

IP 카메라 취약점 진단 시스템

팀명 | **SentriX**

팀원 | 전성배, 김수연, 송채연, 송재혁, 이동하

CONTENTS

01

프로젝트 개요

1. 프로젝트 주제 및 목적
2. 팀원 소개 및 역할

02

프로젝트 개발

1. 구상도
2. 개발 환경
3. 프로그램 구현

03

프로젝트 결과

1. 시연 영상
2. 기대효과

I 프로젝트 개요

1. 프로젝트 주제 및 목적

주제 선정 이유

스마트홈 기기 보급 증가에 따른 보안 위험(특히 IP 카메라) 존재
기본 계정·펌웨어 관리 소홀로 인한 사고 가능성 높음
전문가용 도구의 한계 → 일반 사용자용 실용적 진단 도구 개발 필요

목적

스마트홈 환경에서 광범위하게 활용되는 IP 카메라를 대상
사용자가 손쉽게 실행 가능한 로컬 중심의 취약점 진단 도구 설계 및 구현 → 실무에서 즉시 활용 가능한 보안 점검 체계 제안

본 시스템은 일반 사용자가 자신의 IP 카메라와 스마트홈 기기의 보안 상태를 직관적으로 확인 가능하도록 고안된 웹 기반 진단 도구

기기 연결 정보와 설정 바탕의 IoT 특화 취약점 자동 검사 + 발견된 취약점에 대해 이해하기 쉬운 설명과 단계별 권고안 제시
→ 사용자 스스로 보안 수준 개선에 도움

