서비스 신뢰를 수치로 보여주는 일은 생각보다 단단한 공학 작업이다. 먹튀검증 체크봇은 말 그대로 먹튀 가능성이 있는 사이트나 계정을 자동으로 확인해 신호를 주는 소프트웨어다. 단순히 웹 페이지를 긁어오고, 몇 개의 키워드를 찾는 수준에서 끝나지 않는다. 자료 출처를 설계하고, 데이터를 모으는 경로를 분산하며, 신뢰 점수를 계산하고, 경고를 알맞게 전달하는 전체 파이프라인을 세워야 한다. 여기서는 API와 크롤러를 중심으로, 처음 만들 때 부딪히는 현실적인 문제와 선택지를 정리한다. 실제로 운영해 본 경험을 바탕으로, 코드와 운영의 균형을 맞추는 방법을 가능하면 구체적으로 풀어 놓았다.

무엇을 검증할 것인가를 먼저 정의하기

대상과 지표가 먼저 정리되어야 설계가 흔들리지 않는다. 먹튀검증 체크봇의 대상은 보통 다음 같은 범주로 모아진다. 도메인과 IP, 소셜 계정, 결제 수단, 공지와 사용자 후기, 사업자 등록 정보. 타깃이 명확해야 정보원도 따라 정해진다. 예를 들어, 해외 도메인 신규 등록과 네임서버 변경 이력은 WHOIS와 RDAP API로 확인할 수 있고, 환불 관련 민원 여부는 커뮤니티 게시글을 수집해 텍스트 특징으로 추출한다. 결제 게이트웨이의 상점 ID가 바뀌는지, 페이지 로딩 시점에 의심 라이브챗 위젯을 주입하는지, TLS 인증서 발급 주기가 비정상적으로 짧은지 같은 신호도 유용하다.

검증 로직은 이상 징후를 합성하는 구조가 낫다. 하나의 강한 지표로 단정하기보다, 약한 신호 여러 개를 조합해 점수를 계산하면 허위 양성률을 낮출 수 있다. 운영을 하다 보면 규칙이 늘어난다. 이때 중요 지표 5개 정도를 코어로 두고, 나머지는 보조로 관리하는 방식이 유지보수에 유리하다.

아키텍처 한눈에 보기

체크봇을 구성하는 기본 블록은 크게 수집, 처리, 저장, 알림이다. 수집은 크롤러와 외부 API 호출이 맡는다. 처리 단계에서 정규화와 특징 추출, 점수 계산이 진행된다. 저장은 원본 스냅샷과 정제된 메타데이터를 분리해 보관하는 편이 좋다. 알림은 슬랙, 텔레그램, 이메일 같은 채널 중 운영팀이 바로 반응할 수 있는 매체를 선택하면 된다.

초기에는 단일 프로세스와 간단한 스케줄러로도 충분하다. 그러나 하루 3만 페이지 이상을 긁고, API를 10여 곳 연동하면 큐와 워커가 필요해진다. 경험상, 5만 건대의 일일 작업량에선 메시지 큐와 키 밸류 캐시가 병목을 풀어 준다. RPS 20 이하의 외부 API가 섞이면 토큰 버킷 레이트리미터를 두는 것이 안전하다.

수집 경로 설계, 크롤러와 API의 균형

크롤러는 유연하지만 불안정하고, API는 안정적이지만 제한적이다. 예를 들어, WHOIS 데이터는 파일럿 단계에선 공개 WHOIS 서버를 직접 파싱해도 되지만, 운영 단계에서는 유료 RDAP API가 시간을 아껴 준다. 소셜 언급은 검색엔진의 site: 연산자를 써서 긁으면 빠르게 시작할 수 있고, 일정 규모를 넘어서면 공식 API나 공용 데이터셋으로 전환해야 한다.

페이지 렌더링 전략도 갈린다. 정적 HTML만으로 충분한 사이트가 절반 이상이지만, 결제 모듈이나 채팅 위젯 확인을 하려면 브라우저 렌더링이 필요하다. 셀레니움이나 플레이라이트 같은 헤드리스 브라우저를 선택할 때는, 메모리 사용량과 동시성, 차단 회피 전략을 함께 고려한다. 익명 프록시를 과하게 쓰면 응답이 더 느려지고, 평판이 낮은 IP는 초기 연결부터 막히는 경우가 많다. 합리적인 균형은 전체 작업 중 15에서 30퍼센트 정도만 헤드리스로 렌더링하는 방식이다.

