Отчет по лабораторной работе №8
«Проект OWASP WebGoat»

Выполнил: студент гр. 53501.3 Греченко Л.В.

Изучение

1. Изучить описания десяти самых распространенных веб-уязвимости согласно рейтингу OWASP

A1 Внедрение кода

Внедрение кода всегда являлось одной из самым значимых и распространенных уязвимостей Web-приложений, поэтому не удивительно, что этот тип уязвимостей занимают верхнюю строчку списка OWASP. Существует множество разновидностей этой уязвимости, но до сих пор самой печально известной из них является внедрение SQL-кода, успешно используемое хакерами на протяжении более десяти лет.
Внедрение SQL-кода заключается в том, что злоумышленник вводит SQL-команды в поле ввода. Если код Web-приложения не отфильтровывает вводимые символы, то на Web-сервере можно запустить SQL-команды и выполнять прямые запросы к внутренней базе данных в обход сетевых средств защиты. Путем внедрения SQL-кода злоумышленник может заполучить таблицы с данными, изменить записи в таблицах и даже полностью удалить базу данных.

A2 Некорректная аутентификация и управление сессией

Вторая наиболее опасная уязвимость в списке OWASP связана с методами аутентификации и защиты пользовательских сеансов в Web-приложении. Существует множество типов этой уязвимости, одним из которых является несанкционированное использование сеанса.
Злоумышленник может попытаться украсть (несанкционированно использовать) Web-сеанс пользователя, узнав его секретный идентификатор. Зная секретный идентификатор сеанса, атакующий может представиться Web-серверу аутентифицированным пользователем и скомпрометировать его учетную запись. Если приложение недостаточно хорошо защищает идентификаторы сеансов (например, отображает идентификаторы внутри URL-адреса вместо использования cookie-файлов), то злоумышленнику очень просто получить идентификатор сеанса, обманув пользователя.

A3 Межсайтовый скриптинг (XSS)

Межсайтовый скриптинг (cross-site scripting, XSS) – это еще одна разновидность атаки на Web-приложения, сохраняющая популярность уже много лет. Если Web-приложение содержит XSS-уязвимость, то злоумышленник может внедрить на Web-страницу вредоносный сценарий, выполняющийся при загрузке страницы пользователем.

A4 Небезопасные прямые ссылки на объекты

Брешь, обусловленная наличием небезопасных прямых ссылок на объекты, может привести к тому, что авторизованный пользователь Web-приложения может получить неавторизованный доступ к привилегированным функциям и данным. Если в коде приложения неграмотно или неправильно реализованы методы работы с информационными объектами (например, с файлами, каталогами или ключами баз данных), то пользователи, не обладающие требуемыми привилегиями, могут обойти средства защиты, реализованные в приложении. Используя эту уязвимость, пользователи могут изменять значения параметров таким образом, чтобы непосредственно обращаться к объектам, доступ к которым им запрещен.

A5 Небезопасная конфигурация

Небезопасная конфигурация может присутствовать во всех компонентах Web-приложения, включая платформу (операционную систему), Web-сервер, базы данных или инфраструктуру. Распространенными ошибками являются запущенные без необходимости службы, учетные записи администратора с параметрами по умолчанию, доступное для онлайн-просмотра содержимое файлов и директорий. Однако излюбленной темой хакеров является небезопасная обработка сообщений об ошибках.

A6 Утечка чувствительных данных

Web-приложение должно правильно работать с уязвимыми данными и защищать их. Для предотвращения утечки данных в процессе их обработки, передачи и хранения необходимо обеспечивать защиту на всех уровнях при помощи надежных процедур контроля доступа, общепризнанных криптоалгоритмов и методов управления ключами шифрования.

A7 Отсутствие контроля доступа к функциональному уровню

Пользователи, работающие с Web-приложением, могут иметь различные уровни полномочий. В таких случаях базовая аутентификация может не справиться с тем, чтобы не допустить использования пользователями полномочий, функций и данных, доступ к которым им запрещен.

