기능을 수행하는 서비스 포트 스캐너를 개발하는 것입니다. 이를 통해 네트워크 보안 감사, 시스템 관리, 그리고 네트워크 서비스 모니터링에 도움을 줄 수 있습니다. 구체적으로는 미리 정의된 스캔 포트 목록을 대상으로 포트 오픈 상태와 배너 그레이팅 정보를 받아오는 서비스 포트 스캐너를 구현하여, 네트워크의 보안 상태를 신속하게 평가하고, 필요한 조치를 취할 수 있도록 합니다.

2. 프로젝트 필요성

네트워크 상의 열린 포트와 실행 중인 서비스를 정확히 식별하여 보안 취약점을 관리하는 것은 매우 중요합니다. 따라서 미리 정의된 스캔 목록을 대상으로 포트 오픈 상태와 배너 그레이팅 정보를 받아오는 서비스 포트 스캐너를 개발함으로써, 네트워크의 보안 상태를 평가하고 조치를 취할 수 있습니다.

3. 주요 기능 및 스펙

- 입력 및 출력:** 사용자는 IP 주소, 도메인을 입력하고, 미리 정의된 스캔 포트 목록에 대한 포트 오픈 상태 및 배너 그레이팅 정보를 출력합니다.
- 스캔 기능:** 미리 정의된 스캔 포트 목록을 대상으로 포트 오픈 상태와 배너 그레이팅 정보를 받아옵니다. TCP와 UDP 포트 스캔, 타임아웃 설정, 멀티스레딩을 지원합니다.
- 결과 출력:** 사용자가 쉽게 이해할 수 있는 형식으로 결과를 출력하며, 열린 포트의 상태와 해당 서비스의 배너 그레이팅 정보를 제공합니다.
- 성능 및 보안:** 멀티스레딩과 비동기식 처리를 통해 전체적인 스캔 속도를 향상 시킬 수 있습니다.

4. 기대 효과

미리 정의된 스캔 포트 목록을 대상으로 포트 오픈 상태와 배너 그레이팅 정보를 받아오는 이 스캐너를 활용함으로써, 중요한 포트들에 대한 보안 상태를 신속하게 모니터링하고 관리할 수 있습니다. 이를 통해 보다 효율적인 네트워크 운영이 가능해질 수 있습니다.

5. 구현 계획

본 프로젝트는 다음 단계로 구성됩니다:

- 요구사항 분석 및 설계:** 프로젝트의 목표와 요구사항을 명확히 정의하고, 시스템의 전체적인 구조와 기능을 설계합니다.
- 개발:** Python과 Flask를 사용하여 서비스 포트 스캐너를 개발합니다. 이때, 미리 정의된 스캔 포트 목록을 대상으로 포트 오픈 상태와 배너 그레빙 정보를 받아오는 기능을 구현합니다. 개발 과정에서는 주로 VMware 가상 환경에 구축된 Ubuntu 서버를 주요 테스트 환경으로 활용하며, 구름 IDE를 사용하여 개발을 진행합니다.
- 테스트 및 검증:** 개발된 스캐너의 기능과 성능을 다양한 환경에서 테스트하고 검증합니다. 특히, 미리 정의된 스캔 포트 목록을 대상으로 정확한 포트 오픈 상태와 배너 그레빙 정보를 받아오는지 확인합니다.
- 배포 및 유지보수:** 스캐너를 최종 사용자에게 배포하고, 지속적인 유지보수 및 업데이트를 수행합니다. 이 과정에서 사용자가 미리 정의된 스캔 포트 목록을 설정할 수 있도록 설정 인터페이스를 제공합니다. 배포는 구름 IDE를 통해 이루어집니다.

6. 기술 지원

- 개발 도구:** VMware, Ubuntu, 구름 IDE
- 협업 도구:** Discord
- 문서 도구:** Github, Notion

1.2 요구사항 명세서

▼ 서비스 포트 스캐너 구현 요구사항 명세서

1. 개요 및 목적

이 프로젝트의 목적은 네트워크에서 호스트에 오픈되어 있는 포트를 스캔하여 서비스의 가용성을 확인하는 소프트웨어를 개발하는 것입니다. 이 스캐너는 네트워크 상의 서비스 포트를 탐색하고, 해당 정보를 사용자에게 제공하는 기능을 수행합니다.

- 용어 정의:**
 - 포트 스캐너(Port Scanner):** 네트워크 상의 호스트를 대상으로 열려 있는 포트를 탐색하는 소프트웨어.
 - GUI(Graphical User Interface):** 그래픽 기반의 사용자 인터페이스.
 - CLI(Command Line Interface):** 텍스트 기반의 사용자 인터페이스.
 - Python:** 고급 프로그래밍 언어로, 본 프로젝트의 개발에 사용됩니다.
 - Ubuntu:** 리눅스 기반의 운영 체제로, 본 프로젝트의 개발 및 테스트 환경 중 하나입니다.
 - Flask:** Python 기반의 마이크로 웹 프레임워크로, 본 프로젝트의 웹 인터페이스 개발에 사용됩니다.
 - 구름 IDE:** 온라인 기반의 통합 개발 환경으로, 프로젝트 배포 및 협업을 위해 활용됩니다.

2. 전반적인 설명

- 제품 관점:** 서비스 포트 스캐너는 네트워크 보안 감사, 시스템 관리 및 네트워크 서비스 모니터링을 위해 사용됩니다. 본 프로젝트는 Ubuntu 서버를 사용하여 네트워크 상의 포트 개방 상태를 확인하고, Python 및 Flask를 이용하여 개발되었습니다.
- 제품 기능:**

- 네트워크 상의 특정 호스트 또는 IP 범위에 대한 포트 스캔
- 미리 정의된 포트 목록을 대상으로 포트 오픈 상태와 배너 그레빙 정보를 받아오는 기능 제공
- 스캔 결과 보고서 생성
- **사용자 클래스와 특성:** 네트워크 관리자, 보안 전문가, IT 전문가들은 본 소프트웨어를 사용하여 네트워크의 보안 상태를 평가하고, 필요한 보안 조치를 취할 수 있습니다.
- **운영 환경:** Windows, Linux, macOS 등 다양한 운영 체제에서 실행될 수 있으나, 개발 및 주요 테스트는 Ubuntu 서버 환경에서 이루어졌습니다.
- **설계 및 구현 제약사항:** 프로젝트는 Python 3.x를 사용하여 개발되며, Flask 웹 프레임워크를 통해 사용자 친화적인 웹 인터페이스를 제공합니다. 멀티 스레딩 및 비동기 처리 방식을 활용하여 성능을 최적화할 예정입니다.
- **가정 및 의존성:** 사용자는 기본적인 네트워크 및 시스템 관리 지식을 가지고 있으며, Python 및 필요한 라이브러리가 시스템에 설치되어 있습니다. 본 소프트웨어는 Ubuntu 서버 환경에서의 최적 성능을 가정하고 설계되었습니다.
- **배포:** 프로젝트는 구름 IDE를 활용하여 개발되었으며, 해당 IDE를 통해 배포 및 협업이 이루어집니다.

3. 상세 요구사항

3.1 기능적 요구사항

1. 입력

- 사용자는 스캔 대상으로 IP 주소 또는 도메인을 입력할 수 있어야 합니다.
- 스캔할 포트는 미리 정의된 포트 목록으로 지정됩니다.

2. 출력

- 스캐너는 지정된 포트의 오픈 상태와 해당 포트에 대한 서비스 정보를 제공해야 합니다.

3. TCP와 UDP 포트 스캔 기능

- 스캐너는 TCP 및 UDP 포트를 스캔할 수 있는 기능을 제공해야 합니다.

4. 멀티스레딩을 통한 병렬 스캐닝

- 스캐너는 멀티스레딩을 활용하여 동시에 미리 정의된 포트를 스캔할 수 있어야 합니다.

5. 한눈에 알아보기 쉬운 결과 출력

- 스캔 결과는 사용자가 쉽게 이해할 수 있는 형식으로 출력되어야 합니다.

3.2 비기능적 요구사항

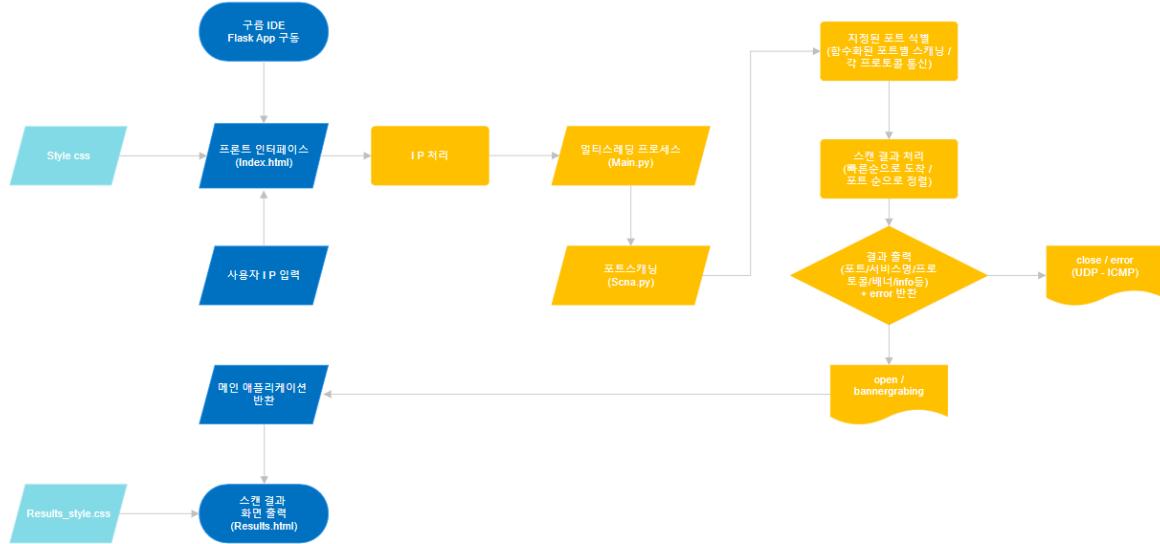
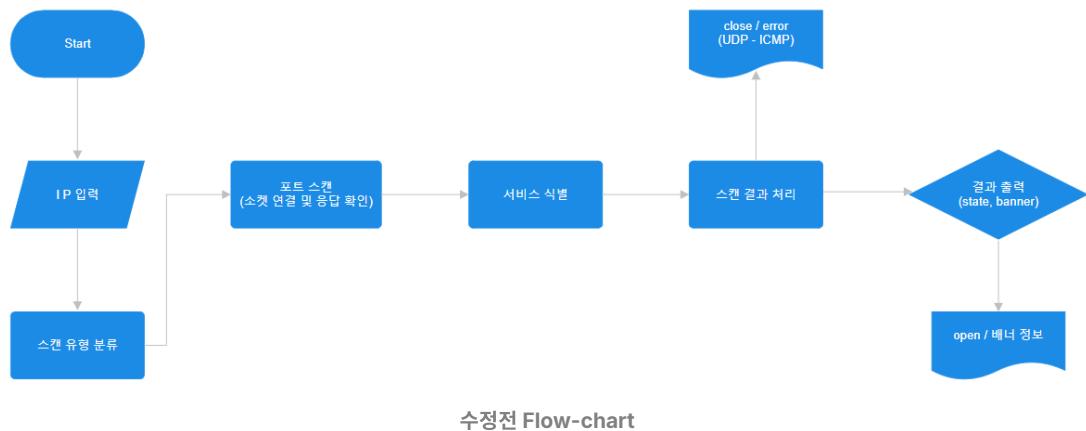
1. 성능 요구사항

- 스캐너는 멀티스레딩을 사용하여 미리 정의된 포트를 스캔할 수 있어야 합니다.
- 이 서비스 포트 스캐너는 간단한 과제용으로 제작되었으므로, 높은 성능을 요구하지 않습니다.

2. 보안 요구사항

- 포트 오픈 여부를 통해 닫힌 포트를 식별하고 관리할 수 있어야 합니다.
- 이 서비스 포트 스캐너는 사용자 인증 기능이 제공되지 않으며, 신뢰성이 낮을 수 있습니다.

1.3 flow chart



2. 실행

2.1 사용 기술 스택들



2.2 포트 스캐너 프로젝트 일정

프로젝트 명	서비스 포트 스캐너 구현
팀명	해커잡조
팀원	홍영창, 구현모, 김동진, 장다솜, 조윤지, 최승희
시작일	24. 01. 31 오전 10:00
종료일	24. 02. 16 오전 10:00

주요 업무	세부 업무	담당	소요일	기간														
				1월		2월												
프로젝트 기획	프로젝트 계획 수립	공통	2일	31	1	2	3	4	5	6	7	8	9	10	11	12	13	
	Well-Known Port 분배	공통	2일														14	15
개발	Backend Development	조윤지	13일															
	포트별 분석 및 구현	공통	12일															
	서비스 스캔 코드 통합	최승희	7일															
	테스트 서버 환경 구축	김동진	7일															
	Flask Frontend	구현모	2일															
문서화 작업	구름 IDE 배포	구현모	2일															
	WBS 및 개발 문서 작성	장다솜	2일															
	보고서 작성 및 Notion 정리	공통	2일															
	프로젝트 총괄, 노선 및 발표	홍영창	2일															
	Notion & Github	공통	16일															

2.3 단계 별 프로세스

일정	실행 내용
[Day 01]	프로젝트 진행 방향 정하기
24.01.31	서비스 스캐너에 대한 정의 및 구조 알아보기 1~65535 구현하는 포트스캐너 보다는 포트의 프로토콜 구조대로 구현하는 포트스캐너 만들기
[Day 02]	Well-known Port 분배
24.02.01	각 3개씩 포트 분배해서 개념 공부하기
[Day 03]	각자 분배된 스캔 코드 공부 및 구현
24.02.02	맡은 포트 스캔 코드 만들기
[Day 04]	[Day 03] 동일 / 진행사항 공유
24.02.05	출력시 포트 오픈+배너그래빙 포함하기
[Day 05]	포트 스캔 응답 확인 및 통합 구현
24.02.06	방화벽 오픈하고 listen 상태로 만들기 테스트 시 발생하는 이슈 관련 공유 및 에러 나는 부분 해결하기 netstat 명령어와 nmap 통해서 포트 오픈 확인하기
[Day 06]	[Day 05]동일 / 각 포트 스캔 코드 통합
24.02.07	배너 정보 못 받아오는 코드 수정 및 문제 있는 코드 서로 도와주기 개인별 작성했던 코드 병합 진행 운지님 - 코드 통합 제안 승희님 - 운지님 코드에 승희님 코드 통합 후 공유 동진님 - 테스트 환경 구축 (Ubuntu)
[Day 07]	멀티스레딩 및 테스트 서버 구축

24.02.08	6명 코드 통합 완료 및 테스트 서버에서 구현해보기. port Scan 병렬처리 구현 완료 / <code>response_data</code> 형식으로 통합 승희님 – 각자 구현한 코드 통합 및 포트 출력 방식 통일시키기 동진님 – vmware / ubuntu 운영체제에서 포트가 모두 오픈된 테스트 환경 구축함 (211.229.25.137) 테스트 구축 후 코드 테스트 진행 6명 코드 통합 및 멀티스레딩 완료 FTP/SSH/Telnet/SMTP/DNS/HTTP/POP3/NTP/IMAP/SNMP/SMB/LDAP/HTTPS/MYSQL/RDP
24.02.11	트러블 슈팅 및 버그 픽스 운지님 현모님 Flask 웹 구현 시도 수정 필요한 부분은 지속적인 검토 및 업데이트 진행
[Day 08]	서비스 스캔 및 Flask Front-end
24.02.13	현모님 - 플라스크 프론트엔드 및 구름 IDE 배포 flask + redis 이용한 컨테이너 작업 운지님, 현모님, 동진님, 승희님 같이 컨테이너 공유해서 작업함 에러수정 및 최종 테스트 영창님 - 팀 노션 및 보고서 작성 틀 제작
[Day 09]	보고서 형식 논의 및 작성
24.02.14	각자 정리한 Github / Notion, 팀 Notion에 정리하기 영창님 - 보고서 작성을 위해 회고 작성 진행 다솜님 - WBS 및 개발 문서 작성 Flow chart 만들기 UDP 배너그래빙 관련한 사항 업데이트
[Day 10]	보고서 점검 및 마무리
24.02.15	개인별 보고서 작성 마무리 및 개인별 느낀점 작성 영창님 - 보고서 점검 및 발표 준비
[Day 11]	제출 및 발표
24.02.16	영창님 - 제출 및 발표

2.4 포트별 스캔 방법

▼ FTP - 21



FTP란?



