- Web 모의해킹 -

# XSS(크로스 사이트 스크립팅)

SK 인포섹, 이호석

leehs2@sk.com

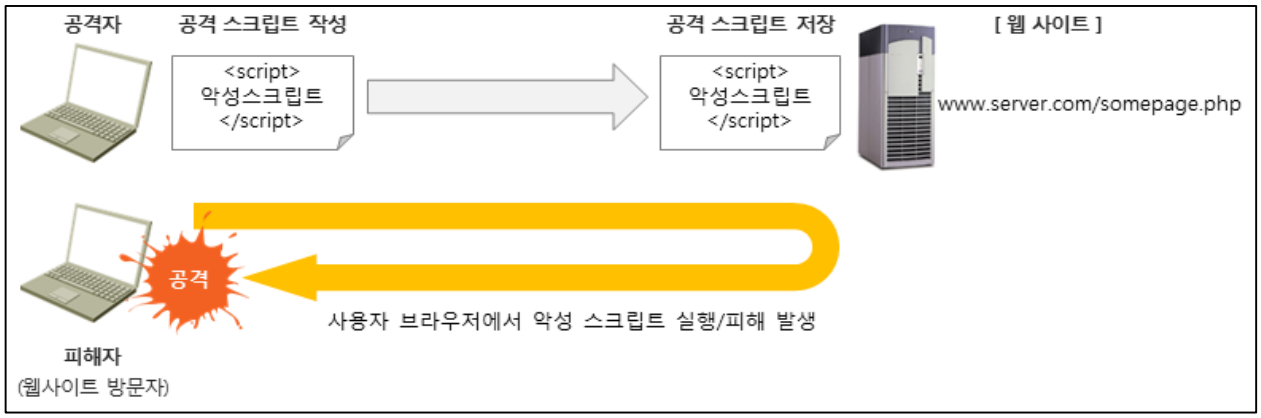
## 1. XSS 공격 방법

XSS(크로스 사이트 스크립팅)은 웹 페이지에 악의적인 스크립트를 포함시켜 사용자 측에서 실행되게 유도하는 취약점으로, 모의해킹/소스코드 진단 시 가장 높은 비율로 탐지되는 취약점입니다. 비공식적인 통계가 90%라고 할 정도로 매우 광범위한 곳에서 발생하고 있는데, 이는 공격하기 쉽고, 완벽하게 방어하기 어려운 XSS 의 특성 때문에 그렇습니다. XSS 의 트렌트부터 점검방법, 보안대책까지 차근차근 하나씩 살펴보겠습니다.