Если защита привилегированных функций заключается лишь в том, что они просто спрятаны в коде приложения, то злоумышленник, скомпрометировавший учетную запись с недостаточными привилегиями, может попытаться получить доступ к закрытым функциями (и данным), подбирая вызовы к ним.

A8 Подделка межсайтовых запросов (CSRF)

Подделка межсайтовых запросов (cross-site request forgery, CSRF) основана на том, что аутентифицированного пользователя Web-приложения обманным путем заставляют запустить вредоносный сценарий, который выполняет действия от имени законного пользователя. Например, на компьютере пользователя может скрытно выполняться CSRF-сценарий, отправляющий Web-приложению запрос на изменение пароля, а после успешной авторизации пользователя Web-приложение исполняет этот запрос.

A9 Использование компонентов с известными уязвимостями

Любой сторонний компонент Web-приложения – будь то двоичный или исходный код, коммерческое или Open Source-приложение – должен проверяться на отсутствие уязвимостей.

Практически каждое Web-приложение использует Open Source-компоненты, например, библиотеку OpenSSL, обеспечивающую TLS/SSH-шифрование Web-сайтов (HTTPS). В апреле 2014 года в нескольких версиях этой библиотеки была обнаружена критическая уязвимость CVE-2014-0160, известная как Heartbleed.

A10 Непроверенные перенаправления и переходы

Web-приложения часто перенаправляют пользователей на другие страницы при помощи определенных параметров. Если эти параметры не проверяются, то злоумышленник может обманным путем перенаправить пользователя на вредоносную Web-страницу, заставив его раскрыть свой пароль и другие важные данные. Например, злоумышленник может создать сообщение электронной почты с фишинговой ссылкой, содержащей имя требуемого Web-сайта. При этом отвечающий за перенаправление параметр, содержащийся в конце URL-адреса, может быть не виден пользователю, в результате чего пользователь с большой вероятностью щелкнет по этой ссылке или скопирует ее в Web-браузер, так как доменное имя в URL-адресе не вызовет у него никаких подозрений.

Практическое задание

Запустить уязвимое приложение WebGoat.

Запустить сканер безопасности ZAP. Запустить инструмент Mantra, настроить его для использования ZAP в качестве прокси-сервера

1. Недостатки контроля доступа

В схеме управления доступом на основе ролей, роль представляет собой набор прав доступа и привилегий. Пользователю может быть назначена одна или несколько ролей. Схема управления доступом на основе ролей, как правило, состоит из двух частей: управление разрешения роли и назначение роли. Нарушенная схема управления доступом на основе ролей может предоставить пользователю доступ, который не разрешен для его роли.

- **Использование матрицы контроля доступа**

Каждый пользователь входит в состав роли, которой позволен доступ только к определенным ресурсам. Были изучены правила контроля доступа, регулирующих этот сайт. В результате вычислили, что только [Admin] группы Larry имеет доступ к ресурсу “Account Manager’.

- **Обман схемы контроля доступа на основе пути**

В схеме управления доступом на основе пути, злоумышленник может заменить информацию относительного пути. Злоумышленник может использовать относительные пути к файлам, которые обычно не являются непосредственно доступными любому. В данной задаче мы перехватили запрос и заменили путь к файлу.

- **Управление доступом на основе роли**

Целью является изучение правил контроля доступа, регулирующих этот сайт. Каждая роль имеет разрешение на определенные ресурсы (AF). Каждому пользователю присваивается одна или несколько ролей. Только пользователь с ролью [Admin] должен иметь доступ к ресурсам 'F'. При успешной атаки, пользователь не имеющий роль [Admin] может получить доступ к ресурсам F.

Этап 1. Обход презентационного уровня управления доступом.

Для того, чтобы от имени пользователя была возможность удалить админа необходимо узнать имя действия для удаления. Мы перехватили действие админа *“employee_id=101&action=DeleteProfile”* связанное с удалением пользователя и узнали, что действие называется “DeleteProfile”. В следующем запросе подменили id на id админа и удалили его.

Этап 2. Добавить контроль доступа бизнес-слоя.