2. 팀원 소개 및 역할



전성배

팀장 및 총괄



김수연

서버 제작



송재혁

스크립트 작성



송채연

펌웨어 점검 수정

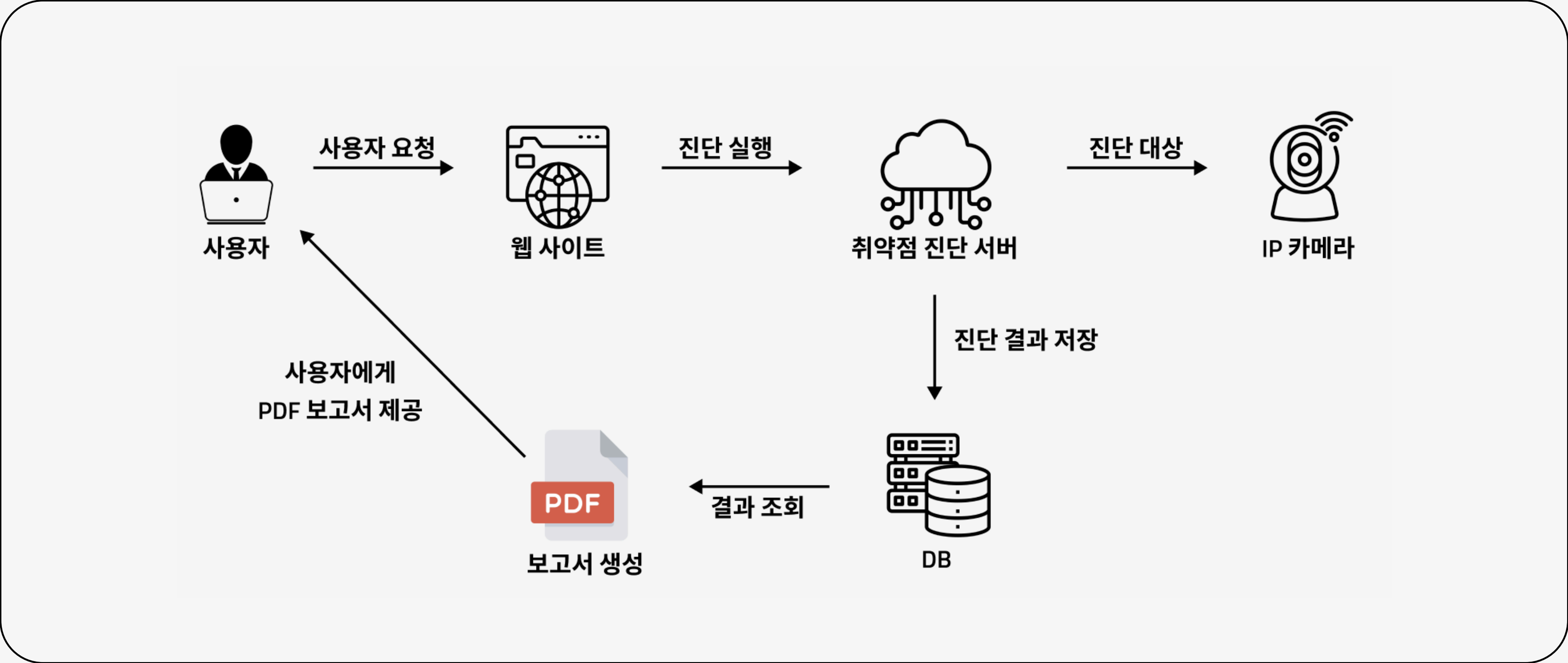


이동하

포트 점검

II 프로젝트 개발

1. 구상도



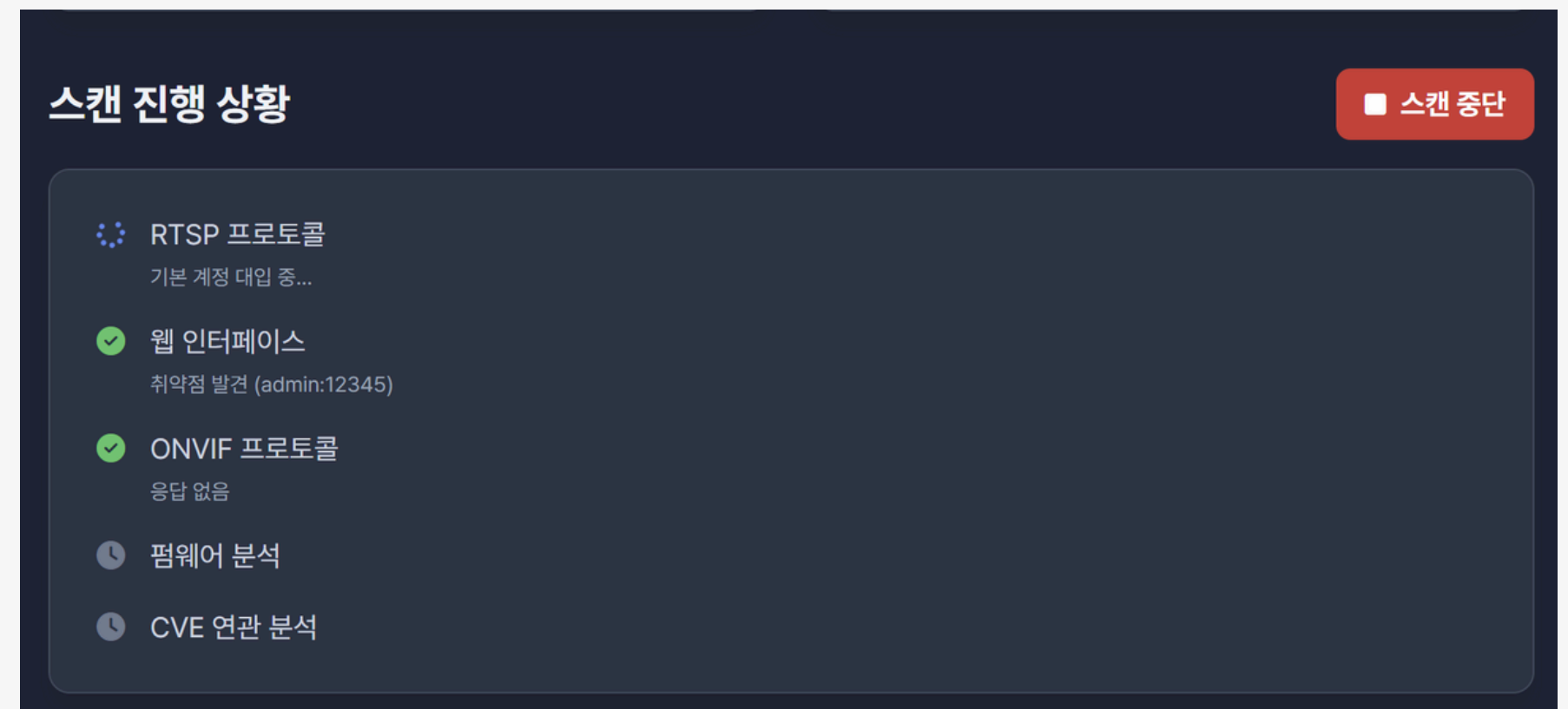
2. 개발 환경



3. 프로그램 구현 - 스캐너 모듈 구현

모듈화된 스캐너(ONVIF / RTSP / Web / Port / Firmware/CVE)로 기기별 검사 수행

- onvif_scanner
: 장비 정보·스트림 URI 수집
- r-tsp_scanner
: 썸네일 캡처·익명 접근 검증(FFmpeg 사용)
- web_scanner·port_scanner
: 웹 인증 취약·위험 포트 연결 여부 검사
- firmware_checker / cve_scanner
: 펌웨어 버전 비교 및 CVE 조회



3. 프로그램 구현 - 취약점 진단 흐름

ONVIF → RTSP/Web → 병렬 검사(Port / Firmware / CVE) → 통합·정규화 → CVSS 부여 → 보고서 생성

1단계: onvif_scanner로 장비·스트림 확보

여러 포트(8899,80,8080,5000)로 .../onvif/device_service 프로빙

유효 응답(200/401/500 또는 XML) 발견 시 base_url 설정

응답 내 UTCDateTime 파싱 → time_offset 보정(WSSE 인증용)

```
async def _probe_and_sync(self) -> bool:
    ports = [8899, 80, 8080, 5000]
    body = '<tds:GetSystemDateAndTime xmlns:tds="http://www.onvif.org/ver10/device/wsdl"/>'
    for p in ports:
        url = f"http://{self.ip}:{p}/onvif/device_service"
        await self.logger(f"Probing endpoint with a time sync request: {url}", "ONVIF")
        r = await self._soap(url, body, auth=False, action="GetSystemDateAndTime")