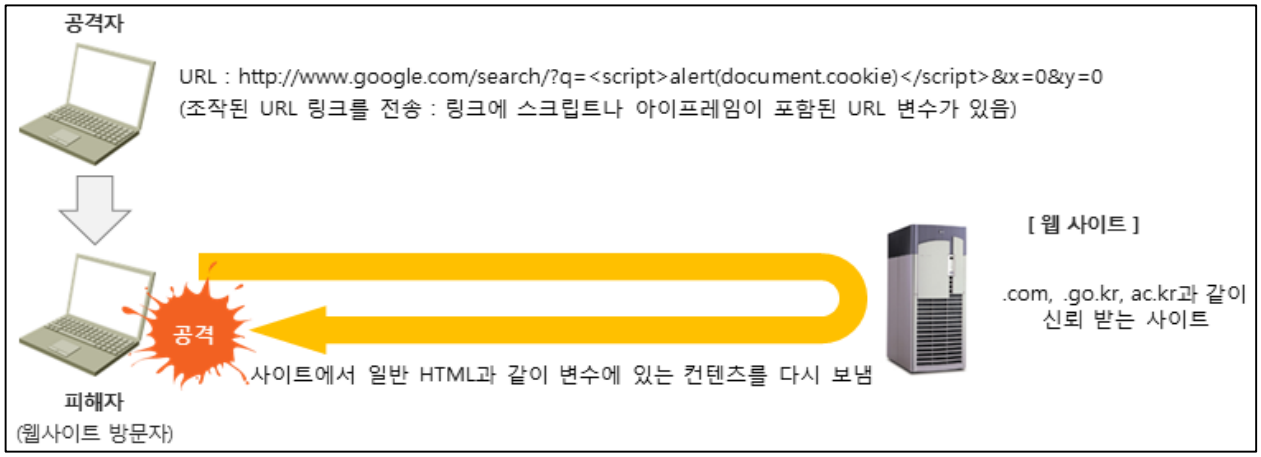
### 1.1. 종류

XSS 취약점은 Reflected XSS, Stored XSS 크게 두 분류로 나눌 수 있습니다. 개념적인 차이는 다음과 같습니다.

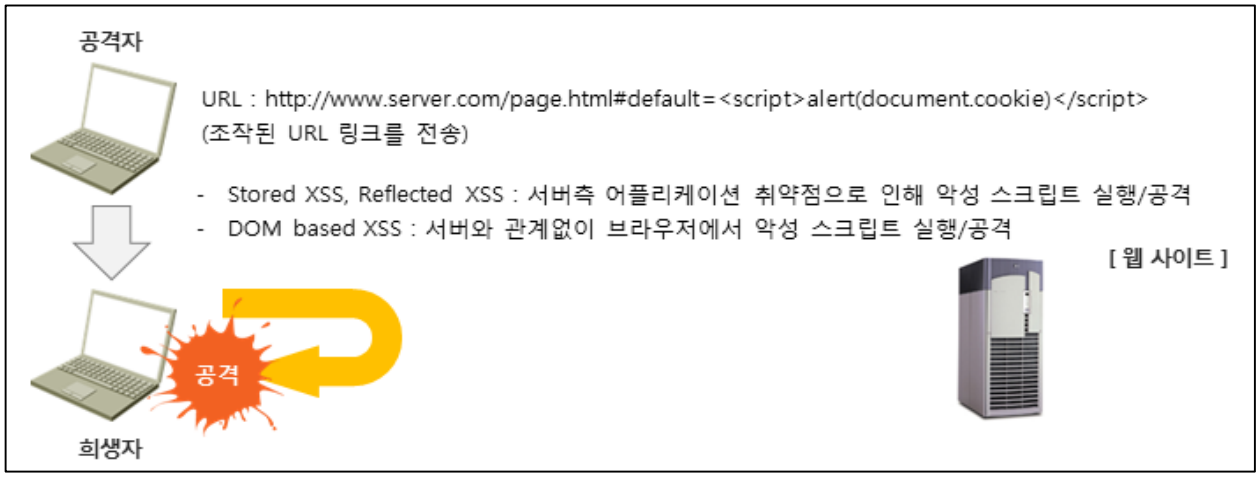
- **Stored XSS** : 공격자가 입력하는 정보가 DB 서버와 같은 저장소에 저장되고, 이 값을 출력하는 페이지에서 피해가 발생하는 취약점




