SQL Injection

Login.java

Statement를 이용한 동적 쿼리 실행을 PreparedStatement를 이용한 정적 쿼리 실행으로 변경

```
	public boolean login(WebSession s, String userId, String password)
	{
		// System.out.println("Logging in to lesson");
		boolean authenticated = false;

		try
		{

//			String query = "SELECT * FROM employee WHERE userid = " + userId + " and password = '" +
password + "'";
//			try
//			{
//				Statement answer_statement = WebSession.getConnection(s)
//						.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);
//				ResultSet answer_results = answer_statement.executeQuery(query);

			// #1 쿼리문의 구조를 정의
			//		변수영역을 ?로 마킹 (컬럼의 데이터 타입을 고려하지 않음)
			String query = "SELECT * FROM employee WHERE userid = ? and password = ? ";
			try
			{
				// #2 Statement -> PreparedStatement
				java.sql.PreparedStatement answer_statement = WebSession.getConnection(s)
						.prepareStatement(query, ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);

				// #3 변수 영역에 값을 바인딩 (컬럼의 데이터 타입을 고려)
				answer_statement.setInt(1, Integer.parseInt(userId));
				answer_statement.setString(2, password);

				// #4 실행
				ResultSet answer_results = answer_statement.executeQuery();
				if (answer_results.first())
				{
					setSessionAttribute(s, getLessonName() + ".isAuthenticated", Boolean.TRUE);
					setSessionAttribute(s, getLessonName() + "." + SQLInjection.USER_ID, userId);
					authenticated = true;
				}
			} catch (SQLException sqle)
			{
				s.setMessage("Error logging in");
				sqle.printStackTrace();
			}
		} catch (Exception e)
		{
			s.setMessage("Error logging in");
			e.printStackTrace();
		}

		// System.out.println("Lesson login result: " + authenticated);
		return authenticated;
	}
```

#과 $의 차이

```
pid >>> 123
ppw >>> abc' or 'a' = 'a
select * from user where id = #pid# and pw = '$ppw$'
>>> … where id = ? and pw = 'abc' or 'a' = 'a'
```

```
pid >>> 123
sharp >>> #
select * from users where id# = #pid#
>>> select * from users where id$sharp$ = #pid#
>>> select * from users where id# = ?
```

login.xml

iBatis, myBatis와 같은 ORM 프레임워크를 사용하는 경우, 외부 입력값을 쿼리 맵에 바인딩할 때는 반드시 #을 이용

```
    <select id="loginCheck2" parameterClass="LoginModel"
          resultClass="LoginModel">
          select
          idx,
          userId,
          userPw,
          userName,
          joinDate
          from board_member
          <!-- 외부 입력값을 $ 기호를 이용해서 바인딩하면 SQL 삽입이 발생
          where userId = '$userId$' and userPw = '$userPw$'
          -->
          where userId = #userId# and userPw = #userPw#
```

```
    </select>
```

UNION Based SQL Injection

[하대동] 검색

```
>>> .../zip_search.jsp?dong=하대동
>>> select * from address where dong = '하대동'


#1. 정상 쿼리의 컬럼 개수를 확인
select * from address where dong = '하대동' order by 1 --'
select * from address where dong = '하대동' order by 8 --'
>>> 오류 발생 >>> 컬럼 개수가 7개

#2. 정상 쿼리의 실행 결과가 없도록 …
… where dong = 'a' and 1 = 2 --'

#3. DBMS 정보를 조회
… where dong = 'a' and 1 = 2 union select
@@version,2,3,4,5,6,7 --'

… where dong = 'a' and 1 = 2 union select 1,
@@version,3,4,5,6,7 --'

#4. sysobjects 테이블을 이용해서 사용자 정의 테이블을 조회
```

```
… where dong = 'a' and 1 = 2 union select
1,name,3,4,5,6,7 from sysobjects where xtype='U' --'

#5. syscolumns 테이블을 이용해서 member 테이블의 컬럼 정보를
조회
… where dong = 'a' and 1 = 2 union select
1,name,3,4,5,6,7 from syscolumns where id=(select id from
sysobjects where name='member') --'

#6. member 테이블에 정보를 조회
… where dong = 'a' and 1 = 2 union select
1,bname,bid,bpass,bphone,bmail,7 from member --'
```

