Zero Trust 기반 원격근무 보안강화 최종발표

유용상, 장예나, 한원표

목차

연구 배경

연구 개요

코로나 팬데믹 이후 원격근무 보편화로 **기업 네트워크 외부에서의 접속** 증가
이에 따른 기존 모델의 한계
=> 사용자, 장치, 위치의 맥락을 고려하여 접근을 매번 검증하는 방식 적용

이론적 배경

MFA 종류
- TOTP: Google Authenticator 등에서 생성되는 6자리 코드
- Push 인증: 로그인 시 등록된 기기로 승인 요청 전송

물리 계층 보안 방안
- 지리적 제약: 특정 장소에서만 원격 로그인 가능하도록 제한
- 허가된 디바이스 인증: TPM 활용해서 등록된 특정 하드웨어만 접속 허용

연구 목표

다중 인증 시스템 (MFA) 적용
- OTP, 생체인식, WebAuthn 등 다양한 인증 기술 비교 및 적용 실험
- MFA 적용 전/후 보안성 및 사용자 편의성 분석

이상 행위 탐지 시스템 구축
- 머신러닝 기반 로그인 패턴 분석
- 비정상 접속 탐지 및 자동 차단 시스템 구현

연구 진행

연구 접근방식

MFA 기술 중 **TOTP**와 **Push 인증** 기법 구현 + **물리 보안 계층** 개념

지리적, 시간적 의심 or 허가되지 않은 디바이스에서 접속 시 경고 표시

여러 경우의 수로 나누어 각각의 방식 비교
1. 기기 인증만 사용한 로그인
2. OTP 로그인
3. Push 인증 로그인
4. OTP + Push 인증 로그인
5. 인증 없는 일반 로그인

연구 접근방식

파일명	분류	특징
Base_login.py	일반 로그인	비밀번호만 검증 가능 -> 봇의 침투
Push_login.py	Push 인증	Push = yes 필요
Otp_login.py	OTP 인증	Otp = 123456 필요
Device_login.py	기기 인증	디바이스 ID 일치 필요
Push_otp_login.py	Push + OTP 인증	복합 MFA -> 높은 침투 차단률

연구 접근방식

성능평가

각 사이트별 로그인 실험 데이터를 CSV로 수집 => 그래프로 시각화

보안 효율성 측정: 무작위 봇 활용하여 존재하지 않는 아이디로 로그인 시도

응답 시간 측정: 실제 가입된 사용자 계정을 이용

1. 기기 인증 로그인 구현

```
function getOrCreateDeviceToken() {
    let token = localStorage.getItem('device_token');
    if (!token) {
        token = crypto?.randomUUID?.() || 'xxxxxxxx-xxxx-4xxx-yxxx-
xxxxxxxxxxxx'.replace(/[xy]/g, c => {
            const r = Math.random() * 16 | 0,
                v = c === 'x' ? r : (r & 0x3 | 0x8);
            return v.toString(16);
        });
        localStorage.setItem('device_token', token);
    }
    return token;
}
```

기기 토큰 생성

localStorage 통해서 각 기기, 브라우저마다
고유 토큰 생성

해당 토큰을 UUID(고유식별자)
생성기를 통해서 UUID로 변경

(+) UUID?
 => 네트워크 상에서 고유성 보장되는 id를
만들기 위한 표준 규약
128비트의 숫자이며, 32자리의 16진수로 표현함

실행 영상


https://push-otp-test.onrender.com/login
로그인
이메일:
비밀번호:
로그인
계정 생성하기
새로운 기기를 등록하시겠습니까?


push-otp-test.onrender.c
가가
로그인
이메일:
비밀번호:
로그인
계정 생성하기
새로운 기기를 등록하시겠습니까?

