OWASP TOP 10

10 대 가장 심각한
웹 어플리케이션 보안 취약점

2007 최신판

© 2002-2007 OWASP Foundation

This document is licensed under the Creative Commons Attribution-ShareAlike 2.5 license

목차

소개

2007 년 OWASP Top10 에 오신 것을 환영합니다. 완전히 재작성된 이번 최신판은 가장 심각한 웹 애플리케이션 취약점에 대해 완전히 새롭게 재작성하였으며, 취약점에 대한 방어 방법에 대해 논의하고, 보다 많은 정보에 대한 웹 사이트 주소 연결을 제공하고 있습니다.

목표

OWASP Top10 의 기본 목표는 가장 공통적인 웹 애플리케이션 보안 취약점의 결과에 대해 **개발자와, 설계자, 아키텍트와 조직들을 교육시키는 것입니다.** Top10 은 보안적으로 안전한 프로그램을 코딩하는데 있어 훌륭한 출발점이며, 이러한 취약점을 방어할 수 있는 기본적인 방법을 제공합니다.

보안은 일회성 이벤트가 되어서는 안 됩니다. 여러분의 코드를 단지 한번 안전하게 하는 것만으로는 불충분합니다. 2008 년에 이 Top 10 은 변경될 것이고, 여러분의 애플리케이션 코드가 변경되지 않는다면, 아마도 여러분은 보안에 취약한 상태에 놓이게 될 것입니다. 더 많은 정보를 위해서는 *추가 정보란*에 있는 조언들을 참고하길 바랍니다.

보안적으로 안전한 코딩의 시작은 프로그램의 모든 라이프 싸이클 단계에서 다루어져야 합니다. 안전한 웹 애플리케이션은 안전한 SDLC(Software Development Lifecycle)가 사용될 때에만 가능합니다. 안전한 프로그램은 설계, 개발, 그리고 전체 과정에 걸쳐 기본적으로 보안을 고려해야 합니다. 웹 애플리케이션의 전체 보안에 영향을 주는 최소한 300 개의 문제가 있습니다. 이러한 300 개 넘는 항목들은 OWASP 지침서에 열거되어 있으며 이 지침서는 오늘날 웹 애플리케이션을 개발하는 사람이라면 누구든지 필독해야 할 내용을 담고 있습니다.

이 문서는 최초이자 주요한 교육 자료일 뿐 표준 자료는 아닙니다. 먼저 OWASP 와 논의없이 이 문서를 정책이나 표준 자료로 채택하지 않길 바랍니다! 만일 보안적으로 안전한 코딩에 대한 정책 혹은 표준이 필요하다면, OWASP 가 보안 코딩 정책과 표준에 관한 프로젝트를 진행하고 있습니다. 이러한 노력에 동참하거나 혹은 재정적으로 지원하는 것을 고려해 주었으면 합니다.

감사의 글

CVE의 취약점 유형 배포 데이터를 사용할 수 있게 해 준 MITRE에 감사합니다.

OWASP Top 10 프로젝트는 Aspect Security사가 지휘하고 후원한 것입니다.

프로젝트 리더 : Andrew van der Stock (OWASP 재단 집행 이사)

공동저자 : Jeff Willams(OWASP 재단 의장), Dave Wichers(OWASP 재단 컨퍼런스 의장)

우리는 다음 검토자들에게 감사하고자 합니다:

- Raoul Endres 는 다시금 Top10 이 진행될 수 있도록 도와 주었고, 귀중한 의견을 주었습니다.
- Steve Christey(MITRE)는 포괄적인 비평을 하고 MITRE CWE 데이터를 추가해 주었습니다.
- Jeremiah Grossman(WhiteHat Security)은 비평을 해 주었고 자동화된 감지 도구에 관한 정보를

제공해 주었습니다.

- Sylvan von Stuppe 는 모범적인 비평을 해 주었습니다.
- Colin Wong, Nigel Evans, Andre Gironda 와 Neil Smithline 은 이메일로 의견을 주었습니다.

OWASP TOP 10 한글 번역 프로젝트는 네이버 보안 카페 SecurityPlus에서 2007 년부터 수행하고 있습니다. 한글 번역상의 문제점이나 제안은 mirrk1@gmail.com 으로 메일 주시기 바랍니다.

요약

A1 – 크로스사이트 스크립팅(XSS)	XSS 취약점은 콘텐츠를 암호화나 검증하는 절차 없이 사용자가 제공하는 데이터를 어플리케이션에서 받아들이거나, 웹 브라우저로 보낼 때마다 발생한다. XSS 는 공격자가 희생자의 브라우저 내에서 스크립트를 실행하게 허용함으로써 사용자 세션을 가로채거나, 웹 사이트를 손상하거나 웜을 심는 것 등을 가능하게 할 수 있다.
A2 – 인젝션 취약점	인젝션 취약점, 특히 SQL 인젝션 취약점은 웹 애플리케이션에서 매우 흔하다. 인젝션은 사용자가 입력한 데이터가 명령어나 질의문의 일부분으로 인터프리터에 보내질 때 발생한다. 악의적인 공격자가 삽입한 데이터에 대해 인터프리터는 의도하지 않은 명령어를 실행하거나 데이터를 변경할 수 있다.
A3 – 악성 파일 실행	원격 파일 인젝션(RFI)에 취약한 코드는 공격자가 악의적인 코드와 데이터의 삽입을 허용함으로써 전체 서버 훼손과 같은 파괴적인 공격을 가할 수 있다. 악성 파일 실행 공격은 PHP, XML, 그리고 사용자로부터 파일명이나 파일을 받아들이는 프레임워크에 영향을 준다.
A4 – 불안전한 직접 객체 참조	직접 객체 참조는 개발자가 파일, 디렉토리, 데이터베이스 기록 혹은 키 같은 내부 구현 객체에 대한 참조를 URL 혹은 폼 매개변수로 노출시킬 때 발생한다. 공격자는 이러한 참조를 조작해서 승인 없이 다른 객체에 접속할 수 있다.
A5 – 크로스 사이트 요청 변조(CSRF)	CSRF 공격은 로그온 한 희생자의 브라우저가 사전 승인된 요청을 취약한 웹 애플리케이션에 보내도록 함으로써 희생자의 브라우저가 공격자에게 이득이 되는 악의적인 행동을 수행하도록 한다. CSRF 는 자신이 공격하는 웹 애플리케이션이 강력하면 할수록 강력해진다.
A6 – 정보 유출 및 부적절한 오류 처리	애플리케이션은 다양한 애플리케이션 문제점을 통해 의도하지 않게 자신의 구성 정보, 내부 작업에 대한 정보를 누출하거나 또는 개인정보 보호를 위반할 수 있다. 공격자는 이러한 약점을 사용해서 민감한 정보를 훔치거나 보다 심각한 공격을 감행한다.
A7 – 취약한 인증 및 세션 관리	자격 증명과 세션 토큰은 종종 적절히 보호되지 못한다. 공격자는 다른 사용자인 것처럼 보이게 하기 위하여 비밀번호, 키, 혹은 인증 토큰을 손상시킨다.
A8 – 불안전한 암호화 저장	웹 애플리케이션은 데이터와 자격 증명을 적절히 보호하기 위한 암호화 기능을 거의 사용하지 않는다. 공격자는 약하게 보호된 데이터를 이용하여 신원 도용이나 신용카드 사기와 같은 범죄를 저지른다.
A9 – 불안전한 통신	애플리케이션은 민감한 통신을 보호할 필요가 있을 때 네트워크 트래픽을 암호화하는데 종종 실패한다.
A10 – URL 접속 제한 실패	애플리케이션이 권한 없는 사용자에게 연결 주소나 URL 이 표시되지 않도록 함으로써, 민감한 기능들을 보호한다. 공격자는 이러한 약점을 이용하여 이 URL 에 직접 접속함으로써 승인되지 않은 동작을 수행한다.

표 1: 2007 년 Top10 웹 애플리케이션 취약점

방법론

2007 년 Top10 을 위한 방법론은 단순하다. 2006 년 MITRE 취약점 (MITRE Vulnerability Trends for 2006) 경향을 참고하여, Top10 *웹 애플리케이션 보안* 문제점을 뽑아냈다. 결과는 아래와 같다:

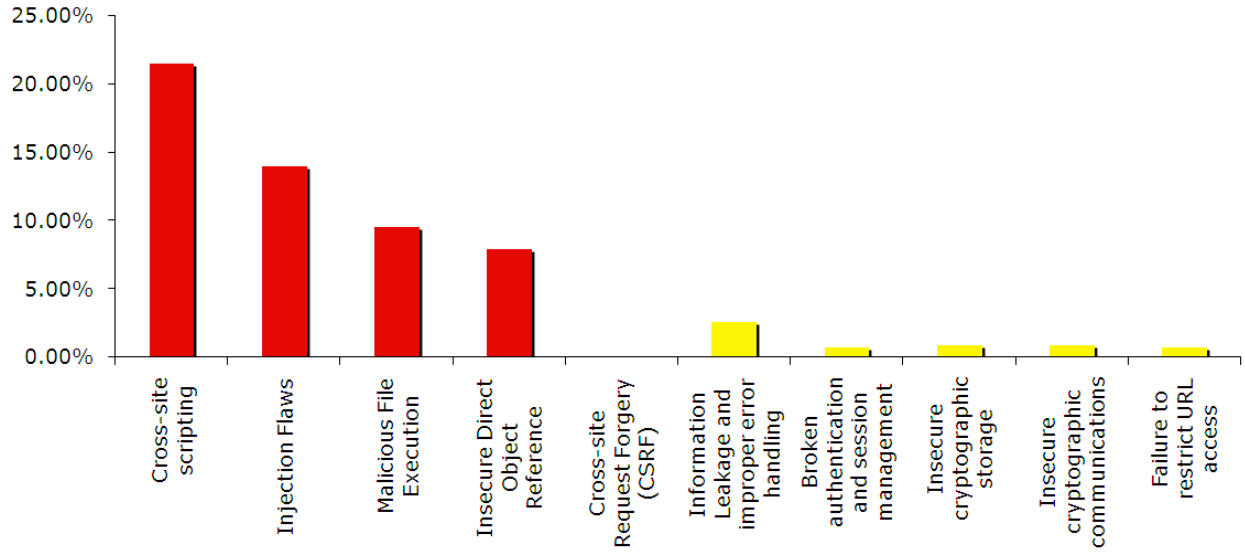

그림 2: 2006 년 Top10 웹 애플리케이션 취약점에 관한 MITRE 데이터

MITRE 본래의 취약점 데이터를 각 항목으로 하나 하나씩 보존하여 상세하게 열거하려 하였으나, 근본 원인과 밀접하게 나열할 수 있도록 신중하게 범주 일부를 변경하였다. 만일 최종 2006 년 MITRE 의 최종 원시 데이터에 관심이 있다면, 우리는 OWASP Top10 웹 사이트에 엑셀 작업표를 포함시킬 것이다.

모든 방어 권고사항들은 가장 중요한 3 가지의 유력한 웹 애플리케이션 프레임워크 즉, Java EE, ASP .NET 과 PHP 를 위한 해결책을 제공하고 있다. Ruby on Rails 이나 Perl 과 같은 다른 공통적인 웹 애플리케이션 프레임워크은 그들만의 특정한 요구 사항에 맞도록 권고사항을 쉽게 적용할 수 있을 것이다.

왜 일부 중요한 문제점들을 누락시켰는가

검증 없는 입력은 모든 개발팀에게 중요한 난제이고, 많은 애플리케이션 보안 문제의 근본이 된다. 사실상 목록에 있는 다른 많은 항목들은 해결책의 일부분으로써 입력을 검증할 것을 권고하고 있다. 우리는 여전히 집중화된 입력 검증 메커니즘을 웹 애플리케이션의 일부분으로 참조할 것을 강력하게 권고한다. 보다 많은 정보를 원하면, OWASP 에서 데이터 확인 기사를 읽어보기 바란다:

- http://www.owasp.org/index.php/Data_Validation
- http://www.owasp.org/index.php/Testing_for_Data_Validation

버퍼 오버플로우, 정수 오버플로우, 포맷 스트링 문제점은 C 나 C++ 같은 언어로 작성된 프로그램에 있어서는 매우 심각한 취약점이다. 이러한 문제점을 위한 권고사항은 SANS, CERT 그리고 프로그래밍 언어 툴 제공업체와 같은 전통적인 비-웹 애플리케이션 보안 커뮤니티에서 다뤄지고 있다. 만일 여러분의 코드가 버퍼 오버플로우를 경험할 수 있는 언어로 작성된다면, 우리는 OWASP 에서 버퍼 오버플로우에 관한 내용을 읽으라고 권하고 싶다:

- http://www.owasp.org/index.php/Buffer_overflow
- http://www.owasp.org/index.php/Testing_for_Buffer_Overflow

서비스 거부는 어떠한 언어로 작성된 어떠한 사이트에서도 영향을 미칠 수 있는 심각한 공격이다. MITRE 에 의한 DoS 의 순위가 금년에는 Top10 에 들기에는 부족했다. 만일 서비스 거부에 대해 관심이 있다면, OWASP 사이트와 테스팅 가이드를 참조하길 바란다:

- http://www.owasp.org/index.php/Category:Denial_of_Service_Attack
- http://www.owasp.org/index.php/Testing_for_Denial_of_Service

불안전한 구성 관리는 어느 정도까지는 모든 시스템, 특히 PHP 에 영향을 미친다. 하지만 이 문제점도 금년에는 MITRE 의 순위에는 들지 못했다. 애플리케이션을 배치할 때, 안전한 구성과 테스팅에 관한 상세 정보를 담고 있는 최신 OWASP 가이드와 OWASP 테스팅 가이드를 참조하길 바란다:

- http://www.owasp.org/index.php/Configuration
- http://www.owasp.org/index.php/Testing_for_infrastructure_configuration_management

왜 일부 중요한 이슈들을 추가했는가

크로스사이트 리퀘스트 변조(CSRF)는 금번 OWASP Top10 최신판에 새로이 추가된 부분이다. 비록 원시 데이터에서는 36 번째 순위를 차지했지만, 오늘날 애플리케이션이 방어에 대한 노력을 시작해야 할 정도로 충분히 중요하며, 특히 높은 가치의 애플리케이션과 민감한 데이터를 다루는 애플리케이션에 있어서는 더욱더 그렇다. CSRF 는 현재 순위가 시사하는 것보다 널리 퍼져있으며, 매우 위험할 수 있다.