        if not r:
            continue
        if r.status_code in (200, 401, 500) or "xml" in r.headers.get("Content-Type", ""):
            self.base_url = url
            await self.logger(f"Valid ONVIF service detected at: {url}", "ONVIF")
            # 시간 파싱 (있으면)
            try:
                root = ET.fromstring(r.content)
                dt = root.find(".*{*}UTCDateTime")
                if dt is not None:
                    def g(tag): return int(dt.findtext(f"./*{*}{tag}", "0"))
                    cam = datetime(g("Year"), g("Month"), g("Day"), g("Hour"), g("Minute"), g("Second"), tzinfo=timezone.utc)
                    self.time_offset = datetime.now(timezone.utc) - cam
                    await self.logger(f"Time synchronized with camera. Offset: {self.time_offset}", "ONVIF")
            except Exception:
                pass
            return True
    await self.logger("Could not detect any valid ONVIF service.", "ONVIF")
    return False
```

3. 프로그램 구현 - 취약점 진단 흐름

2단계: rtsp_scanner·web_scanner 입력 → 실접근성·인증 취약점 확인

RTSP

ONVIF URI와 벤더 경로로 다수의 RTSP 후보를 생성하고 인증 정보는 user:pass@로 자동 삽입
썸네일 경로를 미리 만들고 tried_urls로 중복 차단, 시도 간 짧은 백오프로 안정화