- **Reflected XSS** : 공격자가 입력하는 정보가 별도로 저장되지 않고 같은 페이지에서 바로 출력되면서 피해가 발생하는 취약점

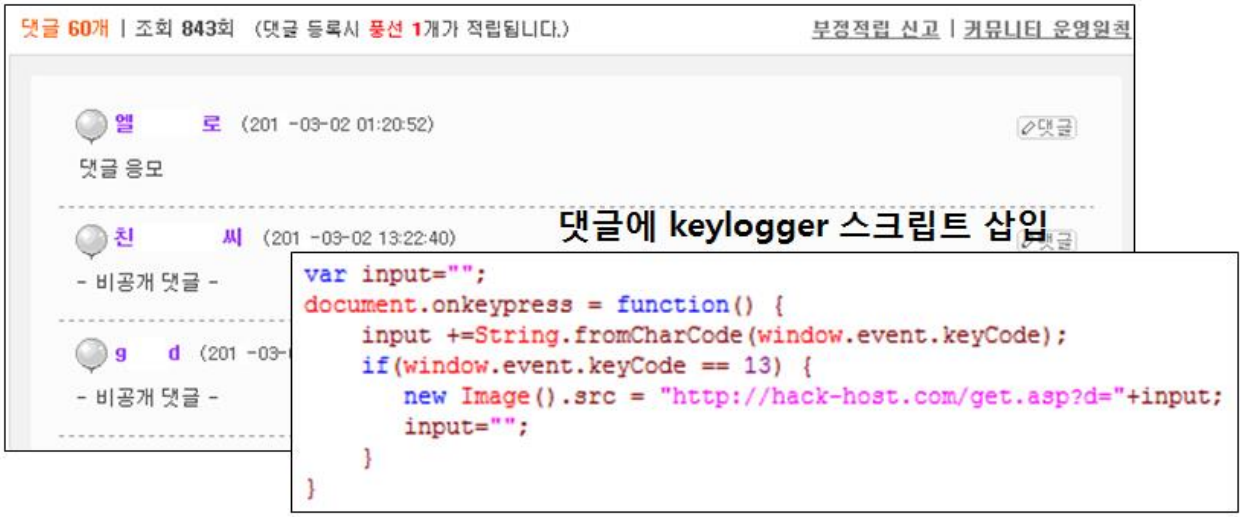



- **DOM XSS** : 공격 스크립트가 DOM 의 일부로 실행됨으로써 브라우저 자체에서 악성스크립트가 실행되는 취약점

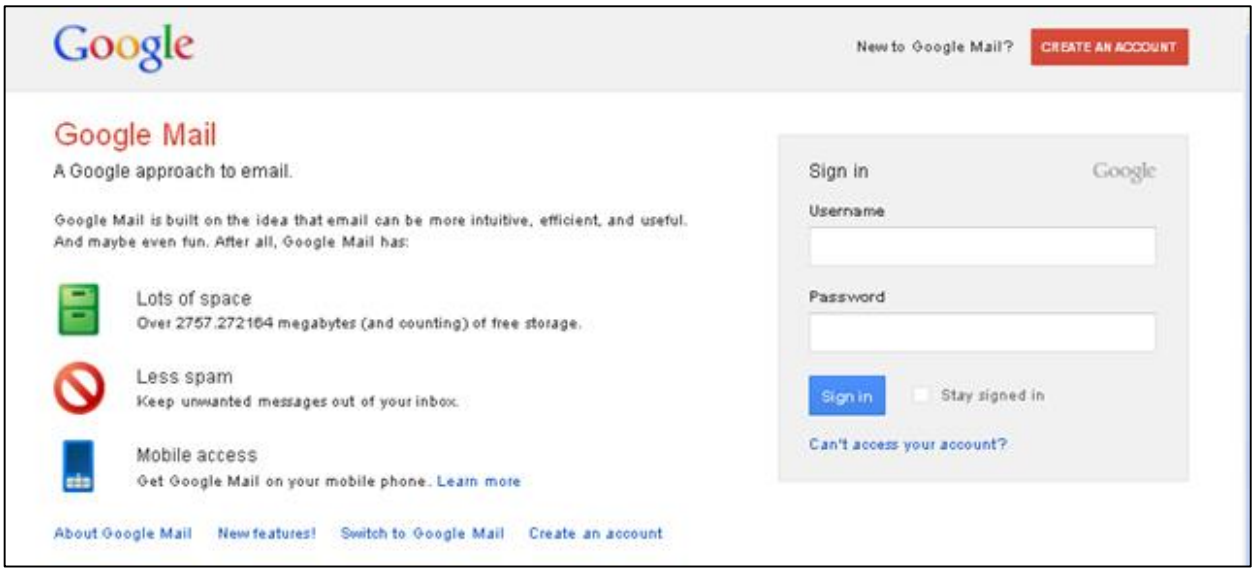


## 1.2. 실제 해킹 공격은?

XSS 취약점은 Javascript alert()함수가 실행되는지 보는 수준으로 점검하여 위험도가 '中'정도라고 생각하는 경우가 많지만, 실제 해킹은 다릅니다. 쿠키값을 공격자 서버로 전송하여 사용자 권한을 도용한다거나, Javascript 로 짜여진 키로거를 이용하여 키보드 입력값을 공격자 서버로 전송할 수 있습니다.




