**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В. И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по лабораторной работе**

**по дисциплине «Разработка защищенных программных систем»**

Тема: «SpEL injection»

|  |  |  |
| --- | --- | --- |
| Студентка гр. 6303 |  | Блинникова Ю.И. |
| Преподаватель |  | Юшкевич И. А. |

Санкт-Петербург

2020

# Задание

Тема: SpEL injection. Язык: Java.

Вы должны разобраться в теме и подготовить следующие материалы:

- Теоретический материал по выбранной теме, который состоит из следующих пунктов.

- Описание уязвимости

- Типичные примеры реализации уязвимости

- Как ее можно обнаружить в коде и во время тестирования.

- Как ее эксплуатировать.

- Как ее необходимо устранить.

- Примеры из открытых источников, где она была обнаружена.

- Реализация лабораторной работы, которая показывает, как та или иная уязвимость выглядит на практике.

Лабораторная работа должна представлять собой исходный код приложения и пошаговый мануал, как она может быть развернута из исходного кода приложения.

Лабораторная работа - приближенное к реальному веб-приложение. Под этим подразумевается наличие трех составляющих:

- Дизайна;

- Разнообразной функциональности;

- “Легенда” - описание веб приложения.

Для каждого веб-приложения должен быть реализован скрипт проверки наличия уязвимости либо ее устранения. При эксплуатации уязвимости пользователем, приложение должно предоставлять “флаг”, который подтверждал бы факт эксплуатации. Все приложения должны использоваться на ОС Linux, без использования Docker. Приложение не должно использовать коммерческие продукты в своей функциональности.

# Теоретический материал

## Описание уязвимости

SpEL — это язык выражений, созданный для Spring Framework, который поддерживает запросы и управление графом объектов во время выполнения. SpEL создан в виде API-интерфейса, позволяющего интегрировать его в другие приложения и фрэймворки.

SpEL используется в Spring Framework, например, в Security, Data. Для языков он может использоваться в Java, Kotlin, Scala и других технологиях, основанных на JVM.

SpEL поддерживает широкий спектр функций, таких как:

* логические и реляционные операторы;
* литеральные выражения;
* регулярные выражения;
* доступ к свойствам, массивам, спискам, картам (map);
* вызов метода;
* операторы отношения;
* присвоение;
* вызов конструкторов;
* тернарный оператор.

## Типичные примеры реализации уязвимости

В следующем коде представлен API SpEL для оценки буквального строкового выражения «Hello World».

ExpressionParser parser = **new** SpelExpressionParser();

Expression exp = parser.parseExpression("**'Hello World'**");

String message = (String) exp.getValue();

Значение переменной сообщения - просто «Hello World».

Классы и интерфейсы SpEL, находятся в пакетах org.springframework.expression и его подпакетах и spel.support.

Интерфейс ExpressionParser отвечает за синтаксический анализ строки выражения. В этом примере строка выражения — это строковый литерал, обозначенный окружающими одинарными кавычками. Интерфейс Expression отвечает за оценку ранее определенной строки выражения. При вызове parser.parseExpression и exp.getValue могут возникать два исключения: ParseException и EvaluationException соответственно.

SpEL поддерживает широкий спектр функций, таких как вызов методов, доступ к свойствам и вызов конструкторов.

В качестве примера вызова метода мы вызываем метод concat для строкового литерала.

ExpressionParser parser = **new** SpelExpressionParser();

Expression exp = parser.parseExpression("**'Hello World'.concat('!')**");

String message = (String) exp.getValue();

Теперь значение сообщения - «Hello World!».

В качестве примера вызова свойства JavaBean можно вызвать свойство String Bytes, как показано ниже.

ExpressionParser parser = **new** SpelExpressionParser();

*// invokes 'getBytes()'*

Expression exp = parser.parseExpression("**'Hello World'.bytes**");

**byte**[] bytes = (**byte**[]) exp.getValue();

SpEL также поддерживает вложенные свойства с использованием стандартной точечной нотации, то есть prop1.prop2.prop3 и установку значений свойств.

Доступны также публичные поля.

ExpressionParser parser = **new** SpelExpressionParser();

*// invokes 'getBytes().length'*

Expression exp = parser.parseExpression("**'Hello World'.bytes.length**");

**int** length = (Integer) exp.getValue();

В переменную length будет записано число 11.

Вместо строкового литерала можно вызвать конструктор String. Тогда строка message будет хранить строку “ HELLO WORLD”.

