

학사학위논문

웹 취약점 진단에 대한
방법론과 자동화 스크립트

Methodology and automation scripts
for web vulnerability diagnosis

지도교수 이병천

이 논문을 정보보호학 학사 학위 논문으로 제출함

2025. 06. 14

중 부 대 학 교

정보보호학과

김현빈

<제목 차례>

제1장 서론	5
제1절 연구의 배경 및 필요성	5
제2절 연구의 목표	5
제2장 관련 연구	6
제1절 웹 취약점 유형(OWASP Top 10)	6
1. A01:2021 - Broken Access Control	6
2. A02:2021 - Cryptographic Failures	7
3. A03:2021 - Injection	8
4. A04:2021 - Insecure Design	9
5. A05:2021 - Security Misconfiguration	10
6. A06:2021 - Vulnerable and Outdated Components	11
7. A07:2021 - Identification and Authentication Failures	12
8. A08:2021 - Software and Data Integrity Failures	13
9. A09:2021 - Security Logging and Monitoring Failures	14
10. A10:2021 - Server-Side Request Forgery (SSRF)	15
제2절 웹 공격 트렌드 분석	15
1. 인공지능(AI) 기반 공격	15
2. 랜섬웨어 공격의 진화	16
3. 제로데이 취약점	16
4. API 공격	16
5. 클라우드 보안	17
제3절 웹 취약점 진단 오픈 소스 스캐너	17
1. OWASP ZAP(Zed Attack Proxy)	17
2. Arachni	18
3. Wapiti	18
제4절 KISA 가이드라인	19
제3장 웹 취약점 진단 및 진단 스크립트	20
제1절 오픈 소스 스캐너를 이용한 웹 진단	20
1. OWASP ZAP	20
2. Arachni	21
3. Wapiti	22
4. 결론	23
제2절 KISA 가이드라인에 따른 웹 진단 스크립트 제작	24

제3절 제작한 스크립트를 이용한 웹 진단	31
제4장 결 론	32
제1절 결과 분석	32
제2절 향후 연구 계획	32

<표 차례>

OWASP ZAP 웹 진단 결과	20
Arachni 웹 진단 결과	21
Wapiti 웹 진단 결과	22
제작한 스크립트 웹 진단 결과	31

제1장 서론

제1절 연구의 배경 및 필요성

클라우드 컴퓨팅, 인공지능(AI), 사물인터넷(IoT) 등 다양한 정보통신 기술이 발전하면서 우리 삶에서 웹 서비스의 의존도가 급격히 증가하고 있다. 이에 따라 공공기관, 기업, 개인 등 다양한 계층의 핵심 인프라로 위치했으며, 동시에 공격자들에게 매력적인 공격 표적이 되고 있다. 한국인터넷진흥원(KISA)이 발표한 2024년 2분기 침해사고 유형 Top 10 자료에 의하면, 전체 침해사고 중 65% 이상이 웹과 관련된 사고로 나타났다. 이는 웹이 여전히 주요 공격 벡터로 악용되고 있음을 의미하며, 단순한 시스템 보호를 넘어 웹 애플리케이션 자체의 취약점을 진단하고 대응하는 것이 보안의 핵심 과제라고 생각한다. 또한, 한 연구에 따르면 평균적인 웹 애플리케이션에서는 약 15개의 취약점이 발견되며, 이 중 2개 이상은 치명적인 보안 위협으로 분류된다. 이러한 수치는 웹 서비스가 복잡해질수록 보안 위협이 비례하여 증가함을 보여주며, 기존의 수동적이고 일회성 중심의 보안 점검 방식만으로는 현재의 웹 보안 환경에 효과적으로 대응하기 어렵다는 한계를 드러낸다. 특히 KISA의 침해사고 통계에 따르면, 웹 기반 공격은 연평균 증가율을 훨씬 넘어서는 약 60%의 급증세를 보이고 있어, 웹 보안 취약점에 대한 체계적이고 자동화된 진단 방법론이 필요하다.

제2절 연구의 목표

본 연구는 기술의 발전으로 계속해서 발전해가는 웹 기반 사이버 위협의 심각성을 인식하고, 이러한 문제에 대한 효과적인 대응 방안을 찾는 것에 목표한다.

첫째, OWASP Top 10 2021에서 정의된 주요 웹 취약점들의 종류와 발생 원인과 잠재적 위험 요소에 대한 근본적인 원인에 대해 분석하고, 이를 통해 웹 취약점의 근본적인 발생 원인을 이해하고자 한다.

둘째, 인공지능(AI) 기반 공격, 랜섬웨어의 진화, 제로데이 공격, API 및 클라우드 보안 위협 등 최신 웹 공격 트렌드에 대해 분석하여 진화하는 보안 위협에 대해 빠르게 대비할 수 있도록 한다.

마지막으로, OWASP ZAP, Arachni, Wapiti와 같은 오픈 소스 웹 취약점 스캐너들의 기능과 장단점에 대해 분석하고 테스트 환경에서 실제로 사용하여 진단 결과를 통해 오픈 소스 스캐너의 효과를 검증하며, KISA 취약점 분석 및 평가 방법 상세 가이드의 점검 방법을 기반으로 제작된 파이썬 스크립트를 테스트 사이트에 점검하여, 오픈 소스

스캐너들과의 진단 결과에 대해 비교 및 분석한다. 이를 통해 제작한 스크립트의 개선 사항 및 웹 서비스 구축 및 운영 시 실질적인 활용 가능한 진단에 대한 방법론을 제시하고자 한다. 본 연구는 이러한 과정을 끊임없이 연구하여 웹 환경에서 지속 가능한 보안을 추구하는 중요한 배경이 될 수 있도록 하는 목표를 하고 있다.

제2장 관련 연구

제1절 OWASP Top 10 웹 취약점

1. A01:2021 - Broken Access Control

접근 제어 실패는 인증된 사용자의 작업 범위를 제한하는 보안 메커니즘이다. 접근 제어 실패란, 제한이 제대로 적용되지 않아 공격자가 권한 외 기능이나 데이터에 접근할 수 있는 상태를 말한다. 이는 OWASP Top 10 2021에서 가장 심각한 위험 요소로 선정될 정도로 발생 빈도가 높고 피해 규모가 크다. 이러한 정책은 애플리케이션 전반에 일관되게 적용되어야 하며, 서버 측에서 강제되어야 한다.

특히, 클라우드 환경에서는 IAM(Identity and Access Management) 설정 오류로 인한 접근 제어 실패 사례가 증가하고 있다. 예를 들어, AWS S3 버킷에 대한 접근 권한 설정이 부족해 민감한 데이터가 외부에 노출되는 사례가 발생하고 있다.

▶ 주요 발생 원인 및 형태

- 1) **불충분한 인가** : 사용자 역할 또는 권한 수준에 따른 접근 제한이 미흡한 경우를 의미한다. 이로 인해 일반 사용자가 관리자 기능에 접근하거나, 다른 사용자의 데이터를 조회, 수정, 삭제할 수 있다. 예를 들면, URL이나 API 엔드포인트 경로를 추측하여 접근하거나, 요청 파라미터의 사용자 ID 값을 변경하여 타인의 정보에 접근하는 방식이 있다.
- 2) **경로 조작** : 사용자 입력값(주로 파일명이나 경로)에 대한 검증 없이 파일 시스템 경로를 구성할 때, 상위 디렉터리 이동 문자를 사용하여 웹 서버 루트 디렉터리 외부의 민감한 파일(예: 설정 파일, 소스 코드, 시스템 파일 /etc/passwd 등)에 접근하거나 다운로드하는 취약점이다.
- 3) **관리자 페이지 노출** : 적절한 접근 제어 없이 관리자 기능 페이지가 외부에 노출되어, 추측이나 검색 엔진을 통해 발견될 경우 비인가 사용자가 접근하여 시스템을 제어할 수 있는 위험이다.
- 4) **부적절한 인가** : 데이터베이스 키 값이나 파일명과 같은 내부 구현 객체 참조가 URL이나 파라미터에 직접 노출되고, 이에 대한 접근 권한 검증이 누락된 경우, 공격자가 참조 값을 조작하여 다른 사용자의 정보에 접근할 수 있다.

5) CORS(Cross-Origin Resource Sharing) 설정 오류 : 웹 브라우저의 동일 출처 정책을 완화하기 위한 CORS 설정 시, 접근을 허용하는 출처를 지나치게 넓게 설정(예: * 사용)하면 악의적인 다른 도메인에서 해당 웹 애플리케이션의 리소스에 접근할 수 있게 된다.

6) 파일 업로드 권한 통제 실패 : 파일 업로드 기능 자체에 대한 접근 권한 검증이 미흡하거나, 업로드된 파일이 저장되는 디렉터리에 대한 접근 제어가 부적절하여 비인가 사용자가 파일을 업로드하거나, 업로드된 악성 파일에 접근하여 실행시킬 수 있는 경우이다.

2. A02:2021 - Cryptographic Failures

암호화 실패는 개인 식별 정보(PII), 금융 정보, 인증 정보, 건강 정보 등 민감한 데이터를 보호하는 과정에서 암호화 기술이 부적절하게 적용되었거나 누락된 경우를 의미한다. 이 항목은 OWASP 2017의 “민감 데이터 노출”에서 명칭이 변경되었으며, 단순 결과보다 실패의 원인과 방식에 초점을 맞추고 있다.

