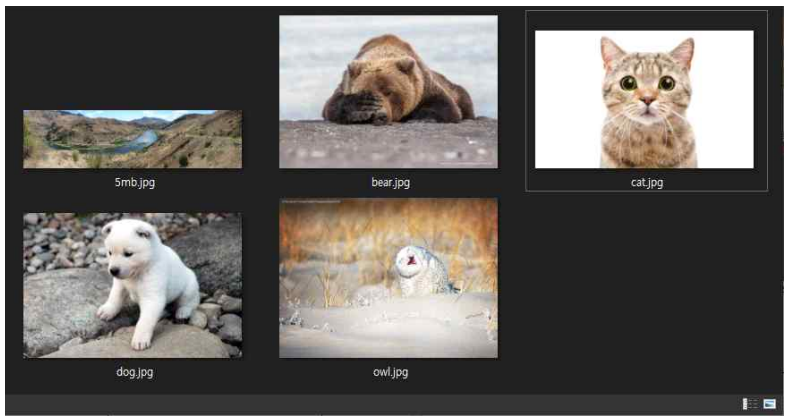
과목명	시큐어코딩
담당교수	우사무엘 교수님
학과	소프트웨어학과
학번	32153180
이름	이상민
제출일자	2020.06.11

1. 파일 업로드/다운로드 취약점 실습

- 업로드 기능에서의 취약점 발생 원인
  - 업로드되는 파일의 타입 및 크기나 개수를 제한하지 않는 경우
  - 업로드된 파일을 외부에서 직접적으로 접근 가능한 경우
  - 업로드된 파일의 이름과 저장된 파일의 이름이 동일하여 공격자가 파일에 대한 인식이 가능한 경우
  - 업로드된 파일이 실행 권한이 있는 경우

- 다운로드 기능에서의 취약점 발생 원인
  - 파일에 대한 접근 권한이 없는 사용자가 직접적인 경로를 이용하여 파일을 다운로드 할 수 있는 경우
  - 악성코드에 감염된 파일을 다운로드 허용하는 경우

(1) 공격 실험



글번호	제목	작성자	댓글수	조회수	추천수	작성일
1	대용량파일	테스트	0	0	0	2020-06-11 00:00:00
2	웹쉘 테스트	테스트	0	1	0	2020-06-11 00:00:00
3	파일 업로드 테스트3	테스트	0	1	0	2020-06-11 00:00:00
4	파일 업로드 테스트2	테스트	0	1	0	2020-06-11 00:00:00
5	파일 업로드 테스트	테스트	0	1	0	2020-06-11 00:00:00

파일 업로드 테스트

파일 업로드 테스트

작성자	조회수	추천수	작성일
테스트	1	0	2020-06-11
내용			

첨부파일: 1591857493155dog.jpg

test

'dog.jpg'라는 이름의 파일이 이미 존재하기 때문에 시간 정보를 앞에 붙었다.

