1 주요 취약점 패턴

1 매개변수 조작

- URL에서 변수 또는 POST 형태로 전송되는 매개변숫값을 조작하여 자신이 소유한 것 외의 권한을 획득하는 공격 기법(권한 획득)

- 매개변수 조작 취약점이 존재하는지 확인하려면 사용자 권한이 필요한 페이지에 접근할 때 세션 처리를 하는 코드를 포함하는지 확인
  - 예) 사용자가 입력한 매개변숫값을 받아들여 쿠키로 저장하고 그 쿠키 값으로 인증하는 경우

2 쿠키 조작

- 브라우저의 쿠키 값을 통해 사용자 권한을 부여하는 코드
(권한 부여)

```
Cookie[] = cookies = request.getCookies();
for(int i = 0; i < cookies.length; i + +)
{
  Cookie c = cookies[i];
  if(c.getName().equals("role")) {
    userRole = c.getValue();
  }
}
```

2 쿠키 조작

- 쿠키 값은 쉽게 조작할 수 있기 때문에 쿠키 값으로 인증하는 코드는 안전하지 않음
  - ➡ 매개변수와 쿠키 조작 취약점 관련 소스코드를 살펴보려면 getCookie와 같이 쿠키를 입력 받아 처리하는 부분이 있는지 확인(getCookie)

3 강제 브라우징

- 권한이 필요한 페이지에 권한 체크 코드가 누락되었을 때 발생(권한 체크 누락)

- /admin/과 같은 관리자 디렉터리 내에 있는 소스코드 중에 세션 처리 등의 코드가 존재하는지 인해봄으로써 강제 브라우징 취약점이 존재하는 페이지를 찾을 수 있음

  ➡ 관리자 페이지 및 인증이 필요한 페이지에 세션 처리 코드가 포함되어 있는지 확인

4 중요 정보 평문 전송

- 패스워드나 주민등록번호와 같은 민감한 내용을 암호화하지 않고 평문으로 전송할 때 발생하는 취약점 (평문 전송 → Telnet, FTP)

5 중요 정보 평문 전송 취약점을 확인하는 방법

- 사용자의 중요 정보를 전송하는 페이지가 HTTPS(SSL) 등의 암호화된 채널로 통신하는지 확인(http vs. https)

- 소스코드 내에서 중요한 정보를 암호화 함수를 통해 전송하는지 확인

6 행정안전부 소프트웨어 개발 보안 가이드 예제

```
void foo( )
{
  try
    {
      Socket socket = new Socket("transit", 4444);
      PrintWriter out = new
PrintWriter(socket.getOutputStream( ), true);
      String password = getPassword( ); // 암호X
      out.write(password)
    }
    catch (FileNotFoundException e)
    {
       ⋮
```

6 행정안전부 소프트웨어 개발 보안 가이드 예제

- 네트워크상에서 Socket 함수로 데이터를 전송할 때 암호화를 하지 않는다면 공격자가 스니핑을 통해 패스워드를 얻을 수 있음(스니핑)

6 행정안전부 소프트웨어 개발 보안 가이드 예제

안전한 코드의 예 JAVA

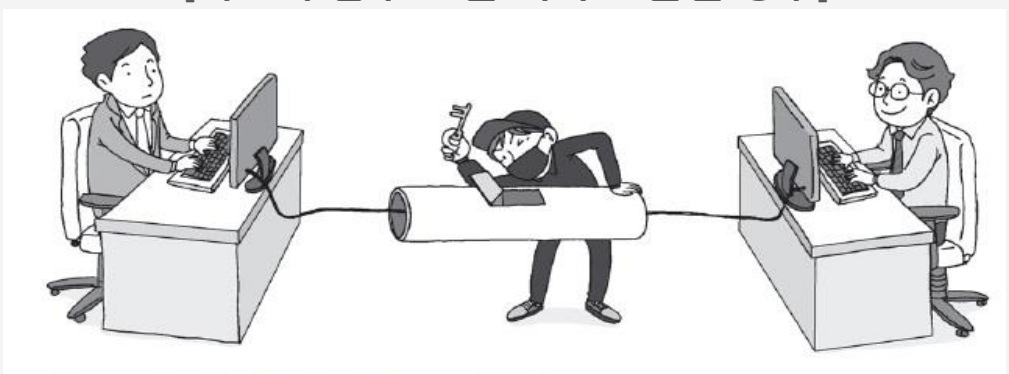
```
1:  // 패스워드를 암호화 하여 전송
2:  try {
3:     Socket s = new Socket("taranis", 4444);
4:     PrintStream o = new PrintStream(s.getOutputStream(), true);
5:     Cipher c = Cipher.getInstance("AES/CBC/PKCS5Padding");
6:     String password = getPassword();
7:     byte[] encPassword = c.update(password.getBytes());
8:     o.write(encPassword, 0, encPassword.length);
9:  } catch (FileNotFoundException e) {
10:        .....
11:  }
```