2. OTP 인증 로그인 구현

```
hashed_pw = bcrypt.hashpw(password.encode(), bcrypt.gensalt()).decode()
otp_secret = pyotp.random_base32()

otp_uri = pyotp.TOTP(otp_secret).provisioning_uri(name=email,
issuer_name="PushOTP")
img = qrcode.make(otp_uri)
buf = io.BytesIO()
img.save(buf, format='PNG')
qr_b64 = base64.b64encode(buf.getvalue()).decode('utf-8')
```

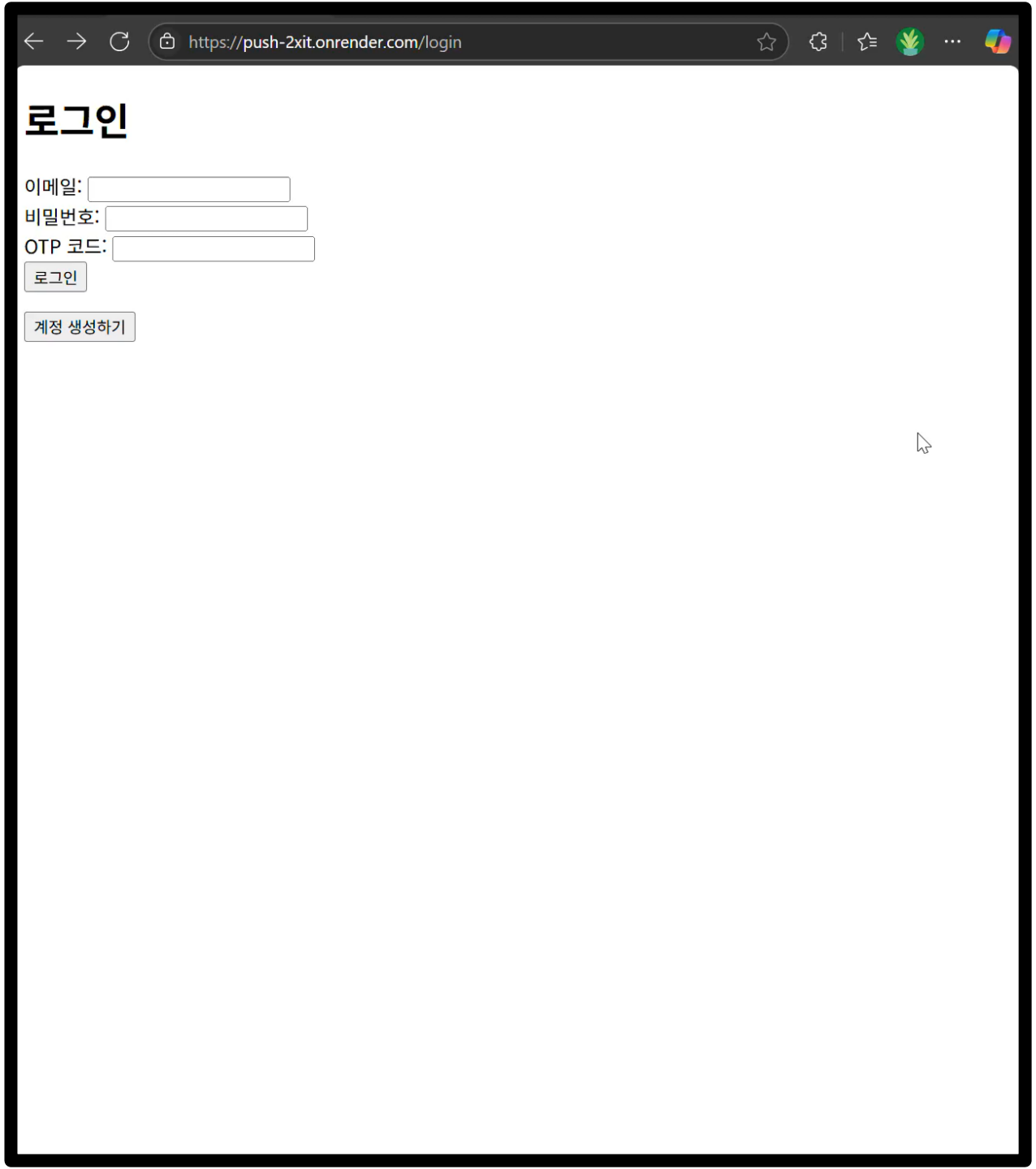
OTP 인증 코드 생성 및 QR 코드 변환
pyotp: OTP 생성·검증 라이브러리
=> 여기서는 (Time-based One-Time Password) 및 HOTP(HMAC-based One-Time Password)를 생성하고 검증

