DOI: 10.3745/KIPSTA.2008.15-A.3.181

웹 응용 프로그램의 문자열 삽입 보안 취약성 분석기 개발

안 준 선[†] · 김 영 민[††] · 조 장 우[†††]

요 약

오늘날 대부분의 웹사이트는 웹 응용 프로그램이 적절한 웹 페이지를 생성하여 전송하는 형태인 동적 웹페이지를 사용하고 있다. 이에 대하여, 취약한 웹 응용 프로그램에 악의적인 문자열을 전달하는 공격의 형태가 증가하고 있다. 본 논문에서는 대표적인 문자열 삽입 공격인 SQL 삽입(SQL Injection) 공격과 크로스 사이트 스크립팅(Cross Site Scripting, XSS) 공격에 대하여 웹 응용 프로그램내의 보안 취약성을 자동으로 찾아 주는 프로그램 정적 분석기를 개발하였다. 요약 해석을 사용한 프로그램 분석을 위하여 가능한 문자열 값을 제외 문자열들과 함께 표현하는 요약 자료 공간과 PHP 언어의 요약된 의미 규칙을 설계하였으며, 이를 기반으로 분석기를 구현하였다. 또한 개발된 분석기가 기존의 연구 결과와 비교하여 경쟁력 있는 분석 속도와 정밀도를 가짐을 실험을 통하여 보였다.

키워드 : 정적 분석, 웹 응용프로그램 보안, SQL 삽입 공격, 크로스 사이트 스크립팅 공격, 요약 해석

Development of a String Injection Vulnerability Analyzer for Web Application Programs

Joonseon Ahn[†] · Yeong-Min Kim[††] · Jang-Wu Jo[†††]

ABSTRACT

Nowadays, most web sites are developed using dynamic web pages where web pages are generated and transmitted by web application programs. Therefore, the ratio of attacks injecting malevolent strings to vulnerable web applications is increasing. In this paper, we present a static program analyzer which analyzes whether a web application program has vulnerabilities to the SQL injection attack and the cross site scripting(XSS) attack. To analyze programs using abstract interpretation framework, we designed an abstract domain which models potential string set along with excluded strings and developed an abstract interpreter for the PHP language. Also, based on them, we implemented a static analyzer. According to our experiments, our analyzer has competitive analysis speed and accuracy compared with related research results.

Keyword : Static Analysis, Web Application Security, SQL Injection Attack, Cross Site Scripting Attack, Abstract Interpretation

1. 서 론

현재 대부분의 웹 사이트 개발에 있어서 웹 응용 프로그램이 현재 상황이나 입력에 맞는 웹 페이지를 생성하여 전송하는 형태인 동적인 웹페이지가 사용되고 있다. JSP, ASP, PHP, Servlet 등의 웹 응용 프로그램은 웹 페이지에 접근하는 사용자의 입력을 받아 데이터베이스에 저장된 최신의 자료를 사용하여 요구에 맞는 웹 페이지를 생성하여 클라이언트에게 전송한다. 따라서 이러한 웹 응용 프로그램은 불가피하게 웹이라는 열린 통로를 통하여 전달된 입력에 의하여 프로그램이 수행되고 내부의 데이터에 접근하는 특성을 가지게 되며 이로 인하여 불가피하게 불특정 다수의 공격에 노출된다. 이러한 환경에서 최근 웹 사이트에 대한 공격은 전통적인 웹 서버 프로그램이나 운영체제에 대한 공격보다는 웹 응용 프로그램이 가지고 있는 취약점을 이용하는 공격이 증가하는 추세이다.[1][2]

웹 응용 프로그램의 보안을 위해서는 다음과 같은 다양한 접근법이 시도되고 있다. Scott 와 Sharp는 응용 프로그램에게 유효하지 않은 악의적인 입력을 필터링 하는 게이트웨이(gateway)의 사용을 제안하였고[3][4], 이러한 방식은 AppSheild[5]와 InterDo[6]과 같이 계층 7에서 응용 프로그램 데이터에 대한 검사를 수행하는 웹 응용 프로그램 게이트웨이 방식의 상용제품에 적용되었다. 그러나 이러한 게이트웨이 방식은 보안상의 안전성을 사용자의 세밀한 환경설

※ 이 논문은 2005년 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임(KRF-2005-003-D00338).
† 정 회 원 : 한국항공대학교 항공전자및정보통신공학부 교수
†† 정 회 원 : 삼성전자 연구원(본 연구는 한국항공대학교 재학 중에 수행되었음.)
††† 정 회 원 : 동아대학교 컴퓨터공학과 교수(교신저자)
논문접수 : 2008년 1월 14일
수 정 일 : 2008년 2월 28일
심사완료 : 2008년 3월 7일

정에 전적으로 의존한다. 또한 보안 게이트웨이는 웹 응용 프로그램의 전단부에서 해당 웹 응용 프로그램에 대한 모든 HTTP 요청을 검사하게 되므로 웹사이트의 성능이 저하되는 단점을 가지고 있다. 또 다른 방식으로, 가상의 공격을 통하여 웹 사이트의 보안 문제점을 찾아내는 방식이 있다. 이 방식은 일반적으로 블랙박스 방법(black-boxed method)을 사용하여 가상의 침투를 시도하는 방법으로 Huanget은 웹 응용 프로그램의 취약성을 식별해 내기 위해 블랙박스 검사(black-boxed testing)를 제공하는 웹 응용 프로그램 보안 평가 구조를 디자인하였다[7]. 이 방식 또한 현재 상업적으로 사용되는 방법이기는 하나, 검사를 통해서는 근본적으로 보안 취약성의 부재를 증명하지는 못하는 한계점을 가진다.

정적 분석에 기반 한 웹 응용 프로그램 분석 방법은 프로그램 내의 자료 흐름을 분석하여 이를 기반으로 외부의 입력에 의한 공격이 시스템의 안전성을 침해할 수 있는지 검사하는 방법을 사용한다. 이러한 정적 분석 방법은 실행시의 성능에 부담을 주지 않으면서 대상 공격 방법에 대한 안전성을 보장할 수 있는 장점을 가진다. Minamide는 자바 문자열 분석[8]에 관한 연구를 기반으로 PHP 프로그램 내에서 사용되는 문자열의 형태를 문맥 자유 문법(context-free grammar)으로 근사하여 분석하는 정적 분석기를 개발하였다[9]. 개발된 분석기는 PHP 내의 보안 취약성을 자동으로 검출하나, 외부 입력에 대하여 사용자가 기술해야 하는 단점을 가진다. Choi 등은 자바 프로그램에 대한 문자열 분석 연구를 발전시켜 정규 표현식(regular expression)을 분석 공간으로 사용하고 넓힘(windening) 근사 방법을 이용하여 분석의 실용성을 높이는 연구를 수행하였다[10]. 또한 Yichen Xie와 Alex Aieken은 문자열을 문자열들의 순서화된 접합(concatenation)으로 나타내고 블록 내(intra-block), 프로시저 내(intra-procedural), 프로시저 간(inter-procedural)의 3단계로 분석을 수행하는 방법을 제안하였다[11]. 그러나 프로그램 내에서 사용되는 데이터들에 대해 단지 안전한 값과 안전하지 않은 값으로 분류함으로써 문자열 조작 함수 사용 등에 따른 안전화 작업에 대하여 잘못된 경고(false alarm)를 하는 부정확성이 존재한다.

