Anyframe Jdbc Support Plugin

Version 1.1.0

I.Introduction

jdbc-support 에서는 오픈소스 p6spy 를 확장하여 SQL Injection 보안 위험을 방어할 수 있는 기능 및 최종 실행 쿼리에 대한 로깅(재처리) 기능을 제공한다. jdbc-support plugin은 이를 사용하는데 필요한 라이브러리 및 기본 설정을 포함하고 있다.

Installation

Command 창에서 다음과 같이 명령어를 입력하여 jdbc-support plugin을 설치한다.

```
mvn anyframe:install -Dname=jdbc-support
```

installed(mvn anyframe:installed) 혹은 jetty:run(mvn clean jetty:run) command를 이용하여 설치 결과를 확인해볼 수 있다.

Plugin Name	Version Range
core [http://dev.anyframejava.org/docs/anyframe/plugin/essential/core/1.6.0/reference/htmlsingle/core.html]	2.0.0 > * > 1.4.0

II.Jdbc Support

최종 실행된 SQL 문을 로깅(또는 재처리)하거나 쿼리 실행 시 SQL Injection 패턴을 판별하고 이에 대해 Warning 또는 Replace 처리를 제공하여 보안 위협을 경감시킬 수 있는 기능을 제공하는 anyframe-jdbc-support 에 대해 설명한다. sql logging 기능을 제공하는 유사한 오픈소스 (log4jdbc) 와의 차이점과 jdbc-support 의 설정 및 유의사항 등에 대해 아래의 항목별로 나누어 설명하고자 한다.

- 구현 배경
- Architecture
- Configuration
- 기본 구현 및 사용자 확장 구현 방안
- 기타 고려사항

1.구현 배경

1.1.사용자 요구사항

- SQL Injection 보안 위험을 방어할 수 있는 기능에 대한 사용자 요구사항 구현 방안
- 최종 실행 SQL 문 (preparedStatement 의 바인드 변수까지 반영된)을 확인 및 재처리할 수 있는 기능에 대한 사용자 요구사항 구현 방안

SQL Logging 관련 기 존재하는 오픈소스 기능(ex. log4jdbc) 등이 있으나 SQL Formatting 알아보기 어렵게 변경되고, logging 외 추가적인 처리가 불가한 문제점 등이 존재한다.

위 InjectionPattern / CompleteQuery 는 서로 관련성이 있다. InjectionPattern 의 판별 후 replace 처리 등을 거친 SQL 문을 최종 실행할 때 확인/재처리 할 수 있어야 한다.

위의 두가지 요구사항은 queryService 등 특정 구현에서 많이 아니라 jdbc 기반의 일반적인 persistence 처리에 동일하게 적용할 수 있다면 더 좋을 것이다.

결론적으로 log4jdbc 등의 오픈소스 DataSource Spy 와 같은 형태로 DataSource 기반의 Connection 을 wrapping 하여 실제 실행되는 sql 을 변경 및 재처리 하는 것이 바람직하지만 log4jdbc 를 사용하는 것은 불가능 하였다. cf.) log4jdbc 는 SQL Logging 만을 위한 구현으로 Logging 을 위한 바인드 변수 데이터 등을 Wrapping 한 Statement 모듈에 미리 가지고 있다가 jdbc.sqlonly 등의 logger 설정에 따라 로그로 출력하는 기능만을 고려하여 만들어졌으며, 확장한다 하더라도 SQL 문의 변경 등은 불가능함을 확인하였음

1.2.p6spy

P6Spy 도 마찬가지로 유사한 JDBC proxy 이다. CompleteQuery 처리를 위한 preparedStatement 의 바인드 변수 로깅 등이 가능하면서 JDBC call 을 delegate 하므로 이를 적절히 확장하면 쿼리에 대한 재처리를 적용하여 JDBC 를 실행하는 것도 가능하다. cf.) P6Spy 의 spy.properties 설정은 번거롭고 해당 기능을 사용한 Logging 도 사용치 않을 것이므로 anyframe-jdbc-support 에서는 P6Factory 확장과 AOP 형식으로 사용한다. (ex. infrared-agent 와 동일한 방식)

2.Architecture

2.1.Architecture

dataSource - AOP Method Interceptor - JdbcAspect (p6spy connection wrapping)

InjectionPatternPostProcessor / CompleteQueryPostProcessor - Spring bean 으로 등록하여 JdbcAspect 에 DI 한다.