Реализовать исправление несанкционированного доступа к функции удаления. Чтобы сделать это, мы должны будете добавить код в class org.owasp.webgoat.lessons.RoleBasedAccesControl.RoleBasedAccessControl.java.

```
if(!isAuthorized(s, getUserId(s), requestedActionName)) {
  throw new UnauthorizedException(); }
```

Этап 3. Нарушение контроля доступа уровня данных.
На этом этапе мы хотим посмотреть профиль другого человека. У нас есть разрешение на действия ViewProfile, но мы не имеем разрешение увидеть профиль другого работника. На данном Этапе при действие просмотра профиля мы заменяем EMPLOYEE_ID например на 101 и видим чужой профиль.
Этап 4. Реализовать исправление отрицания несанкционированного доступа к данным. Мы должны быть уверены, что пользователь имеет право выполнить действие, и что он имеет право делать это действие на конкретного сотрудника. Таким образом, мы должны проверить его разрешение. Необходимо написать код в классе org.owasp.webgoat.lessons.RoleBasedAccesControl.RoleBasedAccessControl.java.

```
if(!isAuthorized(s, getUserId(s), requestedActionName)) {
  throw new UnauthorizedException(); }
if(!action.isAuthorizedForEmployee(s, getUserId(s),
s.getParser().getIntParameter(RoleBasedAccessControl.EMPLOYEE_ID, 0))) {
  throw new UnauthorizedException(); }
```

2. Безопасность AJAX

- **Скрипты кросс-сайта основанные на DOM.**

Объектная модель документа (DOM), представляет интересную проблему с точки зрения безопасности. Это позволяет содержанию веб-страницы динамически изменяться, но это может быть использовано злоумышленниками во время инъекции вредоносного кода . В XSS (Cross Site Scripting — «межсайтовый скриптинг») вредоносное внедрение кода, может произойти, когда непроверенный пользовательский ввод используется непосредственно изменяя содержимое страницы на стороне клиента.
Этап 1. Ввели "<IMG SRC="images/logos/owasp.jpg"/>" и нажали кнопку “Подтвердить”.
Этап 2. Ввели "" и нажали кнопку “Подтвердить”.
Этап 3. Ввели Enter "<IFRAME SRC="javascript:alert('XSS');"></IFRAME>" и нажали кнопку “Подтвердить”.
Этап 4. Ввели "Please enter your password:
<input type = "password" name="pass"/><button onClick="javascript:alert('I have your password: ' + pass.value);">Submit</button>

" и нажали кнопку “Подтвердить”.
Этап 5. Изменили код в файле DOMXSS.js для обеспечения безопасной работы.

```
function displayGreeting(name) {
        if (name != ''){
                document.getElementById("greeting").innerHTML="Hello, " +
escapeHTML(name); + "!";    } }
```

- **DOM внедрение**

AJAX требует XML связи между браузером и веб-приложением. При просмотре источника страницы HTML, мы видим использование XMLHttpRequest.
The XML response contains JavaScript that will activate the button so that you are able to click on it. This requires you to inject JavaScript to manipulate the Document Object Model

of the HTML page in the browser. This requires intercepting the HTTP response in WebScarab!
Ответ XML-содержит JavaScript, который активирует кнопку, так что мы нажимаем на кнопку. Перехватываем HTTP ответ в ZAP. Заменяем в ответе состояние кнопки на document.form.SUBMIT.disabled = false. В результате кнопка активна.

- **Фильтрация на стороне клиента**

Необходимо всегда отправлять клиенту только информацию, к которой он должен иметь доступ. В этом уроке необходимо воспользоваться посторонней информацией возвращаемой сервером, чтобы обнаружить информацию, к которой вы не должны иметь доступ.
Сначала используем любое лицо из списка и открываем Firebug. Видим, что имеется скрытая таблица с информацией, в том числе о заработной плате. В этой же таблице мы видим Невилла, который отсутствует в нашем списке.
Для обеспечения безопасности необходимо изменить код clientSideFiltering.jsp.

```
StringBuffer sb = new StringBuffer();
                sb.append("/Employees/Employee/UserID | ");
        sb.append("/Employees/Employee/FirstName | ");
        sb.append("/Employees/Employee/LastName | ");
        sb.append("/Employees/Employee/SSN | ");
        sb.append("/Employees/Employee/Salary ");
                String expression = sb.toString();
```

