Занятие «Основы технологий, необходимые для понимания уязвимостей. Классификация OWASP top 10»

Цели занятия
разобрать понятие "уязвимость" и классификацию уязвимостей.
Краткое содержание
разберем:
- какие уязвимости бывают;
- классификацию веб-уязвимостей;
- основные подходы к поиску уязвимостей;
- OWASP Top 10.

Лекция

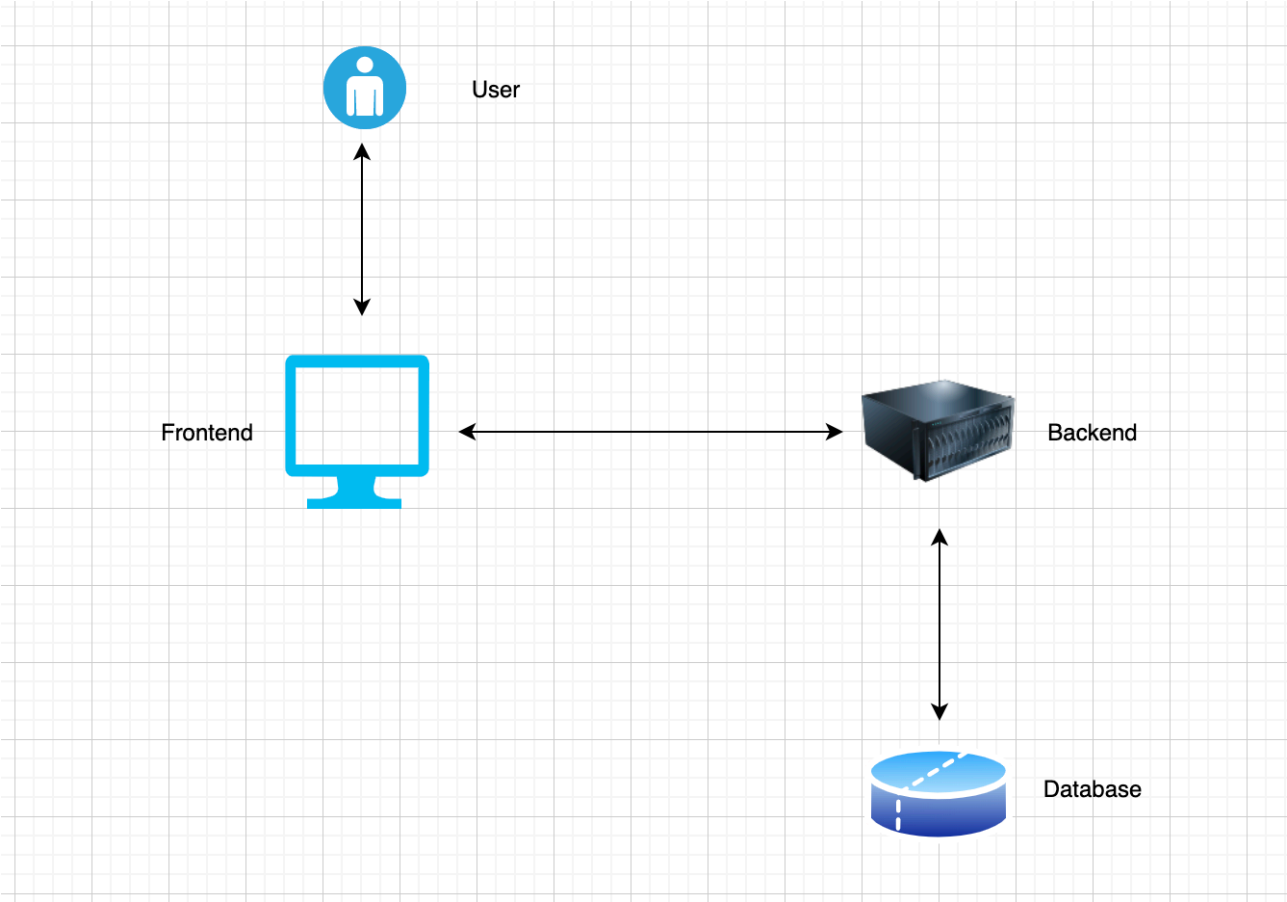
Что такое уязвимость?