인증 완료된 계정에 OTP Secretkey 생성

회원가입에 사용된 사용자 email과 key를 활용하여 사용자마다 고유한 OTP 생성

구글 authenticator에 자동으로 등록시키는 URI: otpauth://totp/{issuer}:{email}?secret={secret}&issuer={issuer}
=> QR코드를 만들어 사용자가 구글 Authenticator 등 앱에 손쉽게 등록 가능

실행 영상


https://push-2xit.onrender.com/login
로그인
이메일:
비밀번호:
OTP 코드:
로그인
계정 생성하기

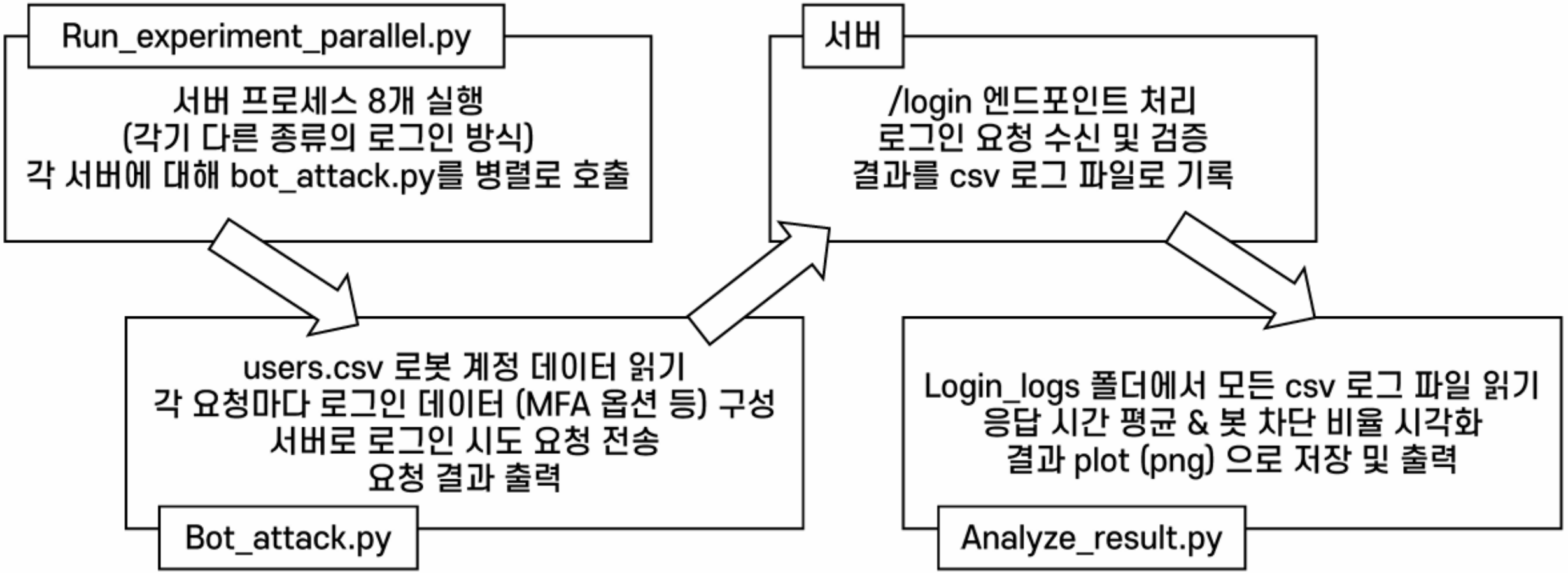
연구 결과

성능평가

소스코드	역할	설명
bot_attack.py	대규모 로그인 시도 수행하는 봇	동일한 형식으로 모든 서버에 공격
Run_experiments.py	봇 공격 수행	여러 서버 동시에 실행해서 병렬 공격 Multiprocessing.Pool 8개 동시에 bot_attack.py 실행 각 서버에 공격시도 10,000회 총 80,000번 로그인 시도
Analyze_result.py	실험 결과 로그 분석, 시각화	응답 시간 그래프 - 각 로그인 방식별 평균 응답 시간 (duration) 봇 차단률 그래프 - Success = False 비율로 위협 차단 성능 확인

