



개발 라이프사이클 관점의 웹 애플리케이션 보안

2011/05/18

한국 IBM
박성민 차장

목 차

- 웹 애플리케이션 보안 동향
- 웹 애플리케이션 보안이 중요한 이유 및 피해 사례
- 웹 취약점 예 : SQL Injection
- 개발 라이프사이클 관점의 보안
 - ▶ 보안 담당자 : Black Box 테스트를 통한 취약성 발견
 - AppScan Standard / Enterprise Edition
 - ▶ 개발자 : White Box 테스트를 통한 취약성의 Early Detection
 - AppScan Source Edition
- Composite Analysis
- 기대 효과 및 사례
- 요약

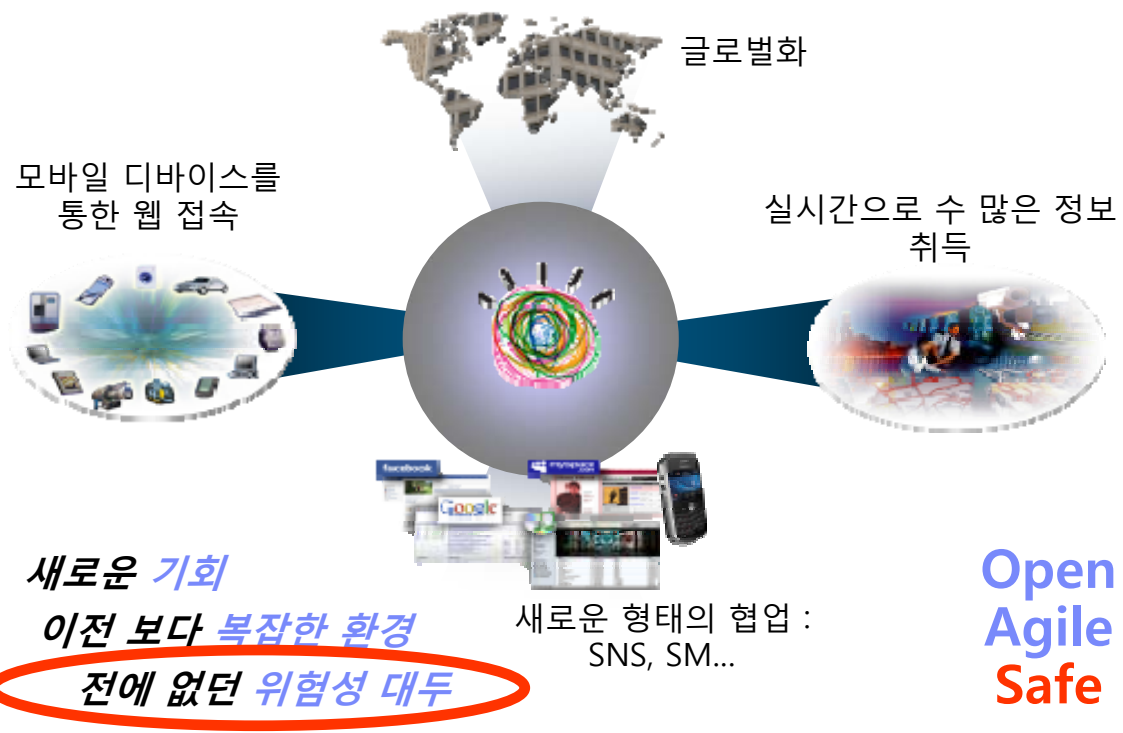


웹 중심의 비즈니스와 보안 동향



새로운 패러다임

보안 동향

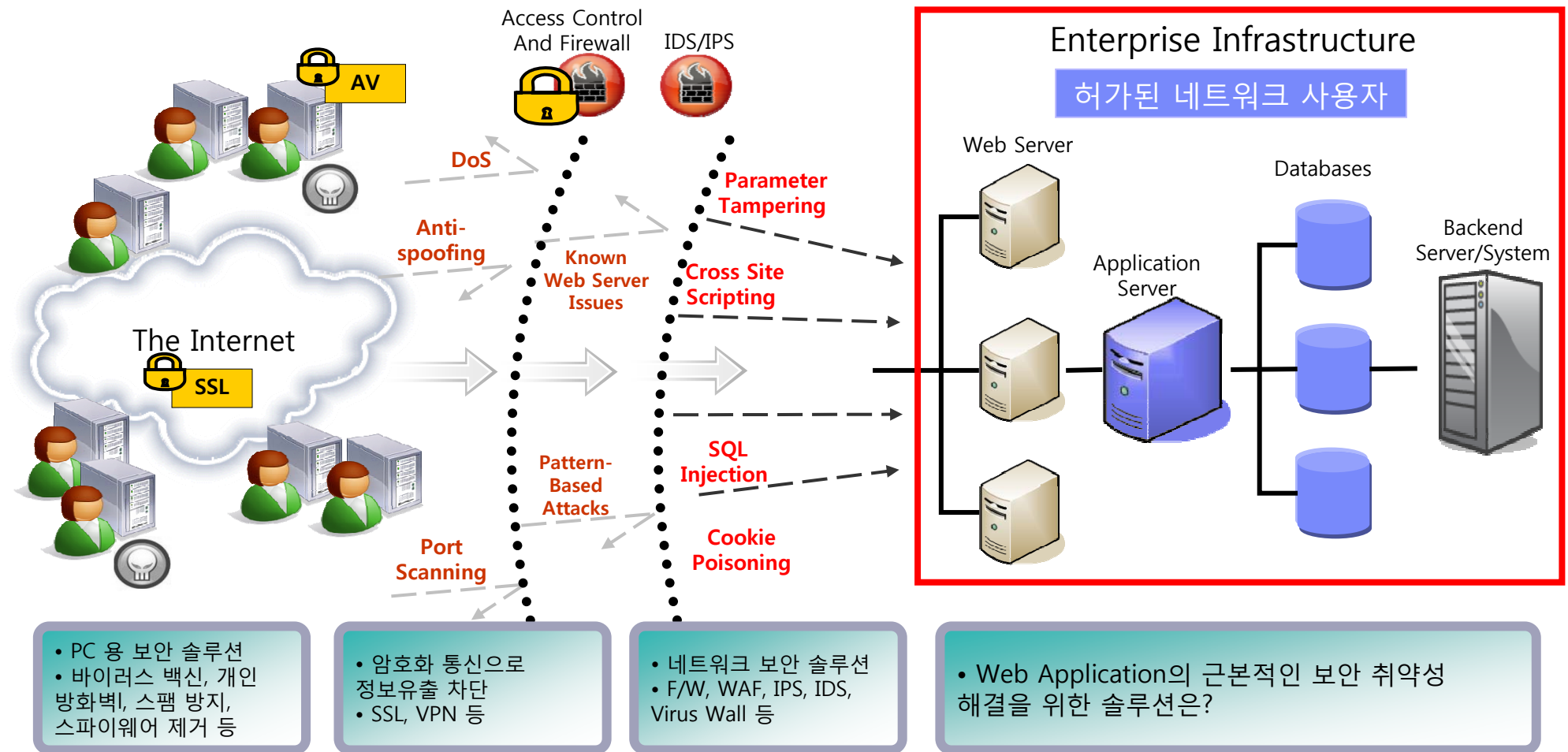


- 사이버 범죄 시장 규모 > 마약 시장 (FBI, 2007)
- 사이버 범죄가 과시를 위한 해킹이 아니라 조직적이며 사업적인 성격을 띠고 있음
- 해커들의 방법이 점차 진화되며 공격이 점점 더 악의적으로 변함
- 피해자의 유형이나 집중되는 분야가 급속도로 확산
- 피해자가 피해를 입었는지도 알기 어려움
- 지난 몇 년간 해킹 공격이 매년 2배 이상 증가

