소스코드를 이용한 웹 응용 취약점 분석에 관한 연구

김 성열* 정 수은* 박중길** 김상천** 한광택**

*건국대학교 인터넷미디어공학부, **국가보안기술연구소

A Study on the Security of Web Application by Source Code Analysis

Sung-Ryul Kim* Soo Eun Chung*

Jung-Gil Park** Sang-Chun Kim** Kwang-Taek Han**

*Div. of Internet and Media Konkuk Univ.

**NSRI

요 약

기존의 고정적 웹 페이지에, 실시간적으로 변화하는 내용의 제공을 가능하게 하기 위해, 추가적으로 코드를 첨가할 수 있도록 만든 것이 웹 응용 프로그램이다. 그 예로는 cgi, php, jsp, java, python 등이 있다. 많은 수의 언어와 다수의 프로그램들이 빠른 속도로 개발됨에 따라 많은 수의 보안 문제점들이 발생하였고 실제로 대단히 많은 서버들이 침입의 대상이 되었다. 웹 응용 프로그램의 보안에 많은 문제점이 발생한 이유는 첫 번째, 기존의 서버 응용 프로그램들에 비하여 웹 응용 프로그램은 훨씬 많은 수가 아주 빠르게 개발되었다는 점이다. 두 번째는 웹 응용 프로그램에서 발생한 새로운 종류의 보안 위험성을 들 수 있다. 기존의 서버 응용 프로그램에서 발생하는 위험성들은 서버 프로그램의 버그를 이용한 것이었고, 이들은 외부 입력의 내용보다는 그 크기 등의 간단히 검사 가능한 특징에 의존하는 경우가 많았다. 하지만, 웹 응용 프로그램이 외부 입력의 내용을 코드의 일부로 사용하는 경우가 많음으로 인해서, 웹 응용 프로그램에서는 간단히 검사하기 어려운 특징인 입력의 내용에 의존하는 위험성들이 많이 발생한다. 본 논문에서는 이러한 새로운 방식의 위험성을 소스코드를 이용해서 어떻게 자동적으로 검사할 수 있을지에 관해서 새로운 아이디어를 제시한다. 이 아이디어는 현재 구현 중에 있으며, 초기 실험 결과 기존의 검사 프로그램들이 찾아내지 못하는 취약점들을 찾아낼 수 있음이 확인되었다.

I. 서론

웹 응용 프로그램은 HTTP 프로토콜을 이용하여 사용자나 다른 시스템에 서비스를 제공하는 클라이언트/서버 응용 프로그램으로 정의될 수 있다. 초기의 웹에서는 웹 페이지들의 내용이 고정적이었으므로 다양한 내용을 제공하기 위해서는 다수의 페이지를 서버에 고정적인 형태로 제공하여야 하였다. 이러한 방법으로는 사용자와 대화하면서 실시간적으로 변화하는 내용의 제공은 불가능하였다. 이러한 단점을 극복하기 위해 기존의 고정적 웹 페이지에 해당하는 데이터에 추가적으로 코드를 첨가할 수 있도록 만든 것이 웹 응용 프로그램이다.

가장 간단하고 제일 처음 만들어진 웹 응용 프로그램은 common gateway interface(CGI)라고 불린다. CGI는 그 구성이 간단하기 때문에 기능은 매우 일반적이어서 임의의 프로그램이 사용가능하고 그 프로그램이 할 수 있는 작업의 성격에도 제

한이 없다. 하지만 매번의 요청에 대해서 독립적인 프로그램을 수행하여야 하는 문제 때문에 성능상이나 보안상에 문제점이 발생할 수 있다

그러한 단점들을 없애고 웹 응용 프로그램을 개발하는 것을 용이하게 만들기 위해서 많은 수의 언어가 개발되었다. 그 예로는 php, jsp, java, python 등이 있다. 이들 새로운 방식의 웹 응용은 웹 서버의 일부로 동작하여 성능과 보안상의 문제를 줄였다. 또한, 웹 응용 프로그램을 위한 전용의 인터페이스를 정의하여 편리한 구현이 가능하도록 만들었다.