암호화. 암호화의 불안전한 사용은 MITRE 의 원시 데이터에 따르면 웹 애플리케이션 보안 이슈에 있어서 8 위와 9 위일 뿐만 아니라, 이것은 많은 개인정보 유출과 컴플라이언스 문제점 (특히 PCI DSS 1.1 컴플라이언스) 의 근본 원인을 제공한다.

공격이 아닌 취약점들

과거 Top10 의 공격, 취약점, 대응 방법들을 혼합하여 담고 있었다. 이번 기회에는 취약점에 대해서만 초점을 맞추고 있지만 일반적으로 사용되는 전문용어에는 때때로 취약점과 공격의 뜻을 포함하고 있다. 만일 조직에서 자신들의 애플리케이션을 안전하게 하고 그들의 사업의 위험부담을 줄이기 위하여 이 문서를 사용한다면 아래의 문제점들에 있어 바로 위험부담을 줄여줄 것이다:

- XSS와 같은 이런 취약점을 악용할 수 있는 피싱(Phishing) 공격과 약하거나 존재하지 않는 인증 또는 승인 확인(A1, A4, A7, A10)
- 불충분한 확인, 비즈니스 규칙, 그리고 취약한 권한 확인(A2, A4, A6, A7, A10)으로 인한 개인정보 보호 위반
- 불충분하거나 존재하지 않는 암호화 통제(A8과 A9), 원격 파일 삽입(A3), 인증, 비즈니스 규칙, 권한 확인(A4, A7, A10)을 통한 신원 도용
- 인젝션(A2)과 원격 파일 삽입(A3)을 통한 시스템 훼손, 데이터 변조, 혹은 데이터 파괴 공격
- 권한 없는 처리와 CSRF 공격(A4, A5, A7, A10)을 통한 금융 손실

- 위와 같은 취약점(A1…A10)에 노출로 인한 명예 훼손

일단 한 조직이 반응적인 통제에 대한 걱정으로부터 벗어나, 그들 사업에 적용되는 위험 부담을 줄이기 위해 사전 대책을 강구하고 전진한다면 정부 규제사항을 더욱더 준수하고, 운영 비용을 줄임으로써 결과적으로는 더욱더 든든하고 안전한 시스템을 구축할 수 있을 것이다.

다른 시각

위에서 언급한 방법론은 불가피하게 보안 연구자 커뮤니티의 Top10 경향의 발견과 다른 견해를 가질 수 있다. 이러한 발견 패턴은 실제 공격자들의 방법과 유사하며 특히 초보 수준의 공격자들("스크립키드")과 관련이 있다. Top10 의 취약점에 대해 여러분의 소프트웨어를 보호하는 것은 가장 일반적인 공격 형태들에 대해 어느 정도 방어해 줄 수 있지만 보다 중요한 것은 자신의 소프트웨어의 보안을 개선시키는 방향을 잡는데 도움이 된다는 점이다.

매핑

제목에도 변경이 있었으며 심지어는 내용면에서는 이전의 내용들에 더 가깝게 연결되었다. 우리는 더 이상 WAS XML 명명규칙을 사용하지 않는다. 왜냐하면 그것은 최근의 취약점, 공격, 그리고 대응방법들을 따라잡지 못하기 때문이다. 아래 표는 이번 판본과 2004 년 Top 10, 그리고 MITRE 의 원래 순위와 어떻게 연결되는지를 보여준다:

2007 년도 OWASP Top 10	2004 년도 OWASP Top 10	2006 년도 MITRE 순위
A1. 크로스 사이트 스크립팅 (XSS)	A4. 크로스 사이트 스크립팅 (XSS)	1
A2. 인젝션 취약점	A6. 인젝션 취약점	2
A3. 악성 파일 실행 (신규)		3
A4. 불안전한 직접 객체 참조	A2. 훼손된 접근 제어 (2007 년 Top10 에서는 분리됨)	5
A5. 크로스사이트 리퀘스트 변조 (CSRF) (신규)		36
A6. 정보 유출 및 부적절한 오류 처리	A7. 부적절한 오류 처리	6
A7. 훼손된 인증 및 세션 관리	A3. 훼손된 인증 및 세션 관리	14
A8. 불안전한 암호화 저장	A8. 불안전한 저장	8
A9. 불안전한 통신 (신규)	A10. 불안전한 구성 관리 부분에서 논의됨	8
A10. URL 접근 제한 실패	A2. 훼손된 접근 제어(2007 년 Top10 에서는 분리됨)	14
<2007 년에 삭제 >	A1. 검증되지 않은 입력	7
<2007 년에 삭제 >	A5. 버퍼 오버 플로우	4, 8, 10

2007 년도 OWASP Top 10	2004 년도 OWASP Top 10	2006 년도 MITRE 순위
<2007 년에 삭제 >	A9. 서비스 거부	17
<2007 년에 삭제 >	A10. 불안전한 구성 관리	29

A1 - 크로스 사이트 스크립팅 (XSS)

XSS 로 더 잘 알려진 크로스 사이트 스크립팅은 사실 HTML 인젝션의 한 부분이다. XSS 는 가장 잘 알려지고 가장 치명적인 웹 어플리케이션 보안 문제이다. XSS 취약점은 콘텐츠를 암호화나 검증하는 절차 없이 사용자가 제공하는 데이터를 어플리케이션에서 받아들이거나, 웹 브라우저로 보낼 때마다 발생한다.

XSS 는 공격자가 희생자의 브라우저에 스크립트를 실행할 수 있게 허용함으로써 사용자의 세션을 가로채거나, 웹 사이트 변조, 악의적인 콘텐츠 삽입, 피싱 공격 행위를 할 수 있고, 악의적인 스크립팅을 이용해 사용자의 브라우저를 가로챌 수 있다. 이러한 악의적인 스크립트는 대개 자바스크립트를 사용하지만 희생자의 브라우저에 의해 지원되는 어떠한 스크립팅 언어로도 이런 공격을 받을 수 있다.

영향을 받는 환경

모든 웹 어플리케이션 프레임워크들은 크로스 사이트 스크립팅에 취약하다.

취약점

잘 알려진 크로스 사이트 스크립팅의 세 가지 형태는 반사, 저장, 그리고 DOM 인젝션이다. 반사된 XSS 는 악용하기 가장 쉬우며, 페이지는 사용자에게 제공된 데이터를 직접 사용자에게 다시 돌려 줄 것이다:

```
echo $_REQUEST['userinput'];
```

저장된 XSS 는 악의적인 데이터를 파일, 데이터베이스 또는 기타 백엔드 시스템에 저장한 다음에 마지막 단계에 여과하지 않고 사용자에게 보여주는 것이다. 이것은 다른 많은 사람들이 개인으로부터 입력된 정보를 이용하는 CMS, 블로그 또는 공개 토론장과 같은 시스템에서는 매우 위험한 것이다.

DOM 을 이용한 XSS 공격들은 HTML 요소보다 사이트의 자바스크립트 코드와 변수들을 조작하는 것이다.

다른 한편으로 공격은 세 가지 방식을 모두 혼합하여 할 수 있다. 크로스 사이트 스크립팅이 위험한 것은 공격하는 방식이 아니라 공격이 가능하다는 것에 있다. 표준화 되지 않거나 예기치 못한 브라우저의 성향은 포착하기 어려운 공격 벡터를 이끌어 낼 수도 있다. XSS 은 잠재적으로 이러한 브라우저에서 사용하는 어떠한 구성 요소를 통해서도 수행될 수 있다.

공격은 보통 강력한 스크립팅 언어인 자바스크립트에서 구현된다. 자바스크립트 사용은 공격자가 새로운 요소들(악의적인 사이트로 자격 증명을 유출하는 코드를 로그인 페이지에 추가한 것)을 추가하는 것을 포함하여, 어떠한 주어진 페이지나 내부 DOM 을 조작하거나, 페이지가 보여지고 표현되는 방법을 삭제 또는 변경하는 것을 허용한다. 자바스크립트는 비록 오늘날 피해 사이트가 AJAX 을 사용하지 않음에도 불구하고 전형적으로 AJAX 기술을 사용하는 사이트에서 사용되는 XmlHttpRequest 사용을 허용한다.

XmlHttpRequest 를 사용하면 브라우저의 동일 소스 개시 정책을 우회할 뿐만 아니라, 희생자의 데이터를 악의적 사이트로 전달하거나 브라우저가 열려 있는 동안 복잡한 웜과 악의적인 좀비 생성이 가능하다. AJAX 공격은 위험한 크로스 사이트 요청 변조 (CSRF) 공격을 이행하기 위해 꼭 드러날 필요도 없고 사용자의 상호 작용을 필요로 하지 않는다 (A-5 참조).

보안 검증

목표는 어플리케이션 안의 모든 매개변수들이 유효한지 또는 HTML 페이지 안에 포함되기 전에 암호화되는지를 검증하는 것이다.

자동적인 접근방법: 자동화된 침투 테스트 도구들은 매개변수 삽입을 매개로 한 반사된 XSS 를 탐지하는 능력은 있으나 지속적으로 XSS 를 찾는 데는 종종 실패한다. 특히, 삽입된 XSS 벡터의 결과를 권한 확인을 통해 방어되고 있다면 더더욱 그렇다(예를 들어, 사용자에 의해 관리자가 나중에서야 볼 수 있는 악의적인 데이터를 입력 하였을 때 등).

수동적인 접근방법: 통합된 검증과 암호화 메카니즘을 사용한다면, 가장 효율적인 보안 검증 방법은 코드를 확인하는 것이다. 만약 분산하여 구현되어 있다면 검증을 하는데 상당히 많은 시간이 걸릴 것이다. 대부분의 어플리케이션의 공격 범위가 상당히 크기 때문에 테스트를 하는데 많은 시간이 걸린다.

방어

XSS 을 위한 최선의 방어책은 모든 입력 데이터의 "화이트리스트"와 출력 데이터의 적절한 암호화의 조합이다. 검증은 공격을 탐지하고, 암호화는 어떠한 성공적인 스크립트 삽입이 브라우저 내에서 동작하지 못하도록 할 것이다.

전체 어플리케이션에 걸쳐 XSS 방어를 위해서는 지속적인 구조적 접근이 요구된다:

- **입력값 검증.** 데이터가 보여지거나 저장되기 전에 길이, 형태, 문법과 비즈니스 규칙에 대한 모든 입력 데이터를 검증하기 위해 표준 입력값 검증 방식을 사용해라. "알고 있는 올바름을 수락"하는 검증 전략을 사용해라. 잠재적으로 악의적인 데이터를 삭제하려고 시도하는 것 보다는 검증되지 않은 입력을 거부해라. 에러 메시지에 검증되지 않은 데이터를 포함해야 한다는 것을 잊지 말아라.
- **강력한 출력값 암호화**. 매우 제한적인 부분이 아니라 모든 문자들이 암호화 되도록 하는 것이 해결 방법이며 모든 사용자에게 제공되는 데이터가 전달되기 전에 적절하게 암호화 되어 있는지 확인하라(출력 방식에 따른 HTML 또는 XML 모두). 이것은 Microsoft Anti-XSS 라이브러리와 이번 OWASP PHP Anti-XSS 라이브러리의 접근 방식이다. 또한 출력하는 각 페이지에 문자 암호화를 한다면 다른 변형 공격에 대한 노출을 감소시킬 것이다.
- **출력값 암호화(ISO 8859-1 또는 UTF 8 같은) 명시.** 공격자가 여러분의 사용자를 위해 출력값 암호화를 선택하도록 허용하지 마라.
- **입력 값에서 XSS를 탐지하거나 또는 출력값을 암호화하기 위한 "블랙리스트" 검증 방식은 사용하지 마라.** 몇몇 문자들을 ("<""">"나 "스크립트와 유사한 문자나 어구) 검색하고 대체하는 것은 취약하고 성공적으로 공격당할 수 있다. 심지어 확인하지 않는 "<b>"태그조차도 어떤 문맥에서는 안전하지 않다. XSS는 블랙리스트 검증방식을 쉽게 피할 수 있는 놀라울 정도 많은 변형 공격 방법을 가지고 있다.
- **일반적인 오류를 주의해라.** 입력 값들은 검증하기 전에 반드시 해독해야 하고, 어플리케이션의 현재 내부 표시를 복호화되어야 한다. 여러분의 어플리케이션이 같은 입력값을 두 번 해독하지 않도록 보증해라. 이러한 오류들은 확인 된 이후에 위험한 입력 값을 도입시켜서 "화이트리스트"의 계획을 우회하기 위하여 사용될 수도 있다.