Blind SQL Injection

```
Account Number : 101
>>> …/check.jsp?account=101
>>> select * from account wherer account_number = 101
>>> 일치하는 정보가 존재 >>> Account number is valid.

Account Number : 999
>>> …/check.jsp?account=999
>>> select * from account wherer account_number = 999
>>> 일치하는 정보가 존재하지 않다. >>> Invalid account
number.

Account Number : 101 and 1=1
>>> …/check.jsp?account=101 and 1=1
```

```
>>> select * from account wherer account_number = 101 and
1=1
>>> 일치하는 정보가 있다 ⇐ Account number is valid.

Account Number : 101 and 1=2
>>> …/check.jsp?account=101 and 1=2
>>> select * from account wherer account_number = 101 and
1=2
>>> 일치하는 정보가 없다 ⇐ Invalid account number.

select * from account wherer account_number = 101 and
(select pin from pins where cc_number =
'1111222233334444') = 2364
```

Command Injection(운영체제 명령어 삽입)

원인

1) 운영체제 명령어 실행부분이 존재할 때
   Runtime.getRuntime().exec()
2) 외부 입력값이 운영체제 명령어 실행 또는 실행의 일부로 사용되는 경우
   String input = request.getParameter("cmd");
   exec(input);
   exec("cmd.exe /c dir c:\data\"+input);
3) 외부 입력값을 검증 또는 제한하지 않았을 때 발생

예상피해

1) 의도하지 않은 시스템 명령어가 실행
2) 해당 서버를 존비화하여 원격지에서 제어

방어

1) 운영체제 명령어 실행 부분이 필요한지 여부를 확인하고, 해당 기능을 삭제 또는 다른 기능으로 대체한다.
2) 사용할 명령어를 미리 정의하여 정의된 목록 범위에서 사용될 수 있도록 한다. (화이트 리스트 방식으로 입력값 제한)

test.jsp

```
<div class="hint" id="div4">
      <form action="command_test.do" id="form5">
      <pre>
      (4) Command 인젝션  <br />
           <select name="data" id="data5">
           <%--
            서버로 전달되는 인자값이 서버에서 사용하는 명령어를 그대로 전달하고 있음
            코드화를 통해 서버 내부 처리를 알 수 없도록 캡슐화
             <option value="type">--- show File1.txt ---</option>
             <option value="dir">--- show Dir ---</option>
            --%>
             <option value="0">--- show File1.txt ---</option>
             <option value="1">--- show Dir ---</option>
       </select> <input type="button" id="button5" value="실행">
      </pre>
      </form>
</div>
```

TestController.java

```
      @RequestMapping(value = "/test/command_test.do", method = RequestMethod.POST)
      @ResponseBody
      public String testCommandInjection(HttpServletRequest request, HttpSession session) {

            // 해당 어플리케이션에서 사용할 명령어를 정의
            String[] allowedCommands = { "type", "dir" };

            StringBuffer buffer = new StringBuffer();
            String data = request.getParameter("data");
```

```
            // 코드로 전달되는 값을 실행할 명령어로 대체
            try {
                  data = allowedCommands[Integer.parseInt(data)];
            } catch (Exception e) {
                  // 0 또는 1이 아닌 다른 숫자가 전달되는 경우 >>> 배열 크기 초과
                  // 숫자가 아닌 문자가 전달되는 경우 >>> 파싱 오류
                  return "정상적인 요청이 아닙니다.";
            }

            if (data != null && data.equals("type")) {
                  data = data + " " + request.getSession().getServletContext().getRealPath("/") +
"files\\file1.txt";
            }

            Process process;
            String osName = System.getProperty("os.name");
            String[] cmd;

            if (osName.toLowerCase().startsWith("window")) {
                  cmd = new String[] { "cmd.exe", "/c", data };
                  for (String s : cmd)
                        System.out.print(s + " ");
            } else {
                  cmd = new String[] { "/bin/sh", data };
            }
            try {
                  process = Runtime.getRuntime().exec(cmd);
                  InputStream in = process.getInputStream();
                  Scanner s = new Scanner(in);
                  buffer.append("실행결과: <br/>");
                  while (s.hasNextLine() == true) {
                        buffer.append(s.nextLine() + "<br/>");
                  }
            } catch (IOException e) {
                  buffer.append("실행오류발생");
                  e.printStackTrace();
            }
            return buffer.toString();
      }
```