초기에 CGI로 만들어진 웹 응용 프로그램의 경우에도 많은 보안 문제점이 존재하였다. [1, 2] 그리고, 이후에 많은 수의 언어와 다수의 프로그램들이 빠른 속도로 개발됨에 따라 많은 수의 보안 문제점들이 발생하였고 실제로 대단히 많은 서버들이 침입의 대상이 되었다.

웹 응용 프로그램의 보안에 많은 문제점이 발생한 이유는 두 가지로 살펴볼 수 있다. 첫 번째로는 기존의 서버 응용 프로그램들에 비하여 웹 응용 프로그램은 훨씬 많은 수가 아주 빠르게 개발되었다는 점이다. 이 때문에 다수의 사이트들이 보안에 전혀 주의를 기울이지 않은 웹 응용 프로그램을 가지고 서비스를 수행하였다. 두 번째는 웹 응용 프로그램에서 발생한 새로운 종류의 보안 위험성을 들 수 있다. 기존의 서버 응용 프로그램에서는 잘 발생하지 않았던 종류의 위험성이 발생한 이유는 웹 응용 프로그램의 특성상 사용자의 입력을 코드의 일부로 사용하는 경우가 많기 때문이다.

기존의 서버 응용 프로그램에서 발생하는 위험성들은 서버 프로그램의 버그를 이용한 것이었고, 이들은 외부 입력의 내용보다는 그 크기 등의 간단히 검사 가능한 특징에 의존하는 경우가 많았다. [4, 5, 6, 7] 하지만, 웹 응용 프로그램이 외부 입력의 내용을 코드의 일부로 사용하는 경우가 많음으로 인해서, 웹 응용 프로그램에서는 간단히 검사하기 어려운 특징인 입력의 내용에 의존하는 위험성들이 많이 발생한다.

이 논문에서는 이러한 새로운 방식의 위험성을 소스코드를 이용해서 어떻게 자동적으로 검사할 수 있을지에 관해서 새로운 아이디어를 제시한다. 이 아이디어는 현재 구현 중에 있으며, 초기 실험 결과 기존의 검사 프로그램들이 찾아내지 못하는 취약점들을 찾아낼 수 있음이 확인되었다.

II. 웹 응용 프로그램에서 발생하는 취약점

웹 응용 프로그램에서 발생하는 취약점의 종류는 매우 다양하다. 웹 응용 프로그램 이외의 기존 서버 응용 프로그램들에서도 모두 공통적으로 발생하는 취약점인 버퍼 오버플로우, 설정 취약점, 불완전한 암호 사용 등의 취약점들은 그대로 웹 응용 프로그램에서도 발생한다. 따라서, 이들의 공통적인 취약점들은 기존의 방법을 이용해서 검사가 가능하다고 할 것이다.

하지만, 웹 응용 프로그램에서는 기존에 보기 어려웠던 새로운 종류의 취약점들이 더 많이 나타난다. 이들은 외부의 입력들 중 문자열로 받아들여진 것들에 의해서 발생하는 특징이 있다. 그 예로는 검증되지 않은 파라미터, Cross-Site Scripting [3], SQL Injection, Command Injection 등이 있고, 이하에서 하나씩 설명한다.

1. 검증되지 않은 파라미터

외부에서 입력으로 들어올 수 있는 폼 파라미터의 내용이 검증되지 않고 사용되는 경우 발생한다. 모든 필드의 내용은 클라이언트가 악의적으로 바꿀 가능성이 있으므로 항상 검증되어야 한다. 이하에 hidden field manipulation의 예를 들어 설명한다.