간단한 HTTP 클라이언트로 시작하려면 다음 정도의 골격이면 된다.

import httpx from urllib.parse import urljoin TIMEOUT = httpx.Timeout(10.0, connect=5.0) HEADERS = "User-Agent": "CheckBot/1.2 (+https://example.com/bot-info)", "Accept-Language": "ko,en;q=0.8", def fetch(url: str) -> tuple[int, str, dict]: with httpx.Client(timeout=TIMEOUT, headers=HEADERS, follow_redirects=True) as client: r = client.get(url) return r.status_code, r.text, dict(r.headers) def fetch_json(api_url: str, params: dict | None = None, key: str | None = None): headers = HEADERS.copy() if key: headers["Authorization"] = f"Bearer key" with httpx.Client(timeout=TIMEOUT, headers=headers) as client: r = client.get(api_url, params=params) r.raise_for_status() return r.json()

여기서 중요한 점은 예외 처리와 재시도 정책이다. 429와 503은 백오프하고, 4xx 중 404는 캐시해도 무방하다. 10초 이상의 서버 지연은 다음 작업으로 넘기고 워커를 놀리지 않도록 한다.

법적, 윤리적 경계 지키기

크롤링은 합법과 위법 사이에 회색 지대가 있다. robots.txt를 따르는 습관 하나만으로 분쟁을 절반은 줄일 수 있다. 서비스 약관이 명시적으로 금지하면 우회하지 말아야 한다. 특히 인증 우회, 결제 단계 모의 진행, 트래픽 폭주를 유발하는 병렬 요청은 명확히 금지한다. 개인정보는 원칙적으로 수집하지 않는다. 공개 게시글이라도 전화번호와 계좌번호는 해시 처리하거나 부분 마스킹을 적용하자. 알림에 포함되는 데이터는 링크와 요약 정도로 제한하고, 원문 스냅샷은 내부 저장소에서만 확인하게 만드는 설계가 안전하다.

신뢰 신호 정의, 점수화의 기준 만들기

먹튀검증은 확정 판정이 어렵다. 그렇다면 점수 기반이 실행가능하다. 예시로, 다음 같은 특징을 설정해 본다.

- 도메인 수명과 네임서버 변경 빈도, TLS 인증서 발급 주기, 페이지 텍스트의 환불 관련 키워드 분포, 공지 업데이트 간격.

여기에 사용자 신고 수, 커뮤니티 후기의 부정 감성 비율, 결제 모듈의 자주 바뀌는 스크립트 해시 같은 값이 더해진다. 점수 모델은 선형 가중치로 시작해도 충분하다. 예를 들어, 도메인 등록 후 30일 이하이며, 공지 업데이트가 60일 넘게 없고, 외부 리뷰에서 부정 키워드가 일정 임계치를 넘으면 경고를 띄우는 식이다. 초기에는 규칙이 단순한 편이 오류 분석이 쉽다. 충분한 라벨 데이터가 모이면 로지스틱 회귀 같은 가벼운 모델로 전환할 수 있다. 복잡한 딥러닝 기반 언어모델을 바로 올리면 재현성과 비용에서 발목을 잡힌다.

다음은 간단한 가중치 기반 계산의 예다.

def score(features: dict) -> float: w = "domain_age_days": -0.015, # 젊을수록 위험 증가 "ns_change_30d": 1.2, "tls_issuance_days": -0.01, # 짧을수록 위험 "refund_kw_density": 2.5, # 환불 관련 키워드 비중 "neg_review_ratio": 3.0, "notice_gap_days": 0.02, "payment_script_hash_changed": 1.0, s = 0.0 for k, weight in w.items(): val = features.get(k, 0) s += weight * val # 0에서 100 스케일로 변환 s = max(0.0, min(100.0, 50 + s * 10)) return s

이 숫자들은 반드시 실제 데이터로 튜닝해야 한다. 초반에는 과감히 로그를 남겨 주기적으로 상관관계를 확인하자. 모델 버전과 가중치를 함께 기록해 A/B 비교가 가능해야 한다.

