애플리케이션 보안 검증 표준

Version 5.0.0

May 2025

Contents

서문

표준 소개

애플리케이션 보안 검증 표준 (Application Security Verification Standard; ASVS) 은 아키텍트, 개발자, 테스터, 보안전문가, tool 공급업체, 소비자가 안전한 애플리케이션을 정의, 구축, 테스트 및 검증하는데 사용할 수 있는 애플리케이션 보안 요구 사항 목록이다.

저작권 및 라이선스

Version 5.0.0, May 2025

Figure 1: license

프로젝트 리드

Elar Lang	Josh C Grossman
Jim Manico	Daniel Cuthbert

작업 그룹

Tobias Ahnoff	Ralph Andalis	Ryan Armstrong	Gabriel Corona
Meghan Jacquot	Shanni Prutchi	Iman Sharafaldin	Eden Yardeni

주요 기여자

Sjoerd Langkemper	Isaac Lewis
Mark Carney	Sandro Gauci

기타 기여자 및 리뷰어

기타 기여자 목록은 부록 E 에 포함되어 있다.

5.x 버전의 크레딧 목록에 누락된 경우, GitHub 에 티켓을 등록하면 향후 5.x 업데이트에서 반영될 수 있다.

애플리케이션 보안 검증 표준은 ASVS 1.0(2008) 부터 4.0(2019) 에 이르기까지 참여한 이들의 작업을 기반으로 한다. 현재 ASVS 에 남아 있는 많은 구조와 검증 항목은 Andrew van der Stock, Mike Boberski, Jeff Williams, Dave Wichers 등 여러 기여자에 의해 최초 작성되었다. 과거에 기여한 모든 분들께 감사의 뜻을 전한다. 이전 기여자에 대한 전체 목록은 각 버전의 문서를 참조한다.

머리말

애플리케이션 보안 검증 표준 (ASVS) 5.0 버전에 오신 것을 환영한다.

소개

2008 년에 글로벌 커뮤니티 협력을 통해 처음 시작된 ASVS 는 최신 웹 애플리케이션과 서비스를 설계, 개발 및 테스트하기 위한 포괄적인 보안 요구 사항을 정의한다.

2019 년 ASVS 4.0 과 이은 2021 년의 작은 업데이트 (v4.0.3) 의 출시에 이어서, 5.0 버전은 소프트웨어 보안의 최신 발전을 반영한 현대의 중요한 이정표와 같다.

ASVS 5.0 은 프로젝트 리더, 작업 그룹 구성원, 그리고 OWASP 전반의 커뮤니티가 광범위하게 업데이트하고 개선하기 위해 기여한 결과다.

5.0 버전의 원칙

이 주요 개정판은 다음과 같은 몇 가지 주요 원칙을 염두에 두고 개발되었다:

- 개선된 범위와 초점: 이 표준 버전은 애플리케이션, 보안, 검증 및 표준이라는 이름의 기본 토대를 보다 충실히 반영하도록 설계했다. 특정 기술 구현을 의무화하는 대신 보안 결함 예방을 강조하기 위해 요구 사항을 다시 작성했다. 요구 사항 내용에서는 왜 그런 요구 사항이 존재하는지 설명한다.
- 보안 결정의 문서화: ASVS 5.0 은 주요 보안 결정을 문서화하기 위한 요구 사항을 도입했다. 이를 통해 추적 가능성이 향상되고 상황에 맞춘 구현을 지원하여 조직이 특정 요구 사항과 위험 레벨에 맞게 보안 상태를 조정할 수 있도록 한다.
- 개정된 레벨: ASVS 는 3 단계 모델을 유지하지만, 레벨의 정의는 ASVS 를 더 쉽게 채택할 수 있도록 개정했다. 레벨 1 은 ASVS 를 채택하기 위한 초기 단계로 설계되어 첫 번째 방어 계층을 제공한다. 레벨 2 는 표준 보안 관행에 대한 포괄적인 관점을 나타내며, 레벨 3 은 높은 신뢰성을 보장하는 수준의 요구 사항을 다룬다.
- 재구성 및 확장된 콘텐츠: ASVS 5.0 에는 17 개의 챕터에 걸쳐 약 350 개의 요구 사항이 포함되어 있다. 챕터는 명확성과 사용성을 위해 재구성했다. 마이그레이션을 용이하게 하기 위해 v4.0 과 v5.0 간의 양방향 매핑이 제공된다.

앞으로의 방향

애플리케이션 보안에는 끝이 없듯이, ASVS 도 마찬가지다. 버전 5.0 은 주요 배포판이지만 개발은 계속되고 있다. 이번 배포판을 통해 더 폭넓은 커뮤니티가 축적된 개선 사항과 추가 사항의 혜택을 누릴 수 있을 뿐만 아니라 향후 개선을 위한 토대를 마련할 수 있다. 더욱이 핵심 요구 사항들을 기반으로 구현 및 검증 지침을 만들기 위한 커뮤니티 주도의 노력도 필요할 것이다.

ASVS 5.0 은 안전한 소프트웨어 개발을 위한 신뢰할 수 있는 기반으로 설계되었다. 커뮤니티는 애플리케이션 보안 상태를 종합적으로 발전시키기 위해 이 표준을 채택, 기여 및 구축할 것을 권장한다.

ASVS 란 무엇인가?

애플리케이션 보안 검증 표준 (Application Security Verification Standard; ASVS) 은 웹 애플리케이션 및 서비스의 보안 요구사항을 정의하며, 안전한 애플리케이션을 설계, 개발, 유지보수하거나 보안을 평가하는 모든 사람에게 유용한 자료이다.

이 장은 ASVS 활용의 핵심 요소를 다루며, 여기에는 범위, 우선순위 기반 레벨 구조, 표준의 주요 사용 사례가 포함된다.

ASVS 의 범위

ASVS 의 범위는 명칭인 애플리케이션, 보안, 검증, 표준에 의해 정의된다. 이는 궁극적으로 달성해야 하는 보안 원칙을 식별하는 목표를 가지고 포함되거나 제외되는 요구사항을 설정한다. 또한 이 범위는 구현 요구사항의 토대 역할을 하는 문서화 요구사항을 고려한다.

공격자에게는 범위라는 개념이 존재하지 않는다. 따라서 ASVS 요구사항은 CI/CD 프로세스, 호스팅, 운영 활동을 포함한 애플리케이션 수명 주기의 다른 측면에 대한 지침과 함께 평가되어야 한다.

애플리케이션

ASVS 는“애플리케이션”을 보안 제어 기능이 통합되어야 하는 개발 중인 소프트웨어 제품으로 정의한다. ASVS 는 개발 수명 주기 활동을 규정하거나 CI/CD 파이프라인을 통해 애플리케이션이 구축되는 방식을 지시하지 않는다. 대신 제품이 달성해야 하는 보안 목표를 명시한다.

웹 애플리케이션 방화벽 (Web Application Firewall; WAF), 로드 밸런서 또는 프록시와 같이 HTTP 트래픽을 처리, 수정 또는 검검증하는 구성 요소는 일부 보안 제어 기능이 해당 요소에 직접 의존하거나 해당 요소를 통해 구현될 수 있으므로 특정 목적을 위해 애플리케이션의 일부로 간주될 수 있다. 이러한 구성 요소는 캐시된 응답, 속도 제한 또는 원본과 대상에 기반한 인바운드 (inbound) 및 아웃바운드 (outbound) 연결 제한과 관련된 요구사항에 대해 고려되어야 한다.

반대로 ASVS 는 애플리케이션과 직접 관련이 없거나 구성이 애플리케이션의 책임 범위를 벗어나는 요구사항을 일반적으로 제외한다. 예를 들어, DNS 문제는 일반적으로 별도의 팀 또는 기능에 의해 관리된다.

마찬가지로, 애플리케이션이 입력을 소비하고 출력을 생성하는 방식에 대한 책임이 있지만, 외부 프로세스가 애플리케이션 또는 해당 데이터와 상호작용을 하는 경우 ASVS 의 범위를 벗어나는 것으로 간주한다. 예를 들어, 애플리케이션 또는 해당 데이터 백업은 일반적으로 외부 프로세스의 책임이며 애플리케이션 또는 개발자가 제어하지 않는다.

보안

모든 요구사항은 보안에 실질적인 영향을 미쳐야 하며, 그 효과가 입증 가능해야 한다. 요구사항이 누락되면 애플리케이션의 보안 수준이 저하되며, 요구사항을 구현하면 보안 위험의 발생 가능성 또는 영향을 감소시켜야 한다.

기능적 측면, 코드 스타일 또는 정책 요구사항과 같은 다른 모든 고려 사항은 범위에 포함되지 않는다.

검증

해당 요구사항은 검증 가능해야 하며, 검증 결과는“실패”또는“통과”여야 한다.

표준

ASVS 는 표준을 준수하기 위해 구현되어야 하는 보안 요구사항의 집합으로 설계되었다. 이는 요구사항이 해당 보안 목표를 달성하기 위한 보안 목표 정의에 제한된다는 의미이다. 기타 관련 정보는 ASVS 를 기반으로 확장하거나, 매핑 문서를 통해 연결할 수 있다.

특히 OWASP 는 많은 프로젝트를 가지고 있으며, ASVS 는 다른 프로젝트의 내용과 중복되는 것을 계획적으로 피한다. 예를 들어, 개발자는“특정 기술 또는 환경에서 특정 요구사항을 어떻게 구현해야 하는가”라는 질문을 할 수 있으며, 이는 Cheat Sheet Series 프로젝트에서 다루어져야 한다. 검증자는“이 환경에서 이 요구사항을 어떻게 테스트해야 하는가”라는 질문을 할 수 있으며, 이는 Web Security Testing Guide 프로젝트에서 다룬다.

ASVS 는 보안 전문가만이 사용하도록 의도된 것은 아니지만, 독자가 내용을 이해하거나 특정 개념을 조사하기 위해 필요한 기술적 지식을 갖추고 있음을 전제한다.

요구사항

“요구사항”이라는 단어는 ASVS 에서 이를 충족하기 위해 무엇을 달성해야 하는지를 설명하므로 특별히 사용된다. ASVS 는 주요 조건으로서 요구사항 (must) 만 포함하며 권고 사항 (should) 은 포함하지 않는다.

다시 말해, 권고 사항은 문제를 해결하기 위한 여러 가능한 옵션 중 하나이거나 코드 스타일 고려사항일 뿐 요구사항을 충족하지 않는다.

ASVS 요구사항은 구현이나 기술에 너무 구애받지 않으면서도 존재하는 이유를 자명하게 설명하면서 특정 보안 원칙을 다루도록 의도되었다. 이는 또한 요구사항이 특정 검증 방법이나 구현을 중심으로 구축되지 않음을 의미한다.

문서화된 보안 결정

소프트웨어 보안에서 보안 설계 및 사용할 메커니즘을 조기에 계획하면 완성된 제품 또는 기능에서 더 일관되고 신뢰할 수 있는 구현으로 이어진다.

또한 특정 요구사항의 경우, 구현이 복잡하고 애플리케이션의 필요에 매우 특화될 수 있다. 일반적인 예시로는 권한, 입력 유효성 검사, 다양한 수준의 민감한 데이터에 대한 보호 제어 기능이 포함된다.

이를 고려하여“모든 데이터는 암호화되어야 한다”와 같은 포괄적인 진술이나 요구사항에서 모든 가능한 사용 사례를 아우르려고 하기보다는, 애플리케이션 개발자가 이러한 제어 기능을 어떻게 접근하고 구성하는지를 문서화하도록 요구하는 것이 포함되었다. 이렇게 문서화된 내용을 기반으로 적절성을 검토할 수 있으며, 실제 구현이 기대치와 일치하는지도 비교하여 평가할 수 있다.

이러한 요구사항은 애플리케이션을 개발하는 조직이 특정 보안 요구사항을 구현하는 방법에 대해 내린 결정을 문서화하기 위한 것이다.

문서화된 요구사항은 항상 장의 첫 번째 절에 위치하며 모든 장에 포함되는 것은 아니다. 또한 해당 내용이 실제로 구현되어야 한다는 관련 구현 요건이 항상 존재한다. 문서가 존재하는지를 검증하는 활동과, 그것이 실제로 구현되었는지를 검증하는 활동은 별개의 작업이다.

이러한 요구사항을 포함하는 데에는 두 가지 주요 동인이 있다. 첫 번째 동인은 보안 요구사항이 종종 파일 유형 허용 범위, 적용해야 할 비즈니스 제어 기능, 특정 필드에 허용되는 문자 등 규칙 적용을 수반한다는 것이다. 이러한 규칙은 애플리케이션마다 다르므로 ASVS 는 이를 규정적으로 정의할 수 없으며, 치트 시트 또는 더 자세한 응답도 이 경우에 도움이 되지 않는다. 마찬가지로, 이러한 결정이 문서화되지 않으면 이러한 결정을 구현하는 요구사항에 대한 검증을 수행할 수 없다.

두 번째 동인은 특정 요구사항의 경우 특정 보안 과제를 해결하는 방법에 대해 애플리케이션 개발에 유연성을 제공하는 것이 중요하다는 것이다. 예를 들어, 이전 ASVS 버전에서는 세션 타임아웃 규칙이 매우 규정적이었다. 실용적으로 볼 때, 특히 소비자 대상 애플리케이션의 많은 경우 훨씬 더 완화된 규칙을 가지며 대신 다른 완화 제어 기능을 구현하는 것을 선호한다. 따라서 문서화 요구사항은 이에 대한 유연성을 명시적으로 허용한다.

개별 개발자가 이러한 결정을 내리고 문서화할 것으로 기대되는 것은 아니다. 이러한 결정은 조직 전체가 내리고, 이를 개발자에게 전달하여 따르도록 하는 것이 바람직하다.

개발자에게 새로운 기능에 대한 사양과 설계를 제공하는 것은 소프트웨어 개발의 표준적인 부분이다. 마찬가지로, 개발자는 매번 주관적으로 결정하기보다는 공통 컴포넌트와 사용자 인터페이스 메커니즘을 사용하는 것이 기대된다. 따라서 이를 보안에 확장하는 것은 놀랍거나 논란의 여지가 있는 것으로 간주되지 않아야 한다.

이를 달성하는 방법에도 유연성이 있다. 보안 결정은 개발자가 참조해야 하는 문서 형태로 문서화될 수 있다. 또는 모든 개발자가 의무적으로 사용해야 하는 공통 코드 라이브러리에 보안 결정이 문서화되고 구현될 수 있다. 두 경우 모두 원하는 결과가 달성된다.

애플리케이션 보안 검증 레벨

ASVS 는 깊이와 복잡성이 증가하는 세 가지 보안 검증 레벨을 정의한다. 일반적인 목표는 조직이 가장 중요한 보안 문제를 해결하기 위해 첫 번째 레벨부터 시작한 다음, 조직 및 애플리케이션 요구사항에 따라 더 높은 레벨로 진행하는 것이다. 레벨은 문서와 요구사항 텍스트에서 L1, L2, L3 로 표시될 수 있다.

각 ASVS 레벨은 해당 레벨에서 달성해야 하는 보안 요구사항을 나타내며, 남은 더 높은 레벨의 요구사항은 권고 사항으로 간주된다.

중복되는 요구사항이나 더 높은 레벨에서 더 이상 관련 없는 요구사항을 피하기 위해 일부 요구사항은 특정 레벨에 적용되지만, 더 높은 레벨에서는 더 엄격한 조건을 가진다.

레벨 평가

레벨은 보안 요구사항 구현 및 테스트 경험을 바탕으로 각 요구사항을 우선순위에 따라 평가하여 정의된다. 주요 초점은 위험 감소와 요구사항 구현 노력 간의 비교에 있다. 또 다른 주요 요인은 진입 장벽을 낮게 유지하는 것이다.

위험 감소는 요구사항이 애플리케이션 내 보안 위험 수준을 얼마나 줄이는지 고려하며, 고전적인 기밀성, 무결성, 가용성 영향 요인을 고려하고 이것이 주요 방어 계층인지 또는 심층 방어로 간주될 수 있는지를 고려한다.

기준과 레벨링 결정에 대한 엄격한 논의 결과는 대부분의 경우에 유효해야 하는 할당이 도출되었지만, 모든 상황에 100% 적합하지 않을 수 있음을 인정한다. 이는 특정 경우에 조직이 자체적인 특정 위험 고려 사항에 따라 더 높은 레벨의 요구사항을 더 일찍 우선순위로 지정할 수 있음을 의미한다.

각 레벨의 요구사항 유형은 다음과 같이 특징지어질 수 있다.

레벨 1

이 레벨은 애플리케이션을 보호할 때 고려해야 할 최소 요구사항을 포함하며, 중요한 시작점을 나타낸다. 이 레벨은 ASVS 요구사항의 약 20% 를 포함한다. 이 레벨의 목표는 가능한 한 적은 요구사항을 포함하여 진입 장벽을 낮추는 것이다.

이러한 요구사항은 일반적으로 다른 취약점이나 사전 조건이 필요 없이 악용될 수 있는 일반적인 공격을 방지하기 위한 중요하거나 기본적인 첫 번째 방어 계층 요구사항이다.

첫 번째 방어 계층 요구사항 외에도, 비밀번호 관련 요구사항과 같이 더 높은 레벨에서는 영향이 적은 일부 요구사항이 있다. 이러한 요구사항은 레벨 1 에서 더 중요하며, 더 높은 레벨부터는 다중 요소 인증 요구사항이 관련성을 가진다.

레벨 1 은 내부 문서나 코드 접근 권한 없이 (예: “블랙 박스 (black box)”테스트) 외부 테스터가 침투 테스트를 수행할 수 있는 것은 아니지만, 요구사항 수가 적으므로 검증이 더 쉬워야 한다.

레벨 2

이러한 요구사항은 일반적으로 덜 흔한 공격 또는 일반적인 공격에 대한 더 복잡한 보호 기능과 관련된다. 이는 여전히 첫 번째 방어 계층일 수 있으며, 공격이 성공하기 위한 특정 사전 조건이 필요할 수 있다.

이러한 요구사항은 일반적으로 덜 흔한 공격 또는 일반적인 공격에 대한 더 복잡한 보호 기능과 관련된다. 이는 여전히 첫 번째 방어 계층일 수 있으며, 공격이 성공하기 위한 특정 사전 조건이 필요할 수 있다.

레벨 3

이 레벨은 최고 수준의 보안을 입증하고자 하는 애플리케이션의 목표가 되어야 하며, 준수해야 하는 나머지 약 30% 의 요구사항을 제공한다.

이 섹션의 요구사항은 일반적으로 심층 방어 메커니즘 또는 기타 유용하지만 구현하기 어려운 제어 기능이다.

달성해야 할 레벨

우선순위 기반 레벨은 조직 및 애플리케이션의 애플리케이션 보안 성숙도를 반영하는 역할을 한다. ASVS 가 애플리케이션이 어떤 레벨에 있어야 한다고 규정적으로 명시하기보다는, 조직은 애플리케이션의 민감도와 애플리케이션 사용자들의 기대치를 고려하여 자체 위험을 분석하고 어떤 레벨에 있어야 한다고 판단하는지 결정해야 한다.

예를 들어, 제한된 민감 데이터를 수집하는 초기 스타트업은 초기 보안 목표로 레벨 1 에 집중하기로 결정할 수 있지만, 은행은 온라인 뱅킹 애플리케이션에 대해 고객에게 레벨 3 미만의 수준을 정당화하기 어려울 수 있다.

ASVS 사용 방법

ASVS 의 구조

ASVS 는 총 약 350 개의 요구사항으로 구성되며, 이는 17 개 장으로 나뉘고 각 장은 다시 섹션으로 세분된다.

장 및 섹션 분할의 목표는 애플리케이션에 적합한 것을 기반으로 장 및 섹션을 선택하거나 필터링하는 것을 단순화하는 것이다. 예를 들어, 머신 투 머신 (machine-to-machine) API 의 경우 웹 프론트엔드와 관련된 V3 장의 요구사항은 관련성이 없을 것이다. OAuth 또는 WebRTC 를 사용하지 않는다면 해당 장도 무시될 수 있다.

릴리스 전략

ASVS 릴리스는“Major.Minor.Patch”패턴을 따르며, 숫자는 해당 릴리스 내에서 변경된 내용에 대한 정보를 제공한다. 주요 릴리스에서는 첫 번째 숫자가 변경되고, 마이너 릴리스에서는 두 번째 숫자가 변경되며, 패치 릴리스에서는 세 번째 숫자가 변경된다.

- 주요 릴리스 - 전체 재편성, 요구사항 번호를 포함하여 거의 모든 것이 변경될 수 있다. 규정 준수 재평가가 필요하다 (예: 4.0.3 -> 5.0.0).
- 마이너 릴리스 - 요구사항이 추가되거나 제거될 수 있지만, 전체 번호는 동일하게 유지된다. 규정 준수 재평가가 필요하지만, 더 쉬워야 한다 (예: 5.0.0 -> 5.1.0).
- 패치 릴리스 - 요구사항이 제거되거나 (예: 중복되거나 오래된 경우) 덜 엄격하게 변경될 수 있지만, 이전 릴리스를 준수했던 애플리케이션은 패치 릴리스 또한 준수할 것이다 (예: 5.0.0 -> 5.0.1).

위 내용은 ASVS 의 요구사항에 특별히 관련된다. 주변 텍스트 및 부록과 같은 다른 콘텐츠의 변경은 주요 변경 (breaking change) 으로 간주되지 않는다.

ASVS 의 유연성

문서화 요구사항 및 레벨 메커니즘과 같이 위에서 설명된 여러 사항은 ASVS 를 더 유연하고 조직 특유의 방식으로 사용할 수 있는 기능을 제공한다.

또한 조직은 애플리케이션의 특정 특성 및 위험 수준에 따라 요구사항을 조정하는 조직 또는 도메인별 포크 (fork) 를 생성하는 것이 강력히 권장된다. 그러나 요구사항 4.1.1 을 통과하는 것이 모든 버전에서 동일한 의미를 갖도록 추적성을 유지하는 것이 중요하다.

이상적으로는 각 조직이 관련 없는 섹션 (예: GraphQL, WebSockets, SOAP, 사용하지 않는 경우) 을 생략하여 자체 맞춤형 ASVS 를 생성해야 한다. 조직별 ASVS 버전 또는 보충 자료는 요구사항 준수 시 사용할 라이브러리 또는 리소스를 자세히 설명하는 조직별 구현 지침을 제공하기에도 좋은 장소이다.

ASVS 요구사항 참조 방법

각 요구사항은.

. 형식의 식별자를 가지며, 각 요소는 숫자이다. 예를 들어, 1.11.3.

- 값은 요구사항이 속한 장에 해당한다. 예를 들어, 모든 1.#.# 요구사항은‘인코딩 및 새니티제이션 (Encoding and Sanitization)’장에 속한다.
- 값은 해당 장 내에서 요구사항이 나타나는 섹션에 해당한다. 예를 들어: 모든 1.2.# 요구사항은‘인코딩 및 새니티제이션’장의‘인젝션 방지 (Injection Prevention)’섹션에 속한다.
- 값은 장 및 섹션 내의 특정 요구사항을 식별한다. 예를 들어, 이 표준의 버전 5.0.0 을 기준으로 1.2.5 는 다음과 같다:

> 애플리케이션이 OS 명령 주입 (OS command injection) 으로부터 보호하고 운영 체제 호출이 매개변수화된 OS 쿼리를 사용하거나 문맥별 명령줄 출력 인코딩을 사용하는지 검증한다.

식별자는 표준 버전 간에 변경될 수 있으므로, 다른 문서, 보고서 또는 도구에서 다음 형식을 사용하는 것이 바람직하다: ’ v-.

.’여기서‘version’은 ASVS 버전 태그이다. 예를 들어: ‘v5.0.0-1.2.5’는 버전 5.0.0 의’인코딩 및 새니티제이션’장의‘인젝션 방지’섹션에 있는 특정 5 번째 요구사항을 의미하는 것으로 이해될 것이다. (이는 v-로 요약될 수 있다.)

참고: 형식의 버전 번호 앞에 오는’v’는 항상 소문자여야 한다.

식별자가‘v’요소를 포함하지 않고 사용되는 경우, 이는 최신 애플리케이션 보안 검증 표준 콘텐츠를 참조하는 것으로 가정해야 한다. 표준이 성장하고 변경됨에 따라 이는 문제가 되므로, 작성자 또는 개발자는 버전 요소를 포함해야 한다.

ASVS 요구사항 목록은 CSV, JSON 및 기타 형식으로 제공되며, 이는 참조 또는 프로그래밍적 사용에 유용할 수 있다.

ASVS 포크

조직은 세 가지 레벨 중 하나를 선택하거나 애플리케이션 위험 수준에 따라 요구사항을 조정하는 도메인별 포크를 생성하여 ASVS 채택으로부터 이점을 얻을 수 있다. 이러한 유형의 포크는 요구사항 4.1.1 을 통과하는 것이 모든 버전에서 동일한 의미를 갖도록 추적성을 유지하는 경우 권장된다.

이상적으로는 각 조직이 관련 없는 섹션 (예: GraphQL, WebSockets, SOAP, 사용하지 않는 경우) 을 생략하여 자체 맞춤형 ASVS 를 생성해야 한다. 포크는 ASVS 레벨 1 을 기준으로 시작하여 애플리케이션의 위험에 따라 레벨 2 또는 3 으로 진행해야 한다.

ASVS 의 사용 사례

ASVS 는 애플리케이션의 보안을 평가하는 데 사용될 수 있으며, 이는 다음 장에서 더 자세히 다루어진다. 그러나 ASVS(또는 포크된 버전) 의 다른 잠재적 사용 사례가 여러 가지 확인되었다.

상세 보안 아키텍처 지침으로서

애플리케이션 보안 검증 표준의 더 일반적인 용도 중 하나는 보안 아키텍트 (security architect) 를 위한 자료로 활용되는 것이다. 특히 최신 애플리케이션에서 보안 애플리케이션 아키텍처를 구축하는 방법에 대한 자료는 제한적이다. ASVS 는 데이터 보호 패턴 및 입력 유효성 검사 전략과 같은 일반적인 문제에 대해 보안 아키텍트가 더 나은 제어 기능을 선택할 수 있도록 하여 이러한 격차를 채우는 데 사용될 수 있다. 아키텍처 및 문서화 요구사항은 특히 이 용도에 유용할 것이다.

전문 보안 코딩 참조 자료로서

ASVS 는 애플리케이션 개발 중 보안 코딩 참조 자료를 준비하기 위한 기반으로 사용될 수 있으며, 개발자가 소프트웨어를 구축할 때 보안을 염두에 두도록 돕는다. ASVS 가 기반이 될 수 있지만, 조직은 명확하고 통일된 자체적인 특정 지침을 준비하는 것이 권장되며, 이상적으로는 보안 엔지니어 또는 보안 아키텍트의 지침을 기반으로 준비되어야 한다. 이의 연장선상에서, 조직은 가능한 한 지침에 참조되고 개발자가 사용할 수 있는 승인된 보안 메커니즘 및 라이브러리를 준비하는 것이 권장된다.

자동화된 단위 및 통합 테스트를 위한 가이드로서

ASVS 는 높은 테스트 가능성을 가지도록 설계되었다. 일부 검증은 기술적일 수 있지만, 다른 요구사항 (아키텍처 및 문서화 요구사항 등) 은 문서 또는 아키텍처 검토를 필요로 할 수 있다. 기술적 수단으로 검증 가능한 요구사항과 관련된 특정 및 관련 악용 사례에 대해 테스트하고 퍼징 (fuzzing) 하는 단위 (unit) 및 통합 (integration) 테스트를 구축함으로써, 각 빌드에서 이러한 제어 기능이 올바르게 작동하는지 확인하기가 더 쉬워야 한다. 예를 들어, 로그인 컨트롤러의 테스트 스위트 (test suite) 에 대해 추가 테스트를 작성하여 일반적인 기본 사용자 이름, 계정 열거 (account enumeration), 무차별 대입 (brute forcing), LDAP 및 SQL 주입 (injection), XSS 에 대한 사용자 이름 매개변수를 테스트할 수 있다. 마찬가지로, 비밀번호 매개변수에 대한 테스트에는 일반적인 비밀번호, 비밀번호 길이, 널 바이트 (null byte) 주입, 매개변수 제거, XSS 등이 포함되어야 한다.

보안 개발 교육을 위해

ASVS 는 보안 소프트웨어의 특성을 정의하는 데도 사용될 수 있다. 많은“보안 코딩”과정은 단순히 코딩 팁이 약간 섞인 윤리적 해킹 과정에 불과하다. 이는 개발자가 더 안전한 코드를 작성하는 데 반드시 도움이 되지 않을 수 있다. 대신, 보안 개발 과정은 하지 말아야 할“Top 10”과 같은 부정적인 사항보다는 ASVS 에서 발견되는 긍정적인 메커니즘에 강력히 초점을 맞춰 ASVS 를 사용할 수 있다. ASVS 구조는 애플리케이션을 보호할 때 다양한 주제를 다루는 논리적인 구조 또한 제공한다.

보안 소프트웨어 조달을 위한 프레임워크로서

ASVS 는 보안 소프트웨어 조달 또는 맞춤형 개발 서비스 조달에 도움이 되는 훌륭한 프레임워크이다. 구매자는 조달하고자 하는 소프트웨어가 ASVS 레벨 X 로 개발되어야 한다는 요구사항을 설정하고, 판매자에게 해당 소프트웨어가 ASVS 레벨 X 를 충족함을 증명하도록 요청할 수 있다.

ASVS 실전 적용

다양한 위협은 각기 다른 동기를 가진다. 일부 산업은 고유한 정보 및 기술 자산과 도메인별 규제 준수 요구사항을 가진다.

조직은 비즈니스 특성을 기반으로 고유한 위험 특성을 깊이 살펴보고, 해당 위험과 비즈니스 요구사항을 기반으로 적절한 ASVS 레벨을 결정하는 것이 강력히 권장된다.

평가와 인증

OWASP 의 ASVS 인증과 신뢰 마크에 대한 입장

OWASP 는 특정 공급업체나 검증자, 소프트웨어를 인증하지 않는 비영리 단체이다. 따라서 ASVS 준수를 주장하는 보증, 신뢰 마크, 인증은 OWASP 가 공식적으로 승인한 것이 아니며, 조직은 제 3 자가 주장하는 ASVS 인증을 신중히 검토해야 한다.

조직은 공식 OWASP 인증을 주장하지 않는 한, 보증 서비스를 제공할 수 있다.

ASVS 준수 여부를 검증하는 방법

ASVS 는 테스트 가이드 수준에서 준수 여부를 정확히 검증하는 방법에 대해 구체적인 지침을 제공하지 않는다. 그러나 몇몇의 핵심 사항은 강조할 필요가 있다.

검증 보고

전통적인 침투 테스트 보고서는 일반적으로 실패한 항목만을 나열하는“예외 보고 방식”으로 작성된다. 그러나 ASVS 인증 보고서에는 점검 범위, 점검한 모든 요구사항에 대한 요약, 예외가 발생한 요구사항, 문제 해결 가이드를 포함해야 한다. 일부 요구사항은 적용되지 않을 수 있으며 (예: 무상태 API(stateless API) 에서의 세션 관리), 이러한 사항은 보고서에 반드시 명시해야 한다.

검증 범위

애플리케이션을 개발하는 조직은 일반적으로 모든 요구사항을 구현하지 않으며, 이는 애플리케이션의 기능에 따라 일부 요구사항이 관련 없거나 중요도가 낮을 수 있기 때문이다. 검증자는 조직이 달성하려는 레벨과 포함된 요구사항을 명확히 하여 검증 범위를 분명히 해야 한다. 이때 검증 범위는’포함하지 않은 항목’이 아닌, ’포함한 항목’을 중심으로 기술하는 것이 권장된다. 또한 구현되지 않은 요구사항을 제외한 근거에 대한 의견도 함께 제시해야 한다.

이는 검증 보고서의 수신자가 검증의 맥락을 이해하고, 애플리케이션에 어느 정도의 신뢰를 부여할 수 있을지에 대해 정보에 기반한 결정을 내릴 수 있도록 한다.

인증 기관은 검증 방법을 자율적으로 선택할 수 있지만, 해당 방법은 보고서에 공개해야 하며 이상적으로는 반복 가능하도록 해야한. 입력 검증과 같은 항목을 확인하기 위해, 애플리케이션과 요구사항에 따라 수동 침투 테스트나 소스 코드 분석과 같은 다양한 방법이 사용될 수 있다.

검증 메커니즘

특정 ASVS 요구사항을 검증하기 위해서는 다양한 기술이 필요할 수 있다. 유효한 자격 증명을 사용하여 애플리케이션 전체를 점검하는 침투 테스트 외에도, 문서, 소스 코드, 설정 정보, 개발 과정에 참여한 인력에 대한 접근이 요구될 수 있으며, 이는 특히 L2 및 L3 요구사항 검증에서 중요하다. 일반적으로는 작업 문서, 스크린샷, 스크립트, 테스트 로그 등을 포함한 상세한 문서를 통해 검증 결과에 대한 충분한 증거를 제공하는 것이 표준이다. 단순히 자동화 도구를 실행하는 것만으로는 충분하지 않으며, 각 요구사항은 반드시 검증 가능하게 테스트해야 한다.

ASVS 요구사항을 검증하기 위한 자동화 도구의 사용은 지속적으로 관심을 받는 주제이다. 따라서 자동화 테스트 및 블랙박스 테스트와 관련된 몇 가지 사항을 명확히 하는 것이 중요하다.

자동화 보안 테스트 도구의 역할

동적 애플리케이션 보안 테스트 (Dynamic Application Secure Test; DAST) 와 정적 애플리케이션 보안 테스트 (Static Application Secure Test; SAST) 와 같은 자동화 보안 테스트 도구를 빌드 파이프라인에 올바르게 구현하면, 초기에 제거되었어야 할 일부 보안 문제를 식별할 수 있다. 그러나 신중하게 구성하고 조정하지 않으면 요구되는 범위를 충족하지 못하며, 과도한 노이즈로 인해 실제 보안 문제가 식별 및 완화되는 것을 방해하게 된다.

이러한 도구들이 출력 인코딩이나 정제 (sanitization) 와 관련된 요구사항처럼 비교적 기본적이고 단순한 기술 요구사항 일부를 검증하는 데 도움이 될 수 있으나, 보다 복잡한 ASVS 요구사항이나 비즈니스 로직 및 접근 제어와 관련된 요구사항은 검증할 수 없다는 점을 명확히 인지해야 한다.

복잡한 요구사항에 대해서도 자동화를 활용할 수 있는 가능성은 있으며, 이를 위해서는 애플리케이션 특화 검증 항목을 직접 작성해야 할 수 있다. 이러한 검증 항목은 조직에서 이미 사용 중인 단위 테스트 (unit test) 나 통합 테스트 (integration test) 와 유사할 수 있다. 따라서 기존의 테스트 자동화 인프라를 활용하여 ASVS 요구사항에 특화된 테스트를 작성할 수 있다. 이러한 작업에는 단기적인 투자가 필요하지만, 장기간 지속적으로 해당 요구사항을 검증할 수 있는 이점이 크다.

요약하면, 자동화로 검증할 수 있다는 것은 시중 도구를 그대로 실행하는 것과 같지 않다.

침투 테스트의 역할

버전 4.0 의 L1 은 문서나 소스 코드 없이 수행되는 블랙박스 테스트에 최적화되어 있었으나, 당시에도 본 표준은 해당 방식이 효과적인 보증 활동이 아니며 적극적으로 지양되어야 한다는 점을 명시했다.

필요한 추가 정보에 접근하지 못한 채 수행되는 테스트는 비효율적일 뿐 아니라 효과적인 보안 검증 방식도 아니다. 이런 방식은 소스 코드를 검토하거나, 위협 요소와 통제 공백을 식별하거나, 더 짧은 시간 안에 훨 더 철저한 테스트를 수행할 수 있는 기회를 놓치게 만든다.

전통적인 침투 테스트보다는 애플리케이션 개발자와 문서에 완전하게 접근할 수 있는 문서 기반 또는 소스 코드 기반 (하이브리드) 침투 테스트를 강력히 권장한다. 많은 ASVS 요구사항을 검증하려면 이러한 방식이 사실상 필수적이다.

v4.x 대비 변경 사항

소개

4.x 버전에 익숙한 사용자는 내용, 범위 및 근본적인 철학의 변경을 포함하는 5.0 버전의 주요 변경 사항을 검토하는 것이 도움이 될 수 있다.