본 연구와 관련된 가장 최신의 앞선 연구는 Wassermann과 Su의 연구 결과로서 [9]의 분석 방식을 기반으로 SQL 삽입 공격에 대하여 사용자의 추가적인 개입 없이 보안 취약성을 분석하는 방식을 제안하였다[12]. 이 연구에서는 SQL 삽입 공격을 정형적으로 명세하는 구문 제한(syntactic confinement) 특성을 정의하였고, 프로그램 내에서 생성되는 문자열의 형태를 문맥 자유 문법 형태로 근사하여 구문 제한 특성을 검사함으로써 SQL 삽입 공격에 대한 취약성을 실용적인 정밀도로 분석하였다. 이러한 문자열 분석과 관련된 [9],[10],[12]의 연구는 모두 정규 표현식이나 문맥 자유 문법을 사용한 범용적인 문자열 분석에 기반하고 있어서 분석의 부담이 크며, 이에 대하여 본 연구는 문자열 삽입 보안 취약성을 고려한 분석 도메인을 사용하여 분석의 효율을 높이고자 하였다.

본 연구에서는 대표적인 웹 응용프로그램 공격인 SQL 삽입 공격과 XSS 공격에 대하여 PHP 프로그램내의 취약성을 자동으로 검출하기 위하여 웹 응용프로그램 내의 보안상 민감한 함수의 인자 값에 문자열 삽입 공격에 사용되는 위험한 문자열이 외부로부터 전달되어 포함되는지를 파악하는 안전한 자료 흐름 분석기(secure dataflow analyzer)를 요약해석(abstract interpretation)의 방법론을 사용하여 개발하였다. 개발된 분석기는 [9]와 비교하여 사용자의 추가적인 명세 작업을 필요로 하지 않으며, [11]과 비교하여 문자열 조작함수를 통한 걸러짐 작업에 대해 정확한 분석을 수행 할 수가 있다. 또한, 가장 최신의 연구인 [12]와 비교하여 문자열의 일반적인 모든 형태를 분석하는 것이 아니라, 위험 문자열의 외부로부터의 전달 여부만을 분석하게 되므로 상대적으로 분석 시간을 절감할 수 있으며, XSS 공격에 대한 취약성도 분석이 가능하다.

본 논문의 전체적인 구성은 다음과 같다. 2장에서는 배경 연구 내용으로서 본 논문의 분석기가 대상으로 하는 웹 응용 프로그램 보안 취약성을 소개한다. 3장에서는 분석기 개발의 기반이 되는 분석을 위한 요약 공간(abstract domain), PHP 언어의 특징 및 수행 규칙을 기술한다. 4장에서는 개발된 PHP 보안 취약성 분석기의 구현과 성능을 설명하며, 마지막으로 5장에서 결론을 맺는다.

2. 웹 응응 프로그램 보안 취약성

2.1 SQL 삽입 (SQL injection) 공격

SQL 삽입 공격은 사용자의 입력값으로 들어오는 신뢰할 수 없는 값들이 검증 없이 SQL 명령문을 생성하는데 사용되어질 때 발생할 수 있다. 이는 데이터베이스에 대한 질의(query)들이 동적으로 생성되는 상황에서, 사용자의 입력이 데이터베이스에서는 의미 있는 명령문으로 사용되지만, 웹 응용 프로그램에서는 단순한 문자열로 취급하는 상황으로 인하여 발생한다.

(그림 1)은 웹 응용 프로그램에서 볼 수 있는 SQL 삽입 공격 취약성의 예이다. 위 코드는 외부의 입력을 받아 데이타베이스에 대한 질의를 생성한다. 이때, 악의적인 사용자가 key값으로 아래와 같은 값을 입력으로 주면

```
'; DROP TABLE ('contents'); --
```

(그림 2)와 같은 SQL문을 $query값으로 생성하며 이러한 SQL 명령문을 실행하면 프로그램 의도와 관계없이 contents 테이블이 삭제되게 된다. 위와 같은 기법의 웹 응용 프로그램에 대한 SQL 삽입공격 취약성은 실제로 많은 웹 사이트에서 발견되고 있다.

```
$key=$_GET['key'];
$query="SELECT * FROM page
WHERE key='" . $key . "'";
mysql_query($query);
```

(그림 1) SQL 삽입공격 취약성의 예

```
SELECT * FROM page
WHERE key=' ';
DROP TABLE('contents'); --'
```

(그림 2) 생성된 SQL 명령문

2.2 XSS (Cross-Site Scripting) 공격

XSS 공격은 사용자의 입력으로부터 온 데이터가 악의적인 스크립트의 수행에 사용되어 HTTP 쿠키와 같은 정보가 유출되는 결과로 이어지는 것을 말한다.

(그림 3)은 XSS 취약성의 예제이다. $key 변수에 대한 값은 HTTP 요청으로부터 오며, 그 값은 HTML의 출력을 생성하는데 사용된다. 공격자가 이 프로그램의 XSS 공격 취약성을 이용하여 공격을 수행하기 위해서는 아래와 같은 HTTP 연결을 공격 대상자에게 전달하여 공격대상자가 접근을 수행하도록 시도하게 될 것이다.

```
http://www.reserch.com/find_key.php?
key=><script>malicious_sript();</script>
```

위와 같은 HTTP 요청을 공격 대상자가 수행하게 될 경우 (그림 3)의 PHP 코드는 (그림 4)와 같은 HTML문을 공격 대상자의 웹 브라우저에서 수행하게 되며, 이는 공격자가 임의로 작성할 수 있는 위험한 스크립트가 공격 대상자의 권한으로 수행될 수 있음을 나타낸다.

XSS의 공격 기법을 사용하면, 공격 대상자의 웹 브라우저는 공격자에 의해 준비된 악성 스크립트가 신뢰할 수 있는 사이트로부터 전송되었다고 생각하기 때문에, 해당 사이트에서 사용하는 브라우저 쿠키, 세션 토큰 등의 민감한 정보에 접근하거나 심지어 HTML 페이지의 내용을 조작하여 변경할 수도 있다.