WEB

80/8080 접속으로 웹 UI 존재 여부(200)만 먼저 확인

웹 발견 시 기본 계정 검사를 실행해 취약점 스캔을 우선화

```
import httpx
from typing import Callable, Coroutine

COMMON_PORTS = [80, 8080]
▼ DEFAULT_CREDS = [
    ('admin', 'admin'),
    ('admin', '1234'),
    ('admin', 'admin1234'),
    ('root', 'root'),
]

▼ class WebScanner:
▼     def __init__(self, ip: str, username: str, password: str, logger: Callable[[str, str], Coroutine], client: httpx.AsyncClient):
        self.ip = ip
        self.username = username
        self.password = password
        self.logger = logger
        self.client = client
        self.tag = "WEB"

▼     async def _check_default_creds(self, base_url: str):
        for user, pw in DEFAULT_CREDS:
            try:
                # Basic Auth를 사용하는 간단한 로그인 시도
                auth = httpx.BasicAuth(user, pw)
                # 일부 카메라는 로그인을 위해 특정 엔드포인트를 요구할 수 있음
                resp = await self.client.get(base_url, auth=auth, timeout=3, follow_redirects=True)
```

```
candidates: List[str] = []
if onvif_stream_uri:
    candidates.append(onvif_stream_uri)
if onvif_stream_path:
    candidates.append(f"rtsp://{self.ip}:554/{onvif_stream_path}")

candidates += [
    f"rtsp://{self.ip}:554/h264/ch1/main/av_stream",
    f"rtsp://{self.ip}:554/live/ch00_0",
    f"rtsp://{self.ip}:554/Streaming/Channels/101",
    f"rtsp://{self.ip}:554/Streaming/channels/101",
    f"rtsp://{self.ip}:554/live.sdp",
]

out_path = os.path.join(self.thumbs_dir, f"{self.thumb_name_prefix}_{self.ip.replace('.', '_')}.jpg")

tried_urls = set()

# 2) 기본(인증 포함/미포함) + tcp/udp
for base_url in candidates:
    # 인증 포함 URL
    url_with_auth = self._with_auth(base_url)
    urls_to_try = [url_with_auth, base_url] if self.username and self.password else [base_url]
```

3. 프로그램 구현 - 취약점 진단 흐름

3단계: port/firmware/cve 병렬 실행 → 통합(normalize_and_guidelines) → CVSS 적용

Port (포트 스캔)

비동기 연결로 FTP(21), Telnet(23) 등 불안전 포트의 열림 여부를 빠르게 확인
열려있으면 취약 항목으로 기록해 차단 권고 및 후속 조치로 연결

Firmware (펌웨어 검사)

장비의 제조사·모델·펌웨어 버전을 읽어 등록된 최신 버전과 비교
구버전이면 FIRMWARE_OUTDATED로 표시하고 업데이트 권고를 포함하여 보고

CVE (취약점 DB 조회)