TCP/IP 네트워크(인터넷)상의 장치가 파일을 전송할 때 사용하는 프로토콜

동작 방식(Active 모드)

1. Client가 Server에 **PORT** 명령을 전송하여 데이터 연결을 생성한다.
2. Server는 Client가 지정한 IP 주소와 포트 번호로 데이터 연결을 수립하고, 데이터를 전송한다.
3. Client는 데이터를 수신하고 PORT 명령으로 지정한 포트 번호에서 수신 대기한다.
4. Server가 데이터를 전송할 때는 Client수신 대기 중인 포트로 데이터를 전송합니다.

동작 방식(Passive 모드)

1. Client가 Server에 **PASV** 명령을 전송하여 패시브 모드로 전환한다.
2. Server는 임의의 포트 번호를 선택하여 클라이언트에게 응답한다.
3. Client는 Server가 응답한 IP 주소와 포트 번호로 데이터 연결을 수립하고, 데이터를 전송하기 위해 서버에게 **RETR** 명령 등을 전송한다.
4. Server는 Client가 수립한 데이터 연결로 데이터를 전송한다.

서비스 설치 및 동작 방식

```
#vsftpd 설치:  
sudo apt-get update  
sudo apt-get install vsftpd
```

```
#vsftpd 서비스 시작  
sudo systemctl start vsftpd
```

구현

```
import socket  
  
def ftp_port_scan(target_host, target_port):  
    # 소켓 객체 생성  
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  
  
    # 소켓 타임아웃 설정  
    sock.settimeout(1)  
  
    try:  
        # FTP 포트에 연결을 시도합니다.  
        sock.connect((target_host, target_port))  
  
        # 연결 성공 시 FTP 포트가 열려 있다는 메시지를 출력합니다.  
        print("FTP port {} is open on {}".format(target_port, target_host))  
    except socket.error:  
        pass
```

```
print(f"FTP port {target_port} is open")

# 서비스의 배너 정보를 수신합니다.
banner = sock.recv(1024).decode("utf-8")
print("Banner information:", banner.strip())
except socket.timeout:
    # 연결 타임아웃 처리
    print(f"FTP port {target_port} is closed")
except ConnectionRefusedError:
    # 연결이 거부됨
    print(f"Connection to port {target_port} refused")
except Exception as e:
    # 기타 예외 처리
    print(f"An error occurred: {e}")
finally:
    # 소켓 닫기
    sock.close()

# 스캔을 실행할 호스트와 FTP 포트를 설정합니다.
target_host = "example.com"
ftp_port = 21

# FTP 포트 스캔 실행
ftp_port_scan(target_host, ftp_port)
```

▼ SSH - 22



SSH란?



네트워크를 통해 안전하게 원격으로 컴퓨터에 접속하고 통신할 수 있는 보안 프로토콜

동작 방식

- 클라이언트가 SSH 서버에 연결하기 위해 SSH 클라이언트를 실행하고, SSH 서버의 IP 주소 및 사용자 이름을 지정한다.
- SSH 서버는 클라이언트의 연결 요청을 받으면, 클라이언트가 제공한 사용자 이름과 비밀번호, 또는 공개키 기반의 인증을 통해 사용자를 인증한다.
- 클라이언트와 서버 간의 모든 통신은 암호화되어 전송된다.
- SSH 세션은 사용자가 원격 시스템에 접속하는 동안 유지되고, 사용자는 원격 시스템에서 명령을 실행하거나 파일을 전송할 수 있다.

서비스 설치 및 동작 방식

```
# ssh 서비스 시작, 리눅스는 ssh 깔려있음
sudo service ssh start

# 포트 열렸는지 확인
sudo netstat -tulnp | grep 22
```

구현

```
import socket

def ssh_port_scan(target_host, target_port):
    # 소켓 객체 생성
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # 소켓 타임아웃 설정
    sock.settimeout(1)

    try:
        # SSH 포트에 연결을 시도합니다.
        sock.connect((target_host, target_port))

        # 연결 성공 시 SSH 포트가 열려 있다는 메시지를 출력합니다.
        print(f"SSH port {target_port} is open")

        # 서비스의 배너 정보를 수신합니다.
        banner = sock.recv(1024).decode("utf-8")
        print("Banner information:", banner.strip())
    except socket.timeout:
        # 연결 타임아웃 처리
```

```
        print(f"SSH port {target_port} is closed")
    except ConnectionRefusedError:
        # 연결이 거부됨
        print(f"Connection to port {target_port} refused")
    except Exception as e:
        # 기타 예외 처리
        print(f"An error occurred: {e}")
    finally:
        # 소켓 닫기
        sock.close()

# 스캔을 실행할 호스트와 SSH 포트를 설정합니다.
target_host = "example.com"
ssh_port = 22

# SSH 포트 스캔 실행
ssh_port_scan(target_host, ssh_port)
```

▼ Telnet - 23



Telnet이란?



네트워크를 통해 원격 시스템에 로그인하고 원격 컴퓨터 또는 장치를 제어하기 위한 프로토콜

동작 방식

- 연결 설정: 클라이언트가 Telnet 서버에 TCP 연결을 시도합니다. 이때 일반적으로 포트 23을 사용합니다.
- 세션 설정: 서버는 클라이언트의 연결 요청을 수락하고 Telnet 세션을 설정합니다.
- 명령 및 데이터 전송: 클라이언트와 서버 간에는 텍스트 기반의 데이터가 전송됩니다. 클라이언트는 사용자가 입력한 명령을 서버로 전송하고, 서버는 해당 명령을 실행한 결과를 클라이언트로 다시 전송합니다.
- 세션 종료: 클라이언트가 명시적으로 연결을 닫거나, 네트워크 문제 등으로 연결이 끊길 수 있습니다.

서비스 설치 및 동작 방식

```
#telnetd 설치:  
sudo apt update  
sudo apt install telnetd  
  
#telnet 서비스 시작  
sudo systemctl start telnet
```

SMTP는 서비스를 설치하지 않더라도 포트는 기본적으로 열려 있음

구현

```
import socket  
import telnetlib  
  
def SYN_scan(host, port):  
    try:  
        # 소켓 생성  
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  
        sock.settimeout(5) # 타임아웃 설정  
  
        # SYN 패킷 전송  
        result = sock.connect_ex((host, port))  
  
        # 결과 확인  
        if result == 0:  
            return True # 포트가 열려있음  
        else:  
            return False # 포트가 닫혀있음  
    except Exception as e:  
        return None # 예외 발생 시 None 반환
```

```

        finally:
            sock.close()

def Telnet_scan(host, port):
    service_name = "Telnet"
    try:
        tn = telnetlib.Telnet(host, port, timeout=5) # Telnet 객체 생성
        banner = tn.read_until(b"\r\n", timeout=5).decode('utf-8') #
        tn.close() # 연결 종료
        return (True, service_name, banner) # 성공적으로 연결되면 True와
    except ConnectionRefusedError:
        return ("Closed", service_name, None) # 연결이 거부되었을 때 "Closed"로 반환
    except Exception as e:
        return (None, service_name, None) # 그 외 예외 발생 시 None 반환

def port_scanner(host, ports_to_scan):
    for port in ports_to_scan:
        # SYN 스캔 수행
        syn_result = SYN_scan(host, port)

        # Telnet 스캔 수행
        telnet_result, service_name, banner = Telnet_scan(host, port)

        if syn_result is True:
            print(f"Port {port} is open (SYN Scan)")
        elif syn_result is False:
            print(f"Port {port} is closed (SYN Scan)")

        if telnet_result is True:
            if banner:
                print(f"Port {port} is open for {service_name} - Banner: {banner}")
            else:
                print(f"Port {port} is open for {service_name} (Banner not available)")
        elif telnet_result == "Closed":
            print(f"Port {port} is closed for {service_name}")
        else:
            print(f"Port {port} status for {service_name} is unknown")

    if __name__ == "__main__":
        host = "192.168.0.48" # 스캔할 호스트 IP 주소
        ports_to_scan = [23] # 스캔할 포트 목록 (포트 23로 설정)

    port_scanner(host, ports_to_scan)

```



SMTP란?



전자 메일을 전송하는 데 사용되는 프로토콜

동작 방식

1. 연결 수립(Connection Establishment):

- SMTP 클라이언트는 전자 메일 서버에 TCP 연결을 시도합니다. 일반적으로 서버의 포트 25를 사용합니다.
- 클라이언트는 서버와의 연결을 수립하기 위해 TCP 핸드쉐이크를 수행합니다.

2. 인사와 인증(Greeting and Authentication):

- 서버와의 연결이 성공적으로 설정되면, 서버는 클라이언트에게 인사 메시지를 보냅니다. 클라이언트는 서버로부터 인사를 받고, 필요한 경우 인증을 위해 자격 증명을 제공합니다.

3. 메일 전송(Mail Submission):

- 클라이언트는 서버에게 전송할 메일을 제출합니다. 이 때, 메일의 송신자, 수신자, 제목, 본문 등의 정보가 포함됩니다.
- SMTP는 메일 전송 시 7비트 ASCII 문자 집합을 사용하며, 필요한 경우 이진 데이터를 Base64 혹은 인코딩 방식을 통해 전송할 수 있습니다.

4. 메일 전송 및 라우팅(Mail Transfer and Routing):

- 서버는 받은 메일을 수신자의 메일 서버로 전송합니다. 이 과정에서 메일 서버는 메시지의 수신자 메인을 확인하고, 해당 도메인에 대한 MX(Mail Exchange) 레코드를 찾아 라우팅을 수행합니다.
- 수신자의 메일 서버는 메일을 받고, 메일박스에 저장하거나 추가적인 처리를 수행합니다.

5. 응답과 연결 종료(Response and Connection Termination):

- SMTP 서버는 메일 전송 작업의 성공 또는 실패에 대한 응답을 클라이언트에게 전송합니다. 이 응답에는 전송이 성공했는지 여부와 관련된 정보가 포함됩니다.
- 클라이언트와 서버는 이후 연결을 종료할 수 있습니다. 종료 과정에서 클라이언트와 서버는 각각 FIN(Finish) 패킷을 교환하여 연결을 안전하게 종료합니다.

서비스 설치 및 동작 방식

```
#Postfix 설치:  
sudo apt update  
sudo apt install postfix
```

```
#Postfix 서비스 시작  
sudo systemctl start postfix
```

SMTP는 서비스를 설치하지 않더라도 포트는 기본적으로 열려 있음

구현

```

import socket
import telnetlib

def SYN_scan(host, port):
    try:
        # 소켓 생성
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(5) # 타임아웃 설정

        # SYN 패킷 전송
        result = sock.connect_ex((host, port))

        # 결과 확인
        if result == 0:
            return True # 포트가 열려있음
        else:
            return False # 포트가 닫혀있음
    except Exception as e:
        return None # 예외 발생 시 None 반환
    finally:
        sock.close()

def Telnet_scan(host, port):
    service_name = "Telnet"
    try:
        tn = telnetlib.Telnet(host, port, timeout=5) # Telnet 객체 생성
        banner = tn.read_until(b"\r\n", timeout=5).decode('utf-8') +
        tn.close() # 연결 종료
        return (True, service_name, banner) # 성공적으로 연결되면 True와 함께 서비스 이름과 빙너를 반환
    except ConnectionRefusedError:
        return ("Closed", service_name, None) # 연결이 거부되었을 때 "Closed" 문자열 반환
    except Exception as e:
        return (None, service_name, None) # 그 외 예외 발생 시 None 반환

def port_scanner(host, ports_to_scan):
    for port in ports_to_scan:
        # SYN 스캔 수행
        syn_result = SYN_scan(host, port)

        # Telnet 스캔 수행
        telnet_result, service_name, banner = Telnet_scan(host, port)

        if syn_result is True:
            print(f"Port {port} is open (SYN Scan)")
        elif syn_result is False:
            print(f"Port {port} is closed (SYN Scan)")

        if telnet_result is True:
            if banner:

```

```
        print(f"Port {port} is open for {service_name} - Banner: {banner}")
    else:
        print(f"Port {port} is open for {service_name} (Banner: {banner})")
    elif telnet_result == "Closed":
        print(f"Port {port} is closed for {service_name}")
    else:
        print(f"Port {port} status for {service_name} is unknown")

if __name__ == "__main__":
    host = "192.168.0.48" # 스캔할 호스트 IP 주소
    ports_to_scan = [23] # 스캔할 포트 목록 (포트 23로 설정)

    port_scanner(host, ports_to_scan)
```

▼ DNS - 53



DNS란?



호스트 이름을 IP 주소로 변환하거나 IP 주소를 호스트 이름으로 변환하는 데 사용되는 인터넷 프로토콜

동작 방식

1. 이름 해결 요청(Request):

- 클라이언트에서는 호스트 이름을 IP 주소로 해석하거나, IP 주소를 호스트 이름으로 변환해야 할 때 DNS 서버에 요청을 보냅니다.
- 이 요청은 일반적으로 UDP(User Datagram Protocol)를 사용하여 전송됩니다.

2. 도메인 쿼리(Query):

- DNS 클라이언트는 DNS 서버에 대한 쿼리를 보냅니다. 이 쿼리에는 호스트 이름에 대한 정보가 포함되어 있습니다.
- DNS 서버는 쿼리를 받고, 요청된 호스트 이름에 대한 IP 주소 또는 다른 관련 정보를 찾기 위해 작업을 시작합니다.

3. DNS 서버 검색:

- DNS 서버는 클라이언트의 요청을 수신하고, 자체 데이터베이스 또는 다른 DNS 서버에 대한 쿼리 실행하여 요청된 정보를 찾습니다.
- DNS 서버는 이 정보를 클라이언트에게 반환합니다. 이때 쿼리에 대한 응답이 캐시되어 이후의 동일한 요청에 대한 성능을 향상시킵니다.

4. 응답(Reply):

- DNS 서버는 호스트 이름에 대한 IP 주소 또는 다른 요청된 정보를 포함하는 응답을 클라이언트에게 보냅니다.
- 이 응답은 일반적으로 UDP 패킷에 포함되어 클라이언트에게 전송됩니다.
- 대부분의 응답은 캐시되어 클라이언트 또는 로컬 DNS 서버에 저장됩니다.

5. 오류 처리 및 재시도:

- 클라이언트는 DNS 서버로부터 응답을 받은 후 적절한 조치를 취합니다. 응답이 제대로 수신되지거나 오류가 발생하면 클라이언트는 다른 DNS 서버에 요청을 재시도할 수 있습니다.

서비스 설치 및 동작 방식

```
#bind 설치:  
sudo apt-get update  
sudo apt-get install bind9
```

'''설정 파일 수정:

BIND의 설정 파일은 일반적으로 /etc/bind 또는 /etc/named 디렉터리에 위치합니다.
DNS 영역 설정:

DNS 서버에는 최소한 하나 이상의 DNS 영역을 설정해야 합니다. 이를 통해 서버는 해당

```
#bind 서비스 시작  
sudo systemctl start bind9
```

DNS는 서비스를 설치하지 않더라도 포트는 기본적으로 열려 있음

구현

```
import socket  
  
def DNS_scan(host, port):  
    try:  
        # UDP 소켓 생성  
        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  
        sock.settimeout(1) # 타임아웃 설정  
  
        # DNS 서버에 데이터 전송  
        sock.sendto(b'', (host, port))  
  
        # 데이터 수신 시 포트가 열려 있다고 가정  
        # UDP 스캔은 응답이 없어도 포트가 열려 있다고 가정합니다.  
        return True  
    except Exception as e:  
        return False  
    finally:  
        sock.close()  
  
def port_scanner(host, port):  
    if DNS_scan(host, port):  
        print(f"Port {port} is open for DNS")  
    else:  
        print(f"Port {port} is closed for DNS")  
  
if __name__ == "__main__":  
    host = "192.168.0.48" # 스캔할 호스트 IP 주소  
    port_to_scan = 53 # 스캔할 포트 번호  
  
    port_scanner(host, port_to_scan)
```

▼ HTTP - 80



HTTP란?



TCP/IP 네트워크(인터넷)상의 장치가 파일을 전송할 때 사용하는 프로토콜

동작 방식(Active 모드)

- 클라이언트가 서버에 데이터를 요청하고, 요청은 HTTP 메서드(GET, POST 등)와 URL을 포함합니다
- 서버가 클라이언트의 요청에 응답하고, 응답은 상태 코드와 함께 클라이언트에게 데이터를 전송합니다
- 서버가 요청에 대한 처리 결과를 나타냅니다.
- HTTP는 기본적으로 상태를 유지하지 않고, 쿠키와 세션을 사용하여 상태를 유지할 수 있습니다.