SQL 삽입공격 이나 XSS 공격은 모두 정보 흐름의 안전성과 관련이 있다. 즉 안전하지 않은 외부로부터의 입력이 스크립트의 수행이나 SQL 명령문의 수행과 같은 민감한 작업에 사용되었을 때 웹 응용 프로그램은 취약성을 가지게 된다. 외부에서 입력된 자료값이 SQL 명령문으로 수행되기 위해서는 문자열을 구분하는 ' " ',와 같은 문자들이 입력되어 사용되어야 하며, XSS 공격에서 악성 스크립트의 삽입을 위해서는 <script>와 같은 태그 문자열이 사용되어야 한다. 따라서 이러한 문자열의 유입을 프로그램 내에서 차단함으로써 문자열 삽입 공격을 예방할 수 있으나, 많은 응용 프로그램에서 이를 간과하고 있는 실정이다. 본 연구에서는

```
$key=$_GET['key'];
echo "<a href=\"result_key
.php?keywds=$key\">";
```

(그림 3) XSS에 대한 취약성 예

```
<a href="result_key.php?keywds=>
<script>malicious_script();</script>
```

(그림 4) 생성된 HTML 출력문

문자열 값의 분석을 통하여 외부로부터 전달된 위험 문자열이 민감함 작업의 수행에 사용되는지를 분석함으로써, 취약성이 존재를 알아내고자 하였다.

3. 문자열 삽입 보안 취약성 분석의 설계

요약 해석에 기반 한 프로그램 분석에서는 실제 자료값을 안전하게 근사하는 유한한 자료 공간에서 프로그램을 수행함으로써 실제 수행 시에 생성되는 값들을 모두 포괄하는 분석 결과를 얻게 된다. 따라서 요약 해석에서는 목적으로 하는 분석결과를 최대한 정밀하게 얻으면서도 최대한 단순화된 자료 공간을 설계하게 되며, 이에 기반 하여 프로그램의 요약된 의미(abstract semantics)를 기술하게 된다. 본 장에서는 문자열 삽입 보안 취약성 분석을 위한 요약 공간과 의미 규칙을 기술한다.

3.1 요약 공간 정의

<표 1>은 분석을 위하여 설계한 라티스(lattice) 구조의 요약 공간과 조인(join) 연산을 보여준다. 설계된 요약공간은 프로그램 각 시점에서의 메모리 상태를 *Memory* 도메인의 원소로 근사하며, 메모리 위치에 저장된 값과 계산되는 값의 문자열 값 정보를 *Value* 도메인의 값으로 표현한다.

Value 공간은 프로그램 내 각 위치에서 계산되는 문자열들의 집합을 근사하기 위하여 최대 크기가 제한된(k) *String*의 멱집합을 기반으로 한다. PHP에서 일반 문자열과 정수, 논리값 등은 상호 호환되어 사용되므로 일반적인 문자열 집합을 나타내는 *String*에는 정수와 논리값을 나타내는 *Integer* 및 *Boolean*이 포함된다. $L_0(e)$는 외부 입력값을 나타내며 여기에서 제외 문자열 집합 e는 외부에서 입력된 해당 문자열들이 제거되었음을 나타낸다. PHP에서 초기화 되지 않은 모든 변수의 값은 외부의 웹페이지 접근으로부터 전달될 수 있으므로 이러한 변수들의 값은 $L_0(\varnothing)$로 초기화 된다.

S와 I는 내부에서 생성되는 안전한 문자열 집합이나 정수 집합의 크기가 한도를 넘을 경우에 이들을 포괄하여 나타내기 위하여 사용되며, 외부로부터의 입력값이 포함될 수 있을 경우에는 제외 문자열 집합 e와 함께 $S'(e)$로 근사된다. PHP 프로그램 내에서 문자열 조작 함수등을 사용하여 보안 취약성 공격에 필요한 특정 문자열을 외부 입력으로부터 제외하였을 경우 이것이 $L_0(e)$와 $S'(e)$의 e에 반영되며, 이를 통하여 잘못된 보안 취약성 경고를 줄일 수 있다.

M은 특정 시점의 메모리 상태를 나타내며 주어진 위치(*Location*)에 대해 해당위치의 값(*Value*)를 반환하는 함수 공간으로 정의된다. *Location*은 프로그램 수행 중 사용될 수 있는 메모리 위치를 근사하며 메모리 위치를 나타내는 *Loc*의 멱집합(powerset)과 정적 분석 시간에 위치를 알아낼 수 없음을 나타내는 T로 표현된다. *Loc*의 원소는 s 와 $s[s]$의 형태를 가지며 프로그램의 변수(예, $x) 이름이나 배열(예, $x[$key])이름과 첨자(index) 문자열을 나타낸다. $s[T]$는 정

