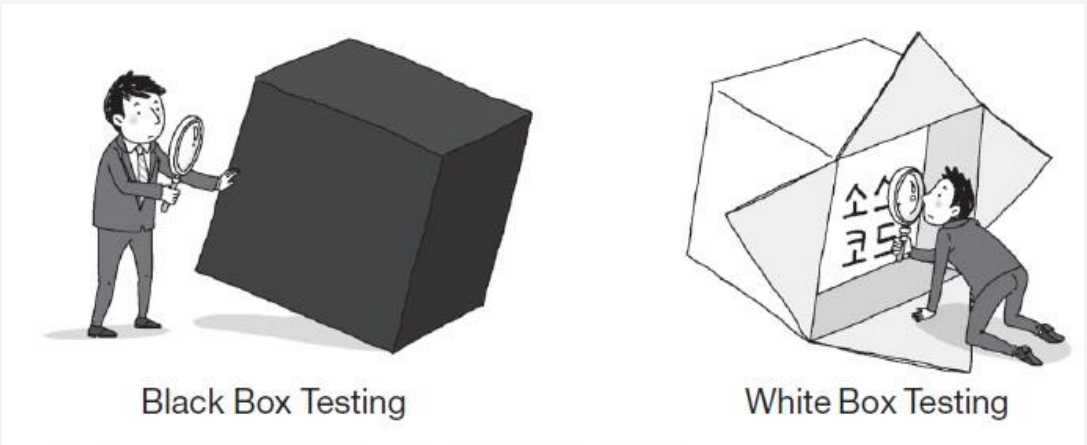
1 소스코드의 취약점

1 'Black Box Testing' 방식

- 소스코드를 보지 않고 웹 애플리케이션의 외부 인터페이스나 구조를 분석하여 취약점을 발견
- White Box Testing에 비해 취약점을 찾는 속도가 빠르며 다양한 취약점 찾기 시도 가능
- 인터페이스 간의 상관관계를 분석하여 취약점이 발생하는 부분 식별 가능(내부가 보이지 않음)

1 'Black Box Testing' 방식

[웹 애플리케이션의 보안 취약점을 찾는 방법]



※ 출처 : 인터넷 해킹과 보안, 김경곤, 한빛아카데미, 2017

1 'Black Box Testing' 방식

- 블랙박스 검사(Black-box testing)는 소프트웨어 검사 방법 중 하나로 어떤 소프트웨어를 내부 구조나 작동 원리를 모르는 상태에서 소프트웨어의 동작을 검사하는 방법을 이르는 말임
- 주로 올바른 입력과 올바르지 않은 입력을 일일이 다 동원하여 올바른 출력을 판별하는 방식으로 검사가 이루어지기 때문에 검사의 진행에 있어 대상이 되는 소프트웨어의 코드나 내부 구조 및 개발 노하우에 대한 정보는 기본적으로 필요로 하지 않음(fuzzing test)

1 'Black Box Testing' 방식

- 필요한 것은 특징, 요구 사항, 검사를 위해 공개된 설계도 등 대외적으로 공개된 사항들이며, 소프트웨어는 무슨 역할을 수행해야 되는가? 와 같이 대상이 되는 소프트웨어의 특징이나 요구 사항 등에 초점을 맞춰 검사가 이루어 짐

- 검사 자체는 기능에 관한 것일 수도 있고, 기능 외의 것에 관한 것일 수도 있음

1 'Black Box Testing' 방식

- 이 방법은 유닛 검사, 인테그레이션 검사, 기능 검사, 시스템 검사, 적합성 검사 이렇게 소프트웨어 검사의 모든 레벨에 적용 가능(유닛 검사, 인테그레이션 검사)

2 ‘ White Box Testing’ 방식

- 개발된 소스코드를 살펴봄으로써 코딩의 취약점을 찾는 것

- 내부 소스코드를 볼 수 있기 때문에 보안 취약점의 존재 유무를 좀 더 확실히 알 수 있으나 시간이 오래 소요 됨(내부를 볼 수 있다)

