**Лекция 2. Безопасность веб-приложений. Основы.**

**Итак, для начала разберёмся в том, что из себя представляет Web.**

В первую очередь это какие-либо статические данные, файлы, CSS (*Cascading Style Sheets)* или таблица стилей, JS (JavaScript) – скриптовый язык программирования, исполняемый в браузере, картинки, HTML файлы…

Но статические данные нам не особо интересны (кроме каких-либо важных файлов, хранящихся на веб-сервере), так как уязвимостей, связанных с ними, обычно не бывает. Нас интересует для начала, на каком языке написано веб-приложение. Язык, на котором написано веб-приложение, может быть абсолютно любой, начиная от дефолтных PHP, Ruby, Python и заканчивая ASM, но, конечно, чаще всего используется какой-либо скриптовый язык.

После языка нам нужно понимать, на чём “крутится” веб-приложение – это сервер, который обрабатывает запросы пользователей и отдаёт им ответы на их запросы, чаще всего это Apache2 или nginx, но бывают и другие варианты, например, для встраиваемых систем часто используют маленькие, легковесные веб-сервера написанные на С/С++.

Далее идут базы данных. БД почти всегда используются более-менее серьёзным веб-приложением, в БД хранятся данные пользователей, настройки сервера, файлы, картинки и прочая очень интересная для взломщика информация. Для больших БД нужны удобные системы управления или СУБД, часто используемые варианты: MySQL, Oracle, SQLite.

И финальный элемент структуры – ОС, на которой всё это находится. Понятно, что она может быть любой, но чаще всего используется какой-нибудь Linux-дистрибутив (серверный).

**Что хочет злоумышленник?**

Изначально хакер хочет получить web-shell. Что такое web-shell?

Суть веб-шелла в том, что хакер каким-либо способом получает возможность выполнять код на языке, на котором написано приложение прямо на сайте, используя браузер. То есть он может вызывать определённые функции, которые есть в этом языке и фактически управлять сайтом через браузер.

Также хакера может интересовать БД веб-приложения, например, если это интернет-магазин, то в ней могут храниться данные банковских карт. Также в базе данных могут находиться логины и пароли (бывает и в открытом виде) пользователей, в том числе и для администратора сайта. Ну и само собой в БД могут находиться очень важные файлы, содержащие в себе какую-либо конфиденциальную информацию или тайну. Соответственно, целью хакера становится копирование (слив, дамп) базы данных к себе.

Следующий пункт, к которому стремится хакер — это получение шелла на сервере. Что даёт шелл? Шелл даёт возможность управлять непосредственно самим сервером, на котором находится веб-приложение, создавать директории, менять какие-либо правила, оставлять свои зловредные программы, например, для майнинга криптовалюты, сниффинга различных данных, добавление взломанной машины в ботнет. Или он может использовать эту машину для атаки на внутреннею сеть (если таковая имеется).

Финальная цель – это получение root-доступа к машине. Это означает, что хакер имеет полный контроль на сервере и может делать с ним абсолютно всё, вплоть до полного захвата машины.

**После того, как мы узнали, из чего состоит веб-приложение и определились с целями, которые хотим осуществить, стоит задуматься над тем, как мы их будем осуществлять и что мы будем использовать при этом.**

Перейдём к уязвимостям, которые помогут нам в осуществлении наших целей. С помощью грамотной эксплуатации уязвимостей мы сможем реализовать всё, что задумывали раньше.

Основной набор уязвимостей и атак описывается в ежегодном отчёте компании OWASP, который называется OWASP TOP 10. Данный топ, представляет собой список самых часто встречающихся в этом году уязвимостей.

В данном топе описываются самые часто встречающиеся за этот год уязвимости и атаки, найденные или проведённые в веб-приложениях. Почти всегда на 1 месте стоят – SQLi (SQL injection) – внедрения SQL кода.

**Как выглядит на практике внедрение SQL кода?**

Допустим, мы имеем определённый запрос к базе данных, в котором в качестве одного из параметров участвуют данные, которые были введены пользователем на веб-сайте. Если эти данные участвуют в составлении запроса к базе данных, то мы можем непосредственно влиять на этот запрос, например, сначала мы запросим товар с идентификационным номером 1, далее с номером 2, если такие номера есть, то мы получим по ним данные, какое-нибудь описание, цену и прочее. Так упрощённо работают запросы к базе данных. Что произойдёт, если мы захотим нарушить логику запроса и запросим не номер товара, а передадим в этих данных какие-либо ключевые слова языка запроса?