※ 출처 : http://www.mois.go.kr/frt/bbs/type001/commonSelectBoardArticle.do?bbsId=BBSMSTR_000000000015&nttId=57473)

7 암호화 함수를 이용한 전송

- 소스코드 또는 중요 정보를 암호화하더라도
  복호화 함수나 복호화 키가 쉽게 노출된다면
  암호화된 정보가 공격자에 의해 악용될 수 있음

[복호화 함수 또는 키가 노출된 경우]



※ 출처 : 인터넷 해킹과 보안, 김경곤, 한빛아카데미, 2017

8 암호화 함수를 이용한 전송

분류		보호함수 목록
최소 안전성 수준		112비트
블록암호		ARIA(키 길이 : 128/192/256), SEED(키 길이 : 128)
블록암호 운영모드	기밀성	ECD, CBC, CFB, OFB, CTR
	기밀성/인증	CCM, GCM
해시함수		SHA-224/256/384/512
메시지 인증코드	해시함수기반	HMAC
	블록기반	CMAC, GMAC
난수발생기	해시함수 /HMAC 기반	HASH_DRBG, HMAC_DRBG
	블록기반	CTR_DRBG
공개키 암호		RSAES - (공개키 길이) 2048, 3072 - RSA-OAEP에서 사용되는 해시함수 : SHA-224/256
전자서명		RSA-PSS, KCDSA, ECDSA, EC-KCDSA
키 설정 방식		DH, ECDH

※ 출처 : http://www.mois.go.kr/frt/bbs/type001/commonSelectBoardArticle.do?bbsId=BBSMSTR_000000000015&nttId=57473

9 하드 코딩된 패스워드

- 하드 코딩(Hard coding)
  : 패스워드 및 시스템 접속 정보와 같은 민감한 정보가 소스코드 내에 그대로 노출되어 있는 경우

- 공격자가 소스코드를 외부에서 열람할 경우 심각한 2차 공격에 악용될 수 있음(편의를 위해)

9 하드 코딩된 패스워드

- 데이터베이스 접속 정보 중 패스워드(Anesra)를 하드 코딩한 예

```
public Connection DBConnect(String url, String id)
{
  try
  {
    conn = DriverManager.getConnection(url, id,
"anesra"); // 패스워드 하드코딩
  }
  catch (SQLException e)
  {
    System.err.println("...");
  }
  return conn;
}
```

9 하드 코딩된 패스워드

안전한 코드의 예 JAVA

```
1:  public class MemberDAO {
2:      private static final String DRIVER = "oracle.jdbc.driver.OracleDriver";
        private static final String URL  = "jdbc:oracle:thin:@192.168.0.3:1521:ORCL";
3:      private static final String USER = "SCOTT"; // DB ID
4:      Member_List mList;
5:                              ......
6:      public Connection getConn() {
7:          Connection con = null;
8:          try {
9:              Class.forName(DRIVER);
10:             String PASS = props.getProperty("EncryptedPswd");
11:             byte[] decryptedPswd = cipher.doFinal(PASS.getBytes());
12:             PASS = new String(decryptedPswd);
13:              con = DriverManager.getConnection(URL, USER, PASS);
14:         } catch (Exception e) {
15:             log.error(e.getMessage());
16:         }
```