Уязвимость - свойство системы (в нашем случае веб-сайтов, серверов, баз данных), используя которое можно нанести данной системе (и не только) ущерб, например, нарушить целостность, вызвать неправильную работу, атаковать пользователей и тд

Какие уязвимости бывают?

Рассмотреть архитектуру современных приложений, обозначить, какие могут быть уязвимости на фронте, бэке, к базе.

Самая популярная архитектура веб-приложения выглядит сейчас примерно так:



На ней мы видим несколько элементов. Это backend - в нем осуществлена вся логика приложения (это обработка пользовательских параметров, выполнение каких-то вычислений, работа с базой и так далее)
Database - база, хранящая в себе данные, и какие-нибудь процедуры.
Front - часть приложения, отвечающее за отображение пользовательского интерфейса.

Соответственно, с каждым из этих элементов могут быть свои проблемы.

Например, уязвимости во **фронте** позволяют атаковать пользователей, открывающих страницы (например, та же атака XSS, которая появляется в результате некорректной обработки пользовательских данных позволяет украсть пользовательскую сессию и заходить от лица пользователя)

На самом деле их, наверное, некорректно называть уязвимостями фронта. Потому что они обусловлены некорректной работой функций именно на бэке. Дело фронта - просто отображать и все.
Но с другой стороны, проявляются атаки именно на фронте. Поэтому их называют атаками на пользователя.

Уязвимости на **бэке** - это те уязвимости, которые позволяют обойти логику приложения, вызвать отказ в обслуживании, получить неправомерный доступ к файлам или самой системе, базе данных и так далее. То есть более серьёзный ущерб веб-приложению. Их называют серверными уязвимостями.

Уязвимости **Базы данных** отдельно обычно не выделяются в контексте веб приложений. Потому что тут надо рассматривать именно взаимодействие бэка с базой. Именно в этом взаимодействии и могут быть какие-то проблемы. Например, слишком частые тяжёлые запросы, которые могут положить базу, или недостаточная санитизация, которая приводит к инъекциям.

Причины возникновения уязвимостей?

Недостаточная проверка пользовательского ввода
Первая причина, конечно же, кроется в том, что разработчик обычно не рассматривает все варианты пользовательского ввода. Например, при передаче айди, который представляет из себя число, он забыл поставить проверку на инт. Вариаций таких уязвимостей очень много. Они могут встретиться где угодно, где есть пользовательский ввод.
Но даже если разработчик предусмотрел обработку, есть так же вероятность, что она может работать некорректно. Это тоже нужно учитывать.

Логика приложения
Тут появляется неправильное разграничение прав пользователей, неправильный контроль доступа, отказ в обслуживании, и так далее

OWASP TOP 10
Что такое овасп?
OWASP - это некоммерческая организация, занимающаяся исследованием безопасности веб приложений.
Сокращение произошло от …

Топ 10 - это рейтинг, в которых входят уязвимости в зависимости от их встречаемости и импакта.

Рассмотрим каждый пункт

Injection
Сюда входят все типы инъекций:
- SQLi
- OS
- Ldap
- NoSQLi

Собственно, что такое инъекция? Это внедрение постороннего кода в логическое выражение, которое затем выполняется операционной системой или базой данных. При этом это внедрение может полностью менять логику выражения, предоставляя злоумышленнику большие возможности.
Понимаю, что возможно просто так со слов не понятно, поэтому давайте я покажу пример. Рассмотрим SQLi. Предположим, у нас есть сайт с товарами. У него на одной из страниц через get параметр передается id для извлечения товара из базы. Выглядит это примерно следующим образом (псевдоязык):
Item = mysqli_query('SELECT * FROM Items WHERE id=' + $_GET['id'])

Подаем id = 1, получаем выражение:
SELECT * FROM Items WHERE id=1

Вроде все ок. Давайте попробуем сделать инъекцию.
Подаем id = -1 or 1=1, получаем выражение:
SELECT * FROM Items WHERE id=-1 or 1=1

Что происходит в этом случае? Вместо одного какого-то предмета мы достанем все. Все дело в том, что подав в параметре id значение, которое разработчик не предусмотрел, мы можем изменить сам SQL запрос к базе данных.