언어별 추천사항들:

- **Java : <bean:write ...> 같은 Struts 결과값 방식을 사용해라.** 또는 <c:out ...> 내에 기본적인 JSTL escapeXML="true" 속성을 사용해라. 중첩되지 않는 <%= ... %> 을 사용하지 마라 (이것은 적당한 결과값 암호화 방식에서 벗어난다.).
- .NET : MSDN 으로부터 자유롭게 사용이 가능한 **Microsoft Anti-XSS 라이브러리 1.5 를 사용해라.** 이 라이브러리를 사용하지 않고 요청된 대상: usename.Text = Request.QueryString ("username") 을 직접적으로 폼 필드 데이터에 할당하지 마라. .NET 는 자동적으로 암호화된 출력 데이터를 조정한다.
- PHP: **출력값이 htmlentities() 또는 htmlspecialchars()를 통과하였는지 확인해라.** 또는 곧 발표 될 OWASP PHP Anti-XSS 라이브러리를 사용해라. register_globals 이 비활성되지 않았다면 비활성화하라.

예제

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4206
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3966
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5204

참조

- CWE: CWE-79, 크로스 사이트 스크립팅(XSS)
- WASC 위협 분류: http://www.webappsec.org/projects/threat/classes/cross-site_scripting.shtml
- OWASP – 크로스 사이트 스크립팅, http://www.owasp.org/index.php/Cross_Site_Scripting
- OWASP – XSS 테스팅, http://www.owasp.org/index.php/Testing_for_Cross_site_scripting
- OWASP Stinger 프로젝트 (Java EE 검증 필터) – http://www.owasp.org/index.php/Category:OWASP_Stinger_Project
- OWASP PHP 필터 프로젝트 - http://www.owasp.org/index.php/OWASP_PHP_Filters
- OWASP 인코딩 프로젝트 - http://www.owasp.org/index.php/Category:OWASP_Encoding_Project
- RSnake, XSS Cheat Sheet, http://ha.ckers.org/xss.html
- Klein, A., DOM 기반 크로스 사이트 스크립팅, http://www.webappsec.org/projects/articles/071105.shtml
- .NET Anti-XSS 라이브러리 - http://www.microsoft.com/downloads/details.aspx?FamilyID=efb9c819-53ff-4f82-bfaf-e11625130c25&DisplayLang=en

A2 - 인젝션 취약점

인젝션 취약점, 특히 SQL 인젝션은 웹 어플리케이션에서는 일반적이다. 인젝션의 종류에는 SQL, LDAP, XPath, XSLT, HTML, XML, OS 명령어 인젝션 등 여러 종류가 있다.

인젝션은 사용자가 입력한 데이터가 명령어나 질의어의 일부로써 인터프리터에 보내질 때 인젝션이 이뤄진다. 공격자들은 특별히 제작된 데이터를 입력하여 인터프리터를 속여 의도되지 않은 명령어들을 실행하도록 한다. 인젝션 취약점은 공격자에게 어플리케이션에 이용 가능한 어떤 임의의 데이터를 생성하고, 읽고, 갱신하고, 삭제하는 것을 허용한다. 최악의 경우는 이러한 취약점들이 공격자로 하여금 어플리케이션을 완전히 손상시키고 시스템을 다운시키고, 심지어 철저하게 숨겨진 방화벽 환경을 우회하는 것을 허용한다.

영향을 받는 환경

인터프리터를 사용하거나 다른 프로세스들을 호출하는 모든 웹 어플리케이션 프레임워크들은 인젝션 공격에 취약하다. 이것은 백엔드 인터프리터를 사용하는 프레임워크의 어떤 구성 요소도 포함한다는 것을 의미한다.

취약점

만일 사용자 입력 값이 검증이나 암호화 없이 인터프리터에 보내질 경우, 그 어플리케이션은 취약하다. 아래와 같은 질의어들에게 사용자의 입력 값이 제공되는지 확인해라:

```
  PHP:
$sql = "SELECT * FROM table WHERE id = '" . $_REQUEST['id’] . "’";
   JAVA:
String query = “ SELECT user-id FROM user-data WHERE user-name = ‘” +
req.getParameter(“userID) + “’ and user-pasword = ‘” req.getParameter(pwd”) + “’”;
```

보안 검증

목표는 사용자 데이터가 어플리케이션에 의해 생성되어 인터프리터로 보내진 명령어와 질의어들의 의미를 수정할 수 없도록 검증하는 것이다.

자동적인 접근방법: 수 많은 취약점 스캔 도구들은 SQL 인젝션을 포함한 인젝션 문제점들을 탐색한다. 정적 분석 도구들은 불안전한 API 해석프로그램 사용을 탐색하는데 유용하지만 검증 확인이나 암호화가 취약점을 보호하기 적절한지 여부는 검증할 수 없다. 만일 어플리케이션이 501/500 내부 서버 에러 혹은 세부적인 데이터베이스 에러에 걸린다면, 그것은 자동화 도구를 방해할 수 있지만 코드는 여전히 위험한 상태일지도 모른다. 자동화 도구들은 LDAP / XML 인젝션 / XPath 인젝션을 탐지할 수 있을 것이다.

수동적인 접근방법: 가장 효과적이고 정확한 접근방법은 인터프리터들을 호출하는 코드를 확인하는 것이다. 검토자는 안전한 API 의 사용 또는 적절한 검증 확인이나 암호화가 발생했는지 여부를 확인해야 한다. 대부분의 어플리케이션의 공격 범위는 넓기 때문에 테스트 하는데 시간이 많이 걸리고 보상이 적을 수 있다.

방어

가능하다면 인터프리터의 사용을 삼가 하라. 만일 인터프리터를 사용해야만 한다면, 인젝션을 피할 수 있는 안전한 방법은 명확하게 분류된 매개 변수화된 질의어와 객체 관계 맵핑(ORM) 라이브러리와 같은 API 를 사용하는 것이다. 이러한 인터페이스들은 모든 데이터의 벗어난 동작을 다루거나 또는 벗어나도록 요청하지 않는다. 안전한 인터페이스들이 문제를 해결하는 동안 검증은 여전히 공격을 탐지하도록 권장한다.

인터프리터를 사용하는 것은 위험하므로, 추가적으로 다음과 같은 사항들을 주의할 필요가 있다:

- **입력 검증.** 나타나거나 저장된 데이터를 받아들이기 전에 길이, 유형, 구성, 비즈니스 규칙에 대한 모든 입력 값을 검증하기 위하여 표준 입력 검증 메커니즘을 사용하라. "알고 있는 올바름을 수락"하는 검증 전략을 사용해라. 잠재적으로 악의적인 데이터를 삭제하려고 시도하는 것 보다는 검증되지 않은 입력을 거부하라. 에러 메시지에 무효한 데이터를 포함해야 한다는 것을 잊지 말라.
- 심지어 저장 프로시져를 호출할 때도 플레이스홀더 대체 표지와 함께 **명확하게 분류된 매개 변수화된 질의 API를 사용하라.**
- 데이터베이스들과 다른 백엔드 시스템들에 접속 시 **최소한 권한을 강제화하라**.
- 공격자에게 유용한 **상세 에러 메세지들을 피하라**.
- 대개 SQL인젝션으로부터 안전한 **저장 프로시져를 사용하라.** 하지만 **인젝션이 가능할 수 있음을 조심하라** (저장 프로시져 내에 exec() 또는 연결된 독립변수에 의해 가능할 수 있다.).
- `mysql_query()`나 이와 유사한 **동적 질의 인터페이스들을 사용하지 마라**.
- PHP의 addslashes()나 str_replace("'", "''")와 같은 문자 치환 함수들과 같은 **간단한 에스케이프 함수들을 사용하지 마라**. 이러한 함수들은 취약해서 공격자들에 의해 잘 이용된다. MySQL 사용할 때는 PHP용으로는 mysql_real_escape_string()를 사용하고 아니면 에스케이프를 필요로 하지 않는 PDO를 사용해라.
- **일반적인 오류를 조심해라.** 입력 값들은 검증되기에 앞서 어플리케이션의 현재 내부 표현을 반드시 해독하고 일반화 되어야 한다. 여러분의 어플리케이션이 같은 입력 값을 두 번 해독하지 않도록 보증해라. 이러한 오류들은 확인 된 이후에 위험한 입력 값을 제출함으로써 "화이트 리스트"의 방어 대책을 우회하기 위하여 사용될 수도 있다.

언어별 구체적인 추천:

- Java EE – 확실하게 분류되어 준비된 명령문 혹은 Hibernate 나 Spring 과 같은 객체 관계 맵핑(ORM)을 사용하라.
- .NET –SqlParameter 를 이용한 SqlCommand 혹은 Hibernate 와 같은 객체 관계 맵핑(ORM)처럼 확실하게 분류된 매개 변수화된 질의어들을 사용하라.
- PHP – 확실하게 분류된 매개 변수화된 질의어(bindParam() 사용)를 가진 PDO 를 사용하라.

예제

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5121
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4953
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4592

참조

- CWE: CWE-89 (SQL 인젝션), CWE-77 (명령어 인젝션), CWE-90 (LDAP 인젝션), CWE-91 (XML 인젝션), CWE-93 (CRLF 인젝션), 기타.
- WASC 위협 분류:
  http://www.webappsec.org/projects/threat/classes/ldap_injection.shtml
  http://www.webappsec.org/projects/threat/classes/sql_injection.shtml
  http://www.webappsec.org/projects/threat/classes/os_commanding.shtml
- OWASP, http://www.owasp.org/index.php/SQL_Injection
- OWASP, http://www.owasp.org/index.php/Guide_to_SQL_Injection
- OWASP, http://www.owasp.org/index.php/Reviewing_Code_for_SQL_Injection
- OWASP, http://www.owasp.org/index.php/Testing_for_SQL_Injection
- SQL 인젝션, http://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf
- 향샹된 SQL 인젝션, http://www.ngssoftware.com/papers/advanced_sql_injection.pdf
- 좀 더 향상된 SQL 인젝션, http://www.nextgenss.com/papers/more_advanced_sql_injection.pdf
- Hibernate, J2EE와 .NET을 위한 향상된 객체 관계 관리자(ORM), http://www.hibernate.org/
- J2EE Prepared Statements, http://java.sun.com/docs/books/tutorial/jdbc/basics/prepared.html
- How to: ASP.Net에서 SQL 인젝션 방어,
  http://msdn2.microsoft.com/en-us/library/ms998271.aspx
- PHP PDO 기능들, http://php.net/pdo

A3 - 악성 파일 실행

악성파일 실행 취약점은 많은 어플리케이션에서 발견된다. 개발자는 자주 공격 가능성이 있는 사용자 입력을 직접적으로 사용하거나 파일이나 스트림 함수와 연결시키거나 또는 입력 파일을 부적절하게 신뢰한다. 많은 플랫폼에서, 프레임워크는 URL 이나 파일 시스템 참조 등의 외부 객체 참조를 사용하는 것을 허용한다. 이 데이터가 충분히 점검되지 않았을 때, 웹 서버로 하여금 임의로 원격에서 악의적인 내용이 포함되고 처리되고 또는 호출되는 것을 야기할 수 있다.

이것은 공격자에게 다음과 같은 것들을 수행하도록 한다:

- 원격 코드 실행
- 원격 루트킷 설치와 전체 시스템 손상
- 윈도우 상에서는 PHP의 SMB 파일 래퍼의 사용을 통해 내부 시스템이 손상될 가능성이 있다.

이 공격은 특히 PHP 에서 널리 사용된다. 또한, 스트림이나 파일 함수를 사용하는 경우, 사용자 입력이 파일 이름에 영향을 미치지 않도록 하기 위해 매우 주의해야 한다.

영향 받는 환경

사용자로부터 파일 이름이나 파일을 받는 모든 웹 어플리케이션 프레임워크는 악의적인 파일 실행에 취약하다. 전형적인 예로 URL 파일명 함수나 로컬 파일을 포함시키기 위해 사용자에게 파일명 선택을 수락하는 코드를 허용하는 .NET 어셈블리가 있다.

PHP 는 특히 어떠한 파일이나 API 을 토대로 한 스트림과 함께 매개변수 조작을 통한 원격 파일 인클루션(RFI) **공격에 취약하다**.

취약점

일반적인 취약점 구성은 다음과 같다:

```
include $_REQUEST['filename'];
```

이것은 단순히 원격 공격 스크립트를 실행하도록 하는 것을 허용할 뿐만 아니라, (만약 php 가 윈도우즈에서 실행된다면) PHP 파일 시스템 래퍼의 SMB 지원을 토대로 로컬 파일 서버에 접근하는 데 사용될 수 있다.

다른 종류의 공격 방법은 다음과 같다:

- 세션 파일, 로그 데이터로 업로드되거나, 이미지 업로드에 의해 악의적인 데이터를 업로드(포럼 소프트웨어의 전형적인 예)
- 압축이나 오디오 스트림의 사용. 예를 들어서 zlib://나 ogg:// 같이 allow_url_fopen 이나 allow_url_include 가 기능을 억제할지라도 내부 PHP URL 플래그를 검사하지 않기 때문에 원격 자원에 접근하는 것을 허용한다.
- PHP:// 입력과 파일보다는 요청된 POST 데이터로부터 입력을 받기 위해 PHP 래퍼를 사용
- PHP 데이터 사용: data:;base64,PD9waHAgcGhwaW5mbygpOz8+ 같은 래퍼

이 기법들은 방대하고 또한 주기적으로 변하기 때문에, 서버 측의 파일명이나 접근 선택에 영향을 주는 사용자의 입력 값을 다룰 때에는 적절하게 설계된 보안 아키텍처와 견고한 설계를 사용하는 것이 꼭

필요하다.