성능평가

문제해결 알고리즘



성능평가

```python
# run_experiment_parallel.py의 병렬화 블록

with multiprocessing.Pool(processes=len(servers)) as pool:
    pool.map(run_attack, servers)
```

MFA 로그인 서버에 대규모 봇 공격 실행하는 코드
멀티코어 활용

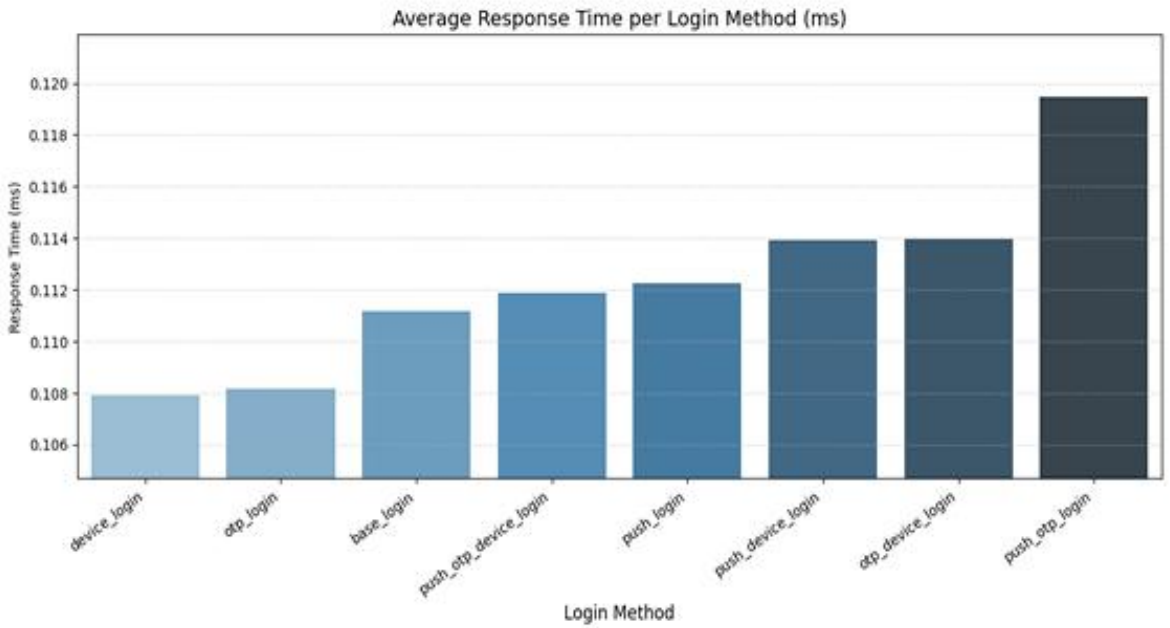
```python
# analyze_result.py의 차단률 시각화 블록

bot_df = df[df['is_bot'] == True]
if not bot_df.empty:
    bot_fail_rate = bot_df.groupby('source')['success'].apply(lambda x: 1
- x.mean())
    ...
```