암호화 실패는 주로 데이터 전송 또는 저장 시점에서 발생하며, 취약하거나 오래된 알고리즘의 사용, 키 관리 미흡, 암호화 누락 등이 주요 원인으로 작용한다.

▶ 주요 발생 원인 및 형태

1) 민감 데이터 평문 전송 : 로그인 자격 증명, 개인 정보, 신용카드 정보 등 민감 정보를 HTTP와 같은 비보안 프로토콜로 암호화 없이 전송하는 경우이다. 이 경우 공격자는 네트워크 스니핑을 통해 데이터를 탈취할 수 있으므로, 반드시 TLS 기반의 HTTPS를 사용해야 한다.

2) 민감 데이터 평문 저장 : 데이터베이스나 파일 시스템에 비밀번호, 주민등록번호 등 민감 정보를 암호화하지 않고 저장하는 경우이다. 이 경우, 데이터베이스가 유출될 경우 심각한 개인정보 침해로 이어질 수 있다. 비밀번호는 반드시 Salt를 추가한 강력한 해시 함수(예: bcrypt, scrypt, Argon2)를 사용해야 하며, 다른 민감 정보는 검증된 암호화 알고리즘(예: AES)을 사용하여 암호화해야 한다.

3) 취약하거나 오래된 암호화 알고리즘 사용 : MD5, SHA-1과 같은 해시 함수나 DES, RC4 같은 대칭키 암호화 알고리즘은 현재 보안 강도가 충분하지 않아 사용해서는 안 된다. 검증된 최신 표준 알고리즘을 사용해야 한다.

4) 부적절한 키 관리 : 암호화 키를 소스 코드에 하드 코딩하거나, 안전하지 않은 방식으로 저장/전송하거나, 주기적으로 교체하지 않는 등 키 관리 부실은 암호화 체계를 무력화시킨다.

5) 오류 메시지를 통한 정보 노출 : 암호화/복호화 과정의 실패 시, 지나치게 상세한 오류 메시지를 사용자에게 노출하면 시스템 내부 정보나 암호화 방식에 대한 힌트를 제공할 수 있다.

6) 부적절한 난수 생성 : 세션 ID, 비밀번호 재설정 토큰, 암호화 키 등을 생성할 때 예측 가능한 난수를 사용하면 공격자가 값을 추측할 수 있다. 암호학적으로 안전한 난수 생성기(CSPRNG)를 사용해야 한다.

3. A03:2021 - Injection

인젝션 취약점은 외부에서 입력된 검증되지 않은 입력 데이터가 애플리케이션의 명령어 또는 쿼리의 일부로 삽입되어, 공격자가 비정상적인 명령 실행이나 데이터 접근을 유도할 수 있는 보안 취약점을 의미한다. 이 취약점은 웹 애플리케이션뿐만 아니라 다양한 환경에서 발생할 수 있으며, 중대한 보안 위협으로 간주된다. 주로 입력값에 대한 적절한 유효성 검사 부족 또는 이스케이프 처리가 누락된 경우 발생한다.

▶ 주요 발생 원인 및 형태

1) **SQL Injection** : 사용자 입력값이 적절한 필터링 또는 이스케이프 처리 없이 SQL 쿼리에 포함되어, 데이터베이스를 조작할 수 있는 대표적인 공격 기법이다. 이를 통해 공격자는 민감 정보 유출, 데이터 변조 및 삭제, 인증 우회, 시스템 명령어 실행 등을 수행할 수 있다.

2) **Cross-Site Scripting(XSS)** : 공격자가 웹 페이지에 악의적인 스크립트(주로 JavaScript)를 삽입하여, 해당 페이지를 방문하는 다른 사용자의 브라우저에서 자동 실행되도록 유도하는 공격이다. 이를 통해 세션 탈취, 악성 사이트 리디렉션, 키 로깅, 피싱 등의 행위가 가능하다. XSS는 Stored, Reflected, DOM 기반 XSS 등 다양한 유형이 있으며, 입출력 값에 대한 이스케이프 처리, CSP(Content Security Policy) 적용 등을 통해 방어할 수 있다.

3) **OS Command Injection** : 사용자 입력값이 운영체제 명령어를 실행하는 함수의 인자로 사용될 때, 이를 조작하여 임의의 시스템 명령어를 실행시키는 공격이다. 심각한 경우 서버 제어권 탈취로 이어질 수 있다.

4) **LDAP Injection** : 입력값이 필터링 없이 LDAP 쿼리에 포함될 경우, 공격자가 디렉터리 서비스 구조를 조작하거나 권한을 탈취할 수 있다.

5) **XPath Injection**: 입력값이 필터링 없이 XPath 쿼리에 포함되어, XML 기반 데이터 구조를 조작하여 민감 정보에 접근할 수 있다. SQL 인젝션과 유사하나, 대상이 XML이라는 부분이 다르다.

6) **SSI Injection** : 서버 측 Include 지시문에 사용자 입력이 포함될 때, 공격자가 악의적인 SSI 지시어를 삽입하여 서버에서 임의의 명령을 실행 또는 파일 내용을 노출을 유도할 수 있다.

7) **기타** : NoSQL 인젝션, SMTP 인젝션, ORM 인젝션 등 다양한 데이터 처리 경로에서 발생 가능하다. 입력값이 실행 가능한 문맥에 포함되는 모든 경우 잠재적인 인젝션 위험이 존재한다.

4. A04:2021 - Insecure Design

안전하지 않은 설계는 소프트웨어 개발 생명주기 초기에 보안 요구사항이 충분히 고려되지 않아서 발생하는 근본적인 보안 취약점을 의미한다. 이는 단순한 구현 오류와 달리 설계 단계에서의 구조적 결함으로 인해 시스템 아키텍처 전반에 걸쳐 심각한 영향을 미칠 수 있다. 위협 모델링 부재, 불완전한 비즈니스 로직, 안전한 설계 패턴 미적용 등 다양한 원인으로 발생하며, 이후 개발 단계에서 코드가 아무리 견고하게 작성되더라도 보안 결함이 제거되지 않는 위험성을 내포하고 있다.

▶ 주요 발생 원인 및 형태

1) **위협 모델링 부재 또는 부족** : 개발 초기 단계에 잠재적인 보안 위협을 식별, 분석, 평가하는 위협 모델링이 수행되지 않거나 미흡할 경우, 공격 경로를 사전에 방지하지 못하고 보안 취약점이 설계에 반영된 채로 구현될 가능성이 높아진다.

2) **안전하지 않은 비즈니스 로직** : 비즈니스 로직은 애플리케이션의 주요 기능 수행 방식을 정의하며, 이 로직이 보안 측면에서 취약할 경우 공격자가 시스템의 흐름을 악용할 수 있다. 예를 들어, 상품 가격을 클라이언트 측에서 조작하여 전송해도 서버 측 검증이 없거나, 결제 과정에서 중간 단계를 생략해도 처리가 완료되는 경우 등이 이에 해당한다.

3) **리소스 제한 부재** : 파일 업로드 크기, 요청 횟수, 계정 생성 수 등에 대한 제한이 없을 때, 공격자는 서비스 거부(DoS) 공격이나 자동화된 대량 요청을 통해 시스템에 과부하를 유발할 수 있다.

4) **민감 정보 처리 로직 오류** : 중요 정보를 로그 파일에 평문으로 기록하거나, 불필요하게 사용자에게 노출하는 등 민감 정보 처리에 대한 설계 실수가 발생할 수 있다.

5) **취약한 암호 복구/변경 기능 설계** : 비밀번호 재설정 과정에서 예측 가능한 질문을 사용하거나, 임시 비밀번호를 안전하지 않은 방법(SMS, 이메일 등)으로 전송, 다단계 인증 미적용 등은 대표적인 취약점이며, 안전한 인증 절차와 다단계 인증(MFA) 적용이 필요하다.

6) **과도한 기능 제공** : 사용자에게 필요 이상의 권한이나 기능을 기본적으로 제공하는 설계는 잠재적인 보안 위협을 증가시킨다. 최소 권한 원칙에 따라, 각 사용자에게 필요한 최소한의 권한만 부여하고, 불필요한 기능은 제거해야 한다.

5. A05:2021 - Security Misconfiguration

보안 설정 오류는 운영체제, 웹 서버, 애플리케이션 서버(WAS), 데이터베이스 관리 시스템(DBMS), 프레임워크 등 애플리케이션을 구성하는 다양한 요소들의 보안 설정이 적절하게 구성되지 않았을 때 발생하는 취약점이다. 이는 시스템의 보안 수준을 약화시키고, 공격자가 시스템 침해, 민감 정보 접근 또는 유출을 시도할 수 있는 공격 경로를 제공하게 된다. 일반적으로 안전한 기본 설정 미적용, 불필요한 기능의 활성화, 클라우드 설정 오류, 상세한 오류 메시지 노출 등의 문제로 발생하며, 대부분 자동화된 보안 스캐너를 통해 탐지가 가능하다.

