**OWASP 2021을 기반으로 한 웹보안 교육 (취약하고 오래된 요소)**

**1. OWASP와 취약하고 오래된 구성요소 취약점의 이해**

**학습목표**

* OWASP Top 10의 개념과 목적을 이해한다.
* A06 취약하고 오래된 구성요소 취약점의 정의와 특징을 파악한다.
* 해당 취약점이 야기할 수 있는 위험성과 영향 범위를 인식한다.
* 국내 주요 보안사고 사례를 통해 취약점의 파급효과를 이해한다.

**학습내용**

**1-1. OWASP 개요 및 A06 취약점 소개**

웹 애플리케이션 보안의 핵심 지침서인 OWASP Top 10 중에서도 특히 중요한 A06 취약하고 오래된 구성요소(Vulnerable and Outdated Components) 취약점에 대해 알아보겠습니다.

OWASP는 Open Web Application Security Project의 약자로, 웹 애플리케이션 보안 강화를 목적으로 하는 국제적인 비영리 기구입니다. 이 기구는 전 세계적인 협력을 통해 웹 애플리케이션의 보안 취약점을 분석하고 이를 정리한 'OWASP Top 10'을 주기적으로 발표합니다. 최신 OWASP Top 10 목록은 2021년에 발표되었으며, 다음과 같은 보안 취약점들이 포함되었습니다.

A01(접근 권한 취약점) : 사용자가 권한 없이 민감한 데이터나 기능에 접근할 수 있는 문제입니다. 예를 들어, 관리 권한이 없는 사용자가 내부 관리자 페이지에 접근할 수 있는 경우입니다.

A02(암호화 오류) : 민감한 데이터가 적절히 암호화되지 않거나 안전하지 않은 암호화 방식이 사용될 때 발생합니다. 암호화 키 관리의 부주의도 여기에 포함됩니다.

A03(인젝션) : SQL, 커맨드 라인, 또는 LDAP 인젝션과 같이 외부 입력값이 실행 코드에 포함되어 의도하지 않은 동작을 유발하는 공격입니다.

A04(안전하지 않은 설계) : 보안 위협을 고려하지 않은 시스템 설계로 인해 발생하는 취약점입니다. 이는 설계 단계에서 이미 예방할 수 있는 문제입니다.

A05(보안 설정 오류) : 보안 설정이 제대로 적용되지 않았거나, 기본 설정(default)이 안전하지 않을 때 발생하는 문제입니다.

A06(취약하고 오래된 구성요소) : 애플리케이션에서 사용하는 서드파티 라이브러리, 프레임워크 또는 운영체제(OS) 등에서 취약점이 발견되었지만, 최신 버전으로 업데이트되지 않을 경우 발생합니다. 이는 현대 소프트웨어 개발에서 오픈소스 라이브러리와 상용 컴포넌트를 광범위하게 사용함에 따라 더욱 중요한 보안 요소로 부각됩니다. 구성요소의 취약점이 있으면 애플리케이션 전체가 취약해질 수 있기 때문입니다. 따라서 이를 주기적으로 점검하고 업데이트하는 것이 필수적입니다.

A07(식별 및 인증 오류) : 사용자의 인증 및 권한 처리 과정에서 발생하는 문제로, 약한 패스워드 및 불완전한 세션 관리 등이 포함됩니다.

A08(소프트웨어 및 데이터 무결성 오류) : 신뢰할 수 없는 소스에서 데이터를 불러오거나 코드를 실행하여 발생하는 문제입니다. 이는 공급망 공격과 관련이 있습니다.

A09(보안 로깅 및 모니터링 실패) : 시스템의 보안 관련 이벤트를 충분히 로깅하지 않거나, 이를 적시에 모니터링하지 않아 공격을 탐지하지 못하는 문제입니다.

A10(서버 측 요청 위조) : 공격자가 서버를 통해 임의의 요청을 실행하도록 유도하는 취약점으로, 민감한 데이터 유출이나 리소스 접근 문제를 야기할 수 있습니다.

특히, A06(취약하고 오래된 구성요소)는 기업에게 있어 주의가 필요한 항목입니다. 현대 개발 환경에서 라이브러리와 프레임워크를 적극적으로 활용하는 만큼, 구성요소를 주기적으로 점검하고 최신 상태로 유지해야 애플리케이션 보안을 강화할 수 있습니다. 반대로 이를 간과할 경우, 애플리케이션 전체가 심각한 보안 위협에 노출될 수 있습니다. 유지보수와 라이브러리 관리가 곧 웹 애플리케이션 보안의 핵심이라고 할 수 있습니다.

