Coding Convention & Secure Coding

201203146 임승한

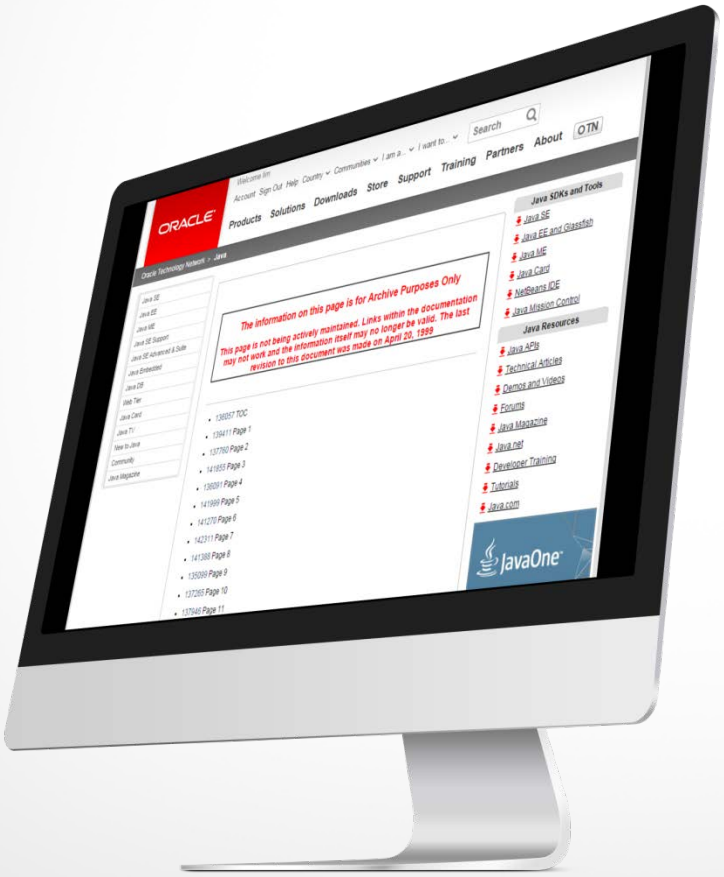
인천대학교 컴퓨터공학부
INCHEON NATIONAL UNIVERSITY
Dept. of Computer Science & Engineering

CONTENTS

Coding Convention

1. 사용 목적 – Java Coding Convention

1.1 Coding Convention



1. 소프트웨어 조각의 수명 비용의 80 % 유지

2. 코드 규칙 엔지니어가보다 신속하고

   철저하게 새로운 코드를 이해 할 수 있도록

   소프트웨어의 가독성 향상

출처 :
http://www.oracle.com/technetwork/java/javase/documentation/codeconventions-139411.html

2. 파일 이름 – Java Coding Convention

2.1 파일 접미사

파일 형식	접미사
자바 소스	.java
자바 바이트 코드	.class

2.2 일반적인 파일 이름

GNUmakefile	• 메이크 파일에 대한 선호하는 이름
README	• 특정 디렉토리의 내용을 요약 한 파일에 대한 선호하는 이름

3. 파일 구성 – Java Coding Convention

- 파일은 빈 줄과 각 섹션을 식별하는 선택적 주석으로 구분해야 섹션으로 구성된다.
- 2000 선 이상 파일은 복잡하고 피해야한다.

3.1 Java 소스 파일의 섹션 구분 순서

- 시작 코멘트
- 패키지 및 가져오기 문
- 클래스와 인터페이스 선언

3. 파일 구성 – Java Coding Convention

3.1.1 Beginning Comments

```
/ *
* 클래스 이름
*
* 버전 정보
*
* 날짜
*
* 저작권
* /
```

3.1.2 Package and Import Statements

```
package java.awt;

import java.awt.peer.CanvasPeer;
```

4. 들여쓰기 – Java Coding Convention

- 4 칸 들여 쓰기의 단위로 사용되어야 한다.
- 들여 쓰기 (스페이스 대 탭)의 정확한 구성은 지정되지 않는다.

4.1 선 길이(Line Length)

- 많은 터미널과 툴에 의해 잘 처리되지 않기 때문에, 80자 이내로 작성할 것

4.2 포장 라인(Wrapping Lines)

- 쉼표 후 휴식
- operator 전에 휴식
- 만약 위의 규칙들이 margin과 code의 혼란을 야기한다면
  8 공간의 들여쓰기로 대체해도 무방

5. 댓글 – Java Coding Convention

5.1 구현 주석 형식

(1) block (2) single-line (3) trailing (4) end-of-line

5.1.1 Block Comments

```
/*
 * Here is a block comment.
 */
```

5.1.2 Single-Line Comments

```
if (condition) {

    /* Handle the condition. */
    ...
}
```

5. 댓글 – Java Coding Convention

5.1.3 Trailing Comments (후행 댓글)

```
if (a == 2) {
        return TRUE;                /* special case */
} else {
        return isPrime(a);          /* works only for odd a */
}
```