가장 크리티컬한 기법은 BeEF 라는 해킹도구와 연계하여 공격하는 것인데, 피해자 PC 에 취약한 JAVA, Flash, PDF 와 같은 다양한 취약점(Exploit)을 이용하여, 피해자 PC 의 Shell 을 획득하고 악성코드에 감염시킨다거나, 가짜 Google/Facebook 로그인 창을 띄워 계정을 탈취하는 등의 공격이 가능합니다. 이 같은 지능화된 기법들은 지속적으로 추가되고 있는 상황입니다.



### 1.3. 점검 방법

XSS 를 점검하는 가장 간단한 방법은 alert() 함수가 실행되는지 확인하는 것인데, 아래와 같이 사용자 입력값이 그대로 반환되는 우편번호 검색 페이지를 예시로 설명하겠습니다.

```
01 <%
02  request.setCharacterEncoding("utf-8");
03  String rdong=request.getParameter("dong");
04  String rid=request.getParameter("id");
05  String rname=request.getParameter("name");
06 %>
07
08 <html><head>
09 <script language="javascript">
10 function test(){
11 var test="<%=rdong%>";              // Javascript 에서 사용
12 }
13 </script></head>
14 <body>
15 ...
16 <!--  test, <%=rdong%>으로 검색중  -->          // 주석에서 사용
17 <center>
18 <form name="checkpost" method="post" action="./checkPost2.jsp">
19 <table border="1" width="100%">
20 <tr><td align="center">
21 우 편 번 호 체 크
22 </td></tr>
23 <tr><td>동 이름 입력
24 <input type="text" name="dong" size="100%" value="<%=rdong%>">          // input 태그에서 사용
25 <input type="submit" value="검색">
26 </td></tr>
27 </table>
```

공격자는 사용자 입력값이 페이지에 출력되는 포인트를 찾고, 문법에 맞게 공격을 하게되는데 이를 Step 별로 정리하면 다음과 같습니다.

**Step 1**. XSS 에 사용할 공격 스크립트를 구성합니다.

- <script>alert(123)</script>

**Step 2.** 입력값이 출력되는 포인트를 찾습니다.

- 입력값 : 03 라인, String rdong=request.getParameter("dong");
- 출력값 : 24 라인, <input type="text" name="dong" size="100%" value="<%=rdong%>">

**Step 3.** 출력되는 포인트에 사용되는 입력값 전,후 문법을 확인합니다.

- <input value="사용자입력값">

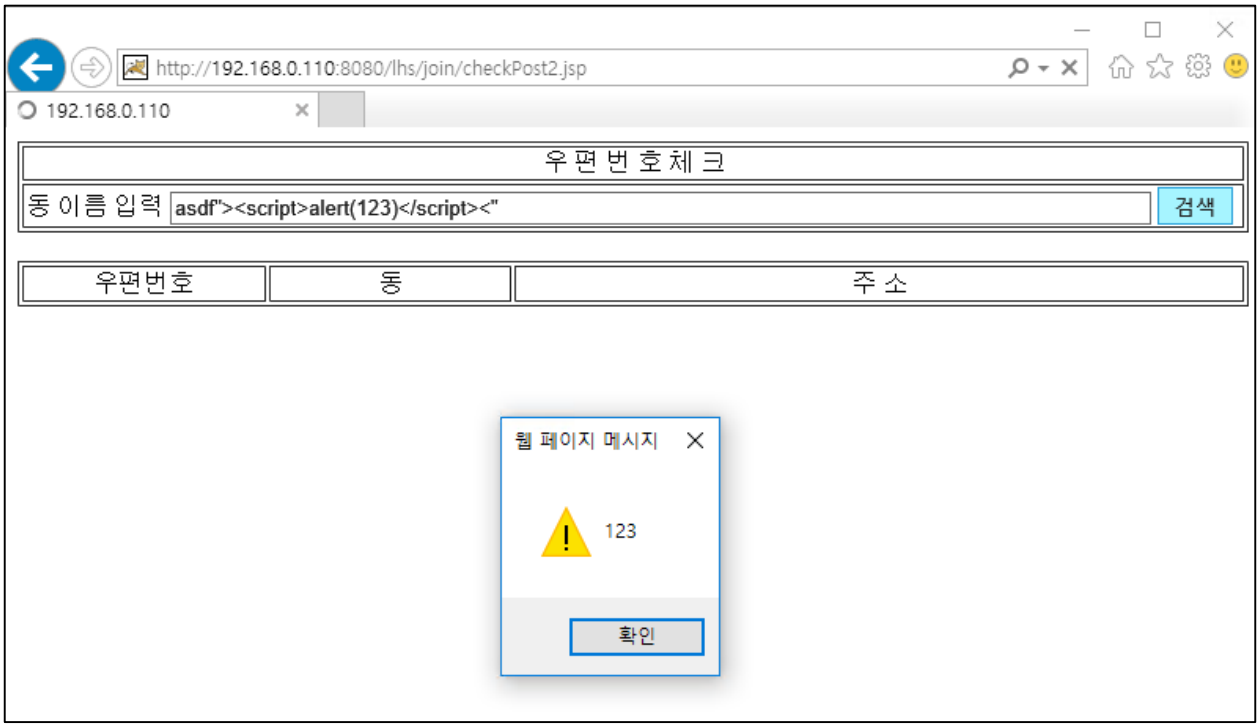
**Step 4.** 스크립트 삽입 후에도 페이지가 정상적으로 로딩 될 수 있도록 스크립트를 수정합니다.

