情報保護學會論文誌
第18卷 第5號, 2008. 10

SQL 질의 애트리뷰트 값 제거 방법을 이용한 효과적인 SQL Injection 공격 탐지 방법 연구*

이 인 용†, 조 재 익, 조 규 형, 문 종 섭‡
고려대학교, 정보경영공학전문대학원

A Method for SQL Injection Attack Detection using the Removal of SQL Query Attribute Values*

In-yong Lee†, Jae-ik Cho, Kyu-hyung Cho, Jong-sub Moon‡
Graduate School of Information Management and Security, Korea University

요 약

인터넷이 발전함에 따라 웹 애플리케이션을 이용한 서비스가 대중화되었고, 웹 애플리케이션의 취약점을 목표로 하는 공격들도 증가하게 되었다. 많은 웹 공격 중의 하나인 SQL Injection 공격은 민감한 데이터를 처리하는 곳에서는 매우 치명적이고 위험하기 때문에 이를 탐지하고 예방하기 위한 연구들이 다양하게 이루어져 왔다. 이로 인하여 SQL Injection 공격들이 많이 감소했지만 아직도 이를 우회하는 방법들이 존재하며, 기존의 연구 방법들 또한 매우 복잡하여 실제 웹 애플리케이션에 적용하여 사용하기 어렵다. 따라서 본 논문에서는 SQL Injection 공격 탐지를 위해 웹 애플리케이션에 고정되어 있는 정적 SQL 질의와 사용자로부터 생성되는 동적 SQL 질의의 애트리뷰트 값을 제거한 정적 및 동적 분석 방법을 제안하고, 실험을 통하여 효율성을 검증하였다.

ABSTRACT

The expansion of the internet has made web applications become a part of everyday life. As a result the number of incidents which exploit web application vulnerabilities are increasing. A large percentage of these incidents are SQL Injection attacks which are a serious security threat to databases with potentially sensitive information. Therefore, much research has been done to detect and prevent these attacks and it resulted in a decline of SQL Injection attacks. However, there are still methods to bypass them and these methods are too complex to implement in real web applications. This paper proposes a simple and effective SQL Query attribute value removal method which uses Static and Dynamic Analysis and evaluates the efficiency through various experiments.

Keywords : Web Application Security, SQL Injection, Static and Dynamic analysis

접수일 : 2008년 8월 4일; 수정일 : 2008년 9월 4일;
채택일 : 2008년 10월 16일
*본 논문은 2006년 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임(KRF-2006-21-00461)
† 주저자, iylee@cist.korea.ac.kr
‡ 교신저자, jsmoon@korea.ac.kr

Ⅰ. 서 론

네트워크와 인터넷의 발전으로 인하여 많은 부분의 오프라인 서비스가 온라인 서비스로 전환되었으며, 현재 온라인 서비스의 대부분을 웹 서비스가 차지하고 있

다. 웹이 언제 어디에서나 서비스 제공이 가능하다는 장점으로 인하여 그 비중은 날이 갈수록 증가하고 있으며, 이를 노리는 공격 또한 증가하고 있다[1,2]. 웹을 대상으로 하는 공격 중에는 많은 부분이 웹 애플리케이션의 취약점을 노리는 공격이며, OWASP[2]에서는 대표적인 웹 애플리케이션 취약점 및 공격에 대해서 다음 [표 1] 같이 분류하여 발표하였다.

〔표 1〕 OWASP TOP 10

OWASP TOP 10 2007	MITRE 2006 Raw Ranking
A1. Cross Site Scripting(XSS)	1
A2. Injection Flaws (SQL Injection 포함)	2
A3. Malicious File Execution (NEW)	3
A4. Insecure Direct Object Reference	5
A5. Cross Site Request Forgery (CSRF) (NEW)	36
A6. Information Leakage and Improper Error Handling	6
A7. Broken Authentication and Session Management	14
A8. Insecure Cryptographic Storage	8
A9. Insecure Communications (NEW)	8
A10. Failure to Restrict URL Access	14

[표 1]에서의 Injection Flaws에 있는 SQL Injection 공격은 다른 공격들에 비해 웹 애플리케이션을 사용하거나 운영하는 시스템에는 위협적이지는 않지만, 공격으로 인하여 민감한 정보를 획득하고 변조할 수 있기 때문에 국방, 은행, 전자상거래와 같은 민감한 정보를 다루는 곳에서는 매우 치명적이다. 이런 위협적인 SQL Injection 공격을 탐지하고 예방하기 위해 여러 분야에서 다양한 기법들이 연구[3-18]되어 왔으며, 대표적으로 웹 프레임워크, 정적 분석 방법, 동적 분석 방법, 정적 및 동적 분석 방법, 기계학습을 이용한 방법들이 있다. 웹 프레임워크[3,4]에서는 입력 값에 대한 필터링 방법을 제공하지만, 입력된 특수문자들에 대해서만 필터링을 하기 때문에 우회 기법들이 많이 존재한다. 정적 분석 방법[5-7]은 사용자 입력 형식을 분석하기 때문에 단순 필터링 방법보다는 효과적이지만, 입력 형식에 맞는 공격은 탐지할 수 없는 단점이 있다. 동적 분석 방법[8-10]은 웹 애플리케이션 수정 없이 취약점을 찾을 수 있지만, 모든 취약점을 찾을 수 없는 단점이 있다. 정적 및 동적 분석 방법[11-14]은 정적 분석과 동적 분석의 단점들을 보완한 방법으로 SQL Injection 공격 탐지에 효과적이지만, 정적 분석과 동적 분석을 혼용하여 사용하기 때문에 복잡하다는 단점이 있다. 기계학습[17,18]을 이용한 방법은 알려지지 않는 공격에 대한 이상 탐지가 가능하다는 장점이 있지만, 오탐지(false-negative)와 과탐지(false-positive)가 발생 한다는 단점이 있다.

따라서 본 논문에서는 SQL Injection 공격을 단순한 방법으로 정확하게 탐지하고 예방할 수 있는 SQL 질의의 애트리뷰트 값을 제거한 정적 및 동적 분석 방법을 제안한다. 또한, 이 방법을 실제 웹 애플리케이션에 적용하고 그 효율성에 대해서 검증 한다.

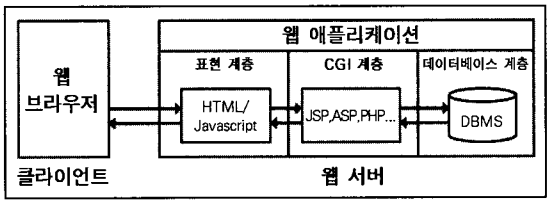
본 논문의 구성은 총 5장으로 구성된다. 2장에서는 관련연구로써 웹 애플리케이션 구조와 SQL Injection에 대해서 설명하고, SQL Injection 공격 탐지 및 예방에 관한 관련연구들을 분석하고 분류한다. 3장에서는 본 논문에서 제안하는 SQL 질의의 애트리뷰트 값을 제거한 정적 및 동적 분석 방법과 탐지 알고리즘에 대해서 설명하고, 이를 이용한 SQL Injection 공격 탐지 방법에 대해서 설명한다. 4장에서는 제안하는 방법을 이용한 실험 및 그 결과를 설명하며, 5장에서 결론을 맺는다.