텍스트 처리, 허술한 키워드 매칭을 넘어서

먹튀 의심 사이트는 겉으로 번지르르한 문구를 쓰는 경우가 많다. 공지사항의 문장 구조, 고객센터 응대 패턴, 약관의 환불 조항이 실마리가 된다. 자연어 처리는 과하게 어려울 필요가 없다. 형태소 분석 대신 n그램 기반의 키워드 밀도와 구문 패턴만으로도 충분히 신호를 잡는다. 특히 환불, 보증, 이벤트, 무상, 지급 지연 등 핵심 표현의

공존 여부가 중요하다. 다만 키워드 리스트가 길어질수록 과적합 우려가 있다. 한 달에 한 번쯤은 상위 기여 키워드를 점검해 쓸모없는 항목을 정리하자.

한국어 텍스트에서 HTML 아트웍이나 보안 글꼴로 조작한 케이스도 있다. 화면에는 환불이라는 단어가 나오지만 DOM에는 문자 코드가 쪼개져 있다. 이럴 때는 렌더링된 텍스트를 캔버스에서 추출하는 방법이나, 서버 사이드 렌더링된 스냅샷을 병행해 비교하는 방식이 도움이 된다. 다만 캔버스 기반 추출은 비용이 높다. 의심 점수가 일정 수준을 넘을 때만 추가로 실행하는 게 효율적이다.

구조화된 데이터의 힘, DNS와 인증서

도메인 생태 정보는 의외로 강력하다. 멋튀검증 네임서버가 짧은 기간에 자주 바뀌면, 호스팅을 전전하거나 차단을 피하려는 움직임일 수 있다. 인증서의 SAN 항목에 낯선 도메인이 잔뜩 묶여 있으면 공유 CDN의 흔적일 수 있고, 아주 이른 만료가 잦다면 자동화가 허술하다는 뜻일 수도 있다. 이 정보는 크롤러 없이도 수집이 가능하다. Python에서 dnspython과 certifi, ssl 모듈만으로도 시작할 수 있다.

import socket, ssl def get_cert(host: str, port: int = 443) -> dict: ctx = ssl.create_default_context() with socket.create_connection((host, port), timeout=5) as sock: with ctx.wrap_socket(sock, server_hostname=host) as ssock: cert = ssock.getpeercert() return cert # subject, issuer, notBefore/After, subjectAltName 등

여기서 추출한 notBefore와 notAfter의 차이를 일 수로 환산하면 발급 주기를 바로 쓸 수 있다. SAN의 개수, 발급 기관의 패턴도 함께 저장하면 나중에 유용하다.

스케줄링, 중복, 캐시

크롤링과 API 호출에는 자연스러운 주기가 있다. DNS는 하루 한 번이면 충분하지만, 공지와 리뷰는 2에서 6시간 간격이 적당하다. 스케줄을 촘촘하게 잡으면 중복이 폭증한다. 경험상 URL 정규화만으로도 중복률을 절반 가까이 줄인다. 쿼리 파라미터에서 추적용 키를 지우고, 대소문자를 통일하며, 슬래시를 정리한다.

한 번 수집한 자원은 짧게라도 캐시하자. 404와 410은 하루 이상 캐시해 재시도를 막고, 200이라도 ETag와 Last-Modified를 활용하면 대역폭을 아낄 수 있다. API는 반대로 레이트리밋이 걸리는 즉시 백오프하고, 남은 한도 정보를 상태 저장소에 기록해 다른 워커가 참고하게 만든다.

차단 회피가 아니라 충돌 최소화

운영을 하다 보면 IP 차단을 몇 번은 겪는다. 문제는 어떻게 뚫느냐가 아니라, 상대와 충돌을 줄이느냐다. 합리적인 요청 속도를 유지하고, 명확한 User-Agent를 쓰고, 봇 안내 페이지를 운영하면 많은 사이트가 봐준다. 필요 시 연락이 닿을 수 있도록 프로필 페이지에 이메일과 목적을 공개하자. 프록시를 돌리는 것보다 기본 매너를 지키는 편이 훨씬 오래간다.