С помощью инъекций можно выполнять код на стороне сервера, получать данные из баз, обходить авторизацию и так далее.

Broken Authentication
Включает в себя:
1) уязвимости JWT

2) Уязвимости сессий
3) Возможность перебора паролей
4) И так далее

В общем заключается именно в слабости процесса аутентификации пользователей.
Прикладные функции, связанные с аутентификацией и управлением сеансами, часто реализуются неправильно, что позволяет злоумышленникам компрометировать пароли, ключи или токены сеансов или использовать другие недостатки реализации для временного или постоянного присвоения сессий других пользователей.

И тут у меня к вам вопрос: чем отличается аутентификация и авторизация?

Sensitive data exposure
Многие веб-приложения и API не обеспечивают должной защиты конфиденциальных данных, таких как финансовые, медицинские и тд. Злоумышленники могут украсть или изменить такие слабо защищенные данные для совершения мошенничества с кредитными картами, кражи личных данных или других преступлений. Конфиденциальные данные могут быть скомпрометированы без дополнительной защиты, такой как шифрование.

Тут у меня вопрос, как хранить пароли?

XML External Entities (XXE)
Многие старые или плохо настроенные XML-обработчики работают с внешними ссылками на сущности в XML-документах. Внешние сущности могут использоваться для раскрытия внутренних файлов с помощью обработчика URI файла, внутренних общих файловых ресурсов, сканирования внутренних портов, удаленного выполнения кода и атак типа "отказ в обслуживании".

Broken Access Control
Ограничения на то, что разрешено делать аутентифицированным пользователям, часто не соблюдаются должным образом. Злоумышленники могут использовать эти недостатки для доступа к несанкционированным функциям и / или данным, таким как доступ к

учетным записям других пользователей, просмотр конфиденциальных файлов, изменение данных других пользователей, изменение прав доступа и т. д.

Очень популярный кейс, когда пользователь изменяет свои данные в лк, на сервер отправляется запрос:
{id : user_id, name : name_str}

При изменении id можно изменить имя другого пользователя не получая доступ к его сессии.

Security Misconfiguration

Неправильное конфигурирование безопасности является наиболее распространенной проблемой. Это обычно является результатом небезопасных конфигураций по умолчанию, неполных или специальных конфигураций, открытого облачного хранилища, неправильно настроенных заголовков HTTP и подробных сообщений об ошибках, содержащих конфиденциальную информацию. Все операционные системы, фреймворки, библиотеки и приложения должны быть не только надежно сконфигурированы, но и своевременно исправлены/обновлены.

Например, то, что веб приложение возвращает версию сервера является не совсем корректным исходом, потому что данная информация может использоваться злоумышленниками для сбора дополнительной информации о веб-приложении.
Отсутствие заголовков HSTS и так далее.

Тут у меня вопрос: кстати, что означает заголовок HSTS?

Cross-Site Scripting XSS

Атаки XSS возникают всякий раз, когда приложение включает ненадежные данные в новую веб-страницу без надлежащей проверки или экранирования, или обновляет существующую веб-страницу с помощью предоставленных Пользователем данных с помощью API браузера, который может создавать HTML или JavaScript. XSS позволяет злоумышленникам выполнять сценарии в браузере жертвы, которые могут перехватывать сеансы пользователя, портить веб-сайты или перенаправлять пользователя на вредоносные сайты.