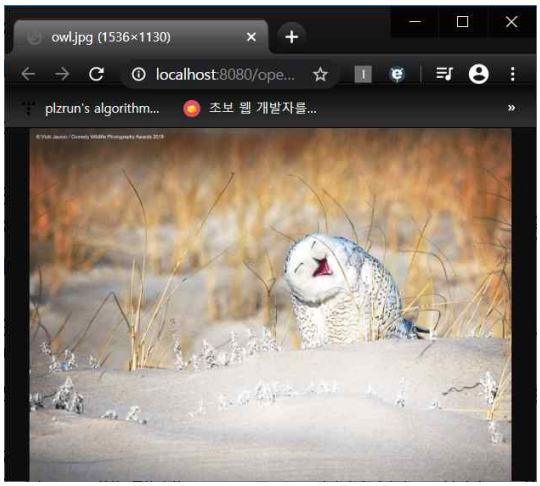
파일 업로드 테스트2

파일 업로드 테스트2

작성자	조회수	추천수	작성일
테스트	1	0	2020-06-11
내용			

첨부파일: owl.jpg

test2



파일 업로드 테스트3

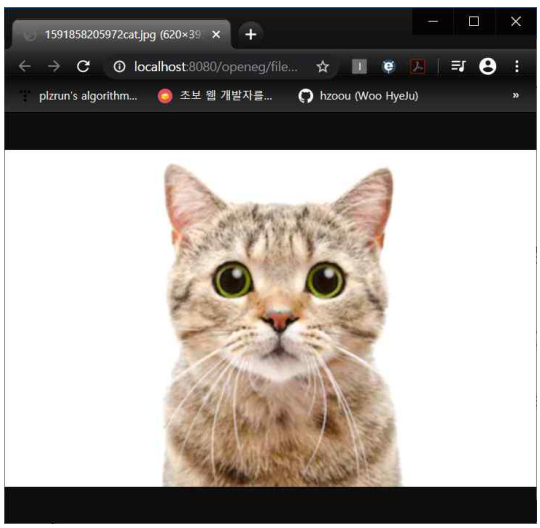
파일 업로드 테스트3

작성자	조회수	추천수	작성일
테스트	0	0	2020-06-11
내용			

첨부파일: 1591858205972cat.jpg

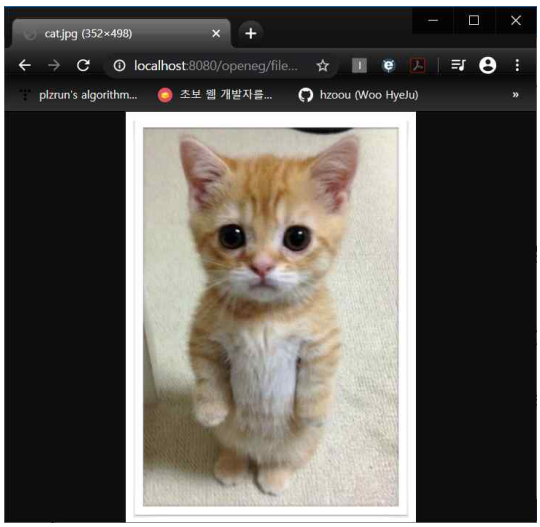
test3

앞의 'dog.jpg'와 같이 'cat.jpg'라는 이름의 파일이 이미 존재하기 때문에 시간 정보를 앞에 붙었다.



localhost:8080/openeg/files/cat.jpg

원래 존재하던 ‘cat.jpg’라는 파일을 경로 조작을 통해 확인



웹쉘 테스트

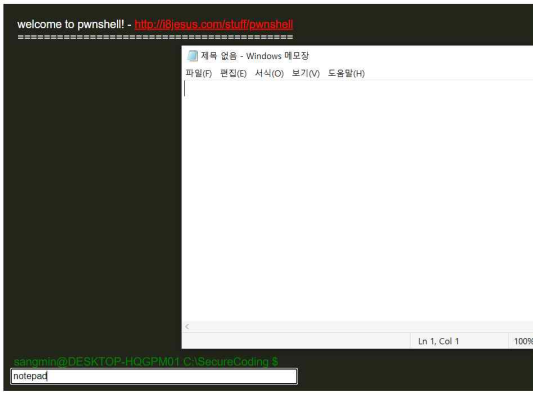
웹쉘 테스트

작성자	조회수	추천수	작성일
테스트	1	0	2020-06-11
내용			

첨부파일: pwnshell.jsp

web shell

notepad 입력 시 메모장이 켜진 것을 확인할 수 있다.



대용량파일

대용량파일

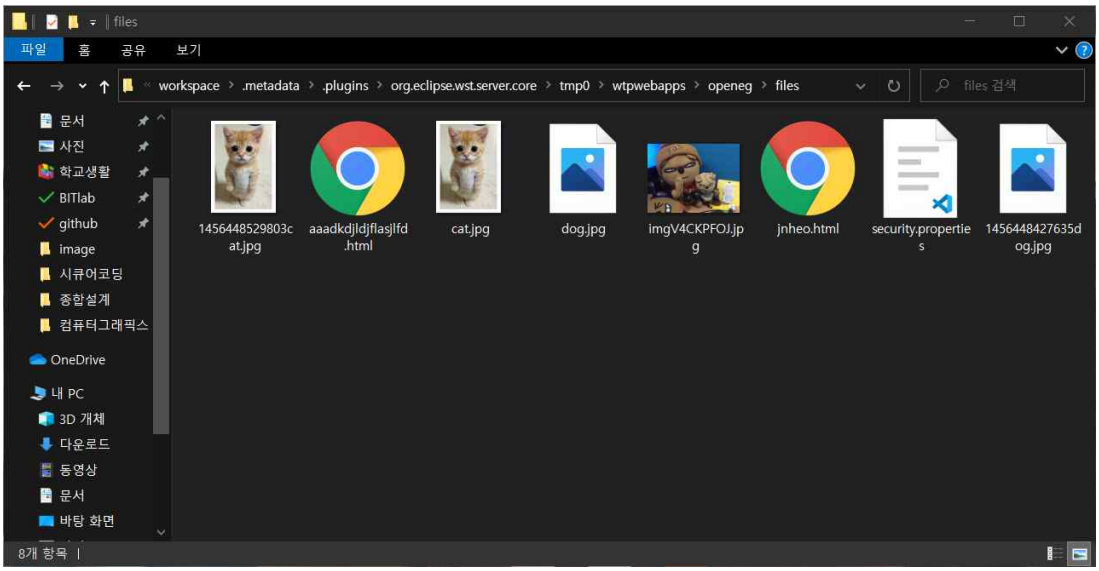
작성자	조회수	추천수	작성일
테스트	0	0	2020-06-11
내용			

첨부파일: 5mb.jpg

큰파일

대용량 사진으로 5MB 크기의 사진을 사용한 것을 알 수 있다.