〈표 1〉 요약 해석을 위한 정보 공간 정의

• $i \in Integer$ • $b \in Boolean$	1)
• $s \in String$ ($Integer \cup Boolean \subset String$)	
• $e \in Exclusion = \mathbb{P}^k$ ($String$)	2)
• $V \in Value = \mathbb{P}^k$ ($Integer \cup String \cup Boolean \cup \{L_0(e)\}$) $\cup \{ I, S, S'(e) \}$	3)
• $\ell \in Loc = s \mid s[s] \mid s[T]$	4)
• $L \in Location = \mathbb{P}(Loc) \cup \{ T \}$	5)
• $M \in Memory : Location \rightarrow Value$	6)
$M[\ T \rightarrow V\ ](L) = M(L) \sqcup V$	6-i)
$M[\ L \rightarrow V\ ](T) = M(T) \sqcup V$	6-ii)
$M[\ \{s_1\} \rightarrow V\ ](\{s_2\}) = V$ if $s_1 = s_2$	6-iii)
$M[\ \{s_1[s_2]\} \rightarrow V\ ](\{s_1'[s_2']\}) = V$ if $s_1 = s_1'$, $s_2 = s_2$	6-iv)
$M[\ \{s_1[s_2]\} \rightarrow V\ ](\{s_1'[T]\}) = M[\{s_1'[T]\}) \sqcup V$ if $s_1 = s_1'$	6-v)
$M[\ \{s_1[T]\} \rightarrow V\ ](\{s_1'[s_2]\}) = M[\{s_1'[s_2]\}) \sqcup V$ if $s_1 = s_1'$	6-vi)
$M[\ \{s_1[T]\} \rightarrow V\ ](\{s_1'[T]\}) = M[\{s_1'[T]\}) \sqcup V$ if $s_1 = s_1'$	6-vii)
$M[\ \{\ell\} \rightarrow V\ ](\{\ell'\}) = M(\{\ell'\})$	6-viii)
$M[\ \{\ell\} \rightarrow V\ ](L) = \sqcup \{ M[\ \{\ell\} \rightarrow V\ ](\{\ell'\}) \mid \ell' \in L \}$	6-ix)
$M[\ L \rightarrow V\ ](L') = \sqcup \{ M[\ \{\ell\} \rightarrow V\ ](L') \mid \ell \in L \}$	6-x)
• 조인 연산 : $\sqcup$	7)
- $S'(e) \sqcup V = S'(e \cap e1)$ if $V = S'(e1)$	7-i)
$S'(e \cap e_1 \cap \ldots \cap e_n)$ if $V = \{L_0(e1), \ldots, L_0(en)\}$	
$S'(e)$ if $V \in \mathbb{P}(Integer \cup String \cup Boolean) \cup \{ I, S, \}$	
- $S \sqcup V = S'(e1 \cap \ldots \cap en)$ if $V \in \mathbb{P}(Integer \cup String \cup Boolean \cup \{L_0(e1), \ldots, L_0(en)\})$ & $V \ni L_0(e1), \ldots, L_0(en)$	7-ii)
S otherwise	
- $I \sqcup V = S'(e1 \cap \ldots \cap en)$ if $V \in \mathbb{P}(Integer \cup String \cup Boolean \cup \{L_0(e1), \ldots, L_0(en)\})$ & $V \ni L_0(e1), \ldots, L_0(en)$	7-iii)
I if $V \subset Integer$ or $V = I$	
S otherwise	
- $V_1 \sqcup V_2 = \lceil V_1 \cup V_2 \rceil_K^{L_0}$	7-iv)
where $\lceil V \rceil_K^{L_0} = V$ if $\mid V \mid \leq K$	
$S'(e1 \cap \ldots \cap en)$ else if $L_0(e1), \ldots, L_0(en) \in V$	
I else if $V \subset Integer$	
S otherwise	
- $M_1 \sqcup M_2\ (x) = M_1\ (x) \sqcup M_2\ (x)$	7-v)

적 분석 시간에 배열의 첨자를 결정할 수 없는 경우에 사용된다. $M[\ L \rightarrow V\ ]$는 메모리 M에 대하여 L에 속하는 한 위치를 V가 나타내는 값으로 갱신함으로써 얻어지는 모든 메모리 상태들을 포괄하여 나타낸다.

메모리로부터 주어진 위치에 대한 값을 얻어내는 규칙은 6-i~6-x에서와 같이 정의되며 위로부터 우선순위를 갖는다. $M[\ T \rightarrow V\ ]$나 $M[\ \{s_1[T]\} \rightarrow V\ ]$와 같이 불확실한 메모리 위치의 갱신에 대하여 주어진 위치의 값을 읽어오는 경우에는 갱신 이전의 메모리에 대한 값과 해당 위치가 갱신되었을 경우의 값을 조인하여 얻어진다. $M[\ L \rightarrow V\ ](T)$ 형태나 $M[\ \{s_1[s_2]\} \rightarrow V\ ](\{s_1'[T]\})$와 같이 불확실한 메모리 위치의 값을 읽어오는 경우에도 관련될 수 있는 모든 메모리 위치에 저장된 값을 조인하는 방법을 사용하여 안전한 근사치를 얻어낸다. 6-ix는 메모리 위치의 집합에 대한 값을 읽어오는 경우를 나타내며 이 경우 포함된 모든 위치에 대한 값을 조인해 주게 된다. 6-x는 메모리 위치의 집합에 대한 갱신이 있는 경우를 나타내며, 이는 각각의 위치 원소에 대한 갱신에 의하여 생성되는 메모리에 대한 결과를 모두 조인하여 결과를 생성한다.

7)의 i)~iii)과 vi)는 각각 *Value*와 *Memory*공간에 대한 조인 연산을 정의하고 있으며 i)~iii)의 규칙은 순서대로 우선순위를 갖는다. 생성되는 *Value*의 원소의 개수가 정해진 한도(k)를 넘을 경우에 S', S 또는 I로 근사함으로써 공간의 크기를 유한하게 제한할 수 있다. 메모리 공간의 경우 정의 구역인 위치(*Location*) 공간이 유한하므로 역시 유한한 공간이 된다.

i)은 $S'(e)$가 포함된 경우를 나타낸다. 똑같은 형태의 원소와 조인되는 경우 제외 문자열을 교집합 하여 결과를 생성하며 따라서 *Value*공간의 최상위 원소는 $S'(\varnothing)$가 된다. $S'(e)$과 L_0형태를 포함하는 값 집합의 조인의 경우에는 모든 제외 문자열 집합의 교집합을 사용하는 S'형태가 된다. 그 외의 경우에는 $S'(e)$가 외부에서 전달되지 않는 모든 값을 포괄하므로 $S'(e)$ 자신이 된다.

ii)는 S 원소가 포함된 조인 관계를 나타낸다. 외부 입력을 포함한 원소와의 조인에 대해서는 모든 제외 문자열 집합을 교집합한 문자열 집합을 제외 문자열 집합으로 갖는 S' 형태를 결과로 생성하여 외부 입력의 영향을 반영한다. 그 외의 경우에는 S가 된다.

iii)은 외부 입력이 포함되지 않은 정수 문자열 집합의 최상위 원소인 I와의 조인 연산을 정의한다. ii)에서와 마찬가지로 L_0를 포함한 값 집합과의 조인 대해서는 모든 L_0의 e 집합을 교집합한 문자열 집합을 원소로 갖는 S'을 결과로 생성한다. 정수 집합과의 조인은 역시 I가 되며, 그 외의 경우에 대해서는 V에 정수가 아닌 문자열이 포함된 경우이므로 S를 결과로 생성한다.

iv)는 S', S, I를 포함하지 않는 그 외의 값 집합들(V_1, V_2)에 대한 조인 연산을 정의한다. 기본적으로 합집합 연산을 사용하며 합집합의 원소의 수가 한도 k를 초과하는 경우에는 분석의 종료를 보장하기 위하여 S', S, I 등을 결과로 생성하게 된다. L_0를 포함하고 있으면 합집합 내의 모든 L_0의 제외 문자열을 교집합한 문자열 집합을 원소로 갖는 S'을 조인 결과로 생성하며, 합집합의 결과가 정수 집합의 부분 집합이면 I를 결과로 생성하게 된다. 그 외의 경우에는 S를 조인 연산의 결과로 생성하게 된다.