P6Factory 는 사용자 jdbc app. 내에서 발생하는 getConnection, getPreparedStatement 에 대해 InjectionPatternPostProcessor / CompleteQueryPostProcessor 기능이 적용된 P6Connection, P6PreparedStatement 등을 제공하게 된다.

JDBC 사용 유형에 따라 connection.prepareStatement(sql), preparedStatement.execute(sql) 등의 point 에서 InjectionPattern 처리, CompleteQuery 처리(로깅 또는 기타의 방법으로 사용자에게 전달) 할 수 있다.

P6Spy Connection Wrapping

- P6Factory 확장
- P6Connection 확장
- P6 Statement/PreparedStatement/CallableStatement 확장

2.2.InjectionPatternPostProcessor

SQL Injection 패턴이 존재하는지 detect 하고 이에 대한 warning을 처리한다. return void

```
public void warningPattern(String sql)
```

Injection 패턴에 대해 replace 처리 후 변경된 sql 문을 리턴한다.

```
public String replacePattern(String sql)
```

Default 구현은 멀티건의 warningPatterns, replacePatterns regex(정규표현식) 패턴을 bean 설정 파일의 property로 정의하여 이에 대한 매칭 시 warn 로깅 및 sql replace 를 처리한다. cf.) 위의 Interface 만 맞추면 Implementation 은 사이트 특성에 맞게 자유롭게 구현해도 된다.

2.3.CompleteQueryPostProcessor

최종 SQL 문에 대해 재처리(ex. SQL Logging)한다.

```
public void processCompleteQuery(String sql)
```

Default 구현은 최종 실행 SQL 문을 commons logging 을 통한 Logger 로 출력한다. cf.) 위의 Interface 만 맞추면 Implementation 은 사이트 특성에 맞게 자유롭게 구현해도 된다.

3.Configuration

3.1.JdbcAspect

Spring 의 MethodInterceptor 를 구현하고 있는 JdbcAspect 를 등록한다. 이때 injectionPatternPostProcessor 와 completeQueryPostProcessor 를 dependency 로 등록해 줘야 한다.

```
<bean id="jdbcAspect" class="org.anyframe.jdbc.support.aspect.JdbcAspect">
    <property name="injectionPatternPostProcessor" ref="injectionPatternPostProcessor" />
    <property name="completeQueryPostProcessor" ref="completeQueryPostProcessor" />
</bean>
```

AOP 를 통한 DataSource interrupt 를 처리하기 위해 Spring 의 aop pointcut 표현식을 지정하여 위에서 등록한 jdbcAspect 를 연결해준다.

```
<aop:config>
    <aop:pointcut id="jdbcPointcut" expression="execution(* *..*DataSource.*(..))" />
    <aop:advisor advice-ref="jdbcAspect" pointcut-ref="jdbcPointcut" />
</aop:config>
```

3.2.injectionPatternPostProcessor 및 completeQueryPostProcessor 설정

기본 구현으로 제공되는 injectionPatternPostProcessor

```
<bean id="injectionPatternPostProcessor"
 class="org.anyframe.jdbc.support.impl.DefaultInjectionPatternPostProcessor">
   <property name="warningPatterns">
       <list>
           <value>-{2,}</value>    <!-- check sql comment pattern -->
           <value>'?1'?\s*=\s*'?1'?</value> <!-- check 1 = 1 pattern - ex. '1' = '1' , 1=
 1, '1'='1' -->
           <!-- etc .. your own patterns -->
       </list>
   </property>
   <property name="replacePatterns">
       <map>
           <entry key=";" value="" />      <!-- delete query statement delimiter -->
           <entry key="-{2,}" value="-" />     <!-- ex. sql comment (dash) changing - (one
 dash) -->
           <entry key="(?:or|OR)\s+'?1'?\s*=\s*'?1'?" value="" /> <!-- ex. delete always
 true text pattern - or '1'='1' -->
           <!-- etc .. your own patterns -->
       </map>
   </property>
</bean>
```

기본 구현으로 제공되는 completeQueryPostProcessor

```
<bean id="completeQueryPostProcessor"
 class="org.anyframe.jdbc.support.impl.DefaultCompleteQueryPostProcessor" />
```

위에서 warningPatterns 와 replacePatterns 에 대한 설정은 정규표현식(regular expressions) 으로 작성한다. warningPatterns 나 replacePatterns 를 생략하면 해당 기능은 skip 할 수 있다. (injectionPatternPostProcessor bean 설정 자체를 없애면 안되며 property 태그 영역을 제거)

조직의 보안부서 에서 보안점검 리스트를 정해 특정한 패턴을 방어 가능한지 확인하는 경우가 많고 이러한 요구사항으로 jdbc-support 가 구현되었다. 정규표현식의 작성은 일반적으로 어렵게 느끼는 경우가 많지만 정규식을 사용하면 강력한 패턴 체크가 가능하므로 이를 잘 활용하면 생산성을 높일 수 있다. 정규식에 대해 더 알고 싶은 경우 참조 링크가 도움이 될 것이다.