- **Защита принципа одинакового источника**

Ключевым элементом AJAX является XMLHttpRequest (XHR), который позволяет JavaScript производить асинхронные вызовы со стороны клиента к серверу. Тем не менее, в качестве меры безопасности эти запросы могут быть сделаны только на сервере, с которого страница клиента возникла.
Это упражнение демонстрирует защиту принципа одинакового источника. Запросы XHR могут быть приняты только к исходному серверу. Попытки передать данные на не оригинальный сервер не удастся.

- **XML внедрение**

Приложения AJAX использует XML для обмена информацией с сервером. Этот XML может быть легко перехвачен и изменен злоумышленником.
После того, как мы ввели идентификатор учетной записи, нам показывают баланс и продукты, которые мы можем себе позволить. Мы перехватываем XML и добавляем пункты в свой допустимый набор наград.

- **JSON внедрение**

JavaScript Object Notation (JSON), это простой и эффективный формат для легкого обмена данными. JSON может быть во многих формах, таких как массивы, списки, хеш-таблицы и другие структуры данных. JSON широко используется в AJAX и Web 2.0 приложениях и предпочитается программистами перед XML из-за его простоты использования и скорости. Тем не менее, JSON, XML, как склонны к Атаке Внедрения.Злоумышленник может внедрить ответ от сервера и ввести некоторые произвольные значения.
В данном уроке мы изменили стоимость дорого билета на самолет с помощью изменения ответа от сервера, перехваченного в ZAP.

- **Атаки “тихих” транзакций**

Любая система, которая молча обрабатывает транзакции с использованием единого представления опасен для клиента. Например, если обычное веб-приложение

позволяет простое представление URL, сессия атаки позволит злоумышленнику завершить транзакцию без разрешения пользователя. В Ajax, все еще хуже: транзакция молчит; это происходит без какой-либо обратной связи с пользователем на странице, так скрипт внедренной атаки может быть в состоянии украсть деньги от клиента без авторизации.
Тестируемое веб-приложение использует JavaScript на клиенте, чтобы начать транзакцию для передачи денег. Рассматривая источник HTML показывает, что используются две функции JavaScript.
Функция processData () вызывается, когда пользователь заполняет номер счета и сумму перевода.Функция processData () проверяет, имеет ли пользователь достаточный баланс перед началом транзакции. После проверки баланса, вызывается функция JavaScript submitData(accountNo, balance), которая на самом деле представляет необходимую информацию, номер счета и переводимую сумму, обратно веб-приложению. Мы в состоянии вызвать эту функцию submitData JavaScript (AccountNo, баланс) от браузера, т.о. обходим проверку на стороне клиента и выполняем эту сделку тихо, без дополнительного согласования или цифровой подписи пользователя.

- **Опасные использование Eval**

Это всегда хорошо - проверить весь ввод на стороне сервера. XSS может возникнуть, когда непроверенный пользовательский ввод непосредственно отражается в HTTP ответе . В этом уроке, непроверенные введенные пользователем данные используются в сочетании с вызовом Javascript Eval (). В XSS нападение, атакующий может обработать URL со скриптом атаки и сохранить его на другой веб-сайт.

- **Insecure client storage**

С помощью Firebug выявляем код купона на получение непреднамеренной скидки. Затем, используем проверку на стороне клиента, чтобы подать заказ со стоимостью ноль.

3. Недостатки аутентификации (Authentication Flaws)

- **Password Strength**

Аккаунты настолько защищены, насколько надежен пароль. Большинство пользователей используют один и тот же слабый пароль везде. Для защиты приложений от атак грубой силы у приложения должны быть хорошие требования к паролям. Пароль должен содержать строчные буквы, заглавные и цифры. Чем длиннее пароль, тем лучше. В данном уроке мы проверили надежность паролей на стороннем сайте, который рассчитывал время необходимое для взлома пароля.

