情報保護學會論文誌
第17卷 第6號, 2007. 12

웹 응용프로그램의 삽입취약점 탐지를 위한 문자열분석*

최 태 형[1†], 김 정 준[2], 도 경 구[1‡]
[1]한양대학교 컴퓨터공학과, [2]넥스원퓨처(주)

String analysis for detection of injection flaw in Web applications

Tae-Hyoung Choi[1†], Jungjoon Kim[2], Kyung-Goo Doh[1‡]
[1]Department of Computer Science and Engineering, Hanyang University
[2]NEX1 FUTURE Co., Ltd

요 약

삽입취약점은 웹 응용프로그램에 공격자가 악성코드를 정상적인 입력 값 대신 넣어 시스템에 피해를 입힐 수 있는 대표적인 취약점이다. 삽입공격에서 안전한 애플리케이션은 외부에서 들어오는 입력 값에 들어있을 수 있는 악성문자를 여과하도록 작성해야 한다. 특정 문자의 여과 여부는 주요지점에서 문자열 변수에 특정 문자가 포함될 수 있는지를 검사하여 정적으로 알아낼 수 있다. 본 논문에서는 조건식의 의미를 분석에 적용하는 향상된 방식으로 응용프로그램의 삽입취약점을 정적으로 판정하는 방법을 제안한다.

ABSTRACT

One common type of web-application vulnerabilities is injection flaw, where an attacker exploits faulty application code instead of normal input. In order to be free from injection flaw, an application program should be written in such a way that every potentially bad input character is filtered out. This paper proposes a precise analysis that statically checks whether or not an input string variable may have the given set of characters at hotspot. The precision is accomplished by taking the semantics of condition into account in the analysis.

Keywords : Static analysis, web application vulnerability

Ⅰ. 서 론

삽입취약점(injection flaw)은 공격자가 정상적인 입력 값 대신 공격코드를 삽입하여 비공개 정보를 빼내거나, 시스템을 조작 또는 파괴할 수 있는 응용프로그램 취약점이다. 삽입공격에서 안전한 프로그램은 외부 입력 값이 프로그래머가 의도하는 값인지를 검증하도록 구현되는데, 이를 입력 값 검증(input validation)이라 한다.

[그림 1]의 프로그램은 실행 중에 SQL질의를 만들어 데이터베이스관리서버(DBMS)로 보내고, DBMS가 처리한 결과를 받는 웹 응용프로그램이다. DBMS에 보낼 SQL질의는 미리 정의된 틀과 사용자가 입력한 문자열이 결합하여 완성된다. 만약 공격자가 문자열

접수일: 2007년 7월 11일; 채택일: 2007년 10월 16일
* 본 연구는 한국과학재단 특정기초연구(R01-2006-000-10926-0) 지원으로 수행되었음.
† 주저자, thchoi@pllab.hanyang.ac.kr
‡ 교신저자, doh@hanyang.ac.kr

```
$owner    = $_REQUEST['owner'];
$itemName = $_REQUEST['itemName'];
...
$result = $DB->query("SELECT * FROM items "
                     ."WHERE owner = '"
                     .$owner
                     ."' AND itemName = '"
                     .$itemName."'");
```

〔그림 1〕 SQL질의를 사용하는 PHP프로그램

"hacker"와 "name'; DELETE FROM items; --"을 각각 변수 $owner와 $item의 값이 되도록 입력한다면, DBMS로 전달될 SQL질의는 "SELECT * FROM items WHERE owner= 'hacker' AND itemName='name'; DELETE FROM items; --'"이 되고, 이 질의는 프로그래머의 의도와는 상관없이 데이터베이스의 items테이블을 삭제한다. 이러한 삽입공격이 가능한 이유는 변수 $item으로 들어온 문자열에 특수문자 ''가 포함되어 미리 정의된 SQL질의틀이 의도하는 구조를 변경시켜서 추가적인 SQL질의를 삽입할 수 있게 되었기 때문이다.

[그림 2]는 [그림 1] 프로그램에서 입력 값을 검증하여 삽입취약점을 제거한 프로그램이다. 이 프로그램은 입력 문자열이 SQL질의틀과 결합하기 전에 입력 문자열이 알파벳과 숫자로만 구성된 문자열인지를 검증하며, 검증된 문자열만이 SQL질의틀과 결합한다. 문자열이 검증되지 않은 경우 프로그램은 종료한다. 알파벳과 숫자로만 구성된 문자열은 SQL질의틀과 결합했을 때 그 구조를 변경시키지 않으므로, 이 프로그램은 삽입공격에서 안전하다.