XSS(Cross-Site Script; 크로스사이트 스크립트)

공격자가 전달한 스크립트 코드가 사용자 브라우저에서 실행

~~~~~~~~~~~~~~~

1) 브라우저의 정보를 빼돌림
2) 가짜 페이지를 생성하고, 사용자의 입력을 유도
~~~~~~~~~~~~~~~

3) 원격에서 해당 브라우저를 조정

BeEF ⇒ 대표적인 XSS 공격 프레임워크

Reflective XSS (반사)

사용자 입력이 다음 화면 출력으로 사용되는 경우
스크립트 코드를 입력하면 다음 화면에서 스크립트가 실행
공격자가 전달한 스크립트 코드가 서버에 저장되지 않고, 취약한 서버를 경유하여 사용자에게 전달되는 구조
예)
ID 찾기 : abc ----------> search.jsp?id=abc

"abc" 존재… <----------
"<%=request.getParameter("id")%>" 존재 ...

공격문자열
<a href="search.jsp?id=<script>...</script>">김혜수...

Stored XSS (저장)

공격자가 전달한 스크립트 코드가 서버에 저장되고, 저장된 스크립트 코드가 서버를 통해서 클라이언트에게 전달되어 실행되는 구조
예) 게시판

DOM XSS

개발자가 작성한 DOM write 스크립트 구문을 이용하여 공격자의 스크립트를 실행

스크립트를 실행하는 방법(유형)

```
<script> …. </scirpt>
<script src="..."></script>
<img src="" onerror="javascript:...">
express(...)
url(...)

<iframe src="alert('xss');"></iframe>
```

대응방법

1) 입력값 검증
   a) 전달되는 내용이 데이터 구조와 일치하는지를 검증
   b) 실행 가능한 코드 포함 여부를 확인
   c) 허용하지 않는 데이터 또는 문자열인 경우 해당 입력을 거부하거나 안전한 문자열로 대체해서 전달
2) 출력값 인코딩
   a) 안전하지 않은 문자열이 포함된 경우 HTML 인코딩 후 브라우저로 전달
3) DOM에 write하는 경우, 스크립트 코드가 포함된 경우 HTML 인코딩 처리하거나, 해당 코드가 실행되지 않도록 텍스트 노드에 write

CSRF(Cross-Site Request Forgery; 크로스사이트 요청위조)

원인 및 대응방법

서버에서 요청에 대한 검증을 불충분하게 한 경우 발생
  1) 요청 주체에 대한 검증 >>> 주요 기능에 대해 재인증,재인가
  2) 요청 절차에 대한 검증 >>> 토큰 검증

```
"789" ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~>

패스워드 변경 요청 페이지      패스워드 변경 처리 페이지
changePwForm.jsp    =======> changePwProc.jsp

New PW: ________              1) 인증 여부 확인
                              2) 세션에서 ID를 추출
                              3) UPDATE oldpw <- newpw
<input type="hidden"
       name="token"
       value="789">

[제목] 김혜수….
       :
<iframe src="changePwProc.jsp?newpw=123" width="0"
height="0"></iframe>
       :
```

CSRF를 이용한 공격 예

게시판에 자동으로 글쓰기가 가능한 확인
자동으로 글쓰기할 수 있도록 코드를 만들어서 넣어보겠ㅅㅂ니다.

게시판에 아래 코드를 입력하고 저장

```
<form action="write.do" method="post" enctype="multipart/form-data">
<input type="text" id="subject" name="subject" class="boardSubject" size="70" value="대출상품안내!!!!"
/>
<input type="hidden" id="writer" name="writer" value="관리자" />
<input type="hidden" id="writerId" name="writerId" value="admin" />
<textarea id="boardContent" name="content" class="boardContent">직장인을 위한 저렴한 대출 상품
안내입니다</textarea>
<input type="submit" value="확인" id="writeBt" />
</form>

<script> document.getElementById("writeBt").click(); </script>
```

저장된 내용을 확인 >>> 입력한 코드가 실행되고 >>> 서버의 write.do가 입력값에 대한 검증을 하지 않으면 >>> 자동으로 글쓰기 처리

토큰 검증 방식으로 정상적인 처리 여부를 확인

1) 글쓰기 페이지 요청이 들어오면, 토큰을 생성 => 세션에 저장
   write.do / GET ⇒ write.jsp

2) 토크을 글쓰기 페이지 내려보낸다.