Например, у нас есть база данных и в ней есть таблица товаров, в которой содержится информация по товарам и к которой поступают запросы от веб-приложения. Допустим, что на сайте есть опция поиска товара по номеру, тогда для данной опции может быть составлен такой запрос:

SELECT \* FROM goods WHERE id = *userinput*,

где *userinput* – это то, что ввёл пользователь. Представим, что никакой обработки не происходит и то, что ввёл пользователь, никак не проверяется и вставляется в запрос как есть. Что тогда произойдёт? Если на месте пользователя окажется злоумышленник или специалист по уязвимостям веб-приложений, то он сможет с помощью ключевых слов языка SQL управлять запросом и доставать из различных таблиц необходимые данные, например, логин и хэш пароля администратора, номера банковских карточек, электронные почты пользователей и прочее, что чаще всего хранится в базе данных. Вот так приблизительно и работает уязвимость, связанная с инъекцией SQL-кода. Почти все уязвимости связаны с неправильной обработкой входных данных, получаемых от пользователя.

**Теперь рассмотрим стандартный план действий при атаке на веб-приложение.**

Итак, на первом этапе анализа веб-приложения происходит сбор информации о данном веб-приложении (сайте). Он бывает двух видов: пассивный и активный. Суть их в том, что при активном режиме вы уже фактически вмешиваетесь в работу веб-приложения, и если у вас нет на это разрешения, то это является нарушением законодательства, по крайне мере в РФ. Активный сбор информация – это, например, сканирование активных портов, которое вы проводите. Иногда активные порты можно посмотреть через различные сервисы наподобие Shodan или онлайн сканеров. Пассивный сбор информации включает в себя сбор информации о веб-приложении, который никак не касается его работоспособности и связан только с общими данными, которые получаются без использования специальных программ и не взаимодействуют с веб-приложением напрямую. Например, с помощью обычного просмотра сайта вы можете скачать какие-нибудь документы и узнать из них электронные почты, по которым потом можно проводить атаку полного перебора на форму авторизации, если она присутствует на сайте. Вариантов много.

После сбора информации обычно идёт поиск уязвимостей. Фактически самая важная часть исследования и одна из самых сложных. Здесь нет как такового верного подхода, у каждого исследователя свой подход к поиску уязвимостей: кто-то использует сканеры, фаззеры и прочие утилиты, кто-то ищет вручную, используя только базовые инструменты по типу Burp Suite или только браузер, но это совсем редко встречается. Поиск уязвимостей особенно в BlackBox’e чаще всего представляет из себя создание разнообразных предположений и гипотез и последующую проверку их. Есть также так называемые “известные” уязвимости – это уязвимости, которые уже известны сообществу и могут быть обнаружены, если веб-приложение не обновляло ту часть, где находится эта уязвимость, как раз для такого рода уязвимостей и существуют сканеры.

После обнаружения уязвимости или ряда уязвимостей, разрабатывается вектор атаки на приложение. **Что это значит?**

Это значит, что, исходя из найденных уязвимостей и их типа, необходимо разработать план нападения на приложение, который должен учитывать, что и как будет проводиться и какие финальные цели, обычно это очень подробно описывается в отчётах исследователей. У злоумышленников вектор атаки зависит от целей, исследователям же приходится расписывать все возможные варианты атаки злоумышленника и последствия, которые будут после удачной или неудачной атаки. Также есть обще описательные вектора уязвимостей, которые содержат в себе тип уязвимости, доступ необходимый для её эксплуатации и критичность.

Далее идёт эксплуатация, это считается самой сложной частью, так как правильно поэксплуатированная одна небольшая, некритичная уязвимость может привести к катастрофическим последствиям.

Под пост-эксплуатацией подразумевается заметание следов и оставление себе удобного доступа к машине, либо к полученным функциям.

Вот приблизительно так выглядит карта атаки на веб-приложение.

Далее в презентации идут примеры первого этапа – сбора информации.