PHP 의 예가 제시되었지만, 이 공격 방법은 PHP 와는 다른 방법을 통해 .NET 과 J2EE 에도 역시 적용 가능하다. 이러한 프레임워크에서 만들어진 어플리케이션은 사용자에 의해 제공되거나 영향을 받는 파일명이 보안 통제를 우회하지 않음을 확실히 하기 위해 코드 접근에 대한 보안 메커니즘에 좀더 주의를 기울어야 한다.

예를 들어서, 공격자가 제시한 XML 문서가 악의적인 DTD 를 가지고, XML 분석기가 원격 조정의 DTD 에 접근하도록 하고 그 결과를 분석 및 처리하도록 할 수 있다. 호주의 보안 단체는 방화벽 너머 포트 스캐닝이 가능함을 증명하였다. 더 많은 정보를 얻기 원한다면 이 장 뒷부분의 더 많은 정보를 위한 참조 중 [SIF01] 부분을 참고하라.

이 특별한 취약점으로 인해 발생하는 피해는 프레임워크 내의 샌드박스의 능력과 플랫폼 격리 제어와 직접적으로 연관이 있다. PHP 는 거의 독립되어 있지 않으며, 샌드박스의 개념이나 보안 구조를 가지고 있지 않고, 제한적으로나 부분적으로 믿을 수 있는 기타 플랫폼 보다 공격에 손상을 더 많이 받을 수 있으며, 또는 웹 어플리케이션이 안전한 관리 덕분에 올바르게 작동되고 설정이 되는 (거의 디폴트는 아니다) JVM 하에 웹 어플리케이션이 작동 될 때와 같은 알맞은 샌드박스에 포함되어 있다.

보안 검증

자동적인 접근방법: 취약점 점검 도구는 파일 포함이나 공격을 위한 문법에 사용된 매개 변수를 식별하는 데 어려움이 있다. 정적인 분석 도구는 위험한 API 의 사용에 대해 조사는 할 수 있지만 취약점에 적절한 검증이나 암호화인지를 확인할 수는 없다.

수동적인 접근방법: 코드 검토는 파일을 어플리케이션에 첨부하는 것을 허용하는 코드를 찾아낼 수 있지만, 이것을 인식하는 것에는 오류가 있을 가능성이 많다. 검사를 하는 것은 이러한 취약점들을 찾아낼 수 있지만, 특정한 매개변수와 적당한 규칙을 분류하는 것은 어려운 일일 수 있다.

방어

원격 파일 인클루드 취약점을 방지하는 것에는 구조 설계와 디자인 측면에서 처음부터 끝까지 철저한 검사와 같은 신중한 계획이 필요하다. 일반적으로 잘 만들어진 어플리케이션은 서버 기반의 자원을 위해 파일명에 사용자 입력 값을 사용하지 않을 것이며 (예: 이미지, XML, XSL 번역문서, 스크립트 첨부) 또한, 방화벽 정책을 적절하게 적용하여 새롭게 생기는 인터넷으로나 내부의 다른 서버로 향하는 외부 지향 연결을 방지한다. 그러나 오랜 시간 사용된 수많은 어플리케이션은 사용자가 공급하는 입력 값을 계속적으로 받아들일 수 밖에 없다.

가장 중요한 고려 대상으로는:

- **간접 객체 참고 맵을 사용하라**(자세한 정보를 원하면 A4 섹션(section)을 보라). 예를 들어, 부분적인 파일명이 사용될 때, 부분 참조에 대한 불필요한 데이터를 고려하라:

```
<select name=" language" >
  <option value=" English" >English</option>
대신에
```

<select name=" language" >
<option value=" 78463a384a5aa4fad5fa73e2f506ecfc" >English</option>
를 사용하라.

간접 객체 참조에 대한 무차별 대입법을 방지하기 위해서 암호의 사용을 고려하라. 대안으로, 1,2,3같은 색인 값을 사용하고 매개변수의 조작을 탐지하기 위해 배열 범위가 검증 되었는지 확인하라.

- 만약 여러분의 언어가 지원한다면, **명시적인 값 변조 확인 기법을 사용하라.** 그렇지 않으면, 변수 이름을 정하는 정책으로 변조 확인을 지원하도록 고려하라:

$hostile = &$_POST; // $_REQUEST 변수가 아니라 $_POST 변수를 참조하라.

$safe['filename']= validate_file_name($hostile['unsafe_filename']); // 안전을 검증하라.

그래서 악의적인 입력 값을 근거로 한 어떤 동작이라도 즉시 인지된다:

☒ require_once($_POST['unsafe_filename'] . 'inc.php');

☑ require_once($safe['filename'] . 'inc.php');

- "알고 있는 올바름을 수락" 하는 전략으로 **강력하게 사용자 입력 값을 검증하라.**
- 웹 서버가 외부의 웹 사이트나 내부 시스템에 새로운 연결을 맺는 것을 방지하기 위해 **방화벽 정책을 부가하라.** 매우 가치가 높은 시스템에 대해 연결된 VLAN 이나 독립망으로부터 웹 서버를 격리하라.
- 세션 객체, 아바타와 이미지, PDF 보고서와 임시 파일 내에 데이터를 변조시키는 것과 같이 다른 제어를 할 수 없도록 사용자가 제공하는 파일이나 파일명을 확인하라.
- 각각의 어플리케이션을 격리시키기 위해 가상화와 같은 **Chroot 감옥**이나 다른 샌드박스 **메커니즘의 구현을 고려하라.**
- PHP: php.ini 에서 allow_url_fopen 과 allow_url_include 를 비활성화하라. 이 기능을 포함시키지 않도록 PHP 를 구성하는 것을 고려하라. 몇몇 안 되는 어플리케이션이 이런 기능을 필요로 하며 따라서 이러한 설정은 특정 어플리케이션의 필요에 따라 활성되어야 된다.
- PHP: **register_global 를 비활성화하고, 초기화되지 않은 변수들을 찾기 위해 E_STRICT 를 사용하라.**
- PHP: 파일이나 스트림 함수(stream_*)를 신중하게 확인하라. 사용자 입력에 아래 함수를 이용한 파일명 인수가 포함되지 않도록 하라:

include() include_once() require() require_once() fopen() imagecreatefromXXX() file() file_get_contents() copy() delete() unlink() upload_tmp_dir() $_FILES move_uploaded_file()

- PHP: 자료가 system() eval() passthru()나 `(backtick 연산자)에 제공되는 경우에는 매우 주의를 기울여라.

- J2EE에서는, 보안 관리자를 활성화하고 적절하게 구성되었는지 그리고 어플리케이션이 실행 권한을 적절하게 요청하는지를 확실히 해라.
- ASP.NET 에서는 부분적으로 신뢰된 문서를 참조하라. 그리고 여러분의 어플리케이션이 신뢰 안에서 분할되도록 설계하라. 그래서 대부분의 어플리케이션이 가장 낮은 신뢰 상태로 존재하는 부분은 가능한 신뢰 상태로 있음을 알 수 있다.

예제

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0360
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5220
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4722

참조

- CWE: CWE-98 (PHP 파일 인클루션), CWE-78 (OS 명령어 인젝션), CWE-95 (Eval 인젝션), CWE-434 (제한되지 않은 파일 업로드)
- WASC 위협 분류: http://www.webappsec.org/projects/threat/classes/os_commanding.shtml
- OWASP 가이드, http://www.owasp.org/index.php/File_System#Includes_and_Remote_files
- OWASP 테스팅 가이드, http://www.owasp.org/index.php/Testing_for_Directory_Traversal
- OWASP PHP Top 5, http://www.owasp.org/index.php/PHP_Top_5#P1:_Remote_Code_Execution
- Stefan Esser, http://blog.php-security.org/archives/45-PHP-5.2.0-and-allow_url_include.html
- [SIF01] Sift 네트워크, 웹 서비스: 새로운 트릭 가르치기, http://www.ruxcon.org.au/files/2006/web_services_security.ppt
- http://www.owasp.org/index.php/OWASP_Java_Table_of_Contents#Defining_a_Java_Security_Policy
- Microsoft – 부분 신뢰를 위한 프로그래밍, http://msdn2.microsoft.com/en-us/library/ms364059(VS.80).aspx

A4 – 불안전한 직접 객체 참조

직접 객체 참조는 개발자가 파일, 디렉토리, 데이터베이스 레코드나 키를 URL 이나 폼 매개변수로 내부에 구현된 객체를 노출시킬 때 발생할 수 있다. 접근 권한 확인이 적절히 이뤄지지 않으면, 공격자는 이러한 참조를 조작함으로써 다른 객체에 인증 없이 접근할 수 있다.

예를 들어 인터넷 뱅킹 어플리케이션에서 일반적으로 계좌번호를 일차 키로 사용하기 때문에, 개발자는 웹 인터페이스 상에서 계좌번호를 바로 사용하도록 부추김을 받게 된다. 만약 개발자가 SQL 인젝션을 방지하기 위해 매개 변수화된 SQL 질의들을 사용하더라도, 사용자가 계좌의 소유자이고 계정을 볼 수 있는 권한이 있다는 것에 대한 확인을 하지 않는다면, 공격자는 계좌번호 매개 변수를 조작하여 **모든** 계좌를 보거나 변경할 수 있다.

2000 년 호주 세무서의 GST Start Up 보조 사이트가 합법적이긴 하지만 악의적인 사용자가 URL 에 나타나는 ABN (회사 세금 아이디)를 변경하는 공격을 받았다. 그 악의적인 사용자는 시스템에서 17,000 개 회사 정보를 빼가서 모든 17,000 개 회사들에게 공격 정보가 든 이메일을 발송했다. 이러한 취약점은 매우 흔한데도 불구하고 수 많은 어플리케이션에서 검사조차 하지 않고 있다.

영향 받는 환경

모든 웹 어플리케이션 프레임워크는 불안전한 직접 객체 참고 공격에 취약하다.

취약점

많은 어플리케이션은 그것의 내부 객체 참조를 사용자에게 노출시킨다. 공격자는 매개변수 변조를 통해 참조를 변경하고 강제적이지 않지만 의도된 접근 통제 정책을 우회한다. 종종 이러한 참조는 파일 시스템과 데이터베이스를 목표로 하지만 모든 노출된 어플리케이션 구조는 취약할 수 있다.

예를 들어, 코드가 특정 파일 이름이나 경로를 사용자가 입력하도록 되어 있다면, 공격자는 어플리케이션의 다른 디렉토리로 이동하여 다른 리소스에 접근할 수 있다.

```
<select name="language"><option value="fr">Français</option></select>
…
require_once ($_REQUEST['language'."lang.php");
```

그러한 코드는 웹 서버의 파일 시스템 내 어떠한 파일에도 접근할 수 있도록 "../../../../etc/passwd%00" 같은 문자열을 이용한 널 바이트 인젝션(자세한 정보는 OWASP Guide를 참고)을 사용하여 공격을 받을 수 있다.

비슷한 방식으로, 데이터베이스 키에 대한 참조도 자주 노출된다. 공격자는 이러한 매개 변수를, 추측하거나 유효한 다른 키를 찾아서 간단히 공격할 수 있다. 때때로 이런 매개 변수가 연속적인 경우가 흔히 존재한다. 아래 예를 보면, 어플리케이션이 권한 없는 카트에 대한 어떠한 연결 주소도 보여 주지 않고 있으며, SQL 인젝션이 불가능한 경우에도, 공격자는 여전히 cartID 매개 변수를 공격자가 원하는 어떤 값으로도 변경할 수 있다.

```
int cartID = Integer.parseInt( request.getParameter( "cartID" ) );
String query = "SELECT * FROM table WHERE cartID=" + cartID;
```

보안 검증

이 검증의 목적은 애플리케이션이 공격자에 의해서 직접 객체 참조가 변조되지 않는 것을 확인하는 것이다.

자동화된 접근방법: 취약점 스캐닝 도구는 매개변수가 조작 가능한 여지가 있는지, 실제로 조작이 되었는지 확인하기 어렵다. 정적인 분석 툴도 공격 시도 이전에는 어떤 매개변수가 접근 권한 확인이 필요한지 알 수 없다.

수동적인 접근방법: 대부분의 경우, 코드 검토는 중요한 매개변수들을 추적하고, 변조 가능한지 확인할 수 있다. 침투 테스트 역시 변조가 가능한지 확인할 수 있다. 그러나 이러한 기술들은 많은 시간이 소비되고 실수가 있을 수 있다.

방어

최고의 보호는 인덱스, 간접 참조 또는 검증이 쉬운 다른 간접적인 방법을 사용하여 사용자가 직접 객체 참조에 노출되는 것을 피하는 것이다. 만약 직접 객체 참조를 꼭 사용해야 한다면, 사용 이전에 사용자가 권한을 부여 받았는지 확인해야 한다.

어플리케이션 객체의 참조에 대한 표준 방법을 확립하는 것이 중요하다:

일차키 또는 파일명과 같이 가능하다면 사적 객체 참조에 사용자가 노출되지 않도록 하라.

- '알고 있는 올바름을 수락' 하는 접근을 통해 사적 객체 참조를 광범위하게 검증하라.

모든 참조된 객체에 대해 권한을 검증 하도록 하라.