▶ 주요 발생 원인 및 형태

- 1) **기본 계정 및 비밀번호 사용** : 운영체제, 데이터베이스, 관리자 콘솔 등에 제공되는 기본 계정(예: admin, root)과 추측하기 쉬운 기본 비밀번호를 변경하지 않고 사용하는 경우, 공격자는 이를 통해 인증을 우회하거나 관리자 권한을 획득할 수 있다.
- 2) **불필요한 기능 활성화** : 디버깅 모드, 샘플 애플리케이션, 불필요한 HTTP 메소드(예: PUT, DELETE, TRACE)가 운영 환경에서 활성화된 경우, 공격자가 해당 기능을 악용해 시스템을 침해할 수 있는 가능성이 증가한다.
- 3) **디렉터리 리스팅 활성화** : 웹 서버에서 특정 경로에 인덱스 파일이 없을 때, 해당 디렉터리 내의 파일 목록이 노출되는 설정 오류이다. 이를 통해 공격자는 애플리케이션 구조 및 민감 파일의 존재 여부를 파악할 수 있다.
- 4) **과도한 오류 메시지 노출** : 애플리케이션에서 예외 발생 시 스택 트레이스, 데이터베이스 정보, 서버 경로 등의 내부 정보가 포함된 상세 오류 메시지를 사용자에게 그대로 노출하는 경우, 이는 정보 유출 및 시스템 구조 노출로 이어질 수 있다.
- 5) **부적절한 HTTP 보안 헤더 설정** : X-Frame-Options, CSP(Content Security Policy), HSTS(HTTP Strict Transport Security), X-Content-Type-Options 등 보안 헤더가 누락되었거나 잘못 설정된 경우, 클랙재킹, 콘텐츠 스니핑 등 공격에 취약해진다.
- 6) **클라우드 서비스 설정 오류** : 예를 들면, AWS S3 버킷의 접근 권한을 Public으로 잘못 설정하거나, 방화벽에서 불필요한 포트를 외부로 개방하는 경우, 민감 정보를 광범위하게 유출될 수 있다.

7) 파일 업로드 처리 설정 오류 : 파일 업로드 기능을 통해 저장된 디렉터리에 실행 권한이 설정되어 있거나, 서버가 특정 확장자(예: .php, .jsp)를 스크립트로 해석하도록 잘못 설정된 경우, Web Shell 업로드하고 이를 실행하여 시스템을 완전히 장악할 수 있다.

6. A06:2021 - Vulnerable and Outdated Components

취약하거나 오래된 구성 요소는 현대 웹 애플리케이션에서 필수적으로 사용되는 오픈 소스 라이브러리, 프레임워크, 외부 모듈 등에 존재하는 알려진 보안 취약점 또는 보안 패치가 더 이상 제공되지 않는 소프트웨어로 인해 발생하는 보안 위협이다.

애플리케이션의 생산성과 개발 속도를 높이기 위해 외부 구성 요소 활용은 필수적이지만 취약한 버전을 사용하는 경우 공격자는 공개된 취약점 정보(CVE 등)를 기반으로 쉽게 공격할 수 있다. 특히 공격 자동화 도구를 사용하는 경우, 해당 구성 요소가 존재하더라도 심각한 침해 사고로 이어질 수 있다.

▶ 주요 발생 원인 및 형태

1) 알려진 취약점이 존재하는 라이브러리/프레임워크 사용 : Log4j, Struts, jQuery, Spring, .NET 등과 같은 서드파티 구성요소에 이미 공개된 보안 취약점이 존재하지만, 개발자가 이를 인지하지 못했거나, 인지했음에도 업데이트하지 않는 경우, 공격자는 해당 버전의 CVE 정보를 기반으로 공격을 시도할 수 있다.

2) 패치 되지 않은 운영체제 및 웹 서버 소프트웨어 : 운영체제(Linux, Windows Server), 웹 서버(Apache, Nginx, IIS), WAS(Tomcat, JBoss) 등의 소프트웨어에 대한 최신 보안 패치를 적용하지 않는 경우, 이미 알려진 취약점을 통해 공격자가 시스템 권한을 획득하거나 서비스 거부(DoS) 공격을 수행할 수 있다.

3) 더 이상 지원되지 않는(End-of-Life, EOL) 소프트웨어 사용 : 개발사에서 공식적인 기술 지원 및 보안 업데이트가 중단된 구성 요소를 사용하는 경우, 새로운 취약점이 발견되더라도 해결 방법이 제공되지 않으므로 보안상 매우 위험하다. 대표적으로 오래된 PHP 버전이나 EOL된 프레임워크, 플러그인 등이 이에 해당된다.

4) 취약한 의존성(Dependency) 관리 : 직접 사용하는 라이브러리뿐만 아니라, 해당 라이브러리가 내부적으로 의존하는 하위 구성 요소에도 취약점이 포함될 수 있다. 하지만 이를 명확히 인식하고 관리하지 않으면, 간접적인 취약점 경로를 통해 공격자가 시스템에 접근할 수 있다. 따라서 SBOM(Software Bill of Materials) 및 SCA(Software Composition Analysis)를 활용한 정기적인 구성 요소 분석이 필요하다.

7. A07:2021 - Identification and Authentication Failures

식별 및 인증 실패는 사용자의 신원 확인 및 인증 처리 과정에서 발생하는 다양한 취약점을 의미한다. 이는 비밀번호 관리, 세션 제어, 다중 인증, 계정 복구 절차, 쿠키 보안, 인증 우회 등 전체 인증 체계에 대한 설계 및 구현 미흡에서 비롯되며, 시스템의 핵심 접근 제어 기능을 무력화할 수 있는 심각한 위협으로 작용한다.

▶ 주요 발생 원인 및 형태

- 1) **취약한 비밀번호 허용 및 관리** : 추측하기 쉬운 비밀번호(예: 123456, password)를 허용하거나, 비밀번호 복잡도 정책이 없거나, 비밀번호를 평문 또는 단순 해시로 저장하는 경우이다. 이러한 경우엔 비밀번호 복잡도 정책 적용, Bcrypt, Argon2 등 강력한 해시 알고리즘 사용하는 것을 권장한다.
- 2) **무차별 대입 공격 및 크리덴셜 스테핑 허용** : 로그인 시도 횟수 제한, 계정 잠금 기능, CAPTCHA, IP 차단 등의 자동화 공격 방어 메커니즘이 없을 때, 공격자가 무작위 비밀번호 대입, 크리덴셜 스테핑으로 인증 우회 가능하기에, 비정상 로그인 시도 제한, MFA 적용, 자동화 탐지 로직 도입하는 것을 권장한다.
- 3) **취약한 세션 관리** : 세션 식별자의 보안성이 부족하거나 세션 종료 처리 미흡 등으로 인증 상태를 탈취당할 수 있다.
- 4) **세션 ID 예측 가능** : 시간 기반, 순차적 값으로 생성된 세션 ID는 공격자가 세션 ID를 추측하여 세션을 탈취할 수 있기에, CSPRNG를 사용하는 것을 권장한다.
- 5) **불충분한 세션 만료** : 사용자가 로그아웃하거나 일정 시간 동안 활동이 없을 때 세션이 서버와 클라이언트 모두 적정 시간 내에 만료되지 않는 경우이다. 공격자가 탈취한 세션 ID를 재사용할 위험이 있다.
- 6) **세션 ID의 URL 노출** : 세션 ID가 URL 파라미터에 포함되어 전송될 경우, 로그 파일, 브라우저 히스토리, Referer 헤더 등을 통해 노출될 위험이 있다. 세션 ID는 반드시 쿠키를 통해 전달하고, HttpOnly, Secure 플래그 설정을 해야 한다.
- 7) **세션 고정** : 공격자가 사용자가 로그인하기 전에 특정 세션 ID 값을 미리 고정시킨 후, 사용자가 해당 세션 ID로 로그인하면 그 세션을 탈취하는 공격이다. 사용자가 인증되면 기존 세션을 무효화하고 새로운 세션 ID를 발급해야 한다.
- 8) **인증 정보 복구/변경 절차 미흡** : 비밀번호 재설정 기능에서 사용자 본인 검증 절차가 충분하지 않거나, 임시 비밀번호나 재설정 링크를 안전하지 않은 방식으로 전달하는 경우 계정 탈취로 이어질 수 있다.

9) 다중 인증 부재 또는 우회 가능 : 중요 시스템 접근이나 민감 작업 수행 시 MFA를 적용하지 않거나, MFA 구현 방식에 보안이 미흡한 경우 인증을 우회할 수 있다.

10) 쿠키 변조를 통한 인증 우회 : 로그인 상태나 사용자 식별 정보가 담긴 쿠키 값을 사용자가 임의로 변조할 수 있고, 서버 측에서 이를 검증하지 않을 경우 인증을 우회할 수 있다. 쿠키에는 민감 정보를 직접 담지 않고, 서명(HMAC)이나 암호화를 적용하고 HttpOnly, Secure, SameSite 플래그를 사용하는 것을 권장한다.

8. A08:2021 - Software and Data Integrity Failures

소프트웨어 및 데이터 무결성 실패는 코드, 중요 데이터, 소프트웨어 업데이트 과정 등에서 무결성 검증이 제대로 이루어지지 않아 발생하는 취약점이다. 특히 자동화된 배포 환경(CI/CD), 파일 업로드 처리, 객체 직렬화/역직렬화, 설정 파일 처리 등에서 광범위하게 발생하며, 공격자가 코드 삽입 또는 데이터 조작을 통해 시스템 권한을 탈취하거나 서비스 동작을 변조할 수 있는 취약점이다.