- 웹이 애플리케이션의 주된 인터페이스 환경
- 비즈니스가 환경, 고객, 파트너, 직원 등에 대해 개방 되어야 하며 변화에 신속하게 대응할 수 있어야 하는 동시에 안전해야 함
- 개발 시점에 보안을 염두 해 두고 개발되지 않음

보안의 영역 및 웹 애플리케이션

- 기존의 정적인 문제 대응 방식의 방법으로는 웹 애플리케이션을 타겟으로 행해지는 공격을 막아낼 수 없기 때문에 근본적인 문제를 해결하는 것이 중요

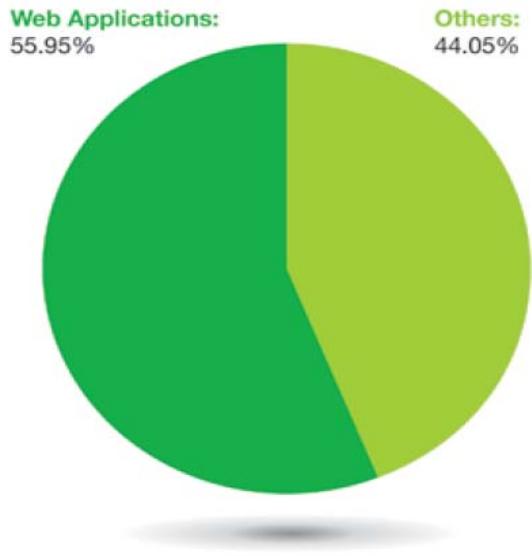




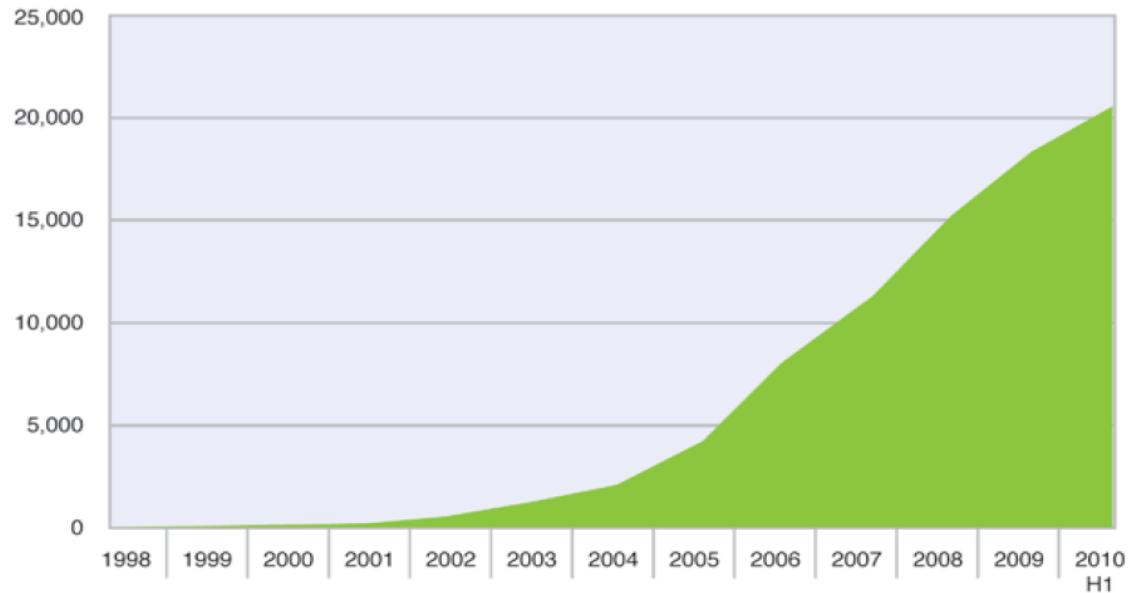
웹 애플리케이션의 취약성은 지속적으로 증가

- 2010년 상반기에 발견된 보안 문제점 중 약 56%가 웹 애플리케이션과 연관
- 발견되는 웹 애플리케이션 취약성은 큰 폭으로 증가 추세에 있음
- 80%의 기업 및 조직들이 애플리케이션 보안과 관련된 문제를 겪을 가능성이 있음 (Gartner)

Percentage of Vulnerability Disclosures that Affect Web Applications 2010 H1



Cumulative Count of Web Application Vulnerability Disclosures 1998-2010 H1



IBM Internet Security Systems 2010 X-Force® Mid-Year Trend & Risk Report



피해 사례

- 보안 사고 발생 시 미디어의 타겟이 되며, 법적 책임 및 이와 관련된 비용과 더불어 기업의 신뢰도 하락 등의 문제가 발생하게 됨



A 홈쇼핑 회원 200만명 개인정보 유출(종합)

- 연합뉴스, 2005년 3월

정보유출 B은행 피해고 1000여명

- 경향신문

C 웹 마켓, 해킹으로 개인정보 유출

- 파이낸셜 뉴스, 2008년 2월

D 게임 '개인정보 유출' 원고 승소

- 디지털타임스, 2009년 6월

국내 유명 백화점 인터넷 사이트나 25개 온라인 사이트의 회원정보 20

- 서울신문, 201

E 금융사

- 해커 협박 전 두 달 간 해킹 사실 파악 못함

- 고객 정보, 신용 등급 정보 유출

- 신용도 하락

버지니아 주 보건부 웹 사이트를 공격한 해커가 거액을 요구

- 워싱턴 포스트, 2009년 5월

국내 해킹 사고 증가세..보안 위협 심각

- 아시아 경제, 2010년 9월

해킹 당한 소니 "1인당 최대 10억원 피해보상

- 한국경제, 2011년 5월

DDOS, 이젠 근원적 처방 고민할 때

- 전자신문, 2011년 3월



피해 사례

- Hackers breach Heartland Payment credit card system – USA Today (2009/1/23)
- 2008년 5월부터 2009년 1월까지 6개월이 넘도록 악성코드를 이용한 해킹을 당해 1억3,000만 건의 개인 정보가 유출되면서 사상 최대의 해킹 사고라는 오점을 남김

- ▶ 기업의 신뢰도 하락
- ▶ 고객 감소
- ▶ 시장 점유율 하락
- ▶ 평판에 타격
- ▶ 법적 책임 및 이에 따른 비용
- ▶ 규제 및 감사 비용
- ▶ ...

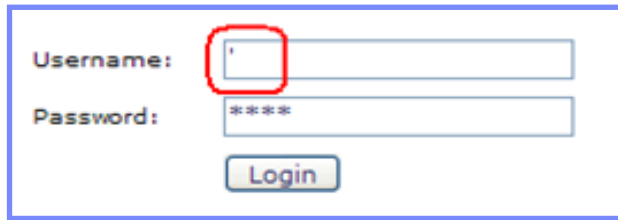


IBM 보안 프레임워크



웹 취약점의 예 : SQL Injection


- 애플리케이션 탐색을 위한 시도



Username:

Password:

Login



An Error Has Occurred

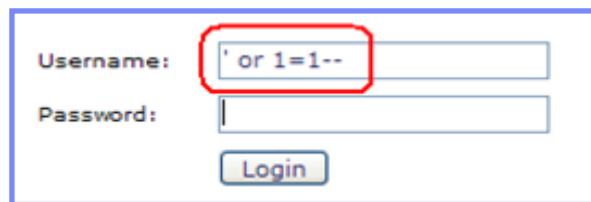
Summary:

Syntax error (missing operator) in query expression 'username = "" AND password = 'xxxx''.

Error Message:

System.Data.OleDb.OleDbException: Syntax error (missing operator) in query expression 'username = "" AND password = 'xxxx''.