- "><script>alert(123)</script><"
- <input value=""><script>alert(123)</script><"">

Input 태그 외에 주석이나 Javascript 에서도 아래와 같이 스크립트를 수정하여 공격이 가능합니다.

| 출력라인 | 종류 | 공격 스크립트(붉은색이 사용자 입력값) |
|---|---|---|
| 24 | Input | <input value=""><script>alert(123)</script><""> |
| 16 | 주석 | <!-- test, --><script>alert(123)</script><!-- --> |
| 11 | Javascript | <script> function test() { test="";}</script><script>alert(123)</script><script>""; }</script> |

이렇게 공격을 시도하면 다음과 같이 alert() 함수가 실행되는 것을 확인 할 수 있습니다.




**※. 유의사항**

- Chrome 은 브라우저 자체에서 방어하고 있어 테스트가 용이하지 않습니다.
- IE7 이하에서는 별도의 설정 없이 테스트가 가능합니다.
- IE8 이상에서는 "도구 > 인터넷옵션> 보안 > 인터넷 > 사용자 지정 수준 > 스크립팅 > XSS 필터 사용 > 사용 안 함"으로 설정하고 테스트가 가능합니다.

## 2. XSS 보안대책

XSS 취약점은 해킹하는것 보다 어떻게 막는지 보안대책을 정확히, 자세히 아는 것이 중요합니다. "어디를 막아야 하는지?", "무엇을 막아야 하는지?", "어떻게 막아야 하는지?" 이렇게 3 가지로 구분하여 필터링하는 원리를 이해하고, 각 사이트에 맞는 보안대책을 구성하는 방법을 알아보겠습니다.

### 2.1. 어디를 막아야 하는가? (입력값 or 출력값)

XSS 를 방어하는 방법은 입력값에서 필터링하는 방식과 출력값에서 필터링하는 방식 이렇게 두 가지가 있습니다. **입력값에서 필터링 할 경우**, DB 서버에 악성스크립트가 저장되는 것 자체를 원천적으로 차단하는 방식이라 가장 많이 사용하는 필터링 방식이고, **출력값에서 필터링 하는 경우**는 정상적으로 작성된 HTML 코드도 모두 치환하기 때문에 일반적으로 사용하지 않습니다. 하지만, 현재 XSS 취약점으로부터 양호하더라도 이전에 이미 악성스크립트가 저장된 시스템의 경우, 실행 방지를 목적으로 사용하는 경우가 종종 있습니다.