버전 4.0.3 의 286 개 요구사항 중 변경되지 않은 것은 11 개에 불과하며, 15 개는 의미를 바꾸지 않는 범위에서 문법적으로 소폭 조정되었다. 총 109 개 (38%) 의 요구사항은 더 이상 버전 5.0 에서 별도의 요구사항으로 존재하지 않으며, 이 중 50 개는 단순히 삭제되었고, 28 개는 중복으로 제거되었으며, 31 개는 다른 요구사항에 병합되었다. 나머지 요구사항들은 일부 수정되었다. 실질적인 수정이 이루어지지 않은 요구사항조차도 재배열이나 구조 변경으로 인해 식별자가 달라졌다.

5.0 버전의 원활한 채택을 위해, 4.x 버전과 5.0 버전의 요구사항 간 대응 관계를 확인할 수 있는 매핑 문서를 제공한다. 이 매핑 문서는 출시 버전과 연동되지 않으며, 필요에 따라 업데이트되거나 보완될 수 있다.

요구사항 철학

범위와 중점

버전 4.x 에는 표준에서 정의한 범위와 맞지 않는 요구사항이 포함되어 있었으며, 이는 제거되었다. 5.0 의 범위 기준에 부합하지 않거나 검증이 불가능한 요구사항 또한 제외되었다.

보안 메커니즘보다 보안 목표에 중점

버전 4.x 에서는 많은 요구사항이 근본적인 보안 목표보다 특정 메커니즘에 집중되어 있었다. 버전 5.0 에서는 요구사항이 보안 목표에 중심을 두며, 특정 메커니즘은 유일한 실질적 해결책인 경우에만 참조하거나 예시 또는 보충 안내로 제공한다.

이 접근법은 주어진 보안 목표를 달성하기 위해 여러 방법이 존재할 수 있음을 인정하며, 조직의 유연성을 제한할 수 있는 불필요한 규제를 지양한다.

또한, 동일한 보안 문제를 다루는 요구사항은 적절한 경우 통합되었다.

문서화된 보안 결정

문서화된 보안 결정이라는 개념은 버전 5.0 에서 새롭게 등장한 것처럼 보이나, 이는 버전 4.0 의 정책 적용과 위협 모델링 관련 요구사항의 진화된 형태이다. 이전에는 일부 요구사항이 허용된 네트워크 연결을 결정하는 등 보안 통제 구현을 위한 분석을 암묵적으로 요구하였다.

구현과 검증에 필요한 정보가 확보되도록, 이러한 기대사항을 분명하고 실행 가능하며 검증 가능한 문서화 요구사항으로 명확히 정의한다.

구조적 변화 및 신규 장

버전 5.0 에서는 여러 장에서 완전히 새로운 내용을 도입하였다:

- OAuth 및 OIDC –접근 위임 및 싱글 사인온 (single sign-on; SSO) 을 위한 이러한 프로토콜의 광범위한 채택을 고려하여, 개발자가 직면할 수 있는 다양한 시나리오를 다루기 위한 전용 요구사항이 추가되었다. 이 영역은 이전 버전에서 모바일 및 IoT 요구사항을 별도로 분리했던 것처럼, 궁극적으로 독립적인 표준으로 발전할 수 있다.
- WebRTC –이 기술의 보급이 확산됨에 따라, 고유한 보안 고려사항과 과제들을 별도의 섹션에서 다루고 있다.

또한, 관련된 요구사항들을 논리적으로 묶어 장과 섹션을 구성하도록 개선하였다.

이러한 구조 개편으로 인해 다음과 같은 신규 장이 추가되었다:

- 자체 포함 토큰–기존에는 세션 관리 항목에 포함되었으나, 이제는 독립적인 메커니즘이자 OAuth 및 OIDC 와 같은 무상태 통신의 기반 요소로 인식되어 별도의 장에서 다뤄진다. 고유한 보안 특성을 고려하여 전용 요구사항이 추가되었으며, 버전 5.x 에서 일부 신규 요구사항이 도입되었다.
- 웹 프론트엔드 보안–브라우저 기반 애플리케이션의 복잡성이 증가하고 API 중심 아키텍처가 확산됨에 따라, 프론트엔드 보안 요구사항을 별도의 장으로 분리하였다.
- 시큐어 코딩 및 아키텍처–기존 장에 포함되지 않았던 일반적인 보안 관행에 대한 신규 요구사항을 이 장에 통합하였다.

버전 5.0 의 기타 구성 변경은 목적의 명확화를 위한 것이다. 예를 들어, 입력 검증 요구사항은 정제 및 인코딩 항목과의 연관성보다는 비즈니스 규칙을 강제하는 역할에 초점을 맞추어 비즈니스 로직과 함께 배치되었다.

기존 V1 아키텍처 장은 삭제되었다. 서두에는 범위에서 벗어난 요구사항이 포함되어 있었고, 이후 섹션들은 관련 장으로 재배치되었으며, 요구사항은 중복 제거 및 명확화 작업이 이루어졌다.

외부 표준과의 직접 매핑 제거

표준 본문에서는 외부 표준과의 직접 매핑이 제거되었다. 이는 OWASP Common Requirement Enumeration (CRE) 프로젝트와의 매핑을 준비하기 위한 것으로, 이를 통해 ASVS 를 다양한 OWASP 프로젝트 및 외부 표준과 연결할 수 있도록 한다.

아래에서 설명하는 바와 같이, CWE 와 NIST 에 대한 직접 매핑은 더 이상 유지되지 않는다.

NIST 디지털 신원 지침과의 결합도 감소

NIST 디지털 신원 지침 (SP 800-63)은 오랫동안 인증 및 권한 부여 제어의 참고 자료로 활용되어 왔다. 버전 4.x 에서는 일부 장이 NIST 의 구조와 용어에 밀접하게 정렬되어 있었다.

이러한 지침은 여전히 중요한 참고 자료이지만, 지나치게 엄격한 정렬은 널리 통용되지 않는 용어 사용, 유사 요구사항의 중복, 불완전한 매핑과 같은 문제를 초래하였다. 버전 5.0 에서는 명확성과 실효성을 높이기 위해 이러한 접근 방식을 지양한다.

CWE(Common Weakness Enumeration) 와의 결합도 감소

CWE(Common Weakness Enumeration)은 소프트웨어 보안 약점에 대한 유용한 분류 체계를 제공한다. 그러나 카테고리 전용 CWE, 단일 CWE 에 대한 매핑의 어려움, 버전 4.x 의 부정확한 매핑 등 여러 문제로 인해, 버전 5.0 에서는 CWE 와의 직접 매핑을 중단하기로 결정하였다.

보안 수준 정의 재검토

버전 4.x 에서는 L1(“최소”), L2(“표준”), L3(“고급”) 으로 수준을 정의하고, 민감한 데이터를 처리하는 모든 애플리케이션은 최소한 L2 를 충족해야 한다고 간주하였다.

버전 5.0 에서는 다음과 같은 단락에서 설명되는 여러 문제점을 해결하였다.

실무적으로는, 버전 4.x 에서 수준 표시로 사용되던 체크 마크 대신, 버전 5.x 에서는 markdown, PDF, DOCX, CSV, JSON, XML 등 모든 형식에서 단순 숫자 표기를 사용한다. 하위 호환성을 위해, 기존 체크 마크 형식을 유지하는 CSV, JSON, XML 출력물도 함께 제공된다.

진입 장벽 완화

레벨 1 이"최소 수준"으로 간주되면서도 대부분의 애플리케이션에 충분하지 않다는 인식, 그리고 약 120 개의 요구사항이 포함된다는 점이 채택을 저해한다는 피드백이 있었다. 버전 5.0 에서는 L1 을 주로 1 차 방어 계층 요구사항 중심으로 정의함으로써, 보다 명확하고 간결한 요구사항으로 진입 장벽을 낮추고자 하였다. 수치적으로는, v4.0.3 에서는 전체 278 개 중 128 개가 L1 로, 46% 를 차지했으며, 5.0.0 에서는 전체 345 개 중 70 개로, 20% 에 해당한다.

테스트 가능성의 오류

버전 4.x 에서 레벨 1 요구사항을 선정하는 주요 기준은 외부 블랙박스 침투 테스트를 통한 평가 가능성이었다. 그러나 이러한 접근은 L1 을 최소 보안 통제 집합으로 정의한 본래 목적과 완전히 부합하지 않았다. 일부 사용자는 L1 이 보안을 보장하기에 부족하다고 보았고, 다른 사용자들은 테스트가 너무 어렵다고 평가하였다.

테스트 가능성을 기준으로 삼는 것은 상대적일 뿐만 아니라 때로는 오해를 불러일으킨다. 요구사항이 테스트 가능하다는 사실이 자동화되거나 간단한 방식으로 테스트할 수 있음을 보장하지는 않는다. 더 나아가, 가장 쉽게 테스트할 수 있는 요구사항이 반드시 가장 큰 보안 효과를 가지거나 가장 구현이 용이한 요구사항인 것도 아니다.

따라서 버전 5.0 에서는 수준 결정 시 위험 감소 효과를 우선적으로 고려하되, 구현에 필요한 노력도 함께 고려하였다.

위험 기반에만 의존하지 않는 접근

특정 애플리케이션에 대해 정해진 수준을 요구하는 위험 기반 접근은 지나치게 경직된 것으로 드러났다. 실제로 보안 통제의 우선순위 결정 및 구현은 위험 감소 효과뿐 아니라 구현 난이도 등 다양한 요소에 따라 달라진다.

따라서 조직은 자신의 성숙도와 사용자에게 전달하려는 메시지에 따라 적절하다고 판단되는 수준을 달성하는 것이 권장된다.

V1 인코딩과 데이터 정제

제어 목표

이번 장에서는 신뢰할 수 없는 데이터를 안전하지 않게 처리하여 발생하는 가장 흔한 웹 애플리케이션 보안 약점들을 다룬다. 이러한 약점들은 신뢰할 수 없는 데이터가 관련 인터프리터의 문법 규칙에 따라 해석되면서 다양한 기술적 취약점들로 이어질 수 있다.

현대의 웹 애플리케이션에서는 파라미터화된 쿼리, 자동 이스케이핑, 또는 템플릿 프레임워크와 같은 더 안전한 API 들을 쓰는 것이 항상 가장 좋다. 즉, 출력값 인코딩, 이스케이핑, 또는 데이터 정제를 신중하게 처리하는 것은 애플리케이션의 보안에 매우 중요하다.

입력값 검증은 예상치 못한 입력이나 위험한 콘텐츠로부터 보호하기 위한 심층 방어 (defense-in-depth) 메커니즘 역할을 한다. 다만, 입력값 검증의 주요 목적은 입력된 데이터가 기능 및 비즈니스 요구사항에 부합하는지를 확인하는 데 있으므로, 이에 관련된 요구사항은“검증 및 비즈니스 로직”장에서 다루고 있다.

V1.1 인코딩 및 데이터 정제 아키텍처

아래의 섹션에서는 위험한 콘텐츠를 안전하게 처리하여 보안 취약점을 방지하기 위한 문법 또는 인터프리터별 요구사항들이 제시된다. 이 요구사항들은 처리가 이루어져야 할 순서와 위치에 대한 내용도 포함하고 있다. 또한 데이터가 저장될 때 인코딩되거나 이스케이프된 형태 (예: HTML 인코딩) 가 아닌 원래의 상태로 저장되도록 보장함으로써 이중 인코딩 문제를 방지하는 데 목적이 있다.

#	설명	레벨
1.1.1	입력값 검증은 오직 한 번만 표준 형태로 디코딩 되거나 언이스케이프 되어야 하며, 인코딩된 데이터가 예상될 때만 디코딩 되어야 한다. 이 과정은 입력이 추가로 처리되기 전에 완료되어야 한다. 예를 들어서 입력값 검증이나 데이터 정제 이후에는 수행되면 안 된다.	2
1.1.2	애플리케이션이 출력값 인코딩 또는 이스케이핑을 인터프리터가 사용하기 전의 최종 단계로 수행하거나 인터프리터 자체에서 처리하는지 검증해야 한다.	2

V1.2 인젝션 (Injection) 방지

잠재적으로 위험 컨텍스트 (context) 에 인접하거나 가까운 위치에서 수행되는 출력값 인코딩이나 이스케이핑은 애플리케이션의 보안에 매우 중요하다. 일반적으로 출력값 인코딩과 이스케이핑은 저장되지 않으며, 대신 해당 출력값을 적절한 인터프리터에서 즉시 안전하게 렌더링하기 위해 사용된다. 이를 너무 이른 시점에 수행하려고 하면 콘텐츠가 잘못 구성되거나 인코딩 및 이스케이핑이 무효화될 수 있다.

많은 경우에서 소프트웨어 라이브러리에는 이를 자동으로 처리하는 안전한 함수들이 포함되어 있지만 현재 컨텍스트에 적합한지 확인이 필요하다.

#	설명	레벨
1.2.1	HTTP 응답, HTML 문서 또는 XML 문서에 대한 출력값 인코딩이 해당 컨텍스트에 적절한지 검증해야 한다. 예를 들어, 메시지나 문서 구조가 변경되지 않도록 HTML 요소, HTML 속성, HTML 주석, CSS, HTTP 헤더 필드 등에 맞는 문맥별 문자 인코딩이 수행되어야 한다.	1
1.2.2	통합 자원 식별자 (Uniform Resource Locator; URL) 들을 동적으로 생성할 때 신뢰할 수 없는 데이터가 해당 컨텍스트에 맞게 인코딩되었는지 검증해야 한다 (예: 쿼리나 경로 파라미터에 대한 URL 인코딩 또는 base64url 인코딩). 또한, 오직 안전한 URL 프로토콜만이 허용되는지 검증해야 한다 (예: javascript: 또는 data: 를 허용하지 않음).	1

#	설명	레벨
1.2.3	동적으로 자바스크립트 콘텐츠 (JSON 포함) 를 생성할 때 메시지나 문서 구조를 바꾸는 것을 방지하기 위해 출력값 인코딩이나 이스케이핑이 사용되고 있는지 검증해야 한다 (자바스크립트 또는 JSON 인젝션을 피하기 위함).	1
1.2.4	데이터 선택 혹은 데이터베이스 쿼리 (예: SQL, HQL, NoSQL, Cypher) 들 파라미터화된 쿼리, 객체 관계 매핑 (Object Relational Mapping; ORM), 엔티티 프레임워크, 또는 다른 구조화된 쿼리 언어 (Structured Query Language; SQL) 인젝션 및 기타 데이터베이스 인젝션 공격으로부터 보호해 주는 것들을 사용하는지 검증해야 한다. 이는 저장 프로시저를 쓸 때도 마찬가지이다.	1
1.2.5	애플리케이션이 운영체제 (Operating System; OS) 명령 인젝션으로부터 보호되며 운영체제 호출 시 파라미터화된 OS 쿼리나 상황에 맞는 명령줄 출력값 인코딩을 사용하는지 검증해야 한다.	1
1.2.6	애플리케이션이 경량 디렉터리 접근 프로토콜 (Lightweight Directory Access Protocol; LDAP) 인젝션 취약점에 대해 보호되어 있는지, 또는 LDAP 인젝션을 방지하기 위한 특정 보안 통제가 구현되어 있는지 검증해야 한다.	2
1.2.7	애플리케이션이 XPath 인젝션 공격을 쿼리 파라미터나 미리 컴파일된 쿼리들을 사용해 보호되고 있는지 검증해야 한다.	2
1.2.8	LaTeX 프로세서가 안전하게 구성되어 있으며 (“–shell-escape”플래그 비사용 등) LaTeX 인젝션 공격을 방지하기 위해 허용된 명령어 목록이 사용되고 있는지 검증해야 한다.	2
1.2.9	애플리케이션이 정규 표현식에서 특수 문자들을 메타 문자로 잘못 해석하는 것을 방지하기 위해 해당 문자들 이스케이프 (보통 백슬래시를 사용) 하고 있는지 검증해야 한다.	2
1.2.10	애플리케이션이 쉼표 구분 데이터 (Comma Separated Values; CSV) 및 수식 인젝션에 대해 보호되어 있는지 검증해야 한다. 그리고 애플리케이션은 CSV 콘텐츠를 내보낼 때 RFC 4180 2.6 과 2.7 부분에 명시된 이스케이핑 규칙을 반드시 따라야 한다. 또한 CSV 나 다른 스프레드시트 포맷 (XLS, XLSX, ODF 등) 으로 내보낼 때 필드 값의 첫 번째가 특수 문자 (‘=’, ‘+’, ‘-’, ‘@’, ‘\t’(탭), ‘\0’(null 문자) 포함) 라면 반드시 작은따옴표로 이스케이프되어야 한다.	3

참고: 파라미터화된 쿼리를 사용하거나 SQL 을 이스케이핑만으로 항상 충분하지 않다. 테이블명과 컬럼명 (“ORDER BY” 컬럼 이름을 포함) 과 같은 쿼리의 일부는 이스케이프 될 수 없다. 이러한 필드에 이스케이프된 사용자 입력 데이터를 포함하면 쿼리가 실패하거나 SQL 인젝션에 취약해질 수 있다.

V1.3 데이터 정제

신뢰할 수 없는 콘텐츠를 안전하지 않은 컨텍스트에서 사용하는 것을 이상적으로 방지하는 방법은 해당 컨텍스트에 맞는 인코딩 또는 이스케이핑을 사용하는 것이다. 이를 통해 안전하지 않은 콘텐츠의 시맨틱 (semantic) 의미는 그대로 유지하면서 해당 컨텍스트를 안전하게 유지할 수 있다. 이에 대한 자세한 내용은 앞 부분에서 다루었다.

이와 같은 처리가 불가능한 경우에 잠재적으로 위험한 문자나 콘텐츠를 제거하는 데이터 정제는 필수적이다. 일부 경우에 이로 인해 입력의 시맨틱 의미가 바뀔 수 있다. 그러나 보안상의 이유로 다른 대안이 없을 수 있다.

#	설명	레벨
1.3.1	위지윅 (WYSIWYG) 에디터나 이와 유사한 것으로부터 오는 모든 신뢰할 수 없는 HTML 입력들은 잘 알려지고 안전한 HTML 데이터 정제 라이브러리나 프레임워크 기능을 사용해 정제되는지 검증해야 한다.	1
1.3.2	애플리케이션이 eval() 또는 스프링 표현 언어 (Spring Expression Language; SpEL) 등과 같은 동적 코드 실행 기능 사용을 피하고 있는지 검증해야 한다. 대안이 없는 경우, 실행 전에 포함되는 모든 사용자 입력이 반드시 정제되어야 한다.	1
1.3.3	잠재적으로 위험한 컨텍스트에 전달되는 데이터가 사전에 정제되어 안전 조치가 적용되는지 검증해야 한다. 예를 들어 해당 컨텍스트에서 안전한 문자만 허용하고 너무 긴 입력값은 잘라내는 등의 조치가 포함된다.	2
1.3.4	사용자로부터 제공된 확장 가능한 벡터 그래픽 (Scalable Vector Graphics; SVG) 의 스크립트 실행 가능 콘텐츠가 애플리케이션에 안전한 태그 및 속성 (예: 그래픽 그리기) 만 포함하도록 검증 또는 정제되는지 검증해야 한다. 예를 들어 스크립트나 foreignObject 는 포함되면 안 된다.	2
1.3.5	애플리케이션이 사용자로부터 제공된 스크립트 실행 가능하거나 표현식 템플릿 언어 (예: 마크다운, CSS, XSL 스타일시트, BBCode) 콘텐츠를 정제하거나 비활성화하는지 검증해야 한다.	2
1.3.6	애플리케이션이 서버측 요청 위조 (Server-side Request Forgery; SSRF) 공격으로부터 보호하기 위해 신뢰할 수 없는 데이터를 프로토콜, 도메인, 경로, 포트의 허용 목록과 대조해 검증하고 다른 서비스를 호출하기 전에 잠재적으로 위험한 문자를 정제 (sanitization) 하는지 검증해야 한다.	2
1.3.7	애플리케이션이 템플릿 인젝션 공격으로부터 보호하기 위해 신뢰할 수 없는 입력값을 기반으로 템플릿을 생성하지 않도록 하는지 검증해야 한다. 대안이 없는 경우에는 템플릿 생성 과정에서 동적으로 포함되는 모든 신뢰할 수 없는 입력값은 정제되거나 엄격히 검증되어야 한다.	2
1.3.8	애플리케이션이 자바 명명 및 디렉터리 인터페이스 (Java Naming and Directory Interface; JNDI) 쿼리에서 신뢰할 수 없는 입력값을 사용하기 전에 적절히 정제하고 JNDI가 JNDI 인젝션 공격을 방지할 수 있도록 안전하게 구성되어 있는지 검증해야 한다.	2
1.3.9	인젝션을 방지하기 위해 애플리케이션이 메모리 캐시 (memcache) 에 콘텐츠를 전달하기 전에 해당 데이터를 정제하는지 검증해야 한다.	2
1.3.10	사용 시 예상치 못하거나 악의적인 방식으로 작동할 수 있는 포맷 스트링 (format string) 들은 처리되기 전에 정제되는지 검증해야 한다.	2
1.3.11	간이 전자 우편 전송 프로토콜 (Simple Mail Transfer Protocol; SMTP) 이나 인터넷 메시지 액세스 프로토콜 (Internet Message Access Protocol; IMAP) 인젝션으로부터 보호하기 위해 애플리케이션이 사용자 입력을 메일 시스템으로 보내기 전에 정제를 하는지 검증해야 한다.	2

#	설명	레벨
1.3.12	정규 표현식에 지수형 백트래킹을 유발하는 요소가 없는지, 또한 신뢰할 수 없는 입력값을 정제하여 정규식 기반 서비스 거부 공격 (Regular Expression Denial of Service; ReDoS) 또는 런어웨이 (Runaway) 정규식 공격을 방지하는지 검증해야 한다.	3

V1.4 메모리, 스트링, 비관리 코드

다음 요구사항들은 안전하지 않은 메모리 사용과 관련된 위험 요소들을 다루며 일반적으로 애플리케이션이 시스템 언어나 비관리 코드를 사용할 때 적용된다.

일부 경우에는 컴파일러 플래그를 설정하여 버퍼 오버플로우 방지 및 경고 기능을 활성화함으로써 보호할 수 있다. 이에는 스택 무작위화, 데이터 실행 방지 등이 포함되고 위험한 포인터, 메모리, 포맷 문자열, 정수 또는 문자열 연산이 발견될 경우 빌드를 중단하도록 설정할 수도 있다.

#	설명	레벨
1.4.1	애플리케이션이 스택, 버퍼, 또는 힙 오버플로를 탐지하거나 방지하기 위해 메모리 안전 문자열, 안전한 메모리 복사 및 포인터 산술을 사용하는지 검증해야 한다.	2
1.4.2	정수 오버플로를 방지하기 위해 부호 확인, 범위 검사, 입력값 검증 등의 기법이 사용되고 있는지 검증해야 한다.	2
1.4.3	댕글링 포인터 (dangling pointer) 나 해체 후 사용 (Use After Free; UAF) 취약점을 방지하기 위해 동적으로 할당된 메모리와 자원이 적절히 해제되었는지 그리고 해제된 메모리를 참조하는 포인터나 참조가 제거되었거나 null 로 설정되었는지를 검증해야 한다.	2

V1.5 안전한 역직렬화

저장되었거나 전송된 데이터를 애플리케이션의 실제 객체로 변환하는 과정 (역직렬화) 은 역사적으로 다양한 코드 인젝션 취약점의 원인이 되어왔다. 이 과정을 조심스럽고 안전하게 수행하는 것은 이러한 종류의 문제를 피하는데 중요하다.

특히, 일부 역직렬화 방법은 프로그래밍 언어나 프레임워크 문서에서 안전하지 않거나 신뢰할 수 없는 데이터를 안전하게 만들 수 없다고 지정되어 왔다. 사용 중인 각 메커니즘에 대해 신중한 검토 (실사) 를 수행해야 한다.

#	설명	레벨
1.5.1	애플리케이션이 확장 가능한 마크업 언어 (eXtensible Markup Language; XML) 파서 (parser) 를 제한적인 설정으로 구성하고 외부 엔티티 해석과 같은 안전하지 않은 기능을 비활성화하여 XML 외부 엔티티 (XML eXternal Entity; XXE) 공격을 방지하는지 검증해야 한다.	1

#	설명	레벨
1.5.2	역직렬화 공격을 방지하기 위해 신뢰할 수 없는 데이터의 역직렬화 시 객체 타입 허용 목록 사용 또는 클라이언트 정의 객체 타입 제한과 같은 안전한 입력 처리 방식이 적용되는지 검증해야 한다. 또한 명확히 안전하지 않은 것으로 지정된 역직렬화 메커니즘은 신뢰할 수 없는 입력에 대해 반드시 사용해서는 안 된다.	2
1.5.3	애플리케이션에서 동일한 데이터 타입 (예: JSON 파서, XML 파서, URL 파서) 에 대해 사용되는 서로 다른 파서들이 일관된 방식으로 파싱 (parsing) 을 수행하고 동일한 문자 인코딩 방식을 사용하는지 검증해야 한다. 이를 통해 JSON 상호운용 취약점이나 원격 파일 포함 (Remote File Inclusion; RFI) 및 SSRF 공격에서 악용될 수 있는 서로 다른 URI 또는 파일 파싱 동작 문제를 방지할 수 있다.	3

참조

자세한 내용은 다음을 참고:

- OWASP LDAP 인젝션 방지 치트 시트 (Cheat Sheet)
- OWASP 사이트 간 스크립팅 (Cross Site Scripting; XSS) 방지 치트 시트
- OWASP 문서 객체 모델 (Document Object Model; DOM) 기반 XSS 방지 치트 시트
- OWASP XXE 방지 치트 시트
- OWASP 웹 보안 테스트 가이드: 클라이언트 측 테스트
- OWASP 자바 인코딩 프로젝트
- DOMPurify - 클라이언트 측 HTML 정제 라이브러리
- RFC4180 - CSV 파일의 공통 포맷 (format) 및 다목적 인터넷 메일 확장 (Multipurpose Internet Mail Extensions; MIME) 타입

역직렬화 또는 파싱 문제에 대한 자세한 내용은 다음을 참고:

- OWASP 역직렬화 치트 시트
- JSON 상호운용 취약점 탐구
- Orange Tsai –트렌딩 (Trending) 프로그래밍 언어의 URL 파서를 악용한 SSRF 의 새로운 시대

V2 유효성 검증 및 비즈니스 로직

제어 목표

이번 장에서는 검증된 애플리케이션이 아래와 같은 상위 수준의 목표를 충족하는지 확인하는 것이 목표이다.

- 애플리케이션에 입력되는 값은 비즈니스 및 기능 요구사항에 부합한다.
- 비즈니스 로직은 정해진 순서에 따라 순차적으로 실행되며, 특정 단계를 건너뛰거나 우회할 수 없다.

- 비즈니스 로직에는 지속적인 소액 송금이나 한 번에 친구 백만 명 추가와 같은 자동화된 공격을 탐지하고 방지하기 위한 제한 및 제어 기능이 포함되어 있다.
- 핵심적인 비즈니스 로직은 악용 사례와 악의적인 공격자를 고려하여 설계되었으며, 스푸핑, 데이터 위변조, 정보 유출, 권한 상승 공격에 대한 보호 조치를 갖추고 있다.

V2.1 유효성 검증 및 비즈니스 로직 문서화

유효성 검증 및 비즈니스 로직 관련 문서는 애플리케이션에 구현해야 할 사항을 명확히 하기 위해서 비즈니스 로직의 제한 사항, 유효성 검증 규칙, 그리고 여러 데이터 항목 간의 문맥적 일관성을 명확하게 정의해야 한다.

#	설명	레벨
2.1.1	애플리케이션 문서에 데이터 항목이 예상되는 구조에 부합하는지 확인하는 입력값 유효성 검증 규칙이 정의되어 있는지 확인한다. 이는 신용카드 번호, 이메일 주소, 전화번호와 같은 일반적인 데이터 형식일 수도 있고, 내부적으로 사용하는 데이터 형식일 수도 있다.	1
2.1.2	애플리케이션 문서에 연관된 데이터 항목들 간의 논리적, 문맥적 일관성을 검증하는 방법이 정의되어 있는지 검증한다. 예를 들어서 세부 주소와 우편번호가 서로 일치하는지 확인하는 경우가 이에 해당한다.	2
2.1.3	비즈니스 로직의 제한사항 및 유효성 검증에 대한 요구사항이 사용자별 기준과 애플리케이션 전체에 적용되는 전역적 기준을 모두 포함하여 문서화되어 있는지 검증한다.	2

V2.2 입력값 검증

효과적인 입력값 검증은 애플리케이션이 받아들이고자 하는 데이터의 유형에 관한 비즈니스 또는 기능적 요구사항을 강제하는 제어 기능이다. 이를 통해 데이터 품질을 높이고 공격 표면을 줄일 수 있다. 하지만 이는 데이터를 다른 컴포넌트에서 사용하거나 출력용으로 표시할 때 필요한 올바른 인코딩, 매개변수화, 또는 정제의 필요성을 대체하거나 없애주는 것은 아니다.

이 문맥에서“입력값”이란 HTML 양식 필드, REST 요청, URL 매개변수, HTTP 헤더, 쿠키, 디스크의 파일, 데이터베이스, 외부 API 등 매우 다양한 출처로부터 유입될 수 있는 데이터를 의미한다.

예를 들어, 비즈니스 로직 제어는 특정 입력값이 100 보다 작은 숫자인지를 확인할 수 있다. 반면 기능적 기대사항은 특정 숫자가 정해진 임계값 이하인지 검사할 수 있다. 이는 해당 숫자가 특정 루프의 반복 횟수를 제어하는 역할을 하며, 값이 지나치게 크면 과도한 처리 부하로 인해 서비스 거부 (DoS) 상태를 초래할 수 있기 때문이다.

스키마 유효성 검증 (schema validation) 이 명시적으로 의무화된 것은 아니지만, JSON 이나 XML 을 사용하는 HTTP API 또는 기타 인터페이스에서 입력 전체에 대한 포괄적인 유효성 검증을 수행하는 데 가장 효과적인 방법이 될 수 있다.

스키마 검증에 대한 아래 사항들에 유의해야 한다:

- JSON 스키마 검증 사양의“공식 배포 버전”은 운영 환경에서 사용 가능할 정도로 안정적이지만, 엄밀히 말해“완전히 안정된”상태는 아니다.

- 사용 중인 JSON 스키마 검증 라이브러리도, 표준이 공식화되면 이를 반영하여 모니터링하고 필요시 업데이트해야 한다.
- DTD(문서 타입 정의) 검증은 사용해서는 안 되며, 프레임워크의 DTD 평가 기능은 반드시 비활성화해야 한다. 이는 DTD 를 악용한 XXE(XML 외부 개체) 공격을 방지하기 위함이다.

#	설명	레벨
2.2.1	입력값에 대한 비즈니스 또는 기능적인 기대치를 충족하기 위해서 입력값의 유효성을 검증해야 한다. 이는 허용 값, 패턴 및 범위 목록에 대한 긍정적 검증을 사용하거나, 미리 정의된 규칙에 따라 입력값을 예상 구조 및 논리적 한계와 비교하여 검증해야 한다. L1 의 경우, 특정 비즈니스 또는 보안 결정을 내리는 데 초점을 맞출 수 있다. L2 이상의 경우에는 모든 입력값에 적용해야 한다.	1
2.2.2	애플리케이션이 신뢰할 수 있는 서비스 계층에서 입력값 유효성 검사를 시행하도록 설계되었는지 확인해야 한다. 클라이언트 측 유효성 검사는 사용성을 향상시키므로 권장해야 하지만, 보안 제어 수단으로 의존해서는 안된다.	1
2.2.3	사전 정의된 규칙에 따라 관련 데이터 항목의 조합이 합리적인지 애플리케이션에서 확인해야 한다.	2

V2.3 비즈니스 로직 보안

이 챕터에서는 애플리케이션이 비즈니스 로직 프로세스를 올바른 방식으로 적용하고 애플리케이션의 로직과 흐름을 악용하는 공격에 취약하지 않은지 확인하기 위한 핵심 요구 사항을 고려한다.

#	설명	레벨
2.3.1	애플리케이션이 예상되는 순차적 단계 순서에 따라 동일한 사용자에 대한 비즈니스 로직 흐름만 처리하고 단계를 건너뛰지 않는지 확인한다.	1
2.3.2	비즈니스 로직 결함이 악용되는 것을 방지하기 위해서 애플리케이션 문서에 따라 비즈니스 로직 제한이 구현되었는지 확인한다.	2
2.3.3	비즈니스 로직 수준에서 트랜잭션이 사용되고 있는지 확인하여 비즈니스 로직 작업이 완전히 성공하거나 이전의 올바른 상태로 롤백되는지 확인한다.	2
2.3.4	제한된 양의 리소스 (ex. 극장 좌석이나 배송 시간대) 가 애플리케이션의 로직을 조작하여 이중으로 예약되는 것을 방지하기 위해서 비즈니스 로직 수준 잠금 메커니즘이 사용되는지 확인한다.	2
2.3.5	핵심적인 비즈니스 로직 흐름에 무단 또는 우발적인 행위를 방지하기 위해서 다중 사용자 승인이 필요한지 확인해야 한다. 이는 거액 자금 이체, 계약 승인, 기밀 정보 접근, 제조 과정의 안전 무시 등이 포함될 수 있지만, 이에 국한되지는 않는다.	3

V2.4 자동화 방지

이 섹션은 인간과 유사한 상호작용을 요구하고, 과도한 자동화 요청을 방지하기 위한 자동화 방지 제어 기능들을 포함한다.

#	설명	레벨
2.4.1	데이터 유출, 더미 데이터 생성, 할당량 소진, 속도 제한 위반, 서비스 거부 (DoS), 핵심적인 리소스의 과다 사용으로 이어질 수 있는 애플리케이션 기능에 대한 과도한 호출을 방어하기 위해, 자동화 방지 제어 기능이 구현되어 있는지 검증한다.	2
2.4.2	비즈니스 로직의 각 단계가 실제 사람이 수행하는 데 필요한 현실적인 시간을 요구하여, 비정상적으로 빠른 속도의 트랜잭션 제출을 방지하는지 검증한다.	3

참조

자세한 내용은 다음을 참고한다.

- OWASP Web Security Testing Guide: Input Validation Testing
- OWASP Web Security Testing Guide: Business Logic Testing
- Anti-automation can be achieved in many ways, including the use of the OWASP Automated Threats to Web Applications
- OWASP Input Validation Cheat Sheet
- JSON Schema

V3 웹 프론트엔드 보안

제어 목표

이번 장에서는 웹 프론트엔드를 통해 수행되는 공격으로부터 보호하기 위한 요구사항에 초점을 둔다. 이러한 요구사항은 기계 간 통신 (M2M) 솔루션에는 적용되지 않는다.

V3.1 웹 프론트엔드 보안 문서

이 섹션에서는 애플리케이션의 공식 문서에서 알려줘야 할 브라우저 내 보안 특징들에 대해 설명한다.

#	설명	레벨
3.1.1	애플리케이션 공식 문서에서 애플리케이션이 반드시 지원해야 하는, 예상되는 보안 특징들을 서술하고 있는지 확인한다. (예: HTTPS, HTTP Strict Transport Security (HSTS), Content Security Policy (CSP), 그리고 다른 관련된 HTTP 보안 메커니즘들). 또한 문서에서 앞선 특징들 (유저에게나 경고하거나 접속을 막는 등의) 이 유효하지 않을 때 애플리케이션이 어떻게 행동하는지 반드시 정의해야 한다.	3

V3.2 의도하지 않은 콘텐츠 접근

잘못된 콘텐츠나 기능을 불러오는 작업은 위협적인 콘텐츠를 실행하거나 불러오는 결과를 초래한다.

#	설명	레벨
3.2.1	브라우저가 HTTP 응답에 대해 콘텐츠나 기능을 불러올 때, 의도하지 않은 상황이 보안적으로 통제되고 있는지 확인한다 (예: API, 유저가 올린 파일, 혹은 다른 리소스에 직접 접근 시). Content-Security-Policy 헤더 필드의 샌드박스 지시어를 사용하거나, Content-Disposition 헤더 필드 내의 attachment disposition 타입을 사용하는 등으로, 불필요한 HTTP 요청 헤더 필드 (Sec-Fetch-* 와 같은) 를 제공하지 않을 수 있다.	1
3.2.2	콘텐츠가 HTML 로 렌더링되지 않고 텍스트로 표시되도록 하여, HTML 이나 JavaScript 등의 의도하지 않은 실행을 방지하는 안전한 렌더링 함수 (예: createTextNode, textContent) 를 사용하는지 확인한다.	1
3.2.3	애플리케이션이 클라이언트 사이드 JavaScript 를 사용할 때, 명시적 변수 사용, 엄격한 타입 검사, 문서에서의 전역 변수 저장 지양, 그리고 네임스페이스 격리 구현 등으로 DOM 클로버링을 막고 있음을 확인한다.	3

V3.3 쿠키 설정

이 섹션에서는 민감한 쿠키를 설정할 때, 애플리케이션 자체에서 만들어졌고, 콘텐츠 내용이 새어 나가거나, 부적절한 수정이 되는 것을 막는 등의 더 높은 수준의 보안을 제공하기 위한 요구사항에 대해 설명한다.