※ 출처 : http://www.mois.go.kr/frt/bbs/type001/commonSelectBoardArticle.do?bbsId=BBSMSTR_000000000015&nttId=57473)

10 주석 처리된 중요 정보

- 주석에는 해당 소스코드가 어떤 기능을 하는지 설명하는 내용이 많은데, 간혹 개발 단계에서 수정이 발생한 소스코드를 삭제하지 않고 주석 처리를 하는 경우가 있음
  - ➡ 언어별 주석 처리를 하는 코드만 발췌하여 해당 코드 내에 패스워드 또는 데이터베이스 접속 정보 등 민감한 내용이 포함되어 있는지 확인 (하드코딩 정보)

10 주석 처리된 중요 정보

안전하지 않은 코드의 예 JAVA

```
1:  public void daoTest() throwsException {
2:    // dbsample : 84d5d0a08a3ec5e2d91a
3:    // 암호화 전, 후 : 1365ADMIN_01, aa84c40031d808196537ad3dcf81f9af
4:    String pwd = "aa84c40031d808196537ad3dcf81f9af";
5:    String pwd1 = ARIAEngine.decARIA(pwd);
6:    System.out.println(pwd1);
7:  }
```

※ 출처 : http://www.mois.go.kr/frt/bbs/type001/commonSelectBoardArticle.do?bbsId=BBSMSTR_000000000015&nttId=57473)

10 보안 기능

※ 출처 : http://www.mois.go.kr/frt/bbs/type001/commonSelectBoardArticle.do?bbsId=BBSMSTR_000000000015&nttId=57473

11 주요 취약점 패턴

1. 적절한 인증 없는 중요기능 허용

가. 개요

- 적절한 인증과정이 없이 중요정보(계좌이체 정보, 개인정보 등)를 열람(또는 변경)할 때 발생하는 보안 약점

11 주요 취약점 패턴

1. 적절한 인증 없는 중요기능 허용

- 나. 보안대책
  - 클라이언트의 보안검사를 우회하여 서버에 접근하지 못하도록 설계하고 중요한 정보가 있는 페이지는 재인증을 적용(은행 계좌이체 등)함, 또한 안전하다고 검증된 라이브러리나 프레임워크(OpenSSL이나 ESAPI의 보안기능 등)를 사용하는 것이 중요(하트블리드)

11 주요 취약점 패턴

2. 부적절한 인가

가. 개요

- 프로그램이 모든 가능한 실행경로에 대해서 접근제어를 검사하지 않거나 불완전하게 검사하는 경우, 공격자는 접근 가능한 실행경로를 통해 정보를 유출할 수 있음

11 주요 취약점 패턴

2. 부적절한 인가

- 나. 보안대책
  - 응용프로그램이 제공하는 정보와 기능을 역할에 따라 배분함으로써 공격자에게 노출되는 공격노출면(Attack Surface)을 최소화하고 사용자의 권한에 따른 ACL(Access Control List)을 관리함, JAAS Authorization Framework나 OWASP ESAPI Access Control 등의 프레임워크를 사용해서 취약점을 제거할 수도 있음(접근 제어)

11 주요 취약점 패턴

3. 중요한 자원에 대한 잘못된 권한 설정

▶ 가. 개요

- SW가 중요한 보안관련 자원에 대하여 읽기 또는 수정하기 권한을 의도하지 않게 허가할 경우, 권한을 갖지 않은 사용자가 해당자원을 사용하게 됨

11 주요 취약점 패턴

3. 중요한 자원에 대한 잘못된 권한 설정

- 나. 보안대책
  - 설정파일, 실행파일, 라이브러리 등은 SW 관리자에 의해서만 읽고 쓰기가 가능하도록 설정하고 설정 파일과 같이 중요한 자원을 사용하는 경우, 허가 받지 않은 사용자가 중요한 자원에 접근 가능한지 검사함(접근 제어)