저장 전략, 로그와 스냅샷의 분리

데이터 저장은 원본과 파생 데이터를 분리하는 게 핵심이다. HTML 스냅샷, 스크린샷, 원문 JSON은 객체 저장소에 버전과 체크섬을 붙여 보관한다. 파싱된 필드와 점수는 관계형 DB에 넣는다. 이 구분이 있어야 재현이 가능하고, 규칙 변경 시 과거 데이터를 재처리할 수 있다. 텍스트 스냅샷은 압축률이 높아, zstd 기준으로 70퍼센트 이상 줄어든다. 스크린샷은 PNG보다는 WebP가 이득이다.

스키마는 처음부터 유연하게 설계하자. features라는 JSON 컬럼을 둬서 실험적인 특징을 담고, 지표가 안정되면 컬럼으로 승격하는 방식이 좋다. score는 숫자와 버전, 기준시각을 함께 저장한다. 점수의 타임라인을 그려 보면, 특정 이벤트 전후의 급변을 한눈에 잡을 수 있다.

알림, 사람이 처리하기 쉬운 형태로

알림은 많을수록 피로해진다. 점수가 임계치를 넘더라도, 같은 도메인에서 비슷한 신호가 연속으로 나오면 묶어서 하나로 보내자. 채널은 팀의 응답 습관에 맞추는 것이 정답이다. 슬랙의 경우, 스레드로 팔로업을 이어가고 원문 링크, 핵심 신호 3개, 마지막으로 수동 확인 버튼을 보낸다. 텔레그램 봇을 쓴다면 인라인 버튼으로 확인, 보류, 오탐, 정탐을 바로 태깅할 수 있게 한다.

간단한 텔레그램 알림 코드는 다음처럼 시작할 수 있다.

import httpx def tg_send(bot_token: str, chat_id: str, text: str): url = f"https://api.telegram.org/botbot_token/sendMessage" payload = "chat_id": chat_id, "text": text, "disable_web_page_preview": True r = httpx.post(url, json=payload, timeout=10.0) r.raise_for_status()

문자 그대로의 링크와 요약을 보내되, 민감한 데이터는 생략한다. 운영자는 필요할 때 내부 대시보드에서만 상세 스냅샷을 본다.

최소 기능 제품으로 시작하기

과한 설계를 경계하자. 일단 하루에 100개의 대상만 꾸준히 확인해도 충분히 쓸모가 있다. 시범 운영 2주 정도면 거짓 경고의 패턴이 보인다. 그 정보를 바탕으로 규칙을 다듬는다. 아래는 시작 시 유효했던 짧은 체크리스트다.

- 대상 목록을 정적 파일로 두고, 매일 자정과 정오에만 수집한다.
- HTML 스냅샷과 헤더만 저장하고, 본문 파싱은 나중에 배치로 돌린다.
- DNS, WHOIS, 인증서는 별도의 워커가 처리하게 분리한다.
- 점수 기준은 단일 임계치 대신, 경고와 주의 두 단계로 나눈다.
- 경고 건수는 하루 20건 이내로 제한하고, 초과분은 다음 날로 이월한다.

이 다섯 가지만 지켜도 초반 피로를 크게 줄일 수 있다. 나중에 대상이 늘고, 규칙이 정교해지면 스케줄, 워커 풀, 캐시 계층을 차근차근 확장하면 된다.

테스트와 품질, 실패에서 배우는 루프

체크봇은 외부 세계와 연결돼 있어 테스트가 까다롭다. 모의 서버와 고정 응답을 준비해 단위 테스트를 돌리고, 실제 대상에 대해서는 하루 한 번의 건강검진 배치를 둔다. 최근 일주일의 성공률, 평균 지연, 4xx와 5xx 비율을 기록해 추이를 본다. 헤드리스 브라우저는 운영체제와 폰트에 민감하니, 도커 이미지와 드라이버 버전을 고정한다.

오탐과 미탐은 금으로 된 데이터다. 운영자가 알림에 태그를 달면, 다음 날 새벽에 그 결과를 학습 데이터로 반영하는 루프를 짠다. 최소한 한 달에 한 번은 상위 기여 특징과 가중치를 재점검하고, 쓸모없는 규칙을 퇴출한다. 실패를 재현할 수 있도록 원본 스냅샷과 파싱 로그를 보관하는 습관이 필요하다.