제조사/모델명으로 CIRCL CVE DB를 조회해 관련 CVE 후보를 수집
CVSS 벡터/점수 추출로 자동 등급화하여 리포트 항목으로 변환

```
class PortScanner:
    # --- 중요 수정: 다른 스캐너와 동일한 표준 __init__ 구조로 변경 ---
    def __init__(self, ip: str, username: str, password: str, logger, client: httpx.AsyncClient):
        self.ip = ip
        self.logger = logger
        # username, password, client는 이 스캐너에서 현재 사용하지 않지만,
        # cross_scanner와의 호환성을 위해 인자로 받습니다.

    async def _check_port(self, port: int) -> bool:
        """지정된 포트가 열려 있는지 비동기적으로 확인합니다."""
        try:
            writer = await asyncio.wait_for(
                asyncio.open_connection(self.ip, port), timeout=2
```

```
# 실제 시스템에서는 외부 DB/설정으로 관리 권장
self.latest_versions = {
    "EFM networks": { "ipTIME-c200": "2.0.0" },
    "Default": "2.0.0",
}

async def scan(self, device_info: Dict[str, Any]) -> Dict[str, Any]:
    await self.logger(f"{self.ip} - 펌웨어 버전 검사를 시작합니다.", "FIRMWARE")
    items = []

    current_version = (device_info or {}).get("FirmwareVersion")
    manufacturer = (device_info or {}).get("Manufacturer")
    model = (device_info or {}).get("Model")

    if not current_version or not manufacturer or not model:
        await self.logger(f"{self.ip} - 펌웨어 버전 검사를 건너뜁니다 (필수 장치 정보 부족).", "FIRMWARE")
        return {"items": []}
```

```
def _vp(device_info: Dict[str, Any]) -> Dict[str, str]:
    manu = (device_info.get("Manufacturer") or "").strip().lower().replace(" ", "")
    model = (device_info.get("Model") or "").strip().lower().replace(" ", "")
    return {"vendor": manu, "product": model}

@staticmethod
def _pick_vector(cve: dict) -> Optional[str]:
    if isinstance(cve.get("cvss-vectors"), list) and len(cve["cvss-vectors"]) > 0:
        return cve["cvss-vectors"][-1].get("vector") # Use the latest vector
    if isinstance(cve.get("cvss3"), dict):
```

3. 프로그램 구현 - 결과 보고서 생성

IoT 보안 점검 보고서

대상 IP	192.168.219.101
제조사	EFM networks
모델	ipTIME-c200
펌웨어 버전	1.032
스캔 시각 (UTC)	2025-10-19 14:15:06Z

카메라 썸네일

2025-10-19 23:15:20

취약점 분석 평가표

No.	분류	점검항목	결과	비고
1	서버/장비 (계정 관리)	Default 계정 또는 추측 가능한 패스워드 사용	취약	기본 관리자 계정명(admin) 사용 중
2	서버/장비 (계정 관리)	계정 잠금 임계값 설정 미흡	양호	-

진단 결과 상세

구버전 펌웨어 사용

MEDIUM

설명: 장비 펌웨어가 최신(2.0.0)보다 낮은 버전(1.032)입니다.

조치: 제조사에서 제공하는 최신 펌웨어로 업데이트하세요.