#	설명	레벨
3.3.1	쿠키가‘Secure’속성을 가지며, 쿠키 이름에’__Host-’접두사가 사용되지 않았고, ’__Secure-’접두사를 사용했는지 확인한다.	1
3.3.2	UI redress 공격, 그리고 CSRF 라 알려진 브라우저 기반 요청 위조 공격들에 대한 노출을 제한하기 위해 각 쿠키의‘SameSite’속성 값이 쿠키의 목적에 맞게 설정되어 있는지 확인한다.	2
3.3.3	다른 호스트들과 명시적으로 공유되는 쿠키에 대해서, ’__Host-’접두사를 가지고 있는지 확인한다.	2

#	설명	레벨
3.3.4	클라이언트 측 스크립트에서 쿠키의 값에 접근이 가능한 걸 의도하지 않았다면 (세션 토큰 등), 쿠키는 반드시‘HttpOnly’속성을 가져야 하고, 반드시‘Set-Cookie’헤더 필드를 통해 클라이언트에게 전달돼야 한다.	2
3.3.5	애플리케이션이 쿠키를 만들 때, 쿠키 이름과 값의 길이 합이 4096 바이트를 넘지 않는지 확인한다. 너무 큰 쿠키는 브라우저에 저장되지 않으며, 요청 전달도 안되고, 해당 쿠키에 의존하는 애플리케이션 기능을 사용하지 못하게 된다.	3

V3.4 브라우저 보안 메커니즘 헤더

이 섹션에서는 브라우저 보안 기능들을 활성화하기 위해 HTTP 응답에 설정되어야 할 보안 헤더들과, 애플리케이션에서의 응답들을 다루기 위한 제한사항들에 대해 설명한다.

#	설명	레벨
3.4.1	이 섹션에서는 애플리케이션의 응답을 처리할 때 브라우저의 보안 기능과, 제한 사항을 활성화하기 위해 HTTP 응답에 설정해야 하는 보안 헤더를 설명한다.	1
3.4.2	교차 출처 리소스 공유 (CORS) 의 Access-Control-Allow-Origin 헤더 필드가 애플리케이션의 고정 값인지 확인한다. 만약 출처 Origin HTTP 요청 헤더 필드 값을 사용하고 있다면, 신뢰할 수 있는 출처 목록으로 검증한다. ’Access-Control-Allow-Origin: *’사용이 필요할 때엔, 응답이 민감한 정보를 포함하지 않는지 확인한다.	1
3.4.3	HTTP 응답이 Content-Security-Policy 응답 헤더 필드를 가지고 있는지 확인한다. 이는 브라우저가 신뢰하는 콘텐츠나 리소스만 불러오고 실행하도록 하고, 위험한 JavaScript 의 실행을 제한한다. 최소한, object-src ‘none’과 base-uri ’none’이라는 지침, 그리고 허용 목록을 정의하거나 논스, 해쉬의 사용을 포함한다는 전역 정책은 반드시 사용돼야 한다. 3 계층 애플리케이션의 경우에는, 논스나 해쉬가 적용된 응답별 정책이 반드시 정의돼야 한다.	2
3.4.4	모든 HTTP 응답이‘X-Content-Type-Options: nosniff’헤더 필드를 포함하는지 확인해야 한다. 이는 브라우저들이 주어진 응답에 대해 콘텐츠 스니핑과 MIME 타입 추측을 하지 않도록, 응답의 Content-Type 헤더 필드 값을 목적지 리소스와 일치하도록 지시한다. 예를 들어서, 응답의 Content-Type 이’text/css’인 스타일만 허용된다고 치자. 이는 기능적으로 브라우저의 Cross-Origin Read Blocking (CORB) 사용 또한 활성화시킨다.	2
3.4.5	‘Referer’HTTP 요청 헤더 필드를 통해 제 3 자 서비스들에 기술적으로 민감한 정보 누출을 막기 위해, 애플리케이션에서 리퍼러 정책을 설정했는지 확인해야 한다. 이는 Referrer-Policy HTTP 응답 헤더 필드나 HTML 속성 항목을 통해 설정할 수 있다. 민감한 데이터는 URL 에 있는 경로나 쿼리 데이터, 그리고 내부 비공개 애플리케이션의 경우 hostname 도 포함된다.	2

#	설명	레벨
3.4.6	웹 애플리케이션이 Content-Security-Policy 헤더 필드의 frame-ancestors 를 사용해 모든 HTTP 응답들에 대해 기본적으로 내부 삽입을 할 수 없고, 필요할 때만 특정 리소스에 대해 내부 삽입을 허용하도록 확인한다. X-Frame-Options 헤더 필드가 브라우저에서 지원되더라도, 오래돼서 신뢰할 수 없음을 알아야 한다.	2
3.4.7	Content-Security-Policy 헤더 필드에 위반 사항을 보고할 영역이 특정되어 있는지 확인해야 한다.	3
3.4.8	문서 렌더링을 시작하는 모든 HTTP 요청 (Content-Type text/html 과 같은 응답) 에 same-origin 지시어나 same-origin-allow-popups 이 있는 Cross-Origin-Opener-Policy 헤더 필드를 포함하는지 확인해야 한다. 이는 탭내빙 (tabnabbing) 및 프레임 카운팅과 같은 Window 객체에 대한 공유 액세스를 악용하는 공격을 방지할 수 있다.	3

V3.5 브라우저 출처 구분

서버 측에서 민감한 기능에 대한 요청을 수락할 때, 애플리케이션은 그 요청이 애플리케이션 자체나 신뢰할 수 있는 당사자에 의해 시작되었는지, 공격자에 의해 위조되지 않았는지 확인해야 한다.

이 섹션에서, 민감한 기능에는 인증된 사용자와 인증되지 않은 사용자의 form post 수락 (인증 요청 등), 상태 변경 작업 또는 리소스를 많이 소모하는 기능 (데이터 내보내기 등) 이 포함된다.

여기서 핵심 보호 장치는 JavaScript 의 동일 출처 정책과 쿠키의 SameSite 로직과 같은 브라우저 보안 정책이다. 또 다른 일반적인 보호 장치는 CORS preflight 메커니즘이다. 이 메커니즘은 다른 출처에서 호출되도록 설계된 엔드포인트에 치명적이지만, 다른 출처에서 호출되도록 설계되지 않은 엔드포인트에 대한 요청 위조 방지 메커니즘으로 유용하게 사용된다.

#	설명	레벨
3.5.1	애플리케이션이 허용되지 않은 교차 출처 요청을 통한 민감한 기능의 사용을 막기 위해 CORS preflight 메커니즘에 의존하지 않는지 확인하고, 이런 요청이 애플리케이션 자체에서 시작된 건지 증명해야 한다. 이는 anti-forgery 토큰의 사용과 증명이나 CORS-safelisted 요청 헤더 필드가 아닌 확장 HTTP 헤더 필드를 요구하는 것으로 가능하다. 사이트 간 요청 위조 공격 (CSRF) 으로 알려진 브라우저 기반의 요청 위조 공격에 대한 방어다.	1
3.5.2	애플리케이션이 민감한 기능의 허용되지 않은 교차 출처 사용을 막기 위해 CORS preflight 메커니즘에 의존하고 있다면, CORS preflight 요청을 작동하지 않는 요청을 호출할 수 있는지 확인해야 한다. 이는‘Origin’과’Content-Type’요청 헤더 필드 값을 확인하거나 CORS-safelisted 헤더 필드가 아닌 확장 필드 사용에 대한 확인이 필요하다.	1

#	설명	레벨
3.5.3	민감한 기능이 HTTP 요청으로 POST, PUT, PATCH, DELETE 와 같은 적절한 HTTP 방식을 사용하는지, HEAD, OPTIONS, 또는 GET 처럼 HTTP 사양이“안전”으로 정의되지 않는 방식을 사용하는지 확인한다. 또는 Sec-Fetch-* 요청 헤더 필드의 엄격한 검증을 사용하여 요청이 부적절한 교차 출처 호출, 탐색 요청 또는 이미지 소스와 같은 리소스 로드에서 발생하지 않았는지 확인할 수 있다.	1
3.5.4	서로 다른 호스트 이름에 별도의 애플리케이션이 호스팅되어 있는지 확인하여 동일한 출처 정책에서 제공하는 제한 사항을 활용한다. 여기에는 한 출처에서 불러온 문서나 스크립트가 다른 출처의 리소스와 상호 작용할 수 있는 방법과 쿠키에서의 호스트 이름 기반 제한 사항이 포함된다.	2
3.5.5	postMessage 인터페이스를 통해 수신한 메시지의 출처를 신뢰할 수 없거나, 구문이 유효하지 않은 경우, 수신한 메시지가 폐기되는지 확인한다.	2
3.5.6	Cross-Site Script Inclusion (XSSI) 공격을 피하기 위해, 애플리케이션 아무 곳에서나 JSONP 기능이 활성화되지 않는지 확인한다.	3
3.5.7	Cross-Site Script Inclusion (XSSI) 공격을 방지하기 위해 JavaScript 파일과 같은 스크립트 리소스 응답에 승인이 필요한 데이터가 포함되지 않았는지 확인한다.	3
3.5.8	인증된 리소스 (예: 이미지, 비디오, 스크립트 및 기타 문서) 가 의도된 경우에만 사용자를 대신하여 불러와지거나 내장될 수 있는지 확인한다. 이는 Sec-Fetch-* HTTP 요청 헤더 필드를 엄격하게 검증하여 요청이 부적절한 교차 출처 호출에서 발생하지 않았는지 확인하거나 브라우저에 반환된 콘텐츠를 차단하도록 지시하는 제한적인 교차 출처 리소스 정책 HTTP 응답 헤더 필드를 설정하여 달성할 수 있다.	3

V3.6 외부 리소스 무결성

이 섹션은 외부 사이트에서 콘텐츠를 안전하게 호스팅하기 위한 지침을 제공한다.

#	설명	레벨
3.6.1	클라이언트 측 자산 (예: JavaScript 라이브러리, CSS 또는 웹 폰트) 이 정적이고 버전이 지정된 경우에만 외부 (예: 콘텐츠 전송 네트워크) 로만 호스팅되는지 확인하고, 하위 리소스 무결성 (SRI) 을 사용하여 자산의 무결성을 검증한다. 이것이 불가능하다면 각 리소스에 대해 이를 정당화하기 위해 문서화된 보안 결정이 필요하다.	3

V3.7 다른 브라우저 보안 고려사항

이 섹션은 클라이언트 측 브라우저 보안을 위해 요구되는 다양한 보안 통제와 현대 브라우저 보안 특징들을 포함한다.

#	설명	레벨
3.7.1	애플리케이션이 여전히 지원되고 안전하다고 간주되는 클라이언트 측 기술만 사용하는지 확인한다. 이 요구 사항을 충족하지 않는 기술의 예로는 NSAPI 플러그인, Flash, Shockwave, ActiveX, Silverlight, NACL 또는 클라이언트 측 Java applets 가 있다.	2
3.7.2	애플리케이션이 사용자를 통제하지 못하는 다른 호스트 이름이나 도메인으로 자동으로 리다이렉션할 때, 허용된 목적지로만 리다이렉션 하는지 확인한다.	2
3.7.3	사용자가 애플리케이션의 통제 범위를 벗어난 URL 로 리다이렉션 될 때 이를 취소할 수 있는 옵션이 있는 알림이 애플리케이션에 표시되는지 확인한다.	3
3.7.4	애플리케이션의 최상위 도메인 (예: site.tld) 가 공공 HSTS preload 목록에 추가되었는지 확인한다. 이렇게 하면 애플리케이션에 TLS 를 사용하는 것이 Strict-Transport-Security 응답 헤더 필드에만 의존하지 않고 메인 브라우저에 직접 내장된다.	3
3.7.5	애플리케이션에 접근하는 브라우저가 예상한 보안 특징을 가지고 있지 않은 경우, 애플리케이션이 문서에 명시된 대로 동작하는지 확인한다 (유저에게 경고하거나 접근을 차단하는 등).	3

참조

더 많은 정보는 다음을 참조한다.

- Set-Cookie __Host- prefix details
- OWASP Content Security Policy Cheat Sheet
- OWASP Secure Headers Project
- OWASP Cross-Site Request Forgery Prevention Cheat Sheet
- HSTS Browser Preload List submission form
- OWASP DOM Clobbering Prevention Cheat Sheet

V4 API 와 WEB 서비스

제어 목표

웹 브라우저나 다른 클리어언트에서 사용하기 위해서 API 를 노출하는 애플리케이션 (일반적으로 JSON, XML, GraphQL 사용) 에는 몇 가지 특별히 고려해야 할 사항이 적용된다. 이번 장에서는 관련 보안 설정과 적용해야할 매커니즘에 대해 다룬다.

다른 장에서 다루는 인증, 세션 관리, 입력값 검증과 관련된 문제들은 API 에도 동일하게 적용되므로, 이 챕터의 내용을 전체 맥락에서 벗어나 개별적으로 테스트해서는 안 된다는 점에 유의해야 한다.

V4.1 일반적인 웹 서비스 보안

이 섹션은 일반적인 웹 서비스 보안 고려사항과 기본적인 웹 서비스 보안 수칙에 대해서 다룬다.

#	설명	레벨
4.1.1	메세지 본문이 포함된 모든 HTTP 응답에는 실제 내용과 일치하는 Content-Type 헤더 필드가 포함되어 있는지 검증한다. 이때 IANA 미디어 타입 (text, /+xml, /xml 등) 에 따라 안전한 문자 인코딩 (ex. UTF-8, ISO-8859-1) 을 지정하는 charset 파라미터도 포함되어야 한다.	1
4.1.2	사용자가 대면하는 엔드포인트 (사람이 직접 웹 브라우저에 접속하는 경우) 만 HTTP 에서 HTTPS 로 자동 리디렉션하고, 그 외에 서비스나 API 엔드포인트는 명백한 리디렉션을 구현하지 않았는지 검증한다. 이는 클라이언트가 실수로 암호화되지 않은 HTTP 요청을 보내고 있음에도, 요청이 자동으로 HTTPS 로 리디렉션되어 민감한 데이터의 유출 사실을 발견하지 못하게 되는 상황을 방지하기 위함이다.	2
4.1.3	로드 밸런서, 웹 프록시, 백엔드-프론트엔드 서비스 등 중간 계층에서 설정되며, 애플리케이션에서 사용하는 모든 HTTP 헤더 필드를 사용자가 재정의할 수 없는지 확인한다. 예를 들어 X-Real-IP, X-Frowarded-*, X-User-ID 등의 헤더가 포함될 수 있다.	2
4.1.4	애플리케이션이나 API 에서 명시적으로 지원하는 HTTP 메서드 (프리플라이트 요청 시에 OPTIONS 포함) 만 사용할 수 있으며, 사용되지 않는 메서드는 차단되어 있는지 확인한다.	3
4.1.5	매우 민감하거나 여러 시스템을 거치는 요청이나 거래에 대한 전송 계층 보호에 더하여 메세지별 디지털 서명들을 사용함으로써 추가적인 보증을 제공하는지 확인한다.	3

V4.2 HTTP 메세지 구조 검증

이번 섹션은 지나치게 긴 HTTP 메세지를 통한 요청 smuggling, 응답 분할, 헤더 인젝션, 그리고 DOS 와 같은 공격을 방지하기 위해 HTTP 메세지의 구조와 헤더 필드를 어떻게 검증해야 하는지를 설명한다.

이러한 요구 사항은 일반적인 HTTP 메세지 처리 및 생성과 관련이 있지만, 서로 다른 HTTP 버전 간에 HTTP 메세지를 변환할 때 특히 중요하다.

#	설명	레벨
4.2.1	모든 애플리케이션 구성 요소 (로드 밸런서, 방화벽, 애플리케이션 서버 포함) 가 HTTP 요청 smuggling 을 예방하는 HTTP 버전의 적합한 메커니즘을 사용하는지에 대해, 수신되는 HTTP 메세지의 끝을 결정하는지 확인한다. HTTP/1.x 에서 Transfer-encoding 헤더 필드가 있는 경우에 RFC 2616 에 따라서 Content-Length 헤더는 무시해야 한다. HTTP/2 또는 HTTP/3 를 사용할 때 Content-Length 헤더 필드가 있는 경우에 수신기는 DATA 프레임의 길이와 일치하는지 확인해야 한다.	2
4.2.2	HTTP 메세지를 생성할 때, 요청 smuggling 공격을 예방하기 위해서 HTTP 프로토콜의 프레이밍 방식에 의해 결정된 Content 의 길이와 Content-Length 헤더 필드가 충돌하지 않는지 확인한다.	3

#	설명	레벨
4.2.3	응답 분할 및 헤더 인젝션 공격을 예방하기 위해서 애플리케이션이 Transfer-encoding 과 같은 connection-specific 헤더 필드를 가진 HTTP/2 또는 HTTP/3 메세지를 전송하거나 수락하지 않는지 확인한다.	3
4.2.4	헤더 인젝션 공격을 예방하기 위해서, 애플리케이션이 헤더 이름과 값에 CR(\r), LF(\n), CRLF(\r\n) 과 같은 줄바꿈 문자가 포함되어 있지 않은 HTTP/2 및 HTTP/3 요청만 허용하는지 확인한다.	3
4.2.5	애플리케이션 (백엔드 또는 프론트엔드) 이 요청을 빌드하고 전송하는 경우, 수신 구성 요소가 수학하기에는 너무 긴 URI(ex. API 호출) 또는 HTTP 요청 헤더 필드 (ex. Authorization 또는 Cookie) 를 생성하지 않도록 검증, 검열 또는 기타 메커니즘을 사용하는지 확인한다. 이로 인해서 지나치게 긴 요청 (ex. 긴 쿠키 헤더 필드) 을 보낼 때와 같이 서비스 거부가 발생하여 서버가 항상 오류 상태로 응답할 수 있다.	3

V4.3 GraphQL

GraphQL 은 다양한 백엔드 서비스와 긴밀하게 연결되지 않은 데이터가 풍부한 클라이언트를 만드는 방법으로, 점점 일반화 되고 있다. 이 섹션에서는 GraphQL 의 보안 고려 사항을 다룬다.

#	설명	레벨
4.3.1	쿼리 허용 목록, 깊이 제한, 양 제한 또는 쿼리 비용 분석을 사용하여 값비싼 중첩 쿼리로 인해 GraphQL 또는 데이터 계층 표현식 거부 (DOS) 를 방지하는지 확인한다.	2
4.3.2	GraphQL API 가 다른 사용자에 의해 사용되지 않는 한, 운영 환경에서 GraphQL 점검 쿼리들이 비비활성화되어 있는지 확인한다.	2

V4.4 WebSocket

WebSocket 은 단일 TCP 연결을 통해 동시에 양방향 통신 채널을 제공하는 통신 프로토콜이다. 2011 년에 IETF 에 의해 RFC 6455 로 표준화되었으며, HTTP 포트 443 및 80 에서 동작하도록 설계되었음에도 불구하고 HTTP 와는 다르다.

이 섹션에서는 실시간 통신 채널을 악용하는 통신 보안 및 세션 관리와 관련된 공격을 방지하기 위한 주요 보안 요구 사항을 제공한다.

#	설명	레벨
4.4.1	모든 WebSocket 연결에 WebSocket over TLS(WSS) 이 사용되는지 확인한다.	1
4.4.2	초기 HTTP WebSocket Handshake 중에 기존 헤더 필드가 애플리케이션에 허용된 기존 목록과 대조되는지 확인한다.	2

#	설명	레벨
4.4.3	애플리케이션의 표준 세션 관리를 사용할 수 없는 경우, 관련 세션 관리 보안 요구 사항을 준수하는 전용 토큰이 사용되고 있는지 확인한다.	2
4.4.4	기존 HTTPS 세션을 WebSocket 채널로 전환할 때 이전에 인증된 HTTPS 세션을 통해 전용 WebSocket 세션 관리 토큰을 처음 얻거나 검증하는지 확인한다.	2

참조

자세한 내용은 다음을 참고한다:

- OWASP REST Security Cheat Sheet
- Resources on GraphQL Authorization from graphql.org and Apollo.
- OWASP Web Security Testing Guide: GraphQL Testing
- OWASP Web Security Testing Guide: Testing WebSockets

V5 파일 처리

제어 목표

파일을 사용하는 것은 서비스 거부, 무단 접근 및 저장 공간 고갈을 포함하여 애플리케이션에 다양한 위험을 초래할 수 있다. 이 장은 이러한 위험을 해결하기 위한 요구사항을 포함한다.

V5.1 파일 처리 문서화

이 섹션은 관련 보안 검사를 개발하고 검증하기 위한 필수 전제 조건으로 애플리케이션이 허용하는 파일의 예상 특성을 문서화하는 요구사항을 포함한다.

#	설명	레벨
5.1.1	업로드 기능별로 허용되는 파일 형식, 예상되는 파일 확장자, 그리고 최대 크기 (압축 해제된 크기 포함) 가 문서에 정의되어 있는지 검증해야 한다. 또한, 악성 파일이 탐지될 때 애플리케이션이 어떻게 동작하는지와 같이, 최종 사용자가 파일을 안전하게 다운로드하고 처리할 수 있도록 하는 방법이 문서에 명시되어 있는지 확인해야 한다.	2

V5.2 파일 업로드 및 콘텐츠

파일 업로드 기능은 신뢰할 수 없는 파일의 주요 소스이다. 이 섹션은 이러한 파일의 존재, 볼륨 또는 내용이 애플리케이션에 해를 끼치지 않도록 보장하기 위한 요구사항을 설명한다.

#	설명	레벨
5.2.1	애플리케이션이 성능 저하나 서비스 거부 공격 (Denial of Service; DoS) 을 유발하지 않고 처리할 수 있는 크기의 파일만 허용하는지 검증해야 한다.	1
5.2.2	애플리케이션이 파일 자체 또는 zip 파일과 같은 아카이브 내에서 파일을 허용할 때, 파일 확장자가 예상 파일 확장자와 일치하는지 확인하고 내용이 확장자로 표현된 유형과 일치하는지 확인하는지 검증해야 한다. 여기에는 초기'매직 바이트 (magic bytes)'확인, 이미지 재작성 수행, 파일 내용 유효성 검사를 위한 전문 라이브러리 사용이 포함되지만 이에 국한되지 않는다. L1의 경우, 특정 비즈니스 또는 보안 결정을 내리는 데 사용되는 파일에만 집중할 수 있다. L2 이상에서는 허용되는 모든 파일에 적용되어야 한다	1
5.2.3	애플리케이션이 압축 파일 (예: zip, gz, docx, odt) 을 압축 해제하기 전에 허용되는 최대 압축 해제 크기와 최대 파일 수를 확인하는지 검증해야 한다.	2
5.2.4	단일 사용자가 너무 많은 파일이나 과도하게 큰 파일로 저장 공간을 채우지 않도록 사용자별 파일 크기 할당량 (quota) 및 최대 파일 수가 적용되는지 검증해야 한다.	3
5.2.5	애플리케이션이 심볼릭 링크 (symlink) 를 포함하는 압축 파일 업로드를 허용하지 않는지 검증해야 한다. 단, 이러한 기능이 특별히 요구되는 경우에는 심볼릭 링크할 수 있는 파일의 허용 목록 (allowlist) 을 강제하는 것이 필요하다.	3
5.2.6	픽셀 플러드 공격 (Pixel Flood Attack) 을 방지하기 위해, 애플리케이션이 업로드된 이미지의 픽셀 크기가 허용된 최대값보다 큰 경우 해당 이미지를 거부하는지 검증해야 한다.	3

V5.3 파일 저장

이 섹션은 업로드 후 파일이 부적절하게 실행되는 것을 방지하고, 위험한 콘텐츠를 탐지하며, 신뢰할 수 없는 데이터가 파일 저장 위치를 제어하는 데 사용되지 않도록 하기 위한 요구사항을 포함한다.

#	설명	레벨
5.3.1	신뢰할 수 없는 입력으로 업로드되거나 생성되어 공개 폴더에 저장된 파일이 HTTP 요청으로 직접 접근될 때 서버 측 프로그램 코드로 실행되지 않는지 검증해야 한다.	1
5.3.2	애플리케이션이 파일 작업을 위한 파일 경로를 생성할 때, 사용자 제출 파일명 대신 내부적으로 생성되거나 신뢰할 수 있는 데이터를 사용하는지 검증해야 한다. 만약 사용자 제출 파일명 또는 파일 메타데이터를 반드시 사용해야 한다면, 경로 탐색 (path traversal), 로컬 또는 원격 파일 포함 (Local File Inclusion; LFI, Remote File Inclusion; RFI), 그리고 서버 측 요청 위조 (Server-side Request Forgery; SSRF) 공격으로부터 보호하기 위해 엄격한 유효성 검사 및 정제 (Sanitization) 가 적용되는지 확인해야 한다.	1
5.3.3	파일 압축 해제와 같은 서버 측 파일 처리가 zip slip 과 같은 취약점을 방지하기 위해 사용자 제공 경로 정보를 무시하는지 검증해야 한다.	3

V5.4 파일 다운로드

이 섹션은 경로 탐색 및 삽입 공격을 포함하여 파일을 다운로드할 때 위험을 완화하기 위한 요구사항을 포함한다. 또한 위험한 콘텐츠를 포함하지 않도록 하는 것도 포함된다.

#	설명	레벨
5.4.1	애플리케이션이 JSON, JSONP 또는 URL 파라미터를 포함한 사용자 제출 파일명을 검증하거나 무시하는지 검증해야 하고, 응답의 Content-Disposition 헤더 필드에 파일명을 명시하는지 검증해야 한다.	2
5.4.2	제공되는 파일 이름 (예: HTTP 응답 헤더 필드 또는 이메일 첨부 파일) 이 문서 구조를 보존하고 삽입 공격 (Injection Attack) 을 방지하기 위해 인코딩되거나 정제되는지 검증해야 한다. (예: RFC 6266 준수)	2
5.4.3	신뢰할 수 없는 소스에서 얻은 파일이 알려진 악성 콘텐츠 제공을 방지하기 위해 안티바이러스 스캐너에 의해 검사되는지 검증해야 한다.	2

참조

더 많은 정보는 다음을 참고한다:

- OWASP File Upload Cheat Sheet
- Example of using symlinks for arbitrary file read
- Explanation of “Magic Bytes”from Wikipedia

V6 인증

제어 목표

인증은 개인이나 장치의 진위를 확립하거나 확인하는 과정이다. 이는 사용자가 주장하는 신원을 검증하고, 위장 행위 (impersonation) 에 대한 저항성을 보장하며, 비밀번호의 복구나 가로채기를 방지하는 것을 포함한다.

NIST SP 800-63은 전 세계 조직에 유용한 현대적이고 증거 기반의 표준이나, 특히 미국 정부 기관 및 그와 상호작용하는 조직에 매우 관련성이 높다.

이 장의 요구사항 중 많은 것은 표준의 두 번째 섹션 (NIST SP 800-63B “디지털 신원 지침 - 인증 및 수명 주기 관리”) 을 기반으로 하지만, 본 장은 보편적인 위협과 자주 악용되는 인증 취약점에 초점을 맞춘다. 또한 표준의 모든 항목을 포괄적으로 다루지는 않는다. 완전한 NIST SP 800-63 준수가 필요한 경우 반드시 원문을 참고해야 한다.

추가적으로, NIST SP 800-63 의 용어와는 일부 경우 다를 수 있으며, 본 장은 명확성을 높이기 위해 더 일반적으로 이해되는 용어를 사용한다.

보다 발전된 애플리케이션의 일반적 기능 중 하나는 다양한 위험 요인에 따라 요구되는 인증 단계를 조정할 수 있는 능력이다. 이러한 기능은 권한 부여 결정에도 고려되어야 하므로“권한 부여”장에서 다룬다.

V6.1 Authentication Documentation

본 섹션은 애플리케이션에서 유지해야 하는 인증 문서화에 대한 요구사항을 포함한다. 이는 관련 인증 제어가 어떻게 구성되어야 하는지를 구현하고 평가하는 데 핵심적이다.

#	설명	레벨
6.1.1	인증 문서가 크리덴셜 스터핑, 비밀번호 무차별 대입 공격 등을 방어하기 위해 속도 제한, 자동화 방지, 적응형 응답 (adaptive response) 과 같은 제어가 어떻게 사용되는지를 정의한다. 또한, 이러한 제어가 어떻게 구성되는지와 악의적으로 계정을 잠그는 것을 방지하는 방법을 명확히 하는지 검증한다.	1
6.1.2	조직명, 제품명, 시스템 식별자, 프로젝트 코드명, 부서명 또는 역할명 등 문맥적으로 특정된 단어의 변형을 포함하여, 비밀번호에 사용되지 않도록 문서화된 단어 목록이 있는지 검증한다.	2
6.1.3	애플리케이션이 여러 인증 경로를 포함하는 경우, 이들 모두가 문서화되어 있으며 각 경로에 대해 일관되게 적용되는 보안 제어와 인증 강도가 정의되어 있는지 검증한다.	2

V6.2 비밀번호 보안

비밀번호는 NIST SP 800-63 에서“기억된 비밀 (Memorized Secret)”로 불리며, 비밀번호, 암호구절 (passphrase), PIN, 잠금 패턴, 특정 이미지 요소 선택 등을 포함한다. 일반적으로“알고 있는 것 (something you know)”으로 간주되며, 단일 요소 인증 방식으로 자주 사용된다.

이 섹션은 비밀번호가 안전하게 생성되고 처리되도록 하기 위한 요구사항을 포함한다. 대부분의 요구사항은 L1 에 속하는데, 이는 해당 수준에서 가장 중요하기 때문이다. L2 이상부터는 다중요소 인증 메커니즘이 요구되며, 이때 비밀번호는 그 요소 중 하나로 사용될 수 있다.

본 섹션의 요구사항은 주로 NIST 가이드라인의 § 5.1.1.2와 관련된다.

#	설명	레벨
6.2.1	사용자가 설정하는 비밀번호가 최소 8 자 이상이어야 하며, 최소 15 자 이상을 강력히 권장하는지 검증한다.	1
6.2.2	사용자가 비밀번호를 변경할 수 있는지 검증한다.	1
6.2.3	비밀번호 변경 기능이 사용자의 현재 비밀번호와 새 비밀번호를 요구하는지 검증한다.	1
6.2.4	계정 등록 또는 비밀번호 변경 시 제출된 비밀번호가 최소 3000 개의 상위 비밀번호 목록과 비교 및 검사되는지, 이때 애플리케이션 비밀번호 정책 (예: 최소 길이) 을 만족하는지 검증한다.	1
6.2.5	비밀번호는 어떤 구성도 허용되며, 허용되는 문자 유형에 대한 제한이 없는지 검증한다. 대/소문자, 숫자, 특수문자에 대한 최소 개수 요구사항이 없어야 한다.	1
6.2.6	비밀번호 입력 필드가 type=password 를 사용하여 입력을 마스킹하는지 검증한다. 애플리케이션은 사용자가 전체 비밀번호나 마지막 입력 문자를 일시적으로 볼 수 있도록 허용할 수 있다.	1

#	설명	레벨
6.2.7	붙여넣기 기능, 브라우저 비밀번호 도우미, 외부 비밀번호 관리자의 사용이 허용되는지 검증한다.	1
6.2.8	애플리케이션이 비밀번호를 사용자로부터 수신한 그대로 검증하며, 대소문자 변경이나 잘림 (truncation) 과 같은 수정 없이 처리하는지 검증한다.	1
6.2.9	비밀번호가 최소 64 자까지 허용되는지 검증한다.	2
6.2.10	사용자의 비밀번호는 손상되었음이 발견되거나 사용자가 교체할 때까지 유효하며, 애플리케이션이 주기적인 자격 증명 교체를 요구하지 않는지 검증한다.	2
6.2.11	문서화된 문맥별 단어 목록을 사용하여 추측하기 쉬운 비밀번호가 생성되지 않도록 하는지 검증한다.	2
6.2.12	계정 등록 또는 비밀번호 변경 시 제출된 비밀번호가 유출된 비밀번호 집합과 비교 및 검사되는지 검증한다.	2

V6.3 일반 인증 보안

이 섹션은 인증 메커니즘의 보안에 대한 일반 요구사항을 포함하며, 레벨별로 상이한 기대치를 제시한다. L2 애플리케이션은 다중요소 인증 (MFA) 을 반드시 사용해야 한다. L3 애플리케이션은 반드시 검증된 신뢰 실행 환경 (Trusted Execution Environment, TEE) 에서 수행되는 하드웨어 기반 인증을 사용해야 한다. 여기에는 장치에 종속된 패스키 (device-bound passkeys), eIDAS 높은 수준 (Level of Assurance, LoA High) 의 강제 인증기, NIST 인증기 보증 수준 3(Authenticator Assurance Level 3, AAL3) 을 충족하는 인증기, 또는 이에 상응하는 메커니즘이 포함될 수 있다.

이는 비교적 강력한 MFA 요구이지만, 사용자를 보호하기 위해 이에 대한 인증 수준을 높이는 것은 필수적이다. 이러한 요구사항을 완화하려는 모든 시도는, 인증과 관련된 위험을 어떻게 완화할 것인지에 대한 명확한 계획을 반드시 수반해야 하며, 이때 NIST 의 지침과 해당 주제에 대한 연구 결과를 고려해야 한다.

출시 시점 기준으로, NIST SP 800-63 은 이메일을 인증 메커니즘 (아카이브) 으로 허용하지 않는다는 점에 유의해야 한다.

이 섹션의 요구사항은 NIST 가이드라인의 다양한 절과 관련이 있으며, 여기에는 § 4.2.1, § 4.3.1, § 5.2.2, and § 6.1.2 등이 포함된다.

#	설명	레벨
6.3.1	애플리케이션의 보안 문서에 따라 크리덴셜 스터핑 및 비밀번호 무차별 대입 공격을 방지하기 위한 제어가 구현되어 있는지 검증한다.	1
6.3.2	“root”, “admin”, “sa”와 같은 기본 사용자 계정이 애플리케이션에 존재하지 않거나 비활성화되어 있는지 검증한다.	1

#	설명	레벨
6.3.3	애플리케이션 접근 시 반드시 MFA 또는 단일 요소 인증 메커니즘의 조합을 사용하는지 검증한다. L3 의 경우, 인증을 수행하려는 사용자의 의도를 검증하기 위해 사용자가 직접 수행하는 동작 (예: FIDO 하드웨어 키 또는 모바일 기기에서 버튼을 누르는 행위) 을 요구함으로써, 피싱 공격에 대한 침해 및 위장 저항성을 제공하는 하드웨어 기반 인증 메커니즘이 반드시 하나의 요소로 포함되어야 한다. 이 요구사항에서 고려 사항을 완화하려는 경우, 완전히 문서화된 근거와 포괄적인 보완 통제가 반드시 수반되어야 한다.	2
6.3.4	여러 인증 경로가 존재하는 경우, 문서화되지 않은 경로가 없어야 하며 보안 제어와 인증 강도가 일관되게 적용되는지 검증한다.	2
6.3.5	사용자가 의심스러운 인증 시도에 대해 성공 여부와 관계 없이 알림을 제공받는지 검증한다. 여기에는 비정상적인 위치나 클라이언트에서의 인증 시도, 부분적으로만 성공한 인증 (여러 요소 중 하나만 성공한 경우), 장기간 비활성 상태 이후의 인증 시도, 또는 여러 번의 인증 실패 후 성공한 인증이 포함될 수 있다.	3
6.3.6	이메일이 단일 또는 다중 요소 인증 메커니즘으로 사용되지 않는지 검증한다.	3
6.3.7	자격 증명 재설정, 사용자명/이메일 수정 등 인증 관련 정보 변경 이후 사용자에게 통지되는지 검증한다.	3
6.3.8	오류 메시지, HTTP 응답 코드, 응답 시간 차이 등을 통해 유효한 사용자를 유추할 수 없으며, 비밀번호 등록/찾기 기능에도 동일한 보호가 적용되는지 검증한다.	3

V6.4 인증 요소 수명 주기 및 복구

인증 요소는 비밀번호, 소프트 토큰, 하드웨어 토큰, 생체 인식 장치 등을 포함할 수 있다. 이들의 수명주기를 안전하게 처리하는 것은 애플리케이션 보안에 매우 중요하며, 본 섹션은 관련 요구사항을 다룬다.