(2) 방어 실험



업로드되는 파일의 타입 제한, 외부에서 직접 접근 가능하지 않은 경로에 저장

① 공격 실험에서 업로드되는 파일의 타입을 제한하지 않아 웹쉘과 같은 위험한 파일이 업로드될 수 있었다. 따라서 파일 업로드 시 파일 타입이나 확장자 검사를 통해 안전한 파일만 업로드를 허용하도록 하여야 한다.

```
//업로드 되는 파일 사이즈 제한
if ( file != null && ! "".equals(file.getOriginalFilename())
    && file.getSize() < 1024 && file.getContentType().contains("image")){
   //업로드 파일명
   String fileName = file.getOriginalFilename();
   if ( fileName.toLowerCase().endsWith(".jpg") ||
        fileName.toLowerCase().endsWith(".png")){
```

파일의 타입을 체크할 때는 getContentType() 메소드를 사용하여 파일의 MIME-TYPE 검사와 확장자 검사를 같이 해야 한다. 확장자 검사를 할 때에는 파일명을 toLowerCase() 메소드를 사용해 소문자로 변환한 후, 지정된 확장자명으로 끝나는지 endsWith() 메소드를 이용해 검사한다. 이렇게 하는 이유는 자바 환경에서는 대소문자를 구분하기 때문이다.

② 공격 실험에서 했던 것처럼 'http://localhost:8080/openeg/files/cat.jpg' 식의 URL 경로를 입력하면 files 디렉토리 내의 파일을 직접 호출할 수 있다. 파일이 저장되는 경로가 이러한 형태로 외부에서 직접 접근이 가능하면 웹쉘을 공격자가 쉽게 호출하여 실행할 수 있다. 따라서 외부에서 직접 접근하지 못하는 경로를 사용해야 한다.

```
String uploadPath = session.getServletContext().getRealPath("/")
                    +"WEB-INF/files/";
```

위의 코드는 외부에서 직접 접근 불가능한 경로에 파일이 업로드되도록 설정한다.

업로드 파일의 개수와 크기를 제한하도록 구현

업로드 파일 크기 체크는 설정 파일을 이용하거나 소스 코드에서 메소드를 사용해 설정 할 수 있다. Spring MVC에서는 MultipartResolver 객체를 등록할 때 프로퍼티 값 설정을 통해 업로드 파일 크기에 제약을 둘 수 있다.

```
<!-- 멀티파트 -->
<bean id="multipartResolver"
  class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
  <property name="maxUploadSize">
    <value>600000</value>
  </property>
  <property name="defaultEncoding">
    <value>UTF-8</value>
  </property>
</bean>
```

'5mb.jpg' 파일의 크기가 5,245,329 바이트이기 때문에 6,000,000 바이트로 maxUploadSize를 정했다.
또한 프로그램에서는 업로드되는 MultipartFile의 getSize() 메소드를 사용해 파일 크기에 제약을 둘 수 있다.

```
MultipartFile file = request.getFile("file");
//업로드 되는 파일 사이즈 제한
if ( file != null && ! "" .equals(file.getOriginalFilename())
    && file.getSize() < 1024 && file.getContentType().contains("image")){
```

업로드 파일 저장 시 랜덤하게 생성된 문자열을 파일명으로 사용

업로드된 파일을 저장하는 경우 랜덤하게 생성된 문자열을 파일명으로 이용하여 공격자가 업로드한 파일을 찾아 사용하지 못하도록 한다.

```
//저장할 파일명을 랜덤하게 생성하여 사용한다.
String savedFileName = UUID.randomUUID().toString();
File uploadFile = new File(uploadPath + savedFileName);
```

생성된 파일명이 이미 사용 중일 경우 파일명이 중복되지 않도록 처리한다.

```
//저장할 파일명을 랜덤하게 생성하여 사용한다.
String savedFileName = null;
File uploadFile = null;
do {
   savedFileName = UUID.randomUUID().toString();
   uploadFile = new File(uploadPath + savedFileName);
} while(uploadFile.exists());
```

위와 같은 코드로 실험을 해본 결과 동일한 이름의 첨부 파일이 존재할 경우 그 다음 게시글에는 첨부 파일이 등록되지 않았다.