▶ 주요 발생 원인 및 형태

1) 안전하지 않은 업데이트 과정 : 소프트웨어나 라이브러리를 업데이트할 때, 업데이트 파일의 출처나 디지털 서명을 검증하지 않아 악성 코드가 포함된 업데이트 파일이 설치될 수 있으므로, 모든 업데이트 파일에 대해 SHA-256 해시 또는 디지털 서명 검증을 수행하고, HTTPS 또는 코드 서명 기반의 안전한 전달 경로를 사용해야 한다.

2) 역직렬화 과정에 무결성 검증 부족 : 직렬화된 객체를 역직렬화할 때, 객체 타입, 구조, 원본 여부 등을 검증하지 않으면, 공격자가 조작된 데이터를 전송하여 원격 코드 실행, 서비스 거부, 권한 상승 등의 공격을 수행할 수 있으므로, 신뢰할 수 없는 소스로부터 역직렬화를 금지하고, JSON 기반 역직렬화 사용 시 화이트리스트 기반 필터를 적용하는 것을 권장한다.

3) CI/CD 파이프라인 보안 미흡 : 소스 코드 저장소(Git 등), 빌드 서버, 배포 자동화 시스템 등의 보안이 미흡하여, 공격자가 코드를 조작해 악성 코드를 배포하거나 백도어를 심을 수 있으므로, Git 저장소에 대한 접근 제어 강화, 빌드 로그/스크립트 검증, 코드 서명 릴리즈 해시 관리, 환경변수 암호화를 하는 것을 권장한다.

4) 데이터 무결성 검증 부재 : 애플리케이션이 설정 파일, 사용자 제공 데이터 등 중요한 요소를 사용할 때, 해당 값이 변경되지 않았음을 검증하지 않으면 중간자 공격(MITM)이나 클라이언트 조작을 통한 정보 변조가 가능해진다. 예를 들어, 클라이언트 쿠키에 저장된 로그인 상태 값이 암호화/서명 없이 노출되는 경우가 있으며, 민감 데이터에 대해 HMAC 서명 적용, TLS 사용, 서버 측 검증 수행 등을 권장한다.

5) 악성 콘텐츠 업로드 및 실행 : 업로드 된 파일의 내용이나 유형을 제대로 검증하지 않아 공격자가 Web Shell, 스크립트 파일(.js, .php 등), 실행 파일을 업로드하여 시스템 제어를 시도할 수 있으므로, 허용 확장자 화이트리스트 적용, 업로드 후 실행 폴더 분리, 콘텐츠 기반 필터링, 실행 권한 제거를 권장한다.

9. A09:2021 - Security Logging and Monitoring Failures

보안 로깅 및 모니터링 실패는 시스템에서 발생하는 보안 이벤트에 대한 기록 부재, 모니터링 미흡, 경고 체계 부재로 인해 공격을 탐지하지 못하거나, 사고 발생 후 신속한 분석 및 대응이 어려운 상황을 의미한다.

이는 단순한 로그 수집 문제가 아닌, 침해 사고 대응 체계 전반의 결함으로 이어지며, 공격자가 오랜 시간 은밀히 활동하는 것을 허용하게 되는 중대한 보안 위험이다.

▶ 주요 발생 원인 및 형태

1) 불충분한 로깅 : 로그인 성공/실패, 관리자 활동, 접근 제어 실패, 입력값 검증 실패 등 중요한 보안 이벤트를 기록하지 않거나, 기록 항목이 누락된 경우이다.

2) 로그 데이터의 부적절한 형식 또는 내용 : 로그가 일관성 없는 형식이거나, 공격 추적에 필요한 정보(예: IP, 사용자 ID, 타임스탬프, 요청 경로 등)를 포함하지 않는 경우 사고 원인 분석이 어렵기에, JSON 형식 기반의 구조화된 로그 포맷 적용, UTC 타임스탬프 및 고유 식별자를 포함하는 것을 권장한다.

3) 로그의 무결성 및 기밀성 미확보 : 로그 파일이 공격자에 의해 쉽게 삭제, 변조될 수 있는 위치에 저장되거나, 로그에 비밀번호, 세션 ID 등 민감 정보가 평문으로 기록되는 경우이다. 이 경우에는 로그 서버 분리 및 접근 제어, WORM(Write Once Read Many) 저장소 적용, 민감 정보 마스킹을 권장한다.

4) 실시간 모니터링 및 경고 시스템 부재 : 로그 수집을 기반으로 SIEM(Security Information and Event Management) 또는 IDS/IPS와 연동된 경고 시스템이 없으면 공격 탐지가 불가능하므로, Splunk, ELK, Graylog, Wazuh 등으로 실시간 탐지 구성 및 알림 임계치를 설정하는 것을 권장한다.

5) 사고 대응 계획 부족 : 침해 사고 발생 시 이를 처리하고 복구하기 위해 사전 정의된 프로세스(IRP, Incident Response Plan)가 존재하지 않으면, 피해가 확산되고 복구가 지연되므로, 보안 사고 대응 시나리오 수립, 로그 분석 담당자 지정, 주기적인 모의 훈련 실시 등을 권장한다.

10. A10:2021 - Server-Side Request Forgery (SSRF)

서버 측 요청 위조(SSRF)는 웹 애플리케이션이 서버 측에서 외부 리소스를 요청하는 기능을 제공할 때, 공격자가 이를 조작하여 서버가 대신 악의적인 요청을 수행하도록 유도하는 취약점이다.

단순 외부 요청뿐 아니라, 내부망의 민감한 리소스로의 접근 통로로 악용되며, 클라우드 환경에서는 권한 탈취, 내부 스캔, 정보 유출 등의 심각한 피해로 이어질 수 있다.

▶ 주요 발생 원인 및 형태

1) URL 기반 리소스 요청 기능 : 사용자가 입력한 URL을 통해 웹 서버가 외부 리소스를 가져오는 기능(예: URL 미리 보기, 이미지 다운로드 등)을 제공할 때, 입력된 URL에 대한 충분한 검증이 부족하면, 공격자가 이를 자가 내부망(IP: 127.0.0.1, 192.168.x.x 등)이나 클라우드 메타데이터 엔드포인트로 요청을 보내도록 조작할 수 있다. 이러한 경우 서버가 내부 시스템에 대한 프록시 역할을 하게 되어 민감한 정보 유출 또는 내부 시스템에 대한 비인가 접근이 가능해진다.

2) 내부 마이크로서비스 연동 시 검증 미흡 : 마이크로서비스 구조에서 사용자 입력을 기반으로 내부 API 주소 또는 파라미터에 대한 검증이 미흡하면 공격자가 이를 조작하여 내부 시스템과 의도치 않은 상호작용을 유도할 수 있다. 이러한 형태의 SSRF는 보안이 취약한 내부 서비스로의 접근, 서버 측 데이터 유출, 또는 추가적인 공격 벡터 확보로 이어질 수 있다.

제2절 웹 공격 트렌드 분석

1. 인공지능(AI) 기반 공격

인공지능(AI) 기술은 기존의 보안 시스템을 우회하고, 사이버 공격을 자동화하는 데 활용되고 있다. 공격자는 AI를 이용하여 보안 시스템의 취약점을 빠르게 분석하고 침투 시나리오를 빠르게 수립하고, 생성형 AI를 이용해 더욱 정교한 피싱 메시지, 악성코드, 랜섬웨어 등을 자동 생성하여 공격의 정밀성과 속도를 높이고 있다. 또한, AI는 기존의 시그니처 기반 탐지를 회피할 수 있는 변종 악성코드를 생성하고, 소셜 엔지니어링을 자동화해 사용자의 심리적 취약점을 공략한다.

AI 기반 공격의 형태는 다음과 같은 형태로 나타난다. 첫째, 공격자는 소셜 미디어 및 공개된 정보에서 수집한 정보를 바탕으로 개인화된 피싱 이메일을 자동으로 생성하여 수신자가 의심 없이 링크를 클릭하게 유도한다. 둘째, 딥페이크 기술과 결합하여 특정 인물의 음성이나 영상으로 위조된 메시지 생성, 신뢰할 수 있는 인물로 위조해 자금 이

체나 민감 정보 유출을 유도할 수 있다. 셋째, AI는 스마트 홈 기기나 IoT 장비의 보안 취약점을 탐지하고, 이를 통해 가정이나 사무실 네트워크에 침투하는데 악용될 수 있다. 이에 대응하기 위해서는 지능형 위협 탐지 및 AI 기반 보안 시스템의 도입이 필요하다.

2. 랜섬웨어 공격의 진화

랜섬웨어 공격은 단순한 데이터 암호화를 넘어, 데이터를 탈취하고 공개를 협박하는 이중 협박 전략으로 발전하고 있다. 공격 대상도 일반 기업에서 병원, 학교, 정부 기관, 핵심 인프라로 확장되고 있으며, 서비스형 랜섬웨어(RaaS)의 확산으로 공격자 간 협업이 활발해지고 있다. 이는 기술적 전문성이 없는 공격자도 쉽게 랜섬웨어를 실행할 수 있게 하여 위협의 범위를 넓히고 있다. 대표적으로 2024년에 발생한 CDK Global 및 C change Healthcare에 대한 공격 사례는 단일 공급망 업체에 대한 공격이 전체 산업에 미치는 파급력을 보여준다. 이처럼 공급망을 노린 랜섬웨어는 전체 비즈니스 생태계를 마비시킬 수 있으며, 정기적인 백업, 다단계 인증(MFA), 침입 탐지 시스템 등이 필수적인 대응 전략으로 나타난다.