2 ‘ White Box Testing’ 방식

- 화이트박스 검사(White Box Test) 기법은 소프트웨어 내부 소스 코드를 테스트하는 기법임 소프트웨어를 테스트하는 방법은 크게 블랙박스 검사(Black-Box Test) 기법과 화이트박스 검사(White-Box Test) 기법이 있음

- 블랙박스 검사 기법은 소프트웨어의 내부를 보지 않고, 입력과 출력값을 확인하여,기능의 유효성을 판단하는 테스트 기법이며, 화이트박스 검사 기법은 소프트웨어 내부 소스코드를 확인하는 기법 임

2 ‘ White Box Testing’ 방식

- 화이트박스 테스트를 하는 이유는 내부 소스코드의 동작을 개발자가 추적 할 수 있기 때문에, 동작의 유효성 뿐만 아니라 실행 되는 과정을 살펴봄으로써, 코드가 어떤 경로로 실행되며, 불필요한 코드 혹은 테스트 되지 못한 부분을 살펴볼 수 있음

2 ‘ White Box Testing’ 방식

- 화이트박스 테스트을 하는 부분은 대개 코드의 실행 경로를 확인해야 하기 때문에 시중에 나와 있는 커버리지 분석도구를 많이 활용함 화이트박스 검사 기법은 블랙박스 검사 기법에 비해 많은 시간과 분석을 필요로 하지만 오류가 발생 되는 결함의 위치 등을 파악하는데 매우 유용하게 사용할 수 있음

2 ‘ White Box Testing’ 방식

- 일반적으로 화이트박스 테스트를 수행하는 도구는 커버리지 분석 도구와 많이 연관되며, Junit과 같은 무료 도구가 있는 반면에 크리티컬한 마켓에 사용되는 상용 도구 또한 있음

3 'Gray Box Testing' 방식

- Black Box Testing과 White Box Testing의 장점을 혼합
- 외부에서 보이는 취약점을 웹 애플리케이션 보안 진단을 통해 확인하고, 소스코드에서 접근 통제, 입력값 검증, 세션 처리 문제 등을 같이 살펴보면서 취약점을 조사하는 방식(내부를 볼 수 있다)

3 'Gray Box Testing' 방식

- 소프트웨어의 내부 구조의 일부만 알고 수행하는 시험의 형태
- 그레이박스 시험은 블랙박스 시험과 화이트박스 시험이 혼합된 방식의 시험 형태

3 'Gray Box Testing' 방식

- 블랙박스 시험은 시험자가 내부 구조를 모르고 시험하는데 반해 화이트박스 시험은 내부 구조를 알고 시험 함, 그레이박스 시험은 내부 구조를 일부만 알고 시험하는데, 시험을 위한 테스트 케이스를 만들 때 내부 구조 정보를 활용하며, 시험은 블랙박스 형태로 수행됨, 주로 통합 시험에서 많이 사용 됨 (테스트 케이스, 통합 시험)

4 소프트웨어 개발 보안 가이드

[행정안전부 소프트웨어 개발 보안 가이드와 'OWASP Top 10(2013년)' 취약점 비교]

행정안전부 소프트웨어 개발 보안 가이드		OWASP Top 10
유형	내용	
입력 데이터 검증 및 표현	프로그램 입력값에 대한 검증 누락 또는 부적절한 검증, 데이터의 잘못된 형식 지정으로 인해 발생할 수 있는 보안 약점이다. 예: SQL 인젝션, 자원 삽입, 크로스 사이트 스크립팅 등	A1 인젝션 취약점 A3 크로스 사이트 스크립팅(XSS) A8 크로스 사이트 요청 변조(CSRF)
보안 기능	인증, 접근 제어, 기밀성, 암호화, 권한 관리 등의 보안 기능을 적절하지 않게 구현했을 때 발생할 수 있는 보안 약점이다. 예: 부적절한 인가 허용, 중요 정보 평문 저장, 하드 코딩된 패스워드 등	A2 인증 및 세션 관리 취약점 A4 취약한 직접 객체 참조 A5 보안 설정 오류 A6 민감한 데이터 노출 A7 기능 수준의 접근 통제 누락 A9 알려진 취약점이 있는 컴포넌트 사용