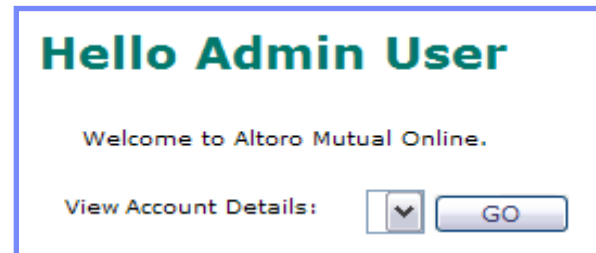
- 에러 메시지가 해커에게 스트링이 사용된다는 점, DB의 종류 등에 대한 정보를 제공
 - ▶ SQL 문 확인 가능 : 'username = "" AND password = 'xxxx''
- **'username = " or 1=1 --' AND password = 'xxxx'**



Username:

Password:

Login



Hello Admin User

Welcome to Altoro Mutual Online.

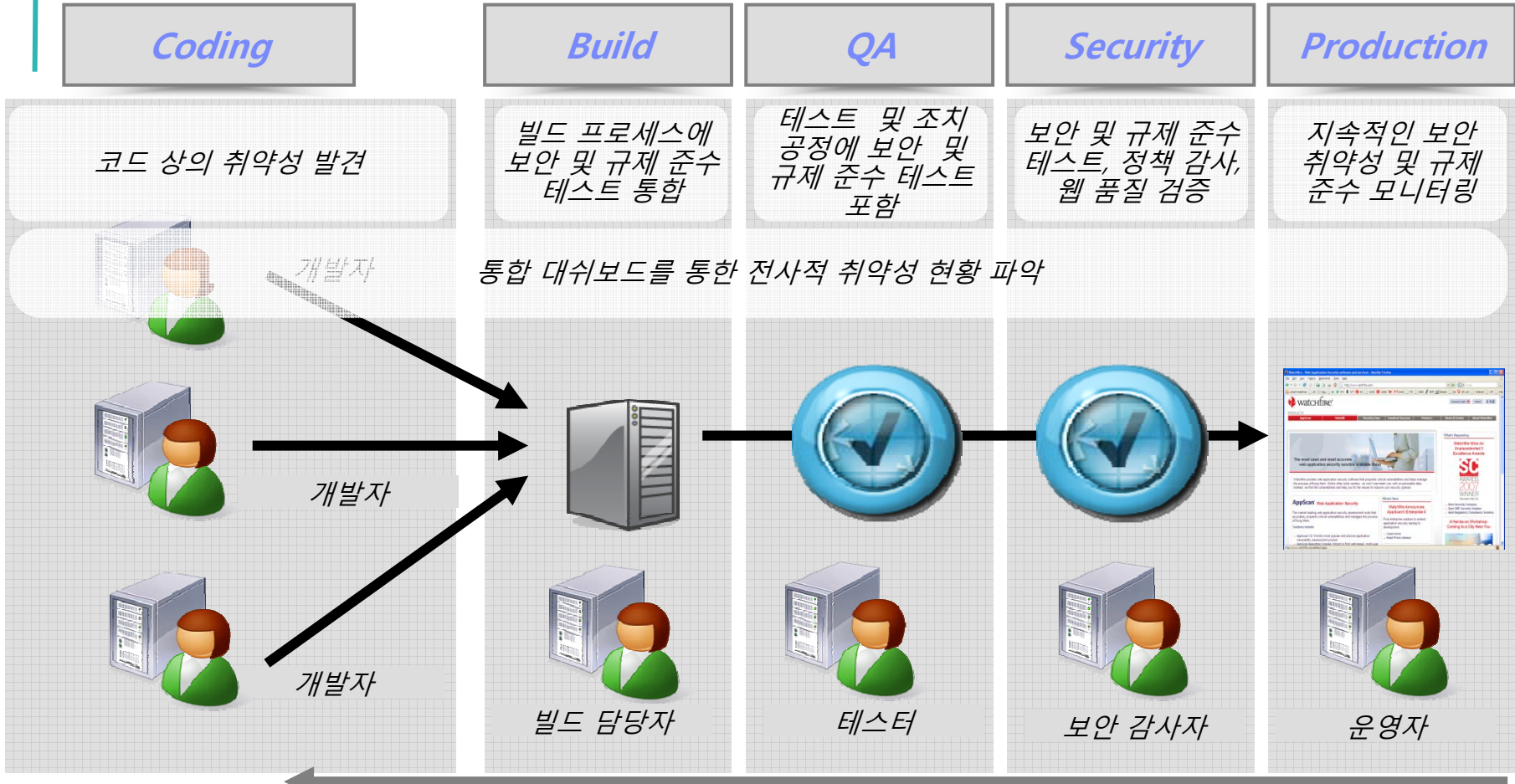
View Account Details:

- 허가되지 않은 데이터베이스의 데이터 접근 및 수정 > 정보 유출
- 테이블 삽입, 삭제
- 데이터베이스 서비스 중지, 명령어 실행 등

개발 공정 상의 보안 테스트



소프트웨어 개발 공정



애플리케이션 보안 테스트의 성숙도





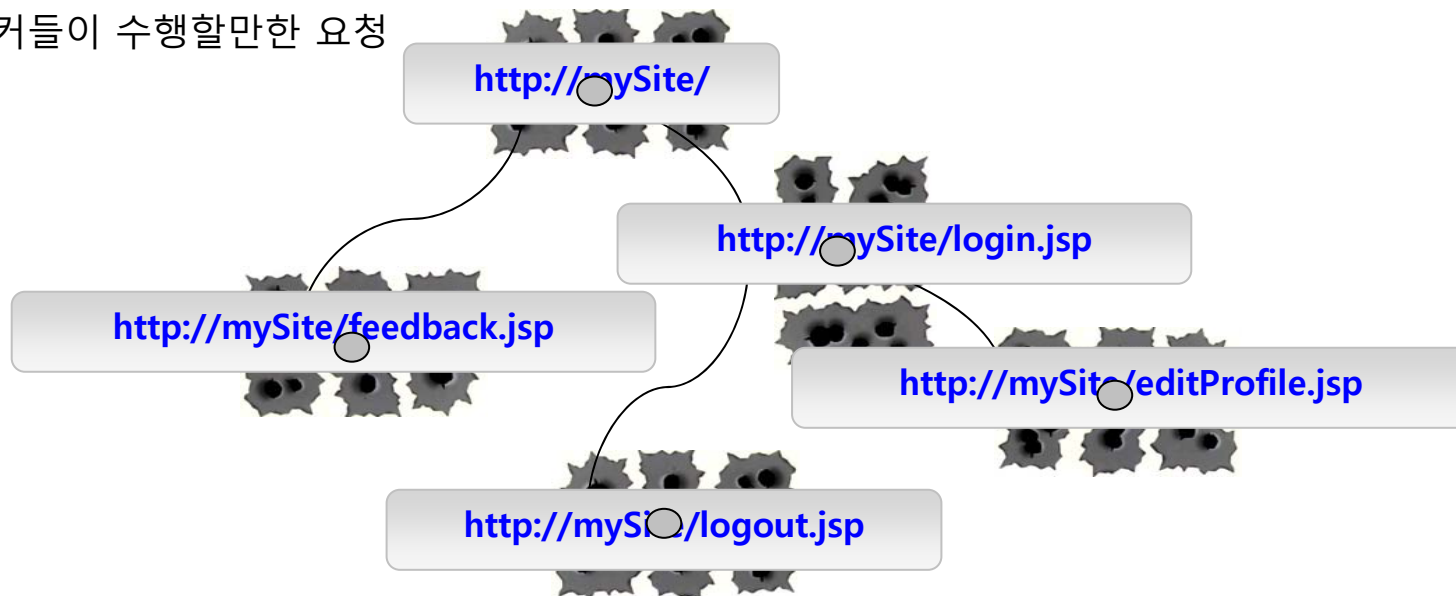
웹 취약성 테스트 방식 1 : Black Box Test

단계 1: 일반 사용자와 같이 페이지 내의 링크를 통해 웹 사이트 탐색

- ▶ 링크 클릭 / 정상적인 입력에 의한 요청
- ▶ `http://mySite/page.jsp?param=value`

단계 2: 변조된 요청(http request)을 통해 웹 애플리케이션의 취약성 검사

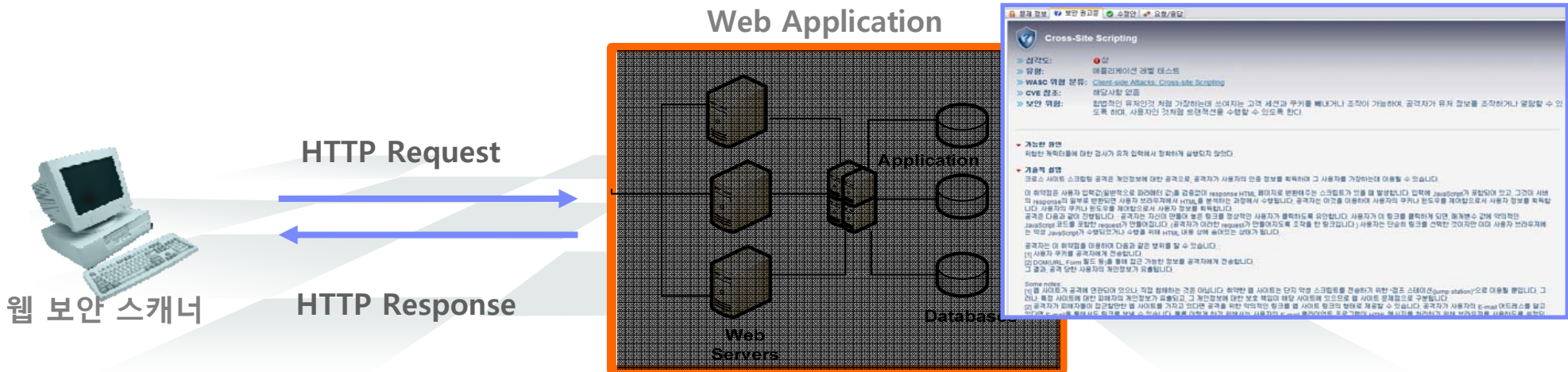
- ▶ `http://mySite/page.jsp?param="" or 1=1 or "" -"`
- ▶ `http://mySite/page.jsp?param= ' or 1=1 --`
- ▶ `http://mySite/page.jsp?param=' --`
- ▶
- ▶ 실제 해커들이 수행할만한 요청



AppScan Standard Edition



- 웹 애플리케이션의 보안 점검 도구로 보안 상의 취약점을 보다 쉽게 발견하고 해결
 - ▶ HTTP 요청의 결과에 근거하여 해당 애플리케이션이 취약한지 평가 : Black Box 방식
- 발견된 문제에 대해서는 단순한 리스트 이상의 정보가 필요하므로 취약점에 대한 포괄적이고 자세한 정보 제공
 - ▶ 문제 정보, 보안 권고문, 수정안 등
- 자바스크립트에 대한 정적 분석을 통해 클라이언트 측의 보안 취약성을 검출
- 새로 발견된 취약성에 대한 주기적인 Rule 업데이트
- 테스트 분석 후 조직 내의 다양한 팀원들과 결과를 공유할 수 있는 여러 관점의 리포트 제공
 - ▶ 산업 표준 / Compliance 등

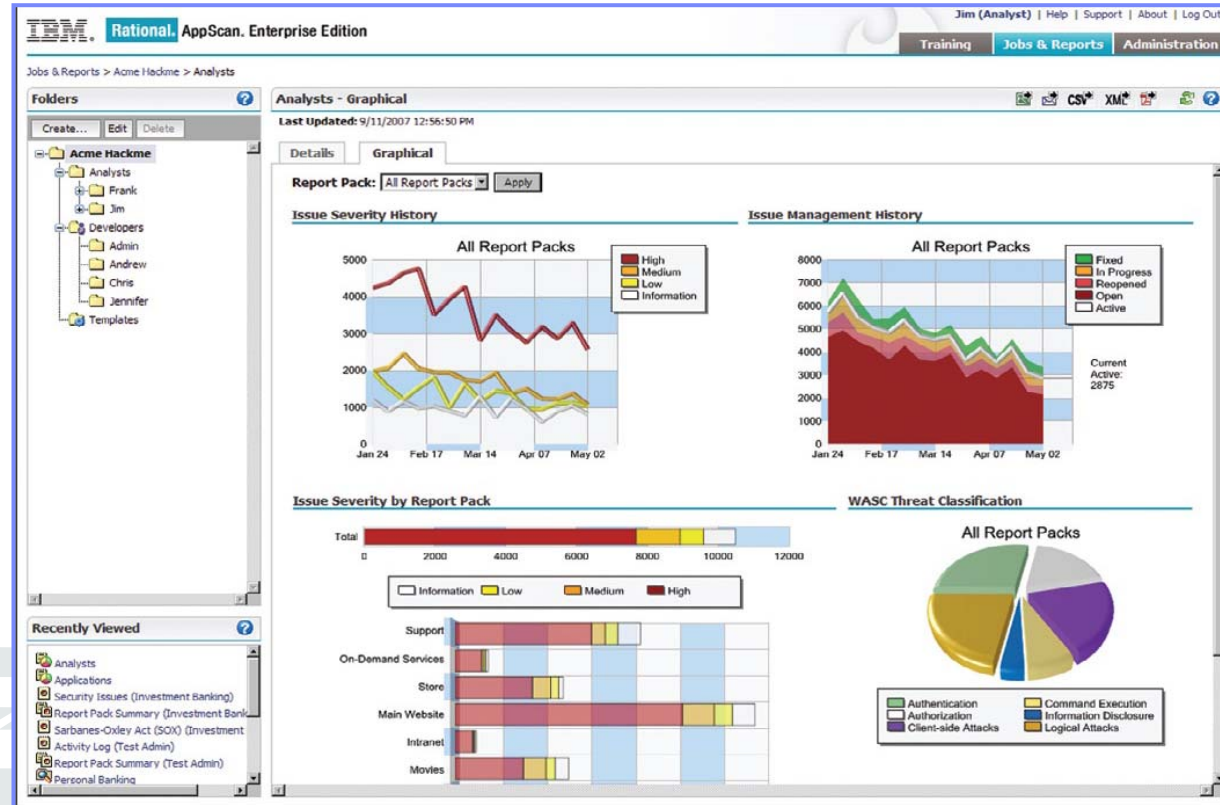


AppScan Enterprise Edition

보안 취약성 검증을 수행하고 이에 관한 정보를 중앙에서 취합하여 평가해야 하는 보안 팀을 위한 웹 기반의 보안 솔루션

AppScan Standard Edition의 기능 +

- 전사적으로 제공되는 가시성
- 대시보드를 통한 중앙 집중화된 관리 및 테스트 진행 상황 파악
 - ▶ 발견된 보안 취약성 / 해결 상황
 - ▶ 테스트 대상 별 문제점 상황
- 이슈 관리 기능 제공
 - ▶ 워크플로우 포함 (In Progress, Fixed ...)
- 사용자 역할 별 업무 할당 및 접근 제어 기능