Hidden field는 종종 클라이언트의 세션에 대한 정보를 저장하거나, 서버의 복잡한 데이터베이스를 유지하는 필요를 없애는데 쓰인다. 클라이언트는 보통 hidden field를 볼 수 없고 그것을 변경할 수 없다. 그러나 폼 필드를 수정하는 것은 매우 간단하다. 예를 들어, 상품의 가격이 흔히 쓰이는 것처럼 hidden field에 있고 어떤 백엔드 시스템에 의해 관리된다고 한다면, 해커는 가격을 바꿀 수 있고, CGI는 해커에게 새로운 가격을 청구할 것이다. 방법은 다음과 같을 수 있다.

- HTML 에디터로 *html* 페이지를 연다.
- *hidden field*에 다음과 같이 적혀있다.

"<type=hidden name=price value=99.95>"

- 내용을 다른 값으로 바꾼다.

"<type=hidden name=price value="1.00>"

- html파일을 저장하고 그것을 열어 본다.
- 물건을 사기 위해 "buy"버튼을 누른다.

검증되지 않은 파라미터는 사실상, 이하의 다른 취약점을 모두 대표할 수 있는 전형적인 형태라고 할 수 있다.

2. Cross-Site Scripting

Cross-site scripting(XSS) 취약점은 침입자가 자바 스크립트 같은 악의적인 코드를 다른 사용자에게 보내기 위해 웹 응용 프로그램을 사용할 때 일어난다. 웹 응용 프로그램이 출력이 발생시키는 일에 대해 필터링하지 않은 채 사용자로부터의 입력을 사용할 때, 침입자는 그 입력에 침입을 삽입하고, 웹 응용 프로그램은 그 침입을 사용자에게 보낸다. 사용자는 웹 응용 프로그램을 믿게 되고, 정상적으로는 허락되지 않는 일들을 행하는 침입은 실행된다. 침입자는 유니 코드를 사용하는 것과 같은 악의적인 태그 부분을 인코드하는 다양한 방법을 사용한다. 그러므로 요청은 사용자에게 덜 의심스러워 보인다.

XSS 기법은 일반적으로 저장과 반향 두 카테고리고 분류된다. 저장 기법은 데이터베이스, 메시지 포럼, 방문자 로그 등의 목표 서버에 영구적으로 삽입된 코드가 저장되는 것이다. 반향 기법은 삽입된 코드가 이메일 메시지나 다른 서버와 같은 다른 희생자로 가는 것이다. 사용자가 링크를 클릭하거나 폼을 수락하는 속임수에 넘어가면, 삽입된 코드는 취약한 웹 서버를 돌아다니고 사용자의 브라우저에 침입을 다시 되돌려 보낸다. 브라우저는 그 코드가 신뢰성 있는 서버로부터 왔기 때문에 그 코드를 실행한다.

3. SQL Injection

웹 응용 프로그램은 대량의 데이터를 다루기 위해 백엔드에 데이터베이스를 사용하는 경우가 많다. 흔히 사용되는 데이터베이스 언어는 SQL이므로, 이 기법은 SQL injection이라고 불린다.

이 기법이 가능한 근본적인 이유는 사용자의 입력이 SQL 쿼리문의 일부로 사용되기 때문이다. SQL을 사용하는 웹 응용 프로그램은 사용자의 요구를 해결하기 위해서 사용자에게 쿼리 문의 일부를 요청한다. 그 예로는 데이터베이스 검색을 위한 필드의 내용의 요청이 있을 수 있다. 거의 모든 경우 웹 응용 프로그램이 원하는 것은 데이터뿐이다. 그러나, 클라이언트는 이 부분에 데이터가 아닌 쿼리문의 코드를 삽입할 수 있다. 이러한 입력이 점증되지 않고 사용되는 경우, 웹 응용 프로그램이 원래 의도했던 것과는 다른 SQL 문에 데이터베이스로 보내지게 된다.