매개변수 변조 공격을 방어하기 위해서는 인덱스 값이나 참조 맵을 사용하는 것이 최적의 방안이다.

```
http://www.example.com/application?file=1
```

만약 데이터베이스 구조에 대한 직접 참조를 노출해야 한다면, SQL 구문이나 다른 데이터베이스 접근 방법들이 아래와 같이 권한 부여된 레코드만 허용하도록 확인해야 한다:

```
int cartID = Integer.parseInt( request.getParameter( "cartID" ) );
User user = (User)request.getSession().getAttribute( "user" );
String query = "SELECT * FROM table WHERE cartID=" + cartID + " AND userID="
+ user.getID();
```

예제

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0329
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4369
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0229

참조

- CWE: CWE-22 (경로 유출), CWE-472 (웹 매개변수 훼손)
- WASC 위협 분류: http://www.webappsec.org/projects/threat/classes/abuse_of_functionality.shtml
  http://www.webappsec.org/projects/threat/classes/insufficient_authorization.shtml
- OWASP, http://www.owasp.org/index.php/Testing_for_business_logic
- OWASP, http://www.owasp.org/index.php/Testing_for_Directory_Traversal
- OWASP, http://www.owasp.org/index.php/Category:Access_Control_Vulnerability

A5 - 크로스 사이트 요청 변조 (CSRF)

'크로스 사이트 요청 변조(CSRF)'는 새로운 공격은 아니지만, 간단하고 매우 위협적인 공격이다. CSRF 공격은 로그인한 피해자의 브라우저가 취약한 웹 어플리케이션에 요청을 보내도록 하여 피해자 대신 선택된 작동을 수행하도록 한다.

아래 요인을 갖는 그 어떤 웹 어플리케이션도 위험해질 수 있으며, 이 취약점은 광범위하게 퍼져 있고 위험하다.

- 취약한 작동에 대한 권한 확인이 없다.
- 요청 범위 내에 기본 로그인 정보가 주어지면 요청을 처리한다 (예. http://www.example.com/admin/doSomething.ctl?username=admin&passwd=admin)
- 액티브 디렉토리와 같이 통합된 인트라넷의 일부로서 제출된 커버로스 토큰이나, 로그인하지 않았지만 "내 계정 정보 기억" 기능을 통하거나, 또는 어플리케이션에 로그인 시 제공되는 세션 쿠키 등 자동으로 제출된 자격 인증만을 기반으로 요청에 대한 권한 판단을 수행하는 어플리케이션.

불행히도 오늘날 대부분의 웹 어플리케이션은 세션 쿠키, 기본 자격 증명, 발신 IP 주소, SSL 인증, 윈도우 도메인 자격 증명과 같은 자동 제출된 자격 증명만을 의존하고 있다.

이 취약점은 '세션 라이딩', ' 원 클릭 공격',' 크로스 사이트 참조 변조', '악의적인 연결' 그리고, '자동화된 공격'이라는 다른 이름으로도 알려져 있으며, 머리글자인 'XSRF' 또한 자주 사용되고 있다. OWASP 와 MITRE 둘 다 'Cross Site Request Forgery' 와 'CSRF'라는 용어로 표준화 되었다.

영향 받는 환경

모든 웹 어플리케이션 프레임워크들은 CSRF 에 취약하다.

취약점

포럼에 대한 전형적인 CSRF 공격은 사용자가 어플리케이션의 로그아웃 페이지와 같이 특정 함수를 불러내도록 하는 유형이다. 피해자에 의해 보여지는 그 어떠한 페이지 내의 태그도 사용자를 로그아웃하는 요청을 생성할 것이다:

```
<img src="http://www.example.com/logout.php">
```

만약 온라인 은행에서 자금이체와 같은 요청을 처리하는 어플리케이션이 허용된다면, 이와 유사한 공격 또한 허용될 것이다:

```
<img src="http://www.example.com/transfer.do?frmAcct=document.form.frmAcct
&toAcct=4345754&toSWIFTid=434343&amt=3434.43">
```

Blackhat 2006에서 Jeremiah Grossman은 '외부로부터의 인트라넷 사이트 해킹' 이란 주제로 발표했고, 여기서 그는 비록 사용자가 DSL 라우터에 웹 인터페이스가 존재한다는 사실을 모를지라도 그들의 DSL 라우터를 허가 없이 변경하는 것이 가능하다는 것을 증명했다. Jeremiah는 공격을 수행하기 위해 라우터의 기본 계정을 사용했다.

사용자의 자격 증명(일반적으로 '세션 쿠키')이 브라우저에 의해 요청 내에 자동으로 포함되기 때문에 공격자에게 자격을 부여하지 않더라도 이러한 모든 공격들은 작용한다.

만약 공격을 포함하는 태그가 취약한 어플리케이션에 입력된다면, 로그인 한 사용자를 찾게 될 가능성은 증가할 것이며, 저장되거나 반사된 XSS 취약점의 위험 증가와 유사할 것이다. CSRF 의 공격을 방어할 수 있는 시스템 상에서 자동으로 입력되지 않는 자격 증명을 도용하기 위해 CSRF 공격이 XSS 결함을 이용할 수 있기 때문에 XSS 취약점을 가진 어플리케이션은 CSRF 에 영향을 받을 지라도 CSRF 공격을 하기 위해 XSS 취약점을 꼭 필요로 하진 않는다. 수많은 어플리케이션 웜이 두 가지 기술을 복합적으로 사용해 왔다.

CSRF 공격에 대한 방어를 구축할 때, XSS 취약점들이 어플리케이션에 구현된 CSRF 방어 수단을 우회할 수 있으므로, XSS 취약점을 제거하는데 초점을 맞춰야 한다.

보안 검증

본 검증의 목적은 브라우저에 의하여 자동으로 제시되지 않는 몇 종류의 권한 토큰을 생성하고 요청함으로써 어플리케이션이 CSRF 의 공격을 방어하도록 검증하는 것이다.

자동적인 접근방법: 어플리케이션 스캐닝 엔진에서 충분히 CSRF 탐지가 가능함에도 불구하고 오늘날 몇몇 자동화된 스캐너는 CSRF 취약점을 거의 탐지할 수 없다. 만약 어플리케이션 스캐너에서 크로스 사이트 스크립팅 취약점이 탐지되었지만 여러분이 CSRF 공격에 대한 방어가 되어 있지 않다면, 여러분은 사전에 준비된 CSRF 의 공격으로 위험하게 될 것이다.

수동적인 접근방법: 모의 해킹은 CSRF 방어가 제대로 구축되어 있는지 검증하는 빠른 방법이다. 메커니즘이 강력하고 적절하게 구축되어 있는지 검증하기 위해서는 코드를 확인하는 것이 가장 효과적인 방법이다.

방어

어플리케이션이 브라우저에 의해 자동으로 제시된 자격이나 토큰을 의지하지 않도록 확인하는 것이다. 유일한 해결 방법은 자동적으로 CSRF 공격에 포함되지 않는 브라우저가 "기억" 하지 못하도록 특별히 설계된 토큰을 사용하는 것이다.

다음 전략들이 모든 웹 어플리케이션 안에 포함되어야 한다:

- **여러분의 어플리케이션 내에 XSS 취약점이 없도록 확인하라**(A1 의 Cross Site Scripting 참조).
- **모든 폼과 URL 내에** 브라우저에 의해 자동으로 제출되지 않도록 **특별히 설계되고 불규칙한 토큰을 삽입하라.** 예를 들면,

```
<form action="/transfer.do" method="post">
<input type="hidden" name="8438927730" value="43847384383">
...
</form>
```

그리고, 제출된 토큰이 현재 사용자에게 적합한지 검증하라. 그와 같이 토큰들은 그 사용자를 위한 특별한 함수 또는 페이지에 유일한 값이거나, 간단하게 전체 세션에서 유일하게 하라. 토큰을 특정한 함수나 특정한 데이터에 초점을 맞추면 맞출수록 방어는 강화되겠지만 이를 구성하고 유지하기는 더욱더 복잡해질 것이다.

- **민감한 데이터나 값에 대한 트랜잭션을 위하여 재 인증이나 트랜잭션 서명을 사용하여** 요청이 진실함을 확인하라. 요청을 검증하거나 요청을 사용자에게 통보하기 위해 'e-mail'이나 '전화'와 같은 외부 메커니즘을 구축하라.
- **민감한 데이터나 또는 값의 트랜젝션 수행을 위해 GET 요청(URLs)은 사용하지 마라.** 사용자의 민감한 데이터를 처리할 때는 POST 방식만을 사용하라. 또한, URL 에 유일하게 생성된 토큰을 포함하고 있어서 CSRF 가 실행되기 거의 불가능 하도록 해야 한다.
- **POST 방식만으로 보호하는 것은 불가능하다.** CSRF 공격을 적절히 방어하기 위해서는 임의의 토큰과 더불어 대역외 인증이나 재인증을 겸비하여야 한다.
- **ASP.NET 은 set a ViewStateUserKey 를 사용하라.** 이는 위에서 언급한 '임의의 토큰'을 점검하는 유사한 형태를 제공한다.

이러한 권장사항들이 여러분의 노출에 대한 위험을 극적으로 줄이겠지만 진보한 CSRF 공격은 이러한 제약사항을 우회할 수 있다. 가장 효과적인 기술은 유일무이한 토큰을 사용하고, 어플리케이션에 존재하는 모든 XSS 취약점들을 제거하는 것이다.

예제

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0192
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5116
- 마이스페이스 웜 설명 http://namb.la/popular/tech.html
- CSRF 공격을 수행하기 위해 퀵타임을 이용한 공격 http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=9005607&intsrc=hm_list

참조

- CWE: CWE-352 (크로스 사이트 요청 변조)
- WASC 위협 분류: 직접적인 맵핑 없음, 그러나 다음 항목이 유사함: http://www.webappsec.org/projects/threat/classes/abuse_of_functionality.shtml
- OWASP CSRF, http://www.owasp.org/index.php/Cross-Site_Request_Forgery
- OWASP, https://www.owasp.org/index.php/Testing_for_CSRF
- OWASP CSRF Guard, http://www.owasp.org/index.php/CSRF_Guard
- OWASP PHP CSRF Guard, http://www.owasp.org/index.php/PHP_CSRF_Guard
- RSnake, "CSRF이란 무엇인가?", http://ha.ckers.org/blog/20061030/what-is-csrf/
- Jeremiah Grossman, "외부에서 인트라넷 사이트 해킹하기" 주제의 슬라이드와 데모들 http://www.whitehatsec.com/presentations/whitehat_bh_pres_08032006.tar.gz
- 마이크로소프트, ViewStateUserKey 상세, http://msdn2.microsoft.com/en-us/library/ms972969.aspx#securitybarriers_topic2

A6 – 정보 유출과 부적절한 에러 처리

어플리케이션은 다양한 문제들을 통해서 의도하지 않게 구성, 내부 작업들에 대한 정보를 유출하거나 개인정보를 침해 할 수 있다. 또한 어플리케이션들은 어떤 작업들을 처리하는데 얼마나 오래 걸리느냐에 따라서 또는 다른 에러 번호와 함께 동일한 에러 메세지를 표시하는 것과 같이 다른 입력 값에 대한 다른 응답에 따라서 내부 상태를 유출할 수도 있다. 웹 어플리케이션들은 상세한 에러 메시지나 디버그 관련 에러 메시지들을 통해서 내부 상태에 대한 정보를 종종 유출한다. 이러한 정보는 더욱 강력한 공격들을 준비하거나 심지어 자동화되기 위한 수단이 된다.

영향 받는 환경

모든 웹 어플리케이션 프레임워크들은 정보 유출과 부적절한 에러 처리에 취약하다.

취약점

어플리케이션들은 에러메시지들을 자주 발생하고 사용자들에게 보여준다. 수많은 이런 에러 메시지들은 취약점을 노출시킬 수 있는 유용한 정보와 구현 상세를 누설하기 때문에 공격자에게 아주 좋은 정보가 된다. 몇 가지 흔한 예들이 있다:

- 스택 추적 정보, 실패한 SQL 명령문 또는 기타 디버그 정보등과 같이 너무 많은 정보를 보여주는 에러가 포함된 세부적인 에러 처리.
- 다른 입력 값을 기반으로 다른 결과값을 생성하는 기능. 예를 들어, 로그인 함수에 같은 사용자 이름을 제공하지만 다른 패스워드를 제공한 경우, 사용자 없음과 틀린 패스워드라는 동일한 메시지를 제공해야 한다. 그러나 많은 시스템들은 다른 에러 코드를 보여 준다.

보안 검증

목적은 어플리케이션이 에러 메시지들이나 다른 수단을 통해서 정보를 유출할 수 없도록 검증하는 것이다.

자동화된 해결방법: 취약점 스캐닝 도구들은 에러 메시지가 생성되도록 할 것이다. 정적인 분석 도구들은 정보를 유출하는 API 들의 용도에 대한 조사를 할 수 있으나 그런 메시지들의 의미를 검증할 수는 없다.

수동적인 해결방법: 코드 검토는 부적절한 에러 처리와 정보를 유출할 수 있는 각각의 패턴들에 대해 조사가 가능하지만 시간이 많이 소비된다. 테스트는 에러 메시지를 생성해내지만 어떤 에러 경로가 영향을 받는지 알아내는 것은 도전적인 일이다.