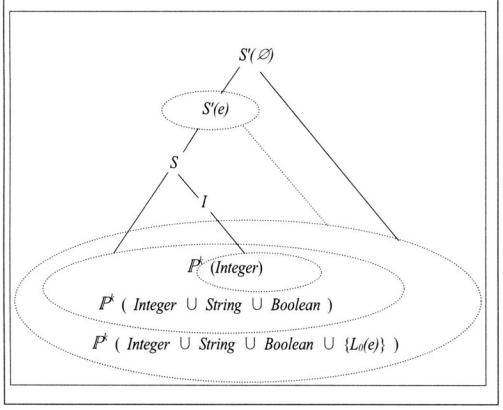
v)는 메모리 상태의 조인 연산으로, 조인의 결과로 생성되는 메모리 상태는 주어진 위치에 대하여 조인되는 각각의 메모리의 저장값을 조인한 값을 결과로 돌려준다.

<표 2>는 요약 공간의 값에 대한 실제적인 값(concrete value)으로의 함수와 요약 공간 값들 간의 포함 관계를 보여준다. $S'(e)$의 형태는 외부 입력 문자열을 포함할 수 있는 모든 문자열 집합에서 외부에서 전달된 e를 뺀 것이 되므로, e의 포함 관계에 따라 $S'(\varnothing)$를 최상위 원소로 하는 하나의 라티스 구조를 이루며, 내부 문자열들의 집합인 S의 상위 원소가 된다. PHP에서 정수와 불(boolean) 값은 문자

열로 사용될 수 있으므로 불 집합과 정수 집합은 문자열 집합의 부분집합으로 취급된다. S와 I 하위에 각각 최대 크기가 유한한(k) 문자열과 정수의 멱집합 라티스가 위치하게 된다. $S'(e)$와 $\mathbb{P}^k(Integer \cup String \cup Boolean \cup \{L_0(e)\})$ 원소들의 실제화 값 사이에는 포함관계가 항상 성립하는 것은 아니며(점선으로 표시) 제외되는 외부 문자열 집합에 따라 포함 관계가 결정된다.

〈표 2〉 요약 공간의 값에 대한 실제화 함수(C)와 요약 공간의 값들 간의 포함 관계

- $C(I)$ = Integer (Integer : 임의의 정수값의 집합)
- $C(S)$ = $String^{in}$ (모든 내부 생성 문자열 값들의 집합)
- $C(S'(e))$ = $String/C(e)$ (모든 외부 입력 및 내부 생성 문자열 집합에서 $C(e)$에 포함된 외부 입력 문자열들을 제거한 집합)
- $C(\{v_1, \ldots, v_n\}) = \{C(v_1), \ldots, C(v_n)\}$
  - $C(i) = \{i\}$
  - $C(b) = \{b\}$
  - $C(s) = \{s\}$
- $C(L_0(e))$ = $String^{ex}/C(e)$ (모든 외부 입력 문자열 집합에서 $C(e)$에 포함된 외부 입력 문자열들을 제거한 집합)



3.2 PHP 프로그램 분석을 위한 의미 정의

3.2.1 PHP 언어의 특징

PHP 언어는 일반적인 프로그래밍언어와는 다르게 자료 흐름 분석에 있어 다음과 같은 특징 때문에 일반적인 자료 흐름 분석(conventional data flow analysis)의 적용이 어렵다.

- 가변 변수(variable variables) : PHP는 문자열 자료값이 변수 명으로 사용될 수 있어 변수의 문자열 값이 자료 흐름에 영향을 미친다. (그림 5)는 가변변수의 예로서 가변 변수는 문자열 변수값을 취해서 변수명으로 사용하며 이 경우 `$foo`의 값을 변수명으로 하는 변수 `$bar`의 값에 `'baz'`가 저장된다.
- 배열의 첨자(index)가 문자열 데이터를 자유롭게 가질 수 있으므로, 배열에 의한 자료 흐름 분석을 위해서는 첨자로 사용되는 문자열 값에 대한 정확한 분석이 필요하다.
- 변수의 타입이 정해져 있지 않아 다양한 타입의 값을 동적으로 가질 수 있다. 따라서 문자열과 숫자가 상호 호환되어 사용되는 경우가 빈번하며, 이러한 경우에 대한 고려가 분석의 정확도를 높일 수 있다.
- PHP의 자료형에서 문자열은 정적문자열(static string) 또는 동적 문자열(dynamic string)로 나눌 수 있는데 이는 작은따옴표(single quotes, ' '),와 큰따옴표(double quotes, " ")로 구분된다. 동적 문자열은 정적 문자열과 달리 문자열 내의 변수는 해당 값이 추출된다. 예를 들어 (그림 6)의 실행결과는 아래와 같다.

```
hi, Guido
hi, $name
```

본 연구에서 사용한 요약 해석 분석 방법은 프로그램을 요약된 공간에서 수행하여 분석을 수행하므로 이러한 어려움을 효과적으로 극복할 수 있는 장점을 가진다.

```
$foo = 'bar';
$$foo = 'baz'; //$bar = 'baz';
```

(그림 5) 가변변수(Variable Variables)의 예

```
$name = "Guido";
echo "hi, $name\n";
echo 'hi, $name';
```

(그림 6) 정적문자열과 동적문자열의 예

2.2. 의미(semantics) 규칙

PHP 문법에 대한 요약된 의미를 수행기반 의미구조(operational semantics)의 형태로 정의하였으며 그 중 일부를 <표 3>에 기술하였다. 프로그램 수행 규칙인 $Eval$은 $Eval : Expr \times Memory \rightarrow V \times Memory$의 타입을 가지며, PHP 문장 또는 식과 수행 전의 메모리 상태를 받아 계산 결과값과 계산 후의 메모리 상태를 생성한다.

변수 $x에 대한 연산은 변수의 이름을 나타내는 문자열 값인 x로 표현되는 주소에 대하여 주어진 메모리에 저장된 값을 읽어낸다. 또한 $x[$e$]의 배열에 대한 연산은 PHP에서 배열의 첨자로 문자열을 가질 수 있으므로 우선 첨자 e 연산을 수행하고 e 연산의 결과 값 V_1을 위치 공간 값으로 변환하여, 배열 이름 x와 계산된 첨자 위치를 가지고 메모리 M_1에서 값(V_2)을 읽어낸다.

$x = e는 대입문(assignment)으로 수행 전의 메모리 상태에서 e의 값을 계산하고, 변수 $x의 이름이 나타내는 위치의 값을 계산 결과 V_1의 값으로 갱신한 메모리 M_1을 생성한다.

$e_2 = e_1$은 가변 변수에 대한 연산으로 위의 대입문에서 e_2에 대한 계산이 추가된 것이다. PHP에서는 문자열 자료값을 다시 변수명으로 취하는 경우가 가능하므로 문자열 자료값을 다시 메모리 위치값으로 계산하는 과정이 필요하다.