- **Forgot Password**

Веб-приложения часто предоставляют своим пользователям возможность восстановить забытый пароль. К сожалению, многие веб-приложения не реализовывают механизм должным образом. Информация, необходимая для проверки личности пользователя часто чрезмерно упрощенна. В уроке мы подобрали пароль ответив на вопрос о любимом цвете.

- **Multi Level Login 2**

Многоуровневый вход должен обеспечить строгую аутентификацию. После вход в систему с именем пользователя и паролем, который попросили для "номер транзакции аутентификации" (TAN Transaction Authentication Number). Это часто используется онлайн-банкингом. Вы получите список с большим количеством TAN, генерируемых только для Вас банке. Каждый TAN используется только один раз. Другой метод,

чтобы обеспечить TAN по SMS. Это имеет то преимущество, что злоумышленник не может получить TAN, предоставленные пользователю.
В этом уроке необходимо войти в чужой аккаунт. У нас есть собственный аккаунт для Webgoat, но мы входим в другую учетную запись только зная имя пользователя жертвы для атаки.

- **Multi Level Login 1**

В этом уроке мы обошли строгую аутентификацию. Взломали другой счет. Имя пользователя, пароль и уже использованный TAN предоставляется. И мы использовали уже использованный TAN.

4. Переполнение буфера.

Состояние переполнения буфера существует, когда программа пытается поместить больше данных в буфер, чем он может вместить, или когда программа пытается поместить данные в область памяти после буфера. Внесение данных за пределы блока выделенной памяти может повредить данные, привести к краху программы или вызвать выполнение вредоносного кода.

- **Off-by-One Overflows**

5. Качество кода.

- **Discover Clues in the HTML**

В данном уроке в исходном коде искали данные, оставленные разработчиками. И зашли с помощью этих данных на сайт.

6. Многопоточность

Веб-приложения могут обрабатывать множество запросов HTTP одновременно. Разработчики часто используют переменные, которые не являются потокобезопасными. Безопасность потока означает, что поля объекта или класса всегда поддерживают правильное состояние, когда используются одновременно несколькими потоками. Часто можно использовать ошибка параллелизма при загрузке той же страницы другого пользователя в то же самое время.

- **Thread Safety Problems**

Используем ошибку параллелизма в веб-приложение и просматриваем регистрационную информацию другого пользователя, который пытается выполнить те же функции, в то же время.

- **Shopping Cart Concurrency Flaw**

В этом упражнении используем ошибку параллелизма в веб-приложение и приобретаем товар по более низкой цене.

7. Межсайтовое выполнение сценариев.

- **Phishing with XSS**

Пользователь имеет возможность добавить форму с просьбой ввести имя пользователя и пароль. На запрос предоставить вход отправляется:
http://localhost/WebGoat/catcher?PROPERTY=yes&user=catchedUserName&password=catchedPasswordName

- **LAB: Cross Site Scripting**

В этом упражнение сохранили и отразили XSS атак. Также осуществили изменения кода в веб-приложении, чтобы отразить эти нападения.

- **Reflected XSS Attacks**

В этом упражнение ввели в поле <script>alert('Bang!')</script>. Нажали кнопку и сценарий запустился.

- **Cross Site Request Forgery (CSRF)**

Cross-Site Request подлог (CSRF / XSRF) является атака, трюки жертву в загрузке страницы, содержащий ссылки IMG как показано ниже:
<IMG SRC = "http://www.mybank.com/sendFunds.do?acctId=123456" />
Когда браузер жертвы пытается исполнить эту страницу, будет выдаваться запрос на www.mybank.com на страницу transferFunds.do с заданными параметрами. Браузер открывает ссылку, чтобы получить изображение, хотя на самом деле это функция переводит средства. Таким образом, злоумышленник может заставить жертву выполнить действия, которые он не намерен был выполнить, например, выход из системы, покупки пункта, или любой другой функции, представленной уязвимым сайтом.
В данном уроке вставили HTML код в окно сообщения. Этот HTML-код должен содержать изображения тег и URL, которое является не реальным изображением, а начинает транзакцию на веб-сервере.