**학습정리**

* OWASP는 웹 애플리케이션 보안 향상을 목적으로 하는 국제 비영리 기구다.
* OWASP Top 10은 가장 중요한 웹 애플리케이션 보안 취약점 10가지를 정리한 목록이다.
* A06(취약하고 오래된 구성요소)는 지원 종료되었거나 알려진 취약점을 포함한 컴포넌트 사용 시 발생하는 위험이다.
* 현대 개발 환경에서는 다양한 서드파티 컴포넌트를 사용하므로 이에 대한 관리가 필수적이다.

**1-2. 취약하고 오래된 구성요소의 위험성과 영향**

취약하고 오래된 구성요소가 왜 이렇게 위험한지, 그리고 어떤 영향을 미칠 수 있는지 구체적으로 살펴보겠습니다.

먼저 이 취약점이 발생하는 주요 원인들을 분석해보겠습니다. 첫 번째는 기술 지원이 종료된 소프트웨어를 계속 사용하는 경우입니다. 많은 기업들이 안정성을 이유로 오래된 버전의 운영체제나 미들웨어를 계속 사용하는데, 이런 소프트웨어는 새로운 보안 패치를 받을 수 없어 취약점에 노출됩니다.

여기서 말하는 기술 지원 종료는 EOS와 EOL이라는 용어로 표현됩니다. EOS는 End of Support의 줄임말로 기술 지원이 종료됨을 의미하고, EOL은 End of Life의 줄임말로 제품의 생명주기가 완전히 끝남을 뜻합니다. 이 시점이 지나면 보안 패치나 업데이트가 더 이상 제공되지 않아 시스템이 새로운 위협에 무방비 상태로 노출됩니다.

두 번째 원인은 취약점이 존재하는 애플리케이션, 프레임워크, 라이브러리 등을 사용하는 경우입니다. 개발자들이 컴포넌트의 보안 상태를 정기적으로 점검하지 않으면 알려진 취약점이 있는 버전을 계속 사용하게 됩니다.

취약한 구성요소들은 해커들에게 시스템 침입의 문을 열어줍니다. 해커들은 이러한 약점을 이용해 데이터베이스 조작, 악성 스크립트 삽입, 서버 장악 등 다양한 공격을 시도합니다. 특히서버 장악 공격은 해커가 서버에서 원하는 명령을 실행할 수 있게 하여 전체 시스템이 완전히 통제당할 위험이 있습니다.

가장 충격적인 사례가 2021년에 발견된 Log4j 취약점입니다. Log4j는 자바 기반 프로그램에서 기록을 남기는 도구로 널리 사용되는데, 이 취약점은 잘못된 설계로 인해 해커가 특정 문자열을 시스템 기록에 남도록 조작하면, Log4j가 이를 실행하여 해커가 서버에 침투할 수 있도록 합니다.

Log4j 취약점의 여파가 큰 이유는 공개 소프트웨어 특성상 수많은 제품에 포함되어 있어서 어떤 시스템이 위험한지 파악하기 극도로 어렵다는 점이었습니다. 많은 기업들이 자산 관리 부족으로 인해 자사에서 Log4j를 사용하는 제품이 무엇인지 찾아내는 것부터 오랜 시간이 걸렸고, 이로 인해 신속한 대응이 불가능했습니다. 하나의 취약점이 전 세계적으로 얼마나 큰 피해를 가져올 수 있는지와 기업의 소프트웨어 자산관리와 의존성 관리의 중요성을 보여주는 대표적인 사례가 되었습니다.

**학습정리**

* 취약하고 오래된 구성요소는 기술 지원 종료와 취약점 미패치가 주요 원인이다.
* Log4j 취약점은 소프트웨어 자산관리와 의존성 관리의 중요성을 보여주는 심각한 취약점으로써 다양한 공격의 경로가 될 수 있다.
* 오픈소스 컴포넌트의 경우 어떤 시스템에서 사용되는지 파악하기 어려워 대응이 복잡하다.

