**Руководство по выполнению лабораторной работы № 2 Анализ безопасности веб-приложений. Методы обход аутентификации.**

**Цель работы:**Получить навыки обхода аутентификации, познакомиться с базовыми уязвимостями веб-приложений и с их использованием.

**Ход работы:**

1) Выполнить 6 представленных заданий, часть заданий может быть не выполнена или выполнена не полностью.

2) Пояснить выполнение каждого задания при помощи изображений и описания произведенных действий.

3) Сформулировать выводы по работе и оформить отчёт.

**Краткая теория:**

**1) Анализ cookie-параметра.**

Cookie — небольшой фрагмент данных, отправленный веб-сервером и хранимый на компьютере пользователя. Веб-клиент (обычно веб-браузер) всякий раз при попытке открыть страницу соответствующего сайта пересылает этот фрагмент данных веб-серверу в составе HTTP-запроса. Применяется для сохранения данных на стороне пользователя, на практике обычно используется для:

* аутентификации пользователя;
* хранения персональных предпочтений и настроек пользователя;
* отслеживания состояния сеанса доступа пользователя;
* ведения статистики о пользователях.

Данный тип параметра можно изменять на стороне клиента, однако обрабатывается данный параметр на сервере, а не в браузере клиента, что может позволить произвести атаку на сервер или обход каких-либо ограничений при существующей уязвимости в данном параметре.

    Не редкость, когда проприетарные приложения обладают уязвимыми cookie-параметрами, хранящими данные об уровне привилегий вашей учётной записи, при использовании слабой криптографии или простого кодирования данных, это может повлечь за собой плохие последствия.

Пример слабо защищённых cookie-параметров представлен на Рис. 1.



Рисунок 1 - Пример запроса с параметром cookie.

**2) Особенности языка PHP при обработке данных.**

Известно, что язык PHP обладает довольно своеобразной функциональностью и логикой. Зачастую результаты обработки данных с помощью некоторых встроенных функций вызывает множество вопросов о корректности действий и логике работы языка. Однако несмотря на многие факторы против данного языка он до сих пор активно используется и развивается. Типичным примером не совсем верной логики языка является отсутствие в некоторых логических операции транзитивности Рис. 2.



Рисунок 2 - Пример необычного поведения кода PHP.

Также это демонстрирует и атаки типа Request Injection, сутью которых является изменение типа передаваемого значения, в надежде на отсутствие проверок типа передаваемых значений или использования уязвимых функций, при обработке введённых значений.

Например, передаются 2 параметра, при этом считается, что эти параметры обязательно будут являться строками. Данное упрощение может быть введено разработчиком, если эти параметры принимаются из стандартной формы ввода логина и пароля. Однако разработчик не учитывает, что тип данных, который к нему придёт может являться совсем не строкой, а, например, массивом.

Вот так передаются строки:

 url/index.php?login=user&paswd=paswd

Тот же скрипт, но пользователь решил передать не строки, а массивы

    url/index.php?login[]=test&passwd[]=test2

При отсутствии должных обработок типов и при использовании уязвимых функций, данный запрос может произвести обход аутентификации или вызвать сбой в работе приложения.

**Уязвимость Magic Hash.**

Из-за ошибки в PHP при работе с хешами в некоторых ситуациях злоумышленник может подобрать пароль к учетной записи пользователя, обойти аутентификацию и другие средства обеспечения безопасности, полагающиеся на хеширование.

Проблема возникает при использовании оператора равно-равно (==) во время сравнения двух хешей. PHP использует кодировку base16, в результате получаем строку, подобную этой: “0e812389…”.

Если строка начинается с «0e» и после нее следуют только цифры, PHP будет интерпретировать эту строку как число с плавающей точкой. В итоге, условие if (0e462097431906509019562988736854 == 0) будет истиной.

Если хеш пароля пользователя в базе данных начинается с 0e, злоумышленник может воспользоваться любой строкой, которая при сравнении будет интерпретирована как 0, и успешно авторизоваться в системе без знания пароля. То есть необходимо найти строку/число, хэш функция которого будет интерпретироваться как число с плавающей точкой.

**3) SQLi - обнаружение и базовые навыки эксплуатации.**

Внедрение SQL-кода (англ. SQL injection) — один из распространённых способов взлома сайтов и программ, работающих с базами данных, основанный на внедрении в запрос произвольного SQL-кода.

Внедрение SQL, в зависимости от типа используемой СУБД и условий внедрения, может дать возможность атакующему выполнить произвольный запрос к базе данных (например, прочитать содержимое любых таблиц, удалить, изменить или добавить данные), получить возможность чтения и/или записи локальных файлов и выполнения произвольных команд на атакуемом сервере.

Атака типа внедрения SQL может быть возможна из-за некорректной обработки входных данных, используемых в SQL-запросах.

Пример использования целочисленного и строкового параметра.

Допустим, серверное ПО, получив входной параметр id, использует его для создания SQL-запроса. Рассмотрим следующий PHP-скрипт:

$id = $\_REQUEST['id'];
$res = mysqli\_query("SELECT \* FROM news WHERE id\_news = " . $id);
Если на сервер передан параметр id, равный 5 (например так: *http://example.org/script.php?id=5*), то выполнится следующий SQL-запрос:

SELECT \* FROM news WHERE id\_news = 5
Но если злоумышленник передаст в качестве параметра id строку -1 OR 1=1 (например, так: *http://example.org/script.php?id=-1+OR+1=1*), то выполнится запрос:

SELECT \* FROM news WHERE id\_news = -1 OR 1=1
Таким образом, изменение входных параметров путём добавления в них конструкций языка SQL вызывает изменение в логике выполнения SQL-запроса (в данном примере вместо новости с заданным идентификатором будут выбраны все имеющиеся в базе новости, поскольку выражение 1=1 всегда истинно).

Предположим, серверное ПО, получив запрос на поиск данных в новостях параметром search\_text, использует его в следующем SQL-запросе (здесь параметры экранируются кавычками):

$search\_text = $\_REQUEST['search\_text'];
$res = mysqli\_query("SELECT id\_news, news\_date, news\_caption, news\_text, news\_id\_author
                      FROM news WHERE news\_caption LIKE('%$search\_text%')");
Сделав запрос вида *http://example.org/script.php?search\_text=Test* мы получим выполнение следующего SQL-запроса:

SELECT id\_news, news\_date, news\_caption, news\_text, news\_id\_author FROM news
  WHERE news\_caption LIKE('%Test%')
Но, внедрив в параметр search\_text символ кавычки (который используется в запросе), мы можем кардинально изменить поведение SQL-запроса. Например, передав в качестве параметра search\_text значение ')+and+(news\_id\_author='1, мы вызовем к выполнению запрос:

SELECT id\_news, news\_date, news\_caption, news\_text, news\_id\_author FROM news
  WHERE news\_caption LIKE('%') and (news\_id\_author='1%')

**Оформление отчёта:** При оформление отчёта о проделанной работе, необходимо привести изображения и описания выполненных заданий, а также сделать выводы о проделанной работе.