Есть различные типы атаки XSS, которые мы рассмотрим на другом занятии

Insecure Deserialization
Небезопасная десериализация часто приводит к удаленному выполнению кода. Даже если недостатки десериализации не приводят к удаленному выполнению кода, их можно использовать для выполнения атак, включая атаки воспроизведения, инъекционные атаки и атаки эскалации привилегий.

Using Components with Known Vulnerabilities
Компоненты, такие как библиотеки, фреймворки и другие программные модули, работают с теми же правами, что и приложение. Если используется уязвимый компонент, такая атака может привести к серьезной потере данных или захвату сервера. Приложения и API, использующие компоненты с известными уязвимостями, могут подорвать защиту приложений и привести к различным атакам и воздействиям

Insufficient Logging & Monitoring
Недостаточное ведение журнала и мониторинг в сочетании с отсутствием или неэффективной интеграцией с реагированием на инциденты позволяют злоумышленникам продолжать атаковать системы, поддерживать постоянство, переходить к большему количеству систем и подделывать, извлекать или уничтожать данные. Большинство исследований нарушений показывают, что время обнаружения нарушения составляет более 200 дней, обычно обнаруживаемых внешними сторонами, а не внутренними процессами или мониторингом.

Основные подходы к поиску уязвимостей

Метод черного ящика:

Метод черного ящика подразумевает поиск уязвимостей в рабочем приложении без исходного кода. То работа в первую очередь начинается с понимания логики приложения, поиска эндоинтов и так далее. Этот этап принято называть разведкой.

В первую очередь надо понимать, какая уязвимость тут может быть. То есть надо хотя бы минимально знать существующие уязвимости (их признаки, последствия, чем обусловлены) и понимать логику тестируемого приложения.
Знать способы детекта той или иной уязвимости. Это поможет при сборе словарей, подготовке к атаке.
Что я здесь имею в виду. Например, вы пытаетесь найти sqli. Подставляете запрос в параметр и получаете ошибку базы данных. Например, что у вас неправильное количество кавычек. Вот это сообщение и будет являться маркером того, что тут возможно SQLi.

Найти все эндпоинты, куда можно передавать информацию. Либо эндпоинты, отвечающие за какие-то тяжелые операции. В первом случае понятно, тут как раз то, о чем мы все время говорили - посылаем на эндпоинт данные, не предусмотренные разработчиком. А что же во втором случае? Как раз в этом случае можно инициировать DOS, если будем часто дергать ручку, отвечающую за какую-нибудь нагружённую операцию.

Какие инструменты могут тут помочь.
Dirsearch
Gobuster
Dirbuster
Burp Suit

Пофаззить найденные эндпоинты. Иногда бывает лень руками вводить несколько значений. Или же бывает так, что стоит ваф, который отсеивает часть запросов. Или же разработчик написал какую-то свое санитизацию. В этом случае целесообразно не вводить руками тысячу всевозможных пейлоадов. Для этого есть специальные инструменты, которые называются фаззерами. Даете на вход урл с параметром и словарь, он сам отправляет туда пейлоад.
Нужно обратить внимание на то, что фаззить нужно не только get и post параметры, но и хэдеры.

webfuzz

Метод белого ящика:
В данном у вас есть код и вы можете анализировать нужные функции в их исходном виде.
Тут есть свои плюсы и минусы. Во-первых, вы сразу видите, как все работает изнутри. В таком случае гораздо легче придумать обход фильтров или понять, что обходить их бесполезно. Однако с другой стороны осложняется тем, что по коду иногда очень сложно отследить какие компоненты с чем взаимодействуют. Так легко пропустить ошибку разграничения доступа и так далее.

Поэтому часто вступает в дело метод **серого ящика**. Это когда веб приложение оценивается в рабочем режиме - то есть смотрится сайт, эндоинты и все все все, что было описано про черный ящик. В то же время есть доступ к исходному коду, чтобы была возможность валидировать ту или иную уязвимость. Потому что бывают моменты, когда из черного ящика не совсем понятно, есть ли тут уязвимость или нет. И чтобы не тратить время, можно заглянуть в исходный код.