4. Command Injection

웹 응용 프로그램은 큰 화이이나, 시스템과 관련된 데이터를 다루기 위해 운영체제의 명령어를 사용하야 하는 경우가 있다. 앞의 SQL injection과 비슷하게, 이 경우에도 클라이언트의 악의적인 행동이 원하지 않는 시스템 커맨드를 실행하게 되는 결과가 일어날 수 있다.

시스템 커맨드를 사용하는 웹 응용 프로그램은 구체적인 커맨드를 구성하기 위해 그 일부를 클라이언트에게 요구할 수 있다. 이 경우 클라이언트의 입력이 제대로 검사되지 않는다면, 그 입력에 데이터뿐 만 아니라 코드가 추가될 수 있다. 그러한 경우, 원치 않는 시스템 커맨드를 실행하는 결과가 일어난다.

III. 소스코드 검사 방법

1. 아이디어

본 절에서 살펴본 취약점이 발생하는 근본적인 이유는 외부에서 입력으로 받아들인 문자열의 내용이 제대로 검사되지 않기 때문이다. 전통적인 서버 프로그램의 취약점에서 유사하다고 할 수 있는 버퍼 오버플로우와 비교해 보면 유사점도 있으나, 다른 점이 더 많다고 할 수 있다. 가장 특징적인 차이점은 문자열의 길이 보다는 내용이 더 중요하다는 점이다. 이 차이점 때문에 버퍼 오버플로우를 검사하기 위해 사용되었던 방법들은 전혀 사용될 수 없다. 더 큰 문제점은 웹 응용 프로그램의 특정한 소스코드가 주어졌을 때 이 소스코드가 입력의 내용을 제대로 검사하는 지 알아낼 수 있는 일반적인 방법은 없다는 점이다.

따라서, 프로그래머로부터 일정부분 도움을 받지 않은 상태에서는 제대로 된 검사를 하기 어려울 것으로 예상한다. 본 연구에서는 다음과 같은 방법을 사용하여 검사를 수행할 수 있을 것이라 예상한다.

- 알려진 입력 검사 라이브러리 루틴에 관한 정보 수집

- 프로그래머가 작성한 입력 검사 루틴에 관한 정보 요청

- 위의 두 가지 자료를 이용하여 소스코드를 검사

소스코드의 검사과정에서는 외부에서 입력이 들어오는 위치를 파악하고, 그 문자열이 알려진 검사 루틴을 통과하는 지의 여부를 검사할 수 있도록 한다.

2. 검사 방법

기존의 취약점 검사 방법들은 패턴 매칭 형식의 방법과 코드 분석 방식으로 대략 구분할 수 있다. 패턴 인식 형식의 방법은 코드에서 발생할 수 있는 위험성의 예들을 이용하여 패턴을 구축하고, 그 패턴들과 비슷한 형태가 소스 코드에 나타나는지를 검사하는 방법이다. 이 방법은 parsing을 수행하지 않고 작업하므로 false positive가 많은 단점이 있다. 또한, 새로운 형태의 위험성인 문자열 내용에 의존하는 위험성에 대해서는 거의 구분이 어려울 것으로 예상된다. 이러한 방법들은 상수로 주어진 문자열의 내용에 대해서만 접근이 가능하기 때문이다. 웹 응용 프로그램에서 발생하는 문자열 내용에 의존한 위험성들은 변수로 주어진 문자열의 내용이 가장 중요한 요소가 되기 때문에 검사가 매우 어려울 것으로 예상한다.

두 번째의 방법인 parsing에 의한 방법은 소스코드를 좀더 자세히 검사할 수 있어서 버퍼 오버플로우 등의 문제점에 대한 상세한 분석이 가능하다. 하지만, 이러한 방법들도 정수 형 변수들이 주요한 문제가 되어서 발생하는 위험성에 대해서만 좋은 성능을 보일 뿐, 문자열 변수들에 의해서 발생하는 문제점에 대해서는 단점을 많이 가질 것으로 예상한다.