3.3.log4j 설정

기본 구현으로 제공되는 injectionPatternPostProcessor 및 completeQueryPostProcessor 에 대한 log4j logger 정의는 다음과 같다.

```
<logger name="org.anyframe.jdbc.support.CompleteQueryPostProcessor" additivity="false">
    <level value="INFO" />
    <appender-ref ref="console" />
</logger>

<logger name="org.anyframe.jdbc.support.InjectionPatternPostProcessor" additivity="false">
    <level value="WARN" />
    <appender-ref ref="console" />
</logger>
```

4.기본 구현 및 사용자 확장 구현 방안

4.1.DefaultInjectionPatternPostProcessor

- warningPattern

  p6spy 를 사용하면 실행 SQL 이 넘어오므로 default 구현은 SQL Injection 패턴을 regex 로 정의(Spring 설정파일에 warningPatterns 로 사이트에 특화된 패턴들을 마음대로 정의가능)하고 해당 패턴을 순차적으로 matching 비교하여 match 되면 detect 로 판단한 다음 이에 대해 org.anyframe.jdbc.support.InjectionPatternPostProcessor Logger 로 match된 패턴 문자열 및 실행 sql 를 WARN Level 로그로 남기는 로직으로 warning 처리를 구현하였다.

- replacePattern

  default 구현은 최종 SQL 에 대해 Spring 설정파일에 replacePatterns 로 정의한 regex 패턴 및 replacement 패턴을 순차적으로 String.replaceAll 로 변경한다.

replacePattern 적용 시 유의사항

최종 query 에 대한 전체 일괄 변경이기 때문에 잘못된 패턴 설정 시 SQL Syntax Error를 유발할 수 있다.

cf.) InjectionPatternPostProcessor 는 Spring Bean 으로 등록하므로 Application 의 어떤 영역에서든지 DI 하여 활용 가능함

- ex1. ServletFilter 로 모든 request parameter 에 대해 일괄 InjectionPatternPostProcessor 의 warning/replace 적용 가능.
- ex2. PreparedStatement 문의 bind 변수 처리가 어려운 사용자 전달 query parameter 에 대해 사용자 DAO 영역에서 선별적으로 InjectionPatternPostProcessor 의 replace 적용 가능.

4.2.DefaultCompleteQueryPostProcessor

default 구현은 최종 실행 SQL 에 대해 org.anyframe.jdbc.support.CompleteQueryPostProcessor Logger 로 INFO Level 로 로그를 남긴다.

logger 의 appender 등을 별도로 지정하면 Pattern Layout 이나 target(File, DB ..) 을 자유롭게 조절할 수 있다.

4.3.확장 구현체 Sample - ThreadLocalCompleteQueryPostProcessor

jdbc 실행을 전(flag 설정) / 후(executedQuery 추출 및 ThreadLocal clear) 하여 ThreadLocal 처리에 신경써야 한다.

실제 JDBC 실행이 목적이 아니고 Query 만 추출하고 싶은 경우 Exception 을 throw 하는 것이 필요할 것이다. (cf. queryService 를 사용하는 경우 stackTrace 가 남는 문제 존재(queryService Logger OFF 가능) –> QueryLogException 등을 별도로 정의하여 일괄 AOP 로 Exception 처리하는 영역 등에서 사용자 UI 에 해당 Query 를 되돌려주기 위한 로직을 공통으로 적용하는 것이 바람직함.

