**보안결함 방지를 위한 웹코딩 상세 구현 가이드**

내용

[개요 2](#_Toc458028142)

[URL 파라미터, form 파라미터의 처리 3](#_Toc458028143)

[SQL Injection 3](#_Toc458028144)

[myBatis 3](#_Toc458028145)

[XSS(Cross Site Scripting) 3](#_Toc458028146)

[jQuery의 .html(), .text() 3](#_Toc458028147)

[JSTL 4](#_Toc458028148)

[참조링크 4](#_Toc458028149)

[CSRF 4](#_Toc458028150)

[Tomcat에서의 구현 4](#_Toc458028151)

[참조링크 4](#_Toc458028152)

[Cross-Domain 4](#_Toc458028153)

[Same Orgin 4](#_Toc458028154)

[postMessage() 4](#_Toc458028155)

[File Download 4](#_Toc458028156)

[암호화 4](#_Toc458028157)

[Spring Framework 4](#_Toc458028158)

[별첨 5](#_Toc458028159)

[행정기관 및 공공기관 정보시스템 구축·운영 지침 5](#_Toc458028160)

[시큐어 코딩 가이드 (행정자치부-2012) 5](#_Toc458028161)

[개발보안 점검 가이드 (행정자치부-2016) 5](#_Toc458028162)

## 개요

### 문서의 목적

웹개발시 시큐어코딩에 대해 웹어플리케이션 수준에서 고려해야할 사항에 대한 내용이 주로 서술되어 있으며, 기존 공공기관에서 배포되는 시큐어코딩 가이드에 포함되지 않는 자바프레임웍, 자바스크립트 라이브러리를 고려한 내용을 포함하여 현실적, 효율적, 범용적으로 웹 구성 환경에 적용할 수 있는 코딩가이드를 제공하기 위해 본 문서를 작성한다.

또한, 보안을 위해 사용이 허용되지 않는 기능을 우회적으로 구현하기 위한 기법도 가이드한다.

### 문서의 포함 범위

웹어플리케이션 경로 하위의 리소스에 대한 내용만을 다룬다. (web.xml, js, jsp, java 등)

웹서버, WAS, Network에 대한 내용은 포함하지 않는다.

## URL 파라미터, form 파라미터의 처리

### 취약점 내용

**SQL Injection, XSS** 등 다수의결함은 클라이언트에서 전달된 파라미터에 대한 보안 결함 방지 처리로 1차적으로 제거할 수 있으며, URL/form 파라미터에 대한 처리는 기본적으로 구현되야할 보안 결함 방지 행위이다.

상세 보안 결함 제거 기법은 하기의 개별 결함 방지 기법에 상세하게 기술되어 있다.

하지만, 게시글과 같은 경우 결함 방지 처리를 하지 말아야 하는 경우도 있어 각 비즈니스 요건별 개별 처리가 필요하다.

**대응 방안**

하기 개별 취약점에 대한 대응 방안 참고

## SQL Injection

### 취약점 내용

웹 사이트는 DBMS와 연동하므로 웹 어플리케이션에서 사용자의 입력값을 통하여 데이타 트랜잭션이 발생한다. 만일 사용자 입력값에 대한 유효성 검증이 누락될 경우, 악의적인 사용자는 정상적인 입력값 대신 SQL 쿼리문이 포함된 조작된 입력값을 전송하여 서버측 쿼리문의 구조를 변경할 수 있으며, 이를 통해서 사용자 인증을 우회하거나 데이터베이스의 정보를 유출시키고 서버의 시스템 명령어를 실행시킬 수도 있다.

### 대응 방안 - myBatis

매퍼에서의 파라미터 처리는 **#{key}** 엘리먼트로 처리하지만, SQL 구문에 대한 처리를 위해 **${sql}** 엘리먼트를 사용할 수도 있으며 이때 SQL Injection 결함이 발생할 수도 있다.

이 엘리먼트를 이용할 때 문자열 패턴 검색을하여 문제 발생의 소지가 있는 문자열을 제거하는 방식이 가이드되는 경우도 있으나, 문자열 패턴을 black list, white list로 단정짓기는 어려운 경우가 많아 이런 경우 사용자가 입력한 값을 이용하지 않고, 검증된 white list를 이용한 검증방식을 적용할 수 있도록 로직을 개발하도록 한다.

**개별보안결함 방지 코딩 내용 추가**

## XSS(Cross Site Scripting)

### 취약점 내용

웹 상에서 가장 기초적인 취약점 공격 방법의 일종으로, 악의적인 사용자가 공격하려는 사이트에 스크립트를 넣는 기법을 말한다. 공격에 성공하면 사이트에 접속한 사용자는 삽입된 코드를 실행하게 되며, 보통 의도치 않은 행동을 수행시키거나 쿠키나 세션 토큰 등의 민감한 정보를 탈취한다.