Ⅱ. 관련연구

SQL Injection 공격을 이해하기 위해서 웹 애플리케이션의 구조와 작동 방법에 대해서 설명하고, 다음으로 이런 웹 애플리케이션에서 발생될 수 있는 SQL Injection 취약성과 공격에 대해서 설명한다.

2.1 웹 애플리케이션 구조

웹 애플리케이션은 웹 브라우저에서 이용할 수 있는 응용 프로그램으로 구조는 다양하게 구성될 수 있지만, 일반적으로 3단계 계층으로 구성되며 [그림 1]과 같다[11,13].



〔그림 1〕 웹 애플리케이션 구조

웹 애플리케이션은 표현 계층에서 사용자 데이터를 입력받고 Common Gateway Interface(CGI) 계층에서 데이터를 적절하게 처리하여 데이터베이스에 데이터를 저장하거나 데이터를 가지고 오며, 표현 계층에 처리 결과를 보여준다. 각 계층별 주요 기능은 다음과 같다.

① 표현 계층

사용자로부터 데이터를 입력받거나 데이터 처리 결과를 사용자에게 보여주는 그래픽 사용자 인터페이스 역할을 한다. 표현 계층은 Flash, HTML, Javascript 등으로 만들어지며, 사용자와 직접적으로 상호작용을 한다.

② CGI 계층

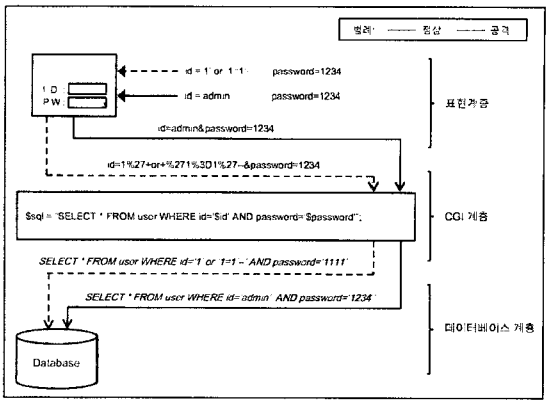
표현 계층과 데이터베이스 계층의 중간에 위치하며, CGI 또는 서버측 스트립트 프로세스(Server-Side Script Process)라고도 한다. 사용자로부터 입력받은 데이터를 데이터베이스에 저장할 수 있도록 적절하게 데이터를 변환하여 처리하며, 데이터베이스에서 데이터를 처리하여 표현 계층으로 넘겨준다. 즉, 웹 애플리케이션에서 실질적으로 데이터 처리를 하는 부분이 CGI 계층이며 JSP, PHP, ASP 등의 서버측 스크립트 언어(Server-Side Script Language)로 구성된다.

③ 데이터베이스 계층

데이터베이스 계층은 웹 애플리케이션에서 사용자 데이터 입력과 처리 결과에 대한 모든 정보를 관리하고 보관한다. 데이터베이스 계층이 웹 애플리케이션에서의 중요 정보들을 관리하고 보관하기 때문에 허가된 사용자에게 데이터를 제공하고 악의적인 접근으로부터 데이터를 안전하게 보호하는 것이 매우 중요하다.

2.2 SQL Injection 공격

SQL Injection 취약점을 이용한 공격은 표현 계층과 CGI 계층 사이에서 이루어지며, 대부분의 취약점은 웹 애플리케이션 개발 단계에서 발생된다. 정상 데이터 입력과 공격 데이터 입력에 대한 각 계층별 데이터 입출력을 자세히 살펴보면 [그림 2]와 같다.



[그림 2] 정상 및 SQL Injection 공격 데이터 흐름

위의 [그림 2]는 사용자 인증 단계를 나타낸 것이다. 정상 사용자가 ID와 Password를 입력하면 표현 계층에서는 GET 또는 POST 방식으로 CGI 계층에 데이터를 전송하고, CGI 계층에서 있는 SQL 질의 처리 구문이 데이터베이스와 연결되어 정상적으로 데이터를 처리하게 된다. 이와 달리 악의적인 공격자가 ID입력 폼에 *1' or '1=1'--*와 같이 입력하면, CGI 계층에 있는 질의 구문은 *SELECT * FROM user WHERE id='1' or '1=1'--' AND passwd='1111';*이 된다. 그러면 -- 뒤에 있는 구문은 모두 주석 처리되고, *or '1=1'* 때문에 항상 참이 되어 정상적인 인증절차 없이 로그인을 할 수 있다. 이와 같이 SQL Injection 공격은 악성 데이터 값을 이용하여 기존의 웹 애플리케이션에 고정된 SQL 질의 구문을 새로운 악의적인 SQL 질의 구문으로 만들어 비정상적으로 데이터베이스에 데이터를 요청하고 처리하는 공격 방법이다. 이런 SQL Injection 공격을 예방하기 위해서 웹 개발자들은 기본적으로 입력 데이터 값을 필터링하는 방법을 사용하고 있지만, 이를 우회하는 방법들이 많이 존재하고 있기 때문에 단순 필터링 방법으로는 SQL Injection 공격을 방지하기 어렵다. 따라서 단순한 필터링 방법보다 발전된 다른 SQL Injection 공격 탐지 및 예방 방법이 필요하다.

2.3 SQL Injection 공격 분류

SQL Injection 공격 분류는 표준화되고 일반화된 분류 기준이 없기 때문에 다양한 기준으로 분류할 수 있다. 본 논문은 W. G. Halfond[20]의 분류 방법을 인용하여 SQL Injection 공격을 분류하였다.

2.3.1 Tautologies

Tautologies 공격은 SQL 질의 구문에 악성 조건 코드를 삽입하여 SQL 질의 구문이 항상 참인 결과를 유도하는 공격이다. *SELECT * FROM user WHERE id='' or 1=1 --' AND password=''* 의 *1=1 --* 와 같이 항상 참이 되는 악성 조건 코드를 삽입하여 SQL Injection 공격을 시도하며, 이 공격은 사용자 인증과 같은 접근 제어를 우회하기 위해 사용된다.

2.3.2 Illegal/Logically Incorrect Queries

이 공격은 악성 SQL 질의 코드를 삽입하여 SQL 질의 구문 오류를 유도하고, 구문 오류 메시지로부터 SQL 질의의 형태나 구조 등의 정보를 수집한다. 이 공격은 다른 SQL Injection 공격을 수행하기 위한 사전 정보 수집 단계로 많이 이용되며, 실질적인 SQL Injection 공격은 아니다.