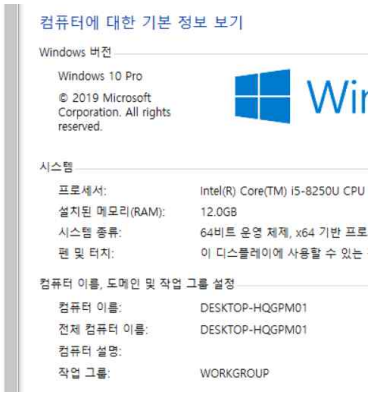
안전하게 파일 다운로드를 처리할 Controller 작성

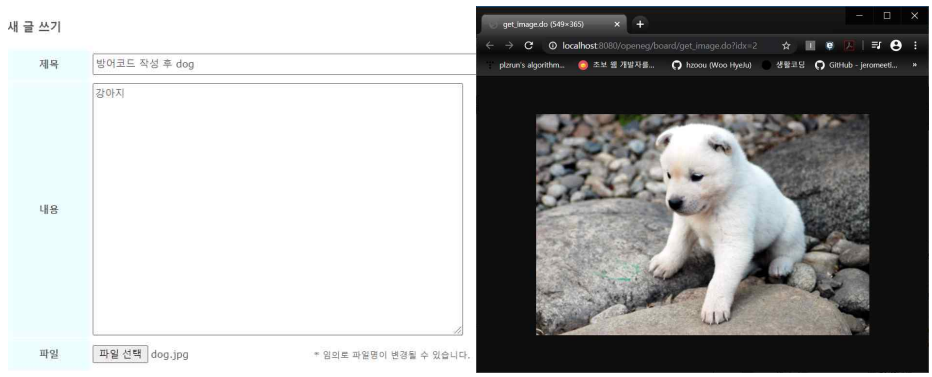
저장된 파일을 클라이언트가 요청하는 경우 파일 다운로드 기능을 제공하는 컨트롤러가 필요하다. 외부에서 접근할 수 없는 경로에 저장된 파일을 읽어 파일이 사용자에게 전송되어도 안전한지 점검하고, 사용자에게 해당 파일을 전송하는 컨트롤러를 구현한다.

결과

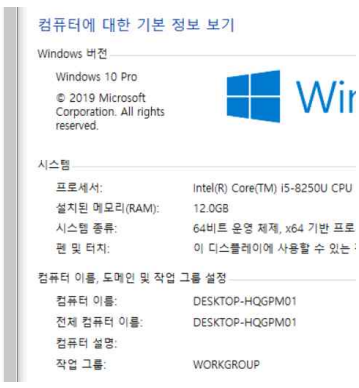

새 글 쓰기
제목
방어코드 작성 후 웹쉘
내용
web shell
파일
파일 선택
pwnshell.jsp
* 임의로 파일명이 변경될 수 있습니다.


방어코드 작성 후 웹쉘
작성자
조회수
추천수
작성일
테스트
1
0
2020-06-11
내용
web shell
댓글
댓글 쓰기
확인


컴퓨터에 대한 기본 정보 보기
Windows 버전
Windows 10 Pro
© 2019 Microsoft Corporation. All rights reserved.
시스템
프로세서: Intel(R) Core(TM) i5-8250U CPU
설치된 메모리(RAM): 12.0GB
시스템 종류: 64비트 운영 체제, x64 기반 프로
펜 및 터치: 이 디스플레이에 사용할 수 있는
컴퓨터 이름, 도메인 및 작업 그룹 설정
컴퓨터 이름: DESKTOP-HQGPM01
전체 컴퓨터 이름: DESKTOP-HQGPM01
컴퓨터 설명:
작업 그룹: WORKGROUP


새 글 쓰기
제목
방어코드 작성 후 dog
내용
강아지
파일
파일 선택
dog.jpg
* 임의로 파일명이 변경될 수 있습니다.
get_image.do (549×365)
localhost:8080/openeg/board/get_image.do?idx=2