※ 출처 : 인터넷 해킹과 보안, 김경곤, 한빛아카데미, 2017

4 소프트웨어 개발 보안 가이드

[행정안전부 소프트웨어 개발 보안 가이드와
'OWASP Top 10(2013년)' 취약점 비교]

시간 및 상태	동시 또는 거의 동시에 수행을 지원하는 병렬 시스템이나 하나 이상의 프로세스가 동작하는 환경에서 시간 및 상태를 부적절하게 관리하여 발생할 수 있는 보안 약점이다. 예: 경쟁 조건(TOCTOU), 제어문을 사용하지 않는 재귀함수 등	N/A
에러 처리	에러를 처리하지 않거나 불충분하게 처리하여 에러 정보에 중요 정보(시스템 등)가 포함될 때 발생할 수 있는 보안 약점이다. 예: 오류 상황 대응 부재, 오류 메시지를 통한 정보 노출 등	A10 검증되지 않은 리다이렉트 및 포워드
코드 오류	타입 변환 오류, 자원(메모리 등)의 부적절한 반환 등과 같이 개발자가 범할 수 있는 코딩 오류로 유발되는 보안 약점이다. 예: 널 포인트 역참조, 부적절한 자원 해제 등	N/A

※ 출처 : 인터넷 해킹과 보안, 김경곤, 한빛아카데미, 2017

4 소프트웨어 개발 보안 가이드

[행정안전부 소프트웨어 개발 보안 가이드와
'OWASP Top 10(2013년)' 취약점 비교]

캡슐화	중요한 데이터 또는 기능성을 불충분하게 캡슐화했을 때 인가되지 않은 사용자에게 데이터 누출이 가능한 보안 약점이다. 예: 제거되지 않고 남은 디버거 코드, 시스템 데이터 정보 노출 등	A6 민감한 데이터 노출
API 오용	의도된 사용에 반하는 방법으로 API를 사용하거나, 보안에 취약한 API를 사용하여 발생할 수 있는 보안 약점이다. 예: DNS lookup에 의존한 보안 결정 등	A9 알려진 취약점이 있는 컴포넌트 사용

※ 출처 : 인터넷 해킹과 보안, 김경곤, 한빛아카데미, 2017

2 소스코드의 취약점 분석 방법

1 입력값 검증 취약점

- 대표적인 예로는 SQL 인젝션, 크로스 사이트 스크립팅, 위험한 형식의 파일 업로드, 디렉터리 경로 조작 등이 있음(SQL 인젝션, XSS)

2 SQL 인젝션 취약점 코드

1. LoginSqlInjection.java 파일 확인
   - 다음 경로의 java 파일을 열면 로그인 정보를 받아와서 처리하는 코드가 있음
   - C:₩[WebGoat 설치 디렉터리]₩extract₩webapps₩WebGoat₩pluginextracted₩org₩owasp₩webgoat₩plugin₩sqlinjection₩LoginSqlInjection.java

2 SQL 인젝션 취약점 코드

1 LoginSqlInjection.java 파일 확인

[SQL 인젝션 취약점 코드]

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

    try
    {
        String query = "SELECT * FROM employee WHERE userid = " + userId + " and password = '" + password + "'";
        // System.out.println("query: " + query);
        try
        {
            Statement answer_statement = websession.getConnection(s)
                    .createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
            ResultSet answer_results = answer_statement.executeQuery(query);
            if (answer_results.first())
            {
                setSessionAttribute(s, getLessonName() + ".isAuthenticated", Boolean.TRUE);
                setSessionAttribute(s, getLessonName() + "." + SQLInjection.USER_ID, userId);
                authenticated = true;
            }
```

※ 출처 : 인터넷 해킹과 보안, 김경곤, 한빛아카데미, 2017

2 SQL 인젝션 취약점 코드

1 LoginSqlInjection.java 파일 확인