크로스 사이트 스크립팅이란 이름 답게, 자바스크립트를 사용하여 공격하는 경우가 많다. 공격 방법이 단순하고 가장 기초적이지만, 많은 웹사이트들이 XSS에 대한 방어 조치를 해두지 않아 공격을 받는 경우가 많다. 여러 사용자가 접근 가능한 게시판 등에 코드를 삽입하는 경우도 많으며, 경우에 따라서는 메일과 같은 매체를 통해서도 전파된다.

### 대응 방안 - jQuery의 .html(), .text()

|  |
| --- |
| jQuery('div').text('<test>a&f"#</test>');​  jQuery('div').html('&lt;test&gt;a&f"#&lt;/test&gt;');​ |

위의 text, .html 함수는 동일한 결과를 출력, **text** 함수는 파라미터를 unescape하여 처리하므로 XSS 방지 효과를 기대할 수 있다.

예를 들어 XSS 안전성이 검증되지 않고 DB에 저장된 사용자 입력 게시판의 내용을 화면에 출력할 경우 사용할 수 있다.

|  |
| --- |
| $(function() {  var untrusted = "<script>alert('xss');</script>";  $("#jquery").html(  $("<div/>").text(untrusted).html()  );  }); |

**html**과 **text** 함수를 위의 예와 같이 혼합하여 사용할 수 있다

### 대응 방안 - JSTL

**<c:out>** 커스텀 태그의 사용법은 **<c:out value="value" [escapeXml="{true|false}"] />** 이다.

경우에 따라 간략하게 ${변수명} 형식으로 사용할 수 있으나, 이 경우 **escapeXml**이 **false**로 설정된 결과가 출력된다.

**<c:out>** 태그를 사용하면 **escapeXml**의 기본설정값은 **true**이므로, **<c:out value="value"/>** 형식으로만 사용하더라도 XSS 방지 효과가 있다.

## CSRF

### 취약점 내용

웹 사이트의 취약점을 이용하여 사용자가 의도하지 않는 요청을 송신하도록 하는 공격의 의미한다. 이는 http프로토콜의 상태없음(stateless) 특성에 기인한 특정 웹 어플리케이션에 대한 일련의 요청들의 상관관계를 특정할 수 없기 때문에 세션 유지등에 일반적으로 사용되는 쿠키 정보 등이 조건만 만족한다면 자동적으로 송신되기 때문에 가능하다.

여기서 상관관계를 특정할 수 없다는 의미는 예를 들어 카트화면 🡪 주문정보 입력 🡪 주문완료로 이어지는 주문 프로세스를 가진 웹 어플리케이션에서 각각의 페이지에대한 요청이 연속적으로 이어지는지에 대한 제어를 할 수 없다는 것을 의미한다. 이 공격수법은 결과적으로 피해자가 의도한 요청과 동일한 과정으로 진행되므로 공격자에 대한 추적이 어려울 수 있으며 피해가에게 인가된 범위안에서만 공격이 이루어진다는 특징이 있다.  
(피해자가 특정 웹 어플리케이션의 관리자 계정으로 인증&인가된 상태라면 피해범위가 커질 수 있다.)

### 대응 방안 - Tomcat

Tomcat에서는 기본적으로 CSRF 방지를 위한 서블릿 필터를 제공한다.

* **web.xml 내용 예시**

|  |
| --- |
| **<!-- 일반 URL 요청 -->**  <filter>  <filter-name>CsrfFilter</filter-name>  <filter-class>org.apache.catalina.filters.CsrfPreventionFilter</filter-class>  <init-param>  <param-name>entryPoints</param-name>  <param-value>/home</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>CsrfFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping>  **<!-- REST API 요청 -->**  <filter>  <filter-name>RestCSRF</filter-name>  <filter-class>org.apache.catalina.filters.RestCsrfPreventionFilter</filter-class>  </filter>  <filter-mapping>  <filter-name>RestCSRF</filter-name>  <!-- Modifying operations -->  <url-pattern>/resources/removeResource</url-pattern>  <url-pattern>/resources/addResource</url-pattern>  <!-- Non-modifying operations -->  <url-pattern>/resources/listResources</url-pattern>  </filter-mapping> |

* **Initialisation parameters**

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| denyStatus | 요청이 거부될 때 사용될 HTTP response status code로 기본값은 403이다. |
| entryPoints | 암호화 토큰(nonce)의 정확성이 검증되어야 할 URL의 목록으로 콤마로 각 항목이 구분 된다. |
| nonceCacheSize | 캐쉬되어야할 암호와 토큰의 수로 동시 요청에 의한 대응하기 위해 지정한다. 기본값은 5이다.. |
| randomClass | 암호화 토큰을 생성할 클래스명으로 Java.util.Random의 instance이여야 한다. 기본값은 java.security.SecureRandom이다. |