기본 관리자 계정명(admin) 사용 중

LOW

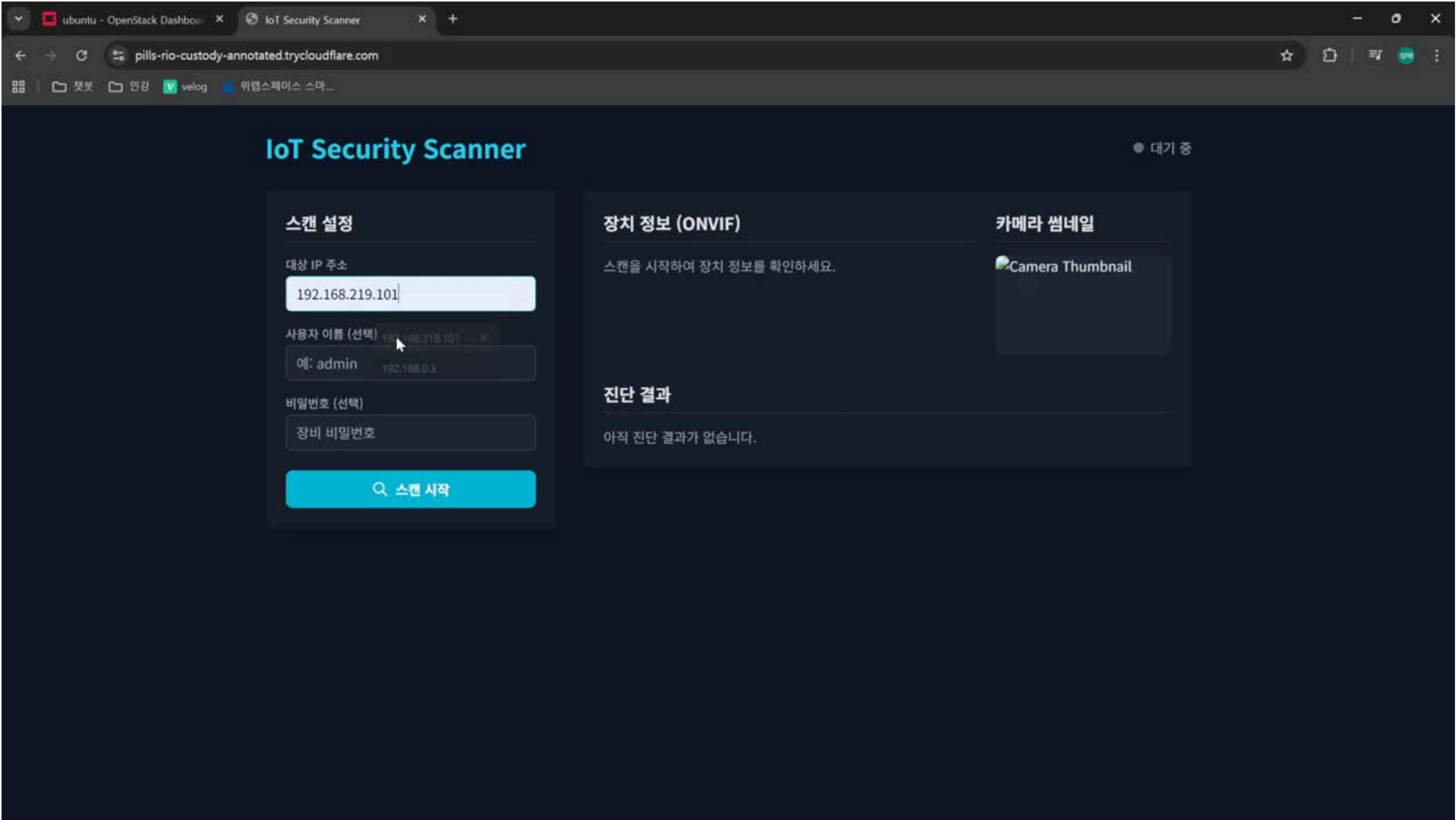
설명: 관리자 계정명이 'admin'으로 유지되고 있어 무차별 대입 공격에 취약합니다.

조치: 관리자 계정명을 변경하거나, 새 관리자 계정 생성 후 기존 admin 계정을 비활성화하세요.

취합된 검사 결과를 표 형식의 요약표와 상세 진단, 썸네일을 포함한 PDF로 자동 생성
각 항목은 영향·권고·CVSS 점수와 심각도 등급 포함

III 프로젝트 결론

1. 시연 영상



2. 기대 효과

- **홈 네트워크 환경에서 IP 카메라의 보안 상태를 주기적으로 점검 가능한 실용적인 셀프 보안 진단 도구로 활용**
- **보안 위협 인지 및 사고 예방**
 - * 사용자는 장비의 보안 취약점을 직접 확인
 - 보안 위협 인지
 - * 기본 비밀번호 변경/펌웨어 업데이트 등과 같은 사전 대응 수행
 - 영상 유출 등의 보안 사고 예방
- **연구 및 실무에 기여**
 - * OpenStack 기반의 웹 서버 환경을 활용해 진단 결과를 안정적으로 처리 후 보고서 제공
 - 실사용 중심의 구조적 완성도 확보
 - * 시스템은 모듈형 구조로 설계되어 향후 다양한 IoT 기기 보안 점검 기능 추가 및 확장 가능성을 갖추
 - 네트워크 환경에서 적용 가능한 보안 도구로 활용 가능

감사합니다