```
public boolean login(WebSession s, String userId, String password)
{
  // System.out.println("Logging in to lesson");
  boolean authenticated = false;
  try
  {
    String query = "SELECT * FROM employee WHERE userid = " + userId + "
and
    password = '" + password + "'";
    // System.out.println("Query:" + query);
    try
    {
      Statement answer_statement = WebSession.getConnection(s)
.createStatement(
      ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
      ResultSet answer_results = answer_statement.executeQuery(query);
```

2 SQL 인젝션 취약점 코드

2 입력값 검증 미비의 문제점 확인

- 문제가 되는 핵심 코드의 실행 과정
  - 사용자가 입력한 아이디와 패스워드는 userId와 password 변숫값으로 저장됨
  - userId와 password 변숫값을 포함하여 SELECT 문장을 만듦
  - 완성된 SELECT 문장을 executeQuery 함수를 통해 실행

2 SQL 인젝션 취약점 코드

2 입력값 검증 미비의 문제점 확인

- 공격자가 아이디, 패스워드에 admin, ‘or”=’를 입력하면 만들어지는 쿼리문

```
SELECT * FROM employee WHERE userid = 'admin'
and password = ''or''=''
```

➡ SQL 인젝션 취약점의 검증을 위해 소스코드를 분석할 때는 SELECT문과 같은 데이터베이스 쿼리문을 검색하고, 필터링 규칙이 포함되어 있는지 살펴봐야 함(특수 문자 사용 금지)

3 크로스 사이트 스크립팅 취약점 코드

1. StoredXss.java 파일 확인
   - 다음 경로의 java 파일을 열면 크로스 사이트 스크립팅 취약점 코드 확인
   - C:₩[WebGoat 설치 디렉터리]₩extract₩webapps₩WebGoat₩plugin_extracted ₩org₩owasp₩ webgoat₩plugin₩StoredXss.java

3 크로스 사이트 스크립팅 취약점 코드

1 StoredXss.java 파일 확인(입력 필터링 없음)

[XSS 취약점 코드]

```
protected void addMessage(webSession s) {
    try {
        String title = HtmlEncoder.encode(s.getParser().getRawParameter(TITLE, ""));
        String message = s.getParser().getRawParameter(MESSAGE, "");

        Connection connection = DatabaseUtilities.getConnection(s);

        String query = "INSERT INTO messages VALUES (?, ?, ?, ?, ? )";

        PreparedStatement statement = connection.prepareStatement(query, ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
        statement.setInt(1, count++);
        statement.setString(2, title);
        statement.setString(3, message);
        statement.setString(4, s.getUserName());
        statement.setString(5, this.getClass().getName());
        statement.execute();
    } catch (Exception e) {
        // ignore the empty resultset on the insert. There are a few more SQL injection errors
        // that could be trapped here but we will let them try. One error would be something
        // like "characters found after end of SQL statement."
        if (e.getMessage().indexOf("No ResultSet was produced") == -1) {
            s.setMessage(getLabelManager().get("CouldNotAddMessage"));
        }
        e.printStackTrace();
    }
}
```

※ 출처 : 인터넷 해킹과 보안, 김경곤, 한빛아카데미, 2017

3 크로스 사이트 스크립팅 취약점 코드

1 StoredXss.java 파일 확인

```
protected void addMessage(WebSession s)
  {
    try
    {
      String title = HtmlEncoder.encode(s.getParser( ).getRawParameter(TITLE,
"""));
      String message = s.getParser( ).getRawParameter(MESSAGE, "");
      :
      statement.setString(2, title);
      statement.setString(3, message);
```

3 크로스 사이트 스크립팅 취약점 코드

2 입력값 검증 미비의 문제점 확인

- 문제가 되는 핵심 코드의 실행 과정
  - 사용자가 입력한 title과 message 변수를 getRawParameter 함수를 통해 받아들임
  - 해당 변수를 필터링 없이 데이터베이스 쿼리문에 포함하여 저장

➡ 사용자가 입력하는 값을 getRawParameter 함수를 통해 받아들이며, 이후 추가적인 필터링 기능이 없는 것 확인(입력 필터링 없음)

3 위험한 형식의 파일 업로드

3 MaliciousFileExecution.java 파일 확인

- 위험한 파일 실행 예제(Malicious File Execution)를 통해 소스코드 보기
- C:₩[WebGoat 설치 디렉터리]₩extract₩webapps₩WebGoat₩plugin_extracted₩org₩owasp₩webgoat₩plugin₩MaliciousFileExecution.java

3 위험한 형식의 파일 업로드

3 MaliciousFileExecution.java 파일 확인(getUserName)