ExpressionParser parser = **new** SpelExpressionParser();

Expression exp = parser.parseExpression("**new String('hello world').toUpperCase()**");

String message = exp.getValue(String.**class**);

## Как ее можно обнаружить в коде и во время тестирования

1) Если есть возможность осуществлять поиск по коду приложения, то необходимо искать такие ключевые слова: SpelExpressionParser, EvaluationContext и parseExpression.

2) Важные для Spring указатели #{SpEL}, ${someProperty} и T(javaclass)

Выражения SpEL начинаются с символа # и заключаются в фигурные скобки: # \{Выражение}. На свойства можно ссылаться аналогичным образом, начиная с символа $ и заключая в фигурные скобки: $ \{property.name}.

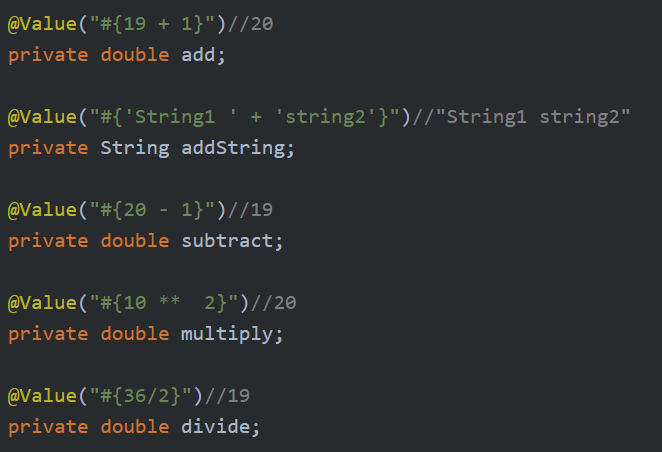
Заполнители свойств не могут содержать выражения SpEL, но выражения могут содержать ссылки на свойства:



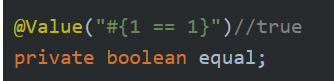
В приведенном выше примере предположим, что someProperty имеет значение 2, поэтому результирующее выражение будет равно 2 + 2, что будет равно 4.

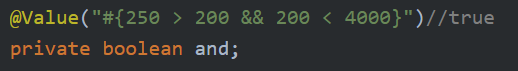
Примеры арифметических операторов:

Все основные арифметические операторы поддерживаются.



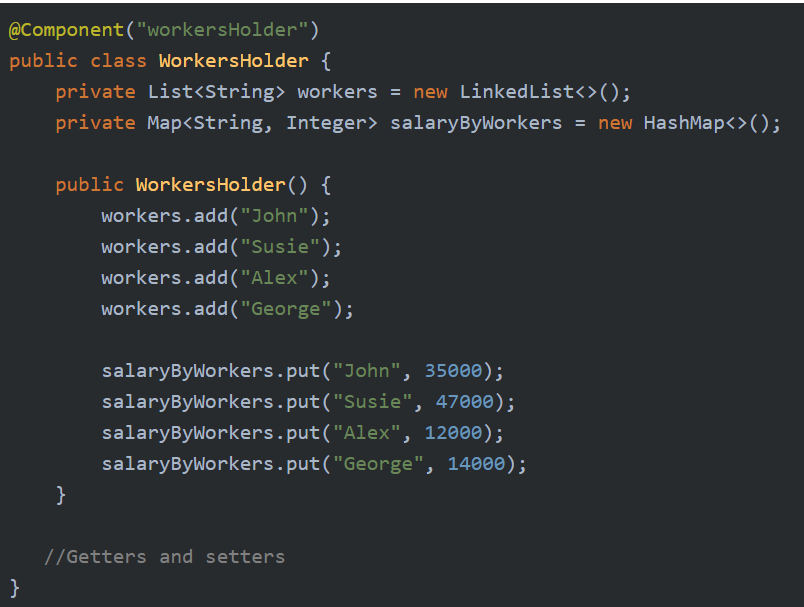
Пример логических операторов:



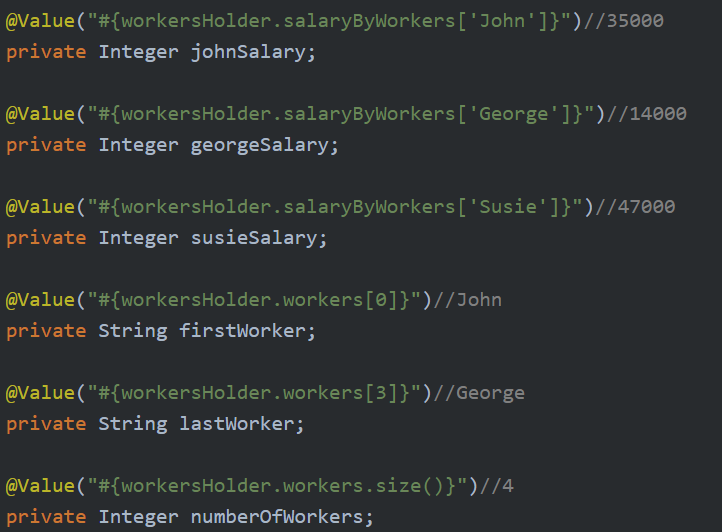


Примеры доступа к объектам List и Map.

С помощью SpEL мы можем получить доступ к содержимому любого Map или List в контексте. Мы создадим новый компонент workersHolder , который будет хранить информацию о некоторых работниках и их зарплатах в List и Map :



Теперь мы можем получить доступ к значениям коллекций, используя SpEL:



## Как ее эксплуатировать

T(org.apache.commons.io.IOUtils).toString(T(java.lang.Runtime).getRuntime().exec("cmd /c dir").getInputStream())

''.class.forName('java.lang.Runtime').getRuntime().exec('calc.exe')

## Как ее необходимо устранить

Необходимо использовать вместо полнофункционального StandardEcalutionContext более урезанный по возможностям SimpleEvaluationContext.

## Примеры из открытых источников

Примеры уязвимостей SpEL:

* [cve-2018-1273](https://tanzu.vmware.com/security/cve-2018-1273)
* [cve-2017-8046](https://tanzu.vmware.com/security/cve-2017-8046)
* [cve-2017-4971](https://www.securitylab.ru/analytics/488855.php)

# Реализация лабораторной работы, которая показывает,

## Как та или иная уязвимость выглядит на практике

Приложение представляет собой приложение по созданию заметок. На рис. 1 – рис. 2 представлен интерфейс приложения.