2.3.3 Union Queries

이 공격은 2개 이상의 SQL 질의 구문을 연결해 주는 Union 연산자를 이용한 공격으로, Union 연산자를 조합한 악성 코드를 삽입하여 SQL Injection 공격을 수행한다. 기존 SQL 질의 구문 *SELECT * FROM user WHERE id='' AND password=''* 에 *SELECT * FROM user WHERE id='' UNION SELECT * FROM member WHERE id='admin' --' AND password=''* 와 같이 UNION 연산자를 추가하면 -- 이후의 SQL 질의 구문은 주석처리 되고 2개의 SQL 질의 구문이 실행된다. 첫 번째 SQL 질의 구문에서는 NULL값을 출력하여 보여주지만, 2번째 SQL 질의 구문에서는 비정상적인 방법으로 admin 정보를 보여준다.

2.3.4 Piggy-Backed Queries

이 공격은 웹 애플리케이션 안에 있는 기존 SQL 질의 구문에 추가적인 악성 SQL 질의 구문을 삽입하는 공격이다. *SELECT * FROM user WHERE id='admin' AND password=''; drop table user -- '* 와 같이 기존 SQL 질의 구문에 *; drop table user --*을 삽입함으로써 기존의 SQL 질의 구문은 NULL값을 출력하여 보여주고, 다음의 *drop table user*가 실행되어 user 테이블이 삭제된다. 이처럼 Piggy- Backed 공격은 기존 SQL 질의 구문에 새로운 SQL 질의 구문을 추가 삽입하는 방법이기 때문에 SQL 질의 구문을 수정하여 SQL Injection 공격을 시도하는 방법과는 다르다.

2.3.5 Stored Procedures

최근 DBMS에서는 SQL 질의 구문을 함수로 만들어 DBMS에 저장하고 필요시 호출하여 사용할 수 있도록 Stored Procedures를 제공한다. 속도가 빠르고 SQL 질의 구문이 노출되지 않기 때문에 보안상 안전하다는 이유로 Stored Procedures를 많이 사용한다. 하지만, 이 역시 Piggy-Backed와 같은 일반적이고 전통적인 SQL Injection 공격에 취약하다.

2.4 SQL Injection 공격 탐지 및 예방

SQL Injection 공격을 탐지하고 예방하는 방법으로서 단순 필터링을 비롯한 여러 가지 탐지 기법들이 연구되고 있다. 본 단원에서는 이와 관련된 연구에 대해서 설명한다.

2.4.1 웹 프레임워크를 이용한 방법

최근 몇몇 웹 프레임워크에서는 SQL Injection 공격을 회피할 수 있는 다양한 기능들을 제공하고 있다. PHP는 magic quotes[3]를 제공하는데 POST, GET, COOKIES에서 오는 입력 값에서 4가지 특수문자 ', ", /, NULL이 있으면, 자동으로 \을 추가함으로써 SQL Injection공격을 방지해 준다. 하지만, 특수문자 4가지만 처리해주기 때문에 우회하는 방법이 존재하며, magic quotes 기능 설정에 따른 웹 애플리케이션 수정이 필요하다. 또한 웹 애플리케이션 개발 시 magic quotes 기능을 고려하면서 개발을 진행해야 하는 번거로움이 있다. Struts의 validator[4]는 미리 정의된 규칙(rule)에 의해서 입력 데이터 값의 유효성을 검사한다. validator는 SQL Injection 공격에 사용되는 특수문자들에 대한 규칙의 정의가 잘 되어있지 않다면 SQL Injection 공격에 취약할 수 있으며, 규칙 설정이 복잡하다는 단점이 있다.

2.4.2 정적 분석을 이용한 방법

정적 분석 방법은 웹 애플리케이션 안에 있는 SQL 질의 구문을 분석하여 SQL Injection 공격을 탐지하거나 예방하는 방법으로써 구문 분석 방법을 적용하기 위해서는 기존 웹 애플리케이션의 수정이 필요하다. 정적 분석 방법은 SQL Injection 공격 탐지를 목적으로 하는 것 보다는 사용자 입력 값 유형의 유효성을 검사하는 것으로 잠재적으로 SQL Injection 취약점을 감소시키는 역할을 한다. JDBC-Checker[5]는 Java String Analysis(JSA) 라이브러리를 이용하며, 동적으로 생성되는 사용자 입력 값 유형의 유효성을 검사하여 SQL Injection 공격을 방지한다. 하지만 공격자의 입력 값이 유형이나 문법적으로 올바르다면 이를 우회할 수 있으며, JSA 라이브러리가 Java에서만 제공된다는 단점이 있다. Wassermann[6]은 *Automated reasoning* 방법과 결합한 정적 분석 방법을 사용했다. 이 방법은 동적으로 생성되는 SQL 질의에 중복되는 SQL 질의 조건 연산자가 없다는 가정하에 SQL 질의를 검증한다. SQL 질의 연사자 중복과 관련된 공격에는 효과적이지만, 다른 형태의 SQL Injection 공격은 탐지하기 어렵다. Stephen [7]은 웹 애플리케이션에 있는 plain text SQL statement, SQL 질의, execution call을 수집하여 사용자 입력의 유형을 검증할 수 있도록 SQL 질의를 *fix generation*한다. 이 방법은 SQL Injection 공격을 탐지하고 예방하는 방법이 아닌 취약한 SQL 질의 구문을 사전에 제거함으로써 SQL Injection 공격을 예방한다. 이 방법은 java 언어로 개발된 웹 애플리케이션에만 가능하며, AST와 SQL parser Zql와 같은 특정 라이브러리가 요구된다.

2.4.3 동적 분석을 이용한 방법

동적 분석 방법은 웹 애플리케이션 외부에 SQL 질의 분석 프로세스를 이용하여 간접적으로 SQL Injection 공격 취약점을 찾는다. 간접적으로 웹 애플리케이션을 분석하기 때문에 정적분석 방법과 다르게 웹 애플리케이션 수정 없이 SQL Injection 공격 취약점을 찾을 수 있다. 오픈소스 프로그램인 Paros[8]는 SQL Injection 공격 취약점뿐만 아니라, 웹 애플리케이션의 다양한 취약점을 스캔하여 찾아준다. Paros는 미리 정의된 공격 코드를 이용하여 스캔을 시도하고, HTTP 응답을 이용하여 공격의 성공여부를 확인하기 때문에 성공여부가 완벽하게 일치하지 않는다. Sania[9]는 클라이언트와 웹 애플리케이션 사이의 정상 SQL 질의와 웹 애플리케이션과 데이터베이스 사이의 정상 SQL 질의를 수집하여, SQL Injection 공격에 취약한 부분을 찾아 공격 코드를 생성한다. 생성된 공격 코드를 이용하여 공격을 시도하고, 공격으로부터 발생된 SQL 질의를 수집한다. 이렇게 수집된 정상 SQL 질의와 공격 SQL 질의를 parse tree를 이용하여 비교 분석함으로써, 공격의 성공여부를 확인한다. parse tree를 이용하기 때문에 HTTP 응답 값을 이용하여 성공여부를 확인하는 것보다 정확도가 높다. Yonghee Shin[10]은 *Input flow Analysis*와 *Input Validation Analysis*을 통하여 white-box을 만들고, 이를 이용하여 테스트 입력 데이터를 만들어 SQL Injection 취약성을 찾는 방법을 제안하였다. 이와 같이 동적 분석 방법은 웹 애플리케이션 수정이 필요 없다는 장점이 있지만, 발견된 웹 애플리케이션 취약점을 개발자가 직접 수정해야 한다는 점과 미리 정의된 공격코드 이외의 취약점은 찾을 수 없다는 단점이 있다.