**1-3. 국내 보안사고 사례 분석**

이제 실제로 국내에서 발생한 보안사고 사례들을 통해 취약하고 오래된 구성요소 취약점의 위험성을 구체적으로 살펴보겠습니다.

2023년 말, 국내 여러 공공기관과 금융기관 등에서 사용하던 인증 소프트웨어인 매직라인(MagicLine4NX)의 구버전에서 심각한 보안 취약점이 발견되었습니다. 해당 소프트웨어는 업데이트가 제대로 이뤄지지 않아, 공격자가 이를 통해 인증 정보를 탈취하거나 내부 시스템에 침투할 수 있는 경로로 악용되었습니다.

이 취약점은 오래된 버전이 사용자 PC에 여전히 남아 있었기 때문에 발생했습니다. 소프트웨어는 업데이트 이후에도 구버전이 자동 삭제되지 않아, 공격자에게는 여전히 유효한 침투 수단이 되었습니다. 특히 이 문제는 사용자와 기관 모두가 보안 업데이트의 실제 적용 상태를 확인하지 않은 데서 비롯되었습니다.

해당 사건 이후 정부는 모든 기관에 취약 버전 삭제 또는 최신 버전으로의 강제 업데이트를 지시했습니다. 주요 보안 업체들은 자동 삭제 도구를 배포하며 대응에 나섰고, 일부 기관에서는 내부 시스템 재구축에 착수했습니다.

이번 사고는 소프트웨어 업데이트의 중요성과 노후화된 구성요소의 위험성을 다시 한번 보여준 사례입니다. 단순히 업데이트를 배포하는 것만으로는 충분하지 않으며, 실제 설치 여부와 잔존 파일에 대한 지속적인 점검이 필요함을 시사합니다.

또 다른 중요한 사례로는 아파치 스트러츠2 취약점을 이용한 지속적인 공격들이 있습니다. Apache Struts2는 대형 기업들이 자바 웹 애플리케이션 개발에 주로 사용하는 프레임워크이기 때문에, 이러한 취약점들이 발견될 때마다 많은 국내 기업들이 영향을 받았습니다. 이는 A06 취약점의 전형적인 사례로, 널리 사용되는 오픈소스 프레임워크의 취약점이 패치되지 않은 채 방치되면서 대규모 피해로 이어지는 양상을 보여줍니다.

특히 2017년에 발견된 CVE-2017-5638 취약점은 전 세계에 충격을 주었습니다. 이 취약점은 파일 업로드 과정에서 발생하는 문제로, 해커가 악의적인 파일명을 이용해 서버에서 자신이 원하는 명령을 실행할 수 있게 하는 심각한 결함이었습니다.

이 취약점의 가장 충격적인 피해 사례는 2017년 에퀴팩스 개인정보 유출 사건입니다. 신용정보 회사인 에퀴팩스는 이 취약점을 통해 해킹당하여 1억 4천 7백만 명의 개인정보가 유출되었습니다. 주민등록번호, 생년월일, 주소, 신용카드 번호 등 민감한 정보가 대량으로 탈취되어 역사상 최악의 개인정보 유출 사건 중 하나로 기록되었습니다.

더욱 심각한 것은 이 취약점에 대한 보안 패치가 이미 공개되어 있었음에도 불구하고, 에퀴팩스가 적절한 시기에 업데이트를 하지 않아 피해가 발생했다는 점입니다. 최근에도 2023년에 심각도 9.8점의 치명적인 취약점(CVE-2023-50164)이 발견되고, 2020년에도 원격 서버 장악 취약점(CVE-2020-17530)이 발견되는 등 지속적인 보안 위협이 계속되고 있습니다.

추가적으로 2022년에는 아파치 커먼스 텍스트(Apache Commons Text)에서 발견된 초고위험도 취약점(CVE-2022-42889) 때문에 보안 업계가 긴장했습니다. 이 취약점을 공격자들이 성공적으로 익스플로잇 할 경우 아파치가 설치된 서버에 원격 접근하여 코드를 실행할 수 있게 되며, 10점 만점에 9.8점을 받았습니다.

**학습정리**

* 단순히 업데이트를 배포하는 것만으로는 충분하지 않으며, 실제 설치 여부와 잔존 파일에 대한 지속적인 점검이 필요하다.
* Apache Struts2 취약점은 대기업들을 대상으로 한 대규모 공격에 악용되었다.
* 에퀴팩스는 이 취약점을 통해 해킹당하여 1억 4천 7백만 명의 개인정보가 유출되었습니다.