필터링 방식을 정했다면, 다음은 어떤값을 필터링 해야 할지 파악하는 작업을 합니다. 보통 파라미터를 100 개 사용하는 페이지가 있다고 하면 100 개를 모두 필터링 할지, 특정 파라미터만 필터링 할지 구분하지 못하는 경우가 많습니다. 기준은 간단합니다. **입력값 중 출력값에 쓰이는 파라미터가 무엇인지 파악하면 됩니다.** 여기서 입력값이란 사용자가 입력한 파라미터 뿐만 아니라 사용자가 입력하지 않은 파라미터도 포함합니다.
(※. 파라미터는 개발자가 페이지에 셋팅한 값이더라도 조작하여 서버에 요청 할 수 있음)

소스코드를 보면서 좀 더 자세히 알아보겠습니다.

```
01 <%
02  request.setCharacterEncoding("utf-8");
03  String rpagename=request.getParameter("pagename");
04  String rdong=request.getParameter("dong");
05  String rid=request.getParameter("id");
06  String rname=request.getParameter("name");
07
08  Connection conn=null;
09  Statement stmt=null;
10  ResultSet rs=null;
11
12  try{
13    Context init = new InitialContext();
14    DataSource ds = (DataSource)init.lookup("java:comp/env/jdbc/shanks123");
15    conn = ds.getConnection();
16
17    String sql = "SELECT * FROM LHSPOST WHERE DONG LIKE '%"+rdong+"%'";
18    stmt = conn.createStatement();
19    rs = stmt.executeQuery(sql);
20 %>
21
22 <html><body><center>
23 <form name="checkpost" method="post" action="./checkPost2.jsp">
24 <table border="1" width="100%">
25 <tr><td align="center"> 우편번호 체크
```

```
26 <input type="hidden" name="pagename" value="<%=rpagename%>"></td></tr>
27 <tr><td>동 이름 입력
28 <input type="text" name="dong" size="100%" value="<%=rdong%>">
29 <input type="submit" value="검색"></td></tr></table>
30 <br><table border="1" width="100%">
31 <tr><td align="center" width="20%">우편번호</td>
32 <td align="center" width="20%">동</td>
33 <td align="center">주소</td></tr>
34
35 <%
36  while(rs.next()){
37 %>
38  <tr><td align="center"><%=rs.getString("DONG") %></td>
39  <td align="center"><%=rs.getString("ADDRS") %></td></tr>
40 ...
41 </table></form></center></body></html>
```

위 예제는 우편번호를 검색하는 페이지인데, 위와 같이 파라미터로 받은 4 개(rdong, rpagename, rid, rname)의 입력값 중 페이지 출력에 사용되는 파라미터는 2 개(rdong, rpagename)입니다. Reflected XSS 취약점은 한 페이지에 입력과 출력이 모두 처리되기 때문에, 출력에 사용되는 값을 추적해 해당값이 외부 입력값이면 이를 필터링하면 됩니다. 정리하면 다음과 같습니다.

| 확인 순서 | 내용 |
|---|---|
| 페이지 출력값이<br>존재하는지 확인 | 26라인(rpagename), 28라인(rdong), 38라인(DONG), 39라인(ADDRS) |
| 외부 입력값인지 확인 | 외부 입력값 : 26라인(rpagename), 28라인(rdong)<br>DB값 : 38라인(DONG), 39라인(ADDRS) |
| 사용자 입력값<br>파라미터 필터링 | 03라인(rpagename), 04라인(rdong) |

***결과 해설 : 03~06 라인에 4 개 파라미터(rpagename, rname, rid, rname) 중 출력값으로 쓰이는 2 개의 파라미터(rpagename, rname)만 XSS 공격이 가능합니다.***

Stored XSS 역시 원리는 동일합니다. 게시판을 예로 들면 출력페이지는 view.jsp, 입력페이지는 write.jsp 와 같이 페이지가 분리되어 있다는 차이 밖에 없습니다.
하지만 Stored XSS 취약점을 소스코드 진단툴이나 제 3 자가 정확하게 판별하기는 어렵습니다. 입력과 출력하는 소스코드가 각각 다른 페이지에서 처리되기 때문에, 이 두페이지의 상관관계를 도출하기가 어렵습니다. 그래서 DB 에 저장하는 Insert 나 Update 에 사용하는 모든 부분을 취약하다고 탐지 할 수 밖에 없고, 오탐이 생길수 밖에 없는 구조입니다. 그렇기 때문에 Stored XSS 취약점은 개발자분들의 관심이 더 필요한 항목이라고 말할 수 있습니다.