본 섹션의 요구사항은 주로 NIST 가이드라인의 § 5.1.1.2, § 6.1.2.3과 관련된다.

#	설명	레벨
6.4.1	시스템이 생성하는 초기 비밀번호나 활성화 코드가 안전하게 난수로 생성되며, 기존 비밀번호 정책을 따르고, 짧은 기간 내 또는 최초 사용 시 만료되는지 검증한다. 초기 비밀번호는 장기 비밀번호로 허용되지 않아야 한다.	1
6.4.2	비밀번호 힌트나 지식 기반 인증 (“비밀 질문”) 이 존재하지 않는지 검증한다.	1
6.4.3	분실된 비밀번호에 대한 재설정 절차가 안전하게 구현되며, MFA 를 우회하지 않는지 검증한다.	2
6.4.4	MFA 요소 분실 시, 등록 시와 동일 수준의 신원 증명이 수행되는지 검증한다.	2
6.4.5	만료 예정인 인증 메커니즘에 대한 갱신 안내가 충분한 시간 전에 발송되며, 필요한 경우 자동 알림이 구성되는지 검증한다	3

#	설명	레벨
6.4.6	관리자가 사용자 비밀번호 재설정 절차를 시작할 수 있으나, 사용자의 비밀번호를 직접 변경하거나 지정할 수 없는지 검증한다. 이는 관리자가 사용자의 비밀번호를 알지 못하도록 한다.	3

V6.5 General Multi-factor authentication requirements

이 섹션은 다양한 다중요소 인증 방식에 적용될 수 있는 일반적인 지침을 제공한다.

여기에는 다음 메커니즘이 포함된다:

- 룩업 시크릿 (Lookup Secrets)
- 시간 기반 일회용 비밀번호 (Time based One-time Passwords, TOTPs)
- 대역 외 메커니즘 (Out-of-Band mechanisms)

룩업 시크릿은 미리 생성된 비밀 코드 목록으로, 거래 승인 번호 (Transaction Authorization Numbers, TAN), 소셜 미디어 복구 코드, 또는 무작위 값이 들어 있는 격자 (grid) 와 유사하다. 이러한 인증 메커니즘은 의도적으로 기억하기 어려워, 어딘가에 저장해야 하므로“소유하고 있는 것 (something you have)”으로 간주된다.

시간 기반 일회용 비밀번호 (TOTPs) 는 지속적으로 변경되는 의사난수 기반의 일회용 챌린지를 표시하는 물리적 또는 소프트 토큰이다. 이러한 인증 메커니즘 역시“소유하고 있는 것 (something you have)”으로 간주된다. 다중요소 TOTPs 는 단일요소 TOTPs 와 유사하지만, 최종 일회용 비밀번호 (OTP) 를 생성하기 위해 유효한 PIN 코드, 생체 인식 해제, USB 삽입 또는 NFC 페어링, 혹은 거래 서명 계산기와 같은 추가 값을 입력해야 한다.

대역 외 메커니즘 (Out-of-Band mechanisms) 의 세부 사항은 다음 섹션에서 제공된다.

본 섹션의 요구사항은 주로 NIST 가이드라인§ 5.1.2의 § 5.1.3, § 5.1.4.2, § 5.1.5.2, § 5.2.1, 그리고 § 5.2.3와 관련된다.

#	설명	레벨
6.5.1	조회 secret, 대역 외 인증 요청 및 코드, 시간 기반 일회용 비밀번호가 단 한 번만 성공적으로 사용 가능한지 검증한다.	2
6.5.2	애플리케이션 백엔드에 저장될 때, 엔트로피가 112 비트 미만인 룩업 시크릿 (예: 무작위 영숫자 19 자 또는 무작위 숫자 34 자) 은 32 비트 무작위 솔트 (salt) 를 포함하는 사전 승인된 비밀번호 저장용 해시 알고리즘으로 해시되는지 검증한다. 룩업 시크릿이 112 비트 이상의 엔트로피를 가진 경우에는 표준 해시 함수를 사용할 수 있다.	2
6.5.3	룩업 시크릿, 대역 외 인증 코드, 그리고 시간 기반 일회용 비밀번호의 시드가 예측 가능한 값을 피하기 위해 암호학적으로 안전한 의사난수 생성기 (CSPRNG) 를 사용하여 생성되는지 검증한다.	2
6.5.4	룩업 시크릿과 대역 외 인증 코드가 최소 20 비트의 엔트로피를 가지는지 검증한다. 일반적으로 무작위 영숫자 4 자 또는 무작위 숫자 6 자가 충분하다.	2

#	설명	레벨
6.5.5	대역 외 인증 요청, 코드 또는 토큰과 시간 기반 일회용 비밀번호가 정의된 수명을 가지는지 검증한다. 대역 외 요청의 최대 수명은 10 분이어야 하며, TOTP 의 최대 수명은 30 초이어야 한다.	2
6.5.6	도난이나 기타 손실의 경우 모든 인증 요소 (물리적 장치 포함) 를 해지할 수 있는지 검증한다.	3
6.5.7	생체 인식 인증 메커니즘은 반드시 보조 요소로만 사용되어야 하며, “소유하고 있는 것”또는“알고 있는 것”중 하나와 함께 사용되는지 검증한다.	3
6.5.8	시간 기반 일회용 비밀번호는 신뢰할 수 있는 서비스의 시간 소스를 기준으로 검증되어야 하며, 신뢰할 수 없거나 클라이언트에서 제공한 시간을 기반으로 해서는 안 된다.	3

V6.6 대역 외 (Out-of-Band) 인증 메커니즘

이는 보통 인증 서버가 보안이 보장된 보조 채널을 통해 물리적 장치와 통신하는 방식을 포함한다. 예를 들어, 모바일 기기에 푸시 알림을 전송하는 경우가 있다. 이러한 유형의 인증 메커니즘은“소유하고 있는 것”으로 간주된다.

전자메일이나 VOIP 와 같은 안전하지 않은 대역 외 인증 메커니즘은 허용되지 않는다. PSTN 과 SMS 인증은 현재 NIST 에 의해 “제한된”인증 메커니즘으로 간주되며, 시간 기반 일회용 비밀번호나 암호학적 메커니즘 또는 유사한 방식으로 대체되어야 한다. NIST SP 800-63B 의 § 5.1.3.3에서는 전화나 SMS 기반 대역 외 인증을 반드시 지원해야 하는 경우, 디바이스 교체, SIM 변경, 번호 이동 또는 기타 비정상적 행위의 위험을 다룰 것을 권장한다. 본 섹션은 이를 필수 요구사항으로 명시하지는 않지만, 민감한 L2 애플리케이션이나 L3 애플리케이션에서 이러한 예방 조치를 취하지 않는 것은 심각한 위험 신호로 간주되어야 한다.

또한 NIST 는 최근 푸시 알림 사용을 권장하지 않는 지침을 제공했다. 이 ASVS 절은 이를 요구사항으로 규정하지 않지만, “푸시 폭탄 (push bombing)”공격 위험을 인지하는 것이 중요하다.

#	설명	레벨
6.6.1	공중 교환 전화망 (PSTN) 을 사용하여 전화나 SMS 를 통해 일회용 비밀번호 (OTPs) 를 전달하는 인증 메커니즘은, 해당 전화번호가 사전에 검증되었을 때만 제공되어야 하며, 더 강력한 대체 방법 (예: 시간 기반 일회용 비밀번호) 도 제공되어야 하고, 서비스는 사용자에게 이러한 보안 위험에 대한 정보를 제공해야 한다. L3 애플리케이션의 경우, 전화와 SMS 는 옵션으로 제공되어서는 안 된다.	2
6.6.2	대역 외 인증 요청, 코드 또는 토큰은 생성된 원래 인증 요청에만 바인딩되어야 하며, 이전이나 이후의 요청에는 사용할 수 없어야 한다.	2
6.6.3	코드 기반 대역 외 인증 메커니즘이 속도 제한 등을 통해 무차별 대입 공격으로부터 보호되는지 검증한다. 또한 최소 64 비트 이상의 엔트로피를 가진 코드를 사용하는 것도 고려해야한다.	2
6.6.4	다중요소 인증에 푸시 알림이 사용되는 경우, 푸시 폭탄 (push bombing) 공격을 방지하기 위해 속도 제한을 적용해야 한다. 또한 숫자 일치 기법 역시 이 위험을 완화할 수 있다.	3

V6.7 암호학적 인증 메커니즘

암호학적 인증 메커니즘에는 스마트카드나 FIDO 키가 포함되며, 사용자가 인증을 완료하기 위해 암호 장치를 컴퓨터에 연결하거나 페어링해야 한다. 인증 서버는 챌린지 논스 (challenge nonce) 를 암호 장치나 소프트웨어로 전송하며, 해당 장치나 소프트웨어는 안전하게 저장된 암호 키를 기반으로 응답을 계산한다. 본 섹션의 요구사항은 이러한 메커니즘에 대한 구현별 지침을 제공하며, 암호 알고리즘에 대한 지침은“암호학 (Cryptography)”장에서 다룬다.

암호학적 인증에 공유 키나 비밀 키가 사용되는 경우, 이는“구성 (Configuration)”장의“비밀 관리 (Secret Management)”섹션에 문서화된 것과 동일한 메커니즘을 사용하여 저장되어야 한다.

본 섹션의 요구사항은 주로 NIST 가이드라인 § 5.1.7.2에 해당한다.

#	설명	레벨
6.7.1	암호학적 인증 단언 (assertion) 을 검증하는 데 사용되는 인증서는 수정으로부터 보호되는 방식으로 저장되는지 검증한다.	3
6.7.2	챌린지 논스 (nonce) 는 최소 64 비트 길이여야 하며, 통계적으로 고유하거나 해당 암호 장치의 수명 동안 고유해야 한다.	3

V6.8 ID 제공자 (IdP) 를 통한 인증

ID 제공자 (IdP) 는 사용자에게 통합 신원을 제공한다. 사용자는 종종 여러 IdP 와 함께 여러 개의 신원을 보유하고 있다. 예를 들어, Azure AD, Okta, Ping Identity 또는 Google 과 같은 기업용 IdP 를 통한 기업 신원, 그리고 Facebook, Twitter, Google, WeChat 과 같은 소비자용 IdP 를 통한 개인 신원을 동시에 가질 수 있다. (이 목록은 특정 기업이나 서비스를 보증하는 것이 아니라, 많은 사용자가 이미 여러 신원을 보유하고 있다는 현실을 개발자가 고려하도록 권장하는 것이다.) 조직은 IdP 의 신원 증명 강도를 기준으로 기존 사용자 신원과의 통합을 고려해야 한다. 예를 들어, 정부 기관은 가짜 혹은 일회성 ID 를 쉽게 만들 수 있기 때문에 민감한 시스템의 로그인에 소셜 미디어 신원을 허용하지 않을 가능성이 큰 반면, 모바일 게임 회사는 활동적인 플레이어 기반을 확장하기 위해 주요 소셜 미디어 플랫폼과의 통합이 필요할 수 있다.

외부 IdP 를 안전하게 사용하려면 신원 위조 (identity spoofing) 나 위조된 단언 (assertion) 을 방지하기 위한 세심한 구성과 검증이 필요하다. 이 섹션은 이러한 위험을 해결하기 위한 요구사항을 제공한다.

#	설명	레벨
6.8.1	애플리케이션이 다중 ID 제공자 (IdP) 를 지원하는 경우, 사용자의 신원이 다른 지원 IdP 를 통해 위조될 수 없는지 검증한다. (예: 동일한 사용자 식별자를 사용하는 방식). 표준적인 대응책은 애플리케이션이 IdP ID(네임스페이스 역할) 와 IdP 내 사용자 ID 의 조합을 사용하여 사용자를 등록하고 식별하는 것이다.	2
6.8.2	인증 단언 (예: JWT 또는 SAML 단언) 의 디지털 서명이 존재하고 무결성이 항상 유지되는지를 검증한다. 서명이 없거나 유효하지 않은 단언은 거부해야 한다.	2
6.8.3	SAML 단언이 재생 공격의 방지를 위해 고유하게 처리되는지, 유효 기간 내에 단 한 번만 사용되는지 검증한다.	2

#	설명	레벨
6.8.4	애플리케이션이 별도의 ID 제공자 (IdP) 를 사용하고 특정 기능에 대해 특정 인증 강도, 방식, 또는 최근성을 요구하는 경우, 애플리케이션은 IdP 가 반환한 정보를 사용하여 이를 검증해야 한다. 예를 들어, OIDC 가 사용되는 경우, 이는‘acr’, ‘amr’, ‘auth_time’(존재하는 경우) 과 같은 ID 토큰 클레임을 검증함으로써 달성할 수 있다. IdP 가 이 정보를 제공하지 않는 경우, 애플리케이션은 최소 강도의 인증 메커니즘 (예: 사용자 이름과 비밀번호를 사용하는 단일요소 인증) 이 사용되었다고 가정하는 문서화된 대체 접근법을 가져야 한다.	2

참조

더 많은 정보는 다음을 참고한다:

- NIST SP 800-63 - Digital Identity Guidelines
- NIST SP 800-63B - Authentication and Lifecycle Management
- NIST SP 800-63 FAQ
- OWASP Web Security Testing Guide: Testing for Authentication
- OWASP Password Storage Cheat Sheet
- OWASP Forgot Password Cheat Sheet
- OWASP Choosing and Using Security Questions Cheat Sheet
- CISA Guidance on “Number Matching”
- Details on the FIDO Alliance

V7 세션 관리

제어 목표

세션 관리 메커니즘은 상태 비저장 (stateless) 통신 프로토콜 (예: HTTP) 을 사용할 때에도, 애플리케이션이 시간 경과에 따라 사용자와 장치 간의 상호작용을 연계할 수 있도록 한다. 최신 애플리케이션은 서로 다른 특성과 목적을 가진 여러 세션 토큰을 사용할 수 있다. 안전한 세션 관리 시스템은 공격자가 피해자의 세션을 획득, 사용 또는 악용하지 못하도록 방지한다. 세션을 유지하는 애플리케이션은 다음의 상위 수준 세션 관리 요구사항을 반드시 충족해야 한다.

- 세션은 각 개인에게 고유하며, 추측되거나 공유될 수 없어야 한다.
- 세션은 더 이상 필요하지 않을 때 무효화되며, 비활성 상태가 지속되면 시간 초과되어야 한다.

이 장의 많은 요구사항은 NIST SP 800-63 디지털 신원 지침 중 일부 제어 항목과 관련이 있으며, 일반적인 위협 및 자주 악용되는 인증 취약점에 초점을 맞춘다.

특정 세션 관리 메커니즘의 구현 세부사항에 대한 요구사항은 다른 장에서 확인할 수 있다.

- HTTP 쿠키는 세션 토큰을 보호하는 데 일반적으로 사용되는 메커니즘이다. 쿠키에 대한 구체적 보안 요구사항은“웹 프론트엔드 보안”장에서 확인할 수 있다.

- 자체 포함 토큰은 세션을 유지하는 수단으로 자주 사용된다. 자체 포함 토큰에 대한 구체적 보안 요구사항은“자체 포함 토큰”장에서 확인할 수 있다.

V7.1 세션 관리 문서화

모든 애플리케이션에 동일하게 적용할 수 있는 단일 패턴은 존재하지 않는다. 따라서 모든 경우에 적합한 범위와 한계를 보편적으로 정의하는 것은 불가능하다. 세션 처리를 구현하고 테스트하기 전에, 이와 관련된 보안 결정을 문서화한 위험 분석을 반드시 수행해야 한다. 이는 세션 관리 시스템이 애플리케이션의 특정 요구사항에 맞게 조정되도록 보장한다.

상태 유지형 (stateful) 또는 상태 비저장형 (stateless) 세션 메커니즘 중 어느 것을 선택하더라도, 선택한 솔루션이 모든 관련 보안 요구사항을 충족할 수 있음을 입증하기 위해 분석은 반드시 완전하게 이루어지고 문서화되어야 한다. 또한 사용 중인 SSO(Single Sign-on) 메커니즘과의 상호작용도 고려하는 것을 권장한다.

#	설명	레벨
7.1.1	사용자의 세션 비활성 시간 초과 및 절대 최대 세션 수명에 대한 문서가 존재하고, 다른 제어와의 조합에서 적절하며, NIST SP 800-63B 재인증 요구사항에서 벗어날 시 그 근거가 문서에 포함되어 있는지 검증한다.	2
7.1.2	문서가 하나의 계정에서 허용되는 동시 (병렬) 세션 수와 활성 세션 수의 최대값에 도달했을 때의 의도된 동작과 수행할 조치를 정의하는지 검증한다.	2
7.1.3	통합 신원 관리 생태계 (예: SSO 시스템) 의 일부로 사용자 세션을 생성하고 관리하는 모든 시스템이 문서화되어 있으며, 세션 수명, 종료, 재인증이 필요한 기타 조건을 조율하기 위한 제어가 포함되어 있는지 검증한다.	2

V7.2 기본 세션 관리 보안

이 절은 세션 토큰이 안전하게 생성되고 검증되도록 하는 세션 보안의 필수 요구사항을 다룬다.

#	설명	레벨
7.2.1	애플리케이션이 모든 세션 토큰 검증을 신뢰할 수 있는 백엔드 서비스에서 수행하는지 검증한다.	1
7.2.2	애플리케이션이 세션 관리를 위해 정적 API 시크릿이나 키가 아닌, 자체 포함 토큰 또는 참조 토큰을 동적으로 생성하여 사용하는지 검증한다.	1
7.2.3	참조 토큰이 사용자 세션을 나타내는 데 사용되는 경우, 그 토큰이 고유하고, 암호학적으로 안전한 의사난수 생성기 (CSPRNG) 를 사용해 생성되며 최소 128 비트의 엔트로피를 갖는지 검증한다.	1
7.2.4	애플리케이션이 사용자 인증 (재인증 포함) 시 새로운 세션 토큰을 생성하고 기존 세션 토큰을 종료하는지 검증한다.	1

V7.3 세션 타임아웃

세션 타임아웃 메커니즘은 세션 하이재킹 및 기타 세션 악용 가능성을 최소화한다. 타임아웃 설정은 반드시 문서화된 보안 결정에 부합해야 한다.

#	설명	레벨
7.3.1	위험 분석 및 문서화된 보안 결정에 따라 재인증을 강제하는 비활성 시간 초과가 존재하는지 검증한다.	2
7.3.2	위험 분석 및 문서화된 보안 결정에 따라 재인증을 강제하는 절대 최대 세션 수명이 존재하는지 검증한다.	2

V7.4 세션 종료

세션 종료는 애플리케이션 자체에서 처리할 수도 있고, 애플리케이션 대신 SSO 제공자가 세션 관리를 담당하는 경우 SSO 제공자에서 처리할 수도 있다. 이 절의 요구사항을 고려할 때, 일부는 SSO 제공자가 제어할 수 있으므로 SSO 제공자가 범위(scope) 에 포함되는지 여부를 결정해야 한다.

세션 종료는 재인증이 필요하도록 하고, 이는 애플리케이션, 통합 로그인 (존재하는 경우), 그리고 모든 의존 서비스에 걸쳐 효과적으로 적용됨이 권장된다.

상태 유지형 세션 메커니즘에서는 종료 시 백엔드에서 세션을 무효화하는 것이 일반적이다. 자체 포함 토큰의 경우, 토큰이 만료 시까지 계속 유효할 수 있으므로 이를 취소하거나 차단하기 위한 추가 조치가 필요하다.

#	설명	레벨
7.4.1	로그아웃이나 만료와 같이 세션 종료가 트리거될 때, 애플리케이션이 해당 세션의 추가 사용을 허용하지 않는지 검증한다. 참조 토큰 또는 상태 유지형 세션의 경우, 이는 애플리케이션 백엔드에서 세션 데이터를 무효화하는 것을 의미한다. 자체 포함 토큰을 사용하는 애플리케이션은 종료된 토큰 목록 유지, 사용자별 날짜·시간 이전에 발급된 토큰 거부, 사용자별 서명 키 회전과 같은 방법을 사용해야 한다.	1
7.4.2	사용자 계정이 비활성화되거나 삭제될 때 (예: 직원 퇴사), 애플리케이션이 모든 활성 세션을 종료하는지 검증한다.	1
7.4.3	비밀번호 재설정·복구를 통한 변경, MFA 설정 변경 등 어떤 인증 요소라도 성공적으로 변경 또는 제거한 후, 모든 다른 활성 세션을 종료할 수 있는 옵션을 제공하는지 검증한다.	2
7.4.4	인증이 필요한 모든 페이지의 로그아웃 기능에 접근하는 것이 쉽고 잘 보이는지 검증한다.	2
7.4.5	애플리케이션 관리자가 개별 사용자 또는 모든 사용자의 활성 세션을 종료할 수 있는지 검증한다.	2

V7.5 세션 악용 방어

이 절은 하이재킹되었거나, 활성 사용자 세션의 존재와 권한을 악용하는 벡터를 통해 남용될 수 있는 활성 세션의 위험을 완화하기 위한 요구사항을 제시한다. 예를 들어, 악성 콘텐츠 실행을 통해 인증된 피해자 브라우저가 피해자의 세션을 사용하여 동작을 수행하게 만드는 경우가 있다.

이 절의 요구사항을 고려할 때, “인증 (Authentication)”장의 레벨별 지침을 함께 참고해야 한다.

#	설명	레벨
7.5.1	이메일 주소, 전화번호, MFA 구성, 계정 복구에 사용되는 기타 정보 등 인증에 영향을 줄 수 있는 민감한 계정 속성을 변경하기 전에 전체 재인증을 요구하는지 검증한다.	2
7.5.2	사용자가 현재 활성 세션을 전체 또는 일부 확인하고 (최소 요소 한 개에 대한 인증을 다시 수행한 후) 종료할 수 있는지 검증한다.	2
7.5.3	매우 민감한 거래나 작업을 수행하기 전에 최소 한 요소 인증 또는 2 차 검증을 요구하는지 검증한다.	3

V7.6 페더레이션 재인증

이 절은 신뢰 당사자 (Relying Party;RP) 또는 ID 공급자 (Identity Provider; IDP) 코드를 작성하는 경우에 적용된다. 요구사항은 NIST SP 800-63C 연방·인증서 발급 지침에서 파생되었다.

#	설명	레벨
7.6.1	RP 와 IdP 간 세션 수명과 종료가 문서대로 동작하며, IdP 인증 이벤트 간의 최대 시간이 도달하는 경우 등 필요한 시점에 재인증을 요구하는지 검증한다.	2
7.6.2	세션 생성 시 사용자 동의 또는 명시적 동작을 요구하여, 사용자 상호작용 없이 새로운 애플리케이션 세션이 생성되지 않도록 하는지 검증한다.	2

참조

더 많은 정보는 다음을 참고한다:

- OWASP Web Security Testing Guide: Session Management Testing
- OWASP Session Management Cheat Sheet

V8 권한 부여

제어 목표

권한 부여는 허가된 소비자 (사용자, 서버, 기타 클라이언트) 에게만 접근을 허용하는 것을 보장한다. 최소 권한의 원칙 (PLOP) 을 적용하기 위해, 검증된 애플리케이션은 반드시 다음의 상위 수준 요구사항을 충족해야 한다:

- 권한 부여 규칙을 문서화하며, 문서에는 의사 결정 요소와 환경적 맥락을 포함한다.
- 소비자는 자신에게 부여된 권한으로 허용된 자원에만 접근할 수 있는 것이 권장된다.

V8.1 권한 부여 문서화

포괄적인 권한 부여 문서는 보안 결정이 일관성 있게 적용되고, 감사 가능하며, 조직의 정책에 부합함을 보장하는 데 필수적이다. 이는 보안 요구사항을 개발자, 관리자, 테스터에게 명확하고 실행 가능하게 전달함으로써 무단 접근 위험을 줄인다.

#	설명	레벨
8.1.1	권한 부여 문서가 소비자 권한과 자원 속성에 따라 기능 수준 및 데이터별 접근을 제한하는 규칙을 정의하는지 검증한다.	1
8.1.2	권한 부여 문서가 소비자 권한과 자원 속성에 따라 필드 수준 접근 제한 (읽기 및 쓰기 모두) 을 정의하는지 검증한다. 이러한 규칙은 관련 데이터 객체의 상태 (state) 나 상태값 (status) 과 같은 다른 속성 값에 의존할 수 있다.	2
8.1.3	애플리케이션 문서가 인증 및 권한 부여와 관련된 보안 결정을 내리는 데 사용되는 환경적, 맥락적 속성 (예: 시간대, 사용자 위치, IP 주소, 디바이스 등) 을 정의하는지 검증한다.	3
8.1.4	인증 및 권한 부여 문서가 기능 수준, 데이터별, 필드 수준 권한 부여에 더해 환경적, 맥락적 요소가 의사 결정에 어떻게 사용되는지 정의하는지 검증한다. 여기에는 평가되는 속성, 위험 임계값, 수행되는 조치 (예: 허용, 추가 인증 요청, 거부, 단계적 인증) 을 포함하는 것을 권장한다.	3

V8.2 일반 권한 부여 설계

기능, 데이터, 필드 수준에서 세분화된 권한 부여 제어의 구현은, 소비자가 명시적으로 부여된 범위 내에서만 접근할 수 있도록 보장할 수 있다.

#	설명	레벨
8.2.1	애플리케이션이 기능 수준의 접근을 명시적 권한이 있는 소비자에게만 허용하는지 검증한다.	1
8.2.2	애플리케이션이 특정 데이터 항목에 대한 명시적 권한이 있는 소비자에게만 데이터별 접근을 허용하여, 안전하지 않은 직접 객체 참조 (IDOR) 및 객체 수준 권한 부여 취약점 (BOLA) 을 방지하는지 검증한다.	1

#	설명	레벨
8.2.3	애플리케이션이 특정 필드에 대한 명시적 권한이 있는 소비자에게만 필드 수준 접근을 허용하여, 객체 속성 수준 권한 부여 취약점 (BOPLA) 을 방지하는지 검증한다.	2
8.2.4	애플리케이션 문서에서 정의한 바에 따라, 소비자의 환경적, 맥락적 속성 (예: 시간대, 위치, IP 주소 또는 장치) 에 기반한 적응형 보안 제어가 인증 및 권한 부여 결정에 구현되는지 검증한다. 이러한 제어는 소비자가 새 세션을 시작할 때뿐만 아니라 기존 세션 중에도 반드시 적용되어야 한다.	3

V8.3 운영 수준 권한 부여

애플리케이션 아키텍처의 적절한 계층에서 권한 부여 변경 사항을 즉시 적용하는 것은, 특히 동적인 환경에서 무단 행위를 방지하는 데 매우 중요하다.

#	설명	레벨
8.3.1	애플리케이션이 권한 부여 규칙을 신뢰할 수 있는 서비스 계층에서 적용하고, 클라이언트 측 JavaScript 처럼 신뢰할 수 없는 소비자가 조작할 수 있는 제어에 의존하지 않는지 검증한다.	1
8.3.2	권한 부여 결정에 영향을 미치는 값의 변경 사항이 즉시 적용되는지 검증한다. 자체 포함 토큰 (self-contained token) 과 같이 즉시 적용이 불가능한 경우, 소비자가 더 이상 권한이 없는 작업을 수행할 때 이를 알리고 변경을 되돌리는 완화 제어를 반드시 두어야 한다. 단, 이 방법은 정보 유출을 방지하지는 못한다.	3
8.3.3	객체에 대한 접근이, 대리인이나 이를 대신 수행하는 서비스의 권한이 아닌, 원래 주체 (예: 소비자) 의 권한에 기반하는지 검증한다. 예를 들어, 소비자가 인증을 위해 자체 포함 토큰을 사용하여 웹 서비스를 호출하고, 해당 서비스가 다른 서비스에 데이터를 요청하는 경우, 두 번째 서비스는 첫 번째 서비스의 머신-투-머신 토큰이 아니라 소비자의 토큰을 사용하여 권한 결정을 내려야 한다.	3

V8.4 기타 권한 부여 고려사항

특히 관리 인터페이스나 다중 테넌트 환경에서의 권한 부여에 대한 추가적인 고려사항은 무단 접근을 방지하는 데 도움이 된다.

#	설명	레벨
8.4.1	다중 테넌트 애플리케이션이 교차 테넌트 제어를 사용하여, 소비자의 작업이 권한이 없는 다른 테넌트에 영향을 미치지 않도록 하는지 검증한다.	2

#	설명	레벨
8.4.2	관리 인터페이스 접근이 연속적인 소비자 신원 검증, 장치 보안 상태 평가, 맥락 기반 위험 분석 등 복수의 보안 계층을 포함하는지 검증한다. 네트워크 위치나 신뢰된 엔드포인트가 무단 접근 가능성을 줄일 수는 있더라도, 단독으로 권한 부여 판단의 유일한 요소가 되어서는 안 된다.	3

참조

더 많은 정보는 다음을 참고한다:

- OWASP Web Security Testing Guide: Authorization
- OWASP Authorization Cheat Sheet

V9 자체 포함 토큰 (self-contained token)

제어 목표

자체 포함 토큰의 개념은 2012 년에 발표된 RFC 6749 OAuth 2.0 원문에서 언급된다. 이는 수신 서비스가 보안 결정을 내릴 때 의존할 데이터 또는 클레임 (claims) 을 포함하는 토큰을 의미한다. 이는 수신 서비스가 데이터를 로컬에서 조회하는 데 사용하는 식별자만 포함하는 단순 토큰과는 구별되어야 한다. 자체 포함 토큰의 가장 일반적인 예시는 JSON 웹 토큰 (JSON Web Token; JWT) 과 SAML 어설션 (SAML assertions) 이 있다.

자체 포함 토큰의 사용은 OAuth 및 OIDC 외부에서도 매우 광범위하게 확산되었다. 동시에 이 메커니즘의 보안은 토큰의 무결성을 검증하고 토큰이 특정 컨텍스트에 유효한지 확인하는 능력에 의존한다. 이 과정에는 많은 위험요소가 있으며, 이 장에서는 애플리케이션이 이를 방지하기 위해 갖춰야 할 메커니즘에 대한 구체적인 세부 정보를 제공한다.

V9.1 토큰 출처 및 무결성

이 섹션에는 토큰이 신뢰할 수 있는 주체에 의해 생성되었으며 변조되지 않았음을 확인하기 위한 요구사항을 포함한다.

#	설명	레벨
9.1.1	자체 포함 토큰 내용을 허용하기 전에 변조를 방지하기 위해 디지털 서명 또는 메시지 인증 코드 (MAC) 를 사용하여 토큰의 무결성을 검증해야 한다.	1
9.1.2	주어진 컨텍스트에 대해 자체 포함 토큰을 생성하고 검증할 때, 허용 목록에 있는 알고리즘만 사용될 수 있는지 검증해야 한다. 허용 목록에는 사용 가능한 알고리즘을 모두 포함해야 하며, 이상적으로는 대칭 또는 비대칭 알고리즘 중 하나만을 포함해야 한다. 또한‘None’알고리즘은 반드시 제외해야 한다. 대칭 및 비대칭 알고리즘 모두를 지원해야 하는 경우, 키 혼동을 방지하기 위해 추가 제어가 필요하다.	1

#	설명	레벨
9.1.3	자체 포함 토큰을 검증하는 데 사용되는 키 자료가 토큰 발행자를 위한 신뢰할 수 있는 사전 구성된 출처에서 비롯되었는지 검증하여, 공격자가 신뢰할 수 없는 출처 및 키를 지정하는 것을 방지해야 한다. JWT 및 기타 JWS 구조의 경우, 'jku', 'x5u', 'jwk'와 같은 헤더는 신뢰할 수 있는 출처의 허용 목록에 대해 검증되어야 한다.	1

V9.2 토큰 내용

자체 포함 토큰의 내용에 기반하여 보안 결정을 내리기 전에, 토큰이 유효 기간 내에 제시되었는지, 수신 서비스에서 사용하도록 의도되었는지, 그리고 제시된 목적에 맞게 사용되었는지 검증해야 한다. 이는 동일한 발급자로부터 발급된 서로 다른 서비스 간 또는 서로 다른 토큰 유형 간의 안전하지 않은 교차 사용을 방지하는 데 도움이 된다.

OAuth 및 OIDC 에 대한 구체적인 요구사항은 별도의 장 V10 OAuth and OIDC 에서 다룬다.

#	설명	레벨
9.2.1	토큰 데이터에 유효 기간이 존재하는 경우, 검증 시간이 이 유효 기간 내에 있을 때만 토큰 및 해당 내용이 허용되는지 검증해야 한다. 예를 들어, JWT 의 경우'nbf'및'exp'클레임이 검증되어야 한다.	1
9.2.2	토큰을 수신하는 서비스가 토큰 내용을 수락하기 전에 올바른 유형이며 의도된 목적을 위한 것인지 토큰을 검증하는지 확인해야 한다. 예를 들어, 인가 결정에는 액세스 토큰만 허용될 수 있으며, 사용자 인증을 증명하는 데는 ID 토큰만 사용될 수 있다.	2
9.2.3	서비스가 대상 서비스 (audience) 와 함께 사용하기 위한 토큰만 허용하는지 검증해야 한다. JWT 의 경우, 서비스에 정의된 허용 목록에 대해'aud'클레임을 검증함으로써 이를 달성할 수 있다.	2
9.2.4	토큰 발행자가 다른 대상 (audience) 에게 토큰을 발행하기 위해 동일한 개인 키를 사용하는 경우, 발행된 토큰이 의도된 대상을 고유하게 식별하는 대상 제한을 포함하는지 검증해야 한다. 이는 토큰이 의도하지 않은 대상에서 재사용되는 것을 방지한다. 대상 식별자가 동적으로 프로비저닝되는 경우, 토큰 발행자는 대상 위장 (audience impersonation) 을 초래하지 않도록 대상들을 검증해야 한다.	2

참조

더 많은 정보는 다음을 참고한다:

- OWASP JSON Web Token Cheat Sheet for Java Cheat Sheet (유용한 일반 지침 포함)

V10 OAuth 와 OIDC

제어 목표

OAuth2(본 장에서는 OAuth 로 지칭) 는 권한 위임을 위한 산업 표준 프레임워크이다. 예를 들어, OAuth 를 사용하면 클라이언트 애플리케이션은 사용자가 클라이언트 애플리케이션에 권한을 위임한 경우 사용자를 대신하여 API(서버 리소스)에 접근할 수 있다.

OAuth 자체는 사용자 인증을 위해 설계되지 않았다. OpenID Connect(OIDC) 프레임워크는 OAuth 위에 사용자 식별 계층을 추가하여 OAuth 를 확장한다. OIDC 는 표준화된 사용자 정보, 싱글 사인온 (SSO), 세션 관리 등의 기능을 지원한다. OIDC 는 OAuth 를 확장한 표준 사양이므로 본 장의 OAuth 요구사항은 OIDC 에도 적용된다.

OAuth 에 정의된 역할은 다음과 같다:

- OAuth 클라이언트는 서버 리소스에 접근하려는 애플리케이션이다 (예: 발급된 액세스 토큰을 사용하여 API 호출). OAuth 클라이언트는 서버에서 동작하는 애플리케이션인 경우가 많다.
  - 기밀 클라이언트는 인가 서버에 자체적으로 인증하는 데 사용하는 자격 증명 (Credentials) 의 기밀성을 유지할 수 있는 클라이언트이다.
  - 공개 클라이언트는 인가 서버에 인증하기 위한 자격 증명의 기밀성을 유지할 수 없다. 따라서 자체적으로 인증 (예: ‘client_id’및‘client_secret’파라미터 사용) 하는 대신, 자체 식별만 한다 (‘client_id’파라미터 사용).
- OAuth 리소스 서버는 OAuth 클라이언트에 리소스를 제공하는 서버 API 이다.
- OAuth 인가 서버는 OAuth 클라이언트에 액세스 토큰을 발급하는 서버 애플리케이션이다. 이 액세스 토큰을 통해 OAuth 클라이언트는 최종 사용자를 대신하여 또는 OAuth 클라이언트 자체를 대신하여 리소스 서버의 리소스에 접근할 수 있다. 인가 서버는 종종 별도의 애플리케이션이지만, (적절한 경우) 요건에 맞는 리소스 서버에 통합될 수 있다.
- 리소스 소유자는 리소스 서버에 호스팅된 리소스에 대해 제한된 접근 권한을 획득하도록 OAuth 클라이언트에 권한을 위임하는 최종 사용자이다. 리소스 소유자는 인가 서버와 상호 작용함으로써 이 권한 위임에 동의한다.