11 주요 취약점 패턴

4. 취약한 암호화 알고리즘 사용

가. 개요

- SW 개발자들은 환경설정 파일에 저장된 패스워드를 보호하기 위하여 간단한 인코딩 함수를 이용하여 패스워드를 감추는 방법을 사용하기도 함, 그렇지만 base64와 같은 지나치게 간단한 인코딩 함수로는 패스워드를 제대로 보호할 수 없음(복호화 가능)

11 주요 취약점 패턴

4. 취약한 암호화 알고리즘 사용

▶ 가. 개요

- 정보보호 측면에서 취약하거나 위험한 암호화 알고리즘을 사용해서는 안 됨, 표준화되지 않은 암호화 알고리즘을 사용하는 것은 공격자가 알고리즘을 분석하여 무력화시킬 수 있는 가능성을 높일 수도 있음, 몇몇 오래된 암호화 알고리즘의 경우는 컴퓨터의 성능이 향상됨에 따라 취약해지기도 해서, 예전에는 해독하는데 오랜 시간이 걸릴 것이라고 예상되던 알고리즘이 며칠이나 몇 시간 내에 해독되기도 함, RC2, RC4, RC5, RC6, MD4, MD5, SHA1, DES 알고리즘이 여기에 해당

11 주요 취약점 패턴

4. 취약한 암호화 알고리즘 사용

- 나. 보안대책
  - 자신만의 암호화 알고리즘을 개발하는 것은 위험하며 학계 및 업계에서 이미 검증된 표준화된 알고리즘을 사용함, 기존에 취약하다고 알려진 DES, RC5등의 암호알고리즘을 대신하여 3DES, AES, SEED 등의 안전한 암호알고리즘으로 대체하여 사용함, 또한 업무관련 내용, 개인정보 등에 대한 암호 알고리즘 적용시, IT보안인증 사무국이 안전성을 확인한 검증필 암호모듈을 사용해야 함(국내 암호)

11 주요 취약점 패턴

8. 충분하지 않은 키 길이 사용

▶ 가. 개요

- 길이가 짧은 키를 사용하는 것은 암호화 알고리즘을 취약하게 만들 수 있음, 키는 암호화 및 복호화에 사용되는데, 검증된 암호화 알고리즘을 사용하더라도 키 길이가 충분히 길지 않으면 짧은 시간 안에 키를 찾아낼 수 있고 이를 이용해 공격자가 암호화된 데이터나 패스워드를 복호화 할 수 있게 됨 (brute-force attack)

11 주요 취약점 패턴

8. 충분하지 않은 키 길이 사용

- 나. 보안대책
  - RSA 알고리즘은 적어도 2,048 비트 이상의 길이를 가진 키와 함께 사용해야 하고, 대칭암호화 알고리즘(Symmetric Encryption Algorithm)의 경우에는 적어도 128비트 이상의 키를 사용

11 주요 취약점 패턴

9. 적절하지 않은 난수값 사용

- 가. 개요
  - 예측 가능한 난수를 사용하는 것은 시스템에 보안약점을 유발함, 예측 불가능한 숫자가 필요한 상황에서 예측 가능한 난수를 사용한다면, 공격자는 SW에서 생성되는 다음 숫자를 예상하여 시스템을 공격하는 것이 가능(암호키)

11 주요 취약점 패턴

9. 적절하지 않은 난수값 사용

- 나. 보안대책
  - 난수발생기에서 시드(Seed)를 사용하는 경우에는 예측하기 어려운 방법으로 변경함, 일반적으로 Java에서는 java.lang.Math.random() 메서드 사용을 자제하고, java.util.Random 클래스를 사용, C에서는 rand() 대신 randomize(seed)를 사용하는 것이 좋음, 세션 ID, 암호화키 등 보안결정을 위한 값을 생성하고 보안결정을 수행하는 경우에는 javasecuritySecureRandom 클래스를 사용