Sample Source

```
public class ThreadLocalCompleteQueryPostProcessor extends DefaultCompleteQueryPostProcessor
 {

    @Override
    public void processCompleteQuery(String sql) {
        super.processCompleteQuery(sql);

        if ("Q".equals(SharedInfoHolder.getJobType())) {
            SharedInfoHolder.setExecutedQuery(sql);
            // throw new QueryLogException(sql);
        }
    }

}
```

```
@Test
public void testCompleteQueryPostProcessor() {
    NamedParameterJdbcTemplate jdbcTemplate = new NamedParameterJdbcTemplate(dataSource);

    StringBuffer testSql = new StringBuffer();
    testSql.append("SELECT LOGON_ID, NAME, PASSWORD FROM TB_USER \n");
    testSql.append("WHERE LOGON_ID = :logonId AND PASSWORD = :password \n");

    Map<String, Object> paramMap = new HashMap<String, Object>();
    paramMap.put("logonId", "admin");
    paramMap.put("password", "adminpw");

    // if ThreadLocal flag set - jobType = "Q"
    SharedInfoHolder.setJobType("Q");

    // execute jdbc - cf.) in ThreadLocalCompleteQueryPostProcessor,
    // executes query actually cause it does not throw Exception
    Map<String, Object> resultMap = jdbcTemplate.queryForMap(testSql.toString(), paramMap);
    assertEquals("admin", resultMap.get("logon_id"));
    assertEquals("adminpw", resultMap.get("password"));

    // check the last executed query (CompleteQuery) in ThreadLocal
    assertEquals(
        "SELECT LOGON_ID, NAME, PASSWORD FROM TB_USER \nWHERE LOGON_ID = 'admin' AND
 PASSWORD = 'adminpw' \n",
        SharedInfoHolder.getExecutedQuery());

    // ThreadLocal must be cleared!
    SharedInfoHolder.clearSharedInfo();
}
```

configuration

```
<!-- some ThreadLocal processing added sample -->
<bean id="completeQueryPostProcessor"
    class="org.anyframe.jdbc.support.ext.ThreadLocalCompleteQueryPostProcessor" />
```

5.기타 고려사항

5.1.DBMS Vendor specific 기능 사용을 위한 P6spyNativeJdbcExtractor 적용

OracleLobHandler 사용 시

Oracle 인 경우 OracleLobHandler 등을 사용하게 되면 내부적으로 native connection 객체로 casting 하다가 exception 나는 문제가 있다.

```
org.springframework.dao.InvalidDataAccessApiUsageException:
 OracleLobCreator needs to work on [oracle.jdbc.OracleConnection], not on
 [org.anyframe.jdbc.support.p6spy.P6ILConnection]:
    specify a corresponding NativeJdbcExtractor; nested exception is
 java.lang.ClassCastException: org.anyframe.jdbc.support.p6spy.P6ILConnection
```

이를 회피할 수 있는 P6spyNativeJdbcExtractor 를 추가로 적용해야 한다.

configuration

```
<!-- NativeJdbcExtractor for P6Spy -->
<bean id="nativeJdbcExtractor"
    class="org.anyframe.jdbc.support.p6spy.P6spyNativeJdbcExtractor"
    lazy-init="true">
    <!-- original nativeJdbcExtractor -->
    <property name="nextNativeJdbcExtractor" ref="commonsDbcpNativeJdbcExtractor" />
</bean>

<bean id="commonsDbcpNativeJdbcExtractor"
 class="org.springframework.jdbc.support.nativejdbc.CommonsDbcpNativeJdbcExtractor"
 lazy-init="true" />

<bean id="lobHandler" class="org.springframework.jdbc.support.lob.OracleLobHandler"
 lazy-init="true">
 <property name="nativeJdbcExtractor" ref="nativeJdbcExtractor" />
</bean>
```

5.2.적용 가능 영역

dataSource 기반의 persistence 처리 기술 (ex. Spring jdbcTemplate, Anyframe queryService, iBATIS, Hibernate 등) 전반에 모두 활용 가능하다. (ORM 인 경우도 실제로 jdbc 를 통해 수행되는 내부 sql 문은 확인 가능함.)

5.3.유의사항

log4jdbc 를 활용한 SQL Logging 과는 기능적으로 중복되는 영역이므로 resultSet logging 등을 사용하지 않는다면 log4jdbc 를 중복으로 적용할 필요가 없다.

p6spy 에 대한 library dependency 충돌 우려가 있는데, 현재 p6spy-1.3.jar 를 pom dependency 로 최종 정의하였다. cf.) Anyframe Monitoring Tool 에 대한 에이전트인 infrared-agent-servlet-all-xxx.jar 를 함께 사용하는 것은 추천하지 않는다. infrared-agent 가 p6spy 관련 모듈을 그대로 copy 하여 포함하고 있음(일부 구현 변경)

Oracle 특화 기능을 사용하는 경우(OracleLobHandler 및 P6spyNativeJdbcExtractor) commons-dbcp 에 대한 기본 dependency 라이브러리가 존재해야 한다.

6.Resources

- 참고자료
  - Download p6spy [http://sourceforge.net/projects/p6spy/]
  - regular expression reference - basic [http://www.regular-expressions.info/reference.html]
  - regular expression reference - advanced [http://www.regular-expressions.info/refadv.html]