2.4.4 정적 및 동적 분석을 혼용한 방법

정적 및 동적 분석을 혼용한 방법은 웹 애플리케이션에 있는 SQL 질의를 정적으로 분석을 하고, 외부에서 발생되는 동적 SQL 질의와 비교 분석하는 방법으로 SQL Injection 공격을 탐지하고 예방한다. SQLCheck [11]는 SQL Injection 공격에 대해서 정의하고 *Context-free grammars*와 *Complier parsing techniques*를 기반으로 한 *Sound and complete algorithm*을 제안하였다. AMNESIA[12]는 웹 애플리케이션에서 SQL 질의가 실행되는 hotspots을 찾은 다음 모든 가능한 SQL 질의들을 생성한다. 생성된 정적 SQL 질의와 사용자로부터 받은 동적 SQL 질의를 JSA Library를 이용하여 SQL 질의를 분류하고 분석한다. Buehrer[13]은 웹 애플리케이션의 정적 SQL 질의와 사용자로부터 생성된 동적 SQL 질의를 parse tree를 이용하여 상호 비교 분석함으로써 SQL Injection 공격 여부를 확인한다. Wei[14]은 웹 애플리케이션에 있는 stored procedure인 정적 SQL 질의와 동적으로 생성되는 동적 SQL 질의를 *Control Flow Graph*을 만들어 비교 분석함으로써 SQL Injection 공격 여부를 탐지하는 방법을 제안하였다.

2.4.5 Instruction-Set Randomization을 이용한 방법

Instruction-Set Randomization방법은 웹 애플리케이션에서 발생되는 SQL 질의 구문에 난수 값들을 넣고 SQL 질의의 변조 여부를 확인함으로써 SQL Injection 공격여부를 확인한다. SQLrand[15]는 웹 서버와 데이터베이스 서버 사이에 Proxy를 두고 SQL 질의에 난수 값을 넣어 난수화한다. 난수 값 추측이 가능하다면 이 방법은 공격 예방에 효과적이지 않다.

2.4.6 SQL Query Profiling을 이용한 방법

Jae-Chul Park[16]은 웹 애플리케이션의 SQL 질의를 프로파일링하여, 동적으로 생성되는 동적 SQL 질의를 *Pairwise sequence alignment of amino acid code formulated* 방법을 이용하여 비교 분석함으로써 SQL Injection 공격여부를 탐지하고 예방한다. 웹 애플리케이션 수정 없이 SQL Injection 공격을 탐지할 수 있는 장점이 있지만, 웹 애플리케이션이 변경될 때마다 다시 프로파일링해야 하는 번거로움이 있다.

2.4.7 기계학습을 이용한 방법

Valeur[17]은 기계학습 방법을 이용한 침입탐지시스템을 제안하였다. 웹 애플리케이션에서 발생되는 SQL 질의를 학습하여 탐지 모델을 만들고, 실시간으로 발생되는 SQL 질의를 학습 모델과 동일한지 체크하여 SQL Injection 공격 여부를 확인한다. 이 방법은 불충분한 학습 데이터 셋을 이용할 경우 오탐지(false-negative)와 과탐지(false-positive)가 발생할 수 있다. WAVES[18]는 web crawler를 통하여 웹 애플리케이션에서 취약한 부분을 찾고 패턴 리스트와 공격 기술들을 바탕으로 공격 코드를 구성한다. 구성된 공격 코드를 이용하여 SQL Injection 공격 취약점을 찾는다. 이 방법은 기계학습 방법을 이용하기 때문에 기존의 전통적인 공격 코드 생성 방법을 개선하였지만, 기존의 취약성 침투 시험의 단점과 같이 모든 취약점을 찾아주지는 못한다.

Ⅲ. 제안하는 방법

3.1 SQL Injection 공격 탐지 알고리즘

본 논문에서 제안하는 SQL Injection 공격 탐지방법은 정적 및 동적 분석 방법을 혼용한 방법으로써 단순하게 SQL 질의의 애트리뷰트 값을 제거하여 비교 분석하는 방법이다. 모든 SQL Query의 애트리뷰트 값은 ', " 으로 둘러싸여 있거나 그냥 값으로 표현되며, 본 논문에서는 설명의 편리성을 위해 모든 애트리뷰트 값은 ' 으로 둘러싸여 있다고 표현한다. 제안 알고리즘의 설명에 필요한 기호는 [표 2]와 같으며 탐지 방법에 대해서 2.2의 내용을 바탕으로 설명하고, 다음으로 제안하는 탐지 알고리즘에 대해서 설명하겠다.

[표 2]의 기호를 2.2의 내용에 적용하면 I_t은 *admin* 과 *1234*가 되고, I_f은 *1' OR '1=1'--* 와 *1234*이 된다. 웹 애플리케이션에 고정된 *FQ*는 *SELECT * FROM user WHERE id='$id' AND password='$password'*이 되며, DQ_t는 *SELECT * FROM user WHERE id='admin' AND password='1234'*, DQ_f는 *SELECT * FROM user WHERE id='1' or '1=1'--' AND password='1111'* 이 된다.