방어코드 작성 후 dog
작성자
조회수
추천수
작성일
테스트
2
0
2020-06-11
내용
첨부파일: dog.jpg
강아지
댓글
댓글 쓰기

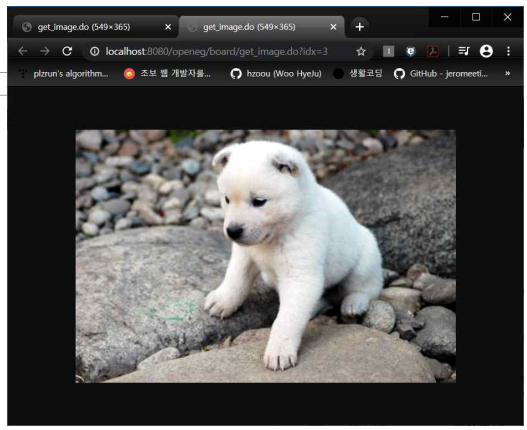

컴퓨터에 대한 기본 정보 보기
Windows 버전
Windows 10 Pro
© 2019 Microsoft Corporation. All rights reserved.
시스템
프로세서: Intel(R) Core(TM) i5-8250U CPU
설치된 메모리(RAM): 12.0GB
시스템 종류: 64비트 운영 체제, x64 기반 프로
펜 및 터치: 이 디스플레이에 사용할 수 있는
컴퓨터 이름, 도메인 및 작업 그룹 설정
컴퓨터 이름: DESKTOP-HQGPM01
전체 컴퓨터 이름: DESKTOP-HQGPM01
컴퓨터 설명:
작업 그룹: WORKGROUP

새 글 쓰기

제목 다시 강아지

내용 강아지

파일 파일 선택 dog.jpg * 임의로 파일명이 변경될 수 있습니다.



다시 강아지

작성자	조회수	추천수	작성일
테스트	1	0	2020-06-11

내용

첨부파일: dog.jpg

강아지

댓글

댓글쓰기

새 글 쓰기

제목 대용량파일

내용 대용량

파일 파일 선택 5mb.jpg * 임의로 파일명이 변경될 수 있습니다.