## 2.2. 무엇을 막아야 하는가? (필터링 해야하는 문자열)

당연한 이야기지만 XSS 취약점은 Client 에서 실행되는 HTML, Javascript 가 실행되지 못하도록 하는게 목적입니다. 이 중 점검방법에서 언급한 HTML 태그 시작/종료 문법, Javascript 시작/종료와 같은 구분자만 막아도 원천적으로 차단된다고 볼 수 있는데, 이를 정리한 것이 아래 특수문자에 해당합니다. 그 외에 태그, 악성 스크립트, 이벤트 핸들러는 비즈니스 로직별 추가 필터링 목록이라고 보면 됩니다.

| 분류 | 필터링 목록 | | | | | |
|---|---|---|---|---|---|---|
| 특수문자 | < | > | ' | " | ( | ) |
| | ; | # | & | { | } | |
| 태그 | base | bgsound | blink | charset | applet | frame |
| | frameset | grameset | href | iframe | ilayer | innerHTML |
| | javascript | layer | link | meta | object | style |
| | title | xml | body | svg | lowsrc | dynsrc |
| | url | marquee | script | | | |
| 악성 스크립트 | cookie | document | alert | msgbox | vbscript | refresh |
| | escape | string | expression | eval | void | bind |
| | create | confirm | prompt | location | fromCharCode | |
| 이벤트 핸들러 | append | onbeforeprint | ondragleave | onmouseenter | onbeforeeditfocus | ondatasetchanged |
| | onbeforepaste | onbeforeuload | ondragover | onmouseleave | onbeforepaste | ondatasetcomplete |
| | onbeforeprint | onbeforeunload | ondragstart | onmousemove | onstart | ondbclick |
| | embed | onbeforeupdate | ondrop | onmouseout | onstop | ondblclick |
| | onerror | onblur | onerror | onmouseover | onsubmit | ondeactivate |
| | onabort | onbounce | onerrorupdate | onmouseup | onundo | ondrag |
| | onactivate | oncellchange | onfilterchage | onmousewheel | onunload | ondragend |
| | onactive | onchange | onfilterchange | onmove | onlayoutcomplete | onresizestart |
| | onafteripudate | onclick | onfinish | onmoveend | onload | onrowenter |
| | onafterprint | oncontextmenu | onfocus | onmovestart | onlosecapture | onrowexit |
| | onafterupdate | oncontrolselect | onfocusin | onpaste | onmouse | onrowsdelete |
| | onbefore | oncontrolselected | onfocusout | onpropertychange | onmousedown | onrowsinserted |
| | onbeforeactivate | oncopy | onhelp | onreadystatechange | onmouseend | onscroll |
| | onbeforecopy | oncut | onkeydown | onreset | ondragenter | onselect |
| | onbeforecut | ondataavailable | onkeypress | onresize | onselectstart | onselectionchange |
| | onbeforedeactivate | ondatasetchaged | onkeyup | onresizeend | onURLFlip | seekSegmentTime |
| | onTimeError | onTrackChange | | | | |

### 2.3. 어떻게 막아야 하는가? (비즈니스 로직별 필터링 형태)

어떻게 막는가의 관점에서는 사용자 입력값을 어떻게 사용하는가에 따라 처리 방식이 좀 다릅니다. 예를 들어 검색어를 입력하는 페이지의 경우 HTML 태그나 Javascript 가 완전히 필요 없습니다. 하지만 게시판 에디터와 같이 게시글 내용중에 HTML 태그가 일부 허용되어야 하는 경우도 있습니다. 이처럼 비즈니스 로직에 따라 조치하는 방법이 다른데 이는 3 가지로 분류 할 수 있습니다.