**2. 취약점 대응 및 실무 적용 방안**

**학습목표**

* 취약하고 오래된 구성요소를 이용한 침투 시나리오를 이해한다
* 효과적인 예방 및 대응 방안을 수립할 수 있다
* 개발 및 운영 단계에서 적용할 수 있는 보안 고려사항을 파악한다
* 실습을 통해 취약점 탐지와 대응 방법을 익힌다

**학습내용**

**2-1. 침투 시나리오와 공격 메커니즘**

최근의 사이버 공격은 단순한 해킹을 넘어, 체계적이고 단계적인 방식으로 진행됩니다. 그중에서도 취약점이 존재하는 오래된 소프트웨어 구성요소는 공격자들에게 반복적으로 악용되는 주요 표적입니다. 이번에는 공격자들이 어떻게 이러한 취약한 구성요소를 이용해 시스템을 침투하는지, 사이버 킬체인(Cyber Kill Chain)의 각 단계를 통해 살펴보겠습니다.

공격자들은 먼저 대상 시스템에서 사용 중인 소프트웨어와 그 버전 정보를 수집합니다. 이 과정에서 보안 패치가 적용되지 않은, 취약점이 알려진 구버전 구성요소들이 주요 탐색 대상이 됩니다. 예를 들어, 웹 서버의 응답정보나 에러페이지를 통해 사용 중인 소프트웨어 버전을 파악하고, 자동화된 스캔 도구를 통해 오래되었지만 여전히 사용 중인, 취약한 라이브러리나 프레임워크를 식별합니다.

지원이 종료되었거나 수년간 보안 업데이트가 이뤄지지 않은 구성요소는 특히 위험합니다. 이미 알려진 취약점이 존재함에도 패치가 적용되지 않은 상태로 남아 있기 때문에, 공격자 입장에서는 별다른 기술적 노력 없이도 침투가 가능한 통로가 되기 때문입니다.

정보 수집 이후, 공격자들은 취약점 데이터베이스(CVE Database)를 검색하여 원격 코드 실행이나 권한 상승이 가능한 취약점을 찾고, 이미 공개된 익스플로잇(Exploit) 코드를 준비합니다. 이러한 코드들은 인터넷에 공개되어 있어 별도의 개발 없이도 즉시 사용할 수 있습니다.

이런 공격 코드는 대상 시스템의 취약한 구성요소를 직접적으로 겨냥해 침투합니다. 예를 들어, 오래된 웹 프레임워크를 사용하는 시스템에서는 악성 요청을 통해 원격 명령이 실행될 수 있고, 취약한 로깅 라이브러리를 이용해 악성 코드를 외부에서 다운로드하도록 유도할 수도 있습니다.

초기 침입에 성공한 뒤, 공격자는 시스템 내부에 남아 있는 다른 취약한 구성요소들을 연쇄적으로 악용합니다. 예컨대, 웹 애플리케이션의 취약점을 이용해 서버에 접근한 후, 운영체제의 미패치된 취약점을 통해 권한을 상승시키고, 오래된 데이터베이스 시스템을 통해 민감한 정보에 접근하는 식입니다.

이 단계에서 공격자들은 백도어(Backdoor)나 원격 접근 도구를 설치하여 시스템에 지속적으로 접근할 수 있는 환경을 구축합니다. 이때도 정상처럼 보이는 오래된 유틸리티나 시스템 도구를 활용해 보안 솔루션의 탐지를 회피할 수 있습니다.

최종적으로, 공격자는 이러한 백도어를 통해 외부 명령을 주고받거나 데이터를 유출합니다. 이때 패치되지 않은 네트워크 프로토콜이나 취약한 압축 라이브러리 등이 활용되며, 데이터 유출, 서비스 마비, 내부 시스템 장악 등의 목표가 실행됩니다.

요약하면, 취약점이 존재하고 패치되지 않은 오래된 구성요소는 단순한 보안 결함을 넘어서, 사이버 공격의 전체 단계에서 반복적으로 활용되는 위험 요소입니다. 초기 침투는 물론, 권한 상승, 내부 확산, 지속적 접근 확보, 데이터 유출 등 모든 과정에서 중요한 역할을 하게 됩니다.