```
File userfile = new File(uploads_and_target_parent_directory
        + TARGET_RELATIVE_PATH + java.io.File.separator
        + s.getUserName() + ".txt");

if(userfile.exists()) {
    makeSuccess(s);
}

Connection connection = DatabaseUtilities.getConnection(s);

ec.addElement(new H1().addElement("WebGoat Image Storage"));

// show the current image
ec.addElement(new P().addElement("Your current image:"));

String image_query = "SELECT image_relative_url FROM mfe_images WHERE user_name = '"
    + s.getUserName() + "'";

Statement image_statement = connection.createStatement(
        ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
ResultSet image_results = image_statement.executeQuery(image_query);

if(image_results.next() == false) {
    // result set was empty
    ec.addElement(new P().addElement("No image uploaded"));
    System.out.println("No image uploaded");
} else {

    String image_url = image_results.getString(1);

    ec.addElement(new IMG(image_url).setBorder(0).setHspace(0).setVspace(0));

    System.out.println("Found image named: " + image_url);
```

※ 출처 : 인터넷 해킹과 보안, 김경곤, 한빛아카데미, 2017

3 위험한 형식의 파일 업로드

3 MaliciousFileExecution.java 파일 확인

```
File userfile = new File(uploads_and_target_parent_directory
      + TARGET_RELATIVE_PATH + java.io.File.separator
      + s.getUserName() + ".txt");
```

```
String image_query = "SELECT image_relative_url FROM mfe_images WHERE
user_name = '" + s.getUserName() + "'";
```

```
Statement image_statement = connection.createStatement(
      ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
ResultSet image_results = image_statement.executeQuery(image_query);
```

3 위험한 형식의 파일 업로드

4 위험한 형식의 파일 업로드에 대한 입력값 검증 미비의 문제점 확인

- getUserName으로 파일명을 가져와서 처리할 때 파일명 및 확장자에 대한 필터링이 없는 것 확인

➡ 웹 애플리케이션의 파일을 업로드하는 부분에서 파일 확장자를 검사하는지 살펴볼 필요가 있음

4 디렉터리 경로 조작

- 파일명을 받아서 처리하는 부분에 파일명에 대한 필터링이 없으면 공격자는 디렉터리 경로를 조작하여 상위 디렉터리에 있는 임의의 파일에 접근할 수 있음 (., ..)

  ➡ 파일명을 받는 file 변수나 getRawParameter 변수를 통해 필터링하고 있는지 확인

4 디렉터리 경로 조작

```
// FIXME: would be cool to allow encodings here -- hex, percent,
// url, etc...
String file = s.getParser().getRawParameter(FILE, "");

// defuse file searching
boolean illegalCommand = getWebgoatContext().isDefuseOSCommands();
if (getWebgoatContext().isDefuseOSCommands())
{

  if (upDirCount(file) == 3 && !file.endsWith("LICENSE"))
  {
    s.setMessage(WebGoatI18N.get("AccessDenied"));
    s.setMessage(WebGoatI18N.get("ItAppears1"));
  }
  else if (upDirCount(file) > 3)
  {
    s.setMessage(WebGoatI18N.get("AccessDenied"));
    s.setMessage(WebGoatI18N.get("ItAppears2"));
  }
  else
  {
    illegalCommand = false;
  }
}

File f = new File((dir + "\\" + file).replaceAll("\\\\", "/"));
```

※ 출처 : 인터넷 해킹과 보안, 김경곤, 한빛아카데미, 2017