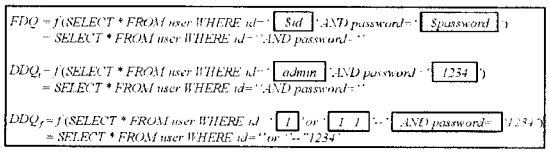
〔표 2〕 SQL Injection 탐지 알고리즘 기호

기 호	설 명
$I_{\{t,f\}}$	사용자 입력 값 {t : 정상 입력 값, f : 비정상 입력 값 }
f	SQL 질의의 애트리뷰트 값을 삭제하는 함수
FQ	웹 애플리케이션에 고정되어 있는 SQL 질의 구문
$DQ_{\{t,f\}}$	사용자 입력으로부터 동적으로 생성되는 SQL 질의 구문 {t : 정상 SQL 질의, f : 비정상 SQL 질의}
FDQ	웹 애플리케이션에 고정되어 있는 SQL 질의에서 애트리뷰트 값을 뺀 SQL 질의 구문
$DDQ_{\{t,f\}}$	사용자 입력으로부터 동적으로 생성된 SQL 질의에서 애트리뷰트 값을 뺀 SQL 질의 구문 {t : 정상 SQL 질의, f : 비정상 SQL 질의}

본 논문이 제안하는 탐지방법은 우선 수식 1과 같이 SQL 질의의 애트리뷰트 값을 삭제해주는 함수 f를 이용하여 웹 애플리케이션에 있는 정적 SQL 질의의 애트리뷰트 값을 제거하고, 동적으로 생성되는 SQL 질의도 f함수를 이용하여 애트리뷰트 값을 제거한다.

$$FDQ = f(FQ),\ DDQ = f(DQ) \quad (1)$$

애트리뷰트 값 제거함수 f는 FQ, DQ를 [그림 3]과 같이 SQL 질의의 애트리뷰트 값을 제거한다.



〔그림 3〕 애트리뷰트 값 제거함수 f

[그림 3]과 같이 애트리뷰트 값 제거함수를 사용하면 비정상 입력 데이터는 SQL 질의 구문 오류가 생기던지, FDQ와 다른 새로운 DDQ_f가 생긴다. 애트리뷰트 값을 제거한 후 수식 2와 같이 FDQ와 DDQ를 XOR하면 정상 SQL 질의인지, 공격 SQL 질의인지를 알 수 있다.

$$FDQ \oplus DDQ \begin{cases} = 0 & \text{정상} \\ \neq 0 & \text{비정상} \end{cases} \quad (2)$$

N, n : 웹 애플리케이션에 고정된 SQL 질의의 총 개수
FQ_i : 웹 애플리케이션에 고정된 SQL 질의
DQ_k : FQ_i로부터 동적으로 생성된 임의의 SQL 질의
f : SQL 질의 애트리뷰트 값 제거 함수

$FQ = \{FQ_1, ..., FQ_n\}$,
$FDQ = \{FDQ_1, ..., FDQ_n\}$,

```
1.  For i=1 to N
2.      Get FQ_i
3.      FDQ_i = f(FQ_i)
4.  End {For}
5.  While(true)
6.      Get DQ_k
7.      DDQ_k = f(DQ_k)
8.      If (FDQ⊕DDQ_k) = 0 then
9.          nonblocking
10.     Else
11.         blocking
12.     End {If}
13. End {While}
```

〔그림 4〕 SQL Injection 공격 탐지 알고리즘

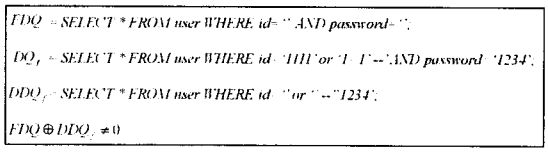
[표 2]에 있는 기호를 이용하여 웹 애플리케이션에 있는 모든 SQL 질의에 대해서 SQL Injection 공격 탐지 알고리즘을 일반화하여 표현하면 [그림 4]와 같다. 1~4줄과 같이 우선 웹 애플리케이션에 있는 FQ을 애트리뷰트 값 제거 함수 f을 이용하여 FDQ로 만든다. 그 다음 5~13줄과 같이 웹 애플리케이션에서 실시간으로 DQ_k가 발생될 때마다 함수 f을 이용하여 DDQ_f으로 만들고, FDQ와 DDQ_f을 XOR한다. XOR한 결과가 0이면 정상 SQL 질의로 판단하고, 0이 아니면 공격 SQL 질의로 판단하여 DQ_k을 차단한다.

3.2 다양한 SQL Injection 공격 탐지 분석

본 제안하는 알고리즘이 2.3에서 언급한 다양한 SQL Injection 공격에 대하여 어떻게 적용하고 탐지할 수 있는지 2.2의 사용자 인증 내용을 바탕으로 3.1 수식을 이용하여 분석한다. FQ는 *SELECT * FROM user WHERE id='$id' AND password='$password'* 이고 FDQ는 *SELECT * FROM user WHERE id='' AND password=''* 이며, 정상 SQL 질의에 대해서는 분석하지 않고 공격 SQL 질의 DQ_i와 DDQ_f에 대해서만 분석한다.

3.2.1 Tautologies 탐지 분석

Tautologies는 FQ에 항상 참이 되는 악성 코드를 삽입하여 새로운 DQ_f을 만드는 공격이다. 제안하는 탐지 알고리즘을 적용하여 Tautologies 공격을 분석하면 [그림 5]와 같다.



〔그림 5〕 Tautologies 탐지 및 분석

[그림 5]를 보면 알 수 있듯이 공격 코드 DQ_f는 참이 되는 조건 때문에 FDQ와 DDQ_f는 항상 차이를 보이며, 이 차이로 인하여 Tautologies 공격을 탐지하고 예방할 수 있다.

3.2.2 Illegal/Logically Incorrect Queries 탐지 분석

Illegal/Logically Incorrect Queries는 FQ에 구문 오류가 발생되는 DQ_f을 만드는 공격이다. 탐지 알고리즘을 적용하여 Illegal/Logically Incorrect Queries 공격을 분석하면 [그림 6]과 같다.

```
FDQ = SELECT * FROM user WHERE id='' AND password='';

DQf = SELECT * FROM user WHERE id='1111' AND password='1234'
      AND CONVERT(char, no) --';

DDQf = SELECT * FROM user WHERE id='' AND password=''
       AND CONVERT(char, no) --';
```

[그림 6] Illegal/Logically Incorrect Queries 탐지 및 분석

[그림 6]에서 DQ_f는 *CONVERT* 함수를 이용하여 형변환을 시도함으로써 오류를 발생시키는 공격이다. Illegal/Logically Incorrect Queries 공격은 구문 오류를 발생시키기 위해서 CONVERT 함수와 같은 SQL 함수나 연산자를 추가하기 때문에 FDQ와 DDQ_f는 차이를 보이며, 이를 통하여 Illegal/Logically Incorrect Queries를 탐지하고 예방할 수 있다.

3.2.3 Union Queries 탐지 분석

Union Queries는 FQ에 Union 연산자가 포함된 악성 코드를 삽입하여 새로운 DQ_f을 만드는 공격이다. 탐지 알고리즘을 적용하여 Union Queries 공격을 분석하면 [그림 7]과 같다.

```
FDQ = SELECT * FROM user WHERE id='' AND password='';

DQf = SELECT * FROM user WHERE id='1111' UNION
      SELECT * FROM member WHERE id='admin'--' AND password='1234';

DDQf = SELECT * FROM user WHERE id='' UNION
       SELECT * FROM member WHERE id=''--'1234';
```

[그림 7] Union Queries 탐지 및 분석

[그림 7]을 보면 DQ_f에는 UNION 연산자가 삽입되어 있기 때문에 FDQ와 DDQ_f는 항상 차이를 보이며, 이를 통하여 UNION Queries 공격을 탐지하고 예방할 수 있다.