3) 글쓰기 처리 페이지(write.do/POST)에서 파라미터로 전달된 토큰과 세션에 저장된 토큰을 비교해서 일치하는 경우에만 저장 처리를 한다.
일치하지 않으면 list.do 페이지로 리다이렉트 한다.

글쓰기 요청 ==========> 글쓰기 처리
제목
내용
:

[저장]

게시판에 존재하는 CSRF 취약점 제거

write.jsp

```
            <div id="content-container">
                  <div id="content">
                        <h3>새 글 쓰기</h3>
                        <form action="write.do" method="post" onsubmit="return writeFormCheck()"
enctype="multipart/form-data">


                                    <%-- 서버에서 생성한 토큰을 히든 필드의 값으로 지정 --%>
                                    <input type="hidden" name="p_token" value="${s_token}" />
```

BoardController.java

```
      // 글쓰기 페이지 요청
```

```
	@RequestMapping("/write.do")
	public String boardWrite(@ModelAttribute("BoardModel") BoardModel boardModel, HttpSession
session) {
		// #1. 유출할 수 없는(= 랜덤 형태) 토큰을 생성
		String token = UUID.randomUUID().toString();

		// #2. 토큰을 세션에 저장 s_token
		session.setAttribute("s_token", token);

		return "/board/write";
	}

	@RequestMapping(value = "/write.do", method = RequestMethod.POST)
	public String boardWriteProc(@ModelAttribute("BoardModel") BoardModel boardModel,
MultipartHttpServletRequest request, HttpSession session) {

		// #4. 사용자 화면에서 파라미터로 전달된 토큰과
		//	서버 세션에 저장된 토큰이 일치하는지 확인
		//	일치 >>> 기존의 저장 처리
		//	불일치 >>> list.do 리다이렉트

		String sToken = (String)session.getAttribute("s_token");
		String pToken = request.getParameter("p_token");

		if (pToken == null || !pToken.equals(sToken)) {
			return "redirect:list.do";
		}




		String uploadPath = session.getServletContext().getRealPath("/") + "files/";
		File dir = new File(uploadPath);
		if (!dir.exists()) {
			dir.mkdir();
		}

		MultipartFile file = request.getFile("file");
		if (file != null && !"".equals(file.getOriginalFilename())) {
			String fileName = file.getOriginalFilename();
			File uploadFile = new File(uploadPath + fileName);
			if (uploadFile.exists()) {
				fileName = new Date().getTime() + fileName;
				uploadFile = new File(uploadPath + fileName);
			}

			try {
				file.transferTo(uploadFile);
			} catch (Exception e) {
				System.out.println("upload error");
			}

			boardModel.setFileName(fileName);
		}

		String content = boardModel.getContent().replaceAll("\r\n", "<br />");
		boardModel.setContent(content);
```

```
            service.writeArticle(boardModel);

            return "redirect:list.do";
    }
```

파일 업로드 취약점

원인

1) 업로드 기능이 존재할 때
   <form method="post" enctype="multipart/form-data">
   <input type="file" … >

   MultipartHttpServletRequest
   MultipartFile
2) 업로드 파일의 크기와 개수를 제한하지 않고
3) 업로드 파일의 종류를 제한하지 않고
   a) 파일 확장자
   b) Content-Type
   c) File Signature
4) 업로드 파일을 외부에서 접근 가능한 경로에 저장

예상피해

1) 큰 크기의 파일 또는 여러 파일을 업로드하여 서버의 가용성을 떨어뜨리는 공격을 시도한다.
2) 서버에서 실행 가능한 파일을 업로드하여 서버를 공격한다. (대표적인 예: 웹쉘)
3) 바이러스와 같은 악성 파일을 업로드한 후 불특정 다수가 내려받고 피해를 입도록 한다.

대응방안

1) 업로드 파일의 크기와 개수를 제한한다.
2) 업로드 파일의 종류를 제한한다.
3) 업로드 파일을 외부에서 접근할 수 없는 경로에 저장한다.
4) 업로드 파일의 저장 경로와 파일명을 외부에서 알 수 없도록 한다.
5) 업로드 파일의 실행 속성을 제거하고 저장한다.