서비스 설치 및 동작 방식

```
#apache2 설치:  
sudo apt-get update  
sudo apt-get install apache2  
  
#apache2 서비스 시작  
sudo systemctl start vsftpd
```

구현

```
import socket  
  
def port80_http(target_host, port):  
    response_data = {  
        'port': port,  
        'status': None,  
        'banner': None,  
        'error_message': None  
    }  
  
    try:  
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:  
            sock.settimeout(5)  
            result = sock.connect_ex((target_host, port))  
  
            if result == 0:  
                response_data['status'] = 'open'  
                http_request = b"HEAD / HTTP/1.1\r\nHost: " + target_host  
                sock.send(http_request)  
                response = b""  
                while b"\r\n\r\n" not in response:  
                    chunk = sock.recv(1024)  
                    if not chunk:  
                        break  
                    response += chunk  
    except Exception as e:  
        response_data['error_message'] = str(e)  
  
    return response_data
```

```
        break
        response += chunk

    banner = response.decode("utf-8").strip()
    response_data['banner'] = banner
else:
    response_data['status'] = 'closed'
except Exception as e:
    response_data['status'] = 'error'
    response_data['error_message'] = str(e)

return response_data
```

▼ POP3 - 110



POP3란?



TCP/IP 네트워크(인터넷)상의 장치가 파일을 전송할 때 사용하는 프로토콜

동작 방식(Active 모드)

1. 이메일 클라이언트가 POP3 서버에 TCP/IP 연결을 설정합니다.
2. 클라이언트는 서버에게 사용자 이름과 비밀번호를 제공하여 인증을 수행합니다.
3. 클라이언트는 서버로부터 이메일을 가져옵니다.
4. 서버는 클라이언트에게 이메일 목록을 제공하고, 클라이언트는 선택한 이메일을 다운로드합니다.
5. 클라이언트가 모든 작업을 마치고 POP3 연결을 종료 후 서버 연결이 종료됩니다.

서비스 설치 및 동작 방식

```
#Dovecot 설치:  
sudo apt-get update  
sudo apt-get install dovecot-pop3d  
  
#Dovecot 서비스 시작  
sudo systemctl start dovecot
```

구현

```
import socket  
  
def pop3_banner_grabbing(target_host, port):  
    response_data = {  
        'port': port,  
        'status': None,  
        'banner': None,  
        'error_message': None  
    }  
    try:  
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:  
            sock.settimeout(3)  
            sock.connect((target_host, port))  
            response = sock.recv(1024).decode('utf-8')  
            response_data['status'] = 'open'  
            response_data['banner'] = response.strip()  
    except socket.timeout:  
        response_data['status'] = 'no response'  
    except Exception as e:  
        response_data['status'] = 'error'  
        response_data['error_message'] = str(e)
```

```
return response_data
```

▼ NTP - 123



NTP(Network Time Protocol)란?



NTP는 네트워크에 연결된 컴퓨터 시스템의 시계를 밀리초 이하의 정밀도로 동기화하는 데 사용되는 프로토콜입니다. 이를 통해 시간에 민감한 애플리케이션과 데이터 로깅 시스템에서 정확한 타임 스탬프를 보장할 수 있습니다.

NTP는 누가 개발했을까?

NTP는 1980년대 초에 데이비드 L. 밀스(David L. Mills) 교수에 의해 개발. 초기 목적은 대학 캠퍼스 내의 다양한 시스템 간에 시간을 동기화하는 것. 이후 NTP는 인터넷을 통해 전 세계 컴퓨터들이 정확한 시간을 유할 수 있도록 확장.

NTP의 역사

- **NTPv0:** NTP의 첫 번째 버전(NTPv0)은 1985년에 RFC 958로 문서화. 이 초기 버전은 기본적인 시동기화 기능을 제공.
- **NTPv1:** NTP의 첫 공식 버전인 NTPv1은 1988년 RFC 1059에서 발표. 이 버전에서는 인증과 에러 검증 기능이 추가.
- **NTPv2:** NTPv2는 1989년 RFC 1119에서 소개, 분산 클라이언트-서버 모델을 도입.
- **NTPv3:** NTPv3는 1992년 RFC 1305에서 발표, 보안, 관리, 그리고 알고리즘 개선.
- **NTPv4:** NTPv4는 여전히 널리 사용되는 버전, 더 나은 성능과 확장성을 제공, RFC 5905에서 정의.

프로토콜의 구조

NTP 프로토콜은 클라이언트-서버 모델을 기반. 클라이언트는 서버에 현재 시간을 요청하고, 서버는 자신으로부터 시계 시간을 기반으로 응답. NTP 메시지는 다음과 같은 주요 필드로 구성:

- **Leap Indicator (LI):** 윤초 조정을 위한 필드.
- **Version Number (VN):** NTP 버전.
- **Mode:** 작업 모드(예: 클라이언트, 서버, 방송).
- **Stratum:** 시간 소스의 계층. Stratum 1은 가장 높은 정확도를 가진 시간 소스(예: 원자 시계)를 의미.
- **Poll Interval:** 클라이언트가 서버에 시간을 요청하는 간격.
- **Precision:** 시스템 시계의 정밀도.
- **Root Delay & Root Dispersion:** 시스템과 참조 시계 소스 간의 지연 및 변동.
- **Reference Identifier:** 참조 시계 소스의 식별자.
- **Reference Timestamp:** 마지막으로 시계가 설정된 시간.
- **Originate Timestamp:** 클라이언트가 요청을 보낸 시간.
- **Receive Timestamp:** 서버가 요청을 받은 시간.
- **Transmit Timestamp:** 서버가 응답을 보낸 시간.

NTP를 통한 시간 동기화의 정확도는 밀리초 단위 또는 그 이하로 매우 높다.

동작 방식(Active 모드)

- 시간 조회:** 클라이언트는 네트워크를 통해 하나 이상의 NTP 서버에 시간 정보를 요청합니다.
- 시간 계산:** 클라이언트는 네트워크 지연 시간을 고려하여 정확한 시간을 계산합니다. NTP는 클라이언트와 서버 간의 왕복 시간(RTT)을 측정하고, 이를 통해 시간을 조정합니다.
- 시간 동기화:** 클라이언트는 계산된 시간을 바탕으로 내부 시계를 조정하여 동기화합니다.

서비스 설치 및 동작 방식

```
# NTP 패키지 설치
sudo apt update
sudo apt install ntp

# NTP 서비스 시작
sudo systemctl enable ntp
sudo systemctl start ntp
```

struct 모듈:

- struct** 모듈은 바이트 문자열을 파이썬의 데이터 구조로 패킹하거나 언패킹하는 데 사용됩니다. 이 스크립트에서는 NTP 서버로부터 받은 응답(바이트 문자열)을 처리할 때 사용됩니다.
- struct.unpack('!12I', response)** 코드는 네트워크에서 받은 바이트 문자열(**response**)을 12개의 unsigned integer(**I**)로 언패킹합니다. **!**I***는 네트워크 바이트 오더(big-endian)를 나타냅니다.
- 이러한 방식으로 언패킹된 데이터 중 10번째 unsigned integer는 NTP 타임스탬프를 포함하고 있으므로 이를 Unix 시간으로 변환하는 데 사용됩니다.

time 모듈:

- time** 모듈은 시간 관련 함수를 제공합니다. 이 스크립트에서는 NTP 응답으로부터 얻은 시간 스탬프를 사람이 읽을 수 있는 형태로 변환하는 데 사용됩니다.
- time.ctime(t)** 함수는 초 단위의 시간(여기서는 **t**)을 받아서 날짜와 시간을 나타내는 문자열로 변환합니다. 이 문자열은 현지 시간대를 기준으로 합니다.
- t -= 2208988800** 코드는 NTP 타임스탬프에서 1900년부터 1970년까지의 초를 빼서 Unix epoch(1970년 1월 1일 자정 UTC 이후의 시간을 초 단위로 나타낸 것)으로 변환합니다. 이 변환은 NTP 시간이 1900년을 기준으로 하고 Unix 시간이 1970년을 기준으로 하기 때문에 필요합니다.

구현

```
import socket
import struct
import time

def port123_ntp(host, port=123, timeout=3):
    # NTP 메시지 포맷 설정 (모드 3 - 클라이언트 요청)
    message = '\x1b' + 47 * '\0'
```

```

# UDP 소켓 생성
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.settimeout(timeout)

try:
    # NTP 서버로 메시지 전송
    sock.sendto(message.encode('utf-8'), (host, port))

    # 서버 응답 대기
    response, _ = sock.recvfrom(1024)

    # 응답 받음 - 포트가 열려 있음
    print(f"NTP port {port} is open on {host}.")

    # NTP 응답 패킷 분석
    unpacked = struct.unpack('!B B B b 11I', response)
    stratum = unpacked[1]
    poll = unpacked[2]
    precision = unpacked[3]
    root_delay = unpacked[4]
    root_dispersion = unpacked[5]
    ref_id = unpacked[6]

    # 추가 정보 출력
    print(f"Stratum: {stratum}, Poll: {poll}, Precision: {precision}")
    print(f"Root Delay: {root_delay / 2**16} seconds, Root Dispersion: {root_dispersion}")
    print(f"Reference ID: {ref_id}")

    # NTP 타임스탬프 추출 및 변환 (응답 패킷의 40번째 바이트부터 8바이트)
    t = struct.unpack('!12I', response)[10]
    t -= 2208988800 # 1900년부터 1970년까지의 초를 빼서 Unix epoch으로
    # 현재 시간으로 변환
    print(f"Server time: {time.ctime(t)}")
    return True

except socket.timeout:
    # 타임아웃 - 응답 없음
    print(f"NTP port {port} did not respond on {host}.")
    return False
except Exception as e:
    # 기타 오류
    print(f"Error scanning NTP port {port} on {host}: {e}")
    return False
finally:
    sock.close()

target_host = 'pool.ntp.org' #'time.windows.com' 'pool.ntp.org' # 로컬
port123_ntp(target_host)

```

The screenshot shows the Microsoft Visual Studio Code interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Editor Area:** A large code editor window titled "Jupyter new". It contains Python code for NTP scanning, specifically "port123_ntp.py". The code includes imports for struct, socket, and time, along with various print statements and logic for handling responses from NTP servers.
- Terminal:** A terminal window at the bottom showing the execution of the script. The output includes the host being scanned ("pool.ntp.org"), the port ("123"), and the server's response time ("Sat Feb 3 18:45:54 2024").
- Bottom Status Bar:** Shows the file path ("C:\Users\VIP\Desktop\new\ntp.py"), line number ("Ln 54, Col 1"), and other status information.
- Sidebar:** Includes sections for Explorer, Outline, Timeline, and Problems.

▼ IMAP - 143, 993



IMAP란?



Internet Message Access Protocol
원격으로 이메일 메시지를 받고 관리하기 위한 프로토콜
993포트는 IMAP의 보안 프로토콜입니다.

동작 방식

1. `imaplib` 모듈을 사용하여 Client가 서버와 연결을 시도합니다
 - a. 143 - `imaplib.IMAP4` 를 사용하여 연결
 - b. 993 - `imaplib.IMAP4_SSL` SSL 연결
2. 연결을 성공하면 `welcome` 속성으로 배너정보를 가져와 출력합니다

구현

```
def scan_imap_port(host, port, timeout = 5):
    response_data = {'service':None,'port': port, 'state': 'closed'}

    try:
        if port == 993:
            imap_server = imaplib.IMAP4_SSL(host,port, timeout=timeout)
            response_data['service'] = 'IMAPS'
        elif port == 143:
            imap_server = imaplib.IMAP4(host,port, timeout=timeout)
            response_data['service'] = 'IMAP'
        else:
            response_data['service'] = 'Unknown Service'
        # 배너정보 가져오기
        banner_info = imap_server.welcome
        response_data['state'] = 'open'
        response_data['banner'] = str(banner_info)
        imap_server.logout()

    except imaplib.IMAP4.error as imap_error:
        response_data['state'] = 'error'
        response_data['error'] = str(imap_error)

    except Exception as e:
        response_data['state'] = 'error'
        response_data['error'] = str(e)

    return response_data
```

▼ SNMP - 161



SNMP란?



Simple Network Management Protocol 네트워크 관리를 위해 개발된 프로토콜로, 네트워크 장비(라우터, 스위치, 서버, 워크스테이션, 프린터, 모뎀 등)의 상태와 성능을 모니터링하고 관리하는데 사용하는 프로토콜

동작방식

1. 커뮤니티 문자열을 설정합니다 `public`
2. 정보를 식별하는데 사용되는 OID를 설정합니다(가져올 정보 설정)
3. GET요청을 생성하여 전송합니다
4. 요청에 대한 응답을 받아 출력합니다

서비스 설치 및 동작방식

```
#서버 시작  
sudo systemctl start snmpd  
  
#서버 정지  
sudo systemctl stop snmpd  
  
#서버 동작 확인  
sudo systemctl status snmpd
```

구현

```
#SNMP Agent 161포트  
  
from pysnmp.hlapi import *  
  
def SNMP_conn(host, port):  
  
    # SNMP 커뮤니티 문자열 설정  
    community = 'public'  
  
    # OID (Object Identifier) 설정  
    sysname_oid = ObjectIdentity('SNMPv2-MIB', 'sysName', 0)  
    sysdesc_oid = ObjectIdentity('SNMPv2-MIB', 'sysDescr', 0)  
  
    # SNMP 요청 생성  
    snmp_request = getCmd(  
        SnmpEngine(),  
        CommunityData(community),  
        UdpTransportTarget((host, port), timeout=30, retries=1), # 브로드캐스팅  
        ContextData(),  
        ObjectType(sysname_oid),
```

```

        ObjectType(sysdesc_oid)
    )

    # SNMP 요청 전송 및 응답 받기
    error_indication, error_status, error_index, var_binds = next(snr)

    # 에러 확인
    if error_indication:
        print(f"에러: {error_indication}")
    elif error_status:
        print(f"에러 상태: {error_status}")
    else:
        for var_bind in var_binds:
            if sysname_oid.isPrefixOf(var_bind[0]):
                print(f"시스템 이름: {var_bind[1].prettyPrint()}")
            elif sysdesc_oid.isPrefixOf(var_bind[0]):
                print(f"시스템 설명: {var_bind[1].prettyPrint()}")

    if __name__=='__main__':
        # SNMP 에이전트 및 포트 설정
        host = '127.0.0.1' # 에이전트의 주소
        port = 161 # SNMP 포트 (기본값은 161)

        SNMP_conn(host, port)

```

▼ LDAP - 389



LDAP란?



네트워크상에서 디렉터리 서비스에 접근하여 정보를 읽고 쓰는 데 사용하는 프로토콜

동작 방식

- 클라이언트가 LDAP 서버에 TCP/IP 연결을 설정하여 통신을 시작한다.
- 클라이언트가 LDAP 서버에 자격 증명을 제공하여 접속 권한을 인증한다.
- 클라이언트가 LDAP 서버에 디렉터리 정보를 요청하고 검색한다.
- 클라이언트가 LDAP 서버에 새로운 항목을 추가하거나 기존 항목을 수정 또는 삭제한다.
- LDAP 서버가 클라이언트의 요청에 대한 결과를 생성하여 반환한다.

서비스 설치 및 동작 방식

```
#OpenLDAP를 사용하여 LDAP 서비스 시작  
sudo systemctl start slapd
```

구현

```
def scan_ldap_port(host, port=389, timeout=1):  
    response_data = {  
        'service': 'LDAP/TCP',  
        'port': port,  
        'state': 'closed',  
    }  
  
    server = Server(host, port=port, get_info=ALL, connect_timeout=timeout)  
  
    try:  
        with Connection(server, auto_bind=True) as conn:  
            response_data['state'] = 'open'  
  
            if server.info:  
                ldap_versions = server.info.supported_ldap_versions  
                if ldap_versions:  
                    response_data['supported_ldap_versions'] = ldap_versions  
  
                naming_contexts = server.info.naming_contexts  
                if naming_contexts:  
                    response_data['naming_contexts'] = naming_contexts  
  
                Supported_SASL_mechanisms = server.info.supported_sasl_mechanisms  
                if Supported_SASL_mechanisms:  
                    response_data['Supported SASL mechanisms'] = Supported_SASL_mechanisms
```

```
except Exception as e:  
    response_data['error'] = f'LDAP Error: {e}'  
  
return response_data
```

▼ HTTPS - 443



HTTPS란?