방어

개발자들은 어플리케이션이 발생하는 에러들을 만들기 위해 OWASP 의 WebScarab 과 같은 도구를 사용해야 한다. 이런 방법으로 테스트 받은 적이 없는 어플리케이션들은 거의 대부분 예상하지 못한 에러를 생성한다. 또한 어플리케이션들은 공격자들에게 원하지 않은 정보가 유출되는 것을 막기 위해서는

표준 예외 처리 아키텍처를 포함하고 있어야 한다.

정보가 유출되는 것을 막기 위해서는 통제가 요구된다. 아래와 같은 실행을 통하여 효과를 보았다:

- **예외 처리를 하기 위해 일반적인 접근 방법을 전체 소프트웨어 개발팀이 공유하도록 확실히 하라.**
- **자세한 오류 처리를 제한하거나 기능을 비활성화하라.** 특히, 최종 사용자에게 스택 추적 정보, 경로 정보와 디버그 정보를 보여 주지 말라.
- **대략 동일 시간대에 유사하거나 동일한 에러 메시지의 다양한 결과를 응답하도록 보안 경로를 확실히 하라.** 이것이 불가능하다면, 공격자로부터 자세한 정보를 숨기기 위해서 모든 트랜잭션들에 대한 불규칙한 대기 시간을 부여하는 것을 고려하라.
- 다양한 계층들은 기초적인 웹 서버(IIS, Apache 기타 등등), 데이터베이스 계층과 같이 심각하거나 예외적인 결과들을 반환할 수 있다. **에러 메시지가 외부 침입자들에게 악용되는 것을 예방하기 위하여 모든 계층들의 에러가 적절하게 점검되고 설정되는 것**은 매우 중요하다
- 일반적인 프레임워크들은 에러가 여러분의 특별히 개발된 코드 내에 있거나 프레임워크 코드 내에 있는지에 따라서 다른 HTTP 에러 코드들을 반환한다는 것을 명심해야 한다. 모든 에러 경로들에서 발생되는 에러 메시지를 대부분의 사용자에게 적절하게 제거되어 반환하는 기본 에러 처리기를 만드는 것은 상당한 가치가 있다.
- 기본 에러 처리기를 우선시 하여 항상 "200" (ok) 에러 화면을 반환하게 함으로써 자동화된 스캐닝 도구가 심각한 에러의 발생 여부를 결정하는 능력을 감소시킨다. 이것은 " 애매함을 통한 보안"으로 특별한 계층의 방어를 제공할 수 있다.
- 몇몇 큰 조직들은 모든 어플리케이션들 중에서 무작위 방식이나 유일한 에러 코드들을 포함시키는 것을 선택했다. 이것은 특정한 에러에 대한 정확한 해결방안을 찾음으로써 안내 데스크를 도울 수는 있지만 그것 또한 공격자가 어떤 경로에서 어플리케이션이 실패하는지 정확하게 결정하도록 할 수 있다.

예제

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4899
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-3389
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0580

참조

- CWE: CWE-200 (정보 유출), CWE-203 (불일치 정보 유출), CWE-215 (디버그 정보를 통한 정보 유출), CWE-209 (에러 메시지 정보 유출), 기타.
- WASC 위협 분류: http://www.webappsec.org/projects/threat/classes/information_leakage.shtml
- OWASP, http://www.owasp.org/index.php/Error_Handling
- OWASP, http://www.owasp.org/index.php/Category:Sensitive_Data_Protection_Vulnerability

A7 - 취약한 인증 및 세션 관리

올바른 인증 및 세션 관리는 웹 어플리케이션 보안에 필수적이다. 이러한 영역의 취약점은 자격 증명과 세션 토큰의 생명 주기동안 보호에 실패하는 것과 연관된다. 이러한 취약점은 사용자나 관리자 계정을 가로챌 수 있고, 권한 및 책임 추구성 관리를 약화시키며, 개인정보 침해를 유발한다.

영향 받는 환경

모든 웹 어플리케이션 프레임워크는 인증 세션 관리 결함에 취약하다.

취약점

주요 인증 메커니즘의 취약점은 흔하지 않지만, 로그아웃, 패스워드 관리, 타임아웃, 기억하기, 비밀 질문과 계정 갱신과 같은 부수적인 인증 기능을 통한 취약점은 꽤 자주 발표되고 있다.

보안 검증

이 목적은 어플리케이션이 적절하게 사용자를 인증을 하고 신원과 그와 관련된 자격 증명을 적절하게 보호하는지 검증하는 것이다.

자동화된 해결방법: 취약점 스캔 도구로 맞춤 인증과 세션 관리 설계를 탐지하는 것은 어렵다. 정적인 분석 도구 또한, 맞춤 코드에서 인증 및 세션 관리 문제를 탐지하는 것은 어렵다.

수동적인 해결방법: 코드 점검 및 테스트를 복합적으로 사용하는 것을 인증 및 세션 관리와 부수적인 기능들이 모두 적절하게 구현되어 있는지를 검증하는 것이 효과적이다.

방어

인증은 안전한 통신과 자격 증명 저장에 달려 있다. 우선적으로 SSL 은 어플리케이션의 모든 인증된 부분을 위한 유일한 옵션(A9 - 불안전한 통신을 보라)이며 모든 자격 인증이 해시나 암호화 형태(A8 - 불안전한 암호화 저장을 보라)로 저장되도록 확실히 하라.

인증 취약점을 보호하는 것은 주의 깊은 계획을 필요로 한다. 가장 중요한 고려사항은 다음과 같다:

- **본래의 세션 관리 메커니즘만 사용한다.** 어떤 상황 하에서도 이차적인 세션 처리기를 사용하거나 개발해서는 안 된다.
- **URL 또는 요청에서 새롭거나 사전 설정되거나 또는 검증되지 않은 세션 식별자를 허용하면 안 된다.** 이것은 세션 고정 공격으로 불린다.
- **"내 계정 정보 기억" 기능 또는 자체 제작된 싱글 사인 온 기능과 같은 인증 및 세션 관리를 위한 여러분의 특별히 개발된 쿠키를 제한하거나 제거한다.** 이는 완전히 입증된 SSO 나 연합 인증 솔루션에 적용되지 않는다.
- **적절한 강도와 요소의 수와 함께 싱글 인증 메커니즘을 사용하라.** 이 메커니즘이 속임수이거나 재입력 공격을 받지 않도록 조심하라. 이 메커니즘을 지나치게 복잡하게 만들어서 자기 스스로 공격받게 되도록 만들지 말라.

- **암호화되지 않은 페이지로부터 로그인 과정이 시작되도록 허용하지 말라.** 자격 증명이나 세션 도난, 피싱 공격 그리고 세션 고정 공격을 방어하기 위해 암호화된 두 번째 페이지에서 항상 로그인을 시작하라.
- **성공적인 인증 또는 권한 수준 변경에 대해 새로운 세션을 재생성시키는 것을 고려하라.**
- **모든 페이지에 로그아웃 연결주소를 두어라.** 로그아웃 함으로써 모든 서버측 세션 상태와 클라이언트 측 쿠키가 제거되어야 한다. 인간적인 요소를 고려하라: 사용자는 성공적으로 로그아웃 하기 보다는, 단지 탭 또는 윈도우를 닫음으로써 끝낼 것이다.
- 보호할 데이터의 가치 대비 비활성화된 세션이 자동 로그아웃 되도록 **타임아웃 기간을 설정하라.** (짧을수록 좋다.)
- **강력한 부수적인 인증 기능(질문과 답변, 암호 재설정)을 사용하라.** 이러한 기능은 사용자명과 암호 또는 토큰이 자격 증명이 되듯 자격 증명이 된다. 내용이 노출된 공격을 막기 위해 답변은 단방향 해쉬를 적용하라.
- **URL 이나 로그에서 세션 식별자나 유효한 자격 증명의 어떤 부분이 드러나지 않도록 하라**(세션을 재작성하거나 로그 파일에서 사용자의 암호를 저장하지 마라). 사용자가 새 암호로 바꿀 때는 이전 암호를 확인하라.
- IP 주소 또는 주소 범위 마스크, DNS 조회 또는 역 DNS 조회, 참조 헤더 또는 이와 유사한 조작 가능한 자격 증명을 유일한 인증 형태로 의존하지 마라.
- **비밀번호 재설정하는 방법으로 등록된 이메일 주소(참고자료의 RSNAKE01 을 참조하라)로 비밀정보를 보내는 것에 주의하라.** 접속을 재설정하기 위해 제한된 시간동안 사용 가능한 임의 번호들만 사용하고 비밀번호가 재설정되자마자 추적 이메일을 발송하라. 자동 등록한 사용자들이 자신의 이메일 주소를 변경하는 것에 주의하고 변경을 규정하기 전에 이전 이메일 주소로 메시지를 보내라.

예제

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6145
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6229
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6528

참조

- CWE: CWE-287 (인증 문제), CWE-522 (불충분하게 보호된 자격 인증), CWE-311 (인증 프로토콜의 재입력 공격), 기타.
- WASC 위협 분류:
  http://www.webappsec.org/projects/threat/classes/insufficient_authentication.shtml
  http://www.webappsec.org/projects/threat/classes/credential_session_prediction.shtml
  http://www.webappsec.org/projects/threat/classes/session_fixation.shtml
- OWASP, http://www.owasp.org/index.php/Guide_to_Authentication
- OWASP, http://www.owasp.org/index.php/Reviewing_Code_for_Authentication
- OWASP, http://www.owasp.org/index.php/Testing_for_authentication
- RSnake01 - http://ha.ckers.org/blog/20070122/ip-trust-relationships-xss-and-you

A8 – 불안정한 암호화 저장

민감한 데이터를 암호화를 통해 보호하는 것은 대부분의 웹 어플리케이션의 중요한 부분이 되었다. 민감한 데이터의 암호화에 쉽게 실패하는 건 매우 일반적이다. 암호화를 수행하는 어플리케이션은 종종 불완전하게 설계된 암호화 알고리즘을 사용하거나, 부적절한 암호키나 강력한 암호키를 사용하여 심각한 오류를 만든다. 이러한 결함들이 컴플라이언스 위반이나 민감한 데이터 유출을 일으킬 수 있다.

영향 받는 환경

모든 웹 어플리케이션 프레임워크는 불안전한 암호화 저장에 취약하다.

취약점

암호화 결함을 방지하기 위해서는 주의깊은 계획이 필요하다. 일반적인 문제점들은 다음과 같다:

- 민감한 데이터를 암호화하지 않음
- 자체 제작 알고리즘 사용
- 강력한 알고리즘의 불안전한 사용
- 취약한 알고리즘의 지속적인 사용 (MD5, SHA-1, RC3, RC4, 등…)
- 하드 코딩된 키와 안전하지 못한 영역에 키 저장

보안 검증

어플리케이션이 저장소 내에 민감한 정보들을 적절하게 암호화 하는지 검증하는 것이다.

자동화된 해결방법: 취약점 스캐닝 도구는 전혀 암호화 저장소를 검증할 수 없다. 코드 스캐닝도구는 이미 알려진 형태의 암호화 API 들의 사용을 인식하나, 적절하게 사용되는지 또는 암호화가 외부 모듈에서 실행되는지 여부를 탐지할 수 없다.

수동적인 해결방법: 스캐닝 방법처럼 테스트로 암호화 저장소를 검증할 수는 없다. 어플리케이션이 민감한 데이터를 적절하게 구현된 메커니즘과 키 관리를 통해 암호화 하는지를 검증하는 데에는 코드 검증이 가장 좋은 방법이다. 이러한 방법은 가끔은 외부 시스템의 구성 검사를 필요로 할 수 있다.

대책

가장 중요한 것은, 암호화 되어야 하는 것들은 반드시 암호화가 되어 있어야 한다는 것이다. 그러려면 반드시 암호화 기술이 적절히 구현되어야 한다. 암호화 기술을 부적절하게 사용하는 경우가 매우 많기 때문에, 다음의 권고사항들을 여러분의 테스트 체제의 일부분으로 받아들여 암호화 자료 처리를 안전하게 진행해야 한다:

- 암호화 알고리즘 개발 **금지하라**. AES, RSA 공개키 암호화, 그리고 SHA-256 등의 검증된 공개 알고리즘만을 이용하라.
- MD5 / SHA1 등과 같이 취약한 알고리즘을 사용하지 마라. SHA-256 혹은 더 좋은 안전한 알고리즘을 추천한다.

- **오프라인 상에서 키를 생성하고 극도의 주의와 함께 개인 키를 보관하라.** 절대로 안전하지 못한 채널을 통해 개인 키를 결코 전송하지 마라.
- 데이터베이스의 자격과 같은 기반 구조 자격 증명 또는 MQ 큐 접근 내역 등은 적절하게 보호하도록 보증하라(엄격한 파일 시스템 권한과 통제를 통해). 또는 안전하게 암호화되어 로컬 또는 원격 사용자를 통해 쉽게 해독되지 않토록 보증하라.
- 디스크에 저장된 암호화된 자료가 쉽게 해독되지 않토록 보증하라. 예를 들어, 데이터베이스 연결 풀(Pool)이 비 암호화 접속을 제공한다면, 데이터베이스 암호화는 가치가 없어진다.
- PCI 데이터 보안 표준 요구사항 버전 3 에 의하면, 신용카드 사용자의 정보를 보호해야만 한다. 2008 년까지 상인들이나 신용카드로 거래를 하는 모든 사람들은 PCI DSS 사용을 의무화해야 한다. 좋은 방법은 마그네틱 선 정보와 주요한 계정 번호(PAN, 신용카드 번호로도 잘 알려져 있다)와 같은 불필요한 데이터를 저장하지 않는 것이다. 만약 PAN 을 저장한다면, DSS 컴플라이언스 요구사항들은 매우 분명하다. 예를 들어, 여러분은 어떠한 이유에서도 CVV 번호(신용카드 뒷면 번호의 뒤 세자리)를 저장할 수 없다. 좀 더 많은 정보들이 필요하다면 PCI DSS 가이드라인과 통제 구현을 참조하기 바란다.

