Тестирование безопасности – Cookies

Что такое cookies?

Файл cookies – это небольшой фрагмент информации, отправляемый веб-сервером для хранения в веб-браузере, чтобы впоследствии он мог быть прочитан браузером. Таким образом, браузер запоминает определенную личную информацию. Если хакер получает информацию о cookies-файлах, это может привести к проблемам с безопасностью.

Свойства cookies

Вот некоторые важные свойства cookies –

- Обычно это небольшие текстовые файлы с заданными идентификаторами, которые хранятся в каталоге браузера вашего компьютера.
- Они используются веб-разработчиками, чтобы помочь пользователям эффективно перемещаться по своим веб-сайтам и выполнять определенные функции.
- Когда пользователь снова просматривает тот же веб-сайт, данные, хранящиеся в cookie, отправляются обратно на веб-сервер, чтобы уведомить веб-сайт о предыдущих действиях пользователя.
- cookie -файлы неизбежны для веб-сайтов, которые имеют огромные базы данных, нуждаются в логинах, имеют настраиваемые темы.

Содержание cookie

Файл cookie содержит следующую информацию:

- Имя сервера, с которого был отправлен файл cookie.
- Время жизни cookie.
- Значение – обычно случайно сгенерированный уникальный номер.

Типы cookie

- **Сеансовые файлы cookie** – это временные файлы cookie, которые стираются, когда пользователь закрывает браузер. Даже если пользователь снова входит в систему, для этого сеанса создается новый файл cookie.
- **Постоянные файлы cookie** – эти файлы cookie остаются на жестком диске, если пользователь не удалит их или не истечет срок их действия. Срок действия Cookie зависит от того, как долго они могут длиться.

Практическое задание

1. Проведите анализ предложенных информационных материалов.
2. Заполните таблицу – какие два типа файлов **cookie** существуют и в чем между ними различие.

Таблица

Тип файлов 1	Тип файлов 2

Тестирование Cookies

Вот способы проверить Cookies –

- **Отключение Cookies -файлов.** В качестве тестера нам необходимо проверить доступ к веб-сайту после отключения Cookies -файлов и проверить, правильно ли работают страницы. Навигация по всем страницам сайта и отслеживание сбоев приложений. Также необходимо информировать пользователя о том, что файлы cookie необходимы для использования сайта.
- **Коррупция файлов cookie.** Еще одно тестирование, которое необходимо выполнить, – это испортить файлы cookie. Чтобы сделать то же самое, нужно найти местоположение файла cookie сайта и вручную отредактировать его с помощью поддельных / недействительных данных, которые можно использовать для доступа к внутренней информации из домена, которая, в свою очередь, затем может быть использована для взлома сайта.
- **Удаление файлов cookie** – удалите все файлы cookie для веб-сайта и проверьте, как веб-сайт реагирует на него.
- **Кросс-браузерная совместимость.** Также важно проверить, правильно ли записываются файлы cookie во всех поддерживаемых браузерах с любой страницы, на которой записываются файлы cookie.
- **Редактирование файлов cookie.** Если приложение использует файлы cookie для хранения информации для входа в систему, в качестве тестера мы должны попытаться изменить пользователя в файле cookie или адресной строке на другого действительного пользователя. Редактирование **cookie** не должно позволять вам войти в другую учетную запись пользователя.

Тестирование межсайтовых сценариев

Межсайтовый скриптинг (XSS) происходит всякий раз, когда приложение берет ненадежные данные и отправляет их клиенту (браузеру) без проверки. Это позволяет злоумышленникам выполнять вредоносные сценарии в браузере жертвы, что может привести к перехвату

пользовательских сеансов, порче веб-сайтов или перенаправлению пользователя на вредоносные сайты.

Типы XSS

- **Сохраненный XSS** – сохраненный XSS, также известный как постоянный XSS, возникает, когда пользовательский ввод хранится на целевом сервере, таком как база данных / форум сообщений / поле комментариев и т. Д. Затем жертва может извлечь сохраненные данные из веб-приложения.
- **Отраженный XSS** – отраженный XSS, также известный как непостоянный XSS, возникает, когда пользовательский ввод немедленно возвращается веб-приложением в сообщении об ошибке / результате поиска или вводом, предоставленным пользователем как часть запроса, и без постоянного сохранения данных, предоставленных пользователем. ,
- **Основанный на DOM XSS** – Основанный на DOM XSS является формой XSS, когда источник данных находится в DOM, приемник также находится в DOM, и поток данных никогда не покидает браузер.