현대의 애플리케이션은 수십 개의 외부 라이브러리와 프레임워크에 의존하고 있습니다. 이 중 하나라도 보안 취약점이 있는 구버전으로 남아 있다면, 전체 시스템은 공격자에게 노출될 수 있습니다. 공격자들은 이 ‘의존성 체인(Dependency Chain)’ 중 가장 약한 고리를 찾아 침투하며, 단 하나의 구성요소가 전체 인프라의 붕괴로 이어질 수 있습니다.

**학습정리**

* 오래된 구성요소는 단순한 단일 취약점이 아니라 정찰부터 목표 달성까지 사이버 킬체인 전 단계에서 연쇄적으로 악용되는 핵심 공격 통로 역할을 합니다.
* 현대 IT 환경에서 하나의 애플리케이션이 수십 개의 외부 라이브러리에 의존하는 상황에서, 단 하나의 오래된 구성요소만 있어도 전체 시스템 보안이 무너질 수 있는 약한 고리가 됩니다.
* 오래된 구성요소의 취약점은 이미 공개되어 있어 공격자들이 별도 개발 없이 기존 익스플로잇 코드를 즉시 활용할 수 있어, 패치 지연이 곧바로 보안 위험으로 직결됩니다.

**2-2. 예방 및 대응 방안**

취약하고 오래된 구성요소 취약점을 효과적으로 예방하고 대응하기 위한 종합적인 방안들을 살펴보겠습니다.

가장 기본적이면서도 중요한 것은 자산 관리입니다. 조직에서 사용하는 모든 소프트웨어 컴포넌트의 목록을 정확히 파악하고 지속적으로 관리해야 합니다. 이는 운영체제, 미들웨어, 웹 서버, 데이터베이스, 프레임워크, 라이브러리 등 모든 계층의 소프트웨어를 포함합니다. 특히 공개 소프트웨어 구성요소의 경우 서로 연결된 관계가 매우 복잡하므로, 소프트웨어 구성 분석 도구를 활용하여 정확한 구성요소 목록을 파악하는 것이 중요합니다. 이러한 구성 분석 도구를 활용해서 위험 요소를 즉시 식별하고 위험도에 따라 어떤 구성요소를 먼저 업데이트해야 할지 판단할 수 있습니다. 소프트웨어 구성 분석과 함께 IT 자산 관리 시스템을 연계하면 더욱 효과적인 보안 관리가 가능합니다. 자산 관리 시스템은 전체 IT 환경에서 사용되는 모든 소프트웨어와 하드웨어를 체계적으로 추적하므로, 취약점이 발견되었을 때 영향을 받는 시스템을 빠르게 파악할 수 있습니다. 이러한 통합 관리 시스템을 활용하면 조직 내 모든 IT 자산의 소프트웨어 구성을 한눈에 확인할 수 있고, 취약점 발견 시 영향을 받는 시스템을 즉시 식별하여 복잡한 IT 환경에서 발생할 수 있는 보안 위험을 크게 줄이고 관리 효율성을 향상시킵니다.

두 번째는 취약점 모니터링 체계 구축입니다. CVE 데이터베이스와 NVD를 정기적으로 모니터링하여 사용 중인 컴포넌트에 새로운 취약점이 발견되는지 확인해 야 합니다. 이를 위해 KISA에서 제공하는 보안 공지를 수시로 확인하고, 신규 내용이 공지될 경우 신속히 조치하는 것이 필요합니다.

세 번째는 패치 관리 프로세스 정립입니다. 취약점이 발견되면 즉시 보안 패치를 적용해야 하지만, 운영 중인 시스템의 안정성도 고려해야 합니다. 따라서 테스트 환경에서 패치의 영향도를 충분히 검증한 후 단계적으로 적용하는 체계적인 패치 관리 프로세스가 필요합니다. Log4j 사례에서 보았듯이, 하나의 취약점에 대해서도 여러 차례에 걸쳐 추가 패치가 발표될 수 있으므로 지속적인 모니터링이 중요합니다. 클라우드 환경에서는 CNAPP 솔루션을 통해 CI/CD 환경에서의 패치 문제도 효과적으로 해결할 수 있습니다. CNAPP는 Cloud Native Application Protection Platform의 줄임말로, 개발 단계부터 운영 단계까지 전체 라이프사이클에서 보안 취약점을 자동으로 탐지하고 패치를 적용할 수 있도록 지원합니다.