파일 업로드 취약점 방어

BoardController.java

```
	@RequestMapping(value = "/write.do", method = RequestMethod.POST)
	public String boardWriteProc(@ModelAttribute("BoardModel") BoardModel boardModel,
MultipartHttpServletRequest request, HttpSession session) {

		// #4. 사용자 화면에서 파라미터로 전달된 토큰과
		//	서버 세션에 저장된 토큰이 일치하는지 확인
		//	일치 >>> 기존의 저장 처리
		//	불일치 >>> list.do 리다이렉트
		String sToken = (String)session.getAttribute("s_token");
		String pToken = request.getParameter("p_token");

		if (pToken == null || !pToken.equals(sToken)) {
			return "redirect:list.do";
		}



		// #1 업로드 파일의 저장 경로를 외부에서 접근할 수 없는 곳으로 변경
		// String uploadPath = session.getServletContext().getRealPath("/") + "files/";
		String uploadPath = session.getServletContext().getRealPath("/") + "WEB-INF/files/";
		File dir = new File(uploadPath);
		if (!dir.exists()) {
			dir.mkdir();
		}

		MultipartFile file = request.getFile("file");

		// #2 파일의 크기를 제한
		if (file.getSize() > 2048000)
			return "redirect:list.do";

		if (file != null && !"".equals(file.getOriginalFilename())) {
			String fileName = file.getOriginalFilename();
```

```
			// #3 파일 확장자 검증을 통해서 이미지 파일만 업로드될 수 있도록 수정
			if (fileName.endsWith(".png") || fileName.endsWith(".jpg")) {

				// #4 저장 파일명을 외부에서 알 수 없도록 변경
				String savedFileName = UUID.randomUUID().toString();

				File uploadFile = new File(uploadPath + savedFileName);
				if (uploadFile.exists()) {
					fileName = new Date().getTime() + fileName;
					uploadFile = new File(uploadPath + fileName);
				}

				try {
					file.transferTo(uploadFile);
				} catch (Exception e) {
					System.out.println("upload error");
				}

				boardModel.setFileName(fileName);	// 원본 파일명
				boardModel.setSavedFileName(savedFileName);	// 저장에 사용한 파일명
			}
		}

		String content = boardModel.getContent().replaceAll("\r\n", "<br />");
		boardModel.setContent(content);
		service.writeArticle(boardModel);

		return "redirect:list.do";
	}
```

파일 업로드 취약점 방어 ⇒ 다운로드 기능이 필요

파일 다운로드 기능 구현

BoardController.java

```
	@RequestMapping("/view.do")
	public ModelAndView boardView(HttpServletRequest request) {
		int idx = Integer.parseInt(request.getParameter("idx"));
		BoardModel board = service.getOneArticle(idx);

		//	게시판 내용을 파일 다운로드에서 참조할 수 있도록 세션에 저장
		request.getSession().setAttribute("board", board);

		service.updateHitcount(board.getHitcount() + 1, idx);

		List<BoardCommentModel> commentList = service.getCommentList(idx);
```

```
		ModelAndView mav = new ModelAndView();
		mav.addObject("board", board);
		mav.addObject("commentList", commentList);
		mav.setViewName("/board/view");
		return mav;
	}


	@RequestMapping("/get_image.do")
	public void getImage(HttpServletRequest request, HttpSession session, HttpServletResponse
response) {
		// 세션에 저장되어 있는 게시판 내용을 조회
		BoardModel board = (BoardModel)session.getAttribute("board");

		// 세션에 저장되어 있는 원본 파일명과 파라미터로 넘어온 원본 파일명을 비교
		// 일치 ==> 정상적인 요청
		String filename = request.getParameter("filename");
		String sFileName = board.getFileName();

		if (filename == null || !filename.equals(sFileName))
			return;

		// 세션으로부터 저장된 파일명을 추출
		String savedFileName = board.getSavedFileName();

		String filePath = session.getServletContext().getRealPath("/") + "WEB-INF/files/" +
savedFileName;
		System.out.println("filename: " + filePath);
		BufferedOutputStream out = null;
		InputStream in = null;
		try {
			response.setContentType("image/jpeg");
			response.setHeader("Content-Disposition", "inline;filename=" + filename);
			File file = new File(filePath);
			in = new FileInputStream(file);
			out = new BufferedOutputStream(response.getOutputStream());
			int len;
			byte[] buf = new byte[1024];
			while ((len = in.read(buf)) > 0) {
				out.write(buf, 0, len);
			}
		} catch (Exception e) {
			e.printStackTrace();
			System.out.println("파일 전송 에러");
		} finally {
			if (out != null)
				try {
					out.close();
				} catch (Exception e) {
				}
			if (in != null)
				try {
					in.close();
				} catch (Exception e) {
				}
		}
```

```
    }
```