5.1.4 End-Of-Line Comments

```
if (foo > 1) {
        // Do a double-flip.
        ...
} else {
        return false;        // Explain why here.
}
//if (bar > 1) {
//
//
// Do a triple-flip.
// ...
//}
//else {
// return false;
//}
```

5. 댓글 – Java Coding Convention

5.2 문서 주석

Doc comments는

Java classes, interfaces, constructors, methods, and fields를 설명

```
/**
 * The Example class provides ...
*/
public class Example { ...
```

6. 선언 – Java Coding Convention

6.1 Number Per Line (번호당 라인)

```
/* 모범 예시 */
INT 수준; // 들여 쓰기 수준
INT 크기; // 테이블의 크기
INT 수준, 크기;

/* 잘못된 예시 : 같은 줄에 다른 유형 */
int foo, fooarray[]; //잘못된 예
```

6.2 초기화

선언되어 있는 지역변수를 초기화 하려는 것

6. 선언 – Java Coding Convention

6.3 배치

```
void myMethod() {
        int int1 = 0;                       // beginning of method block
        if (condition) {
        int int2 = 0;               // beginning of "if" block
        ...
        }
}
```

```
int count;
...
myMethod() {
        if (condition) {
                int count = 0;    // AVOID!
                ...
        }
        ...
}
```

6. 선언 – Java Coding Convention

6.4 클래스와 인터페이스 선언

자바 클래스와 인터페이스를 코딩 할 때,
다음과 같은 형식의 규칙을 따라야 한다 :

```
class Sample extends Object {
        int ivar1; int ivar2;
        Sample(int i, int j) {
                ivar1 = i; ivar2 = j;
        }
        int emptyMethod() {}
        ...
}
```

7. Statements – Java Coding Convention

7.1 단순 문

각 라인은 최대 하나의 문장을 포함해야 한다.

7.2 복합 문

복합 문은 중괄호 "{ statements } "로 묶인 문 목록을 포함하는 문이다

7.3 return 문

일반적으로 반환 값을 가진 Return문은 괄호와 함께 사용하지 않는다.

7. Statements – Java Coding Convention

7.4 if, if-else, if else-if else 문

```
if (condition) {
    statements;
}

if (condition) {
    statements;
} else {
    statements;
}

if (condition) {
    statements;
} else if (condition) {
    statements;
} else {
    statements;
}
```

7. Statements – Java Coding Convention

7.5 for문

for 문은 다음과 같은 형태를 따른다 :

```
for (initialization; condition; update) {
    statements;
}
```

7.6 while 문

while 문은 다음과 같은 형태를 따른다 :

```
while (condition) {
    statements;
}
```

An empty while문은 다음과 같은 형태를 따른다 :

```
while (condition);
```

7. Statements – Java Coding Convention

7.7 do-while문

```
do {
    statements;
} while (condition);
```

7.8 switch 문

```
switch (condition) {
case ABC:
    statements;
    /* falls through */
case DEF:
    statements;
    break;
default:
    statements;
    break;
}
```

7. Statements – Java Coding Convention

7.9 try-catch 문

```
try {
    statements;
} catch (ExceptionClass e) {
     statements;
}
```

try-catch 문 :
try문에 실제 실행되어야 하는 코드를 집어넣고 ,
try 문 안에서 어떤 에러가 발생 시 catch 문이 실행된다.
따라서 비정상적인 실행 시 catch문 안에 코드가 실행되므로
에러발생시 적절한 조치를 할 수 있는 코드를 집어넣는다.

```
try {
    statements;
} catch (ExceptionClass e) {
    statements;
} finally {
    statements;
}
```

Secure Coding

1. SQL 인젝션 - Secure Coding

- 개요

공격자가 입력 폼 및 URL 입력란에 SQL 문을 삽입하여
DB로부터 정보를 열람하거나 조작할 수 있는 보안취약점

- 보안대책

① Prepared Statement 객체 등을 이용하여
DB에 컴파일된 쿼리문(상수)을 전달하는 방법 사용

② Parameterized Statement를 사용하는 경우,
외부 입력데이터에 대하여 특수문자 및 쿼리 예약어 필터링

③ Struts, Spring 등과 같은 프레임워크를 사용하는 경우
외부 입력값 검증 모듈 사용

1. SQL 인젝션 - Secure Coding

- 코드예제

안전하지 않은 코드의 예 JAVA

```
1:  try
2:  {
3:    String tableName = props.getProperty("jdbc.tableName");
4:    String name = props.getProperty("jdbc.name");
5:    String query = "SELECT * FROM " + tableName + " WHERE Name =" + name;
6:    stmt = con.prepareStatement(query);
7:    rs = stmt.executeQuery();
8:    ... ...
9:  }
10:  catch (SQLException sqle) {     }
11:  finally {     }
```

안전한 코드의 예 JAVA

```
1:   try
2:   {
3:    String tableName = props.getProperty("jdbc.tableName");
4:    String name = props.getProperty("jdbc.name");
5:    String query = "SELECT   * FROM  ? WHERE Name = ? ";
6:    stmt = con.prepareStatement(query);
7:    stmt.setString(1, tableName);
8:    stmt.setString(2, name);
9:    rs = stmt.executeQuery();
10:    ... ...
11:  }
12:  catch (SQLException sqle) {    }
13:  finally {    }
```