3.2.4 Piggy-Backed Queries 탐지 분석

Piggy-Backed Queries는 기존 FQ에 새로운 악성 SQL 질의를 삽입하여 두 개 이상의 SQL 질의로 구성된 새로운 DQ_f을 만드는 공격이다. 탐지 알고리즘을 적용하여 Piggy-Backed Queries 공격을 분석하면 [그림 8]과 같다.

```
FDQ = SELECT * FROM user WHERE id='' AND password='';

DQf = SELECT * FROM user WHERE id='admin' AND password='1234';
      DROP TABLE member; --';

DDQf = SELECT * FROM user WHERE id='' AND password='';
       DROP TABLE member; --';
```

[그림 8] Piggy-Backed Queries 탐지 및 분석

[그림 8]을 보면 DQ_f에는 FQ와 한개 이상의 새로운 SQL 질의가 삽입되어 있기 때문에 FDQ와 DDQ_f는 항상 차이를 보이며, 이를 통하여 Piggy-Backed Queries 공격을 탐지하고 예방할 수 있다.

3.2.5 Stored Procedures 탐지 분석

Stored Procedures는 SQL 질의 구문을 함수로 만들어 DBMS에 저장하는 것이 때문에, SQL 질의 구문이 노출되지 않을 뿐 다른 SQL Injection 취약점과 동일하게 일반적이고 전통적인 SQL Injection 공격에 취약하다. 탐지 알고리즘은 정규식과 DBMS에서 제공하는 명령어 및 함수들을 이용하여 간단하게 구현할 수 있으며, 탐지 알고리즘을 적용하여 Stored Procedures 공격을 분석하면 [그림 9]와 같다.

```
FDQ = SELECT * FROM user WHERE id='' AND password='';

DQf = SELECT * FROM user WHERE id='admin' AND password='1234';
      SHUTDOWN; --';

DDQf  SELECT * FROM user WHERE id='' AND password='';
      SHUTDOWN; --';
```

[그림 9] Stored Procedures 탐지 및 분석

[그림 9]의 DQ_f는 Piggy-Backed Queries와 비슷한 유형의 공격으로 *SHUTDOWN*명령어로 DBMS를 종료시킬 수 있다. Piggy-Backed Queries와 같이 DQ_f에 추가적인 SQL 질의 연산자나 명령어가 들어가기 때문에 FDQ와 DDQ_f는 차이를 보이며, 이를 통하여 Stored Procedures 공격을 탐지하고 예방할 수 있다.

Ⅳ. 실험 및 평가

4.1 실험방법

본 논문에서 제안하는 SQL Injection 공격 탐지 알고리즘은 실제 웹 애플리케이션에 다양하게 적용시킬 수 있다. 웹 애플리케이션에 있는 FQ_i를 스캔하고 목록화하여 실시간으로 발생되는 DQ와 비교분석하거나, 웹 애플리케이션과 데이터베이스 서버 사이에서 정상 사용자로부터 생성된 DQ_i를 프로파일링하고 동적으로 생성되는 DQ와 비교분석하여 공격여부를 탐지할 수 있다. 또한, 웹 애플리케이션 개발 시 SQL 질의 구문이 있는 곳에 SQL 질의 체크 함수를 만들어 SQL 질의가 데이터베이스에 저장되기 전에 질의를 체크하여 SQL Injection 공격 여부를 확인하고 차단할 수도 있다. 본 논문에서는 단순하게 SQL 질의 체크 함수를 만들어 웹 애플리케이션에 적용했으며, 웹 애플리케이션은 웹 애플리케이션 연구에 많이 사용되고 있는 "gotocode"를 다른 연구 논문의 실험방법과 동일하게 사용하였다[9,11,12,19]. 또한, 정확한 실험 평가가 가능하도록 웹 애플리케이션의 취약점은 "Paros 3.2.13"[8]를 이용하여 스캔하고 수집하였다.

4.2 실험 결과 분석

본 논문은 동일한 환경의 웹 애플리케이션에 대해서 SQL Injection 공격 탐지율을 다른 연구 결과와 비교 분석하였으며, 공격 형태에 따른 공격 탐지 및 예방 방법들을 비교 분석하였다. 또한, 탐지 및 예방 방법들의 추가요소에 대해서도 비교 분석하였다.

4.2.1 공격 탐지율 비교 분석

SQL Injection 공격 탐지율에 대한 실험평가는 *SQLCheck*, *AMNESIA*과 동일하게 5가지 웹 애플리케이션을 이용하여 실험을 했다. 공격은 SQLCheck, AMNESIA와 달리 객관성을 위해서 "Paros 3.2.13"을 이용했으며, 공격 회수와 탐지 회수를 비교하여 탐지율을 계산하였다. [표 3]와 같이 동일 실험에서의 탐지율은 다른 연구들의 탐지율과 차이가 없다. 이는 SQL Injection 공격 탐지 및 예방 방법들이 기계학습이나 통계적 방법을 이용한 것이 아닌, 분석된 정적 SQL 질의와 동적 SQL 질의를 직접 비교 분석하여 탐지하는 방법이기 때문에 공격 실험에 대한 탐지율이 높다. 따라서 여러 SQL Injection 공격 탐지 및 예방 방법들의 효율성을 탐지율로 비교 분석하는 것은 어렵다.

4.2.2 공격 형태에 따른 공격 탐지 및 예방 방법 비교 분석

W. G. Halfond[20]은 SQL Injection 공격을 다양한 형태로 분류하고 이 분류 방법들을 이용하여 여러 SQL Injection 공격 탐지 및 예방 방법들의 효율성을 비교 분석하였다. 따라서 본 논문은 추가적으로 W. G. Halfond의 비교 분석 방법을 이용하여 다른 SQL Injection 공격 및 탐지 방법들과 효율성을 비교 분석하였으며 결과는 [표 4]와 같다.