네 번째는 형상 관리와 불필요한 구성요소 제거입니다. 사용하지 않는 소프트웨어나 서비스, 기능, 문서 등은 과감히 제거하여 공격 표면을 최소화해야 합니다. 예를 들어, 개발 단계에서만 필요한 디버그 모드나 샘플 코드들이 운영 환경에 그대로 배포되지 않도록 주의해야 합니다.

다섯 번째는 보안 설정 강화입니다. 기본 설정은 대부분 보안성보다는 편의성에 중점을 두고 있으므로, 운영 환경에 맞는 보안 설정으로 변경해야 합니다. 이는 기본 계정과 패스워드 변경, 불필요한 서비스 비활성화, 에러 메시지 최소화 등을 포함합니다.

여섯 번째는 네트워크 수준의 보호 조치입니다. 방화벽 정책을 통해 불필요한 네트워크 접근을 차단하고, 침입 탐지 시스템인 IDS와 침입 방지 시스템인 IPS를 구축하여 공격 시도를 조기에 탐지하고 차단해야 합니다. 특히 인터넷에 직접 노출되는 시스템의 경우 더욱 엄격한 보안 조치가 필요합니다.

마지막으로 보안 모니터링과 로깅 강화가 필요합니다. SKT 사례에서 보았듯이 로그 기록을 4개월여라는 짧은 기간만 보관했기 때문에 장기간에 걸친 공격 활동을 추적하기 어려웠습니다. 따라서 충분한 기간 동안 로그를 보관하고, SIEM 시스템을 통해 실시간으로 보안 이벤트를 모니터링하는 체계를 구축해야 합니다.

이러한 종합적인 대응 방안들을 체계적으로 구현함으로써 취약하고 오래된 구성요소로 인한 보안 위험을 효과적으로 관리하고 예방할 수 있습니다.

**학습정리**

* 자산 관리를 통해 모든 소프트웨어 컴포넌트를 정확히 파악해야 한다.
* CVE/NVD 모니터링으로 새로운 취약점 정보를 신속히 획득해야 한다.
* 체계적인 패치 관리 프로세스를 통해 안정성과 보안성을 동시에 확보해야 한다.
* 공격 표면 최소화와 보안 설정 강화로 전반적인 보안 수준을 향상시켜야 한다.

**2-3. 개발 및 운영 시 고려사항**

개발 단계부터 운영 단계까지 전체 소프트웨어 생명주기에 걸쳐 취약하고 오래된 구성요소 위험을 최소화하기 위한 구체적인 고려사항들을 살펴보겠습니다.

개발 계획 단계에서는 먼저 사용할 컴포넌트들의 보안성을 충분히 검토해야 합니다. 새로운 라이브러리나 프레임워크를 도입할 때는 해당 프로젝트의 유지보수 상태, 보안 패치 이력, 커뮤니티 활성도 등을 종합적으로 평가해야 합니다. 더 이상 업데이트되지 않는 EOL 상태의 컴포넌트는 가급적 사용을 피하고, 활발히 유지보수되는 대안을 찾아야 합니다. 특히 벤더를 통해 상용 프로그램을 구매 및 도입하는 경우에도 개발 계획 단계에서 SCA로 사용할 컴포넌트를 검사해야 합니다.

설계 단계에서는 처음부터 보안을 고려한 설계 원칙을 적용해야 합니다. 구성요소 간의 분리를 통해 하나의 구성요소가 해킹당하더라도 전체 시스템에 미치는 영향을 최소화하는 구조를 설계해야 합니다. 예를 들어 웹 서버, 데이터베이스 서버, 파일 저장소를 각각 독립된 네트워크 영역에 배치하여 웹 서버가 침해되더라도 데이터베이스에 직접 접근할 수 없도록 차단하는 것입니다. 또한 컨테이너나 가상머신을 이용해 각 서비스를 독립된 환경에서 실행하거나, 마이크로서비스 구조로 기능별로 분리하여 한 서비스의 문제가 다른 서비스로 확산되지 않도록 하는 방법도 있습니다. 이와 함께, 최소 권한 원칙을 적용해 각 구성요소가 반드시 필요한 권한만 가지도록 설계하는 것도 중요합니다. 예를 들어, 웹 서버에는 데이터베이스의 모든 테이블에 대한 관리자 권한을 부여하는 대신, 필요한 특정 테이블에 대해서만 제한된 읽기 또는 쓰기 권한을 부여해야 합니다.