2. 운영체제 명령 실행 - Secure Coding

- 개요

적절한 검증절차를 거치지 않은 사용자 입력값이 운영체제 명령어의 일부 또는 전부로 구성되어 실행되는 경우, 의도하지 않은 시스템 명령어가 실행되는 취약점

- 보안대책

① 웹 인터페이스를 통해 서버내부로 시스템 명령어를 전달시키지 않도록 웹 어플리케이션 구성

② 외부 입력에 따라 명령어를 생성하거나 선택이 필요한 경우, 명렁어 생성에 필요한 값들을 미리 지정해 놓고 외부 입력에 따라 선택하여 사용

2. 운영체제 명령 실행 - Secure Coding

- 코드예제

안전하지 않은 코드의 예 JAVA

```
1:  ……
2:    props.load(in);
3:    String version = props.getProperty("dir_type");
4:    String cmd = new String("cmd.exe /K    ₩"rmanDB.bat ₩"");
5:    Runtime.getRuntime().exec(cmd + " c:₩₩prog_cmd₩₩" + version);
6:  ……
```

안전한 코드의 예 JAVA

```
1:  ……
2:    props.load(in);
3:    String version[] = {"1.0", "1.1"};
4:    int versionSelection = Integer.parseInt(props.getProperty("version"));
5:    String cmd = new String("cmd.exe /K    ₩"rmanDB.bat ₩"");
6:    String vs = "";
7:    if (versionSelection == 0)
8:      vs = version[0];
9:    else if (versionSelection == 1)
10:     vs = version[1];
11:   else
12:     vs = version[1];
13:   Runtime.getRuntime().exec(cmd + "    c:₩₩prog_cmd₩₩" + vs);
14: ……
```

3. XQuery - Secure Coding

- 개요

XML 데이터에 대한 동적 쿼리문을 생성할 때 외부 입력값에 대해 적절한 검증 절차가 존재하지 않으면, 공격자가 쿼리문의 구조를 임의로 변경 가능하게 하는 취약점

- 보안대책

① Xquery에 사용되는 외부 입력데이터에 대하여 특수문자 및 쿼리 예약어 필터링

② Xquery를 사용한 쿼리문은 스트링을 연결하는 형태로 구성하지 않고 인자(파라메터)화된쿼리문 사용

3. XQuery - Secure Coding

- 코드예제

안전하지 않은 코드의 예 JAVA

```
1:  ......
2:  // 외부로 부터 입력을 받음
3:  String name = props.getProperty("name");
4:  Hashtable env = new Hashtable();
5:  env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
6:  env.put(Context.PROVIDER_URL, "ldap://localhost:389/o=rootDir");
7:  javax.naming.directory.DirContext ctx = new InitialDirContext(env);
8:  javax.xml.xquery.XQDataSource xqds = (javax.xml.xquery.XQDataSource)
    ctx.lookup("xqj/personnel");
9:  javax.xml.xquery.XQConnection conn = xqds.getConnection();
10: String es = "doc('users.xml')/userlist/user[uname='" + name + "']";
11: // 입력값이 Xquery의 인자로 사용
12: XQPreparedExpression expr = conn.prepareExpression(es);
13: XQResultSequence result = expr.executeQuery();
14: ......
```

안전한 코드의 예 JAVA

```
1:  ......
2:  // 외부로 부터 입력을 받음
3:  String name = props.getProperty("name");
4:  Hashtable env = new Hashtable();
5:  env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
6:  env.put(Context.PROVIDER_URL, "ldap://localhost:389/o=rootDir");
7:  javax.naming.directory.DirContext ctx = new InitialDirContext(env);
8:  javax.xml.xquery.XQDataSource xqds = (javax.xml.xquery.XQDataSource)
    ctx.lookup("xqj/personnel");
9:  javax.xml.xquery.XQConnection conn = xqds.getConnection();
10:
11: String es = "doc('users.xml')/userlist/user[uname='$xpathname']";
12: // 입력값이 Xquery의 인자로 사용
13: XQPreparedExpression expr = conn.prepareExpression(es);
14: expr.bindString(new QName("xpathname"), name, null);
15: XQResultSequence result = expr.executeQuery();
16: while (result.next())
17: {
18:   String str = result.getAtomicValue();
19:   if (str.indexOf('>') < 0)
20:   {
21:     System.out.println(str);
22:   }
```