즉, e_2에 대한 연산이 수행된 후 e_2 연산 후의 메모리 상태 M_2에서 e_2의 계산 결과(V_2)가 나타내는 메모리 위치의 값을 V_1의 값으로 갱신한 새로운 메모리를 M_2와 조인하여 결과 메모리 상태로 생성한다. $\mathcal{L}$은 $Value \rightarrow Location$의 타입을 가지며 값을 받아 S', S, I의 원소이면 T를 반환하며 그 외의 경우에는 그 값을 문자열로 캐스팅한 문자열의 집합을 반환한다.

$x[e_2] = e_1$은 배열 표현식에 대한 연산으로 첨자(index)에 대한 연산이 추가된다. 즉, 첨자에 대한(e_2) 연산 결과값(V_2)을 위치 공간 값으로 계산하고 배열 이름 x와 위에서 계산된 첨자 위치를 가지고표현한 메모리 위치를 V_1값으로 갱신한 메모리 상태를 생성한다.

e_1+e_2는 기본 연산을 나타내는데, 먼저 e_1을 계산하고 e_2를 계산하므로 e_1 계산 후 메모리 상태 M_1이 e_2의 실행 전 메모리 상태가 되며, 기본 연산은 메모리 상태를 변화시키지 않으므로 e_2 실행 후의 메모리 상태가 전체 식의 실행 후 메모리 상태가 되고 전체 식의 수행결과는 V_1과 V_2에 해당 연산자의 의미를 적용하여 생성된 값으로 한다.

echo e_1, e_2, ⋯, e_n은 각각의 부분식 e_1, e_2, ⋯, e_n들을 순서대로 계산하므로 이전 표현식 계산 후 메모리 상태가 다음 표현식의 수행 전 메모리 상태가 되며 echo문은 단지 출력을 수행하므로 전체식의 계산결과는 없다.

if문은 e_1의 계산 결과에 따라 e_2 또는 e_3를 계산하므로 e_1의 실행 후의 메모리 상태 M_1이 e_2와 e_3의 실행 전 메모리 상태가 되며, e_1을 계산한 결과가 *true* 또는 *false*인 경우에 따라 (M_2, V_2) 또는 (M_3, V_3)를 결과로 생성한다. 이때 e_1의 계산 결과를 결정할 수 없는 경우, 요약된 의미는 모든 가능한 경우를 포함하여야 하므로 e_2와 e_3의 계산 후의 메모리 상태 M_2, M_3와 계산 후의 값 V_2, V_3를 각각 조인 연산함으로써 안전(sound)한 결과를 생성한다.

while(e_1) e_2는 정적 분석에서의 모든 가능한 경우를 포함하기 위해 조건식 e_1의 계산 결과로 생성된 메모리 상태 M_1에서 e_2를 수행하여 M_2를 생성하며, 반복문(loop)의 순환에 의해 생기는 접점에서의 메모리 상태 M'은 M과 M_2의 조인 연산의 결과로 나타낼 수 있다. 이때, 접점에서의 메모리 공간의 분석 결과가 더 이상 증가하지 않거나, 또는 e_1의 계산 결과가 거짓(false)인 경우에는 전체 식의 계산 후 메모리 상태는 M_1이 되며 결과값은 V_1로 나타낼 수 있고, 그렇지 않은 경우에는 새로운 메모리 상태 M'에 대해 재귀적으로 정의된다.

str_replace와 같은 표준 라이브러리 함수의 호출에 대해서는 함수의 코드를 접근할 수 없으므로 많이 사용되는 라이브러리 함수의 요약된 의미를 직접 구현하여 해당 함수 호출시 적용되도록 하였다. 예를 들어, str_replace 함수의 요약 공간 의미 규칙은 유한 집합의 일반 문자열 원소인 경우에는 각 원소에 대하여 해당 문자를 대치하게 되며, $S'(e)$, $L_0(e)$의 경우 제외 문자 부분에 대치되는 문자를 추가하게 된다. S, I의 경우에는 내부의 안전한 값은 취약성 분석에 고려되지 않으므로 같은 값을 돌려주게 되고 이는 분석의 안전성에 영향을 미치지 않는다. 정의되지 않은 라이브러리 함수의 경우에는 최상위 요약값을 결과로 돌려주도록 하였다. 또한 사용자 정의 함수에 대해서는 함수 호출의 문맥을 고려한 분석(context-sensitive analysis)을 수행하여 각각의 함수 호출문에 대하여 메모리 상태를 따로 저장하고, 함수 호출 시 해당 위치에서의 이전 호출의 상태와 현재 상태를 조인 연산을 수행하여 실행 전 메모리 상태로 사용하도록 하였다. 또한, 요약 수행 중 함수 호출 시의 상태가 해당 위치의 이전 호출 상태에 포함될 경우에는 이전의 호출 결과를 사용하게 된다.

〈표 3〉 PHP 문법에 대한 실행 규칙

- $Eval\ (M,\ \$x)$ = let $V_1 = M(\{x\})$ in $(M,\ V_1)$

- $Eval\ (M,\ \$x[e])$ = let $(M_1,\ V_1) = Eval\ (M,\ e)$, $V_2 = M_1[\{x\ [\ \mathcal{L}(V_1)\ ]\}\ ]$ in $(M_1,\ V_2)$

- $Eval\ (M,\ \$x = e)$ = let $(M_1,\ V_1) = Eval\ (M,\ e)$ in $(M_1[\{x\} \rightarrow V_1],\ V_1)$

- $Eval\ (M,\ \$e_2 = e_1)$ = let $(M_1,\ V_1) = Eval\ (M,\ e_1)$, $(M_2,\ V_2) = Eval\ (M_1,\ e_2)$ in $(M_2 \sqcup (M_2[\mathcal{L}(V_2) \rightarrow V_1\ ]),\ V_1)$

- $Eval\ (M,\ e_1 + e_2)$ = let $(M_1,\ V_1) = Eval\ (M,\ e_1)$, $(M_2,\ V_2) = Eval\ (M_1,\ e_2)$ in $(M_2,\ V_1 \oplus V_2)$

- $Eval\ (M,\ \$x[e_2\ ] = e_1)$ = let $(M_1,\ V_1) = Eval\ (M,\ e_1)$, $(M_2,\ V_2) = Eval\ (M_1,\ e_2)$ in $(M_2[\{x[\mathcal{L}(V_2\ )]\} \rightarrow V_1\ ],\ V_1\ )$

- $Eval\ (M,$ echo $e_1,\ e_2,\ \ldots\ e_n\ )$ = let $(M_1,\ V_1) = Eval\ (M,\ e_1)$, $(M_2,\ V_2) = Eval\ (M_1,\ e_2)$, ⋮, $(Mn,\ Vn) = Eval\ (Mn_{-1},\ en)$ in $(Mn,\ \varnothing)$