3. 제로데이 취약점

제로데이 취약점은 소프트웨어나 하드웨어에서 아직 제조사나 보안 커뮤니티에 의해 인지되지 않은 결함으로, 보안 패치가 적용되기 전에 공격자가 이를 악용하는 것이 특징이다. 공격자가 이를 이용해 VPN, 방화벽, 이메일 서버 등 인터넷에 연결된 핵심 인프라를 타격할 수 있으며, 피해는 광범위하게 확산된다. 예를 들어, 2023년 발생한 MOVEit 및 Fortinet 취약점 공격은 대규모 사용자 정보 유출로 이어졌으며, 공격자의 반응 속도가 점차 빨라지고 있다. 이에 따라 제로데이 대응을 위한 위협 인텔리전스 활용, 보안 업데이트 자동화, 다중 보안 체계 적용이 중요해지고 있다.

4. API 공격

API(Application Programming Interface)는 애플리케이션 간 데이터 교환을 위해 사용되며, 현대 웹 및 모바일 환경에서 필수적인 역할을 수행한다. 그러나 API 보안 설정이 미흡할 경우 다양한 공격에 노출될 수 있다. 특히 인증과 권한 검증의 실패는 공격자가 민감한 데이터에 접근하거나 시스템을 부적절하게 사용할 수 있게 한다.

▶ API 공격의 형태

1. 깨진 인증 : 취약한 로그인 API를 대상으로 크리덴셜 스테핑이나 무차별 대입 공격을 수행해 계정 탈취를 시도한다.
2. 깨진 권한 부여 : 권한 검증이 부실한 API를 이용해 다른 사용자의 데이터에 무단으로 접근한다.
3. DoS : 요청 제한이 없는 API를 통해 대량의 요청을 전송하여 시스템을 마비시킨다.

이 외에도 입력값 검증이 미흡한 API에서는 인젝션 공격이 발생할 수 있으며, 보안 로깅 및 접근 제어 기능 부족도 API 취약점으로 악용된다. 따라서 API 개발 시 최소 권한 원칙, 입력값 유효성 검증, 속도 제한, 인증/인가 강화 등의 보안 조치가 필수적이다.

5. 클라우드 보안

클라우드 환경은 유연성과 확장성을 제공하지만, 동시에 잘못된 설정 및 보안 미흡으로 주요 공격 대상이 되고 있다. 특히 IAM 설정 오류, 퍼블릭 접근이 허용된 스토리지, 컨테이너 및 서버리스 기능의 보안 문제 등이 주요 공격 경로로 활용되고 있다. 예를 들어, 관리자가 잘못 구성한 AWS S3 버킷이나 과도하게 개방된 보안 그룹은 공격자에게 시스템 접근 기회를 제공할 수 있다. 또한, 자동화 도구를 통해 공개된 클라우드 자원이 쉽게 탐색되며, 이를 통해 데이터 유출, 서비스 중단, 권한 탈취 등의 공격이 발생한다.

클라우드 보안 사고를 예방하기 위해서는 클라우드 보안 프레임워크(CSA, NIST 등)를 따르고, 정기적인 보안 감사를 수행하며, 최소 권한 설정 및 인프라 구성 자동화 도구(IaC)의 보안 검토가 필요하다.

제3절 웹 취약점 진단 오픈 소스 스캐너

1. OWASP ZAP(Zed Attack Proxy)

1) 개요

OWASP(Open Web Application Security Project)에서 개발한 대표적인 웹 애플리케이션 보안 진단 도구로, GPL(General Public Licenses) 하에 배포되는 무료 오픈 소스 소프트웨어로 초보자부터 전문가까지 모두 사용할 수 있도록 설계되었다. GUI 기반과 명령줄 기반 인터페이스를 모두 제공하여 사용 편의성이 뛰어나다.

ZAP은 프록시 기반으로 작동하며, 사용자가 웹 애플리케이션과 상호 작용하는 과정을 기록하고 이를 분석하여 다양한 취약점을 탐지한다. 기본적으로 자동화된 취약점 스캐너와 수동 테스트 기능을 모두 제공하며, 플러그인 형태로 기능 확장이 가능하다.

특히 지속적 통합(CI) 환경이나 DevOps 파이프라인과의 연동 기능을 제공하여 DevSecOps 실현에 유용하다.

2) 주요 기능

1. 자동화된 취약점 스캐닝
2. 프록시를 통한 실시간 트래픽 분석
3. Spider 및 AJAX Spider를 통한 크롤링
4. Fuzzer를 활용한 사용자 정의 요청 테스트

5. 다양한 보고서 형식(HTML, XML, JSON 등) 제공

이러한 기능을 통해 ZAP은 대규모 애플리케이션뿐만 아니라 개발 단계의 보안 검증 도구로도 활용되고 있다.

2. Arachni

1) 개요

Arachni는 Ruby 기반으로 개발된 고성능 웹 애플리케이션 보안 진단 프레임워크로, 고도화된 탐지 로직과 유연한 확장성을 갖추고 있다. 이 도구는 단순한 스캐닝 도구에 그치지 않고, 웹 애플리케이션의 동작 패턴을 모니터링하고 분석하는 학습 기반의 진단 기능을 지원한다.

Arachni는 다양한 플랫폼에서 사용 가능하며, 명령줄 및 웹 UI를 제공하고, REST API를 통해 자동화도 가능하다. 특히 분산 진단 기능을 지원하여 대규모 웹 자산에 대한 병렬 분석이 가능하다.

2) 주요 기능

1. XSS, SQL Injection, CSRF 등 주요 웹 취약점 자동 탐지
2. 크롤러 기능 및 세션 유지 지원
3. Fuzzer 기반 입력 필드 탐색
4. 진단 결과의 다양한 출력 형식 지원(HTML, XML, JSON 등)
5. 플러그인 기반의 확장성 및 스크립트 기반 자동화 지원

Arachni는 다양한 분석 요소를 활용하여 탐지 결과의 신뢰도를 자체적으로 평가하며, 오탐을 최소화하기 위한 필터링 기능도 제공한다.

3. Wapiti

1) 개요

Wapiti는 Python 기반의 경량 웹 애플리케이션 보안 진단 도구로, 블랙박스 방식의 점검을 수행한다. 이는 소스 코드에 대한 접근 없이, 실제 사용자의 관점에서 HTTP 요청과 응답을 분석함으로써 취약점을 진단하는 방식이다.

Wapiti는 크롤링과 퍼징을 기반으로 웹 페이지를 탐색하고, 입력 필드를 식별하여 다양한 공격 벡터를 적용한다. 실제 서비스 환경에서 발생할 수 있는 보안 취약점을 효과적으로 탐지할 수 있도록 설계되었으며, 설치 및 사용이 간편하다는 장점이 있다.

2) 주요 기능

1. XSS, Command Injection, File Disclosure 등 주요 취약점 진단
2. HTTP 요청의 무결성 테스트 및 헤더 분석
3. SWF, JavaScript 내 URL 추출, HTML5 태그 인식
4. 쿠키 및 세션 인증 유지 가능

5. Wappalyzer DB를 활용한 CMS 및 프레임워크 식별

진단 결과는 HTML, TXT, XML 등의 다양한 형식으로 저장 가능하며, 보고서 내에는 각 취약점의 세부 정보 및 대응 방안도 포함되어 있어 실무 활용도도 높다.

제4절 KISA 가이드라인

Web 취약점 분석·평가 항목

점검항목	항목 중요도	항목코드
버퍼 오버플로우	상	BO
포맷스트링	상	FS
LDAP 인젝션	상	LI
운영체제 명령 실행	상	OC
SQL 인젝션	상	SI
SSI 인젝션	상	SS
XPath 인젝션	상	XI
디렉터리 인덱싱	상	DI
정보 누출	상	IL
악성 콘텐츠	상	CS
크로스사이트 스크립팅	상	XS
약한 문자열 강도	상	BF
불충분한 인증	상	IA
취약한 패스워드 복구	상	PR
크로스사이트 리퀘스트 변조(CSRF)	상	CF
세션 예측	상	SE
불충분한 인가	상	IN
불충분한 세션 만료	상	SC
세션 고정	상	SF
자동화 공격	상	AU
프로세스 검증 누락	상	PV
파일 업로드	상	FU
파일 다운로드	상	FD
관리자 페이지 노출	상	AE
경로 추적	상	PT
위치 공개	상	PL
데이터 평문 전송	상	SN
쿠키 변조	상	CC

KISA의 주요통신기반시설 취약점 분석 및 평가 방법 상세가이드의 웹 취약점 분석 및 평가 항목에 대한 점검항목, 항목 중요도, 항목 코드에 대한 사진이다. KISA에서 웹 애플리케이션의 보안성을 체계적으로 분석하고 강화하기 위해 28개의 취약점 항목을 제시하고 있는데, 이 항목들은 웹 애플리케이션에서 발생할 수 있는 주요 보안 위협 요소를 기반으로 구성되며, 각 항목은 실질적인 공격 가능성과 보안 위협의 중요도를 기준으로 평가된다. 모든 항목이 ‘상’ 등급의 중요도로 실질적인 웹 취약점 점검을 할 때, 반드시 보안 적용을 해야 하는 항목들로 구성되어 있다.