4. XPath 인젝션 - Secure Coding

- 개요

외부 입력값을 적절한 검사과정 없이 XPath 쿼리문 전달이 가능해 지면,
공격자가 쿼리문의 구조를 임의로 변경 가능하게 하는 취약점

- 보안대책

① XPath 쿼리에 사용되는 외부 입력데이터에 대하여 특수문자 및
쿼리 예약어 필터링 수행
② 인자화된 쿼리문을 지원하는 Xquery문 사용

4. XPath 인젝션 - Secure Coding

- 코드예제

안전하지 않은 코드의 예 JAVA

```
1:  .....
2:  // 외부로 부터 입력을 받음
3:  String name = props.getProperty("name");
4:  String passwd = props.getProperty("password");
5:  .....
6:  XPathFactory factory = XPathFactory.newInstance();
7:  XPath xpath = factory.newXPath();
8:  .....
9:  // 외부 입력이 xpath의 인자로 사용
10: XPathExpression expr = xpath.compile("//users/user[login/text()='" + name + "' and
    password/text() = '" + passwd + "']/home_dir/text()");
11: Object result = expr.evaluate(doc, XPathConstants.NODESET);
12: NodeList nodes = (NodeList) result;
13: for (int i = 0; i < nodes.getLength(); i++)
14: {
15:   String value = nodes.item(i).getNodeValue();
16: .....
```

안전한 코드의 예 JAVA

dologin.xp 파일

```
1: declare variable $loginID as xs:string external;
2: declare variable $password as xs:string external;
3: //users/user[@loginID=$loginID and @password=$password]
```

XQuery를 이용한 XPath Injection 방지

```
1:  // 외부로 부터 입력을 받음
2:  String name = props.getProperty("name");
3:  String passwd = props.getProperty("password");
4:  Document doc = new Builder().build("users.xml");
5:  // XQuery를 위한 정보 로딩
6:  XQuery xquery = new XQueryFactory().createXQuery(new File("dologin.xq"));
7:  Map vars = new HashMap();
8:  vars.put("loginID", name);
9:  vars.put("password", passwd);
10: Nodes results = xquery.execute(doc, null, vars).toNodes();
11: for (int i=0; i < results.size(); i++)
12: {
13:     System.out.println(results.get(i).toXML());
14: }
```

5. 크로스 사이트 스크립트 - Secure Coding

- 개요

외부 입력이 동적 웹페이지 생성에 사용될 경우, 전송된 동적 웹페이지를 열람하는 접속자의 권한으로 부적절한 스크립트가 수행되는 취약점

- 보안대책

① 사용자가 입력한 문자열에서 <, >, &, ‘, ” 등을 replace등의 문자 변환함수 (혹은 메소드)를 사용하여 &lt, &amp, &quot로 치환

② 게시판 등에서 HTML, 태그 허용 시HTML 태그의 리스트를 선정한 후, 해당 태그만 허용하는 방식 적용

5. 크로스 사이트 스크립트 - Secure Coding

- 코드예제

안전하지 않은 코드의 예 JAVA

```
1:  <h1>XSS  Sample</h1>
2:  <%
3:    String name = request.getParameter("name");
4:  %>
5:  <p>NAME:<%=name%></p>
```

안전한 코드의 예 JAVA

```
1:  <%
2:    String name = request.getParameter("name");
3:  if ( name != null )
4:  {
5:    name = name.replaceAll("&", "&amp;");// &
6:    name = name.replaceAll("<", "&lt;"); // <
7:    name = name.replaceAll(">", "&gt;"); // >
8:    name = name.replaceAll("₩"", "&#34;"); // "
9:    name = name.replaceAll("₩'", "&#39;"); // '
10:   name = name.replaceAll("%00", null);   // null 문자
11:   name = name.replaceAll("%", "&#37;");  // %
12: }
13: else { return; }
14: %>
```

6. 파일 업로드 - Secure Coding

- 개요

서버사이드에서 실행될 수 있는 스크립트 파일(asp, j네, php 파일 등)이 업로드가 가능하고, 업로드된 파일이 웹을 통해 실행될 수 있는 취약점

- 보안대책

① 화이트리스트 방식으로 허용된 확장자만 업로드 허용

② 업로드 되는 파일을 저장할 때에는 파일명과 확장자를 외부사용자가 추측할 수 없는 문자열로 변경하여 저장

③ 저장 경로는 'web document root' 밖에 위치시켜, 웹을 통한 직접 접근 차단

6. 파일 업로드 - Secure Coding

- 코드예제

안전하지 않은 코드의 예 JAVA

```
1:  ……
2:  public void upload(HttpServletRequest request) throws ServletException
3:  {
4:    MultipartHttpServletRequest mRequest = (MultipartHttpServletRequest) request;
5:    String next = (String) mRequest.getFileNames().next();
6:    MultipartFile file = mRequest.getFile(next);
7:
8:    // MultipartFile로부터 file을 얻음
9:    String fileName = file.getOriginalFilename();
10:
11:   // upload 파일에 대한 확장자 체크를 하지 않음
12:   File uploadDir = new File("/app/webapp/data/upload/notice");
13:   String uploadFilePath = uploadDir.getAbsolutePath()+"/" + fileName;
14:   /* 이하 file upload 루틴 */
```