view.jsp

```
                        <c:if test="${board.fileName != null}">
                            <tr>
                                <td colspan="4" align="left">
                                    첨부파일 :
                                    <br />이전방식
                                    <br /><a href="../files/${board.fileName}"
target="_blank">${board.fileName}</a>
                                    <br /><img src="../files/${board.fileName}" />
                                    <br />다운로드 기능을 이용
                                    <br /><a
href="get_image.do?filename=${board.fileName}" target="_blank">${board.fileName}</a>
                                    <br /><img
src="get_image.do?filename=${board.fileName}" />
                                </td>
                            </tr>
                        </c:if>
```

잘못된 기능 접근 제어

접근제어가 잘못 구현되었을 때 발생

~~~~~~~

1) 화면 : 권한 없는 사용자에게 기능 버튼이나 링크를 제공하지 않는 것
2) 기능 : 권한 없는 사용자의 요청을 처리하지 않는 것
3) 데이터

목록 페이지 상세 페이지

list view / detail

~~~~~~~~~~~ view.jsp?id=000 번호 : ----

~~~~~~~~~~~ ==========> 제목 : ----

~~~~~~~~~~~ 유일키 내용 : ----

```
select *                              select *
  from data                             from data
 where xxxx and                        where id = OOO
       xxxx and
       xxxx and
```

HTTP 응답 분할

HTTP 프로토콜의 특징을 이용한 공격

~~~~~~~~~~~~~~~~~~~

1) 요청/응답 구조
2) Stateless

요청 구조

시작줄 >>> 방식(Method) URL(URI) 프로토콜/버전
요청헤더 >   :

요청본문 > (방식에 따라서 존재 유무가 결정)

방식: GET/POST, PUT, DELETE, OPTIONS, HEAD, …
요청헤더와 요청본문은 한줄 띄우는 것으로 구분

예)

```
POST /home/abc.jsp HTTP/1.1
Cookie: role=user;
    :
```
~~~~~~~~~~~~~~~~~~~

name=xyz&age=12

응답 구조

시작줄 >>> 프로토콜/버전 상태코드 상태코드설명
응답헤더 > :

응답본문 > <html>
<head>....

예)
HTTP/1.1 200 OK
Set-Cooke: role=user;
  :

<html>
<head>...</head>
<body>...</body>
</html>

프로그램 로직 중 응답헤더에 값을 쓰는 경우
값이 외부에서 들어오는 경우
값에 개행문자(CR, LF)가 포함되어 있는 경우
⇒ 응답이 여러개로 나눠져서(분리되어) 브라우져에 전달되는 것
⇒ HTTP 응답 분할

```
HTTP/1.1 200 OK
Set-Cooke: role=user;
    ....

<html><head><script> …
</script></head><body>...</body></html>
HTTP/1.1 200 OK
    ....
     :

<html>
<head>...</head>
<body>...</body>
</html>
```

방어기법

외부 입력값을 **응답헤더의 값으로 사용하는 경우** 개행문자 포함 여부를 확인하고, 제거 후 사용한다.