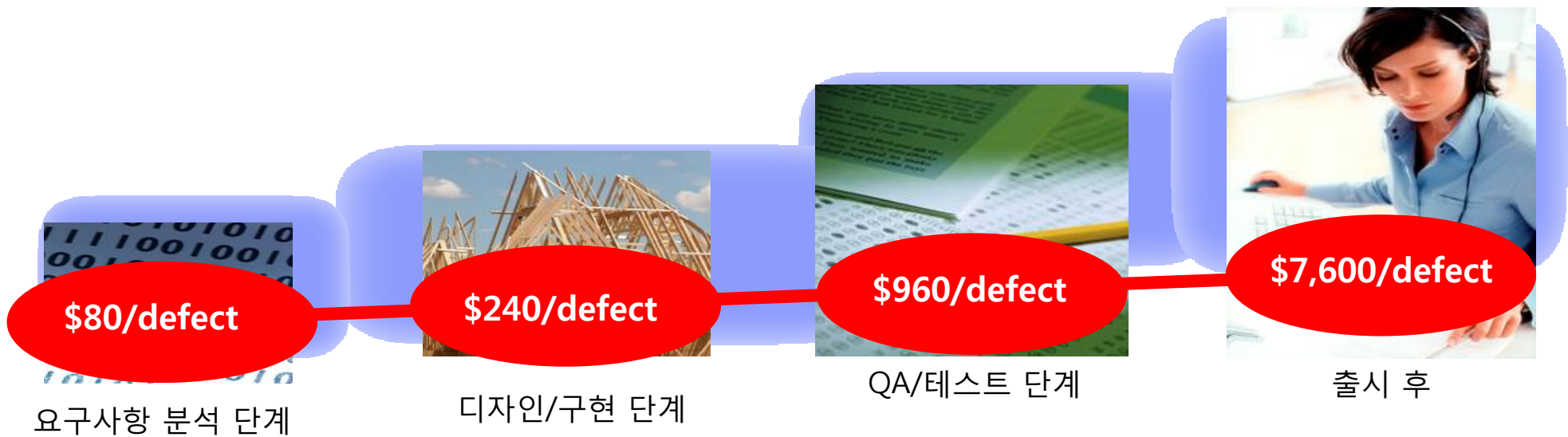




Early Vulnerability Detection

- 개발자 참여 필요
 - ▶ 애플리케이션의 취약점 조치는 개발자가 수행해야 함
- 개발 단계에서부터 검증 필요
 - ▶ 개발 후 보안 취약점 발견 및 해결을 위해서는 별도의 시간 및 비용 소모
 - 개발 완료 시점에는 현실적으로 전수 검사가 불가능
 - 개발 기간 만큼의 시간이 소요될 수 있음 (Gartner)
 - ▶ 개발 공정의 앞 단계에서 보안이 고려될 수록 비용 절감 효과를 얻을 수 있음 (정통부)
 - 설계 단계 21%, 개발 단계 15%, 테스트 단계 12%

취약성 검토 시점





웹 취약성 테스트 방식 2 : White Box Test

- 잘못된 데이터 입력을 인식 (변수, 쿼리문에 반영되는 것 추적)
- 쿼리문의 필터링(validation routine) 없이 Sink에 의해 직접 실행될 수 있음을 확인



Source – 문제가 있는 문자열을 리턴하는 메소드

```
String username = request.getParameter("username");
```

```
// ...
String username = request.getParameter("username");
String password = request.getParameter("password");
```

```
// ...
String query = "SELECT * from tUsers where " +
"userid=" + username + " " +
"A";
```

```
String query = "SELECT ..." + username
```

```
// ...
ResultSet rs = stmt.executeQuery(query);
```

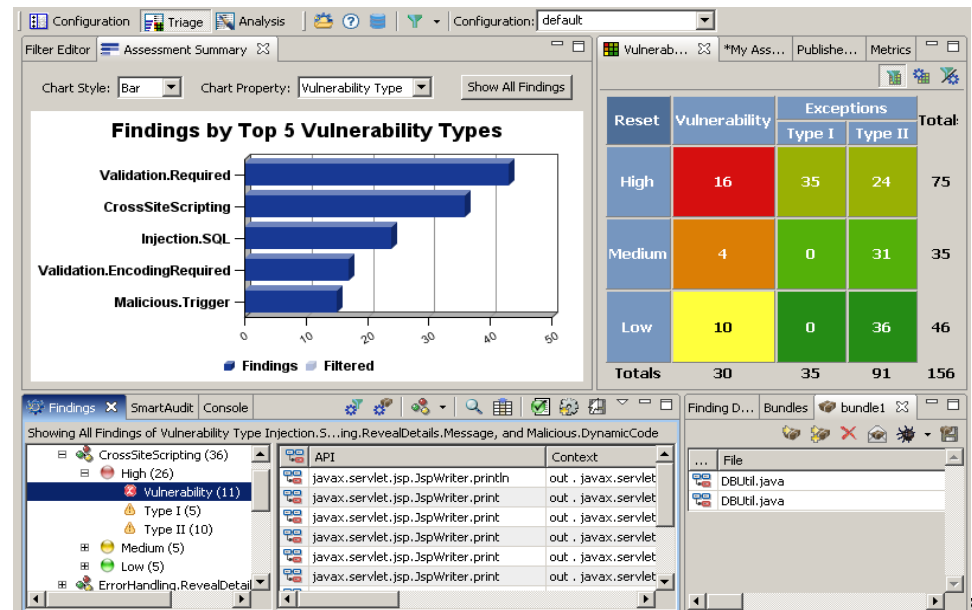
```
ResultSet rs = stmt.executeQuery(query);
```

사용자가 SQL 문을 조작 가능 문제가 되는 문자열을 걸러주는 **Validation Routine**의 부재

Sink – 보안 문제를 발생시킬 여지가 있는 메소드

AppScan Source Edition

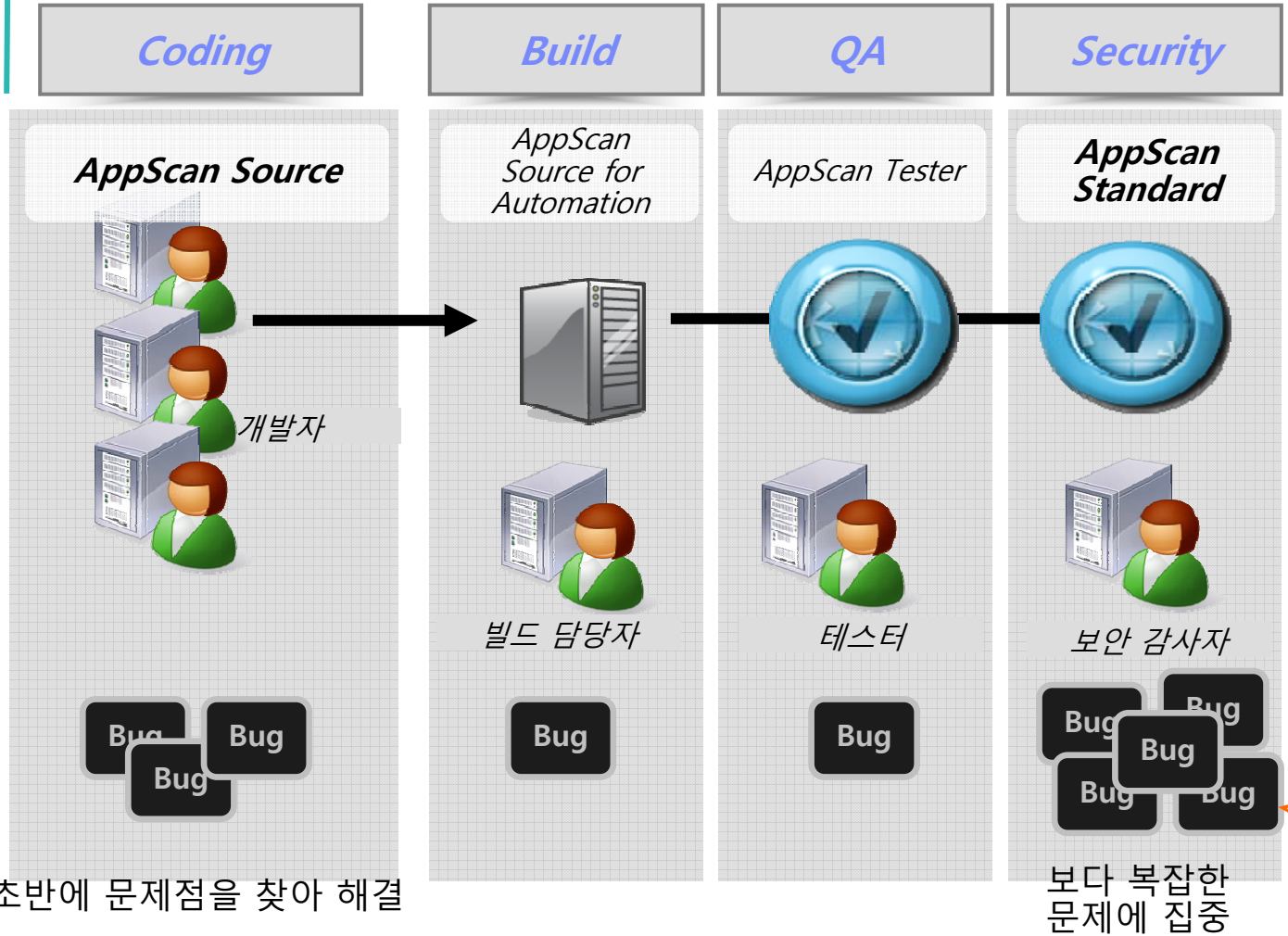
- 현실적으로 보안을 고려하여 개발할 수 있는 개발자가 많지 않음 (Gartner)
- 보안 관점에서 소스 코드 inspection을 수행
 - ▶ C/C++, Java, JSP, .Net, ASP 등
 - ▶ Eclipse, Visual Studio .Net, Rational Application Developer 등과 통합
- 취약성을 파악하고 우선 순위화 할 수 있는 스캔 결과 제공
 - ▶ 발견된 취약점들의 수, 특정한 보안 관점의 고려사항, 애플리케이션 위험요소 평가 등을 포함하여 분류
- 애플리케이션의 취약성을 발견하고 문제를 해결할 수 있도록 가이드 등을 제공
 - ▶ Call 그래프, 데이터 추적 등의 분석 정보 제공
 - ▶ 발견된 각 취약점에 대해서 소스 코드를 리뷰하고 문제 해결을 위한 권고 사항 제공
- 사용자 정의 룰 지원 및 공유
 - ▶ In house 프레임워크 / 취약성에 적용
 - ▶ 전사적으로 일관된 보안 정책 유지 가능