예제

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6145
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1664
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1101 (대부분의 Java EE 서블릿 컨테이너들 역시 적용된다.)

참조

- CWE: CWE-311 (데이터 암호화 실패), CWE-326 (취약한 암호), CWE-321 (하드 코딩된 암호화 키의 사용), CWE-325 (요청된 암호화 단계 누락), 기타.
- WASC 위협 분류: 정확히 매핑되는 항목 없음.
- OWASP, http://www.owasp.org/index.php/Cryptography
- OWASP, http://www.owasp.org/index.php/Guide_to_Cryptography
- OWASP, http://www.owasp.org/index.php/Insecure_Storage
- OWASP, http://www.owasp.org/index.php/How_to_protect_sensitive_data_in_URL's
- PCI 데이터 보안 표준 v1.1, https://www.pcisecuritystandards.org/pdfs/k_pci_dss_v_1-1.pdf
- Bruce Schneier, http://www.schneier.com/
- 차세대 CryptoAPI, http://msdn2.microsoft.com/en-us/library/aa376210.aspx

A9 - 불안전한 통신

어플리케이션은 민감한 통신의 보호가 필요할 때 종종 네트워크 트래픽을 암호화하는데 실패하곤 한다. 암호화(보통은 SSL)는 모든 인증된 연결에 사용해야 하고, 특히 인터넷에 접근 가능한 웹 페이지 뿐만 아니라 백엔드 연결에 있어서도 마찬가지다. 그렇지 않으면 어플리케이션은 인증이나 세션 토큰을 노출될 수 있다. 또한 암호화는 신용카드나 건강 정보들과 같은 민감한 데이터를 전송할 때에는 반드시 사용해야 한다. 암호화를 사용하지 않거나, 암호화 모드를 해제할 수 있는 어플리케이션은 해커에 의해 악용될 수 있다

PCI 표준은 모든 신용카드 정보가 인터넷 상으로 전송될 때 암호화할 것을 요구한다.

영향 받는 환경

모든 웹 어플리케이션 프레임워크는 안전하지 않은 통신에 취약하다.

취약점

민감한 통신을 암호화 하는데 실패한다는 것은 네트워크로부터 트래픽 스니핑을 할 수 있는 공격자가 자격 증명이나 전달된 민감한 정보를 포함한 모든 대화에 접근할 수 있다는 것을 의미한다. 다른 네트워크들은 스니핑에 적게 또는 많게 영향을 받는다는 것을 고려하라. 따라서 결과적으로 호스트는 대부분 모든 네트워크에서 위태롭게 될 수 있고, 공격자들은 다른 시스템의 자격 증명을 수집하기 위해 스니퍼를 빠르게 설치할 수 있다는 것을 깨닫는 것이 중요하다.

최종 사용자가 통신을 위해 SSL 을 사용하는 것은 어플리케이션에 접근할 때 안전하지 않은 네트워크를 흔히 사용하기 때문에 매우 중요하다. 매번 단일 요청을 할 때마다 HTTP 가 인증 자격 증명이나 세션 토큰을 포함하기 때문에 모든 인증된 트래픽은 실제적인 로그인 요청 뿐만 아니라 SSL 상에서 운영되어야 한다.

백엔드 서버의 암호 통신도 역시 중요하다. 이 네트워크가 비록 더 안전해 보일지라도 이들이 전송하는 정보와 자격 증명은 더 민감하고 광범위하다. 그러므로 백엔드 상에서 SSL 을 사용하는 것은 매우 중요하다.

많은 조직에서 신용카드 정보, 주민등록번호와 같은 민감한 데이터를 암호화하는 것은 개인정보보호와 금전적인 규제 사항이 되었다. 그러한 데이터를 다루는 통신에 SSL 의 사용을 하지 않는다는 것은 규제 위반에 직면하게 될 것이다.

보안 검증

이 목적은 어플리케이션이 모두 인증되고 민감한 통신이 적절하게 암호화 되는지를 검증하는 것이다.

자동화된 해결 방법: 취약점 점검 툴은 SSL 이 앞단에서 사용되는 것을 검증할 수 있고, SSL 과 연관된 많은 결점을 찾아 낼 수 있다. 그러나 이 툴은 백엔드 연결에 접근하지 못하고, 그것이 안전한지 검증 할 수 없다. 정적인 분석 도구들은 백엔드 시스템에 특정 요청을 분석하는 것을 도울 수는 있지만, 모든 유형의 시스템들에 요청되는 일반적인 논리는 이해할 수 없을 것이다.

수동적인 해결 방법: 테스트는 SSL 이 사용되는 것과 앞단에 있는 많은 SSL 과 관련된 결점을 찾을 수

있지만, 자동화된 접근이 더 효율적이다. 코드 점검은 모든 백엔드 연결에 SSL 이 적절하게 사용되는지를 검증하는데 매우 효율적이다.

보호

가장 중요한 보호 방법은 모든 인증된 연결이나 민감한 데이터가 전송되는 곳에 SSL 을 사용하는 것이다. 웹 어플리케이션을 위한 SSL 구성이 적합한지와 관련있는 수많은 항목들이 있다. 따라서 여러분의 환경을 이해하고 분석하는 것은 중요하다. 예를 들어, IE 7.0 은 높은 신뢰도를 갖는 SSL 인증서를 위한 '녹색 막대'를 제공한다. 그러나 이것은 암호화가 안전하게 사용되었음을 증명하기 위한 적절한 통제는 아니다.

- 자격 증명이나, 신용카드 상세정보, 건강보험 그리고 다른 사적인 정보와 같이 민감하거나 가치있는 데이터를 인증하거나 전송하는 모든 연결을 위해 SSL 를 사용하라.
- 자격 증명과 고유의 데이터 값을 위한 전송 계층 보안 또는 프로토콜 수준 암호화를 사용함에 의해 웹 서버나 데이터베이스 시스템과 같은 하부 구조 사이의 통신이 적절하게 보호되도록 하라.
- PCI Data 안전 표준 요구사항 4 에서 여러분은 카드 소유자의 정보를 전송시킬 때도 반드시 보호해야만 한다. 2008 년까지 상인들이나 신용카드로 거래를 하는 모든 사람들은 PCI DSS 규제 사용을 의무적으로 지켜야 한다. 일반적으로 클라이언트, 파트너, 스텝 그리고 관리자를 시스템에 온라인으로 접근하기 위해 SSL 이나 그와 유사한 것을 사용하여 암호화해야만 한다. 더불어 더 많은 정보가 필요하다면 PCI DSS 가이드라인과 통제 구현을 참조하라.

예제

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6430
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-4704
- http://www.schneier.com/blog/archives/2005/10/scandinavian_at_1.html

참조

- CWE: CWE-311 (데이터 암호화 실패), CWE-326 (취약한 암호), CWE-321 (하드 코딩된 암호화 키의 사용), CWE-325 (요구된 암호화 단계 누락), 기타.
- WASC 위협 분류: 정확히 매핑하는 항목 없음.
- OWASP *테스팅 가이드*, SSL / TLS을 위한 테스팅, https://www.owasp.org/index.php/Testing_for_SSL-TLS
- OWASP 가이드, http://www.owasp.org/index.php/Guide_to_Cryptography
- Foundstone - SSL Digger, http://www.foundstone.com/index.htm?subnav=services/navigation.htm&subcontent=/services/overview_s3i_des.htm
- NIST, SP 800-52 전송 계층 보안(TLS) 구현의 선택과 사용을 위한 가이드라인, http://csrc.nist.gov/publications/nistpubs/800-52/SP800-52.pdf
- NIST SP 800-95 웹 서비스를 보호하기 위한 가이드, http://csrc.nist.gov/publications/drafts.html#sp800-95

A10 - URL 접근통제 실패

때로는 URL 을 보호하는 유일한 방법은 그 페이지의 연결 주소를 권한없는 사용자들에게 보여주지 않는 것이다. 그러나 동기를 갖고 숙련되거나 혹은 단지 운이 좋은 공격자는 이러한 페이지를 찾아 접근하여 기능을 이용하거나 데이터를 볼 수도 있다. 애매모호함에 의한 보안은 어플리케이션 내에서 민감한 기능과 데이터를 보호하기에는 충분하지 않다. 민감한 기능들에게 요청을 승인하기 전에 접근 제어를 반드시 수행하여 사용자가 그 기능에 접근하기 위해 접근 통제를 받도록 보증해야 한다.

영향 받는 환경

모든 웹 애플리케이션의 프레임웍크는 URL 접근통제에 실패할 때 취약하다.

취약점

이 취약점을 이용한 주요 공격 방법은 "강제적인 브라우징"이라고 부르며, 이는 보호되지 않는 페이지들을 찾아내기 위하여 추측되는 링크와 임의 대입 기법을 포함한다. 애플리케이션은 주로 접근 통제 코드가 발전되고 확산시키는 것을 허용하고 코드 베이스 전체에 걸쳐 유포시킴으로써, 그 결과 개발자와 보안 전문가 모두에게 이해하기 어려운 복잡한 모델이 된다. 이러한 복잡성은 에러를 유발하고 페이지들을 유출시키고 누락되게 한다.

이러한 결함의 일반적인 예로는 다음과 같다:

- “숨겨져 있는” 또는 “특별한” URL 은 표현 계층에서 관리자들이나 특별한 사용자들에게만 보여지지만 /admin/adduser.php 나 /approveTransfer.do 와 같이 페이지가 존재한다는 사실을 알고 있는 모든 사용자들 또한 접근할 수 있다. 이는 특히 메뉴 코드에 널리 퍼져 있다.
- 애플리케이션은 종종 정적인 XML 또는 시스템이 생성한 리포트와 같은 “숨겨진” 파일 접근을 허용하며 이들을 숨기기 위해 모호함을 통해 보안을 확신한다.
- 접근 통제 정책을 강제화하지만, 유효기간이 지나거나 불충분한 코드. 예를 들면, /approveTransfer.do 가 한순간 모든 사용자들에게 허용되었다고 상상해 보자. 그러나, SOX 통제가 적용된 이후 승인된 사용자에게만 허용되었다고 가정하자. 이후 잘못된 수정은 승인되지 않는 사용자들에게는 보여줄 뿐만 아니라, 해당 페이지를 요청할 때는 사실상 접근 통제가 수행되지도 않는다.
- 서버 상보다 오히려 브라우져 상에서 JavaScript를 통해 1700달러의 값어치를 가진 “Platinum”을 승인한 ‘MacWorld2007 에서의 공격’ 에서와 같이 서버가 아닌 클라이언트에서 특권을 평가하는 코드.

보안 검증

이 목표는 어플리케이션 내의 모든 URL 을 위한 표현 계층과 비즈니스 조직 내에 접근 통제가 지속적으로 강제화하는지를 검증하는 것이다.

자동적인 접근방법: 취약점 스캐너나 정적 분석 도구는 모두 URL 접근통제를 검증하기에는 어려움이 있지만, 그 이유는 다르다. 취약점 스캐너의 경우 숨겨놓은 페이지를 추측하거나 어떤 페이지가 각각의 사용자에게 허용되어야 하는지 판단하는데 어려움이 있다. 반면 정적인 분석 엔진은 코드 내에 자체

제작된 접근 통제를 식별하거나 비즈니스 로직과 더불어 표현 계층을 연결하는 데 어려움이 있다.

수동적인 접근방법: 가장 효율적이고 정확한 접근방법은 접근 통제 메카니즘을 검증하기 위해 코드 검토와 보안 테스트를 병행하는 것이다. 만일 이 메커니즘이 중앙 집중화되어 있다면, 검증은 매우 효율적일 수 있다. 그렇지 않고 메카니즘이 전체 코드베이스 내에 분산되어 있다면, 검증절차는 보다 많은 시간이 걸릴 수 있다. 만약 이 메커니즘이 외부에서 수행된다면 그 구성을 반드시 검사하고 테스트해야만 한다.

보호

어플리케이션의 역할과 기능을 매핑 하기 위한 매트릭스를 생성함으로써 접근 권한을 설계하기 위한 시간을 들이는 것은 제어되지 않는 URL 접근으로부터 보호하기 위한 가장 중요한 단계다. 웹 애플리케이션은 모든 URL 과 비즈니스 기능에 접근 통제를 강화해야만 한다. 표현 계층에서만 접근 통제를 하여 비즈니스 로직을 무방비 상태로 유지해서는 않된다. 사용자가 승인이 되었는지 보증하기 위해 프로세스 동안 딱 한번만 확인하고 다음 단계에서 다시 확인을 하지 않는 것 또한 불충분하다. 그렇지 않으면 공격자는 승인이 확인된 단계를 간단히 우회할 수도 있고 다음 단계로 계속 가기 위해 필요한 매개변수 값을 위조할 수도 있다.