- $Eval\ (M,$ if $(e_1)\ e_2,\ e_3)$ = let $(M_1,\ V_1) = Eval\ (M,\ e_1)$, $(M_2,\ V_2) = Eval\ (M_1,\ e_2)$, $(M_3,\ V_3) = Eval\ (M_1,\ e_3)$ in if $V_1 == \{true\}$ then $(M_2,\ V_2)$ else if $V_1 == \{false\}$ then $(M_3,\ V_3)$ else $(M_2 \sqcup M_3,\ V_2 \sqcup V_3)$

- $Eval\ (M,$ while$(e_1)\ e_2)$ = let $(M_1,\ V_1) = Eval\ (M,\ e_1)$, $(M_2,\ V_2) = Eval\ (M_1,\ e_2)$, $M' = M_2 \sqcup M$ in if $M' == M$ or $V_1 == false$ then $(M_1,\ V_1)$ else $Eval\ (M',$ while$(e_1)\ e_2)$

- $\mathcal{L} : Value \rightarrow Location$ $\quad \mathcal{L}(v) = \begin{cases} T & \text{if } v \in \{ S'(e),\ S,\ I \} \\ \{valueToString(v) \mid v \in V\} & \text{otherwise} \end{cases}$

- $V_1 \oplus V_2 = \begin{cases} \lceil \{ i+j \mid j \in V_1,\ j \in V_2 \} \rceil^{L0}_{k} & \text{if } V_1,\ V_2 \sqsubset I \text{ and } V_1,\ V_2 \neq I \\ I & \text{otherwise} \end{cases}$

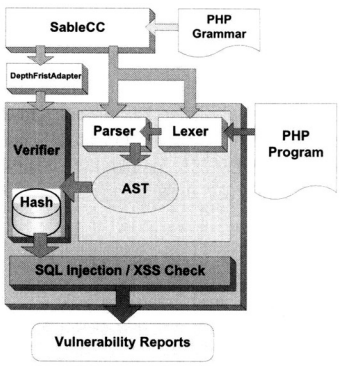
4. PHP 문자열 삽입 보안취약성 정적 분석기 구현

4.1 구현 구조

3장에서 기술한 요약 공간과 의미 규칙을 기반으로 PHP 언어의 부분집합에 대한 보안 취약성 분석기를 구현하였다. (그림 7)은 구현된 PHP 보안 취약성 분석기의 전체적인 수행 구조를 나타낸다. PHP 언어에 대한 전단부와 요약된 의미를 구현하기 위하여 프로그램 정적 분석기 생성 도구인 SableCC를 사용하였다[13]. SableCC는 PHP 문법을 입력으로 받아 PHP 구문 분석기와 어휘 분석기를 자동으로 생성

하며, 분석기에서 핵심이 되는 분석(Verifier) 모듈을 SableCC의 트리 방문 클래스를 상속하여 구현할 수 있다.

분석 대상이 되는 PHP 프로그램으로부터 생성되는 AST를 깊이 우선 순회(depth-first traversal)로 방문하면서 각각의 AST 노드에 대해 앞장에서 제시한 의미 규칙의 *Eval* 함수를 실행하고, 그 시점의 메모리 상태와 결과값을 해쉬 테이블에 각 노드별로 저장하는 과정을 반복한다. AST의 모든 노드들에 대한 메모리 상태와 결과가 더 이상 변화하지 않으면 문자열 분석이 완료하게 된다. 모든 부분식들에 대하여 분석된 결과값을 검사하여 외부에서 입력한 안전하지 않은 값이 `echo`문과 같은 외부 출력 함수의 인자로 사용되거나, `mysql_query`문 등에서 데이터베이스 쿼리의 생성에 사용될 경우 각각 XSS 공격이나 SQL 삽입 공격에 대한 취약성이 있는 것으로 판단하게 된다.



(그림 7) 웹 응용프로그램 보안 취약성 분석기

4.2 실행 결과

구현된 PHP 보안 취약성 정적 분석기를 사용하여 실제로 사용하는 응용프로그램의 PHP 소스에 대한 보안 취약성 분석을 수행하였다. 분석시에 분석공간을 유한하게 근사하기 위한 집합값의 최대 크기는 20으로 사용하였다. <표 4>는 그 수행 결과를 나타낸다. 처음 네 개의 파일은 Eve Activity Tracker 1.0에 포함된 것이며, 마지막 check_user_id.php 프로그램은 제로보드 4.1pl4 버전에 포함된 것이다. 오류 정보의 갯수는 [12]의 결과와 일치하며, 조사 결과 찾아낸 취약성은 모두 잘못된 분석(false positive)이 아닌 실제 존재하는 보안 취약성인 것으로 파악되었다.

구현된 보안 취약성 분석기를 사용하여 [12]에서 실험 대상으로 한 Eve Activity Tracker 1.0 프로그램 전체(905라인)를 2GHz의 프로세서와 1GB의 주기억장치를 가진 Window XP 시스템에서 분석한 결과 0.27초가 소요되었다. [12]의 경우 3GHz 프로세서와 8GB의 주기억장치를 가진 Linux 시스템에서 같은 프로그램을 분석하는데 0.46초가 소요된 것으로 볼 때 수행 속도에 있어 경쟁력이 있는 것으로 판단된다. 또한 구현된 분석기는 XSS 취약성에 대한 분석도 추가적으로 함께 수행하는 장점을 가진다.

<표 4> PHP 프로그램에 대한 보안 취약성 분석

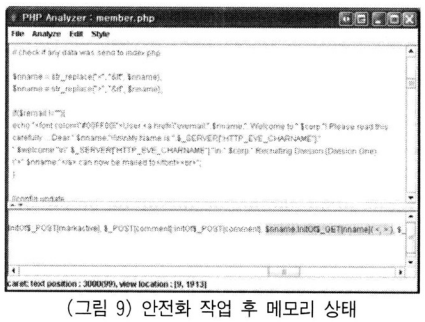
파일명	라인수	SQL삽입 취약성	XSS 취약성
edit.php	84	1	0
user.php	140	1	0
member.php	430	2	1
member_config.php	25	0	1
check_user_id.php	19	2	2

4.3 취약성의 검출과 수정

(그림 8)은 EVE Activity Tracker(1.0)의 member.php 파일에 존재하는 XSS 및 SQL 삽입공격 보안 취약성 부분을 검출하는 화면이다. 개발된 분석기 GUI는 프로그램 내의 보안 취약성 위치와 종류 및 위험한 입력의 통로가 되는 변수를 알려주며, 특정 명령문 수행 후의 메모리 상태를 보여주는 기능을 가지고 있다. 98번째 줄에는 XSS 공격 취약성이 존재하며, 108째 줄과 사용자 정의 함수 `get_trust()` 안의 코드인 388번째 줄에는 SQL 삽입공격 보안 취약성이 존재한다.