보안이 강화된 HTTP 프로토콜로, SSL 또는 TLS를 사용하여 데이터 전송을 암호화하고 중간에서의 데이터 탈취나 조작을 방지하여 안전한 웹 통신을 제공하는 프로토콜

동작 방식

- 클라이언트가 HTTPS 프로토콜을 사용하여 웹 서버에 TCP/IP 연결을 설정하여 통신을 시작한다.
- 클라이언트와 서버 간의 통신은 SSL/TLS 암호화 프로토콜을 사용하여 보호되며, 서버는 클라이언트의 인증서를 확인하여 인증한다.
- 클라이언트가 HTTPS 요청을 보내고, 서버는 해당 요청을 수신하고 처리한다.
- 서버는 클라이언트의 요청에 대한 결과를 생성하여 암호화된 형태로 반환한다.
- 클라이언트와 서버 간의 통신이 완료되면 연결이 종료되고 TCP/IP 연결이 해제된다.

서비스 설치 및 동작 방식

```
#아파치의 SSL 모듈을 활성화  
sudo a2enmod ssl  
  
#아파치를 다시 시작하여 변경 사항을 적용  
sudo systemctl restart apache2
```

구현

```
def scan_https_port(host, port=443, timeout=5):  
    response_data = {'service': 'HTTPS/TCP', 'port': port, 'state': None}  
    urllib3.disable_warnings(InsecureRequestWarning)  
    url = f"https://{host}:{port}"  
  
    try:  
        response = requests.get(url, timeout=timeout, verify=False)  
  
        server_header = response.headers.get('Server', None)  
        if server_header:  
            response_data['banner'] = server_header  
        else:  
            response_data['error'] = 'Server header not found'  
    except requests.exceptions.Timeout:  
        response_data['error'] = 'Connection timed out'  
    except requests.exceptions.SSLError as e:  
        response_data['error'] = 'SSL Error: ' + str(e)  
    except requests.exceptions.RequestException as e:  
        response_data['error'] = 'Request Error: ' + str(e)
```

```
return response_data
```

▼ SMB - 445



SMB (Server Message Block Protocol)란?



SMB (Server Message Block) 프로토콜은 파일, 프린터, 직렬 포트 및 통신 리소스 등 네트워크 상의 다양한 리소스에 대한 액세스를 제공하는 네트워크 파일 공유 프로토콜입니다. SMB는 일반적으로 Microsoft Windows 네트워크에서 파일 및 프린터 공유를 위해 사용되며, Linux 및 macOS를 포함한 다른 운영 체제에서도 널리 지원됩니다.

SMB의 역사:

- 초기 버전:** SMB 프로토콜은 IBM에 의해 1980년대 중반에 개발되었습니다. 초기에는 PC 네트워크에 파일 및 프린터 공유를 위해 사용되었습니다. DOS 네트워크에서 파일 공유를 위해 설계되었습니다.
- Microsoft와의 발전:** Microsoft는 SMB 프로토콜을 채택하고 확장하여, Windows for Workgroup 및 이후 버전의 Windows에서 네트워크 파일 및 프린터 공유의 기본 프로토콜로 만들었습니다.
- CIFS:** 1990년대에 Microsoft는 SMB를 확장하여 "Common Internet File System" (CIFS)을 개발했습니다. CIFS는 인터넷을 통한 파일 공유를 목적으로 했으며, SMB 프로토콜의 향상된 버전으로 간주됩니다.
- SMB 2.0:** 2006년, Microsoft는 Windows Vista와 Windows Server 2008에 SMB 2.0을 도입했습니다. 이 버전은 성능 향상, 보안 강화, 그리고 프로토콜의 단순화를 목표로 했습니다.
- SMB 3.0:** SMB 프로토콜의 더욱 발전된 형태인 SMB 3.0은 Windows 8과 Windows Server 2012에서 처음 소개되었습니다. 이 버전은 더 나은 성능, 향상된 보안 기능, 그리고 클러스터링 및 가상화 지원 같은 새로운 기능을 제공했습니다.

SMB 프로토콜은 TCP/UDP 포트 445를 사용하여 통신. 이 포트는 Microsoft의 디렉터리 서비스와 통신하는 데 사용되며, Windows 2000 이후 버전에서는 NetBIOS 대신 SMB의 직접 전송에 주로 사용됩니다.

smb 프로토콜의 구조

메시지 형식:

- 헤더:** 모든 SMB 메시지는 헤더를 포함합니다. 헤더에는 프로토콜의 버전, 명령 유형, 상태 코드, 플래그 및 메시지에 대한 다양한 필드 등이 포함됩니다. 이는 메시지의 유형을 식별하고, 요청과 응답을 적절히 처리하기 위한 정보를 제공합니다.
- 파라미터:** 명령에 따라, 메시지에는 다양한 파라미터가 포함될 수 있습니다. 이 파라미터는 명령을 실행하는 데 필요한 추가 정보를 제공합니다.
- 데이터:** 메시지에 포함된 실제 데이터 또는 페이로드입니다. 이 부분에는 파일 데이터, 디렉터리 목록, 태 정보 등 실제로 전송하려는 정보가 포함됩니다.

명령 및 응답:

- SMB 프로토콜은 다양한 유형의 명령을 정의하며, 이를 통해 파일 열기, 읽기, 쓰기, 닫기, 디렉터리 생성 및 삭제, 파일 공유 등의 작업을 수행할 수 있습니다. 각 명령에 대한 응답은 요청을 처리한 결과를 포함합니다.

세션 및 인증:

- SMB 프로토콜은 네트워크 상의 서버와 클라이언트 간에 세션을 설정하여 통신합니다. 인증 과정을 통해 사용자의 접근 권한을 확인하고, 이후 세션을 통해 안전하게 데이터를 교환합니다.

보안:

- 초기 버전의 SMB는 보안 측면에서 취약점을 가지고 있었으나, SMB 2.x 및 SMB 3.x에서는 보안이 강화되었습니다. SMB 3.x에서는 암호화, 보안 연결, 더 나은 인증 메커니즘 등을 포함하여 데이터의 안정성을 보장합니다.

서비스 설치 및 동작 방식

```
# Samba 설치  
sudo apt update  
sudo apt install samba  
  
# Samba 서비스 설정 및 시작  
sudo systemctl enable smbd  
sudo systemctl start smbd
```

구현

```
import uuid  
from smbprotocol.connection import Connection  
from smbprotocol.session import Session  
  
def port445_smb(target_host):  
    try:  
        # SMB 연결을 위한 Connection 객체 생성  
        connection = Connection(uuid.uuid4(), target_host, 445)  
        connection.connect()  
  
        # Session 객체 생성 및 익명 로그인 시도  
        session = Session(connection, username="", password "")  
        session.connect()  
  
        # 연결 성공 시, SMB 서비스에 대한 정보 출력  
        negotiated_dialect = connection.dialect  
        print(f"SMB service is responding on {target_host}, port 445")  
        print(f"Negotiated SMB Protocol Dialect: {negotiated_dialect}")  
  
        # 추가 정보가 필요하면 여기서 추출  
        # tree모듈에서 treeconnect클래스로 공유목록 불러오려고 했으나 파라미터값  
  
        return True  
    except Exception as e:  
        print(f"Failed to connect to SMB service on {target_host}:445")  
        return False  
  
target_host = "127.0.0.1"  
  
port445_smb(target_host)
```

uuid란? "Universally Unique Identifier"의 약자로, 전 세계적으로 고유한 식별자를 생성하기 위한 표준입니다. Python의 uuid 모듈은 이러한 식별자를 생성하는 데 사용됩니다. UUID는 128비트의 길이를 가지며, 보통 32자리의 16진수로 표현되며, 이들 사이에는 일반적으로 4개의 하이픈이 포함됩니다 (예: 123e4567-e89b-12d3-a456-42614174000).

UUID는 다양한 목적으로 사용될 수 있지만, 주로 네트워크 상에서 객체를 고유하게 식별하는 데 사용됩니다. 예를 들어, 데이터베이스의 레코드, 네트워크 상의 서비스 인스턴스, 파일의 일부 등을 고유하게 식별하는 데 사용될 수 있습니다.

Python의 uuid 모듈은 여러 가지 방법으로 UUID를 생성할 수 있습니다:

`uuid1()`: 호스트의 MAC 주소와 현재 시각을 기반으로 UUID를 생성합니다. 이 방식은 UUID의 고유성을 보장하지만, MAC 주소를 기반으로 하므로 일부 프라이버시 문제를 일으킬 수 있습니다.

`uuid3(namespace, name)`: 지정된 네임스페이스와 이름(문자열)을 기반으로 MD5 해시를 사용하여 UUID를 생성합니다.

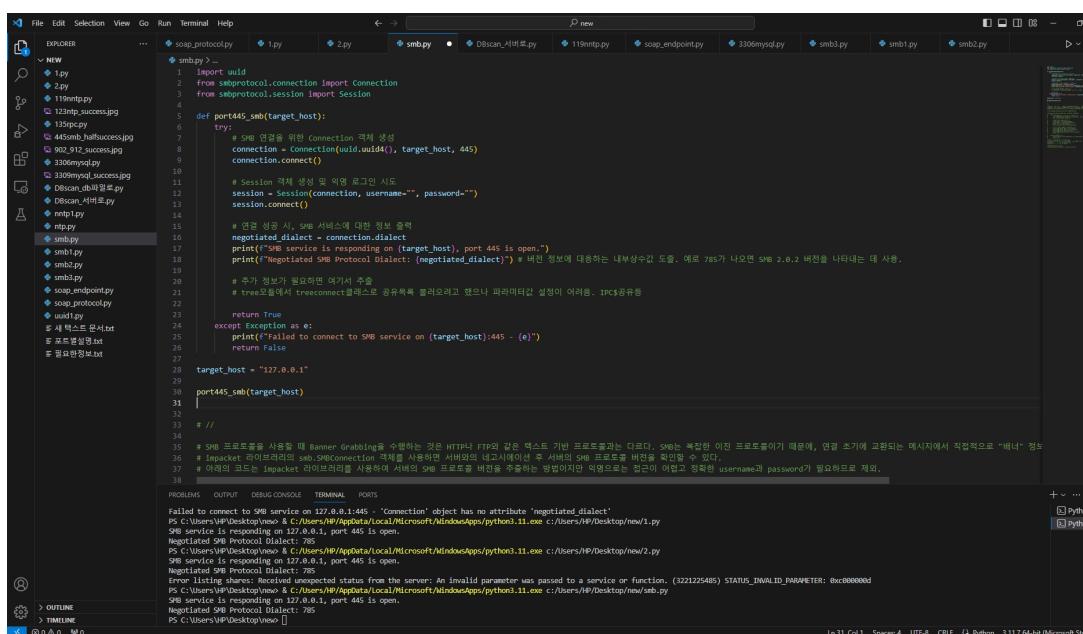
`uuid4()`: 무작위로 생성된 UUID입니다. 이 방식은 가장 일반적으로 사용되며, 고유성이 매우 높지만 완전히 무작위이기 때문에 충돌의 가능성은 이론적으로 존재합니다.

`uuid5(namespace, name)`: `uuid3()`과 유사하지만, SHA-1 해시를 사용하여 UUID를 생성합니다.

SMB 프로토콜을 사용할 때 Banner Grabbing을 수행하는 것은 HTTP나 FTP와 같은 텍스트 기반 프로토콜과는 다르다. SMB는 복잡한 이진 프로토콜이기 때문에, 연결 초기에 교환되는 메시지에서 직접적으로 "배너" 정보를 얻는 것은 표준적인 접근 방식이 아니고,

impacket 라이브러리의 `smb.SMBConnection` 객체를 사용하면 서버와의 네고시에이션 후 서버의 SMB 프로토콜 버전을 확인할 수 있다.

아래의 코드는 impacket 라이브러리를 사용하여 서버의 SMB 프로토콜 버전을 추출하는 방법이지만 익명으로는 접근이 어렵고 정확한 `username`과 `password`가 필요하므로 제외.



```

File Edit Selection View Go Run Terminal Help < - > new
EXPLORER ... smby.py l.py 2.py smby.py DScan_시비로.py 119http.py soap_endpoint.py 3206mysql.py smb3.py smb1.py smb2.py
1 import uuid
2 from smbprotocol.connection import Connection
3 from smbprotocol.session import Session
4
5 def port445_smb(target_host):
6     try:
7         # SMB 연결을 위한 Connection 객체 생성
8         connection = Connection(uuid.uuid4(), target_host, 445)
9         connection.connect()
10
11         # Session 객체 생성 및 익명 로그인 시도
12         session = Session(connection, username="", password="")
13         session.connect()
14
15         # 포트 성공 시, SMB 서비스에 대한 정보 출력
16         negotiated_dialect = connection.dialect
17         print(f"SMB service is responding on {target_host}, port 445 is open.")
18         print(f"Negotiated SMB Protocol Dialect: {negotiated_dialect}") # 버전 정보에 대응하는 내부수준값 표출. 예로 785가 나오면 SMB 2.0.2 버전을 나타내는 데 사용.
19
20         # 추가 정보가 필요하면 여기서 추가
21         # raw으로해서 tracemalloc을 래스터 공유체록 끌리으려고 했으나 파마미티갈 설정이 어려움. TPC$공유등
22
23         return True
24     except Exception as e:
25         print(f"Failed to connect to SMB service on {target_host}:445 - {e}")
26         return False
27
28 target_host = "127.0.0.1"
29
30 port445_smb(target_host)
31
32
33 # //
34
35 # SMB 프로토콜을 사용할 때 Banner Grabbing을 수행하는 것은 HTTP나 FTP와 같은 텍스트 기반 프로토콜과는 다르다. SMB는 복잡한 이진 프로토콜이기 때문에, 연결 초기에 교환되는 메시지에서 직접적으로 "배너" 정보를 얻는 것은 표준적인 접근 방식이 아니고,
36 # Impacket 라이브러리의 smb.SMBConnection 객체를 사용하면 서버와의 네고시에이션 후 서버의 SMB 프로토콜 버전을 확인할 수 있다.
37 # 아래의 코드는 impacket 라이브러리를 사용하여 서버의 SMB 프로토콜 버전을 추출하는 방법이지만 익명으로는 접근이 어렵고 정확한 username과 password가 필요하므로 제외.
38
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Failed to connect to SMB service on 127.0.0.1:445 - "Connection" object has no attribute 'negotiated_dialect'
PS C:\Users\HP\Desktop\new> & C:/Users/HP/AppData/Local/Microsoft/WindowsApps/python3.11.exe ::/Users/HP/Desktop/new/l.py
SMB service is responding on 127.0.0.1, port 445 is open.
Negotiated SMB Protocol Dialect: 785
PS C:\Users\HP\Desktop\new> & C:/Users/HP/AppData/Local/Microsoft/WindowsApps/python3.11.exe ::/Users/HP/Desktop/new/2.py
SMB service is responding on 127.0.0.1, port 445 is open.
Negotiated SMB Protocol Dialect: 785
Error listing shares: Received unexpected status from the server: An invalid parameter was passed to a service or function. (3221225485) STATUS_INVALID_PARAMETER: 0x8000000d
PS C:\Users\HP\Desktop\new> & C:/Users/HP/AppData/Local/Microsoft/WindowsApps/python3.11.exe ::/Users/HP/Desktop/new/smbs.py
SMB service is responding on 127.0.0.1, port 445 is open.
Negotiated SMB Protocol Dialect: 785
PS C:\Users\HP\Desktop\new>

```

▼ SMTP(submission) - 587



SMTP(submission)란?



이메일 클라이언트가 이메일을 메일 서버에 전송하는 데 사용되는 메일 전송 프로토콜. 이 프로토콜은 일반적으로 클라이언트와 서버 간의 통신에 보안 연결(SSL/TLS)을 사용

동작 방식

- 클라이언트 프로그램(예: 이메일 클라이언트)은 메일 서버에 전송할 이메일을 작성한다.
- 클라이언트는 SMTP Submission을 사용하여 메일 서버의 587번 포트로 연결을 시도한다. 이때 보안 결을 위해 SSL/TLS를 사용할 것인지 여부를 확인할 수 있다.
- 연결이 성공하면, 클라이언트는 메일 서버에 이메일의 내용과 수신자 목록을 전송한다.
- 메일 서버는 받은 이메일을 수신자들에게 전달하기 위해 필요한 추가 처리를 수행한다. 이는 수신자의 메일 주소 확인, 스팸 필터링, 메일 큐에 저장 등의 작업을 포함할 수 있다.
- 메일 서버가 수신자에게 이메일을 성공적으로 전달하면, 클라이언트에게 전송 완료 메시지를 반환한다

서비스 설치 및 동작 방식

```
#xrdp 패키지를 설치  
sudo apt update  
sudo apt install postfix  
  
#xrdp 서비스를 시작  
sudo systemctl restart postfix
```

구현

```
def scan_rdp_port(ip, port=3389, timeout=5):  
    response_data = {'service': 'RDP/TCP', 'port': port, 'state': 'closed'}  
  
    try:  
        sock = socket.create_connection((ip, port), timeout=timeout)  
        response_data['state'] = 'open'  
  
    except socket.timeout:  
        response_data['error'] = 'Connection timed out'  
    except socket.error as err:  
        response_data['state'] = 'closed or filtered'  
        response_data['error'] = str(err)  
    finally:  
        if 'sock' in locals():  
            sock.close()  
  
    return response_data
```

▼ MYSQL - 3306



MySQL란?