개발 단계에서는 SBOM 작성이 중요합니다. SBOM은 소프트웨어에 포함된 모든 컴포넌트의 목록과 버전 정보를 상세히 기록한 문서로, 향후 취약점 발견 시 영향 범위를 신속히 파악하는 데 필수적입니다. 또한 SCA 도구를 CI/CD 파이프라인에 통합하여 빌드 과정에서 자동으로 취약점을 탐지하고 차단하는 체계를 구축해야 합니다. 최근 생성형 AI를 활용한 개발이 증가하고 있는데, 이때 AI가 생성한 코드에서 오래된 컴포넌트나 함수를 사용하지 않도록 소스코드 검토를 반드시 수행해야 합니다. 내부적으로 코드 리팩토링이나 코드 최적화를 진행할 때도 취약한 구성요소를 사용하지 않는 부분을 중요한 지표로써 검토해야 합니다.

코딩 단계에서는 입력 검증과 출력 인코딩을 철저히 수행해야 합니다. Log4j 취약점처럼 사용자 입력이 예상치 못한 방식으로 처리되어 보안 문제가 발생할 수 있으므로, 모든 외부 입력에 대해 화이트리스트 기반의 엄격한 검증을 수행해야 합니다. 또한 컴포넌트에서 제공하는 보안 설정 옵션들을 충분히 활용해야 합니다. 외주 개발을 진행하는 경우에는 외주 회사에 대해서도 취약한 구성요소를 사용하지 않아야 함을 명확히 교육하고 계약서에 이를 명시해야 합니다.

테스트 단계에서는 보안 테스트를 반드시 포함해야 합니다. SAST와 DAST 도구를 활용하여 알려진 취약점 패턴을 탐지하고, 침투 테스트를 통해 실제 공격 시나리오에 대한 방어 능력을 검증해야 합니다. 특히 사용 중인 컴포넌트의 최신 취약점 정보를 반영한 테스트 시나리오를 구성하는 것이 중요합니다.

배포 단계에서는 우선 사용하는 모든 구성요소의 보안 상태를 확인해야 합니다. 운영체제의 보안 패치가 최신 상태인지 점검하고, 사용하지 않는 불필요한 서비스나 프로그램들을 제거하여 해커들이 침입할 수 있는 경로를 최소화해야 합니다. 네트워크 설정에서도 필요하지 않은 포트는 모두 차단하고, 방화벽 규칙을 통해 허용된 접근만 가능하도록 설정해야 합니다. 또한, 기본 계정의 비밀번호를 변경하고, 관리자 계정의 접근을 제한하며, 로그 모니터링 시스템을 구축하여 이상 행동을 빠르게 탐지할 수 있도록 해야 합니다.

운영 단계에서는 지속적인 모니터링과 관리가 핵심입니다. 실시간 취약점 스캐닝을 통해 새로운 취약점이 발견되는 즉시 탐지하고, 자동화된 패치 관리 시스템을 통해 신속하게 대응해야 합니다. 또한 WAF나 RASP 같은 런타임 보안 솔루션을 활용하여 알려진 공격 패턴을 실시간으로 차단하는 것도 효과적입니다. 이러한 소프트웨어 생명주기 전반에 걸친 체계적인 접근을 통해 취약하고 오래된 구성요소로 인한 보안 위험을 효과적으로 관리하고 예방할 수 있습니다.

**학습정리**

* 설계 단계에서는 처음부터 보안을 고려한 설계 원칙을 적용해야 한다
* 개발 계획 단계부터 컴포넌트의 보안성과 유지보수 상태를 검토해야 한다.
* 운영 단계에서는 지속적인 보안 관제(모니터링)와 취약점 관리를 통해 서비스의 안정성과 보안을 유지해야 한다.
* 소프트웨어 생명주기 전반에 걸친 체계적인 접근을 통해 취약하고 오래된 구성요소로 인한 보안 위험을 효과적으로 관리하고 예방할 수 있다.

이상으로 OWASP A06 취약하고 오래된 구성요소 취약점에 대한 교육을 마무리하겠습니다. 오늘 학습한 내용들을 바탕으로 여러분의 업무 환경에서 보다 안전한 시스템을 구축하고 업데이트하시길 바랍니다.