대용량파일

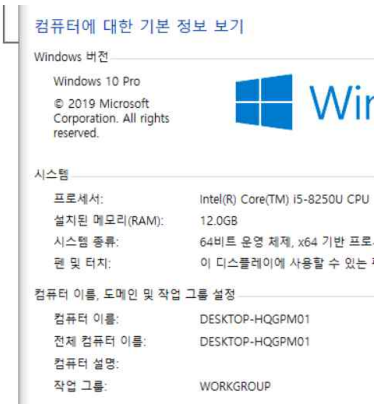
작성자	조회수	추천수	작성일
테스트	0	0	2020-06-11

내용

대용량

댓글

댓글쓰기 확인



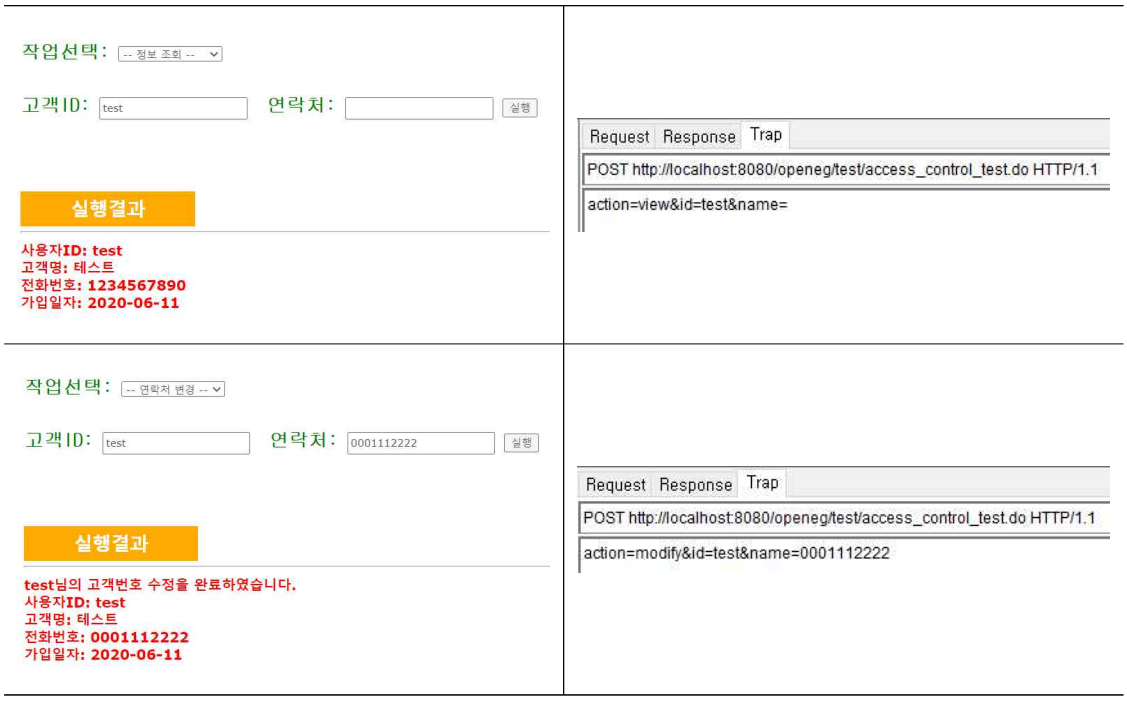
2. 파라미터 조작과 잘못된 접근제어 실습

- 네트워크 접근이 가능한 내부/외부 사용자는 모두 공격자가 될 수 있다. 인증을 통해 시스템 접근이 허락된 공격자는 URL이나 파라미터를 조작해 권한이 필요한 기능에 접근할 수 있다.

- 취약점 발생 원인
  - 적절한 인증과 인가 작업이 수행되지 않는 경우
  - 공격자가 제공한 정보에 대한 점검이 서버 상에서 수행되지 않는 경우

(1) 공격 실습

일반 사용자(test)로 로그인



관리자(admin)로 로그인

작업선택: -- 고객 정보 등록 --

고객ID: kong 고객명: 실행

실행결과

kong 사용자 등록을 완료하였습니다.
사용자ID: kong
고객명:
전화번호:
가입일자: 2020-06-11

Request Response Trap

POST http://localhost:8080/openeg/test/access_control_test.do HTTP/1.1

action=edit&id=kong&name=

작업선택: -- 고객 정보 삭제 --

고객ID: kong 고객명: 실행

실행결과

kong님의 정보를 삭제하였습니다.

Request Response Trap

POST http://localhost:8080/openeg/test/access_control_test.do HTTP/1.1

action=delete&id=kong&name=

공격자(kong)로 권한 우회

작업선택: -- 정보 조회 --

고객ID: kong 연락처: 실행

실행결과

song 사용자 등록을 완료하였습니다.
사용자ID: song
고객명: 송송송
전화번호:
가입일자: 2020-06-11

Request Response Trap

POST http://localhost:8080/openeg/test/access_control_test.do HTTP/1.1

action=edit&id=song&name=%EC%86%A1%EC%86%A1%EC%86%A1

*제목 없음 - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

action=edit&id=song&name=%EC%86%A1%EC%86%A1%EC%86%A1

song 로그인 확인

[song]님 로그아웃

[송송송]님 환영합니다.
로그아웃 정보수정

(2) 방어 실습

기능별 허가된 사용자인지 확인한 뒤 요청이 수행되도록 코드 수정

조작된 파라미터로 허가되지 않은 작업을 요청하더라도 서버 애플리케이션에서 인가 정책을 다시 확인하여 처리함으로써 취약점이 제거되도록 한다.
 	
// 실행결과 사용자에 대한 고객정보 삭제
}else if ("delete".equals(action) && "admin".equals(session.getAttribute("userId"))) {
m=(MemberModel)session.getAttribute("member");
if(m==null) {
buffer.append("사용자정보 조회부터 실행하세요");
} else {
service.deleteMember(m);
session.removeAttribute("member");
buffer.append(m.getUserId()+"님의 정보를 삭제하였습니다.");
}
// 새로운 고객 정보 등록
}else if ("edit".equals(action) && "admin".equals(session.getAttribute("userId"))) {
if(id != null && ! "".equals(id)) {
m=new MemberModel(0,id,id,name,"","");
int result=service.addMember(m);
 	
위와 같이 고객 정보를 등록하고 삭제하는 작업은 admin(관리자) 계정만 수행할 수 있도록 요청한 사용자의 정보를 확인하는 코드를 추가한다.

관리자(admin)로 로그인
작업선택: -- 정보 조회 -- 고객ID: song 고객명: 실행 실행결과 등록되지 않은 사용자입니다.
‘song’이 등록되지 않은 사용자인 것을 확인한다.

<table>
<tr><td colspan="2">공격자(kong)로 권한 우회</td></tr>
<tr><td> 
</td><td>공격 실습해서 했던 것처럼 권한을 우회하여 ‘song’을 등록해본다.</td></tr>
<tr><td> 
</td><td>request trap을 넘긴 후에도 실행 결과에 아무것도 뜨지 않는 것을 볼 수 있다.</td></tr>
<tr><td colspan="2">다시 관리자(admin)로 로그인</td></tr>
<tr><td> 
</td><td>공격자가 권한을 우회하여 새로운 사용자를 등록하는 것에 대해 성공적으로 방어한 것을 확인한다.</td></tr>
</table>

3. 안전하지 않은 리다이렉트와 포워드 실습

- 웹 애플리케이션이 사용자가 입력한 값을 사용하여 접속한 사용자를 다른 페이지로 분기시키는 기능을 가지고 있는 경우, 이동되는 목적지에 대한 검증이 제대로 이뤄지지 않는다면 피싱 사이트나 악성코드를 배포하는 사이트 등으로 사용자 정보를 넘겨 접속하거나 인가되지 않은 페이지로 접근하는 등의 문제가 발생할 수 있다.