OIDC 에 정의된 역할은 다음과 같다:

- 신뢰 당사자 (Relying Party) 는 OpenID 공급자를 통해 최종 사용자 인증을 요청하는 클라이언트 애플리케이션이다. 이는 OAuth 클라이언트의 역할을 수행한다.
- OpenID 공급자는 최종 사용자를 인증하고 신뢰 당사자에 OIDC 클레임 (claims) 을 제공할 수 있는 OAuth 인가 서버이다. OpenID 공급자는 ID 공급자 (IdP) 일 수 있지만, 연합 인증 시나리오 (federated scenarios) 에서는 OpenID 공급자와 ID 공급자 (최종 사용자가 인증하는 곳) 가 다른 서버 애플리케이션일 수 있다.

OAuth 와 OIDC 는 원래 외부 (third-party) 애플리케이션을 위해 설계되었으며, 오늘날에는 내부 (first-party) 애플리케이션에서도 종종 사용된다. 그러나 인증 및 세션 관리와 같은 내부 시나리오에서 사용될 때, 프로토콜이 복잡성을 더하게 되며, 이로 인해 새로운 보안 과제가 발생할 수 있다.

OAuth 와 OIDC 는 다양한 유형의 애플리케이션에 사용될 수 있지만, 본 장의 ASVS 와 요구사항은 웹 애플리케이션과 API 에 중점을 두고 있다.

OAuth 와 OIDC 는 웹 기술 위에 구축된 논리 계층으로 간주될 수 있으므로, 본 장의 내용만을 따로 떼어내서 해석하지 않고, 다른 장의 일반 요구사항과 함께 적용해야 한다.

본 장은 https://oauth.net/2/ 및 https://openid.net/developers/specs/ 에서 찾을 수 있는 사양에 맞춰 OAuth2 및 OIDC 에 대한 현재의 모범 사례를 다룬다. RFC 가 성숙하다고 간주되더라도 자주 업데이트된다. 따라서 본 장의 요구사항을 적용할 때 최신 버전과 일치시키는 것이 중요하다. 자세한 내용은 참조 섹션을 확인한다.

이 영역의 복잡성을 고려할 때, 안전한 OAuth 또는 OIDC 솔루션을 위해서는 잘 알려진 산업 표준의 인가 서버를 사용하고, 권장되는 보안 구성을 적용하는 것이 매우 중요하다.

본 장에서 사용되는 용어는 OAuth RFC 및 OIDC 사양과 일치하지만, OIDC 용어는 OIDC 에 특화된 요구사항에만 사용되며, 그 외에는 OAuth 용어가 사용된다는 점에 유의해야 한다.

OAuth 및 OIDC 의 맥락에서, 본 장의“토큰”이라는 용어는 다음을 의미한다:

- 액세스 토큰은 리소스 서버에서만 사용되어야 하며, 조회 (introspection) 를 통해 검증되는 참조 토큰이거나 특정 키 정보를 사용하여 검증되는 자체 포함 토큰일 수 있다.
- 리프레시 토큰은 토큰을 발급한 인가 서버에서만 사용되어야 한다.
- OIDC ID 토큰은 인가 플로우를 시작했던 클라이언트에서만 사용되어야 한다.

본 장의 일부 요구사항의 위험 수준은 클라이언트가 기밀 클라이언트인지, 공개 클라이언트로 간주되는지에 따라 달라질 수 있다. 강력한 클라이언트 인증을 사용하면 많은 공격 벡터를 줄일 수 있으므로, L1 애플리케이션에서 기밀 클라이언트를 사용하는 경우 일부 요구사항이 완화될 수 있다.

V10.1 일반적인 OAuth 및 OIDC 보안

이 섹션은 OAuth 또는 OIDC 를 사용하는 모든 애플리케이션에 적용되는 일반적인 아키텍처 요구사항을 다룬다.

#	설명	레벨
10.1.1	엄격하게 필요한 구성 요소에만 토큰이 전송되는지 검증해야 한다. 예를 들어, 브라우저 기반 JavaScript 애플리케이션에 프론트엔드 전용 백엔드 (Backend For Frontend; BFF) 패턴을 사용하는 경우, 액세스 토큰 및 리프레시 토큰은 백엔드에서만 접근 가능해야 한다.	2
10.1.2	클라이언트는 인가 서버에서 제공하는 값 (예: 인가 코드 또는 ID 토큰) 이 동일한 사용자 에이전트 세션 및 트랜잭션에서 시작된 인가 플로우의 결과인 경우에만 허용하는지 검증해야 한다. 이를 위해 클라이언트가 생성하는 비밀 값 (PKCE(Proof Key for Code Exchange) 의 ‘code_verifier’, ‘state’, OIDC 의‘nonce’등) 은 추측이 불가능해야 하며, 해당 트랜잭션에 고유해야 하고, 트랜잭션이 시작된 클라이언트와 사용자 에이전트 세션 양쪽에 안전하게 결합 (Binding) 되어 있어야 한다.	2

V10.2 OAuth 클라이언트

이 요구사항은 OAuth 클라이언트 애플리케이션의 책임을 상세히 규정한다. 예를 들어 클라이언트는 웹 서버 백엔드 (일반적으로 BFF 역할 수행), 백엔드 서비스 통합, 또는 프론트엔드 단일 페이지 애플리케이션 (브라우저 기반 애플리케이션) 일 수 있다.

일반적으로 백엔드 클라이언트는 기밀 클라이언트로 간주되고, 프론트엔드 클라이언트는 공개 클라이언트로 간주된다. 그러나 최종 사용자 장치에서 실행되는 네이티브 애플리케이션은 OAuth 동적 클라이언트 등록을 사용하는 경우 기밀로 간주될 수 있다.

#	설명	레벨
10.2.1	코드 플로우를 사용하는 경우, OAuth 클라이언트는 토큰 요청을 유도하는 사이트 간 요청 위조 공격 (Cross-Site Request Forgery; CSRF) 에 대한 보호 기능을 갖추고 있는지 검증한다. 이를 위해 인가 요청 시 전송된‘state’파라미터를 검증하거나, PKCE 기능을 사용해야 한다.	2
10.2.2	OAuth 클라이언트가 둘 이상의 인가 서버와 상호 작용할 수 있는 경우, 믹스업 공격에 대한 방어 대책을 갖는지 검증해야 한다. 예를 들어, 인가 서버가‘iss’파라미터 값을 반환하도록 요구하고 인가 응답 및 토큰 응답에서 이를 검증할 수 있다.	2
10.2.3	OAuth 클라이언트가 인가 서버에 대한 요청에서 필요한 범위 (또는 기타 인가 파라미터) 만 요청하는지 검증해야 한다.	3

V10.3 OAuth 리소스 서버

ASVS 및 본 장의 맥락에서 리소스 서버는 API 를 의미한다. 안전한 접근 제어를 제공하기 위해 리소스 서버는 다음을 수행해야 한다:

- 토큰 형식 및 관련 프로토콜 사양 (예: JWT 검증 또는 OAuth 토큰 조회) 에 따라 액세스 토큰을 검증해야 한다.
- 토큰이 유효한 경우, 액세스 토큰의 정보 및 부여된 권한을 기반으로 인가 결정을 강제해야 한다. 예를 들어, 리소스 서버는 클라이언트 (리소스 소유자를 대신하여 작동) 가 요청된 리소스에 접근할 권한이 있는지 검증해야 한다.

따라서 여기에 나열된 요구사항은 OAuth 또는 OIDC 에 특화되어 있으며, 토큰 검증 후 토큰의 정보를 기반으로 인가를 수행하기 전에 이행되어야 한다.

#	설명	레벨
10.3.1	리소스 서버가 해당 서비스 (audience) 에 사용하도록 의도된 액세스 토큰만 수락하는지 검증해야 한다. 대상은 구조화된 액세스 토큰 (예: JWT 의‘aud’클레임) 에 포함될 수 있거나, 토큰 조회 엔드포인트를 사용하여 확인할 수 있다.	2
10.3.2	리소스 서버가 위임된 인가를 정의하는 액세스 토큰의 클레임을 기반으로 인가 결정을 시행하는지 검증해야 한다. ‘sub’, ‘scope’, ’authorization_details’와 같은 클레임이 존재하는 경우, 이는 결정에 반영되어야 한다.	2
10.3.3	액세스 토큰 (JWT 또는 관련 토큰 조회 응답) 에서 고유한 사용자를 식별해야 하는 접근 제어 결정이 필요한 경우, 리소스 서버가 다른 사용자에게 재할당될 수 없는 클레임에서 사용자를 식별하는지 검증해야 한다. 일반적으로 이는‘iss’및‘sub’클레임의 조합을 사용하는 것을 의미한다.	2

#	설명	레벨
10.3.4	리소스 서버가 특정 인증 강도, 방법 또는 최신성을 요구하는 경우, 제시된 액세스 토큰이 이러한 제약 조건을 충족하는지 검증해야 한다. 예를 들어, 토큰을 제시한 경우 OIDC 의‘acr’, ‘amr’ 및‘auth_time’클레임을 각각 사용해야 한다.	2
10.3.5	리소스 서버가 발신자 제약 액세스 토큰, 즉 OAuth 2 용 상호 TLS(Mutual TLS; mTLS) 또는 OAuth 2 소유 증명 (Demonstration of Proof of Possession; DPoP) 를 요구하여 도난된 액세스 토큰 사용 또는 액세스 토큰 재사용 (권한 없는 당사자로부터) 을 방지하는지 검증해야 한다.	3

V10.4 OAuth 인가 서버

이 요구사항은 OpenID 공급자를 포함한 OAuth 인가 서버의 책임을 상세히 설명한다.

클라이언트 인증의 경우, ‘self_signed_tls_client_auth’메소드는 RFC 8705의 섹션 2.2에서 요구하는 전제조건을 충족하는 경우에 허용된다.

#	설명	레벨
10.4.1	인가 서버는 클라이언트 별로 사전 등록된 URI 의 허용 목록을 기반으로 리디렉션 URI 를 정확한 문자열 비교 방식으로 검증해야 한다.	1
10.4.2	인가 서버가 인가 응답에서 인가 코드를 반환하는 경우, 해당 코드는 토큰 요청에 한번만 사용될 수 있어야 한다. 이미 액세스 토큰을 발급하는 데 사용된 인가 코드를 사용하여 두 번째 요청이 발생하면, 인가 서버는 토큰 요청을 거부하고 해당 인가 코드와 관련된 모든 발급된 토큰을 취소해야 한다.	1
10.4.3	인가 코드가 단기간만 유효한지 검증해야 한다. 최대 수명은 L1 및 L2 애플리케이션의 경우 최대 10 분, L3 애플리케이션의 경우 최대 1 분 이내여야 한다.	1
10.4.4	인가 서버는 특정 클라이언트에 대해, 해당 클라이언트가 사용할 필요가 있는 인가 방식 (grant) 만 사용하도록 허용해야 한다. token(암시적 플로우) 및 password(리소스 소유자 비밀번호 자격 증명 플로우) 방식은 더 이상 사용되어서는 안 된다.	1
10.4.5	인가 서버가 공개 클라이언트에 대한 리프레시 토큰 재사용 공격을 완화하는지 검증해야 한다. 가급적이면 발신자 제약 리프레시 토큰, 즉 소유 증명이나 mTLS 를 사용한 인증서 바인딩 액세스 토큰을 사용해야 한다. L1 및 L2 애플리케이션의 경우, 리프레시 토큰 회전 (rotation) 이 사용될 수 있다. 리프레시 토큰 회전이 사용되는 경우, 인가 서버는 리프레시 토큰을 사용 후 무효화해야 하며, 이미 사용되고 무효화된 리프레시 토큰이 제공되면 해당 인가에 대한 모든 리프레시 토큰을 철회 (revoke) 해야 한다.	1

#	설명	레벨
10.4.6	코드 인가 방식 (code grant) 이 사용되는 경우, 인가 서버가 PKCE 를 요구하여 인가 코드 탈취 공격을 완화하는지 검증해야 한다. 인가 요청의 경우, 인가 서버는 유효한 'code_challenge'값을 요구해야 하며, 'code_challenge_method'값으로'plain'를 허용해서는 안 된다. 토큰 요청의 경우, 'code_verifier'파라미터의 검증을 요구해야 한다.	2
10.4.7	인가 서버가 인증되지 않은 동적 클라이언트 등록을 지원하는 경우, 악의적인 클라이언트 애플리케이션의 위험을 완화하는지 검증해야 한다. 등록된 URI 와 같은 클라이언트 메타데이터를 검증하고, 사용자의 동의를 확인해야 하며, 신뢰할 수 없는 클라이언트 애플리케이션으로 인가 요청을 처리하기 전에 사용자에게 경고해야 한다.	2
10.4.8	슬라이딩 리프레시 토큰 만료 (expiration) 가 적용되더라도, 리프레시 토큰에 절대 만료 시점 (absolute expiration) 이 포함되어 있는지 확인한다.	2
10.4.9	악의적인 클라이언트 또는 도난된 토큰의 위험을 완화하기 위해 인가 서버 사용자 인터페이스를 통해 권한 있는 사용자가 리프레시 토큰 및 참조 액세스 토큰을 철회할 수 있는지 검증해야 한다.	2
10.4.10	토큰 요청, 푸시된 인가 요청 (Pushed Authorization Request; PAR), 토큰 철회 요청과 같은 클라이언트에서 인가 서버로의 백채널 요청 (backchannel request) 에 대해 기밀 클라이언트가 인증되는지 검증해야 한다.	2
10.4.11	인가 서버 구성이 OAuth 클라이언트에 필요한 범위만 할당하는지 검증해야 한다.	2
10.4.12	주어진 클라이언트에 대해 인가 서버가 해당 클라이언트가 사용해야 하는'response_mode' 값만 허용하는지 검증해야 한다. 예를 들어, 인가 서버가 이 값을 예상 값과 비교하여 검증하거나, PAR 또는 JWT 기반 인가 요청 (JWT-secured Authorization Request; JAR) 을 사용하여 검증해야 한다.	3
10.4.13	인가 유형'code'가 항상 PAR 과 함께 사용되는지 검증해야 한다.	3
10.4.14	인가 서버가 mTLS 를 사용하는 인증서 바인딩 액세스 토큰 또는 소유 증명 바인딩 액세스 토큰을 사용하여 발신자 제약 (Proof-of-Possession) 액세스 토큰만 발급하는지 검증해야 한다.	3
10.4.15	최종 사용자 장치에서 실행되지 않는 서버에서 동작하는 클라이언트의 경우, 인가 서버가 'authorization_details'파라미터 값이 클라이언트 백엔드로부터 생성되었는지와 사용자가 해당 값을 변조되지 않았는지를 검증해야 한다. 예를 들어, PAR 또는 JAR 의 사용을 요구하여 확인해야 한다.	3
10.4.16	클라이언트가 기밀 클라이언트인지 검증하고, 인가 서버는 재사용 공격에 강하고 공개키 암호화에 기반한 강력한 클라이언트 인증 방식을 사용하도록 요구해야 한다. 예를 들어, mTLS(tls_client_auth, self_signed_tls_client_auth) 또는 비공개 키 기반 JWT 인증 (private_key_jwt) 등이 이에 해당한다.	3

V10.5 OIDC 클라이언트

OIDC 신뢰 당사자가 OAuth 클라이언트로 작동하므로, "V10.2 OAuth 클라이언트"섹션의 요구사항도 적용된다.

“V6 인증”장의“V6.8 ID 공급자를 사용한 인증”섹션에도 관련 일반 요구사항이 포함되어 있음에 유의해야 한다.

#	설명	레벨
10.5.1	신뢰 당사자로서 클라이언트가 ID 토큰 재사용 공격을 완화하는지 검증해야 한다. 예를 들어, ID 토큰의‘nonce’클레임이 OpenID 공급자에 전송된 인증 요청 (OAuth2 에서는 인가 서버에 전송된 인가 요청) 에서 전송된‘nonce’값과 일치하는지 확인해야 한다.	2
10.5.2	클라이언트가 ID 토큰 클레임, 일반적으로‘sub’클레임에서 사용자를 고유하게 식별하며, 해당 클레임이 ID 공급자 범위 내에서 다른 사용자에게 재할당될 수 없는지 검증해야 한다.	2
10.5.3	클라이언트는 악의적인 인가 서버가 인가 서버 메타데이터를 통해 다른 인가 서버를 가장하려는 시도를 거부하는지 검증해야 한다. 클라이언트는 인가 서버 메타데이터에 포함된 발급자 (issuer) URL 이 클라이언트가 예상하는 사전 구성된 발급자 URL 과 정확히 일치하지 않는 경우, 해당 메타데이터를 거부해야 한다.	2
10.5.4	클라이언트는 토큰의 aud 클레임 값이 클라이언트의 client_id 값과 동일한지 확인하여 ID 토큰이 해당 클라이언트 (audience) 를 대상으로 발급되었음을 검증해야 한다.	2
10.5.5	OIDC 백채널 로그아웃을 사용하는 경우, 신뢰 당사자는 강제 로그아웃을 통한 서비스 거부 공격 (Denial of Service) 및 로그아웃 플로우에서의 JWT 혼동 (cross-JWT confusion) 을 완화하는지 검증해야 한다. 클라이언트는 로그아웃 토큰이‘logout+jwt’값으로 올바르게 지정되어 있는지, 올바른 멤버 이름을 가진‘event’클레임을 포함하는지, 그리고‘nonce’클레임을 포함하지 않는지 검증해야 한다. 또한, 짧은 만료 시간 (예: 2 분) 을 갖는 것이 권장된다.	2

V10.6 OpenID 공급자

OpenID 공급자는 OAuth 인가 서버로 작동하므로, “V10.4 OAuth 인가 서버”섹션의 요구사항도 적용된다.

ID 토큰 플로우 (코드 플로우가 아닌) 을 사용하는 경우, 액세스 토큰이 발급되지 않으며 OAuth 인가 서버의 많은 요구사항이 적용되지 않음에 유의해야 한다.

#	설명	레벨
10.6.1	OpenID 공급자는 response_mode 파라미터 값으로’code’, ‘ciba’, ‘id_token’또는 ‘id_token code’만 허용하는지 검증해야 한다. 이 중‘id_token code’(OIDC 하이브리드 플로우) 보다는‘code’가 선호되며, ’token’(모든 암시적 플로우) 은 사용되어서는 안 된다.	2
10.6.2	OpenID 공급자가 강제 로그아웃을 통한 서비스 거부 공격을 완화하는지 검증해야 한다. 신뢰 당사자로 부터 시작된 로그아웃 요청에 id_token_hint 등의 파라미터가 포함되어 있는 경우, 이를 검증하거나 최종 사용자로부터 명시적인 확인을 받아야 한다.	2

V10.7 동의 관리

이 요구사항은 인가 서버에 의한 사용자 동의 검증을 다룬다. 적절한 사용자 동의 검증이 없으면 악의적인 행위자가 스푸핑 또는 사회 공학을 통해 사용자를 대신하여 권한을 획득할 수 있다.

#	설명	레벨
10.7.1	인가 서버가 사용자로부터 각 인가 요청에 대한 명시적인 동의 (consent) 를 확인하는지 검증해야 한다. 클라이언트의 신원을 보장할 수 없는 경우, 인가 서버는 항상 사용자에게 명시적으로 동의를 요청해야 한다.	2
10.7.2	인가 서버가 사용자 동의를 요청할 때, 동의하는 내용에 대한 충분하고 명확한 정보를 제공하는지 검증해야 한다. 이에 해당하는 경우, 요청된 인가의 성격 (일반적으로 범위, 리소스 서버, RAR(Rich Authorization Requests) 인가 세부 정보 기반), 인가된 애플리케이션의 신원, 인가의 유효 기간이 포함되어야 한다.	2
10.7.3	사용자가 인가 서버를 통해 부여한 동의를 검토, 수정 및 철회할 수 있는지 검증해야 한다.	2

참조

더 많은 정보는 다음을 참고한다:

- oauth.net
- OWASP OAuth 2.0 프로토콜 치트 시트

ASVS 에서 OAuth 관련 요구사항은 다음의 공개 및 초안 상태 RFC 를 사용한다:

- RFC6749 OAuth 2.0 인가 프레임워크
- RFC6750 OAuth 2.0 인가 프레임워크: 베어러 토큰 사용
- RFC6819 OAuth 2.0 위협 모델 및 보안 고려 사항
- RFC7636 OAuth 공개 클라이언트를 위한 코드 교환 증명 키
- RFC7591 OAuth 2.0 동적 클라이언트 등록 프로토콜
- RFC8628 OAuth 2.0 디바이스 인가 부여
- RFC8707 OAuth 2.0 을 위한 리소스 표시자
- RFC9068 OAuth 2.0 액세스 토큰을 위한 JSON 웹 토큰 (JWT) 프로파일
- RFC9126 OAuth 2.0 푸시된 인가 요청 (PAR)
- RFC9207 OAuth 2.0 인가 서버 발급자 식별
- RFC9396 OAuth 2.0 확장된 인가 요청
- RFC9449 OAuth 2.0 PoP(Demonstrating Proof of Possession)
- RFC9700 OAuth 2.0 보안을 위한 현재 모범 사례
- 브라우저 기반 애플리케이션을 위한 OAuth 2.0 초안
- OAuth 2.1 인가 프레임워크 초안

OpenID Connect 에 대한 자세한 내용은 다음을 참조한다:

- OpenID Connect Core 1.0
- FAPI 2.0 보안 프로파일

V11 암호화

제어 목표

본 장의 목표는 암호화의 일반적인 사용에 대한 모범 사례를 정의하고, 암호화 원리에 대한 근본적인 이해를 확립하여 더욱 견고하고 현대적인 접근 방식으로 전환하도록 장려하는 것이다. 본 장에서는 다음을 권장한다.

- 안전하게 실패하고, 진화하는 위협에 적응하며, 미래에도 사용할 수 있는 강력한 암호화 시스템을 구현한다.
- 안전하고 업계 모범 사례에 부합하는 암호화 메커니즘을 활용한다.
- 적절한 접근 제어 및 감사 기능을 갖춘 안전한 암호화 키 관리 시스템을 유지한다.
- 새로운 위험을 평가하고 그에 따라 알고리즘을 조정하기 위해 암호화 환경을 정기적으로 평가한다.
- 애플리케이션의 수명 주기 동안 모든 암호화 자산을 식별하고 보호하기 위해 암호화 사용 사례를 지속적으로 발견하고 관리해야 한다.

이 문서는 일반적인 원칙 및 모범 사례를 설명하는 것 외에도, '부록 C - 암호화 표준의 요구 사항'에 대한 보다 심층적인 기술 정보를 제공한다. 여기에는 이 장의 요구 사항 목적에 따라"승인된"것으로 간주되는 알고리즘 및 모드가 포함되어 있다.

비밀 관리 또는 통신 보안과 같이 별도의 문제를 해결하기 위해 암호화를 사용하는 요구 사항은 본 표준의 다른 장에서 기술될 것이다.

V11.1 암호화 인벤토리 및 문서화

애플리케이션은 데이터 자산의 분류에 따라 이를 보호하기 위한 강력한 암호화 아키텍처로 설계되어야 한다. 모든 데이터를 암호화하는 것은 낭비이며, 아무것도 암호화하지 않는 것은 법적 책임을 초래할 수 있다. 암호화는 일반적으로 아키텍처 설계 또는 상위 수준 설계, 설계 스프린트 (design sprint) 또는 아키텍처 스파이크 (architecture spike) 단계에서 균형 있게 적용되어야 한다. 즉흥적으로 암호화를 설계하거나 사후에 적용하는 것은 초기부터 구축하는 것보다 안전하게 구현하는 데 훨씬 더 많은 비용이 발생하게 된다.

모든 암호화 자산을 정기적으로 식별, 목록화 및 평가하는 것이 중요하다. 수행 방법에 대한 자세한 내용은 부록을 참조할 수 있다..

양자 컴퓨팅의 등장을 대비하여 암호화 시스템의 미래 안정성 (future-proofing) 을 확보하는 것 또한 매우 중요하다. 양자 내성 암호 (Post-Quantum Cryptography; PQC) 는 양자 컴퓨팅에 의해 RSA 및 타원 곡선 암호 (Elliptic Curve Cryptography; ECC) 와 같이 널리 사용되는 알고리즘을 무력화될 것으로 예상됨에 따라 필수적으로 고려되어야 한다.

검증된 PQC 기본 요소 및 표준에 대한 현재 지침은 부록을 참조해야 한다.

#	설명	레벨
11.1.1	'NIST SP 800-57'과 같은 암호화 키 관리 및 키 관리 표준을 따르는 암호화 키 수명 주기에 대한 문서화 된 정책이 있는지 검증해야 한다. 여기에는 키가 과도하게 공유되지 않도록 하는 것이 포함된다 (예: 공유 비밀의 경우 두 개를 초과하는 엔터티와 공유 및 개인 키의 경우 한 개를 초과하는 엔터티에 공유).	2
11.1.2	애플리케이션에서 사용되는 모든 암호화 키, 알고리즘, 인증서를 포함한 암호화 자산 목록 (cryptographic inventory) 이 작성되고, 유지되며, 정기적으로 업데이트되고 있는지 검증해야 한다. 또한, 각 키가 시스템 내에서 사용 가능한 위치와 사용이 제한되는 위치, 해당 키로 보호할 수 있는 데이터 유형과 보호할 수 없는 데이터 유형이 문서화되어 있어야 한다.	2
11.1.3	암호화, 해싱, 서명 작업을 포함하는 시스템 내의 모든 암호화 사용 사례를 식별하기 위해, 암호화 사용 탐지 메커니즘이 적용되는지 검증해야 한다.	3
11.1.4	암호화 자산 목록이 유지 관리되는지 검증해야 한다. 여기에는 미래의 위협에 대응하기 위해 양자 내성 암호 (Post-Quantum Cryptography; PQC) 와 같은 새로운 암호화 표준으로의 마이그레이션 계획을 설명하는 문서화된 계획이 포함되어야 한다.	3

V11.2 안전한 암호화 구현

이 섹션은 애플리케이션의 핵심 암호화 알고리즘 선택, 구현 및 지속적인 관리에 대한 요구 사항을 정의한다. 목표는 현재 표준 (예: NIST, ISO/IEC) 및 모범 사례에 맞춰 강력하고 업계에서 인정하는 암호화 기본 요소만 배포되도록 하는 것이다. 조직은 각 암호화 구성 요소가 동료 검토 (peer-review) 를 거친 증거와 실제 보안 테스트를 기반으로 선택되도록 해야 한다.

#	설명	레벨
11.2.1	암호화 작업에 업계에서 검증된 구현 (라이브러리 및 하드웨어 가속 구현 포함) 이 사용되는지 검증해야 한다.	2
11.2.2	난수, 인증 암호화, MAC, 해싱 알고리즘, 키 길이, 라운드, 암호 및 모드를 언제든지 재구성, 업그레이드 또는 교체하여 암호화 해독으로부터 보호할 수 있도록 애플리케이션이 암호화 유연성 (crypto agility) 을 갖추고 설계되었는지 검증해야 한다. 또한, 키와 비밀번호를 교체하고 데이터를 다시 암호화하는 것도 가능해야 한다. 이를 통해 승인된 PQC 체계 (또는 표준) 의 고신뢰 구현이 널리 사용 가능해지면 PQC 로 원활하게 업그레이드할 수 있다.	2
11.2.3	모든 암호화 기본 구성요소 (primitive) 가 알고리즘, 키 크기 및 구성에 따라 최소 128 비트의 보안강도를 제공하는지 검증해야 한다. 예를 들어, 256 비트 ECC 키는 대략 128 비트의 보안을 제공하며, RSA 는 128 비트의 보안을 달성하기 위해 3072 비트 키를 필요로 한다.	2
11.2.4	민감한 정보가 누출되는 것을 방지하기 위해 모든 암호화 연산은 상수 시간 (constant-time) 으로 수행되어야 하며, 비교, 계산, 반환 과정에서 단락 (short-circuit) 연산이 없는지 검증해야 한다.	3

#	설명	레벨
11.2.5	모든 암호화 모듈이 안전하게 실패하고, 패딩 오라클 공격 (Padding Oracle attacks) 과 같은 취약점을 허용하지 않는 방식으로 오류가 처리되는지 검증해야 한다.	3

V11.3 암호화 알고리즘

AES 및 CHACHA20 을 기반으로 구축된 인증 암호화 알고리즘은 현대 암호화 관행의 근간을 이룬다.

#	설명	래밸
11.3.1	안전하지 않은 블록 모드 (예: ECB) 및 약한 패딩 방식 (예: PKCS#1 v1.5) 가 사용되지 않는지 검증해야 한다.	1
11.3.2	GCM 을 사용하는 AES 와 같이 승인된 암호 및 모드만 사용되는지 검증해야 한다.	1
11.3.3	암호화 된 데이터가 승인된 인증된 암호화 방식을 사용하거나, 승인된 암호화 방식과 승인된 MAC 알고리즘을 조합을 통한 무단 수정으로부터 보호되는지 검증해야 한다.	2
11.3.4	논스 (nonces), 초기화 벡터 및 기타 단일 사용 번호가 둘 이상의 암호화 키 및 데이터 요소 쌍에 사용되지 않는지 검증해야 한다. 생성 방법은 사용되는 알고리즘에 적합해야 한다.	3
11.3.5	암호화 알고리즘과 MAC 알고리즘의 모든 조합이 암호화 후 MAC(Encrypt-then-MAC) 방식으로 동작하는지 검증해야 한다.	3

V11.4 해싱 및 해시 기반 함수

암호화 해시는 디지털 서명, HMAC, 키 유도 함수 (KDF), 무작위 비트 생성 및 비밀번호 저장과 같은 다양한 암호화 프로토콜에서 사용된다. 암호화 시스템의 보안은 사용되는 기본 해시 함수의 강도에 달려 있다. 이 섹션에서는 암호화 작업에서 안전한 해시 함수를 사용하는 요구 사항을 설명한다.

비밀번호 저장 및 암호화 부록의 경우, OWASP 비밀번호 저장 치트 시트도 유용한 맥락과 지침을 제공할 것이다.

#	설명	래밸
11.4.1	디지털 서명, HMAC, KDF 및 무작위 비트 생성을 포함한 일반적인 암호화 사용 사례에 승인된 해시 함수만 사용되는지 검증해야 한다. MD5 와 같이 허용되지 않는 해시 함수는 어떤 암호화 목적으로도 사용해서는 안 된다.	1
11.4.2	비밀번호가 승인된’계산 집약적인 키 유도 함수 (“비밀번호 해싱 함수”라고도 함)’를 사용하여 저장되고, 파라미터 설정이 현재 지침에 따라 구성되었는지 검증해야 한다. 설정은 필요한 보안 수준에 대한 무차별 대입 공격을 충분히 어렵게 만들기 위해 보안과 성능의 균형을 이루어야 한다.	2

#	설명	래밸
11.4.3	데이터 인증 또는 데이터 무결성의 일부로 디지털 서명에 사용되는 해시 함수가 충돌 저항성을 가지며 적절한 비트 길이를 갖는지 검증해야 한다. 충돌 저항성이 필요한 경우, 출력 길이는 최소 256 비트여야 한다. 제 2 역상 (pre-image) 공격 저항성만 필요한 경우, 출력 길이는 최소 128 비트여야 한다.	2
11.4.4	비밀번호로부터 비밀 키를 유도할 때 애플리케이션이 키 스트레칭 (key stretching) 파라미터를 사용하여 승인된 키 유도 함수 (key derivation function) 를 사용하는지 검증해야 한다. 사용 중인 파라미터는 결과적인 암호화 키를 손상시키는 무차별 대입 공격을 방지하기 위해 보안과 성능의 균형을 이루어야 한다.	2

V11.5 무작위 값

암호학적으로 안전한 의사 난수 생성 (CSPRNG) 은 올바르게 구현하기가 매우 어렵다. 일반적으로 시스템 내의 좋은 엔트로피 소스는 과도하게 사용될 경우 빠르게 고갈되지만, 무작위성이 적은 소스는 예측 가능한 키와 비밀로 이어질 수 있다.

#	설명	래밸
11.5.1	예측 불가능해야 하는 모든 난수 및 문자열이 암호학적으로 안전한 의사 난수 생성기 (CSPRNG) 를 사용하여 생성되고 최소 128 비트의 엔트로피를 갖는지 검증해야 한다. UUID 는 이 조건을 충족하지 않는다.	2
11.5.2	사용 중인 난수 생성 메커니즘이 높은 부하 (heavy demand) 에서도 안전하게 작동하도록 설계되었는지 검증해야 한다.	3

V11.6 공개 키 암호화

공개 키 암호화는 여러 당사자 간에 비밀 키를 공유하는 것이 불가능하거나 바람직하지 않은 경우에 사용될 것이다.

이의 일부로, 디피-헬만 (Diffie-Hellman) 및 타원 곡선 디피-헬만 (Elliptic Curve Diffie-Hellman; ECDH) 과 같은 승인된 키 교환 메커니즘이 필요하여 암호화 시스템이 현대 위협에 대해 안전하게 유지되도록 한다. “V12 안전한 통신”챕터는 TLS 에 대한 요구 사항을 제공하므로 이 섹션의 요구 사항은 TLS 이외의 사용 사례에서 공개 키 암호화가 사용되는 상황을 위한 것이다.

#	설명	래밸
11.6.1	키 생성 및 시딩, 디지털 서명 생성 및 검증에 승인된 암호화 알고리즘 및 작동 모드만 사용되는지 검증해야 한다. 키 생성 알고리즘은 알려진 공격에 취약한 안전하지 않은 키 (예: 페르마 인수분해 (Fermat factorization) 에 취약한 RSA 키) 를 생성해서는 안 된다.	2

#	설명	래밸
11.6.2	키 교환에는 디피-헬만과 같은 승인된 암호화 알고리즘이 사용되어야 하며, 안전한 파라미터를 사용하는지 여부를 중점적으로 검증해야 한다. 이는 중간자 공격 (adversary-in-the-middle attacks) 또는 암호화 해독으로 이어질 수 있는 키 설정 프로세스에 대한 공격을 방지할 것이다.	3

V11.7 사용 중 데이터 암호화

데이터가 처리되는 동안 데이터를 보호하는 것은 매우 중요하다. 이를 위해 전체 메모리 암호화, 전송 중 데이터 암호화, 그리고 데이터 사용 직후 가능한 한 빠르게 암호화하는 방식 등의 기술을 적용하는 것이 권장된다.

#	설명	래밸
11.7.1	사용 중인 민감한 데이터를 보호하고 무단 사용자 또는 프로세스의 접근을 방지하기 위해 전체 메모리 암호화가 사용되는지 검증해야 한다.	3
11.7.2	데이터 최소화가 처리 중에 노출되는 데이터의 양을 최소화하고, 사용 직후 또는 가능한 한 빨리 데이터가 암호화되도록 보장하는지 검증해야 한다.	3

참조

더 많은 정보는 다음을 참고한다:

- OWASP 웹 보안 테스팅 가이드: 취약한 암호화 테스트
- OWASP 암호화 저장 치트 시트
- FIPS 140-3
- NIST SP 800-57

V12 보안 통신

제어 목표

이번 장에서는 사용자 클라이언트와 백엔드 서비스 간, 그리고 내부 서비스와 백엔드 서비스 사이에 전송 중인 데이터를 보호하기 위해 마련되어야 할 특정 메커니즘에 관한 요구사항을 포함한다.

이번 장에서 강조되는 주요 개념은 다음과 같다:

- 외부 통신은 물론 가능하다면 내부 통신 또한 암호화되도록 한다.
- 최신 가이드에 따라서 암호화 매커니즘을 구성하고, 권장 알고리즘과 암호를 사용해야 한다.
- 서명된 인증서를 사용해 통신이 허가되지 않은 제 3 자에게 가로채이지 않도록 해야 한다.

게다가 일반적인 원칙과 모범 사례를 제시하는 것 외에도 ASVS 는 부록 C - 암호 표준에서 암호화 강도에 대한 좀 더 세부적인 기술 정보를 제공한다.

V12.1 일반 TLS 보안 가이드

이번 절에서는 TLS 통신을 보호하는 방법에 대한 초기 가이드를 제공한다. 최신 도구를 사용해 TLS 설정을 지속적으로 검토해야 한다.