11 주요 취약점 패턴

10. 하드코드된 암호화 키

▶ 가. 개요

- 코드 내부에 하드코드된 암호화 키를 사용하여 암호화를 수행하면 암호화된 정보가 유출될 가능성이 높아짐, 많은 SW 개발자들이 코드 내부의 고정된 암호키의 해시를 계산하여 저장하면 암호키를 악의적인 공격자로부터 보호할 수 있다고 믿고 있음, 그러나 일부 해시함수들이 역계산이 가능하며 무차별 대입(Brute-Force) 공격에는 취약하다는 것을 고려해야만 함(일방향성)

11 주요 취약점 패턴

10. 하드코드된 암호화 키

- 나. 보안대책
  - 암호화 수행시 상수가 아닌 암호화 키를 사용하도록 설계해야 함, 또한 암호화되었더라도 소스코드 내부에 상수형태의 암호키를 저장하여 사용해서는 안됨

11 주요 취약점 패턴

11. 취약한 비밀번호 허용

가. 개요

- 사용자에게 강한 패스워드 조합규칙을 요구하지 않으면, 사용자 계정이 취약하게 됨, 안전한 패스워드를 생성하기 위해서는 한국인터넷진흥원 「암호이용안내서」의 패스워드 설정규칙을 적용해야 함

11 주요 취약점 패턴

11. 취약한 비밀번호 허용

- 나. 보안대책
  - 패스워드 생성시 강한 조건 검증을 수행,
    비밀번호(패스워드)는 숫자와 영문자, 특수문자 등을
    혼합하여 사용하고, 주기적으로 변경하여 사용하도록
    해야 함

11 주요 취약점 패턴

12. 사용자 하드디스크에 저장되는 쿠키를 통한 정보노출

- 가. 개요
  - 대부분의 웹 응용프로그램에서 쿠키는 메모리에 상주하며 브라우저의 실행이 종료되면 사라짐, 프로그래머가 원하는 경우 쿠키를 디스크에 저장할 수 있으며, 다음 브라우저 세션이 시작되었을 때 메모리에 로드됨, 개인정보, 인증정보 등이 이와 같은 영속적인 쿠키(Persistent Cookie)에 저장된다면, 공격자는 쿠키에 접근할 수 있는 보다 많은 기회를 가지게 되어 시스템을 취약하게 만듦(세션 쿠키 vs. 영구 쿠키)

11 주요 취약점 패턴

12. 사용자 하드디스크에 저장되는 쿠키를 통한 정보노출

- 나. 보안대책
  - 쿠키의 만료시간은 세션이 지속되는 시간을 고려하여 최소한으로 설정하고 영속적인 쿠키에는 사용자 권한 등급, 세션ID 등 중요정보가 포함되지 않도록 함

11 주요 취약점 패턴

14. 솔트 없이 일방향 해시함수 사용

- 가. 개요
  - 패스워드를 저장시 일방향 해시함수의 성질을 이용하여 패스워드의 해시값을 저장함, 만약 패스워드를 솔트(Salt)없이 해시하여 저장한다면, 공격자는 레인보우 테이블과 같이 해시값을 미리 계산하여 패스워드를 찾을 수 있게 됨
- 나. 보안대책
  - 패스워드를 저장시 패스워드와 솔트를 해시함수에 함께 입력하여 얻은 해시값을 저장함

11 주요 취약점 패턴

15. 무결성 검사 없는 코드 다운로드

▶ 가. 개요

- 원격으로부터 소스코드 또는 실행파일을 무결성 검사 없이 다운로드 받고, 이를 실행하는 제품들이 종종 존재함, 이는 호스트 서버의 변조, DNS 스푸핑(Spoofing) 또는 전송시의 코드 변조 등의 방법을 이용하여 공격자가 악의적인 코드를 실행할 수 있도록 함(무결성)

11 주요 취약점 패턴

15. 무결성 검사 없는 코드 다운로드

- 나. 보안대책
  - DNS 스푸핑(Spoofing)을 방어할 수 있는 DNS lookup을 수행하고 코드 전송시 신뢰할 수 있는 암호 기법을 이용하여 코드를 암호화함, 또한 다운로드한 코드는 작업 수행을 위해 필요한 최소한의 권한으로 실행하도록 함(DNS Spoofing)