[표 4]를 보면 *JDBC-Checker*, *Tautology-checker*, *WebSSARI*, *Java Static Tainting*은 정적 분석 방법을 이용한 것이다. 정적 분석방법은 오직 웹 애플리케이션 안에 있는 정적인 SQL 질의만 분석하기 때문에 그 분석 방법에 따라 탐지 효율성에 많은 차이를 보인다. *IDS*, *WAVES*은 기계학습을 이용한 이상탐지 방법이기 때문에 SQL Injection 공격에 대한 많은 학습이 필요하며, 학습에 따라 동일한 형태의 SQL Injection 공격도 탐지할 수 있고 못할 수 있다. 본 제안하는 알고리즘을 비롯한 *AMNESIA*, *SQLCheck*, *SQLGuard*은 정적 및 동적 분석 방법을 혼용한 방법으로 다양한 SQL

〔표 3〕 실험결과

웹 애플리케이션	제안 알고리즘		*SQLCheck*[11]		*AMNESIA*[12]	
	공격/탐지	탐지율(%)	공격/탐지	탐지율(%)	공격/탐지	탐지율(%)
Employee Directory	247/247	100	3937/3937	100	280/280	100
Events	87/87	100	3605/3605	100	260/260	100
Classifieds	319/319	100	3724/3724	100	200/200	100
Portal	288/288	100	3685/3685	100	140/140	100
Bookstore	366/366	100	3473/3473	100	182/182	100

〔표 4〕 SQL Injection 공격 형태에 따른 탐지 및 예방 방법 비교 분석

탐지/예방 방법	Tautologies	Illegal/Incorrect Queries	Union Queries	Piggy-Backed Queries	Stored Procedures	Inference	Alternate Encodings
AMNESIA[12]	●	●	●	●	×	●	●
CSSE	●	●	●	●	×	●	×
IDS	○	○	○	○	○	○	○
Java Dynamic Tainting	-	-	-	-	-	-	-
SQLCheck[11]	●	●	●	●	×	●	●
SQLGuard	●	●	●	●	×	●	●
SQLrand	●	×	●	●	×	●	×
Tautology-checker	●	×	×	×	×	×	×
Web App. Hardening	●	●	●	●	×	●	×
JDBC-Checker	-	-	-	-	-	-	-
Java Static Tainting	●	●	●	●	●	●	●
Safe Query Objects	●	●	●	●	×	●	●
Security Gateway	-	-	-	-	-	-	-
SecuriFly	-	-	-	-	-	-	-
SQL DOM	●	●	●	●	×	●	●
WAVES	○	○	○	○	○	-	○
WebSSARI	●	●	●	●	●	●	●
제안 알고리즘	●	●	●	●	●	●	●

기호 : 1. "●"은 공격에 따른 탐지 및 예방 가능.
2. "×"은 탐지 및 예장이 불가능
3. "○"은 부분적으로 탐지 및 예방이 가능.
4. "-"은 공격 탐지 및 예방과 관련 없음

Injection 공격을 탐지하고 예방한다. 이 방법은 웹 애플리케이션 안에 있는 모든 정적인 SQL 질의와 실시간 발생되는 동적인 SQL 질의를 문법적으로나 구조적으로 상호 비교 분석하기 때문에 정적 SQL 질의와 구조적으로나 문법적으로 동일하지 않는 동적 SQL 질의는 모두 탐지하고 예방한다. 하지만, *AMNESIA, SQLCheck, SQLGuard*은 정적 및 동적 SQL 질의의 구문을 분석하기 위해서 Parse tree로 변환하기 때문에 Stored Procedures 형태의 공격은 탐지하지 못하며, 시간 복잡도가 $O(n^3)$이다. 또한, DBMS마다 SQL 질의의 구조나 문법이 차이가 있기 때문에 Parse tree를 만드는 방법이 DBMS에 의존적이다. 이와 달리 본 논문이 제안하는 알고리즘은 Parse tree와 같은 복잡한 구문 분석이 요구되지 않고 매우 단순하게 애트리뷰트 값만 제거하여 비교 분석하기 때문에 시간 복잡도는 $O(1)$이며, 다양한 DBMS에 범용적으로 적용할 수 있다. 또한, Stored Procedures 형태의 공격을 포함한 다양한 SQL Injection 공격을 탐지하고 예방할 수 있다. 동적 분석 방법은 SQL Injection 공격을 탐지하고 예방하는 방법이 아닌 웹 애플리케이션의 취약성을 간접적으로 찾는 방법이기 때문에 본 비교 분석 방법에는 언급되지 않는다. 참고로, *JDBC-Checker*는 SQL질의에 대한 타입을 검사하여 SQL Injection 공격 취약점을 감소시키기 때문에 SQL Injection 공격 탐지 및 예방과는 관련이 없다.

4.2.3 공격 탐지 및 예방 방법들에 대한 추가요소 비교 분석

[표 5]는 탐지 및 예방 방법들의 추가적인 요소들을 비교 분석한 것이다[20]. 제안하는 알고리즘은 단순하게 애트리뷰트 값을 제거하는 방법이기 때문에 추가적으로 특정 라이브러리가 요구되지 않으며, SQL 질의 애트리뷰트 값 표현 방법이 DBMS 모두 동일하기 때문에 DBMS에 의존적이지 않다. 다만, 본 알고리즘을 구

〔표 5〕 탐지 및 예방 방법들에 대한 추가요소 분석

탐지/예방 방법	소스코드 수정	공격 탐지	공격 예방	필수적인 추가 요소
AMNESIA[12]	불필요	자동	자동	없음
CSSE	불필요	자동	자동	Custom PHP Interpreter
IDS	불필요	자동	리포트 생성	IDS System-Training Set
JDBC-Checker	불필요	자동	코드수정 제안	없음
Java Dynamic Tainting	불필요	자동	자동	없음
Java Static Tainting	불필요	자동	코드수정 제안	없음
Safe Query Objects	필요	해당사항 없음	자동	개발자 학습
SecuriFly	불필요	자동	자동	없음
Security Gateway	불필요	메뉴얼 명세서	자동	Proxy 필터
SQLCheck[11]	필요	부분 자동	자동	Key Management
SQLGuard	필요	부분 자동	자동	없음
SQL DOM	필요	해당사항 없음	자동	개발자 학습
SQLrand	필요	자동	자동	Proxy, 개발자 학습, Key Management
Tautology-checker	불필요	자동	코드수정 제안	없음
WAVES	불필요	자동	리포트 생성	없음
Web App. Hardening	불필요	자동	자동	Custom PHP Interpreter
WebSSARI	불필요	자동	부분 자동	없음
제안 알고리즘(SQL질의 체크함수)	필요	자동	자동	개발자 학습
제안 알고리즘(SQL질의 프로파일링)	불필요	자동	자동	Proxy
제안 알고리즘(SQL질의 목록화)	불필요	자동	자동	없음

현하고 적용하는 방법에 따라 필수적인 추가 요소가 요구된다. 프로파일링 방법을 이용할 경우 필수적인 추가 요소로 Proxy가 요구되며, SQL 질의 체크 함수를 이용할 경우 개발자 학습과 소스 코스 수정이 요구된다. SQL 질의 리스트 방법을 사용할 경우는 필수적인 추가 요소는 요구되지 않는다.