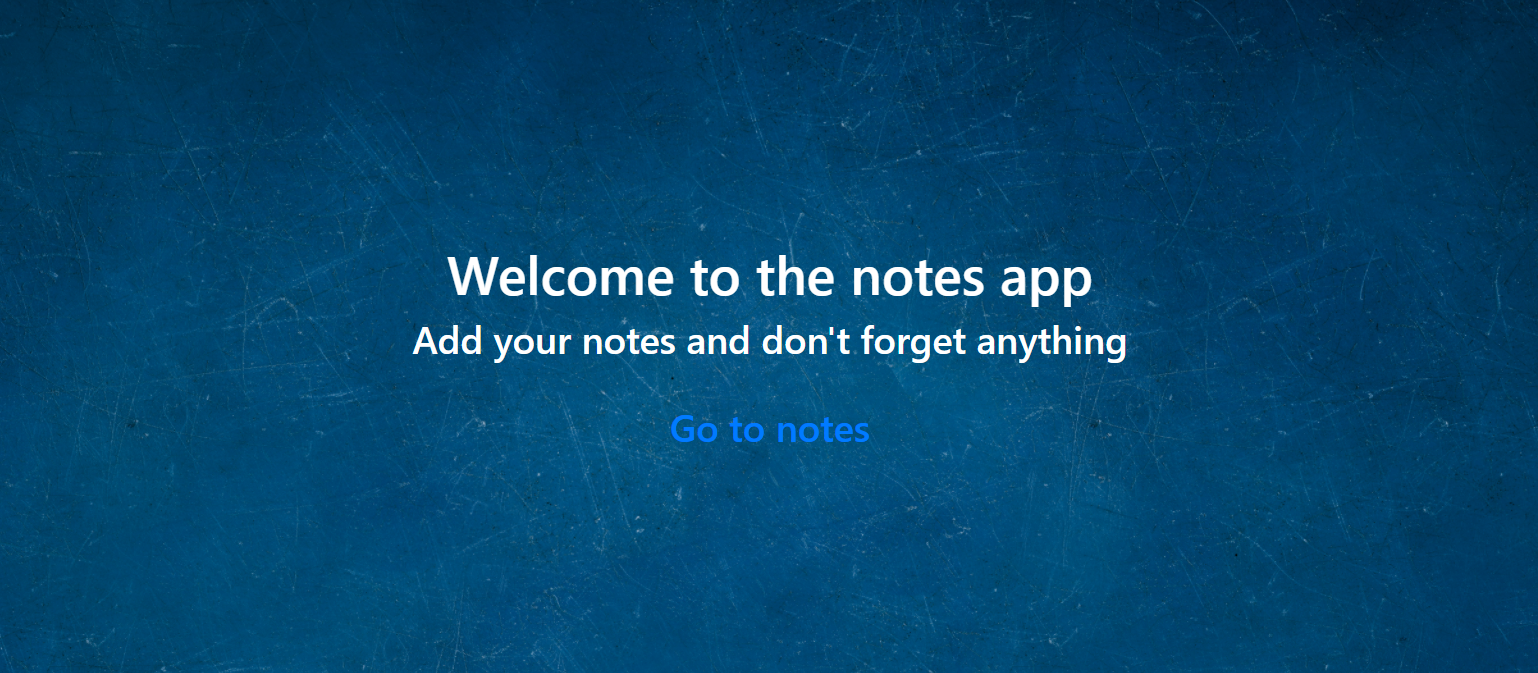


Рисунок 1 – Стартовая страница

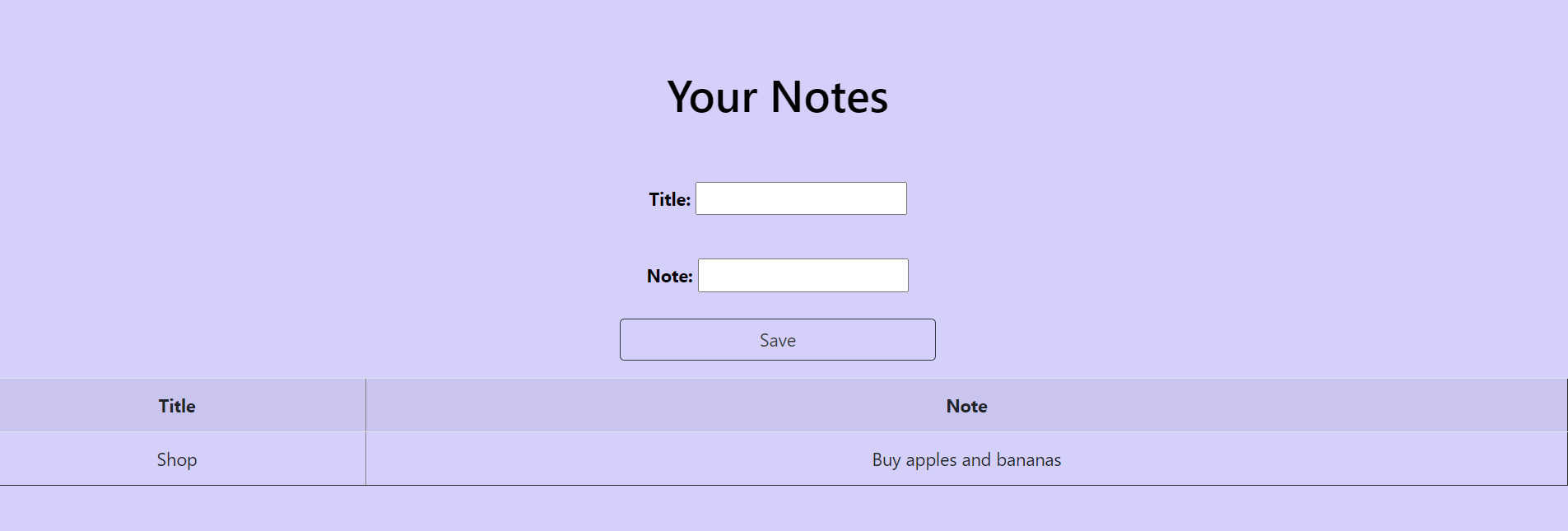


Рисунок 2 – Страница с заметками

Стартовая страница имеет ссылку на страницу с заметками. Страница с заметками имеет 2 поля для ввода: title и note.