안전한 코드의 예 JAVA

```
1:  ……
2:  public void upload(HttpServletRequest request) throws ServletException
3:  {
4:    MultipartHttpServletRequest mRequest = (MultipartHttpServletRequest) request;
5:    String next = (String) mRequest.getFileNames().next();
6:    MultipartFile file = mRequest.getFile(next);
7:    if ( file == null ) return ;
8:    // 화이트리스트 방식으로 업로드 파일의 확장자를 체크한다.
9:    if ( fileName != null )
10:   {
11:     if ( fileName.endsWith(".doc") || fileName.endsWith(".hwp")  ||
       fileName.endsWith(".pdf") || fileName.endsWith(".xls") )
12:     {
13:       /* file 업로드 루틴 */
14:       // 저장 시 파일명을 외부 사용자가 추측할 수 없는 형태로 변경
15:       …
16:     }
17:   elsethrow new ServletExeption("에러");
18:   }
19: /* 이하 file upload 루틴 */
```

7. 파일 다운로드 - Secure Coding

- 개요

외부 입력값에 대해 경로 조작에 사용될 수 있는 문자를 필터링하지 않으면, 예상 밖의 접근 제한 영역에 대한 경로 문자열 구성이 가능해져 시스템 정보등 중요정보가 누출이 되는 취약점

- 보안대책

① 파일경로와 이름을 생성할 때 외부 입력값을 사용하는 경우, 정해진 경로 이외의 디렉토리와 파일에 접근할 수 없도록 처리

② 외부 입력값에 대해 replaceAll() 등의 메소드를 사용하여 예상 밖의 경로로의 접근을 허용하는 위험 문자열(", /, ₩,..)을 제거하는 필터링 적용

7. 파일 다운로드 - Secure Coding

- 코드예제

안전하지 않은 코드의 예 JAVA

```
1: ……
2: public void f(Properties request)
3: {
4:   ……
5:   String name = request.getProperty("filename");
6:   if( name != null )
7:   {
8:     File file = new File("/usr/local/tmp/" + name);
9:     file.delete();
10:  }
11: ……
12: }
```

안전한 코드의 예 JAVA

```
1: ……
2: public void f(Properties request)
3: {
4:   ……
5:   String name = request.getProperty("user");
6:  if ( name != null && !"".equals(name) )
7:   {
8:    name = name.replaceAll("&", "&amp;");// &
9:    name = name.replaceAll("<", "&lt;");// <
10:   name = name.replaceAll(">", "&gt;");// >
11:   name = name.replaceAll("₩"", "&#34;");// "
12:   name = name.replaceAll("₩'", "&#39;");// '
13:   name = name.replaceAll("%00", null);// null 문자
14:   name = name.replaceAll("%", "&#37;");// %
15:         name = name + "-report";
16:         File file = new File("/usr/local/tmp/" + name);
17.         if (file != null) file.delete();
18.  }
19. }
```

8. 버퍼 오버플로우 - Secure Coding

- 개요

정수형 변수의 오버플로우는 정수값이 증가하면서, Java에서 허용된 가장 큰 값보다 더 커져서 실제 저장되는 값은 의도하지 않게 아주 작은수이거나 음수가 될 수 있는 취약점

- 보안대책

① 언어/ 플랫폼 별 정수타입의 범위를 확인하여 사용 정수형 변수를 연산에 사용하는 경우 결과값이 범위 체크하는 모듈 사용

② 외부 입력 값을 동적으로 할당하여 사용하는 경우 변수의 값 범위를 검사하여 적절한 범위내에 존재하는 값인지 확인

8. 버퍼 오버플로우 - Secure Coding

- 코드예제

안전하지 않은 코드의 예 JAVA

```
1:  ……
2:  public static void main(String[] args)
3:  {
4:    int size = new Integer(args[0]).intValue();
5:    size +=  new Integer(args[1]).intValue();
6:    MyClass[] data = new MyClass[size];
7:  ……
```

안전한 코드의 예 JAVA

```
1:  ……
2:  public static void main(String[] args)
3:  {
4:    int size = new Integer(args[0]).intValue();
5:    size +=  new Integer(args[1]).intValue();
6:    // 배열의 크기 값이 음수값이 아닌지 검사한다.
7:    if (size < 0) return ;
8:    MyClass[ ] data = new MyClass[size];
9:  ……
```

9. LDAP 인젝션 - Secure Coding

- 개요

외부 입력값에 대해 특수문자를 필터링 하지 않으면 공격자에 의해
LDAP 명령어가 실행되는 취약점

- 보안대책

① DN과 필터에 사용되는 사용자 입력값에는 특수문자가 포함되지 않도록 특수문자 제거

② 특수문자를 사용해야 하는 경우 특수문자(DN에 사용되는 특수문자는 '₩', 필터에 사용되는 특수문자(= +, <, >, #, ;, ₩ 등)에 대해서는 실행 명령이 아닌 일반문자로 인식되도록 처리

9. LDAP 인젝션 - Secure Coding

- 코드예제

안전하지 않은 코드의 예 JAVA

```
1:  Properties props = new Properties();
2:  String fileName = "ldap.properties";
3:  FileInputStream in = new FileInputStream(fileName);
4:  props.load(in);
5:  String name = props.getProperty("name");
6:  String filter = "(name =" + name + ")";
7:  NamingEnumeration answer=ctx.search("ou=NewHires",filter,new
    SearchControls());
8:  printSearchEnumeration(answer);
9:  ctx.close();
```