제3장 웹 취약점 진단 및 진단 스크립트

제1절 오픈 소스 스캐너를 이용한 웹 진단

오픈 소스 스캐너를 이용한 웹 진단은 테스트 페이지(<http://testphp.vulnweb.com/>, <http://testasp.vulnweb.com/>, <http://demo.testfire.net/>)에 오픈 소스 스캐너를 이용하여 테스트 페이지에 존재하는 취약점을 찾아내고, 어떠한 방식으로 취약점에 대해 탐지하는지를 확인한다. 이 연구에서는 OWASP ZAP, Arachni, Wapiti 등 대표적인 오픈 소스 웹 취약점 스캐너를 활용하여 각각 다른 기술 기반의 테스트 페이지에 대해 진단을 수행하고, 도구별 탐지 항목, 스캔 방식, 탐지 결과에 대해 분석한다.

테스트는 각 스캐너를 동일한 환경에 설치 및 구성한 후, 대상 웹 페이지에 URL을 입력하여 취약점 진단을 시작한다. 스캐너들의 탐지 능력과 사용 편의성, 결과 보고서의 가독성, 그리고 실제 보안 점검 시 활용 가능성 등에 대해서 알아가고자 한다.

1. OWASP ZAP

진단 항목/사이트	testphp.vulnweb.com	demo.testfire.net	testaspnet.vulnweb.com
CSRF 방지 토큰 없음	O	O	O
CSP 미설정	O	O	O
Clickjacking 보호 미비	O	O	O
HTTPS 페이지에 HTTP 콘텐츠 포함 (스크립트 등)	X	O	X
HSTS 미설정	X	X	O
디버그 메시지를 통한 정보 노출	X	O	O
쿠키 Secure 플래그 미설정	X	X	X
SameSite 속성 없음 또는 None 설정된 쿠키	X	O	O
서버 정보 노출	O	O	O
X-Content-Type-Options 헤더 미설정	O	O	O
오류 메시지 노출	X	O	O
기타 정보성	O	O	O

사이트/위험도(개수)	Medium	Low	Informational
testphp.vulnweb.com	3	5	4
demo.testfire.net	4	7	4
testaspnet.vulnweb.com	3	5	5

- OWASP ZAP 진단 결과, 세 개의 테스트 사이트 모두 다양한 취약점이 발견되었다. 특히 CSP, Clickjacking 보호 미비, X-Content-Type-Options 설정과 같은 보안 헤더의 부재는 많은 웹 사이트에서 쉽게 발견되고 조치할 수 있는 부분이다. 또한 CSRF 토큰이나 쿠키의 SameSite 속성처럼 세션 관리와 관련된 취약점은 사용자의 권한을 탈취하는 심각한 공격으로 이어질 수 있다.

2. Arachni

진단 항목/사이트	testphp.vulnweb.com	demo.testfire.net	testaspnet.vulnweb.com
XSS	O	O	O
SQL Injection	O	X	O
Code injection	O	X	X
CSRF	O	X	O
Path Traversal	X	O	X
Insecure Password Form	O	O	O
Common directory	O	X	X
Backup file, directory	O	X	X
Insecure Cross-Domain Policy	O	X	X
Unvalidated Redirect	O	X	X
Missing X-Frame-Options	O	O	O
Common Sensitive Files	O	X	O
Password field with auto-complete	O	O	O

사이트/위험도	High	Medium	Low	Informational
testphp.vulnweb.com	7	5	5	2
demo.testfire.net	3	1	2	2
testaspnet.vulnweb.com	4	1	3	2

- Arachni 진단 결과, 세 개의 사이트 모두 다양한 취약점이 발견되었다. 특히 testphp.vulnweb.com이 가장 다양한 유형의 취약점을 종합적으로 가지고 있는 것이 발견되었고, Arachni를 통해 웹 취약점 진단을 통해 다양한 인젝션 취약점을 식별하고, XSS나 보안 헤더 부재와 같은 공통적인 취약점을 탐지하는데 효과적이었다.

3. Wapiti

진단 항목/사이트	testphp.vulnweb.com	demo.testfire.net	testaspnet.vulnweb.com
CSP 미설정	O	O	O
Clickjacking 보호 미비	O	O	O
Reflected XSS	O	O	X
HttpOnly 미사용 쿠키	X	X	X
Secure 미사용 쿠키	X	O	X
HSTS 미설정	X	X	X
MIME Type Confusion	O	O	O
Internal Server Error	O	O	X
기타 정보성	O	X	X

사이트/위험도(개수)	Medium	Low	Informational
testphp.vulnweb.com	3	1	2
demo.testfire.net	3	2	1
testaspnet.vulnweb.com	2	1	0

- Wapiti 진단 결과, 세 개의 테스트 사이트 모두 방어적인 보안 설정, 특히 HTTP 보안 헤더 적용이 매우 미흡하다는 공통점을 보였다. 이러한 보안 설정들은 XSS, Clickjacking과 같은 공격의 피해를 크게 줄일 수 있는 역할을 한다. 또한 testphp.vulnweb.com과 demo.testfire.net에서는 XSS와 같은 실제 공격 가능한 취약점이 발견되어 즉시 조치가 필요함을 확인했다.

4. 결론

OWASP ZAP은 애플리케이션을 다운로드 받고 실행시켜 URL만 입력하면 복잡한 설정 없이도 취약점 진단을 진행할 수 있기에 초기 접근성이 뛰어나며, 특히 ZAP 마켓

플레이스를 통해 제공되는 다양한 애드온은 스캔 규칙, 보고서 템플릿, 인증 핸들러 등 ZAP의 기능을 특정 요구사항에 맞게 확장시켜 주며, 사용자 커뮤니티를 통해 학습 자료와 문제 해결에 대한 지원을 받을 수 있다. ZAP의 “Use traditional spider”라는 옵션을 사용할 때, 스캔이 완료되지 않는 현상이 발생할 수 있는데, 이는 주로 동적으로 무한히 생성되는 URL 경로(스파이더 트랩), 매우 큰 웹사이트의 구조, 스캔 범위의 부정확한 설정, JavaScript에 의해 동적으로 로드되는 콘텐츠를 traditional spider가 효과적으로 처리하지 못하는 특성 등에서 발생할 수 있다. 이러한 현상을 해결하기 위해서는 스캔 전 스캔 범위를 명확히 정의하고, 스파이더의 탐색 깊이 및 최대 실행 시간 제한하며, 특정 URL 패턴 제외하는 등의 설정을 조정하는 것이 중요하다. 또한 JavaScript 기반의 동적 콘텐츠가 많은 최신 웹 애플리케이션 환경에서는 HTML 태그 위주로 탐색하는 traditional spider보다는 AJAX Spider를 활용하는 것이 더 효과적인 결과를 가져올 수 있다. 또한 10가지 이상의 보고서 템플릿이 있으며, 만약에 3개의 웹 사이트를 검사를 진행했다면, 검사 결과가 저장되어 애플리케이션을 종료하기 전까지 보고서를 생성할 때, 검사한 URL을 선택하여 검사했던 결과에 대한 보고서를 다운받을 수 있다.

Arachni는 Arachni Web UI를 이용하여 검사를 진행하였는데, 초기 설정 단계는 3개의 진단 도구 중에서 가장 시간이 필요했지만, UI에 접속한 후에는 스캔을 진행하기 어려움이 크게 없었다. 특히 Arachni에서 제공해주는 취약점 검사 보고서는 3개의 진단 도구 중 가장 좋았다고 생각한다. 3개의 페이지로 나누어져 있는데, ‘charts’ 페이지는 검사 결과를 다양한 차트를 활용하여 시각적으로 보여주고, ‘issues’ 페이지는 각 취약점의 종류와 발생 위치에 대해 알려주며, ‘OWASP Top 10’ 페이지는 OWASP Top 10 항목에 해당하는 취약점들을 따로 정리해서 보여준다.

위와 같이 여러 관점에서 정보를 다양한 방법으로 제공하여 가독성이 뛰어나, 가장 좋았던 보고서로 생각했다.

Wapiti는 명령 프롬프트에서 pip를 이용해 wapiti3를 다운로드하면 바로 사용할 수 있어 설치가 매우 간단하다. Wapiti는 wapiti --help 명령어를 통해 사용 가능한 모든 옵션과 각 옵션에 대한 상세한 설명이 출력되어, 이를 참고하여 필요한 스캔 명령을 구성할 수 있다.

취약점 검사를 진행할 때는 -u 옵션을 이용해 스캔 대상 URL을 입력하고, -o 옵션을 통해 분석 결과가 저장될 경로를 지정하며, -f 옵션으로 보고서 형식을 지정할 수 있다. 이 외에도 스캔 범위를 직접 설정하거나 특정 유형의 취약점만을 선택하여 검사하는 등, 사용자가 스캔의 범위와 깊이를 자세하게 설정할 수 있어 스캔 시간을 효과적으로 단축할 수 있다.

제2절 KISA 가이드라인에 따른 웹 진단 스크립트 제작

전체 코드 : https://github.com/hybinn/Web_Scanner

이 연구에서는 KISA의 주요통신기반시설 웹 취약점 분석 및 평가 방법 가이드라인을 기반으로 웹 애플리케이션의 보안 진단을 자동화하는 스크립트를 Python으로 제작하였다. 해당 가이드라인은 웹 서비스에서 발생할 수 있는 다양한 보안 취약점에 대해 총 28개의 항목으로 구성되어있으며, 각 항목별로 점검 방법과 판단 기준이 명확히 제시되어 있어 이를 체계적으로 자동 점검에 반영할 수 있다.