MySQL의 3306번 포트는 이 데이터베이스 관리 시스템을 위한 기본 TCP 포트.

MySQL은 1995년에 스웨덴의 회사인 TcX에 의해 처음 개발.

개발자인 Michael Widenius(별명 'Monty')와 David Axmark은 이 프로젝트를 시작.

3306번 포트는 MySQL의 표준 포트로 자리 잡게 되었다.

시간이 지남에 따라 다양한 버전과 포크가 개발. 가장 주목할 만한 포크 중 하나는 Oracle Corporation이 Sun Microsystems를 인수한 후에 발생한 MariaDB입니다.

mysql 프로토콜의 구조

MySQL 프로토콜은 클라이언트와 서버 간의 통신을 관리하는 복잡한 구조를 가지고 있습니다. 이 프로토콜은 여러 단계의 핸드셰이크와 데이터 교환 과정을 포함하여, 클라이언트가 서버에 연결하고 쿼리를 실행하며 결과를 받아볼 수 있도록 설계되었습니다. MySQL 프로토콜의 주요 구조와 단계

1. 연결 설정 (Connection Establishment)

- 클라이언트가 서버의 3306 포트에 TCP 연결을 시도합니다.
- 서버는 클라이언트에게 초기 핸드셰이크 패킷을 보내고, 이에는 서버의 버전, 스레드 ID, 인증 salt, 그리고 서버가 지원하는 기능들의 플래그 등이 포함됩니다.

2. 인증 및 핸드셰이크 (Authentication and Handshake)

- 클라이언트는 사용자 이름, 인증 방식, 인증 데이터(패스워드와 살트를 이용하여 암호화됨), 초기 데이터베이스 이름 등을 포함한 인증 응답 패킷을 서버로 보냅니다.
- 서버는 인증 데이터를 검증하고, 성공 또는 실패 응답을 클라이언트에게 보냅니다.

3. 명령 실행 (Command Phase)

- 인증이 성공하면, 클라이언트는 서버에 명령 패킷을 보낼 수 있습니다. 이 명령은 쿼리 실행, 데이터베이스 변경, 커넥션 설정 조회 등이 될 수 있습니다.
- 서버는 명령을 실행하고 결과를 클라이언트에게 보냅니다. 결과는 결과 세트, 오류 메시지, 상태 변경 등 다양한 형태가 될 수 있습니다.

4. 데이터 전송 (Data Transfer)

- 쿼리 결과(예: SELECT 명령의 결과)는 여러 개의 패킷으로 나뉘어 클라이언트에게 전송될 수 있습니다. 이때 각 패킷은 레코드의 한 행을 나타낼 수 있습니다.
- 또한, 서버는 결과 세트의 메타데이터(열 이름, 타입 등)도 함께 보냅니다.

5. 연결 종료 (Connection Termination)

- 클라이언트 또는 서버 양쪽에서 연결을 종료할 수 있습니다. 클라이언트가 연결을 종료하려면, 종료 명령을 서버로 보냅니다.
- 서버는 연결 종료를 확인하는 응답을 보내고, TCP 연결을 닫습니다.

MySQL 프로토콜의 각 패킷은 길이, 시퀀스 번호, 그리고 페이로드를 포함하는 헤더로 시작합니다. 이 프로토콜은 확장성과 유연성을 고려하여 설계되었으며, 다양한 인증 방식, 압축, SSL/TLS를 통한 암호화 등을 지원합니다.

서비스 설치 및 동작 방식

```
# MySQL 서버 설치
sudo apt update
sudo apt install mysql-server

# MySQL 서비스 보안 설정
sudo mysql_secure_installation

# MySQL 서비스 시작
sudo systemctl enable mysql
sudo systemctl start mysql
```

구현

```
import socket
import struct

def port3306_mysql(host, port=3306, timeout=5):
    try:
        # 소켓 객체 생성 및 타임아웃 설정
        socket.setdefaulttimeout(timeout)
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        # MySQL 서버의 3306 포트로 연결 시도
        s.connect((host, port))

        # 서버로부터 초기 핸드셰이크 메시지 수신
        packet = s.recv(1024)

        if packet:
            print(f"MySQL service is running on {host}:{port}")

            # 서버 버전 정보
            end_index = packet.find(b'\x00', 5)
            server_version = packet[5:end_index].decode('utf-8')

            # 스레드 ID / 클라이언트 연결에 고유한 id생성
            thread_id = struct.unpack('<I', packet[0:4])[0]
            #pid는 파일위치를 특정해야하므로 제외.

            # 서버 능력 설명, 비트마스크 형태, 비트가 설정되어있으면 기능을 지원함
            cap_low_bytes = struct.unpack('<H', packet[end_index + 1: end_index + 3])
            cap_high_bytes = struct.unpack('<H', packet[end_index + 3: end_index + 5])
            server_capabilities = (cap_high_bytes << 16) + cap_low_bytes[0]

            print(f"Server Version: {server_version}")
            print(f"Thread ID: {thread_id}")
            print(f"Server Capabilities: {server_capabilities:032b}")
```

```
        else:
            print(f"No response received from MySQL service on {host}")

    except socket.error as e:
        print(f"Error connecting to MySQL service on {host}:{port}: {e}")
    finally:
        s.close()

target_host = "127.0.0.1"

port3306_mysql(target_host)
```

▼ RDP - 3389



RDP란?



TCP 포트 3389를 통해 원격 데스크톱 연결을 가능하게 하여 클라이언트와 서버 간 통신 시 사용하는 프로토콜

동작 방식

1. 클라이언트는 RDP 프로토콜을 사용하여 원격 컴퓨터에 TCP/IP 연결을 설정하여 통신을 시작한다.
2. 클라이언트는 원격 컴퓨터에 로그인하기 위해 사용자 이름과 암호를 제공하여 인증을 수행한다.
3. 클라이언트는 원격 컴퓨터의 화면 정보를 수신하여 로컬 화면에 렌더링한다.
4. 클라이언트가 로컬에서 발생한 입력(마우스, 키보드 등)을 원격 컴퓨터로 전달하여 원격으로 제어할 수도록 한다.
5. 클라이언트와 원격 컴퓨터 간에 파일이나 기타 데이터를 전송하고, 원격 작업을 수행하기 위한 통신을 공한다.

서비스 설치 및 동작 방식

```
#xrdp 패키지를 설치  
sudo apt update  
sudo apt install xrdp  
  
#xrdp 서비스를 시작  
sudo systemctl start xrdp
```

구현

```
def scan_rdp_port(ip, port=3389, timeout=5):  
    response_data = {'service': 'RDP/TCP', 'port': port, 'state': 'closed'}  
  
    try:  
        sock = socket.create_connection((ip, port), timeout=timeout)  
        response_data['state'] = 'open'  
  
    except socket.timeout:  
        response_data['error'] = 'Connection timed out'  
    except socket.error as err:  
        response_data['state'] = 'closed or filtered'  
        response_data['error'] = str(err)  
    finally:  
        if 'sock' in locals():  
            sock.close()  
  
    return response_data
```

2.6 통합 포트 스캐너 코드

▼ main.py

```
import concurrent.futures
import time
from scan import *

def scan_all(host):
    scan_tasks = [
        (scan_ftp_ssh_port, {'port': 21}),
        (scan_ftp_ssh_port, {'port': 22}),
        (scan_telnet_port, {'port': 23}),
        (scan_smtp_port, {'port': 25}),
        (scan_dns_port, {'port': 53}),
        # (scan_tftp_server, {'port': 69}),
        (scan_http_port, {'port': 80}),
        (scan_pop3_port, {'port': 110}),
        #(scan_rpcbind_port, {'port': 111}),
        (scan_ntp_port, {'port': 123}),
        (scan_imap_port, {'port': 143}),
        (scan_snmp_port, {'port': 161}),
        (scan_ldap_port, {'port': 389}),
        (scan_https_port, {'port': 443}),
        (scan_smb_port, {'port': 445}),
        (scan_ssl_port, {'port': 465}),
        (scan_rip_port, {'port': 520}),
        (scan_smtp_submission_port, {'port': 587}),
        (scan_ssl_port, {'port': 636}),
        #(scan_vmware_port, {'port': 902}),
        (scan_imap_port, {'port': 993}),
        # (scan_pop3S_port, {'port': 995}),
        (scan_mysql_port, {'port': 3306}),
        (scan_rdp_port, {'port': 3389}),
        (scan_rsync_port, {'port': 135})
    ]

    results = []
    with concurrent.futures.ThreadPoolExecutor(max_workers=50) as executor:
        futures = {executor.submit(task, host, metadata['port']): metadata for task, metadata in scan_tasks}
        for future in concurrent.futures.as_completed(futures):
            metadata = futures[future]
            try:
                result = future.result()
                if 'closed' not in result.get('state', '').lower() and 'close':
                    results.append(result)
            except Exception as e:
                print(f"Error occurred while scanning port {metadata['port']} on host {host}: {e}")
    return results
```

```

        except Exception as e:
            error_result = {'port': metadata['port'], 'state': 'error', 'results.append(error_result)

sorted_results = sorted(results, key=lambda x: x['port'])
return sorted_results

if __name__ == "__main__":
    host = '127.0.0.1'
    startTime = time.time()
    scan_all(host)
    endTime = time.time()
    print("Executed Time:", (endTime - startTime))

```

▼ scan.py

```

# 수정사항
# servicename 출력하도록 수정하기
# 함수 이름 통합
# 443, 3389추가
# 중복 코드 병합 - ftp, ssh 통합/smtp, ldap 통합
# IMAP 시간 설정하기

import socket
import struct
import time
import uuid
import imaplib
import telnetlib
import ssl
from pysnmp.hlapi import *
from smbprotocol.connection import Connection
from scapy.all import sr, IP, TCP, UDP, ICMP, sr1
import requests
import urllib3
from urllib3.exceptions import InsecureRequestWarning
import dns.resolver
import dns.query
import dns.message
from ldap3 import Server, Connection, ALL

#SMTPS, HTTPS, LDAPS
def scan_ssl_port(ip, port):
    if port == 465:
        service_name = "SMTPS/TCP"
    elif port == 443:
        #     service_name = "HTTPS/TCP"
    elif port == 636:
        service_name = "LDAPS/TCP"
    else:

```

```

        service_name = "알 수 없는 서비스"

    response_data = {'service':service_name, 'port': port, 'state': 'closed'}
    if syn_scan(ip, port):
        try:
            context = ssl.create_default_context()
            with socket.create_connection((ip, port)) as sock:
                with context.wrap_socket(sock, server_hostname=ip) as ssock:
                    banner = ssock.recv(1024).decode('utf-8')
                    response_data.update({'state': 'open', 'banner': banner})
        except Exception as err:
            response_data.update({'state': 'closed or filtered', 'error': str(err)})
    else:
        response_data['state'] = 'closed or filtered'
    return response_data

#SMTP, LDAP
# def scan_smtp1_port(ip, port):
#     # if port == 25:
#         #     service_name = "SMTP"
#     # if port == 587:
#         #     service_name = "SMTP"
#     # elif port == 389:
#         #     service_name = "LDAP"
#     # else:
#         #     service_name = "알 수 없는 서비스"

#     response_data = {'service':service_name, 'port': port, 'state': 'closed'}

#     if syn_scan(ip, port):
#         try:
#             with socket.create_connection((ip, port), timeout=10) as connection:
#                 banner = connection.recv(1024).decode('utf-8')
#                 response_data.update({'state': 'open', 'banner': banner})
#         except socket.error as err:
#             response_data.update({'state': 'open but unable to receive banner'})
#         else:
#             response_data['state'] = 'closed or filtered'
#     return response_data

def syn_scan(ip, port):
    packet = IP(dst=ip)/TCP(dport=port, flags="S")
    # sr 함수는 (발송된 패킷, 받은 응답) 튜플의 리스트를 반환
    # 여기서는 받은 응답만 필요하므로, _ 를 사용해 발송된 패킷 부분을 무시
    ans, _ = sr(packet, timeout=2, verbose=0) # ans는 받은 응답 리스트
    for sent, received in ans:
        if received and received.haslayer(TCP):
            if received[TCP].flags & 0x12: # SYN-ACK 확인
                return True # 포트열림
            elif received[TCP].flags & 0x14: # RST-ACK 확인

```

```

        return False # 포트 닫힘
    return False # 응답없거나 다른에러

# def scan_udp_port(host, port):
#     #port = 520
#     response_data = {
#         'service': "UDP",
#         'port': port,
#         'state': 'open or filtered'
#     }
#     packet = IP(dst=host)/UDP(dport=port)
#     response = sr1(packet, timeout=3, verbose=0)

#     if response is None:
#         response_data['error'] = 'No response (possibly open or filtered).'
#     elif response.haslayer(ICMP):
#         if int(response.getlayer(ICMP).type) == 3 and int(response.getlayer
#             response_data['state'] = 'closed'
#         else:
#             response_data['error'] = f"ICMP message received (type: {respon
#     else:
#         response_data['error'] = 'Received unexpected response.'

#     return response_data

def scan_telnet_port(host, port):
    #port = 23
    response_data = {'service': "Telnet/TCP", 'port': port, 'state': 'closed'}

    try:
        tn = telnetlib.Telnet(host, port, timeout=5) # Telnet 객체 생성 및 서버
        banner = tn.read_until(b"\r\n", timeout=5).decode('utf-8').strip() #
        tn.close() # 연결 종료
        response_data['state'] = 'open'
        response_data['banner'] = banner
    except ConnectionRefusedError:
        response_data['error'] = '연결거부'
    except Exception as e:
        response_data['state'] = 'error'
        response_data['error'] = str(e)
    return response_data

#123 jo
def scan_ntp_port(host, port, timeout=1):
    message = '\x1b' + 47 * '\0'
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.settimeout(timeout)
    response_data = {}

```

```

        sock.sendto(message.encode('utf-8'), (host, port))
        response, _ = sock.recvfrom(1024)
        sock.close()

        unpacked = struct.unpack('!B B B b 1I', response)
        t = struct.unpack('!12I', response)[10] - 2208988800
        response_data = {
            'service': 'NTP/UDP',
            'port': port,
            'state': 'open',
            'stratum': unpacked[1],
            'poll': unpacked[2],
            'precision': unpacked[3],
            'root_delay': unpacked[4] / 2**16,
            'root_dispersion': unpacked[5] / 2**16,
            'ref_id': unpacked[6],
            'server_time': time.ctime(t)
        }
        return response_data

#445 jo
def scan_smb_port(host, port=445, timeout=1):
    response_data = {
        'service': 'SMB/TCP',
        'port': port,
        'state': 'closed',
        'negotiated_dialect': None,
        'error_message': None
    }

    try:
        connection = Connection(uuid.uuid4(), host, port)

        connection.connect(timeout=timeout)

        response_data['state'] = 'open'
        response_data['negotiated_dialect'] = str(connection.negotiated_diale

    except Exception as e:
        response_data['error_message'] = str(e)

    finally:
        if response_data['state'] == 'open':
            connection.disconnect()

    return response_data

#902 jo
# def scan_vmware_port(host, port=902, timeout=1):
#     response_data = {'service': 'VMWARE/TCP', 'port': port, 'state': 'close
#     #     sock = None

```

```

#     try:
#         sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
#         sock.settimeout(timeout)
#         sock.connect((host, port))

#         response = sock.recv(1024)

#         if response:
#             response_data['state'] = 'open'
#             try:
#                 response_data['banner'] = response.decode('utf-8').strip()
#             except UnicodeDecodeError:
#                 response_data['banner'] = response.hex()
#             else:
#                 response_data['state'] = 'no response'

#         except socket.timeout:
#             response_data['error_message'] = 'Connection timed out'
#         except socket.error as e:
#             response_data['state'] = 'error'
#             response_data['error_message'] = str(e)
#         finally:
#             if sock:
#                 sock.close()

#     return response_data

#3306 jo
def scan_mysql_port(host, port, timeout=1):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.settimeout(timeout)
    s.connect((host, port))
    packet = s.recv(1024)
    s.close()

    if packet:
        end_index = packet.find(b'\x00', 5)
        server_version = packet[5:end_index].decode('utf-8')
        thread_id = struct.unpack('<I', packet[0:4])[0]
        cap_low_bytes = struct.unpack('<H', packet[end_index + 1:end_index + 2])
        cap_high_bytes = struct.unpack('<H', packet[end_index + 19:end_index + 21])
        server_capabilities = (cap_high_bytes << 16) + cap_low_bytes
        response_data = {
            'service': 'MY SQL/TCP',
            'port': port,
            'state': 'open',
            'server_version': server_version,
            'thread_id': thread_id,
            'server_capabilities': f'{server_capabilities:032b}'
        }