와일드카드 TLS 인증서 사용 자체가 원래 안전하지 않은 것이 아니지만, 모든 자체 환경 (예를 들어, 운영, 스테이징, 개발, 테스트) 에서 배포된 인증서가 손상되는 것은 이를 사용하는 애플리케이션의 보안 상태가 손상될 수 있다. 그래서 가능하다면 적절한 보호 및 관리와 각각의 환경별로 별도의 TLS 인증서 사용을 취해야 한다.

#	설명	레벨
12.1.1	TLS 1.2 및 TLS 1.3 과 같은 최신 권장 버전의 TLS 프로토콜만 활성화되어 있으며, 그중 최신 버전이 우선되어야 한다.	1
12.1.2	오직 권장되는 암호화 스위트 (cipher suites) 들이 활성화되어 있으며 그 중 가장 강력한 암호화 스위트로 우선 설정되어 있는지 확인한다. 레벨 3 애플리케이션은 순방향 비밀성 (forward secrecy) 을 제공하는 암호화 스위트만 지원해야 한다.	2
12.1.3	인증이나 권한 부여를 위해 인증서 ID 를 사용하기 전에 애플리케이션이 mTLS 클라이언트 인증서가 신뢰할 수 있는지 검증해야 한다.	2
12.1.4	Online Certificate Status Protocol (OCSP) 스테이플링 (Stapling) 과 같은 적절한 인증서 폐지 프로토콜이 활성화되고 설정되어 있는지 확인해야 한다.	3
12.1.5	TLS 핸드셰이크 프로세스 중 Server Name Indication (SNI) 과 같은 민감한 메타 데이터의 노출을 방지하기 위해 애플리케이션 TLS 설정에서 Encrypted Client Hello (ECH) 가 활성화되어 있는지 확인해야 한다.	3

V12.2 외부 서비스와의 HTTPS 통신

애플리케이션이 노출하는 외부 서비스에 대한 모든 HTTP 트래픽은 공적으로 신뢰되는 인증서를 사용해 암호화되어 전송되는지 확인해야 한다.

#	설명	레벨
12.2.1	TLS 는 클라이언트와 외부 HTTP 기반 서비스 간의 모든 연결에 사용되며, 안전하지 않거나 암호화 되어 있지 않은 통신에 대체 동작 (fall back) 이 발생하지 않는지 확인해야한다.	1
12.2.2	외부 서비스가 공적으로 신뢰된 TLS 인증서를 사용하는지 확인해야 한다.	1

V12.3 일반 서비스 간의 통신 보안

내부와 외부 서버 통신은 HTTP 외에도 다양하게 이루어지며 다른 시스템과의 연결은 가능한 TLS 를 사용하여 보호되어야 한다.

#	설명	레벨
12.3.1	애플리케이션간의 모니터링 시스템, 관리 도구, 원격 엑세스 및 SSH, 미들웨어, 데이터베이스, 메인프레임, 파트너 시스템이나 외부 API 를 포함한 모든 인바운드와 아웃바운드 통신은 TLS 같은 암호화 프로토콜이 사용되는지 확인해야 한다. 이 서버는 안전하지 않거나 암호화 되어있지 않은 프로토콜에 대체 동작 (fall back) 이 발생하지 않아야 한다.	2
12.3.2	TLS 클라이언트가 통신하기 전에 TLS 서버로 부터 받은 인증서를 검증하는지 확인해야 한다.	2
12.3.3	그리고 TLS 또는 애플리케이션 내의 내부 HTTP 기반 서비스간에 사용된 다른 적절한 전송 암호화 매커니즘이 안전하지 않거나 암호화 되어있지 않은 통신에 대체 동작 (fall back) 이 발생하지 않는지 확인해야 한다.	2
12.3.4	내부 서버간에 TLS 통신이 신뢰된 인증서를 사용하는지 확인해야 한다. 내부 인증서나 자체 서명 (self-signed) 인증서를 사용할 경우, 이를 사용하는 서비스는 오직 특정 내부 인증기관 (CA) 과 자체 서명 인증만을 신뢰하게 설정해야 한다.	2
12.3.5	시스템 내부에서 통신하는 서비스 (intra-service communication) 는 각 엔드포인트를 검증하기 위해 강력한 인증을 사용해야 한다. 공용키 (public-key) 기반 구조와 재전송 공격 (replay attacks) 에 저항하는 매커니즘을 사용하여 신원보장을 위해 TLS 클라이언트 인증과 같은 강력한 인증 방식을 사용해야 한다. 마이크로서비스 아키텍쳐의 경우 인증서 관리를 간소화 하고 보안을 강화하기 위해 서비스 매시 (service mesh) 를 사용하는 것을 고려해야 한다.	3

참조

더 많은 정보는 다음을 참고한다:

- OWASP - TLS 참고자료
- Mozilla’s 서버측 TLS 구성 가이드
- Mozilla’s 일반적으로 알려진 좋은 TLS 구성 툴.
- O-Saft - TLS 구성 검증을 위한 OWASP 프로젝트

V13 설정

제어 목표

인터넷상에서 사용하기 위해 애플리케이션의 기본 설정은 안전해야 한다.

이 챕터에서는 이를 달성하기 위해 개발, 빌드, 배포 과정에서 적용되는 설정들을 비롯한 여러 설정에 대한 가이던스를 제공한다.

이 주제는 데이터 유출 방지, 컴포넌트 간의 안전한 의사소통, 비밀 정보 보호 등을 다룬다.

V13.1 설정 문서

이 섹션은 서비스 비접근성으로 인한 가용성 손실을 방지하는 기술을 포함하여, 애플리케이션이 내부, 외부 서비스와 통신을 하기 위한 요구사항들의 문서화에 대해 간략히 서술하고 있다. 또한 비밀 정보와 관련된 문서화에 대해서도 다루고 있다.

#	설명	레벨
13.1.1	애플리케이션의 모든 통신 요구사항이 문서화 되어 있는지를 검증한다. 이는 애플리케이션이 의존하는 외부 서비스나, 사용자가 애플리케이션이 연결할 외부 위치를 제공하는 경우들을 포함한다.	2
13.1.2	애플리케이션이 사용하는 각각의 서비스들에 대해, 문서가 최대 동시 연결 수 (연결 풀 제한) 를 정의하고 있는지, 그리고 그 한계에 다다랐을 때 폴백 (fallback) 이나 복구 메커니즘 등을 포함하여, DoS 공격을 방지하기 위해 애플리케이션이 어떻게 동작하는지를 검증한다.	3
13.1.3	애플리케이션 문서가 모든 외부 시스템이나 애플리케이션이 사용하는 서비스 (예시: 데이터베이스, 파일 핸들, 스레드, HTTP 연결) 들에 대한 자원-관리 전략을 정의하고 있는지 검증한다. 이는 자원-해제 프로시저, 시간초과 설정, 오류 핸들링을 포함하며, 재시도 로직이 구현된 경우 재시도 제한과 지연, 백-오프 알고리즘을 명시해야 한다. 동기 HTTP 요청-응답 명령에 대해서는 짧은 타임아웃에 더해 재시도의 비활성화 혹은 연쇄적인 지연과 자원 고갈을 방지하기 위한 재시도의 엄격한 제한을 의무화해야 한다.	3
13.1.4	애플리케이션 문서가 조직의 위협 모델과 사업적 요구사항에 기반하여, 애플리케이션의 보안과 비밀 정보들의 주기적 갱신 일정에 치명적인 비밀 정보들을 정의하고 있는지 검증한다.	3

V13.2 백엔드 통신 설정

애플리케이션들은 여러 API, 데이터베이스 또는 다른 컴포넌트 등과 함께 상호작용한다. 이것들은 애플리케이션의 표준 접근 제어 메커니즘에 포함되어 있기보다 애플리케이션 내부에 존재한다고 고려되거나, 혹은 완전 외부에 존재한다고 여겨진다. 각각의 경우 모두 이 컴포넌트들과 안전하게 상호작용하고, 또 유사시 그 설정을 보호하는 도록 애플리케이션을 설정하는 것이 필수적이다.

참고: “안전한 통신”챕터는 전송의 암호화를 위한 가이던스를 제공한다.

#	설명	레벨
13.2.1	API 나 미들웨어와 같은 애플리케이션의 표준 사용자 세션 메커니즘을 지원하지 않는 백엔드 애플리케이션 컴포넌트와 데이터 계층 사이의 통신이 인증되어 있는지를 검증한다. 인증은 반드시 개별적인 서비스 계정과 단기간 토큰, 또는 인증서 기반 인증을 사용하고 비밀번호, API 키, 또는 접근 권한이 부여된 공유 계정 등의 자격 증명을 변경하지 않아선 안 된다.	2
13.2.2	로컬 또는 운영체제 서비스, API, 미들웨어, 그리고 데이터 계층을 포함한 백엔드 애플리케이션 컴포넌트 간의 통신이 최소 필요 권한이 할당된 계정들로 수행되는지 검증한다.	2

#	설명	레벨
13.2.3	만약 자격 증명이 서비스 인증을 위해 필요하다면, 소비자로부터 사용되는 그 자격 증명이 기본 자격 증명이 아님을 검증한다. (예시: root/root 또는 admin/admin)	2
13.2.4	얼로우리스트가 애플리케이션의 통신이 허가된 외부 리소스나 시스템을 정의하기 위해 사용되는지 검증한다. (예시: 아웃바운드 요청, 데이터 로드, 파일 접근). 이 얼로우리스트는 애플리케이션 계층, 웹 서버, 방화벽 또는 다른 여러 계층의 조합으로 구현할 수 있다.	2
13.2.5	웹이나 애플리케이션 서버에 요청을 보내거나 데이터나 파일을 로드 할 수 있는 리소스 또는 시스템의 얼로우리스트가 설정되어 있는지를 검증한다.	2
13.2.6	애플리케이션이 별도의 서비스와 연결할 때, 각 연결이 최대 병렬연결, 최대 연결 도달 시의 동작, 연결 타임아웃, 재시도 전략 등 문서화된 설정에 따르는지 검증한다.	3

V13.3 비밀 정보 관리

비밀 정보 관리는 애플리케이션에서 사용되는 데이터를 보호하기 위해 필수적인 설정이다. 암호에 대한 특정 요구사항들은 “암호”챕터에서 찾아볼 수 있고, 이 섹션에서는 비밀 정보들의 관리와 취급의 측면에 초점을 둔다.

#	설명	레벨
13.3.1	키 볼트와 같은 비밀 정보 관리 솔루션이 백엔드 비밀 정보들을 만들고 저장하고, 접근을 제어하고, 파기하기 위해 사용되고 있는지 검증한다. 이는 비밀번호, 키 재료, 데이터베이스와 서드 파티 시스템의 통합, 시간-기반 토큰을 위한 키와 시드, 그리고 API 키들을 포함할 수 있다. 비밀 정보들은 애플리케이션 소스 코드 내부나, 빌드 아티팩트에 포함되어서는 안 된다. L3 애플리케이션에 대해서는, 반드시 HSM 과 같은 하드웨어 기반 솔루션을 사용해야 한다.	2
13.3.2	비밀 정보 자원들에 대한 접근이 최소 권한의 원칙을 고수하는지 검증한다.	2
13.3.3	보안 모듈에 대한 외부의 노출로부터 안전하게 키 재료를 관리하고 보호하기 위해 모든 암호 연산이 고립된 보안 모듈 (볼트나, 하드웨어 보안 모듈 등) 을 사용하여 수행되는지 검증한다.	3
13.3.4	애플리케이션의 문서를 기반으로 비밀 정보들이 만료되고 갱신되도록 설정되었는지 검증한다.	3

V13.4 의도하지 않은 정보 노출

운영환경 설정은 불필요한 데이터가 공개되는 것을 피하기 위해 강화되어야 한다. 의도하지 않은 정보의 노출은 심각한 위협으로 평가되지는 않지만 자주 다른 취약점들과 연계된다. 이러한 문제들이 기본적으로 존재하지 않으면 애플리케이션에 대한 공격 난이도를 높일 수 있다.

예를 들어 서버단 컴포넌트의 버전 정보를 숨긴다고 해서 모든 컴포넌트를 패치할 필요가 없어지는 것은 아니지만, 또 폴더 목록 보기를 비활성화한다고 인증 제어를 사용하거나 공용 폴더로부터 파일을 제거할 필요가 없어지는 것은 아니지만, 이는 공격의 난이도를 높인다.

#	설명	레벨
13.4.1	애플리케이션이.git 이나.svn 폴더 같은 소스 코드 관리 메타데이터 없이 배포되었는지, 또는 이 폴더들이 외부적으로든 애플리케이션 자체에서든 접근할 수 없도록 배포되었는지 검증한다.	1
13.4.2	실제 운영환경에서 디버깅 기능 노출이나 정보 누출을 방지하기 위해, 모든 컴포넌트에서 디버그 모드가 비활성화되어 있는지 검증한다.	2
13.4.3	분명히 의도된 것이 아닌 이상 웹 서버가 폴더 목록을 클라이언트에게 노출하지 않는지 검증한다.	2
13.4.4	잠재적인 정보 누출을 방지하기 위해 실제 운영환경에서 HTTP TRACE 메소드 사용이 지원되지 않는지 검증한다.	2
13.4.5	분명히 의도된 것이 아닌 이상 문서 (내부 API 같은) 와 모니터링 엔드포인트가 노출되지 않는지 검증한다.	2
13.4.6	애플리케이션이 백엔드 컴포넌트의 자세한 버전 정보를 노출하지 않는지 검증한다.	3
13.4.7	웹 계층이 의도적이지 않은 정보, 설정, 소스코드 누출을 방지하기 위해 특정 파일 확장자만 서비스하도록 설정되었는지 검증한다.	3

참조

더 많은 정보는 다음을 참고한다:

- OWASP Web Security Testing Guide: Configuration and Deployment Management Testing

V14 데이터 보호

제어 목표

애플리케이션은 모든 사용 패턴과 사용자 행동을 고려할 수 없으므로, 클라이언트 기기의 민감한 데이터에 대한 비인가 접근를 제한하는 기능 구현이 권장된다.

이 장에서는 보호해야 하는 데이터, 보호해야 하는 방법, 구현해야 하는 구체적인 메커니즘 또는 피해야 할 실수들과 관련된 요구사항이 포함된다.

데이터 보호에서 또 다른 고려 사항은 데이터 대량 추출, 무단 변경, 과도한 자원 사용이다. 각 시스템의 요구사항은 매우 다를 가능성이 높으므로, 무엇이“비정상”인지 판단하려면 위협 모델과 비즈니스 리스크를 고려해야 한다. ASVS 관점에서 이러한 문제들을 탐지하는 것은“보안 로그와 오류 처리”장에서 다루며, 한도 설정은“검증과 비즈니스 로직”장에서 다룬다.

V14.1 데이터 보호 문서화

데이터를 보호하기 위한 핵심 조건은 어떤 데이터를 민감한 데이터로 간주해야 하는지 분류하는 것이다. 민감도에는 몇가지 수준이 있을 수 있으며, 각 수준별로 데이터를 보호하기 위해 필요한 통제는 서로 다르다.

애플리케이션이 민감한 개인정보를 저장, 사용, 전송의 접근 방식에 영향을 미치는 다양한 개인정보보호 규정 및 법률이 존재한다. 이 절에서는 이러한 유형의 테이터 보호 또는 개인정보보호 법률을 반복해서 다루지 않지만, 민감한 데이터를 보호하기 위한 핵심기술의 고려사항에 초점을 맞춘다. 현지 법률 및 규정을 확인하고, 필요에 따라 자격을 갖춘 개인정보보호 전문가나 변호사와 상의한다.

#	설명	레벨
14.1.1	애플리케이션이 생성하고 처리하는 모든 민감한 데이터가 식별되고 보호 수준으로 분류되었는지 확인한다. 여기에는 단순히 인코딩되어 있고 쉽게 디코딩할 수 있는 데이터, 예를 들어 Base64 문자열이나 JWT 내부 평문 페이로드가 포함된다. 보호 수준은 애플리케이션이 준수해야 하는 모든 데이터 보호 및 개인정보 보호 관련 규정과 표준을 반드시 고려해야 한다.	2
14.1.2	모든 민감한 데이터의 보호 수준에 대해 문서화된 요구사항이 있는지 확인한다. 반드시 여기에는 (이에 한정되지 않지만) 일반적인 암호화, 무결성 검증, 보존 기간, 데이터 로그 기록 방식, 로그 내 민감한 데이터에 대한 접근 제어, 데이터베이스 암호화 수준, 사용할 개인정보 보호 및 개인정보 보호 강화 기술, 그리고 기타 기밀성 요구사항들 관련하여 포함되어야 한다.	2

V14.2 일반 데이터 보호

이 절에는 데이터 보호와 관련하여 다양하게 실제로 적용가능한 요구사항들이 포함되어 있다. 대부분은 의도하지 않은 데이터 유출과 같은 특정 문제를 다루지만, 각 데이터 항목에 요구되는 보호 수준에 따라 보호 통제를 구현해야 하는 일반적인 요구사항도 있다.

#	설명	레벨
14.2.1	민감한 데이터는 HTTP 메시지 본문이나 헤더 필드로만 서버에 전송되고, URL 과 쿼리 문자열에는 API 키나 세션토큰처럼 민감한 정보가 포함되지 않도록 검증한다.	1
14.2.2	서버 구성 요소에서 로드밸런서나 애플리케이션 캐시와 같은 민감한 데이터가 캐시되지 않게 하거나, 사용 후 해당 데이터를 안전하게 삭제하도록 검증한다.	2
14.2.3	정의된 민감한 데이터가 애플리케이션의 통제를 벗어난 곳에서 원치 않는 수집을 예방하기 위해 신뢰할 수 없는 대상 (예: 사용자 추적기) 에게 전송되지 않도록 검증한다.	2
14.2.4	암호화, 무결성 검증, 보존, 데이터 로그 기록 방식, 로그 내 민감한 데이터 접근 제어, 개인정보 보호 및 개인정보 보호 강화 기술과 관련된 민감한 데이터 통제가, 해당 데이터 보호 수준에 대한 문서에 정의된 대로 구현되었는지 검증한다.	2
14.2.5	캐시 메커니즘이 해당 리소스에 대해 예상되는 콘텐츠 유형을 가진 응답만 캐시하고, 민감하거나 동적인 콘텐츠는 캐시하지 않도록 구성되어 있는지 검증한다. 존재하지 않는 파일에 접근했을 때 다른 유효한 파일을 반환하는 대신 웹 서버는 404 또는 302 응답을 반환해야 한다. 이는 Web Cache Deception 공격을 예방한다.	3

#	설명	레벨
14.2.6	애플리케이션 기능에 필요한 최소한의 민감한 데이터만 반환을 검증한다. 예를 들어, 전체 신용 카드 번호가 아닌 일부 숫자만 반환해야 한다. 만약 전체 데이터가 필요한 경우, 사용자가 구체적으로 조회하지 않는 한 사용자 인터페이스에서는 마스킹하는 것이 권장된다.	3
14.2.7	민감한 정보는 데이터 보존 분류 기준에 따라 관리되어야 하고, 오래되었거나 불필요한 데이터는 사전에 정의된 작업 또는 상황에 따라 자동으로 삭제되도록 검증한다.	3
14.2.8	사용자가 제출한 파일의 메타데이터에서 사용자가 저장에 동의하지 않는 한 민감한 정보는 제거를 검증한다.	3

V14.3 클라이언트 측 데이터 보호

이 절에서는 클라이언트 또는 사용자 에이전트 측에서 애플리케이션의 데이터가 특정 방식으로 유출되는 것을 방지하기 위한 요구사항을 다룬다.

#	설명	레벨
14.3.1	클라이언트나 세션이 종료된 후에 브라우저 DOM 과 같은 클라이언트 스토리지에서 인증된 데이터를 삭제되는지 검증한다. 이를 위해‘Clear-Site-Data’HTTP 응답 헤더 필드를 사용할 수 있으나, 세션 종료 시 서버 연결이 불가능한 경우에도 클라이언트 측에서 해당 데이터를 삭제하는 것이 권장된다.	1
14.3.2	애플리케이션이 브라우저에서 민감한 데이터가 캐시되지 않도록, (Cache-Control: no-store) 충분한 캐시 방지 HTTP 응답 헤더 필드를 설정을 검증한다.	2
14.3.3	브라우저 저장소 (localStorage, sessionStorage, IndexedDB, 쿠키) 에 저장되는 데이터에는 세션 토큰을 제외한 민감 데이터가 포함되지 않도록 검증한다.	2

참조

자세한 내용은 다음을 참고한다:

- Consider using the Security Headers website to check security and anti-caching header fields
- Documentation about anti-caching headers by Mozilla
- OWASP Secure Headers project
- OWASP Privacy Risks Project
- OWASP User Privacy Protection Cheat Sheet
- Australian Privacy Principle 11 - Security of personal information
- European Union General Data Protection Regulation (GDPR) overview
- European Union Data Protection Supervisor - Internet Privacy Engineering Network
- Information on the “Clear-Site-Data”header

- White paper on Web Cache Deception

V15 보안 코딩 및 아키텍처

제어 목표

많은 ASVS 요구사항은 인증 및 권한 부와 같은 특정 보안 영역에 관련되거나, 로깅 및 파일 처리와 같은 특정 애플리케이션 기능과 관련되어 있다.

이번 장에서는 애플리케이션을 설계하고 개발할 때 고려해야 하는 일반적인 보안 요구사항을 제시한다. 이러한 요구사항은 클린 아키텍처와 코드 품질뿐만 아니라, 애플리케이션 보안을 위해 필요한 구체적인 아키텍처 설계 및 코드 작성 관행에도 중점을 둔다.

V15.1 보안 코딩 및 아키텍처 문서화

보안적이고 방어 가능한 아키텍처를 수립하기 위한 많은 요구사항은, 특정 보안 통제의 구현 및 애플리케이션에서 사용되는 구성요소에 대한 의사결정이 명확하게 문서화되어 있는지에 따라 달려 있다.

이 장에서는 위험한 기능 (dangerous functionality) 을 포함하거나 위험한 구성요소 (risky components) 로 간주되는 항목을 식별하는 것을 포함하여, 문서화에 필요한 요구사항을 설명한다.

위험한 기능을 가진 구성요소는 내부적으로 개발되었거나 서드파티 구성요소일 수 있으며, 신뢰할 수 없는 데이터의 역직렬화 (deserialization), 원시 파일 또는 바이너리 데이터 파싱, 동적 코드 실행, 메모리 직접 조작 등의 작업을 수행할 수 있다. 이러한 유형의 작업에서 발생하는 취약점은 애플리케이션을 손상시키고 그 기반 인프라를 노출시킬 수 있는 높은 위험을 초래한다.

위험한 구성요소는 개발 프로세스나 기능에 대한 보안 통제가 없거나 부실하게 구현된 서드파티 라이브러리를 의미하며, 내부적으로 개발되지 않은 구성요소를 포함한다. 예를 들어, 유지 관리가 제대로 이루어지지 않거나, 지원이 종료되었거나, 생명주기가 끝났거나, 심각한 취약점 이력이 있는 구성요소가 이에 해당한다.

또한 이 장에서는 서드파티 구성요소에 취약점이 발견됐을 때, 이를 얼마나 빨리 해결할 것인지에 대한 대응 기한을 명확히 정하는 것이 중요하다는 점을 강조한다.

#	설명	레벨
15.1.1	애플리케이션 문서에 취약점이 존재하는 서드파티 구성요소 버전 및 일반적인 라이브러리 업데이트에 대해, 해당 구성요소로부터 발생할 수 있는 위험을 최소화하기 위한 위험 기반 대응 기한이 정의되어 있는지 확인해야 한다.	1
15.1.2	사용 중인 모든 서드파티 라이브러리에 대해 소프트웨어 자재 명세서 (SBOM) 와 같은 인벤토리가 유지되고 있으며, 해당 구성요소가 사전에 정의된 신뢰할 수 있고 지속적으로 관리되는 저장소에서 제공되는지 확인해야 한다.	2

#	설명	레벨
15.1.3	애플리케이션 문서에 시간 소모가 크거나 리소스 소비가 많은 기능이 식별되어 있으며, 해당 기능의 과도한 사용으로 인해 가용성이 손실되지 않도록 하는 방법과, 응답 생성이 소비자의 타임아웃보다 오래 걸리는 상황을 방지하는 방법이 포함되어 있는지 확인해야 한다. 가능한 방어 전략에는 비동기 처리, 큐 (queue) 사용, 사용자 및 애플리케이션 단위의 병렬 처리 제한 등이 있다.	2
15.1.4	애플리케이션 문서에 위험한 구성요소로 간주되는 서드파티 라이브러리가 강조되어 있는지 확인해야 한다.	3
15.1.5	애플리케이션 문서에 위험한 기능을 사용하는 애플리케이션의 부분이 강조되어 있는지 확인해야 한다.	3

V15.2 보안 아키텍처 및 종속성

이 장에서는 종속성 관리 (dependency management) 를 통해 위험하거나 구식이거나 보안에 취약한 종속성과 구성요소를 다루기 위한 요구사항을 포함한다.

또한, 앞 장에서 정의한“위험한 동작”이나“위험한 구성요소”의 사용으로 인한 영향을 줄이기 위해 샌드박싱, 캡슐화, 컨테이너화, 네트워크 격리와 같은 아키텍처 수준의 기법을 활용하는 방법도 포함된다. 이러한 기법은 리소스을 과도하게 사용하는 기능 때문에 발생할 수 있는 가용성 저하를 예방하는 데에도 도움이 된다.

#	설명	레벨
15.2.1	애플리케이션이 문서화된 업데이트 및 대응 기한을 위반하지 않은 구성요소만 포함하고 있는지 확인해야 한다.	1
15.2.2	애플리케이션이 시간 소모가 크거나 리소스 소비가 많은 기능의 과도한 사용으로 인한 가용성 손실을 방지하기 위해, 해당 기능에 대한 문서화된 보안 결정 및 전략을 기반으로 방어 조치를 구현하고 있는지 확인해야 한다.	2
15.2.3	운영 환경에 애플리케이션 기능 수행에 필요한 기능만 포함되어 있으며, 테스트 코드, 샘플 코드, 개발용 기능 등 불필요한 기능이 노출되지 않도록 구성되어 있는지 확인해야 한다.	2
15.2.4	서드파티 구성요소 및 그 모든 전이 종속성 (transitive dependencies) 이 내부 또는 외부의 예상된 저장소에서 가져며, 의존성 혼동 (dependency confusion) 공격의 위험이 없는지 확인해야 한다.	3
15.2.5	애플리케이션이 위험한 기능을 포함하거나 위험한 구성요소를 사용하는 부분에 대해 샌드박싱, 캡슐화, 컨테이너화, 네트워크 수준 격리와 같은 추가적인 보호 기법을 구현하여, 애플리케이션의 한 부분이 침해되더라도 공격자가 다른 영역으로 확산하지 못하도록 지연 및 차단하고 있는지 확인해야 한다.	3

V15.3 방어적 (Defensive) 코딩

이 장에서는 특정 프로그래밍 언어에서 비안전한 코딩 패턴을 사용할 때 발생할 수 있는 취약점 유형을 다룬다. 여기에는 타입 혼용 (type juggling), 프로토타입 오염 (prototype pollution) 등이 포함되며, 일부는 모든 언어에 적용되지 않지만, 다른 일부는 특정 언어 또는 프레임워크가 HTTP 파라미터와 같은 기능을 처리하는 방식과 관련될 수 있으며, 언어별 대응 방안이 존재할 수 있다. 또한 애플리케이션 업데이트 시 암호학적 검증을 수행하지 않는 것과 관련된 위험도 포함된다.

또한, 객체를 사용하여 데이터를 표현하고 외부 API 를 통해 이를 수신하거나 반환하는 경우에 발생할 수 있는 위험도 고려한다. 이 경우, 애플리케이션은 사용자 입력에 의해 수정되어서는 안 되는 데이터 필드가 변경되지 않도록 보장해야 하며 (mass assignment), API 는 반환할 데이터 필드를 선택적으로 제한해야 한다. 데이터 필드 접근이 사용자의 권한에 따라 달라지는 경우, 이는 권한 부여 챕터의 필드 수준 접근 통제 요구사항과 함께 고려되어야 한다.

#	설명	레벨
15.3.1	애플리케이션이 데이터 객체에서 필요한 필드만 선택적으로 반환하고 있는지 확인해야 한다. 예를 들어, 일부 필드는 사용자에게 노출되어서는 안 되므로 전체 데이터 객체를 반환해서는 안 된다.	1
15.3.2	애플리케이션 백엔드에서 외부 URL 로 요청을 보낼 때, 의도된 기능이 아닌 경우 리디렉션을 따르지 않도록 구성되어 있는지 확인해야 한다.	2
15.3.3	애플리케이션이 각 컨트롤러 및 액션 단위로 허용된 필드를 제한함으로써, 대량 할당 (mass assignment) 공격에 대한 방어가 구현되어 있는지 확인해야 한다. 예를 들어, 특정 동작에 포함되지 않은 필드 값을 삽입하거나 수정할 수 없어야 한다.	2
15.3.4	모든 프록시 및 미들웨어 구성요소가 사용자의 원래 IP 주소를 신뢰 가능한 데이터 필드를 통해 정확하게 전달하고 있으며, 애플리케이션과 웹 서버가 이 값을 로그 기록이나 속도 제한 (rate limiting) 과 같은 보안 결정에 사용하는지 확인해야 한다. 단, 동적 IP, VPN, 방화벽 등으로 인해 원래 IP 조차 완전히 신뢰할 수 없는 점도 고려해야 한다.	2
15.3.5	애플리케이션이 변수의 타입이 올바른지 명시적으로 확인하고, 엄격한 동등성 비교 및 연산자를 사용하여 타입 혼용 (type juggling) 또는 타입 혼동 (type confusion) 으로 인한 취약점을 방지하고 있는지 확인해야 한다.	2
15.3.6	JavaScript 코드가 프로토타입 오염 (prototype pollution) 을 방지할 수 있도록 작성되어 있는지 확인해야 한다. 예를 들어, 객체 리터럴 대신 `Set()` 또는 `Map()` 을 사용하는 방식이 권장된다.	2
15.3.7	애플리케이션이 HTTP 파라미터 오염 (HTTP parameter pollution) 공격에 대한 방어를 구현하고 있는지 확인해야 한다. 특히, 애플리케이션 프레임워크가 요청 파라미터의 출처 (쿼리 스트링, 본문, 쿠키, 헤더) 를 구분하지 않는 경우 이를 고려해야 한다.	2

V15.4 안전한 동시성

경쟁 상태 (race condition), 검사-사용 시점 불일치 (TOCTOU), 교착 상태, 라이락 (livelock), 스레드 기아 (thread starvation), 부적절한 동기화 등 동시성 이슈는 예측 불가능한 동작과 보안 위험을 초래할 수 있다. 이 장에서는 이러한 위

험을 완화하기 위한 기법과 전략을 제시한다.

#	설명	레벨
15.4.1	다중 스레드 코드에서 캐시, 파일, 다수 스레드가 접근하는 메모리 내 객체 등 공유 객체에 접근할 때, 스레드 세이프 (thread-safe) 타입과 락·세마포어 같은 동기화 메커니즘을 사용하여 경쟁 상태와 데이터 손상을 피하고 안전하게 접근하는지 확인해야 한다.	3
15.4.2	리소스의 존재 여부나 권한과 같은 상태 검사와, 그 검사 결과에 의존하는 동작이 단일 (atomic) 연산으로 수행되어 TOCTOU 경쟁 상태를 방지하는지 확인해야 한다. (예: 파일을 열기 전에 존재 여부를 확인하는 작업과 파일 열기 동작을 분리하지 않기, 접근 권한 검증과 권한 부여를 동일 트랜잭션/원자적 구간에서 수행하기)	3
15.4.3	스레드가 서로를 기다리거나 무한 재시도에 빠지지 않도록 일관된 락 사용을 보장하고, 락 관리 로직을 리소스 관리 책임 코드 내부에 유지하여 외부 클래스 및 코드가 우발적·악의적으로 락을 변경하지 못하도록 하는지 확인해야 한다.	3
15.4.4	**스레드 풀** 활용 등 **리소스 할당 정책**을 통해 공정한 리소스 접근을 보장하고, 우선순위가 낮은 스레드도 합리적 시간 내 진행될 수 있게 하여 **스레드 기아**를 방지하는지 확인해야 한다.	3

참조

자세한 내용은 다음을 참조한다:

- OWASP Prototype Pollution Prevention Cheat Sheet
- OWASP Mass Assignment Prevention Cheat Sheet
- OWASP CycloneDX Bill of Materials Specification
- [OWASP Web Security Testing Guide: Testing for HTTP Parameter Pollution](https://owasp.org/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/07-Input_Validation_Testing_for_HTTP_Parameter_Pollution

V16 보안 로깅과 오류 처리

제어 목표

보안 로그는 오류나 성능 로그와 달리 인증 결정, 접근 제어 결정, 그리고 입력값 확인이나 비즈니스 로직 확인 등의 보안 제어 우회 시도와 같은 보안과 관련된 이벤트를 기록하는데 사용된다. 보안 로그의 목적은 SIEM 과 같은 분석 도구에 고신뢰성의 구조화된 데이터를 제공함으로써 감지, 응답, 조사를 돕는 것이다.

로그는 법적으로 요구되지 않는 이상, 민감한 개인 정보를 포함해서는 안 되고, 모든 로그 데이터는 높은 가치의 자원으로서 보호되어야 한다. 로깅이 프라이버시나 시스템 보안을 훼손해서는 안 된다. 애플리케이션은 오류가 나더라도 안전하게 발생해야 하고, 불필요한 공개나, 혼란을 방지해야 한다.

더욱 자세한 구현 지침은 참조 섹션의 OWASP 치트 시트를 참조한다.

V16.1 보안 로깅 문서화

이번 섹션은 애플리케이션 스택 전체에 걸친 완전하고 명확한 로깅 내역을 보장한다. 이는 효과적인 보안 모니터링, 사고 대응, 그리고 컴플라이언스에 필수적이다.

#	설명	레벨
16.1.1	애플리케이션의 기술 스택의 각 계층에서 수행된 로깅에 대해, 어떤 이벤트들이 로그로 남았는지, 로그의 형식이 무엇인지, 로그가 어디에 저장되었는지, 어떻게 사용되는지, 로그에 대한 접근이 어떻게 제어되는지, 그리고 로그가 얼마나 보관되는지에 대해 문서화한 목록이 존재하는지 검증한다.	2

V16.2 일반 로깅

이번 섹션은 보안 로그가 일관적으로 구조화되어 있고, 필요한 메타데이터들을 포함하고 있음을 보장하기 위한 요구사항들을 제공한다. 목표는 분산 시스템들과 도구들에서 로그를 기계가 읽고, 분석할 수 있도록 만드는 것이다.

보안 이벤트는 일반적으로 민감한 데이터와 연관된다. 만약 데이터가 주의하지 않고 로깅 되었다면, 그 로그들 자체가 기밀로 분류되어 암호화 요구사항, 더 엄격한 보관 정책, 그리고 감사 시 잠재적 공개 위험이 발생한다.

그러므로 정말 필요한 것들만 로깅하고, 또 로그를 다른 민감한 자산들처럼 다루는 것이 매우 중요하다.

아래의 요구사항들은 로깅의 메타데이터, 동기화, 형식, 그리고 제어에 대해 기초적인 요구사항들을 확립한다.

#	설명	레벨
16.2.1	이벤트가 발생했을 때 그 타임라인을 자세히 조사할 수 있도록 각 로그의 항목이 필수적인 메타데이터 (언제, 어디서, 누가, 무엇을 등) 을 포함하는지 검증한다.	2
16.2.2	모든 로깅 컴포넌트의 시간 소스가 동기화되어 있는지, 그리고 보안 이벤트 메타데이터에서 타임스탬프가 UTC 를 사용하거나 혹은 명시적인 시간대 오프셋을 포함하는지 검증한다. UTC 는 분산 시스템 전체에 걸친 일관성을 보장하고, 일광 절약 시간제 전환 시의 혼란을 방지하기 위해 추천된다.	2
16.2.3	애플리케이션이 로그를 저장만 하는지, 혹은 로그 목록에서 문서화 된 파일이나 서비스로 브로드캐스트하는지 검증한다.	2
16.2.4	사용 중인 로그 처리기가 로그를 읽고, 또 상호 연관 지을 수 있는지 검증한다. 가능하다면 공통 로깅 형식을 사용하는 것이 좋다.	2
16.2.5	민감한 데이터를 로깅할때, 애플리케이션이 데이터의 보호 수준을 기반으로 로깅을 수행하는지 검증한다. 예를 들어, 자격 증명이나, 지불 상세 정보 등의 특정 데이터는 로깅 해서는 안 된다. 또한 세션 토큰 등의 데이터는 전체 혹은 일부분이 해싱된, 마스킹 된 상태로 로깅 되어야 한다.	2

V16.3 보안 이벤트

이번 섹션에서는 애플리케이션 내 보안 관련 이벤트들의 로깅을 위한 요구사항들을 정의한다. 이러한 이벤트들을 수집하는 것은 의심스러운 행위를 탐지하고, 조사를 돕고, 컴플라이언스 의무를 달성하기 위해 매우 중요하다.