Для обнаружения зависимости можно воспользоваться строкой:

''.class.forName('java.lang.Runtime').getRuntime().exec('calc.exe')

Тогда будет вызван калькулятор (рис.3)

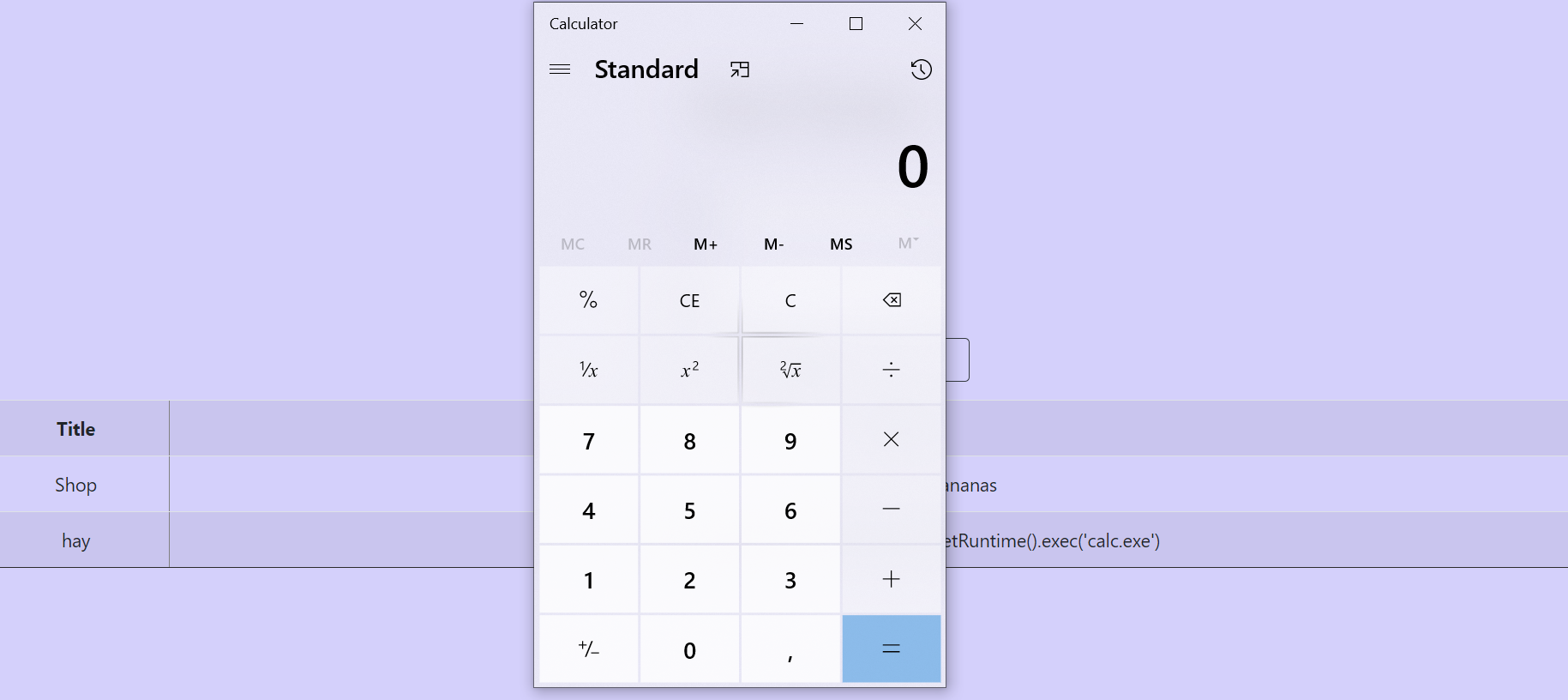


Рисунок 3 – Поиск уязвимой строки

В поле для ввода note имеется уязвимость. Обработка полученных данных происходит с помощью контекста StandardEvaluationContext, который может обращаться к Java-классам (рис. 4)



Рисунок 4 – реализация уязвимости в коде

При вводе в поле note строки в системе Windows (рис. 5):

T(org.apache.commons.io.IOUtils).toString(T(java.lang.Runtime).getRuntime().exec("cmd /c dir").getInputStream())

Или при вводе в поле note строки в системе Linux:

T(org.apache.commons.io.IOUtils).toString(T(java.lang.Runtime).getRuntime().exec("cmd /c ls").getInputStream())