```

```

        return response_data

def scan_imap_port(host, port, timeout = 5):
    response_data = {'service':'','port': port, 'state': 'closed'}
    original_timeout = socket.getdefaulttimeout()
    socket.setdefaulttimeout(timeout)

    try:
        if port == 993:
            imap_server = imaplib.IMAP4_SSL(host,port)
            response_data['service'] = 'IMAPS/TCP'
        else:
            imap_server = imaplib.IMAP4(host,port)
            response_data['service'] = 'IMAP/TCP'

        banner_info = imap_server.welcome
        response_data['state'] = 'open'
        response_data['banner'] = banner_info
        imap_server.logout()

    except imaplib.IMAP4.error as imap_error:
        response_data['state'] = 'error'
        response_data['error'] = imap_error

    except Exception as e:
        response_data['state'] = 'error'
        response_data['error'] = str(e)

    finally:
        socket.setdefaulttimeout(original_timeout)

    return response_data
#승희님 161
def scan_snmp_port(host, port):
    community = 'public'
    response_data = {'service':'SNMP/UDP', 'port': port, 'state': 'closed'}

    # OID 객체 생성
    sysname_oid = ObjectIdentity('SNMPv2-MIB', 'sysName', 0) #시스템 이름
    sysdesc_oid = ObjectIdentity('SNMPv2-MIB', 'sysDescr', 0) #시스템 설명 정보

    try:
        #SNMPD 요청 생성 및 응답
        snmp_request = getCmd(
            SnmpEngine(),
            CommunityData(community),
            UdpTransportTarget((host, port), timeout=2, retries=2),
            ContextData(),
            ObjectType(sysname_oid),

```

```

        ObjectType(sysdesc_oid)
    )

#요청에 대한 결과 추출
error_indication, error_status, error_index, var_binds = next(snmp_re

    if error_indication:
        response_data['state'] = 'error'
        response_data['error'] = str(error_indication)
    elif error_status:
        response_data['state'] = 'error'
        response_data['error'] = f'SNMP error state: {error_status.pretty
    else:
        response_data['state'] = 'open'
        for var_bind in var_binds:
            if sysname_oid.isPrefixOf(var_bind[0]):
                response_data['sysname'] = var_bind[1].prettyPrint()
            elif sysdesc_oid.isPrefixOf(var_bind[0]):
                response_data['sysinfo'] = var_bind[1].prettyPrint()

    except socket.timeout as timeout_error:
        response_data['state'] = 'error'
        response_data['error'] = timeout_error

    except socket.error as socket_error:
        response_data['state'] = 'error'
        response_data['error'] = socket_error

    except Exception as e:
        response_data['state'] = 'error'
        response_data['error'] = f'Unexpected error: {str(e)}'

    return response_data

```

```

# 영창님 21, 22 통합
def scan_ftp_ssh_port(host,port):
    if port == 21:
        service_name = 'FTP/TCP'
    elif port == 22:
        service_name = 'SSH/TCP'
    else:
        service_name = '알 수 없는 서비스'

    response_data = {'service':service_name,'port': port, 'state': 'closed'}

    try:
        # FTP 서버에 연결 시도
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

```

```

        sock.settimeout(5) # 연결 시도 시간 초과 설정
        result = sock.connect_ex((host, port))

        if result == 0:
            # 포트가 열려 있을 때
            banner = sock.recv(1024).decode('utf-8')
            response_data['state'] = 'open'
            response_data['banner'] = banner
        else:
            # 포트가 닫혀 있거나 필터링됐을 때
            response_data['state'] = 'closed'

        except socket.error as err:
            response_data['state'] = 'error'
            response_data['error'] = str(err)

        finally:
            # 소켓 닫기
            sock.close()

    return response_data

#다음님 80
def scan_http_port(target_host, port):
    response_data = {
        'service': 'HTTP/TCP',
        'port': port,
        'state': 'closed',
    }

    try:
        with socket.create_connection((target_host, port), timeout=5) as sock
            sock.sendall(b"HEAD / HTTP/1.1\r\nHost: " + target_host.encode())
            response = b""
            while b"\r\n\r\n" not in response:
                chunk = sock.recv(1024)
                if not chunk:
                    break
                response += chunk

            banner = response.decode("utf-8").strip()
            response_data['state'] = 'open'
            response_data['banner'] = banner
    except socket.timeout:
        response_data['state'] = 'timeout'
        response_data['error'] = 'Connection timed out'
    except socket.error as e:
        response_data['state'] = 'error'
        response_data['error'] = str(e)

```

```

        return response_data

#다솜님 110
def scan_pop3_port(target_host, port):
    response_data = {'service':'POP3/TCP', 'port': port, 'state': 'closed'}
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(3)
        sock.connect((target_host, port))
        response = sock.recv(1024).decode('utf-8')
        response_data['state'] = 'open'
        response_data['banner'] = response.strip()
    except socket.timeout:
        response_data['state'] = 'no response'
    except Exception as e:
        response_data['state'] = 'error'
        response_data['error'] = str(e)
    finally:
        if sock:
            sock.close()

    return response_data

# def scan_rdp_port(ip, port=3389):
#     response_data = {'port': port, 'state': 'closed', 'error': None}
#     if syn_scan(ip, port):
#         try:
#             # RDP 서버에 TCP 연결 시도
#             connection = socket.create_connection((ip, port), timeout=10)
#             response_data['state'] = 'open'
#             # RDP 서비스의 배너 정보를 직접 받는 것은 일반적이지 않으므로, 연결 성공 여부만 확인
#         except socket.error as err:
#             response_data['state'] = 'open but unable to connect'
#             response_data['error'] = str(err)
#         finally:
#             # 연결이 성공적으로 생성되었으면 종료
#             if 'connection' in locals():
#                 connection.close()
#     else:
#         response_data['state'] = 'closed or filtered'
#     return response_data

# 135
def scan_rsync_port(ip, port):
    response_data = {
        'port': port,
        'state': None,
        'banner': None,
        'error_message': None
    }

```

```

try:
    socket.setdefaulttimeout(3)
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((ip, port))
    response = s.recv(1024).decode('utf-8').strip()
    response_data['state'] = 'open'
    response_data['banner'] = response
except socket.timeout:
    response_data['state'] = 'closed'
except Exception as e:
    response_data['state'] = 'error'
    response_data['error_message'] = str(e)
finally:
    s.close() if 's' in locals() else None

return response_data

#443 jo
def scan_https_port(host, port=443, timeout=5):
    response_data = {'service': 'HTTPS/TCP', 'port': port, 'state': 'open', 'url': url}
    urllib3.disable_warnings(InsecureRequestWarning)
    url = f"https://:{host}:{port}"

    try:
        response = requests.get(url, timeout=timeout, verify=False)

        server_header = response.headers.get('Server', None)
        if server_header:
            response_data['banner'] = server_header
        else:
            response_data['error'] = 'Server header not found'
    except requests.exceptions.Timeout:
        response_data['error'] = 'Connection timed out'
    except requests.exceptions.SSLError as e:
        response_data['error'] = 'SSL Error: ' + str(e)
    except requests.exceptions.RequestException as e:
        response_data['error'] = 'Request Error: ' + str(e)

    return response_data

#53 jo
def scan_dns_port(host, timeout=5):
    response_data = {'service': 'DNS/TCP|UDP', 'port': 53, 'state': 'closed', 'rrsets': []}

    try:
        query = dns.message.make_query('version.bind', dns.rdatatype.TXT, dns.rdatatype.TXT)
        response = dns.query.udp(query, host, timeout=timeout)

        for rrset in response.answer:
            for txt in rrset:
                response_data['rrsets'].append(rrset.to_text())
    except dns.exception.DNSException as e:
        response_data['error'] = str(e)

    return response_data

```

```

        response_data['state'] = 'open'
        response_data['banner'] = txt.strings[0].decode('utf-8')
        break

    except (dns.exception.Timeout, dns.query.BadResponse, dns.query.UnexpectedResponse):
        response_data['error'] = f'DNS Query Error: {e}'
    except Exception as e:
        response_data['error'] = f'Error: {e}'

    return response_data
#389 jo
def scan_ldap_port(host, port=389, timeout=1):
    response_data = {
        'service': 'LDAP/TCP',
        'port': port,
        'state': 'closed',
    }

    server = Server(host, port=port, get_info=ALL, connect_timeout=timeout)

    try:
        with Connection(server, auto_bind=True) as conn:
            response_data['state'] = 'open'

            if server.info:
                ldap_versions = server.info.supported_ldap_versions
                if ldap_versions:
                    response_data['supported_ldap_versions'] = ldap_versions

            naming_contexts = server.info.naming_contexts
            if naming_contexts:
                response_data['naming_contexts'] = naming_contexts

            Supported_SASL_mechanisms = server.info.supported_sasl_mechanisms
            if Supported_SASL_mechanisms:
                response_data['Supported SASL mechanisms'] = Supported_SASL_mechanisms

    except Exception as e:
        response_data['error'] = f'LDAP Error: {e}'

    return response_data
#3389 jo
def scan_rdp_port(ip, port=3389, timeout=5):
    response_data = {'service': 'RDP/TCP', 'port': port, 'state': 'closed', 'version': 'Unknown'}

    try:
        sock = socket.create_connection((ip, port), timeout=timeout)
        response_data['state'] = 'open'

```

```

        except socket.timeout:
            response_data['error'] = 'Connection timed out'
        except socket.error as err:
            response_data['state'] = 'closed or filtered'
            response_data['error'] = str(err)
        finally:
            if 'sock' in locals():
                sock.close()

    return response_data

#520 jo
def scan_rip_port(host, port=520, timeout=3):
    response_data = {'service': 'RIP', 'port': port, 'state': 'closed or filt
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.settimeout(timeout)

    try:
        sock.sendto(b"Hello", (host, port))
        data, _ = sock.recvfrom(1024)
        response_data['state'] = 'open'
    except socket.timeout:
        response_data['error'] = 'No response (possibly open or filtered).'
    except Exception as e:
        response_data['error'] = str(e)
    finally:
        sock.close()

    return response_data

#69 jo
# def scan_tftp_server(host, port=69):
#     sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
#     message = b"\x00\x01fakefile\x00octet\x00"
#     response_data = {'service': 'TFTP/UDP', 'port': port, 'info': 'unknown
#
#     try:
#         sock.sendto(message, (host, port))
#         data, _ = sock.recvfrom(1024)

#         # TFTP 에러 패킷의 OpCode는 5, 에러 메시지는 패킷의 나머지 부분
#         if data[1] == 5:
#             error_message = data[4:].decode('utf-8').lower()
#             if 'netkit' in error_message:
#                 response_data['info'] = 'Netkit TFTP'
#             elif 'atftp' in error_message:
#                 response_data['info'] = 'atftp'
#             else:
#                 response_data['info'] = 'Other TFTP Server'

```

```

#     except socket.timeout:
#         print('No response from server')
#     except Exception as e:
#         print(f'Error: {e}')
#     finally:
#         sock.close()

#     return response_data

#111 jo
# def scan_rpcbind_port(host, port=111, timeout=3):
#     response_data = {'service': 'RPC/TCP', 'port': port, 'state': 'closed',
#
#     rpc_request = b'\x72\xFE\x1D\x13'
#
#     try:
#         with socket.create_connection((host, port), timeout=timeout) as soc
#             sock.sendall(rpc_request)
#             response = sock.recv(1024)
#
#             if response:
#                 response_data['state'] = 'open'
#                 response_data['details'] = 'Received RPC response'
#             else:
#                 response_data['details'] = 'No response from RPC service'
#
#         except socket.timeout:
#             response_data['details'] = 'Connection timed out'
#         except socket.error as e:
#             response_data['details'] = f'Socket Error: {e}'
#
#     return response_data

#995 jo
# def scan_pop3S_port(host, port=995, timeout=3):
#     response_data = {'service': 'POP3S/TCP', 'port': port, 'state': 'closed',
#
#     try:
#         with socket.create_connection((host, port), timeout=timeout) as soc
#             banner = sock.recv(1024).decode('utf-8').strip()
#             response_data['state'] = 'open'
#             response_data['banner'] = banner
#
#         except socket.timeout:
#             response_data['error'] = 'Connection timed out'
#         except socket.error as e:
#             response_data['error'] = 'Socket Error: ' + str(e)
#
#     return response_data

```

```

#25 jo
def scan_smtp_port(host, port=25, timeout=7):
    response_data = {'service': 'SMTP/TCP', 'port': port, 'state': 'closed',
                     }

    try:
        with socket.create_connection((host, port), timeout=timeout) as sock:
            sock.settimeout(timeout)
            banner = sock.recv(1024).decode('utf-8').strip()
            if banner:
                response_data['state'] = 'open'
                response_data['banner'] = banner
            else:
                response_data['details'] = 'No response from SMTP service'

    except socket.timeout:
        response_data['details'] = 'Connection timed out'
    except socket.error as e:
        response_data['details'] = f'Socket Error: {e}'

    return response_data

#587 jo
def scan_smtp_submission_port(host, port=587, timeout=7):
    response_data = {'service': 'SMTP-Submission/TCP', 'port': port, 'state': 'closed'}

    try:
        with socket.create_connection((host, port), timeout=timeout) as sock:
            sock.settimeout(timeout)
            banner = sock.recv(1024).decode('utf-8').strip()
            if banner:
                response_data['state'] = 'open'
                response_data['banner'] = banner
            else:
                response_data['details'] = 'No response from SMTP-Submission'

    except socket.timeout:
        response_data['details'] = 'Connection timed out'
    except socket.error as e:
        response_data['details'] = f'Socket Error: {e}'

    return response_data

```

2.7 Flask 구현 코드

▼ app.py

```

from flask import Flask, request, render_template
from main import scan_all
from redis import Redis

app = Flask(__name__)

```

```

@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        host = request.form.get('host')
        results = scan_all(host)
        return render_template('results.html', host=host, results=results)
    return render_template('index.html')

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8080, debug=True)

```

▼ index.html

```

<!-- index.html -->

<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>해커잡조 Port Scanner</title>
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='style.css') }}>
</head>
<body>
    <div class="container">
        <h1>해커잡조 Port Scanner</h1>
        <form action="/" method="post">
            <label for="host">Enter Hostname or IP:</label>
            <input type="text" id="host" name="host">
            <button type="submit">Scan</button>
        </form>
    </div>
</body>
</html>

```

▼ results.html

```

<!-- results.html -->

<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Scan Results</title>
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='style.css') }}>
</head>
<body>
    <h1>Scan Results for {{ host }}</h1>

```

```

{%- for result in results %}
    <div>
        <strong>Port {{ result.port }}:</strong> Status: {{ result.state }}
        {% for key, value in result.items() %}
            {% if key not in ['port', 'state'] %}
                , {{ key | capitalize | replace('_', ' ') }}: {{ value }}
            {% endif %}
        {% endfor %}
    </div>
{% endfor %}
<a href="/">Back to Scan</a>
</body>
</html>

```

▼ styles.css

```

body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f4;
    margin: 0;
    padding: 0;
}

.container {
    width: 50%;
    margin: 0 auto;
    text-align: center;
    padding-top: 50px;
}

h1 {
    color: #333;
}

form {
    margin-top: 20px;
}

label {
    display: block;
    margin-bottom: 5px;
}

input[type="text"] {
    width: 300px;
    padding: 8px;
    font-size: 16px;
    border: 1px solid #ccc;
    border-radius: 4px;
}

```

```
button[type="submit"] {
    padding: 10px 20px;
    font-size: 16px;
    background-color: #4CAF50;
    color: white;
    border: none;
    border-radius: 4px;
    cursor: pointer;
}

button[type="submit"]:hover {
    background-color: #45a049;
}
```