11 주요 취약점 패턴

16. 반복된 인증시도 제한 기능 부재

- 가. 개요
  - 일정 시간 내에 여러 번의 인증을 시도하여도 계정 잠금 또는 추가 인증 방법 등의 충분한 조치가 수행되지 않는 경우, 공격자는 예상 ID와 비밀번호들을 사전(Dictionary)으로 만들고 무차별 대입(brute-force) 하여 로그인 성공 및 권한 획득이 가능

11 주요 취약점 패턴

16. 반복된 인증시도 제한 기능 부재

- 나. 보안대책
  - 인증시도 횟수를 적절한 횟수로 제한하고 설정된 인증실패 횟수를 초과했을 경우 계정을 잠금하거나 추가적인 인증 과정을 거쳐서 시스템에 접근이 가능하도록 함

2 주요 취약점 함수 종류

2 | 주요 취약점 함수 종류

1 자바

입력값 처리 함수

함수	설명
getParameter getRawParameter getParameterValues getParameterMap getQueryString	URL에서 사용하는 변숫값과 POST 요청에서 전달되는 값을 처리할 때 사용하는 API이다.
getHeader getHeaders getHeaderNames	HTTP 헤더에서 전달받은 값을 처리할 때 사용하는 API이다.
getRequestURI getRequestURL	URI, URL에서 전달받은 값을 처리할 때 사용하는 API이다.
getCookies	쿠키 값을 받는 데 사용하는 API이다.

※ 출처 : 인터넷 해킹과 보안, 김경곤, 한빛아카데미, 2017

2 | 주요 취약점 함수 종류

1 자바

사용자 세션 처리 함수

함수	설명
setAttribute putValue	현재 세션 내에 데이터를 저장할 때 사용하는 API이다.
getAttribute getValue getAttributeNames getValueNames	현재 세션 내에 저장된 데이터를 가져오는 데 사용하는 API이다.

※ 출처 : 인터넷 해킹과 보안, 김경곤, 한빛아카데미, 2017

2 | 주요 취약점 함수 종류

1 자바

그 외에 살펴볼 함수

함수	설명
FileInputStream FileOutputStream FileReader FileWriter	파일을 읽거나 쓰는 데 사용하는 함수이다.
createStatement executeQuery	데이터베이스에 접근하는 함수이다.

※ 출처 : 인터넷 해킹과 보안, 김경곤, 한빛아카데미, 2017

2 ASP.NET

입력값 처리 함수

함수	설명
Params	URL에 포함된 인자값, POST 요청의 본문 등에서 전달되는 값을 처리하는 데 사용하는 함수이다.
QueryString	요청 쿼리문에 있는 변수명과 값의 집합을 처리하는 함수이다.
Url RawUrl	URL에 포함된 값을 처리하는 데 사용하는 함수이다.
UrlReferer	요청 HTTP Referer 헤더에 정의된 URL 정보를 가져오는 함수이다.
InputStream BinaryRead	클라이언트로부터 받은 요청을 그대로 반환하거나 다른 API에서 받아온 정보를 처리하는 데 사용하는 함수이다.

※ 출처 : 인터넷 해킹과 보안, 김경곤, 한빛아카데미, 2017

2 ASP.NET

사용자 세션 처리 함수

함수	설명
Add	세션에 새로운 항목을 추가하는 함수이다.
Item	해당 Set에 지정한 항목의 값을 가져오거나 설정하는 함수이다.
Keys	해당 Set의 모든 항목명을 반환하는 함수이다.

※ 출처 : 인터넷 해킹과 보안, 김경곤, 한빛아카데미, 2017

2 ASP.NET

그 외에 살펴볼 함수

함수	설명
FileStream StreamReader StreamWriter	파일을 읽거나 쓰는 데 사용하는 함수이다.
SqlCommand SqlDataAdapter OleDbCommand OdbcCommand	데이터베이스에 접근하는 함수이다.

※ 출처 : 인터넷 해킹과 보안, 김경곤, 한빛아카데미, 2017