비용과 성능, 현실적인 숫자

대략적인 감으로, 텍스트 크롤링 1만 페이지당 네트워크는 1에서 3GB, 저장소는 압축 후 수백 MB 수준이다. 헤드리스 렌더링은 건당 150에서 400ms의 CPU 시간을 쓴다. 인증서 조회와 DNS는 매우 가볍다. 외부 유료 API는 월 단위로 과금되니, 초반에는 무료 할당량을 넘기지 않도록 요청을 모아 배치 처리하자. 예를 들어, 동일 도메인에 대해 WHOIS를 하루에 두 번 이상 조회할 이유가 거의 없다. 반대로 리뷰 크롤링은 신규 게시글이 빠르게 늘 수 있어, 페이지네이션을 깊게 타지 않도록 커서 기반 수집을 적용하는 편이 비용 대비 효율이 좋다.

간단한 파이프라인 예시

작은 파일럿을 상정해, 스케줄러, 워커, 저장소를 한 프로세스 안에서 구현한 예시 흐름을 정리해 본다.

from datetime import datetime, timedelta from queue import Queue import threading, time, sqlite3 targets = ["https://example-a.com", "https://example-b.net",] q = Queue(maxsize=1000) results = [] def producer(): while True: for url in targets: q.put(("html", url)) q.put(("dns", url)) q.put(("cert", url)) time.sleep(6 * 3600) # 6시간 주기 def worker(): while True: job, url = q.get() try: if job == "html": code, html, headers = fetch(url) features = extract_features_html(html, headers) elif job == "dns": features = extract_features_dns(url) else: host = url.split("//", 1)[1].split("/", 1)[0] cert = get_cert(host) features = extract_features_cert(cert) results.append((url, features, datetime.utcnow())) except Exception as e: # 로그 남기기 pass finally: q.task_done() def extract_features_html(html: str, headers: dict) -> dict: # 간단한 예시 density = sum(html.count(k) for k in ["환불", "보증", "지급 지연"]) / max(len(html), 1) return "refund_kw_density": density, "content_length": len(html) def extract_features_dns(url: str) -> dict: # 생략: dnspython 등으로 NS, A, TTL 조회 return "ns_change_30d": 0 def extract_features_cert(cert: dict) -> dict: # notBefore/After 파싱, SAN 개수 return "tls_issuance_days": 90 def aggregator_and_store(): conn = sqlite3.connect("checkbot.db") conn.execute(""" CREATE TABLE IF NOT EXISTS checks (url TEXT, ts TEXT, score REAL, features TEXT)""") while True: if not results: time.sleep(1) continue url, feats, ts = results.pop(0) s = score(feats) conn.execute("INSERT INTO checks VALUES (?,?,?,?)", (url, ts.isoformat(), s, str(feats))) conn.commit() if s >= 75: tg_send("", "", f"[경고] url 점수 s\n주요 특징: list(feats.items())[:3]") # 스레드 가동 threading.Thread(target=producer, daemon=True).start() for _ in range(4): threading.Thread(target=worker, daemon=True).start() threading.Thread(target=aggregator_and_store, daemon=True).start() while True: time.sleep(60)

이 코드는 교육용으로 지나치게 단순화되어 있다. 하지만 흐름은 그대로다. 수집, 특징, 점수, 저장, 알림. 파일럿을 통해 병목과 허점을 파악하는 용도로는 충분하다.

사용자 인터페이스, 운영자의 시간을 아낀다

체크봇이 유용해지려면 운영자의 선별 시간이 줄어야 한다. 내부 대시보드에는 다음만 넣어도 효과가 크다. 최근 경고 목록, 도메인별 점수 추이 차트, 주요 특징 상위 5개, 원본 스냅샷 링크. 두세 화면 안에서 판단과 라벨링이 끝나도록 레이아웃을 좁게 잡는다. 컬러는 최소화하고, 신호 강도에 따라 아이콘만 바뀌게 하면 시각 피로가 줄어든다. 라벨이 쌓일수록 모델 개선 속도가 붙는다.

실전에서 자주 만나는 함정

연속 리다이렉트와 지리 기반 차단이 섞여 있으면, 봇은 200 대신 301, 302만 보게 된다. 실제 이용자는 브라우저 스택에서 자바스크립트를 통해 최종 페이지로 안내받는다. 이럴 때는 Accept-Language와 GeoIP를 조정한 두세 개의 대표 환경을 만들어 테스트한다. 또 하나, 이미지로만 된 공지 페이지는 OCR 없이는 분석이 어렵다. OCR은 비용이 많이 든다. 의심 점수가 높고 텍스트가 없을 때만 제한적으로 돌리자.