본 논문에서는 새로이 발생한 위험성에 대해 유용한 검사방법으로써 다음과 같은 방법을 사용할 것을 제안한다. 문자열의 내용에 의해 발생하는 위험성들은 그 문자열들이 변수에 저장됨으로 인해서 소스코드만 보아서는 위험성이 있는지 없는지에 관해 정확한 분석이 대단히 어려운 특징을 가진다. 일반적으로, 문자열의 내용이 어떤 방식으로 변화할 지를 소스코드만 가지고 검사하는 방법은 없는 것으로 알려져 있다. 따라서, 이 부분에서는 프로그래머의 도움이 반드시 필요하다. 또한, 문제점들을 해결하기 위해 많은 수의 프로그래머들이 공통적으로 사용하고 있는 검사방법들이 루틴으로 만들어져 있다. 따라서, 본 연구에서 제안하는 검사방법에서는 우선 문자열 변수가 위험성을 가지고 있는지 아닌지 검사하는 루틴들의 데이터베이스를 사용한다. 이 데이터베이스는 공통적으로 알려져 있는 루틴들의 이름과 프로그래머가 제공하는 루틴들의 이름을 가지고 구성된다.

소스 코드를 분석함에 있어서, 우선 외부 입력으로 들어오는 문자열 변수들을 정리한다. 그 다음 소스코드 상에서 이 문자열 변수들이 입력되는 시점, 검사되는 시점과, 사용되는 시점을 구분한다. 이 구분을 이용해서 문자열 변수들이 입력된 후 사용되기 전에 검사를 거치는 지의 여부를 판단하여 위험성이 소스코드에 있는지를 알아낼 수 있을 것이다.

IV. 구현 및 결과

1. 프로그램 구성

검사 프로그램은 다음과 같은 요소들로 구성된다. 검사 대상은 현재로는 PHP 코드로 제한하였지만, 다른 언어로도 쉽게 확장이 가능하다.

1) lexical analysis

입력 PHP 코드를 의미 있는 개별 단위로 구분한다. 모든 identifier와 punctuation들은 하나하나씩 구별된다.

2) 변수 판정

코드 상에 나타나는 모든 변수들을 구분하여 메모리에 저장한다. 특히 변수들이 외부 입력을 가지는 경우를 따로 구분하여 상태에 표시한다.

3) statement 판정

개별 statement를 모두 구분하여 종류를 판단한다. 가장 중요한 것들은 assignment statement와 함수 호출 statement이다. Assignment statement는 변수의 내용을 복사하게 되므로, 한 변수의 상태가 다른 변수에 전이되는 결과를 얻게 된다.

4) 함수 판정

클라이언트로부터 입력을 수행하는 함수들을 구별하여, 이 경우 보안 문제가 있는 입력이 들어올수 있음을 변수에 표시한다.

Html의 내용에 출력을 하는 함수를 판정한다. 만약 출력의 내용이 보안 문제가 있는 변수가 사용된 것이면 위험성을 출력하기 위해 준비한다.

변수의 내용이 다른 변수로 전이되는 함수는 assignment statement와 동등하게 다룬다.

5) 변수 상태 판정

프로그램의 시작에서부터 변수의 상태를 statement-by-statement로 유지하면서, 프로그램을 traverse한다. 변수의 상태는, 입력일 가능성이 있는 변수인 경우 초기에는 위험, 아닌 경우 초기에는 안전으로 표시한다. 이후, 각 statement 별로 변수의 상태가 앞에서 설명한 바와 같이 바뀐다.

특정 statement가 출력함수를 가지고 있는데, 사용되는 변수가 위험인 상태인 경우 위험성이 있음을 출력함으로써 최종 결과를 얻는다.

2. 실험 결과

다음과 같은 XSS의 형태는 기존의 검사 프로그램인 RATS 등에서 검사가 불가능한 것이다. 실험결과, 본 논문에서 제안한 방법으로 구현한 프로그램으로는 검사가 가능함을 확인하였다.