개발 공정에 걸친 취약성 점검



소프트웨어 개발 공정



초반에 문제점을 찾아 해결

잠재적인 취약성

- 1. 문제 해결 비용 감소
- 2. 같은 노력으로 품질 개선





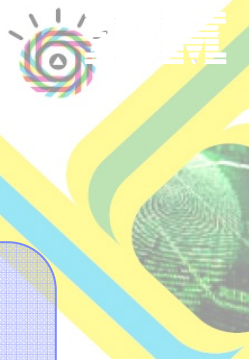
Black box 분석

- 결과의 정확성
- 소스가 필요 없음
- 코드 커버리지
- HTTP 관점의 테스트에 국한
- 완성된 애플리케이션이 필요
- 테스트 선 조건이 많지 않음
- 원격의 해커와 유사하게 테스트

Composite Analysis

- 코드 커버리지
- 주어진 코드에 국한됨
- HTTP 검증 이상의 테스트
- 부분적인 애플리케이션 테스트 가능
- 지원 언어 / 프레임워크 제약
- 완성된 애플리케이션이 필요치 않음
- 오탐이 상대적으로 많음

Black Box / White Box 점검 결과



URL 기반 -

- 내 애플리케이션 (50)
- http://appscandemo08:8080/ (50)
- altoromutual (50)

내 애플리케이션에 대한 보안 문제: 50개 (변형: 190개)

- 세션 ID가 업데이트되지 않음 (1)
- http://appscandemo08:8080/altoromutual/doLogin (1)
- 예측 가능한 로그인의 실패 (1)
- 예측 가능한 로그아웃의 실패 (1)
- 올바르지 않은 계정 잠금 (1)
- 크로스 사이트 스크립팅(XSS) (13)
- http://appscandemo08:8080/altoromutual/bank/customize.jsp (2)
 - lang
 - customize.jsp
- http://appscandemo08:8080/altoromutual/bank/queryxpath.jsp (2)
- http://appscandemo08:8080/altoromutual/bank/showAccount (2)
- http://appscandemo08:8080/altoromutual/bank/showTransactions (3)
- http://appscandemo08:8080/altoromutual/search.jsp (1)
- http://appscandemo08:8080/altoromutual/sendFeedback (3)
- 크로스 사이트 스크립팅(XSS) 사이트 유형: 주요 (8)

Black Box 분석 결과

- 실제 문제가 발생된 웹 페이지
- 문제가 된 요청 / 응답

White Box 분석 결과

- 취약성이 존재하는 코드의 위치
- 데이트 흐름, 메소드 정보

IBM Rational AppScan Source Editor for Security

찾은 결과 (680)

위험성(11)	유형 II(8)	유형 I(9)	중간(5)	Info(S19)	Injection(S19)	출력(19)	유형 I(17)				
AccessControl.Bypass(8)	Authentication.EnhBy(2)	CrossSiteScripting(33)	높음(28)	위험성(11)	유형 II(8)	유형 I(9)	중간(5)	Info(S19)	Injection(S19)	출력(19)	유형 I(17)

Rendered Test Response

AltoroMutual

위약성

JSP JspWriter

javax.servlet.jsp.JspWriter 클래스는 HTML 페이지를 사용자의 브라우저에 송신합니다. 이 클래스는 일반적으로 보통 Java 파일에 직접 사용되지 않습니다. 그 대신 주로 JSP 서버의 JSP 페이지에서 사전 컴파일된 Java 파일에서 찾을 수 있습니다.