- **CSRF Prompt By-Pass**

В данном уроке отправили письмо, которое содержит несколько вредоносных запросов: первый перевести средства, а второй, подтвердить перевод, в итоге первый запрос срабатывает.

- **CSRF Token By-Pass**

Cross-Site Request Forgery (CSRF / XSRF) является атакой, приводящую жертву к загрузке страницы, содержащий 'поддельный запрос " для выполнения команд с учетными данными жертвы. Маркер проверки подлинности на основе запроса отпугивает эти нападения. Этот метод вставляет маркеры на страницы, которые выдают запросы. Эти маркеры должны завершить запрос, и помочь убедиться, что запросы не по скрипту. CSRFGuard от OWASP использует эту технику, чтобы помочь предотвратить CSRF атаку. Тем не менее, этот метод может быть обойден, если существует CSS уязвимости сайта.
В данном уроке отправили письмо, которое содержит вредоносный запрос на перевод средств. Для успешного завершения получили действительный маркер запроса.

- **HTTPOnly Test**

Чтобы помочь смягчить Cross Site Scripting угрозу, Microsoft представил новый атрибут cookie под названием «HttpOnly. Если этот флаг установлен, то браузер не должен позволить клиентскому сценарий доступ к cookie . Атрибут является относительно новым, и некоторые браузеры не обрабатывают новый атрибут должным образом.
В данном уроке проверили, поддерживает ли наш браузер флаг HttpOnly cookie. Когда браузер поддерживает HttpOnly, и мы включаем его для cookie, код на стороне клиента не должен быть в состоянии читать или писать cookie, но браузер все еще можете отправить это значение на сервер.

- **Cross Site Tracing (XST) Attacks**

Это хорошо - чистить весь ввод, особенно те входы, которые впоследствии будут использованы в качестве параметров команд ОС, сценариев и запросов к базе данных. Это особенно важно для контента, который будет постоянно хранится где-то в приложении. Люди не должны быть в состоянии создать сообщения, которые могут вызвать у другого пользователя загрузку страниц или нежелательно содержание, когда сообщение пользователя извлекается.

Tomcat настроен для поддержки команды HTTP TRACE. Выполнили Cross Site Tracing(XST) атаку.

8. Неправильная обработка ошибок

- **Fail Open Authentication Scheme**

В связи с проблемой обработки ошибок в механизме аутентификации, можно аутентифицировать как "Webgoat 'пользователя без ввода пароля. Попробовали войти как пользователь Webgoat без указания пароля. Проблема в том, что обработчик исключений в Java код выполняет блок catch для успешной аутентификации. Исключение возникает из-за NullPointer исключение при считывании параметра пароль.

10. Отказ в обслуживании

Отказ в обслуживании являются важным вопросом в веб-приложениях. Если конечный пользователь не может вести бизнес или выполнить услугу, предлагаемую веб-приложением, то время и деньги потрачены впустую.

- **Denial of Service from Multiple Logins**

В данном уроке сайт позволяет пользователю войти в систему несколько раз. Этот сайт имеет пул соединений с базой данных, что позволяет 2 соединения. Мы получаем список действительных пользователей и создаем в общей сложности 3 логина.

11. Небезопасное сетевое взаимодействие

Конфиденциальные данные никогда не должны быть отправлены в незашифрованном виде. Часто приложения переключаются на безопасное соединение после авторизации. Злоумышленник может просто узнать логин и использовать собранную информацию, чтобы ворваться в аккаунт. Необходимо, чтобы веб-приложения всегда заботились о шифровании конфиденциальных данных.

- **Insecure Login**

Этот урок состоит из двух этапов. На первом этапе мы узнаем пароль, который отправляется в незашифрованном виде. На втором этапе мы пытаемся сделать тоже самое, но при безопасном соединении. Видим, что весь трафик зашифрован.