안전한 코드의 예 JAVA

```
1:  Properties props = new Properties();
2:  String fileName = "ldap.properties";
3:  FileInputStream in = new FileInputStream(fileName);
4:  if (in == null || in.available() <= 0) return;
5:  props.load(in);
6:  if (props == null || props.isEmpty()) return;
7:  String name = props.getProperty("name");
8:  if (name == null || "".equals(name)) return;
9:  String filter = "(name =" + name.replaceAll("₩₩*", "") + ")";
10: NamingEnumeration answer = ctx.search("ou=NewHires", filter, new SearchControls());
11: printSearchEnumeration(answer);
12: ctx.close();
```

10. HTTP 응답분할 - Secure Coding

- 개요

HTTP 요청에 들어 있는 인자값이 HTTP 응답헤더에 포함되어 사용자에게 다시 전달 될 때 입력값에 CR(arriage Return)이나 LF(Line Feed)와 같은 개행문자가 존재하면 HTTP 응답이 2개 이상으로 분리되어, 공격자는 개행문자를 이용하여 첫 번째 응답을 종료시키고, 두 번째 응답에 악의적인 코드를 주입하여 XSS 및 캐시 훼손(cache poisoning) 공격 등이 가능한 취약점

- 보안대책

① 외부에서 입력된 인자값을 HTTP 응답헤더(Set Cookie 등)에 포함시킬 경우 CR, LF 등의 개행문자를 제거

10. HTTP 응답분할 - Secure Coding

- 코드예제

안전하지 않은 코드의 예 JAVA

```
1:  throws IOException, ServletException
2:  {
3:    response.setContentType("text/html");
4:    String author = request.getParameter("authorName");
5:    Cookie cookie = new Cookie("replidedAuthor", author);
6:    cookie.setMaxAge(1000);
7:    response.addCookie(cookie);
8:    RequestDispatcher frd = request.getRequestDispatcher("cookieTest.jsp");
9:         frd.forward(request, response);
10: }
```

안전한 코드의 예 JAVA

```
1:  throws IOException, ServletException
2:  {
3:    response.setContentType("text/html");
4:    String author = request.getParameter("authorName");
5:    if (author == null || "".equals(author)) return;
6:    String filtered_author = author.replaceAll("₩r", "").replaceAll("₩n", "");
7:    Cookie cookie = new Cookie("replidedAuthor", filtered_author);
8:    cookie.setMaxAge(1000);
9:    cookie.setSecure(true);
10:   response.addCookie(cookie);
11:   RequestDispatcher frd = request.getRequestDispatcher("cookieTest.jsp");
12:   frd.forward(request, response);
13: }
```

11. URL/파라미터 변조 - Secure Coding

- 개요

실행경로에 대해서 접근제어를 검사하지 않거나 불완전하게 구현하여 공격자로 하여금 값을 변조하여 중요정보에 접근 가능해지는 취약점

- 보안대책

① 중요한 정보가 있는 페이지(개좌이체 등)는 인증 적용

② 안전하다고 확인된 라이브러리나 프레임워크를 사용

③ 응용프로그램이 제공하는 정보와 기능을 역할에 따라 배분함으로써 공격자에게 노출되는 공격노출면 최소화

④ 사용자의 권한에 따른 ACL(Access Control List) 관리

11. URL/파라미터 변조 - Secure Coding

- 코드예제

안전하지 않은 코드의 예 JAVA

```
1: public void sendBankAccount(String accountNumber,double balance)
2: {
3:     BankAccount account = new BankAccount();
4:     account.setAccountNumber(accountNumber);
5:     account.setToPerson(toPerson);
6:     account.setBalance(balance);
7:     AccountManager.send(account);
8: }
```

안전한 코드의 예 JAVA

```
1: public void sendBankAccount(HttpServletRequest request, HttpSession session, String
   accountNumber,double balance)
2: {
3:     String newUserName = request.getParameter("username");
4:     String newPassword = request.getParameter("password");
5:     if ( newUserName == null || newPassword == null )
6:     {
7:         throw new MyEception("데이터 오류");
8:     }
9:     String password = session.getValue("password");
10:    String userName = session.getValue("username");
11:    if (isAuthenticatedUser() && newUserName.equal(userName) &&
       newPassword.equal(password) )
12:    ...
13: }
```

12. 취약한 계정 생성 허용 - Secure Coding

- 개요

회원가입 시 안전한 패스워드 규칙이 적용되지 않아, 무차별 대입법 공격 등에 의해 공격자에게 패스워드가 노출될 수 있는 취약점

- 보안대책

① 사용자가 취약한 패스워드를 사용할 수 없도록 패스워드 생성규칙을 강제 할 수 있는 로직 적용

12. 취약한 계정 생성 허용 - Secure Coding

- 코드예제

안전하지 않은 코드의 예 JAVA

```
1:  public boolean pwchk(String pw)
2:  {
3:    int pwlen = strlen(pw);
4:    if( pwlen > 6 ) return true;
5:    else return false;
6:  }
```