이번 섹션은 로깅 되어야 할 이벤트들에 대해 간략히 설명하고 있지만, 모든 세부 사항을 다 다루려는 것은 아니다. 각 애플리케이션은 자신만의 위험 요소들과 운영의 맥락을 가진다.

ASVS 는 보안 이벤트의 로깅을 범위에 포함하는 반면, 경보 (alerting) 와 상관분석 (correlation, 예시: SIEM 규칙이나 모니터링 인프라스트럭처) 은 범위에 포함하지 않는다. 이는 운영 및 모니터링 시스템에서 다루어진다.

#	설명	레벨
16.3.1	성공한 시도와 실패한 시도를 포함하여 모든 인증 명령을 로깅 하는지 검증한다. 인증의 종류나, 사용한 인증 요소 등의 추가적인 메타데이터 또한 수집해야 한다.	2
16.3.2	실패한 인가 시도들도 로깅 하는지 검증한다. L3 에 대해서는 반드시 민감한 데이터가 접근되었을 때를 포함하여 (민감한 데이터 자체는 제외하고) 모든 인가 결정들의 로그를 포함해야 한다.	2
16.3.3	애플리케이션이 이 문서에서 정의한 보안 이벤트들을 로깅하고, 또한 입력값 확인, 비즈니스 로직, 그리고 자동화 방지 등의 보안 제어를 우회하려는 시도들도 로깅 하는지 검증한다.	2
16.3.4	애플리케이션이 백엔드 TLS 오류와 같은 예상치 못한 오류나 보안 제어 오류들을 로깅 하는지 검증한다.	2

V16.4 로그 보호

로그는 중요한 포렌식 아티팩트이며 반드시 보호해야 한다. 만약 로그가 쉽게 수정되거나 삭제된다면 로그는 자신의 무결성을 잃고, 사고 조사나 법적 절차에서 신뢰할 수 없게 된다. 로그는 내부 애플리케이션의 동작이나 민감한 메타데이터를 노출할 수 있으므로 공격자들에게 매력적인 표적이 될 수 있다.

이번 섹션은 로그를 비인가 접근, 위조, 공개로부터 보호하고 로그들이 안전하게 전송되고 또 안전하고 격리된 시스템에 저장되도록 보장하기 위한 요구사항들을 정의한다.

#	설명	레벨
16.4.1	로그 삽입을 방지하기 위해 모든 로깅 컴포넌트가 적절히 데이터를 인코딩하는지 검증한다.	2
16.4.2	로그가 비인가된 접근으로부터 보호되고 수정할 수 없는지 검증한다.	2
16.4.3	분석, 탐지, 경보, 그리고 에스컬레이션 (escalation) 을 위해 로그가 논리적으로 분리된 시스템으로 안전하게 전송되는지 검증한다. 애플리케이션이 침해되더라도 로그는 훼손하지 못하도록 보장하는 것이 목적이다.	2

V16.5 오류 처리

이번 섹션은 애플리케이션이 민감한 내부 상세 정보를 공개하지 않고 안전하고 우아한 오류 발생을 보장하기 위한 요구사항들을 정의한다.

#	설명	레벨
16.5.1	예상치 못한 혹은 보안 측면에서 민감한 오류가 발생했을 때 스택 흔적, 쿼리, 비밀키, 그리고 토큰 등의 민감한 내부 시스템 데이터의 노출 방지를 보장하기 위해 유저에게 구체화하지 않은 메시지를 반환하는지 검증한다.	2
16.5.2	외부 리소스 접근 오류가 발생하더라도 애플리케이션이 계속해서 안전하게 작동하는지 검증한다. 예를 들어 서킷 브레이커 (circuit breaker) 나 우아한 성능 저하와 같은 패턴을 사용할 수 있다.	2
16.5.3	예외가 발생할 때, 검증 로직의 오류에도 불구하고 거래를 처리하는 등의 페일-오픈 (fail-open) 상태 방지를 포함하여 애플리케이션이 우아하고 안전하게 실패하는지 검증한다.	2
16.5.4	처리되지 못한 모든 예외를 처리할“최후의 수단”오류 핸들러가 정의되어 있는지 검증한다. 이는 로그 파일에 기록되어야 하는 오류 상세 정보들의 손실을 피하기 위함과 동시에, 오류로 인해 전체 애플리케이션 프로세스가 종료되어 가용성 손실로 이어지지 않도록 보장하기 위함이다.	3

참고: Swift 나 Go 등의 특정 언어들, 그리고 일반적인 설계 관례상의 많은 함수형 언어들은 예외나 최후의 수단 이벤트 핸들러를 지원하지 않는다. 이 경우에는 설계자나 개발자들이 애플리케이션이 안전하게 예외와 예상치 못한 혹은 보안 관련 이벤트들을 처리하도록 보장하기 위해 패턴이나, 언어, 프레임워크 친화적인 방법을 사용해야 한다.

참조

더 많은 정보는 다음을 참고한다:

- OWASP Web Security Testing Guide: Testing for Error Handling
- OWASP Authentication Cheat Sheet section about error messages
- OWASP Logging Cheat Sheet
- OWASP Application Logging Vocabulary Cheat Sheet

V17 WebRTC

제어 목표

웹 실시간 통신 (WebRTC) 은 최신 애플리케이션에서 음성과 동영상 및 데이터를 실시간으로 교환할 수 있도록 한다. 도입이 늘어남에 따라 WebRTC 인프라스트럭처 보안이 점점 중요해지고 있다. 이 섹션은 WebRTC 시스템을 개발, 제공, 혹은 통합하는 이해관계자들에게 보안 요구사항을 제공한다.

WebRTC 시장은 크게 세 개의 분야로 나눌 수 있다:

1. 제품 개발자: WebRTC 제품과 솔루션을 제작하고 공급하는 소유자나 오픈 소스 벤더이다. 그들은 다른 사람들이 사용할 수 있는 강건하고 안전한 WebRTC 기술을 개발하는 것이 목표이다.
2. 서비스형 통신 플랫폼 (CPaaS): WebRTC 기능을 가능하게 하기 위한 API, SDK 와 필수적인 인프라스트럭처 혹은 플랫품을 제공하는 제공자이다. CPaaS 제공자는 서비스를 제공하기 위해 첫번째 분야의 제품을 사용하거나 자체 개발한 WebRTC 소프트웨어를 사용할 수 있다.
3. 서비스 제공자: 제품 개발자 혹은 CPaaS 제공자의 제품, 또는 자체 개발한 WebRTC 솔루션을 개발하고 활용하는 조직이다. 이런 조직은 온라인 컨퍼런스나 헬스케어, e-러닝 등 다양한 도메인에서 실시간 통신이 필요한 애플리케이션을 제작하고 구현한다.

보안 요구사항은 이곳에 제시되어 있으며 우선적으로 다음과 같은 제품 개발자, CPaaS 와 서비스 제공자 초점이 맞추고 있다:

- WebRTC 애플리케이션을 만들기 위해 오픈 소스 솔루션 활용
- 인프라스트럭처에서 부분적으로 상용 WebRTC 제품 사용
- 내부적으로 개발된 WebRTC 솔루션을 사용하거나 유기적인 서비스 제품에 다양한 컴포넌트를 통합

보안 요구사항은 예외적으로 CPaaS 에서 제공하는 SDK 와 API 를 사용하는 개발자들에게는 적용되지 않는다는 것을 알아두어야 한다. CPaaS 제공자는 당사 플랫폼의 보안 우려점에 대한 일반적인 책임이 있으며 ASVS 같은 일반적인 보안 표준은 그들의 수요를 모두 충족하지 못할 수 있다.

V17.1 TURN 서버

이 섹션은 고유의 TURN(Traversal Using Relays around NAT) 을 실행하는 시스템을 위한 보안 요구사항을 정의 한다. TURN 서버는 제한된 네트워크 환경에서 미디어 중계를 보조하는데, 설정이 잘못된 경우 위험을 초래할 수 있다. 이 제어는 안전한 주소 필터링과 자원 고갈을 방지하는데 초첨을 맞추고 있다.

#	설명	레벨
17.1.1	Traversal Using Relays around NAT(TURN) 서비스가 특정 목적을 위해 보존된 주소 (예: 내부 네트워크, 브로드캐스트, 루프백) 를 제외한 주소에 대해서만 접근을 허용하도록 한다.	2
17.1.2	Traversal Using Relays around NAT(TURN) 서비스가 정당한 사용자의 TURN 서버 포트의 대량 개방 시 자원고갈에 취약하지 않도록 한다.	3

V17.2 미디어

이러한 요구사항은 선택적 포워딩 유닛 (SFUs, Selective Forwarding Units), 다중점 제어 유닛 (MCUs, Multipoint Control Units), 기록 서버 혹은 게이트웨이 서버 등 고유의 WebRTC 미디어 서버를 호스팅하는 시스템에게만 적용된다. 미디어 서버는 미디어 스트림을 처리하고 배포하므로, 피어 간 통신을 보호하기 위한 서버 보안이 매우 중요하다. 사용자 프라이버시와 통신 품질을 위협할 수 있는 도청, 변조, 서비스 거부 공격을 방지하기 위해 WebRTC 애플리케이션에서 미디어 스트림 보호는 최우선 과제이다.

특히, 속도 제한, 타임스탬프 검증, 실시간 간격 일치를 위한 동기화된 시계 사용, 오버플로우 방지 및 적절한 타이밍 유지를 위한 버퍼 관리와 같은 플러드 공격에 대한 보호 장치를 구현해야 한다. 특정 미디어 세션의 패킷이 너무 빠르게 도착할 경우 초과 패킷은 삭제되어야 한다. 또한 입력 검증 구현, 정수 오버플로우 안전 처리, 버퍼 오버플로우 방지 및 기타 견고한 오류 처리 기법을 통해 시스템이 잘못된 형식의 패킷으로부터 보호받는 것이 중요하다.

중간 미디어 서버의 개입 없이 웹 브라우저 간 피어 투 피어 미디어 통신에만 의존하는 시스템은 이러한 특정 미디어 관련 보안 요구사항에서 제외된다.

이 섹션은 WebRTC 환경에서 데이터그램 전송 계층 보안 (DTLS) 의 사용을 다룬다. 암호화 키 관리에 대한 문서화된 정책 수립과 관련된 요구사항은“암호화”장에서 확인할 수 있다. 승인된 암호화 방법에 대한 정보는 ASVS 의 암호화 부록 또는 NIST SP 800-52 Rev. 2 or BSI TR-02102-2 (Version 2025-01) 등의 문서에서 확인할 수 있다.

#	설명	레벨
17.2.1	데이터그램 전송 계층 보안 (DTLS) 인증서의 키가 암호화 키 관리에 대한 문서화된 정책에 따라 관리되고 보호되는지 확인해야 한다.	2
17.2.2	미디어 서버가 승인된 데이터그램 전송 계층 보안 (DTLS) 암호 모음을 사용하고 지원하도록 구성되었는지, 그리고 보안 실시간 전송 프로토콜 (DTLS-SRTP) 을 위한 키 생성에 사용되는 DTLS 확장을 위한 안전한 보호 프로파일을 구성했는지 확인해야 한다.	2
17.2.3	미디어 서버에서 보안 실시간 전송 프로토콜 (SRTP) 인증이 선택되어 있는지 확인한다. 이를 통해 실시간 전송 프로토콜 (RTP) 삽입 공격으로 인한 서비스 거부 상태 발생 또는 오디오/비디오 미디어가 미디어 스트림에 삽입되는 것을 방지할 수 있다.	2
17.2.4	미디어 서버가 잘못된 형식의 보안 실시간 전송 프로토콜 (SRTP) 패킷을 접했을 때도 수신 미디어 트래픽 처리를 계속할 수 있는지 확인해야 한다.	2
17.2.5	미디어 서버가 정상 사용자로부터의 보안 실시간 전송 프로토콜 (SRTP) 패킷이 대량으로 유입되는 상황에서도 수신 미디어 트래픽 처리를 계속할 수 있는지 확인해야 한다.	3
17.2.6	미디어 서버가 Datagram Transport Layer Security(DTLS) 의“ClientHello”경합 조건 취약점에 노출되지 않았는지 확인해야 한다. 이를 위해 해당 미디어 서버가 공개적으로 취약한 것으로 알려져 있는지 확인하거나 경합 조건 테스트를 수행해야 한다.	3
17.2.7	미디어 서버와 연결된 모든 오디오 또는 비디오 녹화 장치가 합법적인 사용자로부터의 보안 실시간 전송 프로토콜 (SRTP) 패킷이 대량으로 유입되는 상황에서도 수신 미디어 트래픽 처리를 계속할 수 있는지 확인해야 한다.	3
17.2.8	데이터그램 전송 계층 보안 (DTLS) 인증서가 세션 설명 프로토콜 (SDP) 지문 속성과 대조되어 검증되었는지 확인하고, 검증이 실패할 경우 미디어 스트림을 종료하여 미디어 스트림의 진위성을 보장해야 한다.	3

V17.3 신호 처리

이 섹션은 자체 WebRTC 신호 서버를 운영하는 시스템에 대한 요구 사항을 정의한다. 신호는 피어 투 피어 통신을 조정하며, 세션 설정 또는 제어를 방해할 수 있는 공격에 대해 복원력을 가져야 한다.

안전한 신호 처리를 보장하기 위해 시스템은 잘못된 형식의 입력을 우아하게 처리하고 부하 상태에서도 가용성을 유지해야 한다.

#	설명	레벨
17.3.1	신호 서버가 플러드 공격 중에도 합법적인 수신 신호 메시지 처리를 계속할 수 있는지 확인해야 한다. 이는 신호 수준에서 속도 제한을 구현함으로써 달성되어야 한다.	2
17.3.2	서비스 거부 상태를 유발할 수 있는 잘못된 형식의 신호 메시지를 접했을 때 신호 서버가 정상적인 신호 메시지 처리를 계속할 수 있는지 확인해야 한다. 여기에는 입력 검증 구현, 정수 오버플로우 안전 처리, 버퍼 오버플로우 방지 및 기타 강력한 오류 처리 기법 적용이 포함될 수 있다.	2

참조

자세한 내용은 다음을 참조한다:

- The WebRTC DTLS ClientHello DoS is best documented at Enable Security's blog post aimed at security professionals and the associated white paper aimed at WebRTC developers
- RFC 3550 - RTP: A Transport Protocol for Real-Time Applications
- RFC 3711 - The Secure Real-time Transport Protocol (SRTP)
- RFC 5764 - Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP))
- RFC 8825 - Overview: Real-Time Protocols for Browser-Based Applications
- RFC 8826 - Security Considerations for WebRTC
- RFC 8827 - WebRTC Security Architecture
- DTLS-SRTP Protection Profiles

부록 A: 어휘

- **절대 최대 세션 수명** - NIST 에서는"Overall Timeout"이라고도 언급되었다. 유저 상호작용과 관계없이 다음 인증까지 세션이 액티브하게 유지될 수 있는 최대 시간이다. 세션 만료의 한 요소이다.
- **허용 목록 (allowlist)** - 허용된 데이터 혹은 작업들의 리스트. 예를 들어 입력 검증이 허용된 문자들의 리스트.
- **위조 방지 토큰** - 요청에 하나 혹은 그 이상의 토큰이 전달되고 그 요청이 예상되는 엔드포인트로부터 왔음을 보장하도록 애플리케이션 서버에서 검증되는 메커니즘.
- **애플리케이션 보안** - 예를 들어 기저에 있는 운영 체제 혹은 연결된 네트워크에 집중하기보다는 OSI 모델에서 애플리케이션 계층을 이루는 구성 요소들의 분석에 초점을 두는 애플리케이션 단계의 보안.
- **애플리케이션 보안 검증** - OWASP ASVS 를 따르는 애플리케이션의 기술적 평가.
- **애플리케이션 보안 검증 보고서** - 검증자가 특정 애플리케이션에 대한 전반적인 결과와 보조적인 분석을 기록한 보고서.
- **인증** - 애플리케이션 유저가 주장하는 신원의 검증.

- **자동화된 검증** - 문제를 찾기 위한 취약점 시그니처를 사용하는 자동화 도구 (동적 분석 도구, 정적 분석 도구 혹은 둘 다) 의 사용.
- **블랙박스 시험** - 내부 구조나 동작을 자세히 들여다보지 않고 애플리케이션의 기능을 검사하는 소프트웨어 시험 방법.
- **공통 취약점 목록** (CWE) - 커뮤니티에서 개발한 공통 소프트웨어 보안 취약점 목록. 공용어로 제공되며, 소프트웨어 보안 도구들을 위한 척도이자, 취약점의 식별, 완화책, 예방을 위한 노력의 기준선이다.
- **컴포넌트 (component)** - 다른 컴포넌트들과 통신하는 디스크나 네트워크 인터페이스들과 연관된 자체-포함 단위의 코드.
- **자격 증명 서비스 제공자** (CSP) - IdP 라고도 불린다. 다른 애플리케이션들로부터 인증의 근원으로써 사용될 수 있는 유저 데이터의 근원.
- **교차 사이트 스크립트 포함** (XSSI) - 웹 애플리케이션이 외부 리소스로부터 악의적 코드를 받고 그 코드를 자기 콘텐츠의 일부로 포함하는 XSS 의 변이 공격이다.
- **교차 사이트 스크립트** (XSS) - 콘텐츠에 클라이언트 단 스크립트 주입을 허용하는 웹 애플리케이션에서 전형적으로 찾아볼 수 있는 보안 취약점이다.
- **암호 모듈** - 암호화 알고리즘을 구현하고/구현하거나 암호 키를 생성하는 하드웨어, 소프트웨어, 그리고/또는 펌웨어.
- **암호학적으로 안전한 의사 난수 생성기** (CSPRNG) - 암호 기법에서 사용하기 적합한 의사 난수 생성기, 암호 난수 생성기 (CRNG) 라고도 한다.
- **데이터그램 전송 계층 보안** (DTLS) - 네트워크 연결 상에서 통신 보안을 제공하는 암호 프로토콜이다. TLS 프로토콜을 기반으로 하지만, (보통 UDP 상에서의) 데이터그램-지향 프로토콜 보호를 위해 개량되었다. RFC 9147 에서 DTLS 1.3 으로 정의되었다.
- **실시간 전송 프로토콜 보호의 키 수립을 위한 데이터그램 전송 계층 보안 확장** (DTLS-SRTP) - SRTP 세션의 키 수립을 위해 DTLS 핸드셰이크를 사용하는 메커니즘. RFC 5764 에서 정의되었다.
- **설계 검증** - 애플리케이션 보안 구조에 대한 기술적 평가.
- **동적 애플리케이션 보안 테스팅** (DAST) - 애플리케이션이 동작하는 상태에서 보안 취약점을 드러낼 수 있는 조건들을 발견하기 위한 기술들.
- **동적 검증** - 애플리케이션이 동작하는 동안 문제를 찾기 위해 취약점 시그니처를 활용하는 자동화 도구의 사용.
- **온라인 간편 인증** (FIDO) - 생체인증, TPM(Trusted Platform Modules), USB 보안 토큰 등을 포함해 다양한 인증 방법들을 이용할 수 있게 하는 인증 표준들의 집합.
- **HSM** (Hardware Security Module) - 보호된 방식으로 암호키나 기밀들을 저장하는 하드웨어 컴포넌트.
- **HQL** (Hibernate Query Language) - SQL 과 문법적으로 비슷해 보이는 히버네이트 ORM 라이브러리에서 사용하는 쿼리 언어.
- **HSTS** (HTTP Strict Transport Security) - 브라우저에 해당 도메인에 대해 유효한 인증서가 제시된 TLS 연결만 허용하도록 지시하는 정책이다. Strict-Transport-Security 응답 헤더 필드를 통해서 활성화된다.
- **HTTP** (HyperText Transfer Protocol) - 분산된 협력적인 하이퍼미디어 정보 시스템을 위한 애플리케이션 프로토콜이다. 월드 와이드 웹 (World Wide Web) 데이터 통신의 토대이다.
- **SSL/TLS 상에서의 HyperText Transfer Protocol** (HTTPS) - 전송 계층 보안 (TLS) 으로 암호화하여 HTTP 통신을 보호하는 방법.
- **신원 제공자** (IdP) - NIST references 로부터 자격 증명 서비스 제공자 (CSP) 로도 언급된다. 다른 애플리케이션에 인증 근원을 제공해 준다.
- **비활동 타임아웃** - 유저가 활동하지 않고 세션이 유지될 수 있는 시간의 길이이다. 세션 만료의 한 요소이다.
- **입력 검증** - 신뢰할 수 없는 유저 입력의 정규화와 검증
- **JSON 웹 토큰** (JWT) - 해당 객체의 유효성을 어떻게 검증할지 설명하는 헤더 섹션, 클레임들 (claims) 을 포함한

바디 섹션, 바디 섹션의 내용들을 검증할 디지털 시그니처를 담고 있는 시그니처 섹션으로 구성되어 있는 JSON 데이터 객체. 자체-포함 토큰의 한 종류이다. RFC 7519 에서 정의되었다.

- **로컬 파일 삽입** (LFI) - 애플리케이션에서 취약한 파일 삽입 과정을 악용하는 공격으로, 서버에 이미 존재하는 로컬 파일에 대한 삽입으로 이어질 수 있다.
- **악의적 코드** - 개발 과정 중 애플리케이션 소유자가 알지 못하게 삽입되어 애플리케이션의 보안 정책을 우회하는 코드. 바이러스나 웜과 같은 멀웨어와는 다르다!
- **멀웨어** - 애플리케이션 실행 중에 유저나 관리자가 알지 못하게 애플리케이션에 삽입된 실행 가능한 코드.
- **메시지 인증 코드** (MAC) - MAC 생성 알고리즘을 통해 산출되어 데이터의 무결성과 인증성을 보장하는 데이터에 대한 암호 체크섬.
- **다중 인증** (MFA) - 둘 혹은 그 이상의 단일 요소를 포함한 인증.
- **상호 TLS** (mTLS) - 'TLS 클라이언트 인증'을 확인.
- **객체-관계 매핑** (ORM) - 애플리케이션-호환 객체 모델을 사용하여 관계형 데이터베이스를 애플리케이션 내에서 참조하고 질의할 수 있게 하는 시스템.
- **일회용 비밀번호** (OTP) - 한 번만 사용되도록 유일하게 생성된 비밀번호.
- **오픈 웹 애플리케이션 보안 프로젝트** (OWASP) - OWASP 는 애플리케이션 소프트웨어의 보안 증진에 초점을 맞춘 자유롭고 개방적인 커뮤니티이다. 우리의 사명은 애플리케이션 보안을"가시적"으로 만들어 사람들과 단체들이 애플리케이션 보안 위험에 대해 현명한 선택을 할 수 있도록 하는 것이다. https://www.owasp.org/를 확인.
- **암호-기반 키 유도 함수 2** (PBKDF2) - 특별한 단방향 알고리즘으로 (비밀번호와 같은) 입력값 텍스트와 추가적인 무작위 솔트 값으로부터 강한 암호 키를 생성하고, 기존 비밀번호 대신 저장하여 오프라인에서 비밀번호를 알아내기 어렵게 만든다.
- **공개키 인프라스트럭처** (PKI) - 공개키를 각각의 개체와 신원과 연결하는 배열이다. 이러한 배열은 인증기관들로부터 등록 과정과 인증서 발급을 통해 수립된다.
- **공중 교환 전화망** (PSTN) - 유선 전화와 휴대 전화를 포함한 전통적인 전화 네트워크.
- **실시간 전송 프로토콜** (RTP) 과 **실시간 전송 제어 프로토콜** (RTCP) - 멀티미디어 스트림을 전송하기 위해 함께 사용되는 두 프로토콜이다. 웹 실시간 통신 스택에 의해 사용된다. RFC 3550 에서 정의되었다.
- **레퍼런스 토큰 (Reference Token)** - 서버에 저장된 상태나 메타데이터의 포인터나 식별자로서 동작하는 토큰의 한 종류. 가끔'임의 토큰'이나'불투명 토큰'이라고도 한다. 토큰 자체에 토큰 자신과 관련 있는 데이터들이 포함된 자체-포함 토큰과 다르게, 레퍼런스 토큰은 고유한 정보를 포함하지 않는 대신, 맥락을 위해 서버에 의존한다. 또한 레퍼런스 토큰은 세션 식별자가 되거나, 그것에 포함된다.
- **신뢰 당사자** (RP) - 일반적으로 유저가 별도의 인증 제공자로부터 인증을 받았다고 신뢰하는 애플리케이션. 인증 제공자로부터 제공된 몇 종류의 토큰이나 서명된 주장 (assertion) 들의 집합으로 유저를 신뢰한다.
- **원격 파일 삽입** (RFI) - 애플리케이션에서 취약한 파일 삽입 과정을 악용하는 공격. 외부 파일의 삽입으로 이어질 수 있다.
- **확장형 벡터 그래픽스** (SVG) - 2 차원 기반의 벡터 그래픽을 표현하는 XML 기반의 마크업 언어.
- **실시간 전송 보안 프로토콜** (SRTP) 와 **실시간 전송 보안 제어 프로토콜** (SRTCP) - 메시지의 암호화, 인증, 무결성의 보호를 지원하는 RTP 와 RTCP 의 프로파일 (profile). RFC 3711 에 정의되어 있다.
- **보안 아키텍처** - 애플리케이션 설계의 추상화로 보안 제어가 어디서 어떻게 사용되는지와 유저와 애플리케이션 데이터의 위치와 민감도를 식별하고 설명한다.
- **보안 보장 마크업 언어** (SAML) - 신원 제공자와 신뢰 당사자 간 서명된 보장 (주로 XML 객체) 의 전달을 기반으로 하는 싱글 사인 온 인증을 위한 개방 표준이다.
- **보안 구성**- 보안 제어가 어떻게 사용되는지에 영향을 주는 애플리케이션의 런타임 구성이다.

- **보안 제어** - 보안 검사를 수행하는 기능이나 요소 (예: 인증 검사) 혹은 보안적 측면에서 영향을 초래하는 경우. (예: 감사 레코드를 생성)
- **보안 정보와 이벤트 관리** (SIEM) - 컬렉션과 여러 소스로부터의 보안-관련 데이터들의 분석을 통한 조직의 IT 인프라스트럭처에서의 위협 감지, 컴플라이언스, 보안 사고 관리를 위한 시스템.
- **자체-포함 토큰** - 서버 단의 상태나 다른 외부 저장소에 의존하지 않는 하나 이상의 속성들을 캡슐화한 토큰. 이 토큰들은 포함되어 있는 속성들의 인증성과 무결성을 보장하고, “무상태”정보들을 여러 시스템에 걸쳐 안전한 교환이 가능하게 한다. 자체-포함 토큰은 보통 인증성, 무결성 그리고 어떤 경우에는 데이터의 기밀성을 보장하기 위해 디지털 서명, 메시지 인증 코드 (MAC) 등의 암호 기술들을 이용하여 보호된다.
- **서버 측 요청 위조** (SSRF) - 서버의 내부 리소스들을 읽고 업데이트하는 기능을 남용하는 공격. 공격자는 데이터를 읽거나 제출하는 코드인 서버상에서 작동하는 URL 을 제공하거나 변형한다.
- **세션 기술 프로토콜** (SDP) - 멀티미디어 세션을 구성하기 위한 메시지 형식이다. (예로 웹 실시간 통신에서 사용된다.) RFC 4566 에서 정의되었다.
- **세션 식별자** 또는 **세션 ID** - 백엔드에 저장된 상태 유지 세션을 식별하는 키이다. 레퍼런스 토큰으로서, 혹은 레퍼런스 토큰 내에서 클라이언트에게 혹은 클라이언트로부터 전달된다.
- **세션 토큰** - (자체-포함 토큰을 사용하는) 무상태 세션 메커니즘이나 (레퍼런스 토큰을 사용하는) 상태 유지 세션 메커니즘에서 사용되는 토큰이나 값을 이 표준에서 포괄적으로 이르는 말.
- **NAT 용 세션 탐색 유틸리티** (STUN) - peer-to-peer 통신을 수립하기 위해 NAT 탐색을 돕는 프로토콜. RFC 3489 에서 정의되었다.
- **단일 요소 인증기** - 유저가 인증되었는지 확인하는 메커니즘. 당신이 알거나 (기억하는 비밀, 비밀번호, 비밀 구절, PIN), 당신 자신이거나 (생체 인증, 지문, 얼굴 인식), 당신이 가지고 있는 것 (OTP 토큰, 스마트카드와 같은 암호 디바이스) 들이 될 수 있다.
- **싱글 사인 온 인증** (SSO) - 유저가 한 애플리케이션에 로그인한 다음 다시 인증을 할 필요 없이 다른 애플리케이션에 자동으로 로그인될 때 발생한다. 예를 들어, 구글에 로그인할 때, 유저는 자동으로 유튜브, 구글 문서, G 메일 같은 다른 구글 서비스들에 로그인된다.
- **소프트웨어 구성 명세서** (SBOM) - 소프트웨어 애플리케이션을 만들거나 어셈블 (assemble) 할때 요구되는 모든 컴포넌트, 모듈, 라이브러리, 프레임워크 등 리소스들의 구조적이고 포괄적인 리스트이다.
- **소프트웨어 구성 분석** (SCA) - 사용 중인 특정 컴포넌트 버전의 보안 취약점을 위한 애플리케이션 구성, 의존성, 라이브러리 그리고 패키지들을 분석하기 위해 설계된 기술들의 집합. 현재 흔히 SAST 라고 불리는 소스코드 분석과 혼동해서는 안 된다.
- **소프트웨어 개발 수명 주기** (SDLC) - 초기 요구 사항부터 배포와 유지보수까지 소프트웨어가 개발되는 단계별 과정이다
- **SQL 인젝션** (SQLi) - 엔트리 포인트에 악성 SQL 구문이 삽입되는, 데이터 기반 애플리케이션을 공격하기 위해 사용되는 코드 주입 기술
- **상태 유지 세션 메커니즘** - 상태 유지 세션 메커니즘에서는, 애플리케이션이 CSRPNG 를 사용하여 생성된 세션 토큰에 해당하는 세션 상태를 백엔드 단에 보유하며, 이 토큰은 최종 사용자에게 발급된다.
- **무상태 세션 메커니즘** - 무상태 세션 메커니즘은 서비스 내에 저장될 필요가 없고 클라이언트에게 전달되는 자체-포함 토큰을 사용하고, 토큰을 받고 검증한다. 그러나 실제로는, 서비스가 필요한 보안 제어를 시행하기 위해 약간의 세션 정보 (JWT 파기 목록과 같은) 에 대한 접근이 필요하다.
- **정적 애플리케이션 보안 테스팅** (SAST) - 보안 취약점을 시사하는 코딩, 설계 조건들을 찾기 위해 애플리케이션의 소스 코드, 바이트 코드, 바이너리들을 분석하도록 설계된 기술들의 집합. SAST 솔루션들은 비실행 상태에서 애플리케이션을“내부에서 외부로”분석한다.

- **위협 모델링** - 위협 주체, 보안 영역, 보안 제어, 그리고 중요한 기술과 사업 자산들을 식별하기 위해 보안 아키텍처를 점점 더 정교하게 개발해 나가는 기술이다.
- **검사 시점과 사용 시점** (TOCTOU) - 애플리케이션이 리소스를 사용하기 전에 리소스의 상태를 검사하더라도, 리소스의 상태는 리소스의 검사 시점과 사용 시점 사이에 바뀔 수 있다. 이는 검사 결과를 무효화하고 상태 비일치로 인한 잘못된 동작의 수행을 초래할 수 있다.
- **시간 동기 일회용 패스워드** (TOTPs) - 현재 시각이 비밀번호를 생성하는 알고리즘의 일부로 사용되는 OTP 생성의 한 방식.
- **TLS 클라이언트 인증**, 또는 **상호 TLS** (mTLS) - 표준 TLS 연결에서, 클라이언트는 서버의 신원을 검증하기 위해 서버가 제공한 인증서를 사용할 수 있다. TLS 클라이언트 인증이 사용될 때, 클라이언트 또한 자신의 비밀 키와 인증서를 사용하여 서버로 하여금 클라이언트의 신원을 검증하도록 허용할 수 있다.
- **전송 계층 보안** (TLS) - 네트워크 연결 상에서 통신 보안을 제공하는 암호 프로토콜.
- **NAT 를 중심으로 릴레이를 이용한 트래버설** (TURN) - peer-to-peer 직접 연결이 수립되지 못할 때 TURN 서버를 중계 서버로 사용하는 STUN 프로토콜의 확장. RFC 8656 에서 정의되었다.
- **신뢰 실행 환경** (TEE) - 애플리케이션이 시스템의 다른 영역과 관계없이 안전하게 실행될 수 있는 고립된 처리 환경.
- **신뢰 플랫폼 모듈** (TPM) - 마더보드와 같은 더 큰 하드웨어 컴포넌트에 부착되어 그 시스템의“신뢰의 근원”역할을 하는 HSM 의 한 종류.
- **신뢰 서비스 계층** - 마이크로서비스, 서버리스 API, 서버 단, secure boot 을 가진 클라이언트 장치의 신뢰되는 API, 파트너 또는 외부 API 등등의 제어 집행 지점 (control enforcement point). ‘신뢰되는’이란, 신뢰할 수 없는 유저가 그 계층에 구현된 계층이나 제어를 우회하거나 통과할 우려가 없음을 의미한다.
- **인터넷 식별자** (URI) - 웹페이지, 메일 주소, 장소 등을 식별하는 고유의 문자열.
- **인터넷 주소** (URL) - 인터넷상의 리소스들의 위치를 명시하는 문자열.
- **범용 단일 식별자** (UUID) - 소프트웨어에서 식별자로 사용되는 고유의 레퍼런스 번호.
- **검증자** - 애플리케이션을 OWASP ASVS 요구사항에 따라 리뷰하는 사람이나 팀.
- **웹 실시간 통신** (WebRTC) - 보통 화상회의에서, 웹 애플리케이션에서 멀티미디어 스트림을 전송하기 위해 사용되는 프로토콜 스택과 관련된 웹 API. SRTP, SRTCP, DTLS, SDP 그리고 STUN/TURN 을 기반으로 한다.
- **TLS 상의 웹 소켓** (WSS) - 웹 소켓 통신을 TLS 프로토콜 상에서 계층화하여 보호하는 한 방법.
- **위지위그** (WYSIWYG) - 렌더링을 제어하는 코드 자체를 보이기보다 콘텐츠가 렌더링 되었을 때 실제로 어떻게 보일지를 보여주는 리치 콘텐츠 에디터의 한 종류.
- **X.509 인증서** - 공개키가 인증서에 포함된 해당 유저, 컴퓨터 혹은 서비스의 신원에 속해있음을 검증하고, 널리 받아들여지는 국제 X.509 PKI 표준을 사용하는 디지털 인증서.
- **XML 외부 개체** (XXE) - 선언된 시스템 식별자를 통해 로컬 혹은 리모트 콘텐츠에 접근할 수 있는 XML 개체의 한 종류. 다양한 삽입 공격을 초래할 수 있다.

부록 B: 참고 문헌

다음은 본 표준의 사용자와 채택자들에게 유용할 OWASP 프로젝트들이다:

OWASP 핵심 프로젝트

1. OWASP Top 10 Project: https://owasp.org/www-project-top-ten/

2. OWASP Web Security Testing Guide: https://owasp.org/www-project-web-security-testing-guide/
3. OWASP Proactive Controls: https://owasp.org/www-project-proactive-controls/
4. OWASP Software Assurance Maturity Model (SAMM): https://owasp.org/www-project-samm/
5. OWASP Secure Headers Project: https://owasp.org/www-project-secure-headers/

OWASP 치트 시트 시리즈 (Cheat Sheet Series) 프로젝트

이 프로젝트는 ASVS 의 다양한 주제에 관한 치트 시트를 제공한다.