스크립트는 URL을 입력받아 검사를 진행하고, 인증이 필요한 항목을 제하고 14가지의 취약점에 대해 검사를 해준다. 각 항목에 대해 취약할 경우 마지막에 취약한 부분에 대하여 출력을 해준다. 이러한 자동 진단 스크립트는 보안 전문가가 아닌 사용자도 일정 수준의 보안 진단을 수행할 수 있도록 도와주며, 반복적이고 표준화된 점검 업무의 효율성을 높이는 데 기여할 수 있다.

1. 필드 찾기

- 메인 페이지의 모든 <a> 태그를 분석하여 href 속성 값이나 링크 텍스트에 'login', 'signin', 'register' 등과 같은 로그인 관련 키워드로 로그인 페이지를 찾아서 저장하고, <form> 태그를 분석하여 type이 'text', 'search'인 <input>태그나 name에 'search', 'query', 'q' 등이 포함된 <input> 태그를 검색 폼으로 저장하고, 탐색한 로그인 페이지에서 아이디와 비밀번호 입력 필드의 name을 특정하여, 아이디는 'user', 'id', 'email', 비밀번호는 'password', 'pwd', 'pass' 등으로 키워드를 탐색하여 정확한 아이디, 비밀번호의 파라미터 이름을 탐색하여 저장한다.

```
login_links = []
login_keywords = ['login', 'signin', 'sign-in', 'sign_in', 'admin', 'log-in', 'log_in']

for a in soup.find_all('a', href=True):
    href = a.get('href', '').lower()
    text = a.get_text(strip=True).lower()

    if any(keyword in href or keyword in text for keyword in login_keywords):
        full_url = urljoin(self.target_url, a['href'])
        login_links.append(full_url)

self.login_urls = login_links
```

2. 버퍼 오버플로우

- 사전에 탐지된 로그인 폼과 검색 폼에 5000자 길이의 대량 문자열을 입력하여 버퍼 오버플로우 취약점을 테스트한다. 로그인 폼에서는 식별된 username, password 필드의 정확한 파라미터명을 사용하고, 검색 폼에서는 'q', 'query', 'search' 파라미터를 사용한다.

```
def buffer_overflow(self):
    long_str = 'A' * 5000
    test_cases = [
        f"{self.target_url}?test={long_str}",
        f"{self.target_url}/search?query={long_str}",
    ]
```

3. 포맷스트링

- 포맷스트링 취약점 탐지를 위해 %n, %s 등의 포맷 지정자를 포함한 악성 패턴을 생성하여 테스트한다. 메모리 쓰기를 유발하는 %n 패턴과 문자열 출력을 유발하는 %s 패턴을 URL 파라미터로 전송한다.

```
patterns = [
    "%n%n%n%n%n%n%n%n%n%n%n",
    "%s%s%s%s%s%s%s%s%s%s",
    "%1!n!%2!n!%3!n!%4!n!%5!n!%6!n!%7!n!%8!n!%9!n!%10!n!",
    "%1!s!%2!s!%3!s!%4!s!%5!s!%6!s!%7!s!%8!s!%9!s!%10!s!"
]
```

4. LDAP 인젝션

- LDAP 인젝션 취약점 탐지를 위해 LDAP 필터 구문의 특수문자('*', ')', '(', '&', '|')를 조작한 악성 페이로드를 생성한다. 사전에 식별된 로그인 폼의 username, password 필드에 각 페이로드를 삽입하여 POST/GET 요청을 전송하고, 서버 응답에서 HTTP 500 에러 또는 'ldap', 'directory', 'error' 등의 LDAP 관련 에러 키워드를 탐지하여 취약점 존재 여부를 판단한다.

```
def ldap_injection(self):
    ldap_payloads = [
        "*",
        "admin(&))",
        "*)(&",
        ")(cn=*))",
        "*)(|&' ",
        "*(|(objectclass=*))",
        "*)(uid=*)(|(uid=",
        "admin*)((|userpassword=*)"
    ]
```

5. 운영체제 명령 실행

- 운영체제 명령 실행 취약점 탐지를 위해 셸 메타문자(';', '|', '&', '', '\$()') 와 시스템 명령어를 조합한 악성 페이로드를 생성한다. URL 파라미터를 통해 페이로드를 전송하고, 서버 응답에서 'root:', 'uid=', 'Directory of' 등 명령어 실행 결과를 나타내는 특정 패턴을 탐지하여 취약점을 판단한다.

```
def os_command_injection(self):  
    os_payloads = [  
        '; ls -la',  
        '| dir',  
        '& whoami',  
        '; cat /etc/passwd',  
        '`id`',  
        '$(whoami)'  
    ]
```

6. SQL 인젝션

- SQL 인젝션 취약점 탐지를 위해 논리적 항상 참 조건(' OR '1'='1'), 주석 처리('--'), UNION 기반 공격 등의 다양한 SQL 페이로드를 생성한다. 사전에 식별된 로그인 폼의 username, password 필드와 hidden 필드를 포함한 모든 폼 데이터를 구성하여 완전한 요청을 전송한다.

```
def sql_injection(self):  
    sql_payloads = [  
        "' OR '1'='1",  
        "' OR 1=1--",  
        "' UNION SELECT NULL--",  
        "admin'--",  
        "' AND 1=2 UNION SELECT 1,version()--"  
    ]
```

7. SSI 인젝션

- SSI 인젝션 취약점 탐지를 위해 웹 서버의 SSI 지시자를 활용한 악성 페이로드를 생성한다. '`<!--#echo var="DOCUMENT_ROOT" -->`' 를 통한 환경변수 출력, '`<!--#exec cmd="ls -al" -->`' 를 통한 시스템 명령 실행, '`<!--#include virtual="/etc/passwd" -->`' 를 통한 파일 포함 공격을 수행한다. URL 파라미터와 검색 폼의 GET/POST 방식 모두에서 테스트하며, 서버 응답에서 '/var/www', 'root:', 'drwx' 등 명령어 실행 결과나 시스템 정보가 노출되는지 탐지하여 취약점을 판단한다.

```
ssi_payloads = [  
    '<!--#echo var="DOCUMENT_ROOT" -->',  
    '<!--#exec cmd="ls -al" -->',  
    '<!--#include virtual="/etc/passwd" -->'  
]  
vulnerable = False  
details = ""  
ssi_messages = ['/var/www', '/usr', 'root:', 'drwx', 'total ']
```

8. XPath 인젝션

- XPath 인젝션 취약점 탐지를 위해 2단계 검증 방식을 사용한다. 1단계에서는 논리적 항상 참 조건('' and 'a'='a')과 항상 거짓 조건('' and 'a'='b')을 각각 전송하여 서버 응답의 길이나 상태 코드 차이를 비교한다. 2단계에서는 XPath 특화 함수인 count()와 parent:: 축을 활용한 복합 페이로드를 사용하여 XML 구조 기반 공격을 수행한다.

```
step2_payloads = [  
    "' or count(parent::*[position()=1])=0 or 'a'='b",  
    "' or count(parent::*[position()=1])>0 or 'a'='b",  
    "1 or count(parent::*[position()=1])=0",  
    "1 or count(parent::*[position()=1])>0"  
]
```

9. 디렉터리 인덱싱

- 디렉터리 인덱싱 취약점 검사는 웹 서버에서 디렉터리 목록이 노출되는 보안 취약점을 탐지하는 과정이다. 일반적으로 중요한 정보가 저장될 가능성이 높은 디렉터리 경로(/admin/, /backup/, /uploads/ 등)에 대해 직접 접근을 시도하고, URL 인코딩 및 파일 확장자 추가 등의 우회 기법을 적용하여 서버의 디렉터리 구조가 노출되는지 확인한다.

```
def directory_indexing(self):  
    test_dirs = ['/admin/', '/backup/', '/uploads/',  
                 '/files/', '/images/']
```

- 10. XSS (크로스사이트 스크립팅)

- XSS 취약점은 웹 애플리케이션에서 사용자 입력 데이터의 적절한 검증 및 필터링 없이 출력될 때 발생하는 보안 취약점을 탐지하는 과정이다. 다양한 XSS 페이로드(<script>, , <svg> 태그 등)를 URL 파라미터, 검색 폼, 로그인 폼 등의 입력 지점에 삽입하여 서버 응답에서 해당 스크립트가 필터링 없이 그대로 반영되는지 확인한다. 이를 통해 Reflected XSS, Stored XSS, DOM-based XSS 등의 취약점 존재 여부를 판단할 수 있다.

```
def xss(self):  
    payloads = [  
        "<script>alert('XSS')</script>",  
        "<img src=x onerror=alert('XSS')>",  
        "javascript:alert('XSS')",  
        "<svg onload=alert('XSS')>"  
    ]
```