* **개발**

## Cross-Domain

## Same Orgin

### postMessage()

## File Upload/Download

## 암호화

### Spring Framework

## 에러 페이지 미비

### 취약점 내용

공격자가 대상 시스템의 현황을 파악하기 위하여 다양한 에러를 유발시킴으로써 반응하는 에러 결과값으로 웹 프로그램의 구조 및 환경설정을 추측하여 공격에 이용되므로 이러한 Error를 위한 별도의 페이지를 작성하여 Error발생시 기본 Error Page로 Redirect 시킴으로써, 불필요한 정보가 노출되지 않도록 해야 한다.

SQL Injection 공격으로 대상시스템의 SQL Query문장의 구성을 알아내기 위한 Foot Printing 시도와 같은 것이 이 항목에 해당된다.

### 보안 대책

모든 에러(404 error,403 error,500 error, …)발생 시 에러 메시지를 외부에 제공하지 않도록 한다.

에러가 발생한 경우 에러 발생 메시지를 사용자 클라이언트의 브라우저에 표시하지 않게 하고 메인 페이지 또는 별도로 만든 에러 페이지로 Redirect 시키도록 한다.

### 에러 페이지의 지정

web.xml에 다음 예시를 참고로 에러 페이지를 추가한다.

|  |
| --- |
| <!-- http 상태 코드를 이용한 에러페이지 지정 -->  <error-page>  <error-code>404</error-code>  <location>/error/404code.jsp</location>  </error-page>  <!-- java Exception 클래스를 이용한 에러페이지 지정 -->  <error-page>  <exception-type>java.lang.Throwable</exception-type>  <!-- 에러페이지의 경로는 파일 경로 지정 가능 -->  <location>/WEB-INF/jsp/common/error/error.jsp</location>  </error-page> |

## 검색 엔진 노출

### 취약점 내용

Google, Naver, Daum 등 대부분의 검색엔진은 robots.txt 파일에 의한 검색 거부 규약을 지키고 있으므로 규약에 맞게 robots.txt 파일을 작성하여 적용하면 검색엔진에 노출되는 것을 원치 않는 URL에 대해서 검색을 거부할 수 있다.

### robots.txt 작성 방법

* 모든 검색엔진이 자신의 웹사이트 검색을 금지함

|  |
| --- |
| User-agent: \*  Disallow: / |

* 모든 검색엔진에게 자신의 특정 디렉토리에 대한 검색을 금지함

|  |
| --- |
| User-agent: \*  Disallow: /my\_directory1/  Disallow: /my\_directory2/my\_directory3 |

* 특정 검색엔진이 자신의 웹사이트를 검색하는 것을 금지함

|  |
| --- |
| User-agent: Googlebot  Disallow: / |

### robots.txt 작성시 주의 사항

* robots.txt 파일은 웹 URL의 최상위 디렉토리에 위치해야 함
* robots.txt 에서 디렉토리 경로는 URL에서 도메인 주소를 뺀 나머지 디렉토리 트리 구조로 표현함
* robots.txt 파일은 텍스트 파일로서 누구나 URL을 통해서 열람이 가능한 파일이므로 관리자 페이지경로와 같은 민감한 경로는 포함해서는 안됨

### 메타 태그 설정 (특정 페이지에 대한 검색 거부)

메타 태그를 이용하면 특정 페이지에 대한 정보 수집을 거부할 수 있다. 메타 태그를 사용하기 위해서는HTML의 <head> 태그 사이에 다음 구문을 삽입한다.

|  |
| --- |
| <meta name="robots" content="noindex, nofollow"> |

## 참조링크

### XSS

https://www.owasp.org/index.php/XSS\_(Cross\_Site\_Scripting)\_Prevention\_Cheat\_Sheet

### CSRF

https://tomcat.apache.org/tomcat-7.0-doc/config/filter.html

## 별첨

### 행정기관 및 공공기관 정보시스템 구축·운영 지침

* 행정기관 및 공공기관 정보시스템 구축·운영 지침.hwp
* 행정기관 및 공공기관 정보시스템 구축·운영 지침-별첨- [별표 3] 소프트웨어 보안약점 기준(제52조 관련).hwp
* 행정기관 및 공공기관 정보시스템 구축·운영 지침-[별표 4] 소프트웨어 보안약점 진단원의 자격기준(제54조 관련).hwp

### 시큐어 코딩 가이드 (행정자치부-2012)

* JAVA\_시큐어코딩\_가이드-201209.pdf

### 개발보안 점검 가이드 (행정자치부-2016)

* 공개SW를\_활용한\_소프트웨어\_개발보안\_점검가이드.pdf