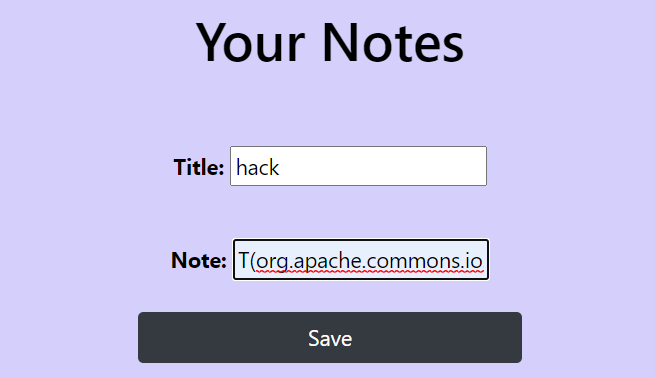


Рисунок 5 – Ввод уязвимости в поле Note

Будет выведено содержимое корневой папки, в которой можно обнаружить секретную папку с названием “spel\_parser\_flag” (рис. 6)

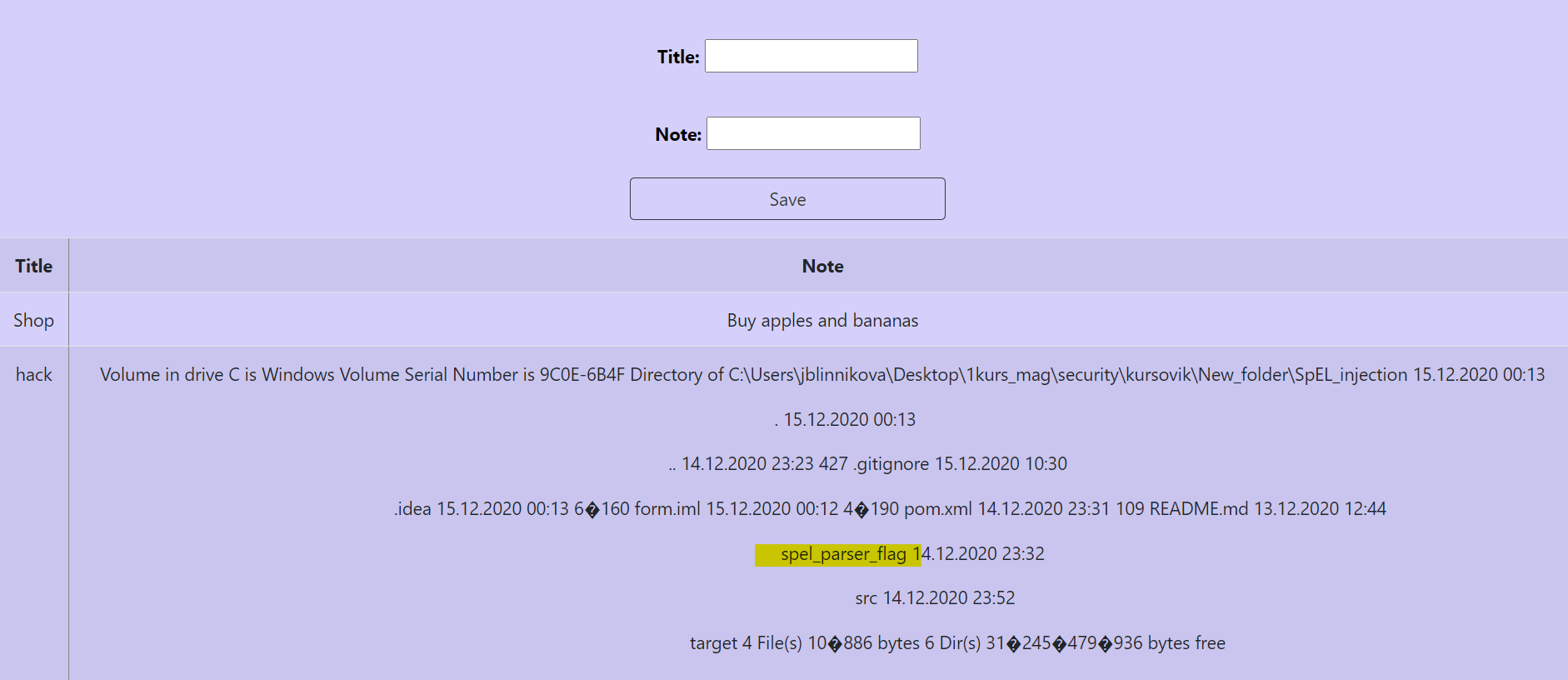


Рисунок 6 – Результат после ввода уязвимости

Для исправления данной уязвимости необходимо воспользоваться другим, более безопасным контекстом SimpleEvaluationContext (рис. 7)