11. 약한 문자열 강도

- 약한 인증 취약점 검사는 웹 애플리케이션의 인증 보안을 평가하는 종합적인 과정으로, 두 가지 주요 검사를 수행한다. 첫째, 약한 자격증명 검사에서는 일반적으로 사용되는 기본 계정(admin/admin, root/root 등)과 추측 가능한 패스워드 조합을 시도하여 무차별 대입 공격(Brute Force Attack)에 취약한지 확인한다. 둘째, 계정 잠금 정책 검사에서는 동일한 사용자 계정으로 연속해서 인증 실패를 발생시켜(일반적으로 3-5회) 시스템이 적절한 계정 잠금 메커니즘을 적용하는지 검증한다.

```
def weak_auth(self):
    login_url = self.login_urls[0]
    vulnerable = False

    weak_credentials = [
        ('admin', 'admin'), ('admin', 'password'), ('admin', '1234'), ('admin', ''),
        ('administrator', 'administrator'), ('manager', 'manager'),
        ('guest', 'guest'), ('test', 'test'), ('scott', 'scott'),
        ('tomcat', 'tomcat'), ('root', 'root'), ('user', 'user'),
        ('operator', 'operator'), ('anonymous', 'anonymous')
    ]
```

12. 세션 예측

- 웹 서버에 5회 연속 접속하여 발급받은 세션 ID들의 숫자 부분 간 차이가 모두 100 이하일 때 세션 ID가 예측 가능하다고 판단한다.

```
if all(diff <= 100 for diff in diffs):
    vulnerable = True
    details = "세션 ID 예측 가능"
```

13. 자동화 공격

- 로그인 시도, POST, GET 요청 등 다양한 유형의 요청을 반복적으로 전송할 때, 반복적인 요청에 대해 차단하는 정책이 있는지 확인한다.

```
def auto_attack(self):
    vulnerable = False

    try:
        block_keywords = [
            'captcha', 'rate limit', 'too many', 'blocked',
            'suspicious', 'temporarily', 'verification'
        ]
        login_data = {'username': 'test', 'password': 'test'}

        for i in range(5):
            if self.login_urls:
                login_url = self.login_urls[0]
                if login_url in self.login_form_fields:
                    form_fields = self.login_form_fields[login_url].copy()
                    username_field = form_fields.get('username', 'username')
                    password_field = form_fields.get('password', 'password')
```

14. 관리자 페이지 노출

- 관리자 페이지의 로그인 창에 기본 관리자 계정(admin, administrator, manager 등) 및 패스워드를 입력하여 로그인이 되는지 확인하고, 추측하기 쉬운 관리자 페이지 경로(/admin, /manager, /master, /system 등), 추측하기 쉬운 포트(7001, 8080, 8443, 8888 등) 접속을 시도하여 관리자 페이지가 노출되는지 확인한다.

```
def admin_page_exposure(self):
    if self.admin_login_success:
        details = f"{self.successful_login_detail}"
        self.log_result("관리자 페이지 노출", True, details)
        return
    admin_paths = [
        '/admin', '/administrator', '/manager', '/master', '/system', ':7001', ':8080', ':8443', ':8888'
    ]
```

15. 경로 추적

- 웹 브라우저에 표시할 페이지를 지정하는 파라미터 값을 임의의 경로가 포함된 값으로 변조 후 전송하여 해당 경로의 파일 내용이 웹 브라우저에 표시되는지 확인한다.

```
def path_route(self):
    path_payloads = [
        '../../../../../../../../../../../../etc/passwd',
        '../../../../../../../../../../../../winnt/win.ini',
        '../../../../../../../../../../../../boot.ini',
    ]
```

제3절 제작한 스크립트를 이용한 웹 진단

취약점/사이트	testphp.vulnweb.com	demo.testfire.net	testaspnet.vulnweb.com
버퍼 오버플로우	O	O	O
포맷스트링	X	X	X
LDAP 인젝션	O	O	X
운영체제 명령 실행	X	X	X
SQL 인젝션	X	O	O
SSI 인젝션	X	X	X
XPath 인젝션	X	X	O
디렉터리 인덱싱	O	O	O
XSS	O	O	X
약한 문자열 강도	O	O	O
세션 예측	X	X	X
자동화 공격	O	O	O
관리자 페이지 노출	O	O	X
경로 추적	X	X	X

- 자동화된 웹 취약점 진단 스크립트를 활용하여 총 세 개의 사이트에 대해 주요 보안 취약점을 진단하였다. 진단 결과, 세 사이트 모두에서 공통적으로 발견된 취약점은 버퍼 오버플로우, LDAP 인젝션, SQL 인젝션, 디렉터리 인덱싱, XSS, 약한 문자열 강도, 자동화 공격, 관리자 페이지 노출 등이었다. 반면, 포맷스트링, 운영체제 명령 실행, SSI 인젝션, 세션 예측, 경로 추적 등의 취약점은 자동화 진단에서는 발견되지 않았지만, 해당 사이트들이 해당 취약점에 대해 안전하다는 것을 의미할 수 있으나, 동시에 자동화 도구의 한계로 실제 취약점이 존재하지 않는다고 배제할 수 없다.

제4장 결 론

제1절 결과 분석

오픈 소스 웹 취약점 스캐너 OWASP ZAP, Arachni, Wapiti와 직접 제작한 Python 기반 스크립트를 이용하여 동일한 웹 사이트의 취약점에 대한 진단하였다. 3개의 오픈 소스 웹 취약점 스캐너마다 취약점을 분류하는 이름이나 탐지하는 범위도 달랐다. 오픈 소스 스캐너는 광범위한 취약점 데이터베이스와 다양한 탐지 기법을 기반으로 포괄적인 진단을 진행해주는데, 특히 OWASP ZAP은 AJAX Spider를 진행할 때, Selenium을 활용하여 진행 상태에 대해 볼 수 있었고, Wapiti가 탐지 속도의 측면에서는 가장 빨랐고, 보고서 측면에서는 Arachni가 가장 좋았다. 제작한 스크립트는 KISA 주요통신기반시설 취약점 분석 및 평가 방법 상세 가이드의 점검 방법과 판단 기준에 초점을 맞추어 진단 하도록 만들었고, 오픈 소스 스캐너는 포괄적이고 정밀한 진단을 해주지만, 내가 제작한 스크립트는 KISA 가이드라인의 점검 방법과 판단 기준을 기반으로 URL을 입력하면 빠르고 효율적인 진단을 해주지만, 복잡한 취약점에 대해서는 탐지에 한계가 존재한다.

제2절 향후 연구 계획

본 연구에서는 KISA의 웹 취약점 진단 가이드라인을 바탕으로 주요 항목에 대해 자동으로 점검할 수 있는 스크립트를 개발하고, 이를 오픈 소스 스캐너와 비교 분석하였다. 그러나 본 스크립트는 몇 가지 제한점을 가지고 있으며, 이를 개선하여 보다 효과적으로 적용할 수 있도록 향후 다음과 같은 방향으로 연구하고자 한다.

첫째는, 현재 스크립트는 인증이 필요한 페이지나 로그인 이후 영역에 대한 진단이 불가능하다. 따라서 향후에는 세션 유지 기능 및 로그인 자동화 기능을 도입하여 인증 기반 취약점에 대한 분석을 가능하게 할 생각이다.

둘째는, 복합적인 공격 벡터나 다단계 요청 구조를 분석할 수 있는 기능이 부족하다. 이를 보완하기 위해 요청 흐름 추적 기능, DOM 기반 취약점 진단 기능을 강화하고 Java Script 렌더링 분석할 수 있도록 Selenium 등 브라우저 자동화 도구와의 연계를 고려하고 있다.

마지막으로, 진단 결과에 대한 보고서 생성 기능을 추가하여, 사용자가 취약점 현황을 보고서 형태로 저장할 수 있게 할 것이다.

참고문헌

1. OWASP Top 10:2021 <https://owasp.org/Top10/>
2. 주요정보통신기반시설 기술적 취약점 분석 평가 상세 가이드
https://www.kisa.or.kr/2060204/form?postSeq=12&lang_type=KO&page=1
3. OWASP Zed Attack Proxy (ZAP) <https://www.zaproxy.org/blog/>
4. Arachni Github <https://github.com/Arachni/arachni>
5. Wapiti Github <https://github.com/wapiti-scanner/wapiti>
6. KISA 2024년 하반기 사이버 위협 동향 보고서
<https://www.kisa.or.kr/20205/form?postSeq=1027&page=1#fndoDocumentPreview>
7. SentinelOne, 사이버보안
<https://www.sentinelone.com/cybersecurity-101/cybersecurity/>
8. SentinelOne, 위협 인텔리전스
<https://www.sentinelone.com/cybersecurity-101/threat-intelligence/>
9. <https://www.skshieldus.com/blog-security/security-trend-idx-39>
10. https://image.ahnlab.com/atip/content/file/20241203/AhnLab_Annual%20Threat%20Analysis%20and%20Prediction_2024_KR.pdf
11. <https://www.skshieldus.com/blog-security/security-trend-idx-21>
12. <https://www.boannews.com/media/view.asp?idx=131817>
13. OWASP Cheat Sheet Series <https://cheatsheetseries.owasp.org/>
14. <https://www.boannews.com/media/view.asp?idx=100824&page=1&kind=1>
15. <https://www.skshieldus.com/blog-security/security-trend-idx-46>
16. <https://www.sangfor.com/kr/cybersecurity/innovations/ransomware-evolution>
17. <https://www.boannews.com/media/view.asp?idx=92617>
18. <https://cybersecurity.asee.io/blog/cybersecurity-statistics/>
19. <https://global.ptsecurity.com/analytics/web-vulnerabilities-2020-2021>
20. <https://purplesec.us/resources/cybersecurity-statistics/>