12. Небезопасная конфигурация

Принудительный просмотр это метод, используемый злоумышленниками для получения доступа к ресурсам,на которые не ссылаются, но, тем не менее доступны. Один из методов это манипулировать URL в браузере, удалив разделы с конца, пока незащищенный каталог не найден.

- **Forced Browsing**

В данной работе необходимо было угадать URL для интерфейса "Config". Ссылка "Config" доступен только для обслуживающего персонала по ссылке “conf”.

13. Небезопасное хранилище

- **Encoding Basics**

В этом уроке рассматриваем различные схемы кодирования. Ввели строку "ABC". В списке ниже вы видим закодированное значение строки. Для rot13 кодирующих это "NOP". Теперь введите строку "a c" и видим, что кодирование URL удалось декодировать.

14. Исполнение злонамеренного кода

Форма ниже позволяет загрузить изображение, которое будет отображаться на этой странице. Такие возможности, часто встречаются на форумах и в социальных сетях. Эта функция является уязвимой для вредоносных исполняемых файлов.

- **Malicious File Execution**

В данном уроке создали вредоносный скрипт создающий новый файл.

15. Подделка параметров

Проверку на стороне клиента не следует рассматривать как безопасное средство для проверки параметров. Эти подтверждения необходимы только чтобы уменьшить количество времени обработки данных сервером для обычных пользователей, которые не знают, необходимый формат ввода. Злоумышленники могут легко обойти эти механизмы различными способами. Любая проверка на стороне клиента должна быть продублирована на стороне сервера.

- **Bypass HTML Field Restrictions**

Пользователь должен иметь возможность отправить на сайт входные данные, которые не ожидают.

В данном уроке, для того, чтобы сломать проверку на стороне клиента и отправить на сайт входные данные, которые не ожидают, необходимо было разорвать все 6 валидаторов в одно и то же время.

- **Exploit Hidden Fields**

Разработчики используют скрытые поля для отслеживания, логин, ценообразования и т.д .. информацию на загруженной странице. Хотя это удобный и легкий механизм для разработчика, они часто не проверяют информацию, полученную от скрытого поля. В данном уроке мы находим и изменяем скрытые поля, чтобы получить продукт по цене, отличной от указанной цены.

- **Exploit Unchecked Email**

Необходимо проверять все входные данные. Большинство сайтов позволяют не прошедшим проверку подлинности пользователям отправлять электронную почту "другу". Это большой механизм для спамеров - отправить электронную почту, используя свой корпоративный почтовый сервер.

- **Bypass Client Side JavaScript Validation**

Изменяем адрес получателя в ZAP.

Вывод:

В данной лабораторной работе мы ознакомились с такими инструментами как Zap, Mantra, Webgoat.

Специалисты безопасности и разработчики должны смотреть на сеть глазами потенциального взломщика, понимать суть атак и видеть проблемные места. Разнообразные руководства, которые легко найти в интернет могут дать лишь теоретические знания. Без их практического закрепления они не очевидны, легко забываются и будут очень поверхностными. Вот для такой ситуации разработчиками OWASP (Open Web Application Security Project) создана специальная обучающая система WebGoat, позволяющая в наглядном виде изучать приемы взлома веб-приложений. Реализована база для проведения около 30 различных видов атак. Причем основной упор сделан именно на образовательную сторону вопроса, а не

создание уязвимой платформы для опытов. В WebGoat реализованы все сопутствующие элементы – лекции, проверки знаний. По ходу обучения ведется статистика, показывающая результат на каждом этапе. Список курсов обширен и затрагивает базовые знания по HTML, контроль доступа, и различные виды атак – XSS, различные виды Injections, Buffer Overflow, работа со CSS и скрытыми полями в формах и так далее. По ходу обучения объясняется суть проблемы, даются все необходимые подсказки и код, этап завершается практической демонстрацией взлома с использованием уязвимости. Пройденная лекция подсвечивается зеленым флажком. Удобно, что в WebGoat уже есть все необходимое, то есть не нужно самостоятельно собирать тестовую среду, чтобы проверить все на практике.