Профилактические механизмы

- Разработчики должны убедиться, что они избегают всех ненадежных данных на основе контекста HTML, такого как тело, атрибут, JavaScript, CSS или URL-адрес, в который помещаются данные.
- Для тех приложений, которым в качестве входных данных требуются специальные символы, должны существовать надежные механизмы проверки, прежде чем принимать их в качестве допустимых входных данных.

Практическое задание

1. Проведите анализ предложенных информационных материалов.
2. Заполните таблицу – типы межсайтовых скриптингов и их характерные особенности.

Таблица

Небезопасные прямые ссылки на объекты

Прямая ссылка на объект может возникать, когда разработчик предоставляет ссылку на внутренний объект реализации, такой как файл, каталог или ключ базы данных, без какого-либо механизма проверки, который позволяет злоумышленникам манипулировать этими ссылками для доступа к неавторизованным данным.

Профилактические механизмы

Разработчики могут использовать следующие ресурсы / точки в качестве руководства, чтобы предотвратить небезопасную прямую ссылку на объект во время самой фазы разработки.

- Разработчики должны использовать только одного пользователя или сеанс для косвенных ссылок на объекты.
- Также рекомендуется проверять доступ перед использованием прямой ссылки на объект из ненадежного источника.

Неправильная настройка безопасности

Неверная конфигурация безопасности возникает, когда параметры безопасности определены, реализованы и поддерживаются как значения по умолчанию. Хорошая безопасность требует безопасной конфигурации, определенной и развернутой для приложения, веб-сервера, сервера базы данных и платформы. Не менее важно иметь программное обеспечение в актуальном состоянии.

Пример

Вот некоторые классические примеры неправильной настройки безопасности:

- Если список каталогов не отключен на сервере и если злоумышленник обнаружит его, злоумышленник может просто перечислить каталоги, чтобы найти любой файл и выполнить его. Также возможно получить реальную базу кода, которая содержит весь ваш пользовательский код, а затем найти серьезные недостатки в приложении.
- Конфигурация сервера приложений позволяет пользователям возвращать следы стека, что потенциально может выявить недостатки. Злоумышленники собирают ту дополнительную информацию, которую предоставляют сообщения об ошибках, которой достаточно для проникновения.
- Серверы приложений обычно поставляются с примерами приложений, которые недостаточно защищены. Если его не удалить с рабочего сервера, это может привести к компрометации вашего сервера.

Профилактические механизмы

- Все среды, такие как среды разработки, контроля качества и производственные среды, должны быть настроены одинаково с использованием разных паролей, используемых в каждой среде, которые нелегко взломать.

- Убедитесь, что принята мощная архитектура приложения, которая обеспечивает эффективное и безопасное разделение между компонентами.
- Это также может минимизировать вероятность этой атаки, запуская автоматическое сканирование и периодически проводя аудит.

Практическое задание

1. Проведите анализ предложенных информационных материалов.
2. Заполните таблицу – какие профилактические механизмы существуют в случае небезопасных прямых ссылок на объект и в случае неправильной настройки безопасности.

Таблица

Небезопасные прямые ссылки на объекты	**Неправильная настройка безопасности**

Тестирование безопасности – раскрытие конфиденциальных данных

Поскольку онлайн-приложения ежедневно заполняют Интернет, не все приложения защищены. Многие веб-приложения должным образом не защищают конфиденциальные данные пользователя, такие как данные кредитных карт / данные банковского счета / учетные данные для аутентификации. Хакеры могут в итоге украсть эти слабо защищенные данные для мошенничества с кредитными картами, кражи личных данных или других преступлений.

Пример

Вот некоторые из классических примеров неправильной настройки безопасности:

- Сайт просто не использует SSL для всех аутентифицированных страниц. Это позволяет злоумышленнику отслеживать сетевой трафик и похищать файл cookie сеанса пользователя, чтобы захватить сеанс пользователя или получить доступ к его личным данным.
- Приложение хранит номера кредитных карт в зашифрованном виде в базе данных. После извлечения они расшифровываются, позволяя хакеру выполнить атаку SQL-инъекции, чтобы получить всю важную информацию в виде открытого текста. Этого можно избежать, зашифровав номера кредитных карт с помощью открытого ключа и разрешив фоновым приложениям расшифровать их с помощью закрытого ключа.

Профилактические механизмы

- Рекомендуется не хранить конфиденциальные данные без необходимости и должны быть очищены как можно скорее, если это больше не требуется.
- Важно убедиться в том, что мы используем надежные и стандартные алгоритмы шифрования и используем правильное управление ключами.
- Этого также можно избежать, отключив автозаполнение в формах, собирающих конфиденциальные данные, такие как пароль, и отключив кэширование для страниц, содержащих конфиденциальные данные.