본 논문은 프로그램의 주요지점(예: SQL질의가 실행되는 부분)에 도달하는 입력 값에 대하여, 그 값을 구성할 수 있는 문자들의 집합을 정적으로 분석하여 삽입취

```
$owner    = $_REQUEST['owner'];
$itemName = $_REQUEST['itemName'];
...
if (!eregi('^[0-9a-z]+$', $owner) ||
    !eregi('^[0-9a-z]+$', $itemName)) {
  exit;
}
$result = $DB->query("SELECT * FROM items "
                     ."WHERE owner = '"
                     .$owner
                     ."' AND itemName = '"
                     .$itemName."'");
```

〔그림 2〕 입력값 검증으로 삽입취약점을 제거한 PHP프로그램

$$
\begin{aligned}
e &::= c \mid s \mid x \mid e + e \mid f(e) \\
b &::= true \mid false \mid e = e \mid \neg b \mid b \wedge b \mid b \vee b \mid p(e) \\
S &::= x := e \mid S; S \mid if\ b\ S\ S \mid while\ b\ S \mid nop \\
P &::= read\ x; S; write\ e \mid read\ x; P
\end{aligned}
$$

〔그림 3〕 대상언어의 추상구문

약점 존재여부를 판정하는 방법을 제안한다.

앞에서 살펴 본 대로, 입력문자열에 ''와 같은 특정한 문자가 포함되어 주요지점에 도달 가능할 때 그 응용프로그램은 삽입취약점이 있다. 삽입취약점 존재여부 판정에 특정 문자의 포함여부를 기준으로 하는 경우, 프로그램의 문자열은 그것을 구성하는 문자들의 집합으로 요약하는 것으로도 충분하다. 따라서 본 분석은 문자집합을 요약 값으로 한다.

프로그램은 실행상태에 따라 특정한 경로로 진행하며 실행되지만, 정적분석은 특정한 경로를 결정할 수 없기 때문에 모든 경로에서 얻을 수 있는 결과를 포괄한다. 따라서 정적분석의 결과에는 실행하면서 나올 수 없는 거짓양성 값이 포함될 수 있어서 분석의 정확도가 떨어진다. 본 논문에서는 주어진 요약실행상태를 조건식이 참 또는 거짓이 되게 하는 두 개의 요약실행상태로 제약하고, 이를 분석에 반영하여 정확도를 높이는 방법을 제안한다. 기존의 취약성검사방법[5][6][7][9][10]은 특정 입력 값 검증 함수 호출 여부만 판정할 수 있는데 비해서, 이 논문에서 제안하는 새로운 방법은 추가적으로 검증 코드의 직접적인 분석이 가능하다. 본 논문의 분석은 요약해석(abstract interpretation) 프레임워크[3][4][8]를 사용하였다.

Ⅱ. 분석의 정의

본 논문에서는 문자열만 처리하도록 축소된 언어를 대상으로 정적분석을 설명한다. 대상언어의 추상구문은 [그림 3]과 같다. c는 문자 상수, s는 문자열 상수, x는 문자 또는 문자열 변수를 나타낸다. 문자열연산 +는 문자열 붙이기(concatenation)이며, 문자열처리함수 f와 문자열검사함수 p는 라이브러리 함수이다. S는 문장이며, 프로그램 P는 입력 문자열을 받아서 처리한 후 출력하는 프로그램을 모델링한다.

본 분석에서는 문자열을 그 자신을 구성하는 문자들의 집합으로 요약한다. 예를 들어, 문자열 "ab", "aaba", "abbaabb" 등은 모두 자신을 구성하는 문자들의 집합

$$s^\sharp \in \mathsf{String}^\sharp = \mathcal{P}(\Sigma)_\perp$$

$$\alpha : \mathcal{P}(\mathsf{String}) \rightarrow \mathsf{String}^\sharp$$
$$\alpha(\varnothing) = \perp$$
$$\alpha(T) = \{c \mid s \in T,\ c\text{는 } s\text{에 포함된 문자}\}$$

$$\gamma : \mathsf{String}^\sharp \rightarrow \mathcal{P}(\mathsf{String})$$
$$\gamma(\perp) = \varnothing$$
$$\gamma(\varnothing) = \{\epsilon\}$$
$$\gamma(s^\sharp) = (s^\sharp)*$$

$$\alpha(T) \subseteq s^\sharp \text{ iff } T \subseteq \gamma(s^\sharp)$$

〔그림 4〕 문자열의 요약도메인과 요약화 및 구체화 함수정의

{'a', 'b'}로 요약된다. 요약도메인은 [그림 4]와 같이 정의된다. 요약실행상태(abstract state)는 실행 시 각 변수가 저장하고 있는 요약 값을 나타낸다.

