**Практика 3. Базовые уязвимости веб-приложений.**

**1. SQLi**

Уже не раз в данном курсе на лекциях затрагивалась данная тема. Однако, ей не посвящалось отдельного блока и подробного рассмотрения устройства данной атаке – не проводилось. В данной лекции данное упущение будет закрыто и SQLi будет рассмотрен достаточно подробно.

Для представления механизма работы данной атаки, нужно представлять, что такое база данных и скрипты, в которых производятся запросы к базе данных, а также знать хотя бы на примитивном уровне SQL и иметь навык написания базовых запросов.

Грубо говоря, обычная, в нашем понимании, БД состоит из множества таблиц. У каждой таблицы, естественно, есть столбцы и есть строки. Собственно, это ключевой момент. Возьмем к примеру таблицу пользователей какого-либо интернет-магазина. Для каждого пользователя должно быть описано несколько параметров (логин, адрес электронной почты, дата регистрации, сохранённая карточка и т.д.).

В итоге каждый столбец определяет какой-либо параметр пользователя, а каждая строка - конкретного пользователя. А в пересечении нужного нам столбца и строки будет информация о параметре нужного пользователя.

Для полного понимания можно привести **пример**:

*Представим, что вы (скрипт) идёте в магазин (базу данных), и просите в магазине (создаёте SQL запрос и отправляете его в базу данных) продавца: “Мне нужна одна бутылка воды за 25 рублей”.*

*Теперь переделаем это в SQL запрос к базе данных, получим следующее:*

SELECT \* FROM магазин WHERE (тип=’вода’ AND цена=25) LIMIT 1

Собственно, в ответ на вашу просьбу (SQL запрос) вы (скрипт) получаете бутылку (информацию), и продавца (Базу данных) уже не волнует, что вы будете с ней делать, так как свою работу он выполнил. Вы можете ее выпить, вылить, подарить (обработать, вывести, рассчитать) и тд.

­­Для полного понимания запроса в базу данных необходимо знать на базовом уровне язык SQL, однако даже без особых знаний можно заметить, что запрос выглядит понятным, так как используются довольно известные слова и они трактуются в данном случае однозначно. Вопросы может вызвать разве что окончание запроса: “LIMIT 1” – данная конструкция объявляет, что необходимо выдать только первый найденный результат, если вдруг их больше одного (бутылок по такой цене может быть очень много, но нам нужна только одна).

Все остальные мнемоники (команды) запроса предельно понятны:

SELECT – выбери (возьми)

FROM – из (указывает от куда нужно взять, чаще всего после данной команды следует название одной из таблиц в текущей базе данных)

WHERE – где (указывает какие значения различных параметров должны быть у той строки которую мы хотим получить, по сути указывает критерии для того товара который мы хотим получить)

Обобщенно атака типа SQL инъекция (SQL injection) возникает в случае если злоумышленник может каким-то образом модифицировать запрос к БД. Рассмотрим всё тот-же **пример** с магазином.

*Допустим вы решили пойти в магазин за кефиром, и специально написали на бумажке список продуктов, которые хотите купить, чтобы ничего не забыть. Предположим, что у вас дома оказался друг, который очень любит шоколад, однако ему нельзя есть его много и на его просьбу, вы ответили отказом и не стали записывать в листочек его пожелание. Итого вы получаете листочек, в котором на каждой новой строке записаны продукты, которые вам необходимы в магазине. Допустим, что каждый из продуктов будет представлять собой некий запрос к базе данных (магазину) как и в прошлом примере. Вы просто отдаёте продавцу свой листочек (скрипт передаёт запросы в базу данных) и он, читая очередную строку кладёт в ваш пакет тот или иной продукт (сохраняет результаты, того что удалось получить из базы данных). Теперь представим, что в данный листок ваш друг всё-таки умудрился вписать шоколад (провел атаку SQL injection) и продавец, читая очередную строчку может увидеть следующее: “Масло за 100 рублей или шоколад за 150 рублей”. В магазине может не оказаться масла, или продавцу долго за ним идти, и он положит вам в пакет шоколад. Продавец в данном случае не имеет представления о том, что вашему другу нельзя шоколад и вам он тоже особо не нужен, он просто выполняет свою работу, предоставляет продукты согласно переданному ему списку (база данных будет осуществлять работу по возврату данных на основание входящих запросов, если они были составлены правильно). Переведя это в SQL запрос получим следующее:*

SELECT \* FROM магазин WHERE (тип=’масло’ AND цена=’100’) OR (тип=‘шоколад’ AND цена=‘150’) LIMIT 1

Атака удалась потому что вы не проверили листочек перед тем как отдать его продавцу (скрипт не проверил входные данные перед формированием запроса и его отправки в базу данных).

SQLi может возникать в местах, где есть какие либо входящие параметры, например формы поиска товаров в интернет магазине или форма авторизации, которые есть на большей части всех веб-приложений. Любой не фильтруемый пользовательский ввод – почти всегда приводит к плачевным последствиям.

В случае с SQLi есть некоторые особенности, связанные с запросом, в который подставляются введённые пользователем данные. Например, тип значение, в которое поступают данные от пользователя может быть целочисленным или строковым, или стоять в нескольких скобочных вложениях, что немного затруднит написание зловредного запроса.

В случае с запросом на авторизацию, необходимо понимать, что как таковые данные с помощью такого запроса получить не удастся, так как такого рода запросы чаще всего рассчитываются на получение хэша от пользовательского пароля и сравнение его с высчитанным хэшом от входного пароля.

Большинство из SQLi обнаруживаются путём нарушения общего мнения о том, что должно быть введено в данное поле конечным пользователем, чаще всего срабатывает символ одинарной кавычки, так как многие из параметров – строковые. Подставив в какой-либо параметр кавычку и получив ошибку, содержащую слова “SQL syntax error” или что-то подобное, можно сразу сказать, что скорей всего здесь есть SQLi.

Однако, есть варианты что кавычка фильтруется, или отключен отчёт об ошибках, в этом случае нужно прибегать к другим методам и к сожалению, мы не будет рассматривать их в данном курсе, так как это довольно обширная тема и очень активно развивающаяся.

**2. LFI**

Local File Include (LFI) — уязвимость, которая позволяет удаленному пользователю получить доступ к нужной информации с помощью специально сформированного запроса к произвольным файлам. Грубо говоря, LFI представляет собой подключение любого файла на сервере к вызываемому файлу. Что делит LFI на две ветки: выполнение содержимого подключаемого файла и чтение содержимого подключаемого файла. Уязвимости класса LFI чаще всего встречается в различных менеджерах файлов.

**3. RCE**

**RCE** (англ. ***R****emote* ***C****ode* ***E****xecution*) – компьютерная уязвимость, при которой происходит удаленное выполнение кода на взламываемом компьютере, сервере и т.п. RCE является максимальной угрозой класса А1 по классификации OWASP. А также это гарантированный способ взлома сайтов и веб приложений. RCE - является одной из самых опасных уязвимостей.

Возможность удаленного внедрения кода в серверный скрипт в 100% случаев приводит к взлому ресурса. С помощью RCE злоумышленник сразу получает доступ к серверу атакуемого сайта, размещая на нем веб-шеллы, или любой другой вредоносный код.

Возможность эксплуатации RCE возникает из-за грубейших ошибок разработки сайта, отсутствия фильтрации передающих параметров, использование небезопасных функций и приемов программирования.

Подробнее про уязвимости смотрите в Лекции 4.

**Задание на практику:**

Решить 3 предоставленных задания, найдя в каждом контрольное значение и составить отчёт.