member.php파일에 존재하는 XSS 보안 취약성을 제거하기 위해 PHP의 `str_replace` 함수를 사용하여 '<'와'>'를 각각 '`&lt`'와 '`&lt`' 문자로 대치하는 코드를 추가하였다. 또한 108번째줄과 388번째 줄에 존재하는 SQL 삽입공격 취약성 을 제거하기 위해 이에 대한 안전화 작업 코드를 추가하였다. 이러한 작업을 수행하면 분석기는 오류가 없음을 보고 하게 된다. (그림 9)는 안전화 작업 후 98번째 줄의 메모리 상태를 보여주는 화면이다. `$nname : InitOf$_GET[nname](<,>)`로 표현된 부분은 변수 `$nname`이 외부에서 입력되는 '<'문자와'>'문자가 제거된 `$_GET[nname]`의 초기값을 가짐을 나타낸다. 즉, `$nname` 변수는 `$_GET`을 통해서 외부

(그림 8) 웹 응용프로그램 보안 취약성 분석기



(그림 9) 안전화 작업 후 메모리 상태

로부터 그 값이 오기는 하나 str_replace함수를 통해서 '<'와'>'를 포함하지 않음을 알 수 있으며 따라서 분석 결과가 오류의 제거를 반영하게 된다. 이와 같이 제외 문자 집합을 사용하여 효과적으로 잘못된 경고를 줄일 수 있게 된다.

5. 결 론

본 논문에서는 웹 응용 프로그램의 취약성을 찾아내는 정적 분석기를 개발하였다. 개발된 분석기는 PHP 프로그램을 입력으로 받아 프로그램 내에 존재하는 SQL 삽입공격이나 XSS 공격에 대한 취약성을 자동으로 검출한다.

취약성을 검출하기 위해서는 프로그램 내의 자료의 흐름을 파악하여 외부의 검증되지 않은 입력이 프로그램 내에서 어떻게 사용되는지를 분석하여야 한다. 이를 위하여 요약 해석에 기반한 안전한 문자열 자료 흐름 분석기를 개발하였다. 개발한 요약 해석은 문자열의 형태를 근사하는 요약된 자료 공간 내에서 프로그램을 실행하여 프로그램 내의 모든 부분식에서 생성되는 문자열의 형태를 적절하게 분석할 수 있다. 또한 요약 공간을 설계함에 있어 외부의 검증되지 않은 입력을 통하여 위험한 문자열이 민감한 작업에 사용되는지를 효과적으로 나타내도록 함으로써 분석의 정확도를 높일 수 있었다.

향후 연구로서, 개발된 분석기가 PHP 언어의 전체 구조와 라이브러리를 지원하도록 확장하여 다양한 프로그램에 대한 추가적인 실험을 수행함으로써 분석기의 정확도와 성능을 높이는 작업을 수행할 것이다.

참 고 문 헌

[1] Curphey, M., Endler, D., Hau, W., Taylor, S., Smith, T., Russell, A., McKenna, G., Parke, R., McLaughlin, K., Tranter, N., Klien, A., Groves, D., By-Gad, I., Huseby, S., Eizner, M., McNamara, R. "A Guide to Building Secure Web Applications," The Open Web Application Security Project, v.1.1.1, Sep., 2002.

[2] D. Turner, S. Entwisle, "Symantec Internet Security Threat Report Vol.IX - Trends for July 05-December 05," Symantec, March, 2006.

[3] Scott, D., Sharp, R. "Abstracting Application-Level Web Security," Proc. 11th Int'l Conf. World Wide Web (WWW2002), pp.396-407, May, 17-22, 2002.

[4] Scott, D., Sharp, R. "Developing Secure Web Applications," IEEE internet Computing, Vol.6, No.6, pp.38-45, Nov., 2002.

[5] Sanctum Inc. "AppShield 4.0 Whitepaper," http://www.sanctuminc. com, 2002.

[6] Kavado, Inc. "InterDo Version 3.0," Kavado Whitepaper, 2003.

[7] Huang, Y. W., Huang, S. K., Lin, T. P., Tsai, C. H. "Web Application Security Assessment by Fault Injection and Behavior Monitoring," In Proc. 12th International World Wide Web Conference (WWW2003), pp.148-159, May, 21-25, 2003.

[8] Aske Simon Christensen, Anders Møller, and Michael I. Schwartzbach, "Precise Analysis of String Expressions," 14th International Static Analysis Symposium (SAS '03), pp.1-18, June, 2003.

[9] Minamide, Y., "Static Approximation of Dynamically Generated Web Pages," 14th International World Wide Web Conference(WWW 2005), pp.432-441, May, 2005.

[10] Tae-Hyoung Choi, Oukseh Lee, Hyunha Kim, Kyung-Goo Doh, "A Practical String Analyzer by the Widening Approach," Asian Symposium on Programming Languages and Systems, pp.374-388, November, 2006.

[11] Yichen, X., Alex, A., "Static Detection of Security Vulnerabilities in Scripting Languages," 15th USENIX Security Symposium, 2006.

[12] Wassermann, Zhendong Su., "Sound and Precise Analysis of Web Applications for Injection Vulnerabilities," In Proceedings of PLDI 2007, pp.32-41, San Diego, CA, June, 10-13, 2007.

[13] Gagnon, E. M., Hendren, L. J., "SableCC, an Object-Oriented Compiler Framework," In Proc. 1998 Conf. Technology of Object-Oriented Languages and Systems (TOOLS-98), pp.140-154, Santa Barbara, California, Aug., 3-7, 1998.

안 준 선

e-mail : jsahn@kau.ac.kr

1992년 서울대학교 계산통계학과(이학사)

1994년 KAIST 전산학과(공학석사)

2000년 KAIST 전자전산학과(공학박사(전산학))

2000년~2001년 KAIST 프로그램분석 시스템연구단 연구원

2001년~현 재 한국항공대학교 항공전자 및 정보통신공학부 교수

관심분야 : 프로그램 분석, 프로그램 보안, 유비쿼터스 컴퓨팅 등

김 영 민

e-mail : ymkim44@kau.ac.kr

2006년 한국항공대학교 정보통신과(공학사)

2008년 한국항공대학교 정보통신과(공학석사)

2008년~현재 삼성전자 연구원

관심분야 : 통신공학, 프로그램 분석, 웹 보안

조 장 우

e-mail : jwjo@dau.ac.kr

1992년 서울대학교 계산통계학과(이학사)

1994년 서울대학교 전산과학과(이학석사)

2003년 한국과학기술원 전산학과(공학박사)

현 재 동아대학교 컴퓨터공학과 부교수

관심분야 : 프로그램 분석, 임베디드 시스템