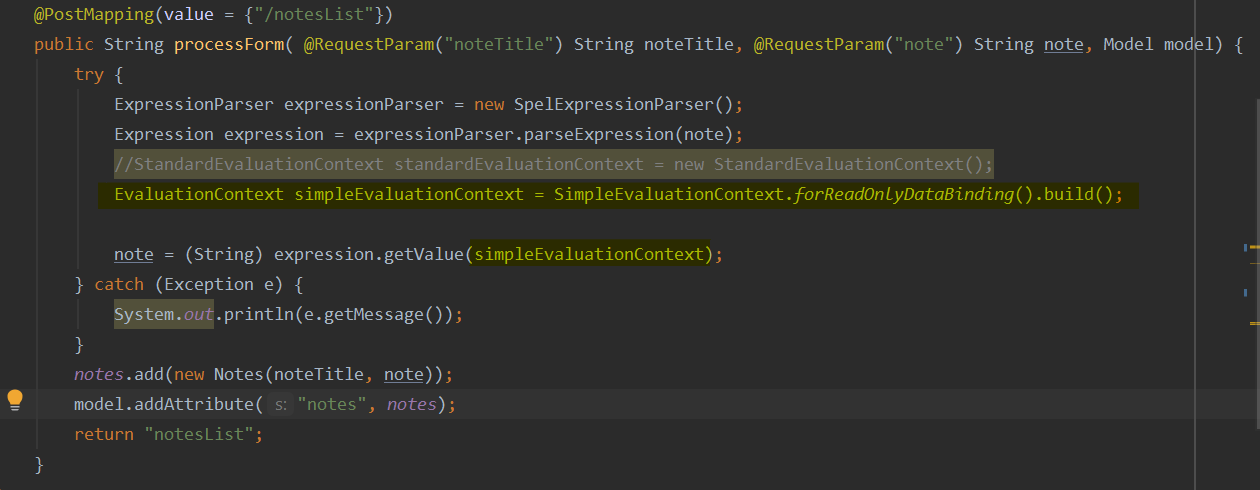


Рисунок 7 – Реализация исправления уязвимости

Данный контекст не позволит выполнить уязвимую строку и данные будут сохранены только в виде строки (рис. 8).