- **원천적으로 차단** : HTML 태그나 Javascript 를 사용할 수 없도록 함 > **특수문자 치환**
- **WhiteList Filtering** : 일부 태그만 사용 > **원천적으로 차단 후 사용하는 태그만 다시 원복**
- **BlackList Filtering** : 일부 허용한 태그에서 이벤트 핸들러와 같은 동적 함수 사용이 가능할 때 > **WhiteList Filtering 적용 후 이벤트 핸들러 추가 필터링**

이를 소스코드로 구현하면 다음과 같습니다. (※. 필터링 해야하는 전체 문자열은 "무엇을 막아야 하는가?" 참고)

```
01  <%
02  String test_str = request.getParameter("param_str");
03  String test_str_low = "";
04
05  if(test_str != null) {
06          //근본적인 해결책, HTML tag 를 모두 제거
07          test_str = test_str.replaceAll("<","&lt;");
08          test_str = test_str.replaceAll(">","&gt;");
09          test_str = test_str.replaceAll("(","&#40;");
10          test_str = test_str.replaceAll(")","&#41;");
11          test_str = test_str.replaceAll("&","&amp;");
12          test_str = test_str.replaceAll("₩"","&quot;");
13          test_str = test_str.replaceAll("₩'","&#39;");
14
15          //White List Filtering, 허용할 HTML tag 만 변경
16          test_str = test_str.replaceAll("&lt;p&gt;", "<p>");
17          test_str = test_str.replaceAll("&lt;P&gt;", "<P>");
18          test_str = test_str.replaceAll("&lt;br&gt;", "<br>");
19          test_str = test_str.replaceAll("&lt;BR&gt;", "<BR>");
20
21          //Black List Filtering, 스크립트 문자열 필터링
22          test_str_low= test_str.toLowerCase()
23
24          if(test_str_low.contains("javascript")   || test_str_low.contains("script")){
25          test_str = test_str_low;
26          test_str = test_str.replaceAll("javascript", "x-javascript");
27          test_str = test_str.replaceAll("script", "x-script");
28          }
29  }else{
30          test_str = "";
31  }
    %>
```

위 소스코드를 보면 문자열 길이에 따라 필터링하는 방식이 다른데 구분하는 방법은 다음과 같습니다.

- 1 글자 필터링 : 1 글자에 해당하는 특수문자 같은 경우 "<"를 ""로 치환하여 삭제시키거나, 고유기능을 인식하지 못하도록 "<"를 "<"와 같이 HTML 인코딩 하는 방법이 있습니다.
- 여러 글자 필터링 : 특수 문자 외에 글자 길이가 2 글자 이상인 경우 "script"를 ""로 치환하게 되면 "scscriptript"와 같이 입력했을 때 중간에 있는 script 가 없어지고 앞,뒤 문자열이 합쳐지면서 다시 script 라는 문자열만 남게 되어 결국 script 라는 문자열을 사용할 수 있게 됩니다. 따라서 여러 글자를 필터링 할 경우 문자를 추가하는 방식으로 치환하면 우회가 불가능 합니다. "script" -> "x-script"

추가로 XSS 방지를 위한 오픈소스가 존재하는데, XSS 취약점이 매우 빈번하게 일어나는 커뮤니티와 같은 사이트에서는 정규 표현식을 통해 여러가지 케이스별로 정교하게 필터링 해야하는 경우, 아래 모듈 사용을 검토하는 것도 괜찮은 방법입니다.

Naver - Lucy XSS Filter : https://github.com/naver/lucy-xss-filter
OWASP - ESAPI : https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API
Microsoft - AntiXSS(ASP.NET 한정) : https://www.owasp.org/index.php/.NET_AntiXSS_Library

**결론** : 이와 같이 XSS 취약점의 공격 방법과, 보안대책에 대해서 알아보았습니다. 서두에 말씀드린 것과 같이 XSS 취약점은 쉽고 빈번하게 일어나는 공격 중에 하나입니다. 장황하게 썼지만 어렵지 않은 내용이라 1~2 번만 꼼꼼하게 읽어보면 쉽게 조치가 가능하므로, 취약점을 제거하려는 투자와 노력만 있으면 완벽하게 조치가 가능합니다.