URL 접근통제 구현을 위해서는 신중하게 계획을 세워야 한다. 주요 고려사항은 다음과 같다:

- 접근통제 매트릭스는 비즈니스, 아키텍트 그리고 애플리케이션 디자인의 일부가 됨을 보증하라.
- 모든 URL 과 비즈니스 기능은 그 어떠한 처리과정이 일어나기 이전에 사용자의 역할과 권한을 검증하는 효과적인 접근 통제 메커니즘에 의해 보호됨을 보증하라. 이와 같은 것이 다양한 단계의 처리의 시작 단계에 한번만 실행되도록 하지 말고 모든 단계에서 실행이 되도록 확실히 하라.
- 배치나 코드 전달에 앞서 침투 테스트를 수행하라. 어플리케이션이 동기가 부여된 숙련된 공격자에 의해 오용되지 않도록 하라.
- include / library 파일을 주의를 기울여라. 특히 .php 와 같이 실행 가능한 확장자를 가진 경우는 더더욱 주의하라. 가능하다면 이들은 웹 루트의 외부에 두어야만 한다. 예를 들어 라이브러리의 호출에 의해서만 생성될 수 있는 일정 상수의 확인을 통해 이들이 직접적으로 접근되지 않을 것이란 것을 검증해야 한다.
- 사용자들이 특별하거나 숨겨진 URL이나 API을 모를 것이라고 **간과하지 마라.** 관리적이고, 높은 특권을 가진 행위는 보호됨을 항상 보증하라.
- 여러분의 어플리케이션이 절대 제공하지 않는 모든 파일 형태에 대한 **접근을 차단하라.** 개념적으로 이 필터는 “알고 있는 올바름을 수락” 하는 접근을 따르며, html, pdf, php 등 여러분이 서비스하려고 의도한 파일 형태만을 허용한다. 이렇게 하면 여러분이 직접적으로 제공하고자 하지 않았던 로그 파일이나 xml 파일 등의 접근 시도는 차단될 것이다.
- 사용자가 제공한 데이터를 처리하는 XML 프로세서, 워드 프로세서, 이미지 프로세서 등과 같은 모듈에 **바이러스 백신과 패치를 최신으로 유지하라.**

예제

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0147
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0131
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-1227

참조

- CWE: CWE-325 (직접 요청), CWE-288 (대체 경로를 통한 인증 우회), CWE-285 (누락하거나 불충분한 접근 통제)
- WASC 위협 분류: http://www.webappsec.org/projects/threat/classes/predictable_resource_location.shtml
- OWASP, http://www.owasp.org/index.php/Forced_browsing
- OWASP, http://www.owasp.org/index.php/Guide_to_Authorization

지향해야 할 방향

OWASP TOP 10 은 여러분의 웹 어플리케이션의 안전을 위한 여정의 첫 단계이다.

전 세계의 60 억 인구는 두 그룹으로 나누어 질 수 있다. 첫번째 그룹은 모든 좋은 소프트웨어 회사가 버그를 알고서고 제품을 판매한다는 것을 아는 사람들이며 두번째 그룹은 이를 모르는 사람들이다. 이 첫번째 그룹의 사람들은 초기의 낙천주의가 현실에 의해 망쳐지기 이전의 삶이 어땠는지 잊으려 하는 경향이 있다. 때로 우리는 소프트웨어 회사들이 모든 마지막 버그가 수정되기도 전에 제품을 판매한다는 것을 알고 충격을 받는 두번째 그룹의 사람을 마주한다.

Eric Sink, Guardian 2006 년 5 월 25 일

대부분의 사용자들과 고객들은 두번째 그룹에 속한다. 이 문제를 어떻게 다루는지는 일반적으로 여러분의 코드와 웹 어플리케이션의 보안 상태를 개선할 수 있는 기회이다. 매년 수십억 달러가 손실되고 있으며 수많은 사람들이 이 문서에서 논의된 취약점들로 인한 신용정보 도용과 사기에 시달리고 있다.

설계자와 디자이너들에게

여러분의 어플리케이션을 적절하게 보호하기 위해 여러분은 여러분이 지켜야 할 것을 알아야 하며, 불안전할 경우 겪을 수 있는 위협과 위험을 알아야 하고 이를 조직화 된 방법으로 설명해야 한다. 중요한 어플리케이션을 디자인하는 것은 훌륭한 보안이 요구된다.

- 위협 위험 모델링과 자산 분류를 토대로 “충분한 보안”이 적용됨을 보증하라. 그러나 컴플라이언스 법칙(SOX, HIPAA, Basel 등)이 부담을 증가시키는 것처럼, 베스트프랙티스가 잘 알려져 있고 미미함 보다 더 힘들더라도 오늘날 그 미미함을 충족하는 것보다 시간과 자원을 더 투자하는 것이 절절하다.
- 비니지스 요구사항에 대하여 질문하라. 특히 누락된 비 기능 요구사항들을 물어 보라
- 위협 모델링 사용을 통해 심도 있는 방어와 더 단순한 구조를 포함한 더 안전한 디자인을 장려하라. (참고서적의 [HOW1]을 참조하라).
- 기밀성, 무결성, 가용성 그리고 부인 방지를 고려하였는지 보증하라.
- 여러분의 디자인이 COBIT 나 PCI DSS 1.1 과 같은 보안 정책과 표준에 모순되지 않는지 보증하라.

개발자들에게

많은 개발자들은 이미 웹 어플리케이션의 보안 기본 사항을 잘 다룬다. 웹 어플리케이션 보안 영역을 효과적으로 통제하기 위해서는 연습이 필요하다. 어떤 사람이 깰 수도 있지만(예를 들어 침투 테스트 실행), 안전한 소프트웨어를 구축하기 위해서는 숙련되어야 한다. 숙련자가 되는 것을 목표로 삼는다.

- OWASP에 가입을 고려하고 지역 지부 회의에 참가하여라.
- 훈련 예산이 있을시 보안 코드 훈련을 요구하라. 만약 예산이 없을 경우 훈련 예산을 요구하라.
- 주안점들을 안전하게 디자인 하라. 심도 있게 방어와 디자인 상의 단순함을 고려하라
- 더 안전한 코드 구조를 격려하는 코드 표준을 채택하라
- 여러분이 선택한 플랫폼에 더 안전한 구조를 사용하기 위해 기존의 코드를 재분해하라
- OWASP 가이드를 검토하고 선택된 통제를 여러분의 코드에 적용하기 시작하라. 다른 보안 가이드와는 달리 파괴되지 않고 여러분이 안전한 소프트웨어를 구축할 수 있게 디자인되어 있다

- 보안 결함된 부분을 위해 여러분의 코드를 테스트하라. 그리고 여러분의 테스트 단위와 웹 테스트 방식의 일부분으로 만들어라.
- 참고 서적을 검토하고 어떤 것들이 여러분의 환경에 적용 가능한지 확인하라.

오픈 소스 프로젝트를 위해

소스를 개방하는 것은 웹 어플리케이션 보안에 있어서 특이한 도전 중 하나이다. 한 개발자의 프로젝트로부터 Apache, Tomcat 그리고 PostNuke 와 같은 대규모의 웹 어플리케이션의 중요한 프로젝트까지 정말로 수백만 개의 오픈 소스 프로젝트가 있다.

- OWASP에 가입을 고려하고, 지역 지부 회의에 참가하여라.
- 여러분의 프로젝트에 4명 이상의 개발자가 참여했다면 최소한 한 명의 개발자는 보안 담당이 되어야 한다.
- 주안점들을 안전하게 디자인 하라. 심도 있게 방어와 디자인 상의 단순함을 고려하라.
- 더 안전한 코드 구조를 격려하는 코드 표준을 채택하라.
- 보안 결함된 부분이 적절하게 처리되고 있는지 확인하기 위해 신뢰할 수 있는 공개된 정책을 채택하라.
- 참고 서적을 검토하고 어떤 것들이 여러분의 환경에 적용 가능한지 확인하라.

어플리케이션 소유자들에게

상업에 종사하는 어플리케이션 소유자는 종종 시간과 자원이 제한된다. 어플리케이션 소유자들을 아래와 같이 해야 한다:

- OWASP 보안 소프트웨어 계약 부록을 통해 소프트웨어 생산자들과 작업하라.
- 비즈니스 요구사항들이 보안 요구사항과 같이 비기능성 요청(NFR)을 포함하는지 확인하라.
- 기본적인 특성으로 보안을 포함하는 디자인을 지향하고, 심도 있게 방어와 디자인 상의 단순함을 고려하라.
- 강력한 보안 지식을 가진 개발자들을 고용하거나 훈련시켜라
- 프로젝트의 처음부터 끝까지 보안 결점에 대한 테스트를 실행하라: 디자인, 설계, 테스트 그리고 배치
- 보안 사항들을 수정하기 위해서라면 프로젝트의 계획에서 자원, 예산 그리고 시간의 투자를 허용하라.

최고 경영자에게

여러분의 조직은 조직에 알맞은 보안 개발 라이프 사이클(SDLC)을 반드시 가지고 있어야 한다. 취약점들은 여러분이 제품을 판매한 후에 수정하는 것보다 개발 단계에서 수정하는 것이 훨씬 비용이 적게 든다. 합리적인 SDLC 는 To10 의 테스트를 포함할 뿐만 아니라 아래의 것들도 포함한다:

- 출하 대기 중인 소프트웨어를 위해서 보안 요구사항을 포함한 정책과 계약을 구매하였음을 보증하라.
- 자체 개발 코드를 위해 여러분의 정책과 표준 내에서 보안 코딩 원칙을 채택하라.

- 여러분의 개발자들에게 안전한 코드 기술을 훈련하고 이들이 그 기술을 최신으로 유지하도록 보증하라.
- 여러분의 예산에 보안 관련 코드 분석 도구를 포함하라.
- 여러분의 소프트웨어 생산자들이 보안의 중요성을 항상 인지하도록 하라.
- 여러분의 아키텍트, 디자이너 그리고 사업상의 사람들이 웹 어플리케이션 기본 보안 사항에 대해 알도록 훈련시켜라.
- 독립적인 평가를 제공할 수 있는 제 3자의 코드 감사자를 활용하는 것을 고려하라.
- 신뢰성이 있는 공개된 프랙티스를 채택하고 여러분의 제품 취약점 리포트에 적절하게 대응하는 프로세스를 구축하라.

참조

OWASP 프로젝트들

OWASP는 웹 어플리케이션의 보안을 위한 최고의 사이트이다. OWASP 사이트는 많은 프로젝트, 포럼, 블로그, 프리젠테이션, 툴, 그리고 문서들을 갖고 있다. OWASP의 일 년에 주요한 두 개의 웹 어플리케이션 보안 컨퍼런스를 개최하고 80 개가 넘는 지역 지부를 소유하고 있다.

다음의 OWASP 프로젝트가 가장 많이 사용된다:

- 안전한 웹 어플리케이션 구축을 위한 OWASP 가이드
- OWASP 테스팅 가이드
- OWASP 코드 검토 프로젝트 (개발 중)
- OWASP PHP 프로젝트 (개발 중)
- OWASP Java 프로젝트
- OWASP .NET 프로젝트

참고서적

- [GAL1] Gallagher T., Landauer L., Jeffries B., "*Hunting Security Bugs*", Microsoft Press, ISBN 073562187X
- [HOW1] Howard M., Lipner S., "*The Security Development Lifecycle*", Microsoft Press, ISBN 0735622140
- [HOW2] Howard M., Le Blanc D., "Writing Secure Code", 2nd ed., Microsoft Press, ISBN 0735617228
- [SCH1] Schneier B., "*Practical Cryptography*", Wiley, ISBN 047122894X
- [WYS1] Wysopal et al, *The Art of Software Security Testing: Identifying Software Security Flaws*, ISBN 0321304861

웹 사이트

OWASP, http://www.owasp.org

MITRE, 공통 취약점 목록(CWE) – 취약점 동향, http://cwe.mitre.org/documents/vuln-trends.html

SANS Top 20, http://www.sans.org/top20/

신용 카드 정보를 저장하거나 처리하는 모든 조직에 관련된 PCI 표준을 발표하는PCI 보안 표준 협회, https://www.pcisecuritystandards.org/

PCI DSS v1.1, https://www.pcisecuritystandards.org/pdfs/k_pci_dss_v_1-1.pdf

US CERT의 보안 설계, https://buildsecurityin.us-cert.gov/daisy/bsi/home.html

한글화 번역 프로젝트에 도움을 주신 전문가분들

- 번역 프로젝트 메니저 및 수석 감수: 박형근(Hyungkeun Park, CISSP/CISA, IBM Korea)
- 번역 참여 (ㄱㄴㄷ 순임)
  구형준(Koo, Hyung Joon, CISSP/CISA/CCNA, 삼성네트웍스)
  김광훈(Gwanghoon Kim, ISO27001 Lead Auditor, LG CNS)
  김경기(Kim Kyung Gi, CISSP/CCNA, 금융 ISAC 파트)
  김지선(Elleannah Kim, ㈜하우리)
  김학수(Kim, Hak Soo, ㈜이글루시큐리티)
  김현욱(Hyun_ouk, Kim, CISSP/CISA, 지에스서비스)
  박종일(PARK, JONG-IL, CCNA/CCIP/SCJP, 인하대)
  배선봉(Bae, Sunbong, CISSP/CISA, ㈜인젠시큐리티서비스)
  서영규(Youngkyu Seo, CISSP/CISA, CJ 시스템즈)
  송용준(Yong Jun Song, KAIST)
  채규혁(Chae, Kyu-hyeog, CISSP, 아이티시에스)

그 밖의 본 문서에 대한 감수와 2007 OWASP TOP10 한글화 번역 프로젝트에 참여 및 도와 주신 모든 SecurityPlus 가족 여러분들께 감사의 뜻을 전합니다. 끝.