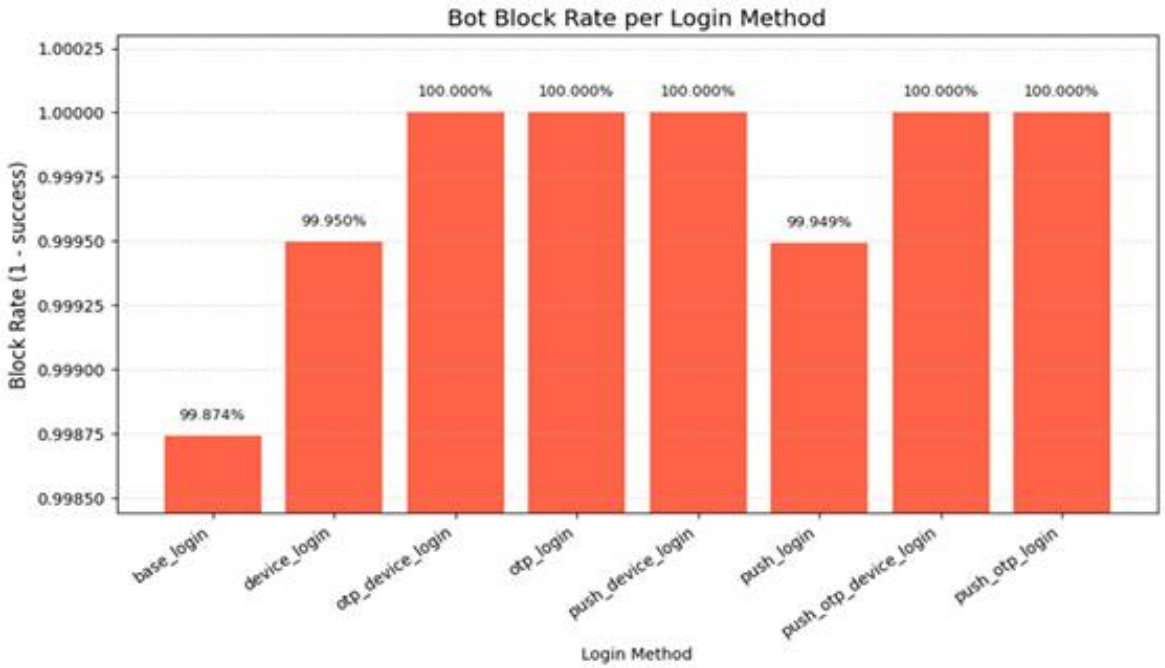
각 로그인 서버의 봇 공격 차단률 확인하는 코드
봇 공격 실험의 성능을 계산