▼ results_styles.css

```
body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f4;
    margin: 0;
    padding: 0;
}

.container {
    width: 80%;
    margin: 0 auto;
    padding-top: 50px;
}

h1 {
    color: #333;
    text-align: center;
}

div {
    background-color: #fff;
    border: 1px solid #ccc;
    border-radius: 4px;
    margin-bottom: 20px;
    padding: 20px;
}

strong {
    color: #4CAF50;
}

a {
    display: block;
    text-align: center;
```

```

margin-top: 20px;
color: #007bff;
text-decoration: none;
}

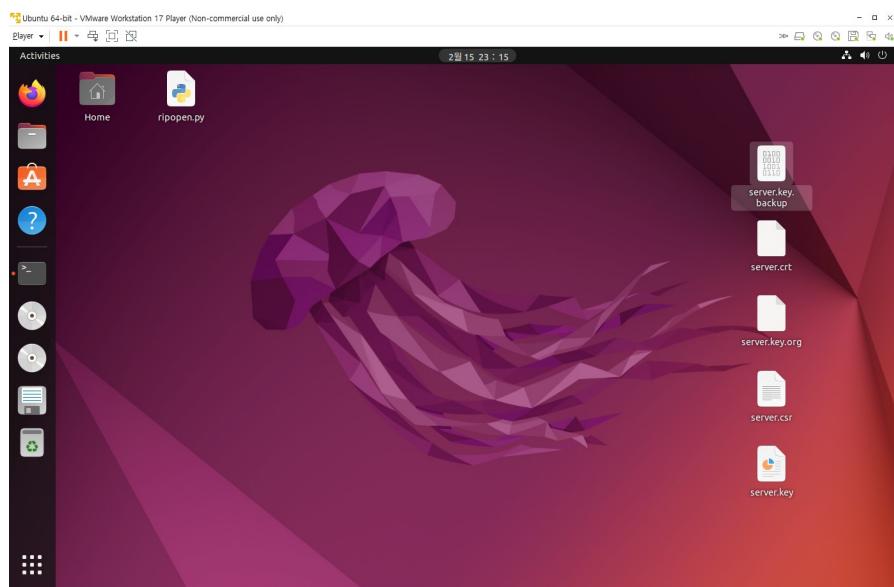
a:hover {
    text-decoration: underline;
}

```

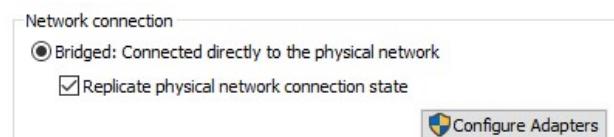
2.8 테스트 환경

▼ Test Server

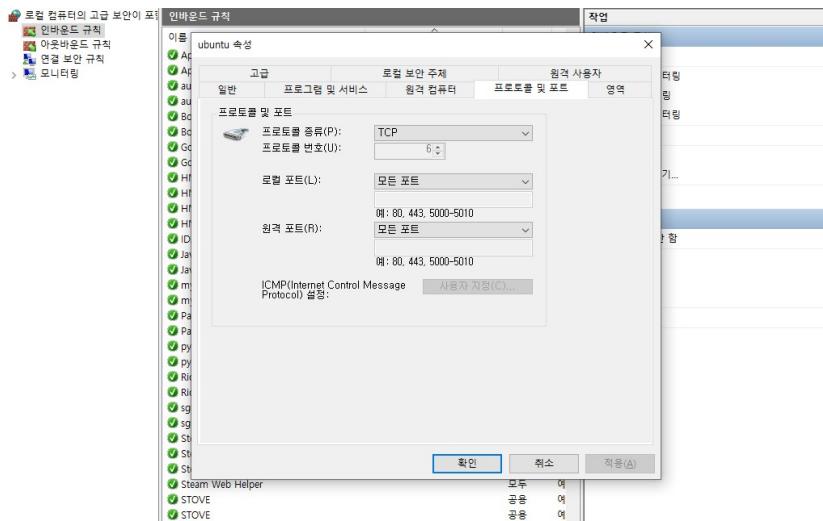
- VMware Workstation 17 Player를 활용하여 Ubuntu 22.04.1 LTS 설치



- 브릿지(Bridge)와 포트 포워딩(Port Forwarding)을 설정하여 외부에서 포트 스캔할 수 있도록 활성화



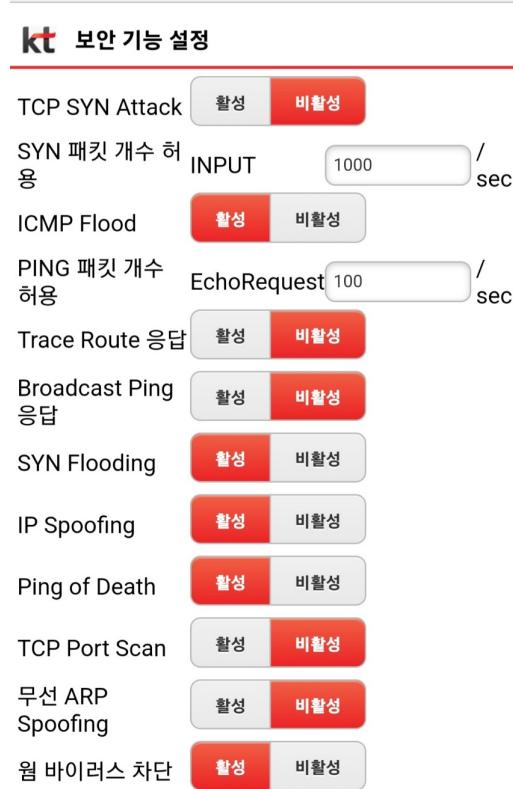
- 호스트 운영체제 방화벽 인바운드 아웃바운드 규칙 추가



- 가상 운영체제 방화벽 끄기

```
estar1230@estar1230-virtual-machine:~/Desktop$ sudo ufw disable
Firewall stopped and disabled on system startup
```

- 공유기 보안설정에서 **TCP SCAN** 및 **SNY SCAN 방지** OFF



- 팀원이 구현한 포트의 서비스를 설치

```
estar1230@estar1230-virtual-machine:~/Desktop$ sudo apt-get install snmpd
[sudo] password for estar1230:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
snmpd is already the newest version (5.9.1+dfsg-1ubuntu2.6).
0 upgraded, 0 newly installed, 0 to remove and 182 not upgraded.
```

- nmap으로 포트 open 확인

```
estar1230@estar1230-virtual-machine:~/Desktop$ sudo nmap -sU -p161
Starting Nmap 7.80 ( https://nmap.org ) at 2024-02-15 23:20 KST
Nmap scan report for estar1230-virtual-machine ( [REDACTED] )
Host is up (0.022s latency).

PORT      STATE SERVICE
161/udp  open   snmp

Nmap done: 1 IP address (1 host up) scanned in 0.25 seconds
```

- 포트가 close일 때 해당 서비스 conf파일 수정

```
# agentaddress: The IP address and port number that the agent will listen on.
# By default the agent listens to any and all traffic from any
# interface on the default SNMP port (161). This allows you to
# specify which address, interface, transport type and port(s) that you
# want the agent to listen on. Multiple definitions of this token
# are concatenated together (using ':'s).
# arguments: [transport:]port[@interface/address],...

agentaddress 0.0.0.0,[::1]
```

- 결과

```
estar1230@estar1230-virtual-machine:~/Desktop$ sudo nmap -p- -sT -sU
[sudo] password for estar1230:
Starting Nmap 7.80 ( https://nmap.org ) at 2024-02-15 22:01 KST
Nmap scan report for estar1230-virtual-machine ( [REDACTED] )
Host is up (0.00012s latency).
Not shown: 131042 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
25/tcp    open  smtp
53/tcp    open  domain
80/tcp    open  http
110/tcp   open  pop3
111/tcp   open  rpcbind
139/tcp   open  netbios-ssn
143/tcp   open  imap
389/tcp   open  ldap
443/tcp   open  https
445/tcp   open  microsoft-ds
587/tcp   open  submission
993/tcp   open  imaps
995/tcp   open  pop3s
3306/tcp  open  mysql
3389/tcp  open  ms-wbt-server
53/udp   open  domain
69/udp   open|filtered tftp
111/udp  open  rpcbind
123/udp  open  ntp
137/udp  open  netbios-ns
138/udp  open|filtered netbios-dgm
161/udp  open  snmp
631/udp  open|filtered ipp
5353/udp open  zeroconf
42937/udp open|filtered unknown

Nmap done: 1 IP address (1 host up) scanned in 6.13 seconds
```

▼ Flask Portscanner (구동 환경 : goorm IDE Flask 컨테이너)

Flask 설치

```
pip install flask
```

▼ 실행 화면

```
root@goorm:/workspace/Flask_Redis# pip install flask
Requirement already satisfied: flask in /usr/local/lib/python3.7/site-packages (1.1.2)
Requirement already satisfied: itsdangerous>=0.24 in /usr/local/lib/python3.7/site-packages (from flask) (1.1.0)
Requirement already satisfied: Werkzeug>=0.15 in /usr/local/lib/python3.7/site-packages (from flask) (1.0.1)
Requirement already satisfied: click>=5.1 in /usr/local/lib/python3.7/site-packages (from flask) (7.1.2)
Requirement already satisfied: Jinja2>=2.10.1 in /usr/local/lib/python3.7/site-packages (from flask) (2.1.2)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/site-packages (from Jinja2>=2.10.1->flask) (1.1.1)
WARNING: You are using pip version 20.2.4; however, version 23.3.2 is available.
You should consider upgrading via the 'curl https://pypi.org/simple/get-pip.py | python' command.
```

Redis 설치 및 설정

#터미널 창1

```
apt-get update

pip install redis

apt-get install redis-server

redis-server --version

vi /etc/redis/redis.conf

maxmemory 256mb
maxmemory-policy allkeys-lru

(./maxmemry로 검색 후 n 상단 내용 발견 후 i 입력하여 수정 > :wq 저장)
```

▼ 실행 화면

```
root@goorm:/workspace/Flask_Redis# apt-get update
[...]
W: An error occurred during the signature verification. The repository is not updated and the previous index files will be used. GPG error: https://cli-assets.heroku.com/apt ./ InRelease: 다음 서명들은 공개키가 없기 때문에 인증할 수 없습니다: NO_PUBKEY S36F8F1DE88F6A35
W: An error occurred during the signature verification. The repository is not updated and the previous index files will be used. GPG error: https://cf-cli-debian-repo.s3.amazonaws.com/stable/InRelease: 다음 서명이 솔바르지 않습니다: EXPKEYSIG 172B5989FC021EF8 CF CLI Team <cf-cli-eng@pivotal.io>
W: An error occurred during the signature verification. The repository is not updated and the previous index files will be used. GPG error: https://security.ubuntu.com/ubuntu/bionic-security/multiverse amd64 Packages [23.8 kB]
W: An error occurred during the signature verification. The repository is not updated and the previous index files will be used. GPG error: https://security.ubuntu.com/ubuntu/bionic-security/main amd64 Packages [3,373 kB]
W: An error occurred during the signature verification. The repository is not updated and the previous index files will be used. GPG error: https://security.ubuntu.com/ubuntu/bionic-security/restricted amd64 Packages [1,698 kB]
W: Some index files failed to download. They have been ignored, or old ones used instead.
```

```
root@goorm:/workspace/Flask_Redis# pip install redis
Collecting redis
  Downloading redis-5.0.1-py3-none-any.whl (250 kB)
    |██████████| 250 kB 13.9 MB/s
Requirement already satisfied: importlib-metadata>=1.0; python_version < "3.8" in /usr/local/lib/python3.7/site-packages (from redis) (2.0.0)
Requirement already satisfied: typing-extensions; python_version < "3.8" in /usr/local/lib/python3.7/site-packages (from redis) (3.7.4.3)
Collecting async-timeout>=4.0.2; python_full_version < "3.11.2"
  Downloading async_timeout-4.0.3-py3-none-any.whl (5.7 kB)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/site-packages (from importlib-metadata>=1.0; python_version < "3.8"->redis) (3.4.0)
Installing collected packages: async-timeout, redis
Successfully installed async-timeout-4.0.3 redis-5.0.1
WARNING: You are using pip version 20.2.4; however, version 23.3.2 is available.
You should consider upgrading via the '/usr/local/bin/python3.7 -m pip install --upgrade pip' command.
```

```
root@goorm:/workspace/Flask_Redis# apt-get install redis-server
페키지 목록을 읽는 중입니다... 완료
의존성 트리를 만드는 중입니다
상태 정보를 읽는 중입니다... 완료
다음의 주가 패키지가 설치될 것입니다 :
  libjemalloc redis-tools
제안하는 패키지 :
  ruby-redis
다음 *서로운* 패키지들을 설치 :
  libjemalloc redis-server redis-tools
0개 업그레이드, 3개 새로 설치, 0개 제거, 316개 업그레이드 안 함.
634 kB의 이트 아카이브를 받아야 합니다.
이 작업 후 3,012 kB의 디스크 공간을 더 사용하게 됩니다.
계속 하시겠습니까? [Y/n] Y
다운로드 완료: http://ap-northeast-2.ec2.archive.ubuntu.com/ubuntu bionic/universe amd64 libjemalloc amd64 3.6.0-11 [82.4 kB]
다운로드 완료: http://ap-northeast-2.ec2.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 redis-tools amd64 5:4.0.9-lubuntu0.2 [516 kB]
다운로드 완료: http://ap-northeast-2.ec2.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 redis-server amd64 5:4.0.9-lubuntu0.2 [35.4 kB]
2조에 634 kB 드롭드 (269 kB/s)
debconf: 프로그램을 설치할 수 없음: Dialog
debconf: (dialog나 x와 비슷한 프로그램을 설치하지 않았으므로, 다이얼로그 프론트엔드는 사용할 수 없습니다. at /usr/share/perl5/Debconf/FrontEnd/Dialog.pm line 76, < line 3.)
debconf: 다음 프론트엔드를 대신 사용: Readline
Selecting previously unselected package libjemalloc.
(데이터베이스 읽는중 ... 현재 97521개의 파일과 디렉터리가 설치되어 있습니다.)
Preparing to unpack .../libjemalloc_3.6.0-11_amd64.deb ...
Unpacking libjemalloc (3.6.0-11) ...
Selecting previously unselected package redis-tools.
Preparing to unpack .../redis-tools_5k3d4.0.9-lubuntu0.2_amd64.deb ...
Unpacking redis-tools (5:4.0.9-lubuntu0.2) ...
Selecting previously unselected package redis-server.
Preparing to unpack .../redis-server_5k3d4.0.9-lubuntu0.2_amd64.deb ...
Unpacking redis-server (5:4.0.9-lubuntu0.2) ...
libjemalloc (3.6.0-11) 설정하는 중입니다 ...
Processing triggers for libc-bin (2.27-3ubuntu1.29) ...
Processing triggers for systemd (237-3ubuntu16.29) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
redis-tools (5:4.0.9-lubuntu0.2) 설정하는 중입니다 ...
redis-server (5:4.0.9-lubuntu0.2) 설정하는 중입니다 ...
invoke-rc.d: could not determine current runlevel
invoke-rc.d: policy-rc.d denied execution of start.
Created symlink /etc/systemd/system/redis.service → /lib/systemd/system/redis-server.service.
Created symlink /etc/systemd/system/multi-user.target.wants/redis-server.service → /lib/systemd/system/redis-server.service.
```

```
root@goorm:/workspace/Flask_Redis# redis-server --version
Redis server v=4.0.9 sha=00000000:0 malloc=jemalloc-3.6.0 bits=64 build=9435c3c2879311f3
```

```
root@goorm:/workspace/Flask_Redis# vi /etc/redis/redis.conf
```

#터미널 창2

```
redis-server
```

▼ 실행 화면

```

root@goorm:/workspace/Flask_Redis# redis-server
1533:C 26 Jan 02:20:23.014 # o000o000o00o Redis is starting o000o000o000
1533:C 26 Jan 02:20:23.014 # Redis version=4.0.9, bits=64, commit=00000000, modified=0, pid=1533, just started
1533:C 26 Jan 02:20:23.014 # Warning: no config file specified, using the default config. In order to specify a config file use redis-server /path/to/redis.conf
1533:M 26 Jan 02:20:23.015 * Increased maximum number of open files to 10032 (it was originally set to 1024).

                               Redis 4.0.9 (00000000/0) 64 bit
                               Running in standalone mode
                               Port: 6379
                               PID: 1533

                               http://redis.io

1533:M 26 Jan 02:20:23.016 # Server initialized
1533:M 26 Jan 02:20:23.016 # WARNING overcommit_memory is set to 0! Background save may fail under low memory condition. To fix this issue add 'vm.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit_memory=1' for this to take effect.
1533:M 26 Jan 02:20:23.016 # WARNING you have Transparent Huge Pages (THP) support enabled in your kernel. This will create latency and memory usage issues with Redis. To fix this issue run the command 'echo never > /sys/kernel/mm/transparent_hugepage/enabled' as root, and add it to your /etc/rc.local in order to retain the setting after a reboot. Redis must be restarted after THP is disabled.
1533:M 26 Jan 02:20:23.016 * DB loaded from disk: 0.000 seconds
1533:M 26 Jan 02:20:23.016 * Ready to accept connections