문자열붙이기 *e+e*의 경우, 하위 계산식의 요약해석 결과들을 합집합 하는 것으로 요약해석 한다. 문자열처리함수는 각 함수별로 미리 정의해 둔다. 예를 들어, 주어진 문자열에서 특수문자 ''를 제거하는 함수 *remove_quote*는 인자로 주어진 계산식을 요약해석 한 결과에서 ''를 제거한 값을 결과로 내어주는 것으로 정의한다.

본 논문에서는 보다 정확한 분석결과를 얻기 위하여 조건문의 불계산식을 경로에 따라 다르게 계산하여 적용한다. 예를 들어, 조건문 *if x=y then skip else* z:=z+x에 대하여 문자변수 *x*와 *y*의 요약 값이 각각 {'b', 'c', '#'}, {'#'}, 그리고 문자열변수 *z*의 요약 값이 {'a', 'b'}인 요약실행상태가 주어졌을 때, 본 분석은 다음과 같이 결과를 내준다. 먼저, 불계산식 *x=y*가 참이 되는 *x*의 요약 값은 {'#'}일 경우이나, 이 경우 z의 요약 값은 변하지 않는다. 반면에 *x=y*가 거짓이 되는 *x*값은 {'b', 'c'}이며, 이를 통해 계산된 *z*의 값은 {'a', 'b', 'c'}이다. 결과적으로 *z*의 최종 요약 값은 {'a', 'b'}∪{'a', 'b', 'c'}={'a', 'b', 'c'}가 된다. 이 방법은 조건문의 불계산식을 고려하지 않고 계산했을 때의 결과인 {'a', 'b', 'c', '#'}보다 정확한 결과를 얻을 수 있게 한다. 각각의 불계산식에 대한 요약해석 방법은 다음과 같다.

불계산식 *x=c*의 경우, *x*의 요약 값이 *c*만으로 구성된 경우 참이 되는 경로만 실행되고, *c*가 포함되어 있지 않을 경우 거짓이 되는 경로만 실행된다고 확언할 수 있다. 나머지 경우, 참 경로에는 *x*의 요약 값이 *c*만으로 구성되어 있다고 제한하고, 거짓 경로에는 *x*의 요약 값에서 *c*를 제외한 값으로 제한한다. 예를 들어 조건식 x='b'에 대해 *x*값이 {'a', 'b', 'd'}로 주어졌을 때, 참 경로에서 *x*값은 {'b'}이고 거짓 경로에서 *x*값은 {'a', 'd'}이다. 다른 등식의 경우, 조건을 분석하여 추가적으로 알 수 있는 정보가 별로 없으므로, 주어진 요약실행상태는 그대로 전달된다.

복합 불계산식 $b_1 \vee b_2$의 경우, 참 경로에서는 하위 계산식의 결과를 합집합 하고 거짓 경로에서는 하위계산식의 결과를 교집합 한다. 예를 들어 복합 불계산식 (x='a'∨x='1')에 대하여 *x*의 값으로 {'a', 'b', 'c', '1', '2'}가 주어졌다면 다음과 같이 요약해석 한다. 먼저 하위 계산식들을 계산하는데, 계산식 x='a'에서 얻을 수 있는 *x*값은 참 경로에서 {'a'}, 거짓 경로에서 {'b', 'c', '1', '2'}이며, 계산식 x='1'에서 의 *x*값은 참 경로에서 {'1'}, 거짓 경로에서 {'a', 'b', 'c', '2'}이다. 참 경로에서는 하위계산식의 결과들을 합집합 하므로 *x*값은 {'a', '1'}, 거짓 경로에서는 하위계산식의 결과들을 교집합 하므로 *x*값은 {'b', 'c', '2'}로 제약된다. 복합 불 계산식 $b_1 \wedge b_2$는 반대로 참 경로에서는 하위 계산식의 결과를 교집합 하고, 거짓경로에서는 합집합 하여 요약실행상태를 제약한다.