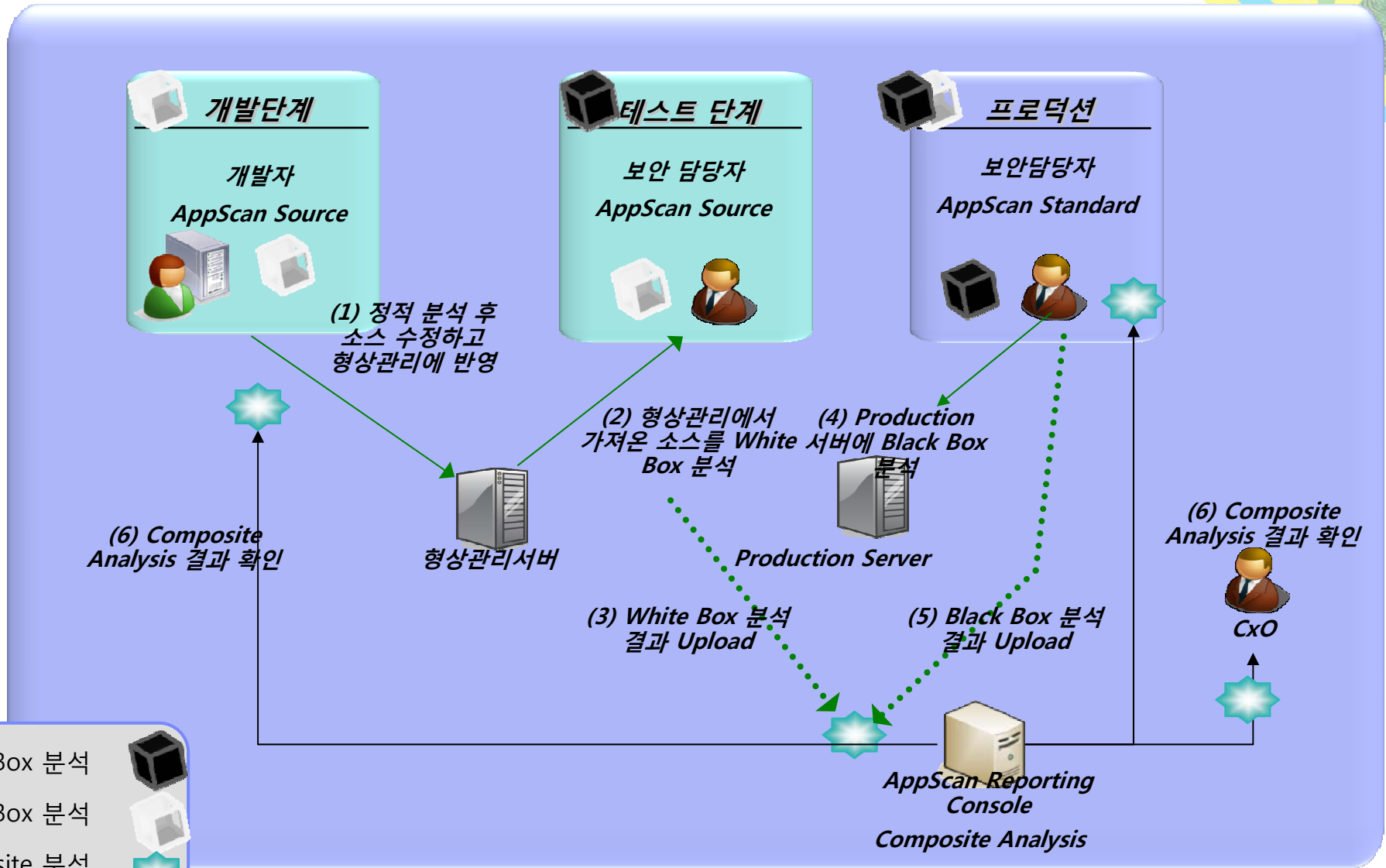
이 JspWriter API는 문자열을 HTML 페이지에 송신합니다. 문자열이 정적 문자열이거나 신뢰성 있는 소스에서 생성되는 경우 문제가 되지 않습니다. 그러나 문자열이 사용자 입력 또는 다른 신뢰되지 않는 소스에서 생성되는 경우, 공격자는 악성 스크립트(예: JavaScript)를 출력 HTML 페이지에 가져올 수 있습니다. 자체 보안이 스크립트는 스크립팅 가능 브라우저에서 해석되며, 발전 웹 사이트와 통신하기 위해 사용자의 보안 컨텍스트를 보낸 후 실행되므로 XSS(Cross-site scripting) 공격을 유발할 수 있습니다.

연관

이러한 문제점을 방지하려면, 애플리케이션은 문자열에 대한 적절한 유효성 검증 및 인코딩을 적용해야 합니다. 유효성 검증 메커니즘은 문자열에 악성 데이터 및 코드가 없는지 확인해야 합니다. HTML 렌더링 인코딩은 스크립트로 해석하도록 의도되지 않는 데이터에 적용해야 합니다. 유효성 검증 및 인코딩에 대한 자세한 사항은 Validation.Required 및 Validation.EncodingRequired를 참조하십시오.



Composite Analysis 점검 절차 예시





Composite Analysis : Black Box/White Box 결과의 연계

- 동적 분석 도구와 정적 분석 도구로 발견된 취약성들을 서로 연관시켜 문제를 분석
- 문제 해결의 우선 순위를 판단하는데 도움
 - 두 가지 방법에 의해 모두 발견된 취약성은 실제 존재하며 악용될 가능성이 높음

상관된 보안 문제

마지막 업데이트 날짜: 2011-04-28 오후 1:51:20

조치: Excel로 내보내기 | 적용

문제 유형

링크 인젝션(크로스 사이트 요청 위조 유도)

	동적 문...	테스트 URL	요소	정적 문...	소스 파일	API	행
<input type="checkbox"/>	5407	http://appscandemo08:8080/alto...	lang	6051	%workspace%\AltoroJ 2.1\WebC...	javax.servlet.j...	23
<input type="checkbox"/>	5384	http://appscandemo08:8080/alto...	query	5629	%workspace%\AltoroJ 2.1\WebC...	javax.servlet.j...	12
<input type="checkbox"/>	5428	http://appscandemo08:8080/alto...	query	5694	%workspace%\AltoroJ 2.1\WebC...	javax.servlet.j...	24
<input type="checkbox"/> (4개 항목 합계)							

크로스 사이트 스크립팅(xss)

	동적 문...	테스트 URL	요소	정적 문...	소스 파일	API	행
<input type="checkbox"/>	5394	http://appscandemo08:8080/alto...	lang	6051	%workspace%\AltoroJ 2.1\WebC...	javax.servlet.j...	23
<input type="checkbox"/>	5398	http://appscandemo08:8080/alto...	query	5629	%workspace%\AltoroJ 2.1\WebC...	javax.servlet.j...	12
<input type="checkbox"/>	5383	http://appscandemo08:8080/alto...	query	5694	%workspace%\AltoroJ 2.1\WebC...	javax.servlet.j...	24
<input type="checkbox"/> (3개 항목 합계)							

블라인드 SQL 인젝션

	동적 문...	테스트 URL	요소	정적 문...	소스 파일	API	행
<input type="checkbox"/>	5412	http://appscandemo08:8080/alto...	uid	5824	%workspace%\AltoroJ 2.1\src\co...	java.sql.State...	112
<input type="checkbox"/>	5403	http://appscandemo08:8080/alto...	passw	5824	%workspace%\AltoroJ 2.1\src\co...	java.sql.State...	112
<input type="checkbox"/>	5412	http://appscandemo08:8080/alto...	uid	5648	%workspace%\AltoroJ 2.1\src\co...	java.sql.State...	135
<input type="checkbox"/> (3개 항목 합계)							

SQL 인젝션을 사용한 인증 무시(bypass)

	동적 문...	테스트 URL	요소	정적 문...	소스 파일	API	행
<input type="checkbox"/>	5401	http://appscandemo08:8080/alto...	passw	5824	%workspace%\AltoroJ 2.1\src\co...	java.sql.State...	112
<input type="checkbox"/> (1개 항목 합계)							

Black Box 분석 결과 :
취약성이 발견된 페이지
정보

White Box 분석
결과 : 소스 파일/행
및 API 정보



기대 효과

자동 테스트 수행을 통한 비용 절감 및 품질 개선

- 자동 스캔을 통해 수작업 진단 대비 높은 생산성 기대
- 주기적인 애플리케이션 / 코드 스캔으로 웹 애플리케이션에 대한 비용 효과적인 보안 검증이 가능
- 일관된 보안 정책 적용과 스캔 커버리지 개선으로 애플리케이션의 보안 품질 향상



- 전문 업체에 의한 진단은 일반적인 웹 애플리케이션 당 일정 금액의 예산이 필요하며, 매 회 진단에 따른 비용 지불로 주기적인 진단 시 장기적으로 높은 지출이 불가피
- 수작업에 의한 진단 신뢰성이 낮으며 상시 진단이 어려움

개발 프로세스 상의 조기 취약성 검증 - 비용 절감

- 80%의 개발 비용이 결함을 발견하고 이를 조치하는데 소요되며, 개발 완료 시점에 가까울 수록 결함을 발견하고 조치하는데 더 많은 비용과 노력 소요
- 개발 프로세스 초기(코딩 단계)에서 문제를 찾아 해결함으로써 불필요한 비용 절감 가능



- 문제 발견 및 조치 비용
 - ▶ 코딩 단계에서 \$80, QA/테스팅 단계에서 \$960 *
 - ▶ Ex) 연간 50 개의 웹 애플리케이션과 애플리케이션 당 25개의 취약성을 가정할 때 코딩 시점에 검증이 이루어질 경우 연간 \$1.1M 절약 가능

보안 사고 예방

- 개인 정보 유출 사고와 같은 보안 사고 시
 - 법적 책임 및 이에 따른 비용
 - 규제 및 감사 비용
 - 기업의 신뢰도 하락 및 고객 감소 등의 문제 발생