성능평가

10,000 번 시도



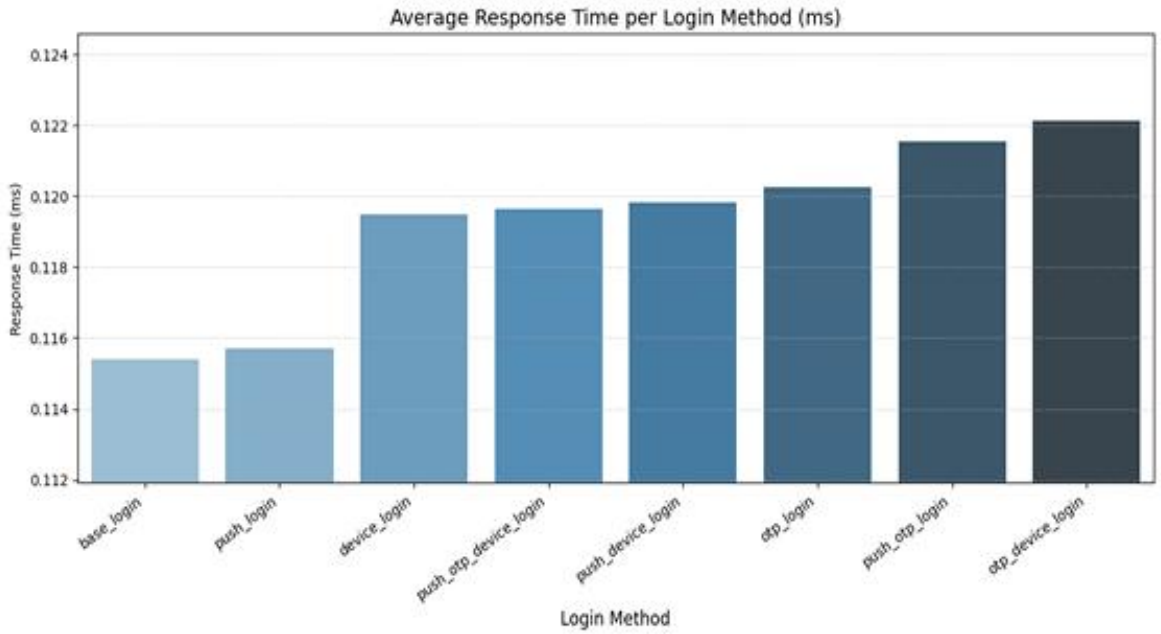
> 로그인 방식별 평균 응답 시간



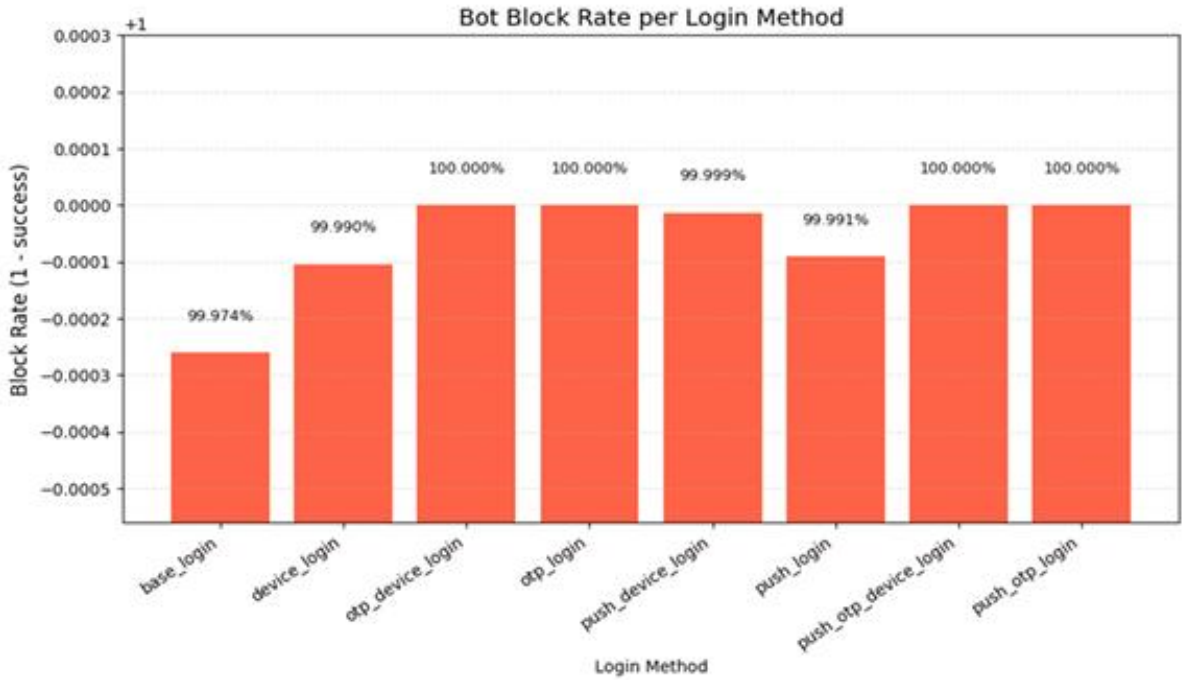
> 로그인 방식별 봇 차단률

성능평가

100,000 번 시도



> 로그인 방식별 평균 응답 시간



> 로그인 방식별 봇 차단률

결과 요약과 시사점

* 오름차순

응답 시간

일반 로그인 -> Push 인증 -> **기기 인증** -> Push + 기기 인증 ->
OTP 인증 -> **OTP + Push 인증** -> OTP + 기기 인증

봇 차단률

일반 로그인 -> **기기 인증** -> Push 인증 -> OTP + 기기 인증
/ Push + 기기 인증 / OTP / **OTP + Push 인증**

결과 요약과 시사점

Push + OTP 인증의 **보안 성능 강화** 효과 확인

그러나 보안 수준이 올라갈수록 **응답 시간이 증가**
Push +OTP 인증할 때 0.12ms로 일반 로그인 방식에 비해 10% 이상 느려짐

=> 높은 보안이 필요한 환경에서 Push + OTP 인증

=> 사용자 경험이 중요한 환경에서는 단일 기기 인증

한계점 및 개선방향

한계점 및 개선방향

1. 사이트 간 비교 실험 계획에서 반복적인 서버 문제 발생
(무료 서버가 공격 부하 감당을 못한 것으로 추측)

    => 실험 신뢰성 확보를 위해 로컬 컴퓨터에서 실험 진행함

2. 특정 환경 이외의 다른 네트워크에서도 같은 결과가 나오는지 재검증 필요

추후 개선 방향

1. 반복 실험 가능하도록 안정적인 서버 인프라 확보
2. 다양한 시스템 환경에서 시도

감사합니다