계산식 ¬*b*의 경우, *b*에 대해 반영하여 얻은 요약실행상태를 서로 맞바꾼다. 예를 들어 ¬(x='b')에 대해 *x* 값이 {'a', 'b', 'd'}로 주어졌을 때, 참 경로에서 *x*값은 *b*가 아니므로, x='b'의 거짓 경로와 같이 *x*값은 {'a', 'd'}이지만, 거짓 경로에서 *x*값은 {'b'}이다.

문자열검사함수에 대한 요약해석은 문자열처리함수 마찬가지로 각각 함수에 대해 미리 정의해 둔다. 예를 들어 주어진 문자열이 숫자로만 구성되었는지를 검사하는 함수 *is_numeric(e)*는 다음과 같이 정의할 수 있다. 먼저 함수의 인자로 변수 *x*와 그 값으로 *s#*이 주어졌을 때, 참 경로에서는 숫자로 구성된 값만이 허용되므로 *x* 값은 *s#*∩{'0', '1', ..., '9'}로 구할 수 있다. 반면에 거짓 경로에서는 더 정확한 정보를 알 수 없으므로 *s#*이 그대로 전달된다. 변수 이외의 다른 계산식에 대해서는 다음 두 경우를 제외하고 더 얻을 수 있는 정보가 없다. 주어진 인자의 값을 s#이라 할 때, ① *s#*∩{'0', '1', ..., '9'}=∅이면 참 경로는 실행되지 않고, ② *s#*⊆{'0', '1', ..., '9'}이면 거짓 경로는 실행되지 않는다.

입력문 *read(x)*는 사용자 또는 외부시스템에서 입력 값을 받아 변수 x에 저장하는데, 입력 값은 정적으로 알 수 없는 값이므로 그 요약 값은 전체문자집합 Σ이다. 출력문 *write(e)*는 응용프로그램에서 주요 지점

(hotspot)을 나타내며, 출력함수의 인자 e를 요약해석하고, 그 결과를 안전한 문자집합 또는 위해한 문자집합과 비교하여 삽입취약점 여부를 검사한다. 응용프로그램의 분석 결과가 안전한 문자집합의 부분집합이거나 분석 결과와 위해문자 집합 사이에 공통원소가 없다면, 즉, 위해한 문자가 출력될 가능성이 없다면, 응용프로그램은 삽입공격에서 안전한 것으로 판정하다. 반대로, 응용프로그램의 분석 결과가 안전한 문자집합의 부분집합이 아니거나 분석 결과와 위해문자 집합 사이의 공통원소가 존재한다면, 응용프로그램에는 삽입취약점이 존재하는 것으로 판정한다.

Ⅲ. 분석 예제

[그림 5]는 본 분석을 사용할 예제 프로그램으로 [그림 2]를 분석 대상언어로 바꾼 것이다. 함수 isalphanumeric은 주어진 문자열이 알파벳과 숫자로만 구성된 문자열인지를 검사하는 함수이다. 따라서 isalphanumeric(x)에 의해 얻게 되는 참 경로에서 x값은 주어진 x값과 {'A', ..., 'Z', 'a', ..., 'z', '0', ..., '9'}의 교집합이며, 거짓 경로에서 x의 값은 주어진 x의 값이 그대로 넘어가는데, "abc%123"나 "ab d"와 같은 경우도 거짓이 되기 때문이다. read(x)는 x의 값을 전체 문자집합 Σ로 초기화한다. 3-4번 줄은 변수 owner와 item을 두 가지 경우 즉, 두 변수의 값이 모두 알파벳과 숫자로만 구성된 경우와 그 외의 경우로 실행 경로를 나누어 분석한다. 조건식이 거짓이 될 때 변수 owner와 item의 값은 { 'A', ..., 'Z', 'a', ..., 'z', '0', ..., '9' }가 된다. 조건식은 참이 될 때 변수 owner와 item의 값은 초기값 Σ가 유지된다. 그러나 5-6번 줄에서 두 변수의 값은 공집합으로 바뀌게 된다. 따라서 7번 줄에서 얻게 되는 분석결과는 알파벳과 숫자의 집합이며, 이는 안전한 문자집합의 부분집합이 되므로 본 논문의 방법은 예제 프로그램 1을 삽입취약점이 없는 것으로 판정한다.