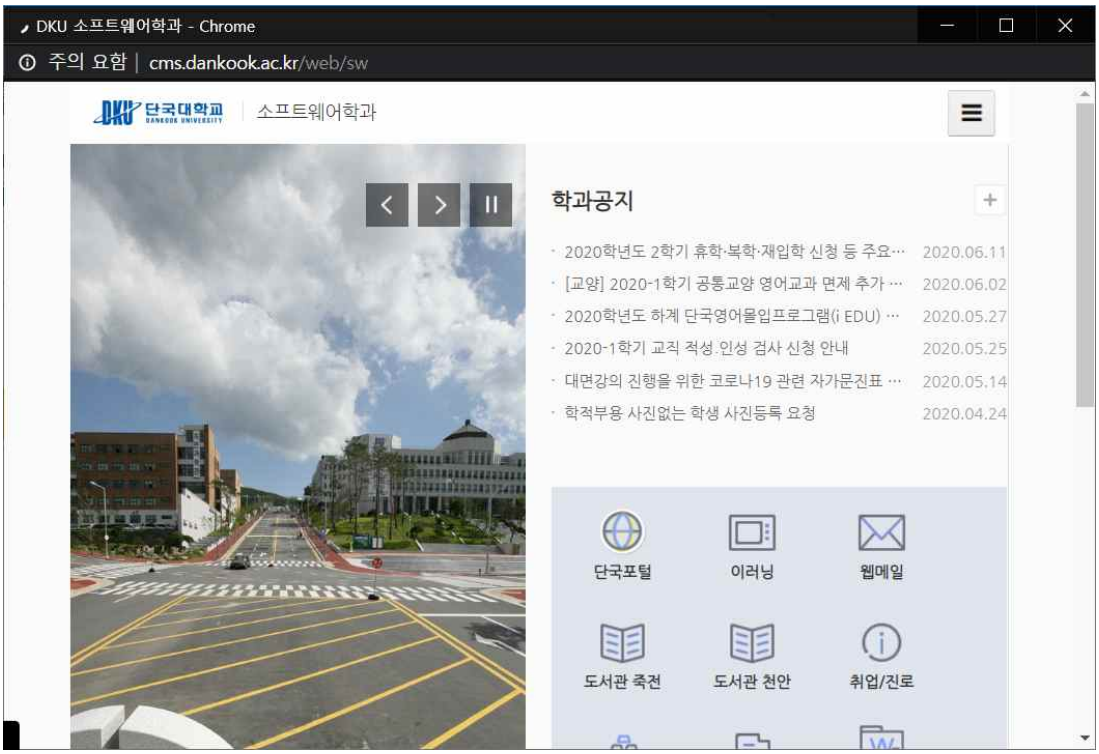
- 취약점 발생 원인
  - 웹 페이지 내 검증되지 않은 분기 페이지가 존재하는 경우
  - 외부에서 입력된 파라미터 값에 포함된 URL 정보를 직접적으로 이용하거나 해당 URL 정보가 허용된 페이지에 대한 확인 절차 없이 목적지 이동에 사용되는 경우

- 오픈 리다이렉트 취약점을 갖는 코드 예시

```
@RequestMapping(value="/test/send_redirect_test.do", method =
RequestMethod.POST)
@ResponseBody
protected void testSendRedirect(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException  {
    String data = request.getParameter("data");
     response.sendRedirect(url);
}
```