V. 결 론

본 논문에서는 웹 애플리케이션에 고정된 SQL 질의와 동적으로 생성되는 SQL 질의의 애트리뷰트 값을 제거하여 비교 분석하는 정적 및 동적 분석 방법을 혼용한 SQL Injection 공격 탐지 방법을 제안하였다. 또한, SQL Injection 공격에 취약한 웹 애플리케이션에 제안한 방법을 적용하여 실험함으로써 성능을 검증하였고, 다양한 공격 및 탐지 방법들과 비교 분석함으로써 효율성을 확인하였다. 본 제안 방법은 단순하게 SQL질의의 애트리뷰트 값만 제거하여 분석하기 때문에 DBMS에 의존적이지 않으며, 다른 탐지 및 예방 방법과 같이 특정 라이브러리를 요구하거나 Parse tree와 같은 복잡한 탐지 방법이 요구되지 않는다. 비록 본 제안 방법이 웹 애플리케이션에서 발생되는 모든 취약점에 대한 공격을 탐지하고 예방하지 못하지만, 정적 및 동적 분석 방법을 혼용하여 사용하기 때문에 웹 애플리케이션에 대한 SQL Injection 공격은 효과적으로 탐지하고 예방한다.

본 제안 방법은 웹 애플리케이션에만 국한된 것이 아닌 데이터베이스와 연결되어 있는 어느 애플리케이션에도 적용할 수 있으며, SQL 질의 프로파일링, SQL 질의 목록화, 탐지 프로그램 모듈화 등 다양한 방법으로 응용하여 활용할 수 있다. 향후, SQL Injection 공격뿐만 아니라 XSS와 같은 다양한 웹 애플리케이션 공격에 대하여 본 논문에서 제안하는 탐지 방법과 기계학습 방법들을 이용한 탐지 연구가 필요하다.

참 고 문 헌

[1] 국가사이버안전센터, "2008 국가 정보보호 백서", 2008.

[2] The Open Web Application Security Project, "OWASP TOP 10 Project", http://www.owasp.org/ .

[3] PHP, magic quotes, http://www.php.net/magic_quotes/ .

[4] Apache Struts project, Struts. http://struts.apache.org/ .

[5] C. Gould, Z. Su, P. Devanbu, "JDBC Checker : A Static Analysis Tool for SQL/JDBC Applications", *In Proceedings of the 26th International Conference on Software Engineering (ICSE)*, pp. 697-698, 2004.

[6] G Wassermann, Z. Su, "An Analysis Framework for Security in Web Applications", *In Proceedings of the FSE Workshop on Specification and Verification of Component-Based Systems(SAVCBS)*, pp. 70-78, 2004.

[7] Thomas. S, Williams. L, "Using Automated Fix Generation ot Secure SQL Statements", *In Proceeding of the 29th international Conference on Software Engineering Workshops (ICSEW. IEEE Computer Society)*, pp. 54, 2007

[8] Paros. Parosproxy.org, http://www.parosproxy.org/ .

[9] Kosuga. Y, Kernel. K, Hanaoka. M, Hishiyama. M, Takahama. Yu, "Sania : Syntactic and Semantic Analysis for Automated Testing against SQL Injection", *In Proceedings of the Computer Security Applications Conference 2007*, pp. 107-117, 2007.

[10] Yonghee Shin, "Improving the Identification of Actual Input Manipulation Vulnerabilities", *14th ACM SIGSOFT Symposium on Foundations of Software Engineering ACM*, 2006.

[11] Z. Su, G. Wassermann, "The Essence of Command Injection Attacks in Web Applications", *In Conference Record of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 372-382, 2006.

[12] Halfond W. G, Orso. A, "AMNESIA : Analysis and Monitoring for NEutralizing SQL-Injection Attacks", *In Proceedings of the 20th IEEE/ACM international Conference on Automated Software Engineering*, pp. 174-183, 2005.

[13] Buehrer. G, Weide. B. W, Sivilotti. P A, "Using Parse Tree Validation to Prevent SQL Injection Attacks", *In Proceedings of the 5th international Workshop on Software Engineering and Middleware*, pp. 105-113, 2005.

[14] Wei. K, Muthuprasanna. M, Kothari. S, "Preventing SQL injection attacks in stored procedures", *Software Engineering Conference 2006. Australian*, pp. 18-21, 2006.

[15] S. Boyd, A. Keromytis, "SQLrand : Preventing SQL injection attacks", *Applied Cryptography and Network Security LNCS*, Volume 3089, pp. 292-302, 2004.

[16] Jae-Chul Park, Bong-Nam Noh, "SQL Injection Attack Detection : Profiling of Web Application Parameter Using the Sequence Pairwise Alignment", *Information Security Applications LNCS*, Volume 4298, pp. 74-82, 2007.

[17] F. Valeur, D. Mutz, G. Vigna, "A Learning-Based Approach to the Detection of SQL Attacks", *In Proceedings of the Conference on Detection of Intrusions and Malware and Vulnerability Assessment*, pp 123-140, 2005.

[18] Huang. Y, Huang. S, Lin. T, Tasi. C, "Web application security assessment by fault injection and behavior monitoring", *In Proceedings of the 12th international Conference on World Wide Web*, pp 148-159, 2003.

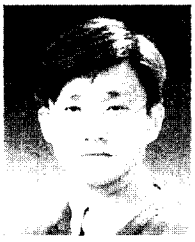
[19] GotoCode, http://www.gotocode.com/ .

[20] W. G. Halfond, J. Viegas, A. Orso, "A Classification of SQL-Injection Attacks and Countermeasures", *In proceeding on International Symposium on Secure Software Engineering Raleigh*, NC, USA, pp. 65-81, 2006.

〈著者紹介〉

이 인 용 (In-yong Lee) 정회원
2007년 2월 : 남서울대학교 컴퓨터학과 학사
2007년 3월~현재 : 고려대학교 정보경영공학전문대학원 석사과정
<관심분야> 시스템 보안, 네트워크 보안, 침입탐지

조 재 익 (Jae-ik Cho) 정회원
2005년 2월 : 동국대학교 컴퓨터학과 학사
2008년 2월 : 고려대학교 정보경영공학전문대학원 석사
2008년 3월~현재 : 고려대학교 정보경영공학전문대학원 박사과정
<관심분야> 네트워크 모델링, 패턴인식



조 규 형 (In-yong Lee) 정회원
2001년 2월 : 서울시립대학교 수학과 학사
2003년 3월 : 고려대학교 정보경영공학전문대학원 석사
2006년 8월 : 고려대학교 정보경영공학전문대학원 박사 수료
2006년 8월~현재 : 고려대학교 정보경영공학전문대학원 연구원
<관심분야> 패턴인식, 시스템 보안, 네트워크 보안

문 종 섭 (Jong-sub Moon) 정회원
1981년~1985년 : 금성 통신 연구소 연구원
1991년 : Illinois Institute of technology 졸업(전산학 박사)
1993년~현재 : 고려대학교 전자 및 정보공학부 교수
<관심분야> 생체인식, 침입탐지, 운영체제