1) addCookie()

```
String name = request.getParameter("name");
Cookie c = new Cookie();
c.setCookie("role", name);
response.addCookie(c);
```

```
    >>> Set-Cookie: role=OOO;
  2)  sendRedirect()
    String url = request.getParameter("url");
    response.sendRedirect(url);

    >>> location: OOO;
  3)  setHeader()
    String value = request.getParameter("value");
    response.setHeader("value", value);

    >>> value: OOO;
```

```
String filename = request.getParameter("filename");
String filePath =
session.getServletContext().getRealPath("/") +
"WEB-INF/files/" + savedFileName;

BufferedOutputStream out = null;
InputStream in = null;
try {
    response.setContentType("image/jpeg");
    response.setHeader("Content-Disposition",
"inline;filename=" + filename);
            :
```

입력값 검증 절차

정규화/규범화 → 새니타이즈 → 검증

주요 기능에 사용되는 입력값은 사용자 입력에 의존하지 않는다.
입력 → 처리 → 출력
~~~~
  1) 신뢰할 수 있는 데이터 = 서버가 가지고 있는 값
  2) 신뢰할 수 없는 데이터 = 사용자 입력값
⇒ 사용자 입력값은 쉽게 위변조될 수 있기 때문에 항상 서버에서 검증 후 사용해야 한다.
⇒ 중요한 기능에 대해서는 사용자 입력을 사용하지 않도록 처리를 구성해야 한다.

## NullPointer 역참조

Null 객체를 참조했을 때 발생

오류
- 명시적 오류
```
String s = null;
if (s.equals("abc")) …
```

- 암시적 오류 = 런타임 오류
```
String s = request.getParameter("name");
if (s.equals("abc")) … ⇒ NullPointerException
if (s != null && s.equals("abc"))
```

```
/* Tomcat 4.1.2 BUG */
public static int cardinality (Object obj, final Collection col) {
   int count = 0;
```
~~~~

```
    if (col == null) {
        return count;
    }
    Iterator it = col.iterator();
    while (it.hasNext()) {
        Object elt = it.next();
        // obj가 null이고, elt가 null이 아닐 때 NullPointerException이 발생
        if ((null == obj && null == elt) || obj.equals(elt)) {
            count++;
        }
    }
    return count;
}
```

코드의 품질 측정

1) 동적분석 : 실행 O => 원하는 결과, 성능, 안정성
2) 정적분석 : 실행 X

http://www.kisa.or.kr/public/laws/laws3.jsp
공개SW를 활용한 소프트웨어 개발보안 검증가이드

오류 상황 대응 부재

오류 처리

1) 오류 메시지에 많은 정보가 노출되면 안 된다.
2) 발생한 오류에대한 처리가 없을 때
3) 오류를 세분화

```
/* org.owasp.webgoat.lessons.FailOpenAuthentication */
protected Element createContent(WebSession s) {
```

```
    :
  try {
     username = request.getParameter("userName");
     password = request.getParameter("password");
     if (!"webgoat".equals(username) || !password.equals("webgoat")) {
        s.setMessage("Invalid username and password entered.");
        return (makeLogin(s));
     }
  } catch (Exception e) {
     //s.setMessage(e.getMessage());
     //return (makeLogin(s));
  }
  // authentication is good, show the content
     :
}
```

캡슐화
= 데이터 은닉

Public 메소드로부터 반환되는 Private 배열

안전하지 않은 코드의 예

안전한 코드의 예

```
private String[] colors;
public String[] getColors() { return
colors; }
```

•public으로 선언된 getColors()로 참조

```
private String[] colors;
public void onCreate(Bundle
savedInstanceState) {
  super.onCreate(savedInstanceState);
  String[] newColors = getColors();
}
public String[] getColors() {
  String[] ret = null;
  if (this.colors != null) {
      ret = new String[colors.length];
      for (int i = 0; i < colors.length;
i++) {
      ret[i] = this.colors[i];
      }
  }
  return ret;
}
```

•private 배열의 복사본을 만들어서 수정 방지

Private 배열에 Public 데이터 할당

안전하지 않은 코드의 예

안전한 코드의 예

```
private String[] userRoles;
public void setUserRoles(String[]
userRoles)
{
      this.userRoles = userRoles;
}
```

•userRoles 필드는 private이지만, public인 setUserRoles()를 통해 외부의 배열이 할당되면, 사실상 public 필드가 됨

```
private String[] userRoles;
public void setUserRoles(String[]
userRoles)
{
      this.userRoles = new
String[userRoles.length];
    for (int i = 0; i < userRoles.length;
++i) {
            this.userRoles[i] =
userRoles[i];
    }
                :
```

•입력된 배열의 reference가 아닌, 배열의 "값"을 private 배열에 할당하여 private 멤버로서의 접근권한을 유지

```
<%  String name; %>
<%! String name; %>
```

download.jsp ⇒ download_jsp.java