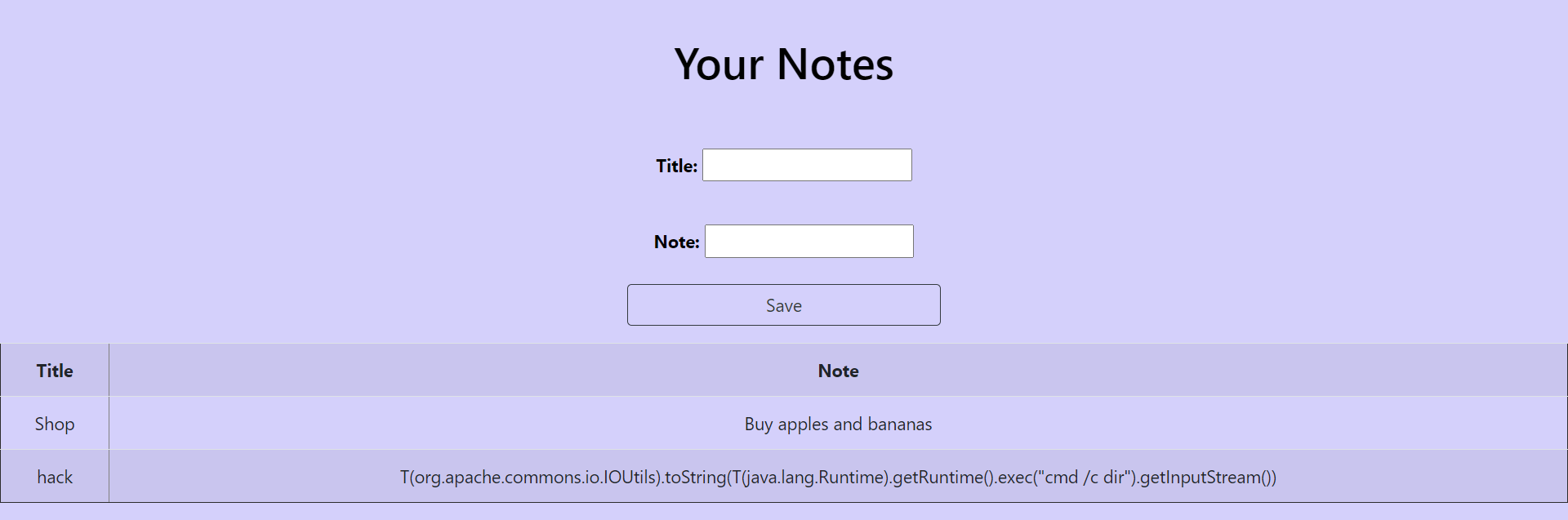


Рисунок 8 – Результаты работы после исправления уязвимости

## Мануал по развертыванию

Код можно скачать на github по ссылке: <https://github.com/JoolsBlinnikova/SpEL_injection.git>

Необходимо скачать в папку данный проект:

1) Создать папку, зайти в эту папку и выполнить команду:  
git clone <https://github.com/JoolsBlinnikova/SpEL_injection.git>

2) Затем запустить сервер в данной папке:

a) mvn clean install

б) mvn tomcat7:run

## Скрипт проверки

Создан sh скрипт проверки и помещен в корневую директорию (рис. 9). Запуск осуществляется из корневой директории с помощью команды:

checkSpELInjection.sh

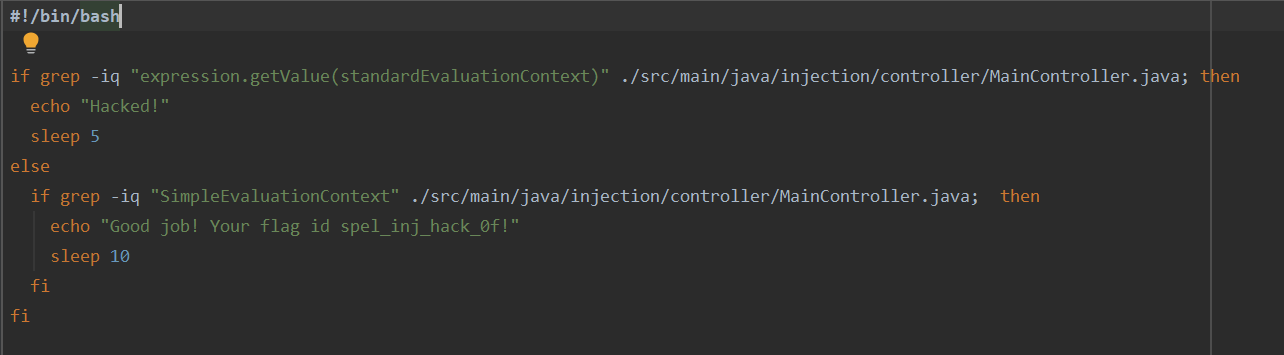


Рисунок 9 – Реализация скрипта проверки

Результаты работы скрипта представлены на рис. 10 – рис.11.

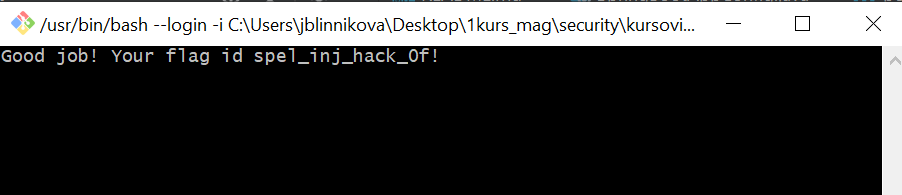


Рисунок 10 – Уязвимость исправлена

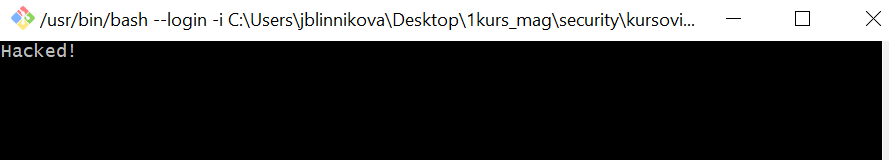


Рисунок 11 – Уязвимость не исправлена

## Выводы

В рамках данной лабораторной работы было реализовано приложение, реализующее уязвимость SpEL. В приложениях Spring рекомендуется использовать вместо полнофункционального StandardEcalutionContext более урезанный по возможностям SimpleEvaluationContext, так как у SimpleEvaluationContext нет, и ссылаться на другие бины. Поэтому данный контекст более безопасный, чем StandardEcalutionContext.