ASVS 와의 매핑은 여기서 확인할 수 있다: https://cheatsheetseries.owasp.org/IndexASVS.html

모바일 보안 관련 프로젝트

1. OWASP Mobile Security Project: https://owasp.org/www-project-mobile-security/
2. OWASP Mobile Top 10 Risks: https://owasp.org/www-project-mobile-top-10/
3. OWASP Mobile Security Testing Guide and Mobile Application Security Verification Standard: https://owasp.org/www-project-mobile-security-testing-guide/

OWASP 사물인터넷 (IoT) 관련 프로젝트

1. OWASP Internet of Things Project: https://owasp.org/www-project-internet-of-things/

OWASP 서버리스 (Serverless) 프로젝트

1. OWASP Serverless Project: https://owasp.org/www-project-serverless-top-10/

기타

다음 웹사이트들도 본 표준의 사용자와 채택자들에게 유용하게 활용될 것이다.

1. SecLists Github: https://github.com/danielmiessler/SecLists
2. MITRE Common Weakness Enumeration: https://cwe.mitre.org/
3. PCI Security Standards Council: https://www.pcisecuritystandards.org/
4. PCI Data Security Standard (DSS) v3.2.1 Requirements and Security Assessment Procedures: https://www.pcisecuritystandards.org/documents/PCI_DSS_v3-2-1.pdf
5. PCI Software Security Framework - Secure Software Requirements and Assessment Procedures: https://www.pcisecuritystandards.org/documents/PCI-Secure-Software-Standard-v1_0.pdf
6. PCI Secure Software Lifecycle (Secure SLC) Requirements and Assessment Procedures: https://www.pcisecuritystandards.org/documents/PCI-Secure-SLC-Standard-v1_0.pdf
7. OWASP ASVS 4.0 Testing Guide https://github.com/BlazingWind/OWASP-ASVS-4.0-testing-guide

부록 C: 암호 기법 표준

“암호 기법”장은 단순히 모범 사례를 정의하는 것을 넘어선다. 이 장은 암호학 원리에 대한 이해를 향상시키고, 보다 견고하고 현대적인 보안 방법의 채택을 장려하는 것을 목표로 한다. 이 부록에서는 각 요구 사항에 대한 상세한 기술 정보를 제공하며, “암호 기법”장에 설명된 전반적인 표준을 보완한다.

이 부록은 다양한 암호화 메커니즘 (mechanism) 에 대한 승인 수준을 정의한다:

- 승인된 메커니즘 (A) 은 애플리케이션에서 사용할 수 있다.
- 기존 메커니즘 (L) 은 애플리케이션에서 사용해서는 안 되지만, 기존 애플리케이션이나 코드와의 호환성을 위해 제한적으로 사용될 수 있다. 이러한 메커니즘의 사용은 현재 본질적으로 취약점으로 간주하지는 않지만, 가능한 한 빠른 시일 내에 더 안전하고 미래 지향적인 메커니즘으로 대체되어야 한다.
- 허용되지 않은 메커니즘 (D) 은 현재 취약한 것으로 간주되거나 충분한 보안을 제공하지 않으므로 사용해서는 안 된다.

이 목록은 특정 애플리케이션의 맥락에서 여러 가지 이유로 재정의 될 수 있으며, 예를 들어 다음과 같다:

- 암호학 분야의 새로운 발전;
- 규제 준수.

암호 자산 목록 및 문서화

이 문단은 V11.1 암호 목록 및 문서화에 대한 추가 정보를 제공한다.

알고리즘, 키, 인증서와 같은 암호 자산은 정기적으로 발견되고, 목록화되며, 평가되어야 한다. 3 단계에서는 애플리케이션에서 암호 기법의 사용을 식별하기 위해 정적 및 동적 스캐닝을 포함해야 한다. SAST 및 DAST 와 같은 도구는 이에 도움이 될 수 있으나, 더 포괄적인 범위를 위해서는 전용 도구가 필요할 수 있다. 프리웨어 도구의 예시는 다음과 같다:

- CryptoMon - Network Cryptography Monitor - using eBPF, written in python
- Cryptobom Forge Tool: Generating Comprehensive CBOMs from CodeQL Outputs

암호 매개변수의 동등 강도

다양한 암호 시스템의 상대적 보안 강도는 이 표에 제시되어 있다. (출처: NIST SP 800-57 Part 1, p.71):

보안 강도	대칭키 알고리즘	유한체	소인수 분해	타원 곡선
<= 80	2TDEA	L = 1024 N = 160	k = 1024	f = 160-223
112	3TDEA	L = 2048 N = 224	k = 2048	f = 224-255
128	AES-128	L = 3072 N = 256	k = 3072	f = 256-383
192	AES-192	L = 7680 N = 384	k = 7680	f = 384-511
256	AES-256	L = 15360 N = 512	k = 15360	f = 512+

애플리케이션 예시:

- 유한체 암호 기법: DSA, FFDH, MQV
- 소인수 분해 암호 기법: RSA
- 타원 곡선 암호 기법: ECDSA, EdDSA, ECDH, MQV

참고: 이 문단은 양자컴퓨터가 존재하지 않는다고 가정한다. 만약 그러한 컴퓨터가 존재한다면, 마지막 3 개 열의 추정치는 더 이상 유효하지 않게 된다.

무작위 값

이 문단은 V11.5 무작위 값에 대한 추가 정보를 제공한다.

이름	버전/참조	비고	상태
`/dev/random`	Linux 4.8+ (Oct 2016), iOS, Android, 그리고 다른 Linux 기반 POSIX 운영체제에서도 사용된다. RFC7539를 기반으로 한다.	ChaCha20 스트림을 활용한다. iOS 의 `SecRandomCopyBytes` 및 Android 의 `SecureRandom`에서 각각 올바른 설정이 적용된 상태로 제공된다.	A
`/dev/urandom`	무작위 데이터를 제공하는 Linux 커널의 특수 파일	하드웨어 난수를 통해 고품질의 엔트로피 소스를 제공한다.	A
`AES-CTR-DRBG`	NIST SP800-90A	일반적인 구현에서 사용되며, 예를 들어 `BCRYPT_RNG_ALGORITHM`로 설정된 Windows CNG API `BCryptGenRandom`이 사용된다.	A
`HMAC-DRBG`	NIST SP800-90A		A
`Hash-DRBG`	NIST SP800-90A		A
`getentropy()`	OpenBSD은 Linux glibc 2.25+ 및 macOS 10.12+에서 지원한다.	단순하고 최소화된 API 를 통해 커널의 엔트로피 소스에서 보안 무작위 바이트를 직접 제공한다. 이는 보다 최신 방식이며, 기존 API 와 관련된 문제점을 피할 수 있다.	A

HMAC-DRBG 또는 Hash-DRBG 와 함께 사용되는 기반 해시 함수는 반드시 해당 용도로 승인되어야 한다.

암호 알고리즘

이 문단은 V11.3 암호화 알고리즘에 대한 추가 정보를 제공한다.

승인된 암호 알고리즘은 선호도 순서로 제시되어 있다.

대칭키 알고리즘	참조	상태
AES-256	FIPS 197	A
Salsa20	Salsa 20 specification	A
XChaCha20	XChaCha20 Draft	A
XSalsa20	Extending the Salsa20 nonce	A
ChaCha20	RFC 8439	A
AES-192	FIPS 197	A
AES-128	FIPS 197	L
2TDEA		D
TDEA (3DES/3DEA)		D
IDEA		D
RC4		D
Blowfish		D
ARC4		D
DES		D

AES 암호 모드

AES 와 같은 블록 암호는 다양한 운용 모드와 함께 사용할 수 있다. 전자 코드북 (Electronic codebook; ECB) 와 같은 많은 운용 모드는 안전하지 않으며, 사용해서는 안 된다. 갈루아/카운터 모드 (Galois/Counter Mode; GCM) 와 CBC-MAC 을 사용하는 카운터 모드 (Counter with Cipher Block Chaining Message Authentication Code; CCM) 는 인증 암호화를 제공하며, 최신 애플리케이션에서 사용하는 것이 권장된다.

승인된 모드는 선호도 순서로 제시되어 있다.

모드	인증 여부	참조	상태	제한 사항
GCM	예	NIST SP 800-38D	A	
CCM	예	NIST SP 800-38C	A	

모드	인증 여부	참조	상태	제한 사항
CBC	아니오	NIST SP 800-38A	L	
CCM-8	예		D	
ECB	아니오		D	
CFB	아니오		D	
OFB	아니오		D	
CTR	아니오		D	

비고:

- 모든 암호화된 메시지는 반드시 인증되어야 한다. CBC 모드를 사용하는 모든 경우에 반드시 메시지 무결성을 검증하기 위한 해시 기반 MAC 알고리즘을 함께 사용해야 한다. 일반적으로 반드시 암호화 후 해싱 방식 (Encrypt-Then-Hash) 을 적용해야 한다. (단, TLS 1.2 는 해시 후 암호화 방식 (Hash-Then-Encrypt) 을 사용한다.) 이것이 보장되지 않는다면, CBC 를 절대 사용해서는 안 된다. MAC 알고리즘 없이 암호화가 허용되는 애플리케이션은 디스크 암호화로 한정된다.
- CBC 모드를 사용하는 경우, 패딩 검증이 상수 시간 (constant time) 으로 수행되도록 보장해야 한다.
- CCM-8 을 사용할 경우, MAC 태그의 보안 강도는 64 비트에 불과하다. 이는 최소 128 비트 이상의 보안 강도를 요구하는 요구사항 6.2.9 를 준수하지 않는다.
- 디스크 암호화는 ASVS 의 적용 범위에 포함되지 않는다. 따라서 이 부록에서는 디스크 암호화를 위한 승인된 방법을 제시하지 않는다. 이러한 용도의 경우, 인증 없는 암호화가 일반적으로 허용되며 XTS, XEX, LRW 모드가 주로 사용된다.

키 래핑

암호 키 래핑 (cryptographic key wrap, 및 해당 언래핑) 은 기존 키를 추가적인 암호화 메커니즘으로 캡슐화 (즉, 래핑) 하여 전송 과정 등에서 직접적으로 노출되지 않도록 보호하는 방법이다. 기존 키를 보호하기 위해 사용되는 이 추가 키를 래핑 키 (wrap key) 라고 한다.

이 작업은 신뢰할 수 없는 것으로 간주되는 환경에서 키를 보호하거나, 민감한 키를 신뢰할 수 없는 네트워크나 애플리케이션 내에서 전송해야 하는 경우에 수행될 수 있다. 그러나 래핑/언래핑 절차를 수행하기 전에 원래 키의 속성 (예: 키의 식별 정보 및 용도) 을 충분히 이해하는 것을 신중히 고려해야 하며, 이는 보안 측면뿐 아니라 특히 컴플라이언스 측면에서 원본 및 대상 시스템 또는 애플리케이션 모두에 영향을 미칠 수 있고, 컴플라이언스 요구사항에는 키 기능 (예: 서명) 에 대한 감사 추적 (audit trail) 과 적절한 키 저장 방식이 포함될 수 있다.

특히, 키 래핑에는 NIST SP 800-38F를 준수하고 양자 위협에 대비한 향후 권고 사항을 고려하여 반드시 AES-256 을 사용해야 한다. AES 를 사용하는 암호 모드는 선호도 순서로 제시되어 있다:

키 래핑	참조	상태
KW	NIST SP 800-38F	A
KWP	NIST SP 800-38F	A

AES-192 와 AES-128 은 사용 사례에서 필요할 경우 사용할 수 있지만, 그 사용 사유는 해당 엔터티의 암호 인벤토리 (cryptography inventory) 에 반드시 문서화해야 한다.

인증 암호화

디스크 암호화를 제외하고, 암호화된 데이터는 인증 암호화 (AE) 방식, 일반적으로는 연관 데이터가 있는 인증 암호화 (AEAD) 방식을 사용하여 무단 변경으로부터 반드시 보호해야 한다.

애플리케이션은 승인된 AEAD 방식을 사용하는 것이 바람직하다. 대안으로, 승인된 암호화 방식과 승인된 MAC 알고리즘을 결합하여 암호화 후 MAC(Encrypt-then-MAC) 구조를 사용할 수도 있다.

MAC 후 암호화는 방식은 레거시 애플리케이션과의 호환성을 위해 여전히 허용된다. 이 방식은 TLS 1.2 에서 구식 암호 제품군 (cipher suites) 과 함께 사용된다.

AEAD 메커니즘	참조	상태
AES-GCM	SP 800-38D	A
AES-CCM	SP 800-38C	A
ChaCha-Poly1305	RFC 7539	A
AEGIS-256	AEGIS: A Fast Authenticated Encryption Algorithm (v1.1)	A
AEGIS-128	AEGIS: A Fast Authenticated Encryption Algorithm (v1.1)	A
AEGIS-128L	AEGIS: A Fast Authenticated Encryption Algorithm (v1.1)	A
Encrypt-then-MAC		A
MAC-then-encrypt		L

해시 함수

이 문단은 V11.4 해싱과 해시 기반 함수에 대한 추가 정보를 제공한다.

범용 해시 함수

다음 표는 디지털 서명과 같은 일반적인 암호 사용 사례에서 승인된 해시 함수를 제시한다.

- 승인된 해시 함수는 강력한 충돌 저항성을 제공하며, 높은 보안 수준을 요구하는 애플리케이션에 적합하다.

- 이들 알고리즘 중 일부는 적절한 암호 키 관리와 함께 사용될 경우 강력한 공격 저항성을 제공하므로, HMAC, KDF, RBG 기능에도 추가적으로 승인된다.
- 출력 길이가 254 비트 미만인 해시 함수는 충돌 저항성이 불충분하므로 디지털 서명 또는 충돌 저항성이 필요한 다른 애플리케이션에 사용해서는 안 된다. 다른 용도의 경우, 호환성 및 검증을 위해서만 레거시 시스템에서 제한적으로 사용할 수 있지만, 새로운 설계에서는 사용해서는 안 된다.

해시 함수	참조	상태	제한 사항
SHA3-512	FIPS 202	A	
SHA-512	FIPS 180-4	A	
SHA3-384	FIPS 202	A	
SHA-384	FIPS 180-4	A	
SHA3-256	FIPS 202	A	
SHA-512/256	FIPS 180-4	A	
SHA-256	FIPS 180-4	A	
SHAKE256	FIPS 202	A	
BLAKE2s	BLAKE2: simpler, smaller, fast as MD5	A	
BLAKE2b	BLAKE2: simpler, smaller, fast as MD5	A	
BLAKE3	BLAKE3 one function, fast everywhere	A	
SHA-224	FIPS 180-4	L	HMAC, KDF, RBG, 디지털 서명에 적합하지 않다.
SHA-512/224	FIPS 180-4	L	HMAC, KDF, RBG, 디지털 서명에 적합하지 않다.
SHA3-224	FIPS 202	L	HMAC, KDF, RBG, 디지털 서명에 적합하지 않다.
SHA-1	RFC 3174 & RFC 6194	L	HMAC, KDF, RBG, 디지털 서명에 적합하지 않다.
CRC (길이 무관)		D	
MD4	RFC 1320	D	
MD5	RFC 1321	D	

비밀번호 저장을 위한 해시 함수

안전한 비밀번호 해싱을 위해서는 전용 해시 함수를 반드시 사용해야 한다. 이러한 느린 해싱 (slow-hashing) 알고리즘은 비밀번호 크래킹의 연산 난이도를 높여 무차별 대입 공격과 사전 대입 공격을 완화한다.

KDF	참조	필수 매개변수	상태
argon2id	RFC 9106	t = 1: m ≥ 47104 (46 MiB), p = 1	A
		t = 2: m ≥ 19456 (19 MiB), p = 1	A
		t ≥ 3: m ≥ 12288 (12 MiB), p = 1	A
scrypt	RFC 7914	p = 1: N ≥ 2^17 (128 MiB), r = 8	A
		p = 2: N ≥ 2^16 (64 MiB), r = 8	A
		p ≥ 3: N ≥ 2^15 (32 MiB), r = 8	A
bcrypt	A Future-Adaptable Password Scheme	cost ≥ 10	A
PBKDF2-HMAC-SHA-512	NIST SP 800-132, FIPS 180-4	iterations ≥ 210,000	A
PBKDF2-HMAC-SHA-256	NIST SP 800-132, FIPS 180-4	iterations ≥ 600,000	A
PBKDF2-HMAC-SHA-1	NIST SP 800-132, FIPS 180-4	iterations ≥ 1,300,000	L

승인된 비밀번호 기반 키 유도 함수 (Password-based Key Derivation Function; PBKDF) 는 비밀번호 저장에 사용할 수 있다.

키 유도 함수 (Key Derivation Functions; KDFs)

범용 키 유도 함수

KDF	참조	상태
HKDF	RFC 5869	A
TLS 1.2 PRF	RFC 5248	L
MD5 기반 KDFs	RFC 1321	D
SHA-1 기반 KDFs	RFC 3174 & RFC 6194	D

비밀번호 기반 키 유도 함수

KDF	참조	필수 매개변수	상태
argon2id	RFC 9106	t = 1: m ≥ 47104 (46 MiB), p = 1	A
		t = 2: m ≥ 19456 (19 MiB), p = 1	A
scrypt	RFC 7914	p = 1: N ≥ 2^17 (128 MiB), r = 8	A
		p = 2: N ≥ 2^16 (64 MiB), r = 8	A
		p ≥ 3: N ≥ 2^15 (32 MiB), r = 8	A
PBKDF2-HMAC-SHA-512	NIST SP 800-132, FIPS 180-4	iterations ≥ 210,000	A
PBKDF2-HMAC-SHA-256	NIST SP 800-132, FIPS 180-4	iterations ≥ 600,000	A
PBKDF2-HMAC-SHA-1	NIST SP 800-132, FIPS 180-4	iterations ≥ 1,300,000	L

키 교환 메커니즘

이 문단은 V11.6 공개 키 암호에 대한 추가 정보를 제공한다.

키 교환 체계

모든 키 교환 체계에서 최소 112 비트 이상의 보안 강도를 반드시 보장해야 하며, 구현 시 아래 표의 매개변수 선택을 따라야 한다.

스키마	도메인 매개변수	순방향 보안	상태
유한체 디피-헬만 (Finite Field Diffie-Hellman; FFDH)	L >= 3072 & N >= 256	예	A
타원 곡선 디피-헬만 (Elliptic Curve Diffie-Hellman; ECDH)	f >= 256-383	예	A
RSA-PKCS#1 v1.5 기반 암호화 키 전송		아니오	D

다음은 매개변수 정의이다:

- k 는 RSA 키의 크기이다.
- L 은 유한체 암호 기법에서 공개 키의 크기이며, N 은 개인 키의 크기이다.
- f 는 ECC 에서 키 크기의 범위이다.

새로운 구현에서는 NIST SP 800-56A, NIST SP 800-56B, NIST SP 800-77 표준을 준수하지 않는 어떤 체계도 사용해서는 안 된다. 특히, IKEv1 은 운영 환경에서 사용해서는 안 된다.

디피-헬만 그룹

다음 그룹들은 디피-헬만 키 교환 구현 시 승인된 그룹이다. 보안 강도에 대한 상세 내용은 NIST SP 800-56A 부록 D 및 NIST SP 800-57 Part 1 Rev.5에 문서화되어 있다.

그룹	상태
P-224, secp224r1	A
P-256, secp256r1	A
P-384, secp384r1	A
P-521, secp521r1	A
K-233, sect233k1	A
K-283, sect283k1	A
K-409, sect409k1	A
K-571, sect571k1	A
B-233, sect233r1	A
B-283, sect283r1	A
B-409, sect409r1	A
B-571, sect571r1	A
Curve448	A
Curve25519	A
MODP-2048	A
MODP-3072	A
MODP-4096	A
MODP-6144	A
MODP-8192	A
ffdhe2048	A

그룹	상태
ffdhe3072	A
ffdhe4096	A
ffdhe6144	A
ffdhe8192	A

메시지 인증 코드 (Message Authentication Codes; MAC)

메시지 인증 코드 (MAC) 는 메시지의 무결성과 진위 여부를 검증하기 위해 사용되는 암호학적 구성 요소이다. MAC 은 메시지와 비밀 키를 입력으로 받아 고정 길이의 태그 (MAC 값) 를 생성한다. MAC 는 TLS, SSL 등과 같은 보안 통신 프로토콜에서 널리 사용되며, 통신 당사자 간에 교환되는 메시지가 인증되었고 변경되지 않았음을 보장한다.

MAC 알고리즘	참조	상태
HMAC-SHA-256	RFC 2104 & FIPS 198-1	A
HMAC-SHA-384	RFC 2104 & FIPS 198-1	A
HMAC-SHA-512	RFC 2104 & FIPS 198-1	A
KMAC128	NIST SP 800-185	A
KMAC256	NIST SP 800-185	A
BLAKE3 (keyed_hash mode)	BLAKE3 one function, fast everywhere	A
AES-CMAC	RFC 4493 & NIST SP 800-38B	A
AES-GMAC	NIST SP 800-38D	A
Poly1305-AES	The Poly1305-AES message-authentication code	A
HMAC-SHA-1	RFC 2104 & FIPS 198-1	L
HMAC-MD5	RFC 1321	D

디지털 서명

서명 체계는 반드시 NIST SP 800-57 Part 1에서 승인한 키 크기와 매개변수를 사용해야 한다.

서명 알고리즘	참조	상태
EdDSA (Ed25519, Ed448)	RFC 8032	A
XEdDSA (Curve25519, Curve448)	XEdDSA	A

서명 알고리즘	참조	상태
ECDSA (P-256, P-384, P-521)	FIPS 186-4	A
RSA-RSSA-PSS	RFC 8017	A
RSA-SSA-PKCS#1 v1.5	RFC 8017	D
DSA (키 길이 무관)	FIPS 186-4	D

양자 내성 암호 표준

양자 내성 암호 (Post-quantum cryptography; PQC) 구현은 FIPS-203, FIPS-204, FIPS-205를 준수해야 한다. 현재 이들 표준에 대해 보안이 강화된 코드 예제나 참조 구현은 아직 많지 않다. 자세한 내용은 NIST announcement of the first three finalized post-quantum encryption standards (August 2024)을 참조하라.

제안된 mlkem768x25519 양자 내성 하이브리드 TLS 키 합의 방식은 Firefox release 132와 Chrome release 131과 같은 주요 브라우저에서 지원된다. 이 방식은 암호화 테스트 환경에서 사용하거나, 산업계 또는 정부에서 승인한 라이브러리에서 사용할 수 있다.

부록 D: 권장 사항

소개

애플리케이션 보안 검증 표준 (Application Security Verification Standard; ASVS) 버전 5.0 을 준비하면서, 몇 가지 기존 항목과 새로이 제안된 항목들이 버전 5.0 의 요구사항으로서 포함되기에 부적절하다는 점이 확실해졌다. 이는 버전 5.0 의 정의에 따라 해당 항목들이 더는 ASVS 의 범위에 있지 않거나, 유용하지만 필수 사항으로 지정하기에는 어려움이 있기 때문이다.

해당 항목들이 누락되지 않도록 일부 항목을 본 부록에 수록하였다.

권장되는 범위 내 매커니즘

다음 항목들은 ASVS 의 범위 내에 있으며, 필수적이지는 않으나 안전한 애플리케이션의 일부로서 고려하는 것이 강력하게 권장된다.

- 사용자들이 더 강력한 비밀번호를 설정할 수 있도록 비밀번호 강도 측정기 (password strength meter) 를 제공하는 것이 권장된다.
- 애플리케이션의 루트 또는.well-known 디렉터리에 공개적으로 접근 가능한 security.txt 파일을 생성하여, 보안 문제에 관해 소유자에게 연락할 수 있는 링크 또는 이메일 주소를 명확히 정의해야 한다.
- 신뢰할 수 있는 서비스 계층에서의 검증과 함께 클라이언츠 측에서의 입력값 검증을 강제하는 것이 권장된다. 이는 누군가가 애플리케이션을 공격하기 위해 클라이언트 측 통제의 우회 여부를 탐지할 수 있는 기회를 제공한다.

- robots.txt 파일, X-Robots-Tag 응답 헤더 또는 robots html meta tag 를 사용하여 우연히 접근될 수 있는 민감한 페이지가 검색 엔진에 노출되는 것을 막아야 한다.
- GraphQL 을 사용하는 경우, 모든 개별 인터페이스마다의 권한 부여를 관리하지 않아도 않도록 GraphQL 이나 리졸버 계층이 아닌 비즈니스 로직 계층에서 권한 부여 로직을 구현해야 한다.

참조:

- security.txt 에 관한 자세한 정보 (RFC 링크 포함)

소프트웨어 보안 원칙

다음의 항목들은 ASVS 의 이전 버전에 포함되어 있었으나 실제 요구사항은 아니다. 보안 통제를 구현할 때 고려해야 할 원칙에 가까우며, 이를 따랐을 때 더 강건한 보안 통제를 구현할 수 있다. 해당하는 항목은 다음과 같다:

- 보안 통제는 중앙집중화되고, 단순하고 (설계의 경제성), 검증 가능하도록 안전하며, 재사용 가능할 것이 권장된다. 이로써 중복, 누락, 또는 비효율적인 통제를 방지할 수 있다.
- 통제를 처음부터 구현하기보다는, 가능한 사전에 작성되어 있으며 철저히 검증된 보안 통제 구현을 사용하는 것이 좋다.
- 이상적으로 보호된 데이터와 리소스에 접근할 때 단일 접근 통제 매커니즘을 사용하는 것이 권장된다. 복사, 붙여넣기 또는 안전하지 않은 대체 경로를 방지하기 위해 모든 요청은 해당 단일 매커니즘을 통과하는 것이 권장된다.
- 속성 기반 또는 기능 기반의 접근 통제는 권장되는 패턴이며, 이 패턴에서는 코드가 단순히 사용자의 역할만 확인하기보다는 기능이나 데이터 항목에 대한 사용자의 권한을 검사한다. 권한은 여전히 역할 기반으로 할당하는 것이 권장된다.

소프트웨어 보안 프로세스

몇 가지의 보안 프로세스는 ASVS 5.0 에서 제외되었지만 여전히 유용하다. 이 프로세스들을 효과적으로 구현하는 방법은 OWASP SAMM 프로젝트에서 확인할 수 있다. 이전 버전의 ASVS 에 포함되었던 항목은 다음과 같다:

- 모든 개발 단계에서 보안을 다루는 안전한 소프트웨어 개발 생명 주기를 사용하고 있는지 확인해야 한다.
- 모든 설계 변경 또는 스프린트 계획에서의 위협 모델링이 사용되는지 확인해야 한다. 이는 위협을 식별하고, 대응책을 계획하고, 적절한 위험 대응을 촉진하며, 보안 검사를 안내하기 위함이다.
- 모든 사용자 스토리와 기능이 보안 제약 조건을 포함하는지 확인해야 한다. 보안 제약 조건의 예시는 다음과 같다. “사용자로서, 나는 나의 프로필을 조회하고 수정할 수 있어야 한다. 또한 나는 다른 사람의 프로필을 조회하거나 수정할 수 없어야 한다.”
- 모든 개발자와 테스터가 시큐어 코딩 체크리스트, 보안 요구사항, 가이드라인 또는 정책을 사용할 수 있는지 확인해야 한다.
- 애플리케이션 소스 코드가 백도어, 악성 코드 (예: 살라미 공격, 논리 폭탄, 시한 폭탄), 문서화되지 않았거나 숨겨진 기능 (예: 이스터 에그, 안전하지 않은 디버깅 도구) 로부터 안전함을 확인하는 프로세스가 지속적으로 수행되고 있는지 확인해야 한다. 이를 준수하기 위해서는 서드 파티 라이브러리를 포함한 소스 코드에 대한 완전한 접근이 필요하므로 최고 수준의 보안을 요구하는 애플리케이션에만 적합할 가능성이 크다.

- 배포된 환경에서 설정 편차를 탐지하고 응답할 수 있는 매커니즘이 존재하는지 확인해야 한다. 이는 불변 인프라의 사용, 안전 기준선에서의 자동화된 재배포, 승인된 설정과 현재의 상태를 비교하는 드리프트 탐지 도구 (drift detection tools) 를 포함할 수 있다.
- 모든 서드파티 제품, 라이브러리, 프레임워크 및 서비스에 대해, 개별적인 권장 사항에 따라 보안 강화가 수행되는지 확인해야 한다.

참조:

- OWASP Threat Modeling Cheat Sheet
- OWASP Threat modeling
- OWASP Software Assurance Maturity Model Project
- Microsoft SDL

부록 E - 기여자들

우리는 ASVS 4.0.0 릴리즈 이후부터 의견을 제공하거나 풀 리퀘스트 (pull request) 를 올려준 다음의 기여자들에게 감사한다.

오류를 발견하거나 이름이 다른 방식으로 표기되길 바랄 경우 알려주기 바란다.

Johan Sydseter (sydseter)	luis servin (lfservin)	Oleksii Dovydkov (oleksiidov)	IZUKA Masahiro (maizuka)
James Sulinski (jsulinski)	Eli Saad (ThunderSon)	kkshitish9	Andrew van der Stock (vanderaj)
Rick M (kingthorin)	Bankde Eakasit (Bankde)	Michael Gargiullo (mgargiullo)	Raphael Dunant (Racater)
Cesar Kohl (cesarkohl)	inaz0	Joerg Bruenner (JoergBruenner)	David Deatherage (securitydave)
John Carroll (yosignals)	Jim Fenton (jimfenton)	Matteo Pace (M4tteoP)	Sebastien gioria (SPoint42)
Steven van der Baan (vdbaan)	Jeremy Bonghwan Choi (jeremychoi)	craig-shony	Riccardo Sirigu (ricsirigu)
Tomasz Wrobel (tw2as)	Alena Dubeshko (belalena)	Rafael Green (RafaelGreen1)	mjang-cobalt
clallier94	Kevin W. Wall (kwwall)	Jordan Sherman (jsherm-fwdsec / deleterepo)	Ingo Rauner (ingo-rauner)
Dirk Wetter (drwetter)	Moshe Zioni (moshe-apiiro)	Patrick Dwyer (coderpatros)	David Clarke (davidclarke-au)

Takaharu Ogasa (takaharuogasa)	Arkadii Yakovets (arkid15r)	Motoyasu Saburi (motoyasu-saburi)	leirn
wet-certitude	timhemel	RL Thornton (thornshadow99)	Thomas Bandt (aspnetde)
Roel Storms (roelstorms)	Jeroen Willemsen (commjoen)	anonymous-31	Kamran Saifullah (deFr0ggy)
Steve Springett (stevespringett)	Spyros (northdpole)	Hans Herrera (hansphp)	Marx314
CarlosAllendes	Yonah Russ (yruss972)	Sander Maijers (sanmai-NL)	Luboš Bretschneider (bretik)
Eva Sarafianou (esarafianou)	Ata Seren ataseren	Steve Thomas (Sc00bz)	Dominique RIGHETTO (righettod)
Steven van der Baan (svdb-ncc)	Michael Vacarella (Aif4thah)	Tonimir Kisasondi (tkisason)	Stefan Streichsbier (streichsbaer)
hi-unc1e	sb3k (starbuck3000)	mario-platt	Devdatta Akhawe (devd)
Michael Gissing (scolytus)	Jet Anderson (thatsjet)	Dave Wichers (davewichers)	Jonny Schnittger (JonnySchnittger)
Silvia Väli (silviavali)	jackgates73	1songb1rd	Timur - (timurozkul)
Gareth Heyes (hackvertor)	appills	suvikaartinen	chaals (chaals)
DanielPharos (AtlasHackert)	will Farrell (willfarrell)	Alina Vasiljeva (avasiljeva)	Paul McCann (ismisepaul)
Sage (SajjadPourali)	rbsec	Benedikt Bauer (mastacheata)	James Jardine (jamesjardine)
Mark Burnett (m8urnett)	dschwarz91	Cyber-AppSec (Cyber-AppSec)	Tib3rius
BitnessWise (bitnesswise)	damienbod (damienbod)	Jared Meit (jmeit-fwdsec)	Stefan Seelmann (sseelmann)
Brendan O'Connor (ussjoin)	Andrei Titov (andrettv)	Hans-Petter Fjeld (atluxity)	markehack
Neil Madden (NeilMadden)	Michael Geramb (mgeramb)	Osama Elnaggar (ossie-git)	mackowski
Ravi Balla (raviballa)	Hazana (hazanasec)	David Means (dmeans82)	Alexander Stein (tohch4)

BaeSenseii (baesenseii)	Vincent De Schutter (VincentDS)	S Bani (sbani)	Mitsuaki Akiyama (mak1yama)
Christopher Loessl (hashier)	victorxm	Michal Rada (michalradacz)	Veeresh Devireddy (drveresh)
MaknaSEO	darkzero2022	Liam (LiamDobbelaere)	Frank Denis (jedisct1)
Otto Sulin (ottosulin)	carllaw6885	Anders Johan Holmefjord (aholmis)	Richard Fritsch (rfricz)
mesutgungor	Scott Helme (ScottHelme)	Carlo Reggiani (carloreggiani)	Suyash Srivastava (suyash5053)
Mark Potter (markonweb)	Arjan Lamers (alamers)	Gøran Breivik (gobrtg)	flo-blg
Guillaume Déflache (guillaume-d)	Toufik Airane (toufik-airane)	Keith Hoodlet (securingdev)	Sinner (SoftwareSinner)
iloving	Jeroen Beckers (TheDauntless)	Joubin Jabbari (joubin)	yu fujioka (fujiokayu)
execjosh (execjosh)	Alicja Kario (tomato42)	Sidney Ribeiro (srjsoftware)	Gabriel Marquet (Gby56)
Drew Schulz (drschulz)	bedirhan	muralito	Ronnie Flathers (ropnop)
Philippe De Ryck (philippederyck)	Malte (mal33)	MazeOfThoughts	Andreas Falk (andifalk)
Javi (javixeneize)	Daniel Hahn (averell23)	borislav-c	Robin Wood (digininja)
miro2ns	Jan Dockx (jandockx)	vipinsaini434	priyanshukumar397
Nat Sakimura (sakimura)	Benjamin Häublein (BenjaminHae)	unknown-user-from	Ali Ramazan TAŞDELEN (alitasdln)
Pedro Escaleira (oEscal)	Josh (josh-hemphill)	Tim Würtele (SECtim)	AviD (avidouglen)
SheHacksPurple (shehackspurple)	fcerullo-cycubix	Hector Eryx Paredes Camacho (heryxpc)	Irene Michlin (irene221b)
Jonah Y-M (TG-Techie)	Dhiraj Bahroos (bahroos)	Jef Meijvis (jefmeijvis)	IzmaDoesItbeta
Abdessamad TEMMAR (TmmmmmR)	sectroyer	Soh Satoh (sohsatoh)	regoravalaz

james-t (james-bitherder)	Aram Hovsepyan (aramhovsepyan)	JaimeGomezGarciaSan	ValdiGit01
iwatachan (ishowta)	Vinod Anandan (VinodAnandan)	Kevin Kien (KevinKien)	paul-williamson-swoop
endergzr	Radhwan Alshamamri (Rado0z)	Grant Ongers (rewtd)	Cure53 (cure53)
AliR2Linux	Ads Dawson (Gang-GreenTemperTatum)	William Reyor (BillReyor)	gabe (gcrow)
mascotter	luissaiz	Suren Manukyan (vx-sec)	Piotr Gliźniewicz (pglizniewicz)
Tadeusz Wachowski (tadeuszwachowski)	Nasir aka Nate (andesec)	settantasette	Lars Haulin (LarsH)
Terence Eden (edent)	JasmineScholz	Arun Sivadasan (teavanist)	Yusuf GÜR (yusuffgur)
Troy Marshall (troymarshall)	Tanner Prynn (tprynn)	Nick K. (nickific)	raoul361
Azeem Ilyas (TheAxZim)	Evo Stamatov (avioli)	Tim Potter (timpotter87)	Gavin Ray (GavinRay97)
monis (demideus)	Marcin Hoppe (MarcinHoppe)	Grambulf (ramshazar)	Jordan Pike (computersarebad)
Jason Rogers (jason-invision)	Ben Hall (benbhall)	JamesPoppyCock (jamesly123)	WhiteHackLabs (whitehacklabs)
Alex Gaynor (alex)	Filip van Laenen (filipvanlaenen)	jeurgen	GraoMelo
Andreas Kurtz (ay-kay)	Tom Tervoort (TomTervoort)	old man (deveras)	Marco Schnüriger (marcortw)
stiiin	infoseclearn (teaminfoseclearn)	hljupkij	Noe (nmarher)
Lyz (lyz-code)	Martin Riedel (mrtnrdl)	KIM Jaesuck (tcaesvk)	Barbara Schachner (bschach)
René Reuter (AresSec)	carhackpils	Tyler (tyler2cr)	Hugo (hasousa)
Wouter Bloeyaert (Someniak)	Mark de Rijk (markderijkinfosec)	Ramin (picohub)	Philip D. Turner (philipdturner)
Will Chatham (willc)			