- 개인 정보 한 건당 \$204의 가치
- 보안 사고 당 \$6.6M의 비용 소요**
- 모 은행 2억 여 원 배상 사례 + 무형 손실

* Source: GBS Industry standard study
** Source: Ponemon Institute 2009-10



사례 및 효과

1. 게임 포털 사이트 'P'를 보유하고 있는 N 포털 사는 급증하는 방문자 트래픽과 이에 따른 악의적 해킹 시도로부터 사이트를 보호하고, 안전한 웹 서비스를 제공하기 위한 보안 강화로 솔루션 도입

보안 점검 효율성

- 개발자가 수작업으로 모의 해킹
- 점검 업무 하루(8시간) 이상 소요
- 웹 취약성 점검 후 서비스의 퍼블리쉬(결과 정리, 개발팀에 요청, 해결, 검증 및 서비스 공개)까지 일주일 이상 소요
- 수작업으로 리포트 작성



- 모의 해킹 자동화
- 자동 점검을 통해 3시간 내로 단축
- 취약성 점검 후 퍼블리쉬 과정이 1일 이내로 80% 이상 단축 효과
- 리포트 자동 생성

웹 서비스 보안 안정성

- 개발자 경험에 전적으로 의존한 보안 점검의 한계로 최신 해킹 패턴에 대한 대응 부재
- 이미 서비스된 애플리케이션은 보안 점검이 어려움



- 광범위한 해킹 패턴 보유 및 국내/외 최신 해킹 패턴의 주기적인 업데이트로 보다 완전한 보안 점검이 가능
- 스케줄링 기능으로 대외 서비스의 정기/비정기적 상시 보안 점검이 가능

2. K 통신사는 보안 솔루션 도입으로 주기적인 자체적인 인증제를 포함하는 웹 애플리케이션 진단 프로세스를 확립

- 제한적인 애플리케이션 보안 : 진단 비율이 30% 수준으로 여러 보안 문제를 내제

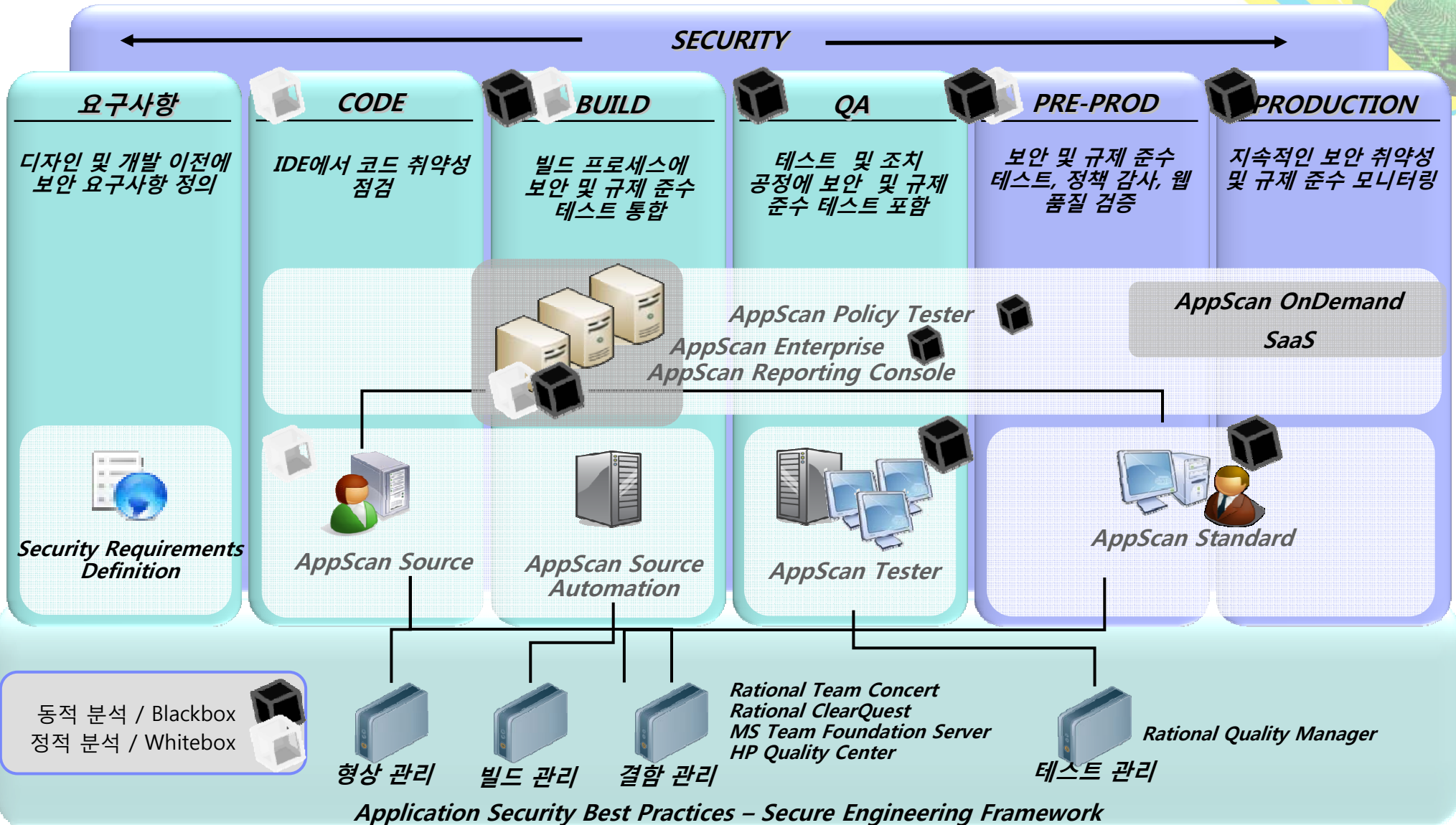


- 외주 개발사를 포함한 애플리케이션 보안 진단 비율이 95% 이상으로 높아짐
- 웹 애플리케이션 보안 및 안정성 향상으로 기존 대비 보안사고 90% 감소 효과





소프트웨어 개발 공정 전체의 보안 솔루션



요약

- 웹 응용프로그램 에는 근본적인 취약성이 존재하고 그에 따른 피해는 심각
- 보안 피해는 "소 잃고 외양간 고치는" 꼴이므로 현실적으로 가시화되고 있는 피해에 대하여 적극적으로 대처 해야함

For Security Team

- 고객 고충
 - ▶ 인력에 의한 주기적인 진단은 시간/비용이 많이 소요되며, 일관된 진단 및 결과를 기대하기 어려움
 - ▶ 발견된 취약성을 해결하기 위한 정보가 부족하며
- 고객이 얻는 가치
 - ▶ 자동화 도구의 최신 취약성 패턴으로 상시 취약성을 통한 진단 결과의 신뢰성
 - ▶ 발견된 취약성 및 이에 대한 다양한 정보를 제공하며 이를 개발팀과 공유하여 효과적인 문제 해결

For Development

- 고객 고충
 - ▶ 개발 단계에서 발견하여 해결할 수 있는 문제들이 적절한 인력과 솔루션의 부재로 적시에 발견되지 못함
 - ▶ 이에 따라 개발 공정 후반에 발견된 문제를 해결하는 데 많은 비용과 시간이 소요되어 비능률적
- 고객이 얻는 가치
 - ▶ 개발단계에서 코드 수준에서 발견할 수 있는 보안 취약성 검증
 - ▶ 개발 프로세스에서 보다 일찍 보안 이슈들을 발견하고 조치하도록 하여, 프로세스를 개선

- 개발 공정 전체에 걸쳐 여러 팀원들이 보안 문제 해결을 위해 노력
- 결과의 정확성 개선

보안 위험성 감소 및
비용 감소



감사합니다