안전한 코드의 예 JAVA

```
1:  public boolean pwchk(String pw)
2:  {
3:    BufferedReader reader = new BufferedReader(new FileReader("weakpwList"));
4:    String weakpw = null;
5:    while ((weakpw = reader.readLine()) != null) {
6:      if( weakpw.equalsIgnoreCase(pw) == true ){
7:        return false;
8:      }
9:    }
10:   reader.close();
11:   int pwlen = strlen(pw);
12:   if( pwlen > 8 ) return true;
13:   else return false;
14: }
```

13. 불충분한 세션관리 - Secure Coding

- 개요

인증 시 마다 동일한 세션 ID가 발급되거나 세션 타임아웃을 너무 길게 설정하여 공격자가 세션을 재상용 할 수 있는 취약점

- 보안대책

① 세션 ID는 로그인 시 마다 추측할 수 없는 새로운 세션 ID로 발급

② 세션 타임아웃 설정을 통해 일정시간(최대 30분 이상) 동안 움직임이 없을 경우 자동 로그아웃 되도록 구현

13. 불충분한 세션관리 - Secure Coding

- 코드예제

안전하지 않은 코드의 예 JAVA

```
1:  public void setKeepTime(){
2:    // Session의 유지 시간을 Setting
3:    String strTime = Param.getPropertyFromXML("SessionPersistenceTime");
4:    if(strTime == null) {
5:      session.setMaxInactiveInterval(60*10000);
6:    } else {
7:      session.setMaxInactiveInterval(new Integer(strTime)).intValue());
8:    }
9:  }
10: ... 중략
```

안전한 코드의 예 JAVA

```
1:  public void setKeepTime(){
2:    // Session의 유지 시간을 Setting
3:    String strTime = Param.getPropertyFromXML("SessionPersistenceTime");
4:    if(strTime == null) {
5:      session.setMaxInactiveInterval(60*20);
6:    } else {
7:      session.setMaxInactiveInterval(new Integer(strTime)).intValue());
8:    }
9:  }
10: public boolean isUsing(String userID)
11: {
12:   boolean isUsing = false;
13:   Enumeration e = loginUsers.keys();
14:   String key = "";
15:   while(e.hasMoreElements()) {
16:     key = (String)e.nextElement();
17:     if(userID.equals(loginUsers.get(key))) {
18:       isUsing = true;
19:     }
20:   }
21:   return isUsing;
22: ... 중략
```

14. 데이터 평문전송 - Secure Coding

- 개요

중요 정보와 관련된 민감한 데이터(개인정보, 비밀번호 등)를 평문으로 송수신할 경우, 통신 채널 스니핑을 통해 인가되지 않은 사용자에게 민감한 데이터가 노출될 수 있는 취약점

- 보안대책

① 중요정보와 관련된 민감한 데이터(개인정보 비밀번호 등) 전송 시 통신채널(또는 전송데이터) 암호화 적용

② Parameterized Statement를 사용하는 경우,
외부 입력데이터에 대하여 특수문자 및 쿼리 예약어 필터링

③ Struts, Spring 등과 같은 프레임워크를 사용하는 경우
외부 입력값 검증 모듈 사용

14. 데이터 평문전송 - Secure Coding

- 코드예제

안전하지 않은 코드의 예 JAVA

```
1:  void foo()
2:  {
3:    try
4:    {
5:    Socket socket = new Socket("taranis", 4444);
6:    PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
7:    String password = getPassword();
8:    out.write(password);
9:    }
10:   catch (FileNotFoundException e)
11:   {
12:     ...
```

안전한 코드의 예 JAVA

```
1:  void foo()
2:  {
3:    try
4:    {
5:      Socket socket = new Socket("taranis", 4444);
6:      PrintStream out = new PrintStream(socket.getOutputStream(), true);
7:      Cipher c = Cipher.getInstance("AES/CBC/PKCS5Padding");
8:      String password = getPassword();
9:      encryptedStr= c.update(password.getBytes());
10:     out.write(encryptedStr,0,encryptedStr.length);
11:   }
12:   catch (FileNotFoundException e) {
```

15. 쿠키 변조 - Secure Coding

- 개요

웹 서비스에서 사용자 인증 등 중요기능 구현 시 쿠키를 활용할 경우
공격자의 패킷 스니핑을 통해 해당 쿠키가 탈취되어 타 사용자로 로그인이
가능해지는 취약점

- 보안대책

① 사용자 인증 등 중요기능 구현 시 가급적이면 Cookie 대신
Session 방식 사용

② 사용자 인증 등 중요기능 구현 시 Cookie(또는 Session) 방식 황용 시
안전한 알고리즘(SEED, 3DES, AES 등)을 사용

15. 쿠키 변조 - Secure Coding

- 코드예제

안전하지 않은 코드의 예 JAVA

```
1:  public void createCookie(){
2:    Cookie cookie = new Cookie("cookie_key", "1234");
3:    cookie.setMaxAge(60*60*24);
4:    cookie.setPath("/");
5:    response.addCookie(cookie);
6:  }
```