(1) 공격 실습

사이트 선택



전달되는 파라미터 값 교체

오픈리다이렉트

사이트선택 : --- SW --- 이동

실행결과

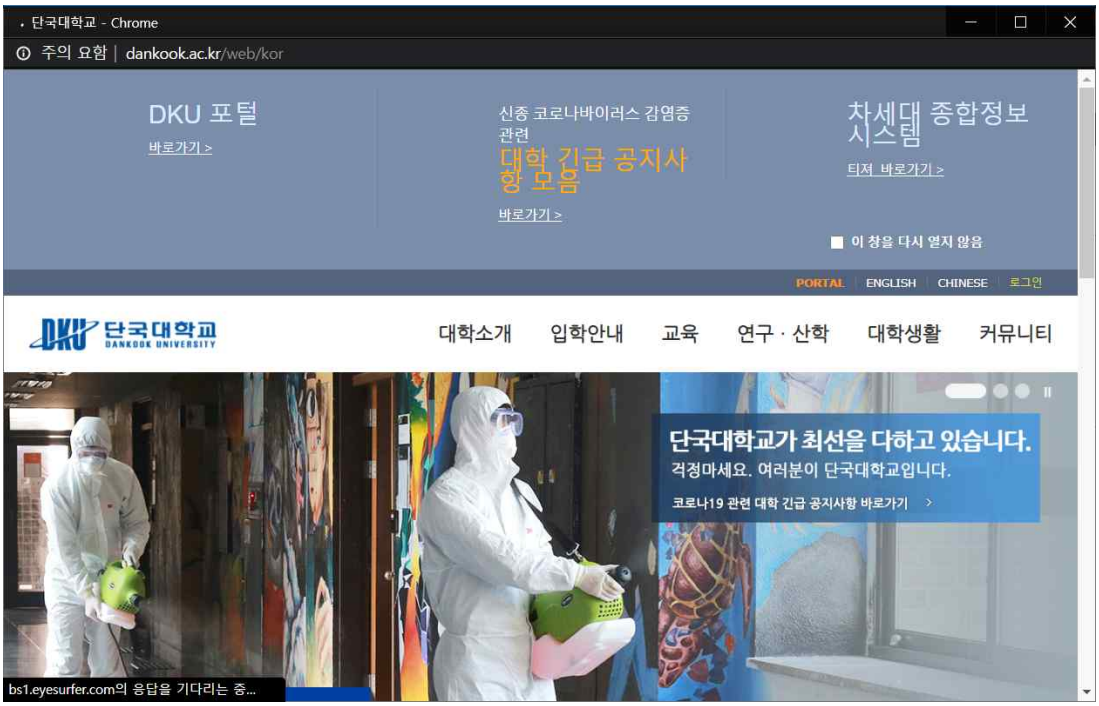
data=http%3A%2F%2Fwww.dankook.ac.kr%2Fweb%2Fkor

*제목 없음 - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

단국대학교 메인 홈페이지
http://www.dankook.ac.kr/web/kor
data=http%3A%2F%2Fwww.dankook.ac.kr%2Fweb%2Fkor

SW학과 홈페이지
http://cms.dankook.ac.kr/web/sw
data=http%3A%2F%2Fcms.dankook.ac.kr%2Fweb%2Fsw



(2) 방어 실습

리다이렉트에 사용되는 URL을 하드코딩하도록 수정

외부에서 입력된 파라미터를 사용해야 한다면, 입력값을 철저히 검증하고 분기가 허용된 URL 정보만을 사용할 수 있도록 제약을 두어야 한다.
서블릿의 sendRedirect() 메소드를 검증한 후, 분기를 지원하는 EASPI의 send Redirect() 메소드로 대체해 사용한다.

```
// 오픈리다이렉트
@RequestMapping(value="/test/forward_test.do", method = RequestMethod.POST)
@ResponseBody
public String testForwarding(HttpServletRequest request,HttpServletResponse response)
      throws ServletException, IOException {
   String allowURL[] = {"http://cms.dankook.ac.kr/web/sw",
                        "http://iacf.dankook.ac.kr",
                        "http://lib.dankook.ac.kr"};
   String url=request.getParameter("data");

   try {
      Integer n = Integer.parseInt(url);
      if (n >= 0 && n < 3) {
      response.sendRedirect(allowURL[n]);
      }
   } catch (NumberFormatException nfe) {
      return "redirect 테스트 오류";
   }
   return null;
}

사이트선택: <select name="data" id="data8">
        <option value="0">--- SW ---</option>
        <option value="1">--- 산학 ---</option>
        <option value="2">--- 도서관 ---</option>
    </select> <input type="button" value="이동" onClick="postPopUp('form8')"> <br/>
```

외부 입력값을 사용하여 이동해야 하는 경우, 이동 가능한 페이지의 URL 정보를 소스 코드에 목록화한다. 또한 위의 사진처럼 외부 입력값은 목록화된 정보 중 필요한 정보를 선택하여 사용할 수 있는 인덱스값으로 구현한다.

따라서 사용할 URL을 배열에 미리 넣어 두어 동적으로 변경할 수 없게 하고, 웹 페이지에서의 요청은 인덱스를 통해 이루어진다.

아래의 사진들처럼 data 파라미터 값이 URL 인코딩 값이 아닌 인덱스값으로 구현된 것을 볼 수 있다. 이로써 파라미터 값을 변조하는 공격에 대해 성공적으로 방어를 할 수 있다.