```

#터미널 창1

```
netstat -nlpt | grep 6379
redis-cli
```

▼ 실행 화면

Python

#app.py

```

from flask import Flask, request, render_template
from main import scan_all
#from redis import Redis

app = Flask(__name__)

@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        host = request.form.get('host')
        results = scan_all(host)
        return render_template('results.html', host=host, results=results)
    return render_template('index.html')

```

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

CSS

#styles.css

```
body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f4;
    margin: 0;
    padding: 0;
}

.container {
    width: 50%;
    margin: 0 auto;
    text-align: center;
    padding-top: 50px;
}

h1 {
    color: #333;
}

form {
    margin-top: 20px;
}

label {
    display: block;
    margin-bottom: 5px;
}

input[type="text"] {
    width: 300px;
    padding: 8px;
    font-size: 16px;
    border: 1px solid #ccc;
    border-radius: 4px;
}

button[type="submit"] {
    padding: 10px 20px;
    font-size: 16px;
    background-color: #4CAF50;
    color: white;
    border: none;
    border-radius: 4px;
    cursor: pointer;
}
```

```
}
```

```
button[type="submit"]:hover {
```

```
    background-color: #45a049;
```

```
}
```

#result_styles.css

```
body {
```

```
    font-family: Arial, sans-serif;
```

```
    background-color: #f4f4f4;
```

```
    margin: 0;
```

```
    padding: 0;
```

```
}
```

```
.container {
```

```
    width: 80%;
```

```
    margin: 0 auto;
```

```
    padding-top: 50px;
```

```
}
```

```
h1 {
```

```
    color: #333;
```

```
    text-align: center;
```

```
}
```

```
div {
```

```
    background-color: #fff;
```

```
    border: 1px solid #ccc;
```

```
    border-radius: 4px;
```

```
    margin-bottom: 20px;
```

```
    padding: 20px;
```

```
}
```

```
strong {
```

```
    color: #4CAF50;
```

```
}
```

```
a {
```

```
    display: block;
```

```
    text-align: center;
```

```
    margin-top: 20px;
```

```
    color: #007bff;
```

```
    text-decoration: none;
```

```
}
```

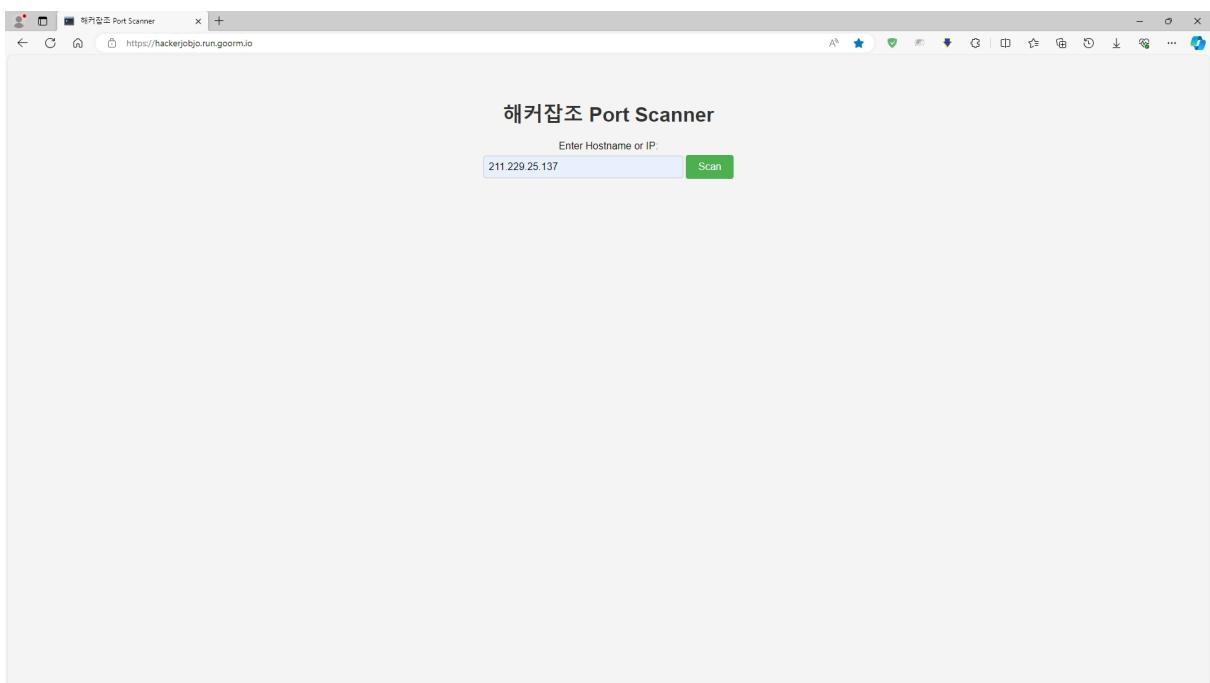
```
a:hover {
```

```
    text-decoration: underline;
```

```
}
```

2.9 최종 결과

2.9.1 스크린샷



Scan Results for 211.229.25.137	
Port 21:	Status: open , Service: FTP/TCP , Banner: 220 (vsFTPD 3.0.5)
Port 22:	Status: open , Service: SSH/TCP , Banner: SSH-2.0-OpenSSH_8_9p1 Ubuntu-3ubuntu0.6
Port 23:	Status: open , Service: Telnet/TCP , Banner: Ubuntu 22.04.1 LTS
Port 25:	Status: open , Service: SMTP/TCP , Banner: 220 estar1230-virtual-machine ESMTP Sendmail 8.15.2/8.15.2/Debian-22ubuntu3; Thu, 15 Feb 2024 10:28:15 +0900; (No UCE/UBE) logging access from: ec2-43-202-37-12.ap-northeast-2.compute.amazonaws.com(OK)-ec2-43-202-37-12.ap-northeast-2.compute.amazonaws.com[43.202.37.12]
Port 53:	Status: open , Service: DNS/TCP UDP , Banner: 9.18.18-0ubuntu0.22.04.1-Ubuntu
Port 80:	Status: open , Service: HTTP/TCP , Banner: HTTP/1.1 200 OK Date: Thu, 15 Feb 2024 01:28:10 GMT Server: Apache/2.4.52 (Ubuntu) Last-Modified: Thu, 08 Feb 2024 02:47:00 GMT ETag: "29af-610d5d35ba8ff" Accept-Ranges: bytes Content-Length: 10671 Vary: Accept-Encoding Content-Type: text/html
Port 110:	Status: open , Service: POP3/TCP , Banner: +OK Dovecot (Ubuntu) ready.
Port 123:	Status: open , Service: NTP/UDP , Stratum: 0 , Poll: 3 , Precision: -24 , Root delay: 0.0 , Root dispersion: 0.001983642578125 , Ref id: 1229867348 , Server time: Thu Feb 15 01:28:10 2024
Port 143:	Status: open , Service: IMAP/TCP , Banner: b" OK [CAPABILITY IMAP4rev1 SASL-IR LOGIN-REFERRALS ID ENABLE IDLE LITERAL+ STARTTLS LOGIN DISABLED] Dovecot (Ubuntu) ready."
Port 161:	Status: open , Service: SNMP/UDP , Sysname: estar1230-virtual-machine , Sysinfo: Linux estar1230-virtual-machine 6.5.0-17-generic #17~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Tue Jan 16 14:32:32 UTC 2 x86_64
Port 389:	Status: open , Service: LDAP/TCP , Supported ldap versions: ["3"], Naming contexts: ["dc=nodomain"], Supported sasl mechanisms: ["GSS-SPNEGO", "GSSAPI", "GS2-KRB5", "GS2-IAKERB", "SCRAM-SHA-256", "SCRAM-SHA-1", "DIGEST-MD5", "CRAM-MD5", "NTLM"]
Port 443:	Status: open , Service: HTTPS/TCP , Banner: Apache/2.4.52 (Ubuntu)
Port 587:	Status: open , Service: SMTP-Submission/TCP , Banner: 220 estar1230-virtual-machine ESMTP Sendmail 8.15.2/8.15.2/Debian-22ubuntu3; Thu, 15 Feb 2024 10:28:15 +0900; (No UCE/UBE) logging access from: ec2-43-202-37-12.ap-northeast-2.compute.amazonaws.com(OK)-ec2-43-202-37-12.ap-northeast-2.compute.amazonaws.com[43.202.37.12]
Port 993:	Status: open , Service: IMAPS/TCP , Banner: b" OK [CAPABILITY IMAP4rev1 SASL-IR LOGIN-REFERRALS ID ENABLE IDLE LITERAL+ AUTH=PLAIN] Dovecot (Ubuntu) ready."
Port 3306:	Status: open , Service: MY SQL/TCP , Server version: jHost 'ec2-43-202-37-12.ap-northeast-2.compute.amazonaws.com' is not allowed to connect to this MySQL server , Thread id: 110 , Server capabilities: 0x010101001100110000000011011100
Port 3389:	Status: open , Service: RDP/TCP , Error: None
Back to Scan	

2.9.2 최종 Release

해커잡조 Port Scanner

<https://hackerjobjo.run.goorm.io/>

final.zip

▼ 개별

app.py

main.py

option_total.py

readme.md

requirements.txt

scan.py

index.html

results.html

results_styles.css

styles.css

<https://prod-files-secure.s3.us-west-2.amazonaws.com/601f866a-fc0b-4e79-9803-30a518d349f4/e22b6857-8bf2-4d9b-9a4c-b5c2e9ae329a/HackerJobJO.mp4>

3. 회고

3.1 프로젝트 자체 평가 및 느낀 점

홍영창

이번 프로젝트를 통해 프로젝트팀장을 처음 맡게 되었는데 처음 해보는 프로젝트 파악에 어려움을 겪어 팀장으로서 어떻게 업무 분담을 시키고 이끌어 가야 하는지 고민이 많았고 잘되지 않았습니다. 하지만 먼저 프로젝트에 대해 파악하신 팀원분들께서 먼저 어떻게 하면 좋을지 제시해 주시고 능동적으로 맡은 일을 담당해 주셔서 프로젝트를 어떻게 진행하면 될지 파악하게 되었고 팀원분들의 도움으로 무사히 프로젝트를 마칠 수 있었습니다.

프로젝트 중반부터 노션페이지를 만들게 되어서 의도치 않게 노션 정리를 통한 팀원 작업 공유를 하지 않고 서로 각 업무를 알아서 프로젝트를 진행한 경험과 노션에서 각 업무를 Task화 시켜서 팀원 전체 작업 상황을 공유하면서 프로젝트를 진행하는 경험을 둘 다 하게 되었습니다.

그래서 노션을 통한 팀원들의 정보 및 작업 상황을 공유하는 중요성에 대해 다시 한번 깨닫는 계기가 되었습니다.

구현모

23 25 53 포트를 맡아서 진행을 했다.

처음엔 과제가 이해조차 되지 않아 커리큘럼에 있는 포트스캐너 강의를 들었으나 그럼에도 불구하고 정확하게 이해하지 못한 채 gpt의 도움을 받아 코드를 작성하고 그것을 수정해나가면서 포트 스캔을 작성하였고 vmware로 테스트할 환경을 구축하여 포트를 개방한 후 배너그래빙 출력을 진행하였다.

버벅였지만 다른 회차 분들이 한 것을 참고하여 배너그래빙을 출력하였고 해냈다는 뿌듯함에 기뻤다...

코드 통합 이후 flask로 웹페이지에서 스캐너 구현을 하기로 함에 따라 구름ide에서 flask로 웹을 구현하고 css로 디자인했다...

팀플은 처음이었지만 다들 도움을 주신 덕에 적당히 허둥거린 것 같다...
좋은 경험으로 나아가는 발판이 되지 않을까 싶다.

김동진

프로젝트 협업 경험이 많지 않아서 걱정했는데 팀원 각자 능동적으로 참여하셔서 프로젝트가 잘 진행되었던 것 같다. 처음에는 포트스캐너를 어떻게 구현해야하나 고민이 많았는데 나중에는 더 많은 포트나 옵션, 기능들을 구현하지 못한 것이 아쉽게 느껴졌다. 그리고 프로젝트를 진행하면서 어떤 것이 부족하고 필요한지 느끼게 되었다. 그래도 처음 목표했던 기능들이나 서비스들이 제대로 구현된 것 같아서 뿌듯하다.

장다솜

팀원들과 함께 프로젝트를 진행하면서 부족한 부분이 많았고 갈피를 못 잡는 상황에서도 함께 고민해주어서 너무 고마웠다.

처음 해보는 프로젝트라 개발하는 부분이 익숙하지 않았고 모르는 부분이 많아서 짐이 되는 것은 아닐까 생각했지만, 다른 부분에서 그 점을 채우려고 노력했다.

팀원들이 Notion과 Github에 정리한 내용을 보면서 문서 정리하기가 편리해졌고, Flow-chart를 작성하면서 개발 과정을 더 잘 이해할 수 있었다.

이후 프로젝트에는 나도 정리를 잘 해두어야겠다고 생각했다.

앞으로 남은 프로젝트에서 더 발전하여 팀에 이전보다 더 도움이 되는 팀원이 되고 싶다.

2조 해커잡조 화이팅!!

조운지

프로젝트를 진행하다 보면 무임승차 하는 인원이 있기 마련인데 한 명도 빠짐없이 자신이 맡은 일을 끝까지 해내는 모습을 보고 감동.

누가 시키지 않아도 먼저 나서서 일거리를 맡는 팀원과 자기 시간 써가며 프로젝트에 열중하는 팀원을 보면서 우리 팀원들 정말 대단하다고 생각했다.

남은 프로젝트들도 화이팅!

최승희

이론으로 배우기만 한 것과 직접 이론을 바탕으로 구현을 하는 것은 많이 다르다는 것을 느꼈습니다.

이런 프로젝트가 낯설기에 시작부터 막막했지만 이끌어주는 팀원들이 있어 결과를 낼 수 있었던 것 같습니다. 좋은 팀원들을 만난 것에 감사합니다. 또한 직접 포트 스캐너를 구현해봄으로써 이론으로 배웠을 때는 와닿지 않았던 네트워크 취약점, 보안이라는 것에 대해서 조금 더 체감할 수 있었습니다. 좋은 경험을 쌓은 것 같아 감사합니다.

4. 참고 문헌

SNMP란 무엇이며 네트워크 성능을 모니터링하는 데 어떻게 도움이 됩니까?

SNMP란 무엇입니까? SNMP(Simple Network Management Protocol)는 네트워크 관리에 도움이 되는 귀중한 네트워크 프로토콜입니다.

☞ <https://fiberroad.com/ko/resources/glossary/what-is-snmp/>



- 2024-01-25에 업데이트 된 Port에 관한 IANA의 공식 문서

Service Name and Transport Protocol Port Number Registry

☞ <https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml?=&skey=-2&page=1>

Nmap 모든 옵션/스캔 방식 종 정리(설명과 예시)

Nmap 모든 옵션/스캔 방식 종 정리(설명과 예시) 1. 대상 사양(TARGET SPECIFICATION) 옵션 설명 예시 -iL 스캔할 대상을 파일에서 가져옴 nmap -iL /hagsig/scanlist.txt -iR 지정한 숫자만큼 무작위 대상을 스캔 nmap -iR 50 --exclude 특정 대상을 제외하고 스캔 nmap --exclude

☞ <https://hagsig.tistory.com/94>

**Nmap
모든 옵션 정리**

[1부] 프로토콜 표준 스펙에서 정의한 Socket(소켓), Port(포트), TCP connection(연결) 개념
#socket #port #TCPconnection #UDP #네트워크 #쉬운코드 #백발백중

인터넷이나 네트워크 상에서 통신하기 위해서는 소켓과 포트, TCP 커넥션(connection)의 개념을 정확히
▶ <https://www.youtube.com/watch?v=X73Jl2nsqiE>



네트워크 스캐닝 : TCP Open, SYN, FIN, NULL, XMAS Scan 개념, 실습(Nmap + Wireshark)
* 국내에서의 스캐닝은 불법입니다. 절대로 공개되어 있는 IP/도메인에 대해 스캐닝을 하시면 안 됩니다....

▶ https://blog.naver.com/is_king/221573970526