안전한 코드의 예 JAVA

```
1:  public void createCookie(){
2:    String value = "1234";
3:    value = getEncrypt(value);
4:    Cookie cookie = new Cookie("cookie_key", value);
5:    cookie.setMaxAge(60*60*24);
6:    cookie.setPath("/");
7:    response.addCookie(cookie);
8:  }
```

16. 취약한 암호화 알고리즘 사용 - Secure Coding

- 개요

중요정보를 암호화 하기 위해 사용되는 방법이 인코딩(또는 암호화)를 하는데 취약한 인코딩 방법 및 암호화 알고리즘을 사용할 경우, 공격자가 해독이 가능하여 중요정보가 노출될 수 있는 취약점

- 보안대책

① 개인정보 등 중요정보를 보호하기 위해 사용하는 암호알고리즘 적용 시, IT보안인증사무국이 안정성을 확인한 검증필 암호모듈 사용

16. 취약한 암호화 알고리즘 사용 - Secure Coding

- 코드예제

안전하지 않은 코드의 예 JAVA

```
1:  try
2:  {
3:    Cipher c = Cipher.getInstance("DES");
4:    c.init(Cipher.ENCRYPT_MODE, k);
5:    rslt = c.update(msg);
6:  }
7:  catch (InvalidKeyException e) {
```

안전한 코드의 예 JAVA

```
1:  try
2:  {
3:    Cipher c = Cipher.getInstance("SEED/CBC/PKCS5Padding");
4:    c.init(Cipher.ENCRYPT_MODE, k);
5:    rslt = c.update(msg);
6:  }
7:  catch (InvalidKeyException e) {
```

17. 취약한 패스워드 복구 - Secure Coding

- 개요

패스워드 복구 메커니즘(아이디/비밀번호 찾기 등)이 취약한 경우 불법적으로 다른 사용자의 패스워드를 획득, 변경, 복구 되는 취약점

- 보안대책

① 사용자를 시별하기 위한 수단 활용 시 그 사용자의 유일한 값 사용

② 사용자 본인인증 매커니즘 구현 시 추측이 불가하게 구현

③ 임시 패스워드 발급 시 안전한 난수 값 사용

17. 취약한 패스워드 복구 - Secure Coding

- 코드예제

안전하지 않은 코드의 예 JAVA

```
1:  try
2:  {
3:  String tableName = props.getProperty("jdbc.tableName");
4:  String userid = props.getProperty("jdbc.id");
5:  String query = "SELECT   * FROM  ? WHERE Name = ? ";
6:  stmt = con.prepareStatement(query);
7:  stmt.setString(1, tableName);
8:  stmt.setString(2, userid);
9:  rs = stmt.executeQuery();
10: ... ... 11:   }
12: catch (SQLException sqle) {    }
13: finally {    }
```

안전한 코드의 예 JAVA

```
1:  try
2:  {
3:  String tableName = props.getProperty("jdbc.tableName");
4:  String userid = props.getProperty("jdbc.id");
5:  String useremail = props.getProperty("jdbc.email");
6:  String query = "SELECT   * FROM  ? WHERE Name = ? AND Email = ? ";
7:  stmt = con.prepareStatement(query);
8:  stmt.setString(1, tableName);
9:  stmt.setString(2, userid);
10: stmt.setString(3, useremail);
11: rs = stmt.executeQuery();
12: ... ...
13: }
14: catch (SQLException sqle) {    }
15: finally {    }
```

18. 주석을 통한 정보 노출 - Secure Coding

- 개요

소스코드 주석문에 민감한 정보(개인 정보, 시스템 정보 등)이 포함되어 있는 경우, 외부 공격자에 의해 패스워드 등 보안 관련정보가 노출될 수 있는 취약점

- 보안대책

① 디버깅 목적으로 주석 ID, 패스워드, 시스템 관련정보 등 보안관련 정보는 개발완료 후 제거 필요

18. 주석을 통한 정보 노출 - Secure Coding

- 코드예제

안전하지 않은 코드의 예 JAVA

```
1:  .....
2:  // ID : tiger, Password : "tiger."
3:  public boolean DBConnect()
4:  {
5:  String url = "DBServer";
6:  String password = "tiger";
7:  Connection con = null;
8:
9:  try
10: {
11:     con = DriverManager.getConnection(url, "scott", password);
12:   }
13:   catch (SQLException e)
14:   }
15:       .....
```

안전한 코드의 예 JAVA

```
1:  .....
2:  //디버깅 등의 용도로 소스 주석에 적어놓은 패스워드 삭제 조치 필요
3:  public Connection DBConnect(String id, String password)
4:  {
5:  String url = "DBServer";
6:  Connection conn = null;
7:  try
8:     {
9:   String CONNECT_STRING = url + ":" + id + ":" + password;
10:   InitialContext ctx = new InitialContext();
11:  DataSource datasource = (DataSource) ctx.lookup(CONNECT_STRING);
12:  conn = datasource.getConnection();
13:    }
14: catch (SQLException e)  { .....
15:       return conn;
16: }
```

THANK YOU!