침입 URL:

<A HREF =

"http://auction.example.com/<script>

alert('hello')</script>">Click here</A>

침입당한 Web Page 내용

<HTML>

404 page not found:

<script>alert('hello')</script>

<HTML>

위와 같은 침입은 404 에러 페이지를 출력하는 PHP 코드가 입력을 전혀 검사하지 않는 경우 발생할 수 있다. 예로서는 다음과 같은 코드가 될 수 있다. 이 코드를 이용해서 우리의 프로그램이 어떻게 검사를 할수 있는 지 설명하도록 한다.

```
<HTML>
<?php echo "404 page not found:\n";
      echo $error_URL;
<HTML>
```

위의 코드에서 $error_URL이라는 변수는 코드에 처음 등장하므로, 입력 위치와 사용 위치가 같다고 할 수 있다. 이 경우 검사를 거칠 수 있는 방법이 전혀 없으므로 Cross Site Scripting에 취약하게 된다. 만약, 프로그래머가 해당 변수에 검사 함수를 적용하였고, 우리의 프로그램에 그 검사 함수가 등록되었다면 취약점을 출력하지 않을 것이다. 방금 본 예는 매우 간단한 것이지만, 더욱 복잡한 경우라도 검사하는 것이 가능함을 알수 있고, 실험으로도 확인하였다.

약 15개의 PHP 소스코드를 이용하여 RATS와의 성능 비교실험을 수행하였다. 사용된 PHP 소스코드는 본교의 한 웹 페이지의 소스코드 일부와, Open Web Application Security Project의 취약점을 보여주는 예제들로 구성되었다.

실험결과, 새로운 프로그램은 모든 소스코드에서 3-6개의 취약점을 찾아낸 데 반해, RATS는 4개의 소스 코드에서 2-7개의 취약점을 찾아내는 데 그쳤다. 본 논문의 목표는 이전의 취약점 검사 프로그램들이 찾아내지 못하는 새로운 종류의 취약점을 찾아내는 것이 가장 중요한 목표라고 할 수 있다. 이 관점에서도 좋은 성능을 보였는데, RATS는 새로운 프로그램이 찾아낸 취약점을 하나도 찾아내지 못하였다.

V. 결론

기존의 서버 응용 프로그램에서 발생하는 보안 위험성들은 외부 입력의 내용보다는 그 크기 등의 간단히 검사 가능한 특징에 의존하는 경우가 많았다. 하지만, 웹 응용 프로그램에서는 외부 입력의 내용을 코드의 일부로 사용하는 경우가 많음으로 인해서, 입력 문자열의 내용에 의존하는 위험성들이 많이 발생한다. 본 논문에서는 이러한 새로운 방식의 위험성을 소스코드를 이용해서 어떻게 자동적으로 검사할 수 있을지에 관해서 새로운 아이디어를 제시하고 초기 실험 결과 기존의 검사 프로그램들이 찾아내지 못하는 취약점들을 찾아낼 수 있음이 확인하였다.

현재 검사 프로그램은 완전한 파싱을 수행하지는 않는다. 따라서, if, while, case 문등 복잡한 형태로 만들어진 PHP 프로그램을 정확히 이해하고 분석할 능력은 가지지 않는다. 앞으로 복잡한 구문들을 더 이해할 수 있는 형태로 발전시켜 나가기 위한 작업을 계속할 예정이다.

참고문헌

[1] CERT, http://www.cert.org
[2] CGISecurity, http://www.cgisecurity.com
[3] CGISecurity, The Cross-Site Scripting FAQ, May, 2002
[4] Flawfinder, http://www.flawfinder.com
[5] PScan http://www.striker.ottawa.ca/~aland/pscan
[6] RATS, http://www.securesoftware.com/rats
[7] SPLINT, http://www.splint.org