[그림 6]은 입력 값에서 특수문자 "'"를 제거하는 프로그램이다. 문자열처리함수 head(e)는 e의 계산결과에서 첫 문자를 내주며, tail(e)은 e의 계산결과에서 첫 문자를 제외한 나머지 문자열을 내준다. 두 함수의 요약해석 결과는 주어진 인자의 요약해석 결과와 같다. 본 논문의 방법으로 이 프로그램을 분석한 결과, buf 변수의 요약 값은 Σ-{"'"}이고, 여기에는 위해문자집합에 속한 "'"가 절대로 존재하지 않는다. 따라서 예제 프로그램 2도 삽입취약점이 없는 프로그램으로 판정된다.

Ⅳ. 결 론

본 논문에서는 웹 응용프로그램에서 입력 값 검증처리 여부를 정적으로 분석하여 삽입취약점의 존재여부를 검사하는 방법을 제안하였다. 이 방법은 요약해석 프레임워크를 사용하였고, 조건을 반영하여 분석함으로써 최대한 정확한 결과를 얻었다.

기존의 문자열분석들이 프로그램의 정적문자열을 추적하여 특정위치에서 얻을 수 있는 문자열을 정규표현식[1][2]이나 문맥자유문법[7]으로 요약한다. 반면에, 본 논문의 문자열분석은 프로그램의 특정위치에 전달되는 프로그램의 입력 값인 동적문자열을 추적하여 문자집합으로 요약한다. 문자집합은 정규표현식이나 문맥자유문법에 비해 가지고 있는 정보는 적지만 본 분석의 목적인 삽입취약점 탐지에는 충분하며, 정규표현식이나 문맥자유문법으로 요약했을 때 보다 차지하는 공간도 적다. 또한 기존 취약점분석[5][6][7][9][10]은 할 수 없었던, [그림 6]의 프로그램처럼 문자비교 등으로 직접 작

```
1  read(owner);
2  read(item);
3  if !(isalphanumeric(owner) &&
4       isalphanumeric(item)) {
5    owner := "";
6    item  := "";
7  } else nop;
8  write("SELECT * FROM items WHERE owner = '"
9        + owner + "' AND itemName = '" + item
10       + "'");
```

〔그림 5〕 예제 프로그램 1

```
1  read(x);
2  buf := "";
3  while !(x="") {
4    c := head x;
5    if !(c = "'")
6      buf := buf + c;
7    else nop;
8    x := tail x;
9  }
10 write(buf);
```

〔그림 6〕 예제 프로그램 2

성한 여과코드를 분석할 수 있다. 향후 과제는 PHP, ASP, JSP 등의 실제 언어에 대해 구현하여 실험하는 것이다.

참고문헌

[1] Tae-Hyoung Choi, Oukseh Lee, Hyunha Kim, and Kyung-Goo Doh. "A practical string analyzer by the widening approach". In Naoki Kobayashi, editor, *Proceedings of the Asian Symposium on Programming Languages and Systems,* volume 4279 of Lecture Notes in Computer Science, pp. 374-388. Springer, 2006.

[2] Aske Simon Christensen, Anders Mller, and Michael I. Schwartzbach. "Precise analysis of string expressions". In *Proceedings of the International Static Analysis Symposium*, volume 2694 of Lecture Notes in Computer Science, pp. 1-18. Springer-Verlag, June 2003.

[3] Patrick Cousot and Radhia Cousot. "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints". *In Proceedings of the ACM Symposium on Principles of Programming Languages*, pp. 238-252, January 1977.

[4] Patrick Cousot and Radhia Cousot. "Abstract interpretation frameworks". *Journal of Logic and Computation*, 2(4):511-547, 1992.

[5] N. Jovanovic, C. Krugel, and E. Kirda. Pixy: "A static analysis tool for detecting web application vulnerabilities (short paper)". In *ACM SIGPLAN Workshop on Programming Languages and Analysis for Security*, Ottowa, Canada, June 2006.

[6] M. Martin, B. Livshits, and M. S. Lam. "Finding application errors and security flaws using PQL: a program query language". In *OOPLSA '05: Proceedings of the 20th annual ACM SIGPLAN conference on Object oriented programming systems languages and applications*, p. 365-383, 2005.

[7] Yasuhiko Minamide. "Static approximation of dynamically generated web pages". In *Proceedings of the International World Wide Web Conference Committee*, pp. 432-441, 2005.

[8] Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer-Verlag, 1999.

[9] Gary Wassermann and Zhendong Su. "Sound and precise anlysis of web applications for injection vulnerabilities". In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, June 2007.

[10] Y. Xie and A. Aiken. "Static detection of security vulnerabilities in scripting languages". In *Proceedings of the 15th USENIX Security Symposium*, pp. 179-192, July 2006.