리뷰 수집에서는 중복 계정이 만든 가짜 후기가 혼란을 준다. 계정 생성일, 글 간 간격, 동일 구문 반복률 같은 메타 특징을 쓰면 어느 정도 걸러진다. 실제로 가짜 후기의 60에서 80퍼센트는 문장 패턴이 좁다. 다만 너무 공격적으로 걸러내면 정상 후기까지 지워진다. 기준값을 한꺼번에 올리지 말고, 매주 5퍼센트포인트씩만 조정하자.

보안과 투명성

체크봇 자체가 악용 대상이 될 수 있다. 봇의 대시보드와 알림 채널은 접근 통제를 명확히 하고, 토큰과 키는 독립된 비밀 저장소에서 관리한다. 감사 로그를 남겨 누가 어떤 항목을 봤는지, 어떤 판정을 내렸는지 기록한다. 외

부에 공개하는 리포트에는 근거를 단정적으로 적지 말고, 신호와 점수, 확인 필요 여부로 표현을 조심하자. 먹튀검증이라는 이름 때문에 오탐이 큰 피해를 줄 수 있다. 투명하게 수정하고, 정정보도 수준의 공지를 준비하는 태도가 필요하다.

확장과 장기 운영

처음에는 단일 서버, 하루 수천 건이면 되지만, 성공하면 요청량이 기하급수로 늘어난다. 워커를 컨테이너로 분리하고, 메시지 큐를 중앙에 둔다. 크롤링과 API 호출을 도메인 단위로 샤딩하면 핫스팟을 피할 수 있다. 대상이 수십만으로 커지면, 크롤러의 주기 대신 변경 감지 이벤트에 반응하는 구조가 유리하다. 예를 들어, 인증서 투명성 로그, 도메인 신규 등록 피드, 커뮤니티의 RSS를 훅으로 받아온다. 불필요한 폴링을 줄이면 비용이 급감한다.

신뢰를 만드는 운영 습관

결국 먹튀검증 체크봇의 목표는 고품질의 경고다. 품질을 좌우하는 요소는 코드보다 운영 습관일 때가 많다. 규칙 변경과 모델 업데이트를 기록하고, 근거 없는 지표는 제거한다. 내부적으로는 샘플에 대한 수동 검증을 지속하고, 외부 신고창구를 통해 유의미한 사례를 수집한다. 데이터 보존 기간과 폐기 정책을 문서화해, 필요 이상의 정보를 오래 들고 있지 않도록 한다. 팀이 커지면 온콜 체계를 만들고, 야간 경고는 임계치를 높인다. 사람의 수면을 보호하는 알림 정책이 장기 성과를 좌우한다.

마지막으로, 현실적인 적색 신호들

초보자도 금방 체감할 수 있는 적색 신호가 있다. 아래 항목들은 데이터 없이도 1차 필터로 쓸 만하다.

- 도메인이 최근 30일 이내에 등록됐고, 공지 페이지의 마지막 업데이트가 오래됐다.
- 환불이나 지연 지급 관련 문구가 자주 보이지만 실제 약관의 환불 섹션이 비어 있거나 이미지로만 제공된다.
- 결제 모듈 스크립트의 해시가 며칠 간격으로 바뀌고, 상점 ID가 일치하지 않는다.
- 고객센터 채널이 텔레그램, 카카오 채널 하나뿐이며, 사업자 정보가 푸터에 없다.
- 외부 커뮤니티에서 같은 문장 패턴의 후기 글이 짧은 시간에 다수 올라온다.

이 신호만으로 단정할 수는 없지만, 점수 계산의 강한 입력이 된다. 규칙은 시간이 흐르면서 바뀐다. 정답은 축적된 데이터와 책임감 있는 운영에서 나온다. 체크봇은 그 과정을 빠르고 일관되게 돕는 도구다. API와 크롤러라는 기본기를 단단히 쌓아 두면, 분석의 깊이와 범위를 꾸준히 넓힐 수 있다.