Отсутствует контроль доступа на уровне функций

Большинство веб-приложений проверяют права доступа на уровне функций, прежде чем сделать эту функцию доступной для пользователя. Однако, если те же проверки контроля доступа не выполняются на сервере, хакеры могут проникнуть в приложение без надлежащей авторизации.

Профилактические механизмы

- Механизм аутентификации должен запрещать любой доступ по умолчанию и предоставлять доступ к определенным ролям для каждой функции.
- В приложении на основе рабочего процесса проверьте состояние пользователей, прежде чем разрешить им доступ к любым ресурсам.

Практическое задание

1. Проведите анализ предложенных информационных материалов.
2. Сформулируйте характерные особенности раскрытия конфиденциальных данных и отсутствия контроля доступа на уровне функций.

Таблица

Раскрытие конфиденциальных данных	Отсутствует контроль доступа на уровне функций

Непроверенные перенаправления и пересылки

Большинство веб-приложений в Интернете часто перенаправляют пользователей на другие страницы или другие внешние веб-сайты. Однако, не проверяя достоверность этих страниц, хакеры могут перенаправлять жертв на фишинговые или вредоносные сайты или использовать пересылки для доступа к несанкционированным страницам.

Пример

Вот некоторые классические примеры непроверенных перенаправлений и переадресаций:

- Допустим, у приложения есть страница – redirect.jsp, которая принимает параметр *redirectrul* . Хакер добавляет вредоносный URL, который перенаправляет пользователей, которые выполняют фишинг / устанавливают вредоносное ПО.
    - Все веб-приложения используются для перенаправления пользователей в разные части сайта. Чтобы добиться того же, некоторые страницы используют параметр, чтобы указать, куда следует перенаправить пользователя в случае успешной операции. Злоумышленник создает URL-адрес, который проходит проверку контроля доступа приложения, а затем перенаправляет злоумышленника на административные функции, к которым у злоумышленника нет доступа.

Профилактические механизмы

- Лучше избегать использования перенаправлений и форвардов.
- Если это неизбежно, то это должно быть сделано без привлечения пользовательских параметров при перенаправлении адресата.

Тестирование безопасности – отказ в обслуживании

Атака отказа в обслуживании (DoS) – это попытка хакеров сделать сетевой ресурс недоступным. Обычно он прерывает хост, временный или неопределенный, который подключен к Интернету. Эти атаки обычно нацелены на службы, размещенные на критически важных веб-серверах, таких как банки, шлюзы оплаты кредитными картами.

Симптомы DoS

- Необычно низкая производительность сети.
- Недоступность определенного веб-сайта.
- Невозможность получить доступ к любому веб-сайту.
- Резкое увеличение количества полученных спам-писем.
- Долгосрочный отказ в доступе к сети или любым интернет-сервисам.
- Недоступность конкретного сайта.

Профилактические механизмы

- Выполните тщательную проверку ввода.
- Избегайте операций с высокой загрузкой процессора.
- Лучше отделить диски с данными от системных дисков.

Тестирование безопасности – Инструменты автоматизации

Для тестирования безопасности приложения доступны различные инструменты. Существует несколько инструментов, которые могут выполнять

сквозное тестирование безопасности, в то время как некоторые из них предназначены для выявления определенного типа недостатка в системе.

Инструменты с открытым исходным кодом

Некоторые инструменты тестирования безопасности с открытым исходным кодом, как дано –

S.No.	Имя инструмента
1	**Zed Attack Proxy** Предоставляет автоматические сканеры и другие инструменты для выявления недостатков безопасности. https://www.owasp.org
2	**Ports Wigger** Инструмент для перехвата и Modyfying трафика и работает с работой с пользовательскими сертификатами SSL. https://www.portswigger.net/Burp/
4	**Firefox Tamper Data** Используйте tamperdata для просмотра и изменения заголовков HTTP / HTTPS и параметров публикации https://addons.mozilla.org/en-US
5	**Инструменты веб-разработчика Firefox** Расширение Web Developer добавляет различные инструменты для веб-разработчиков в браузер. https://addons.mozilla.org/en-US/firefox

Специальные наборы инструментов

Следующие инструменты могут помочь нам определить определенный тип уязвимости в системе:

S.No.	Ссылка на сайт
1	**DOMinator Pro – Тестирование для DOM XSS** https://dominator.mindedsecurity.com/
2	**Sqlninja – SQL-инъекция** http://sqlninja.sourceforge.net/
3	**SQLInjector – SQL-инъекция** https://sourceforge.net/projects/safe3si/
4	**Ncat – пароль грубой силы** https://nmap.org/ncat/
5	**OllyDbg – тестирование переполнения буфера** http://www.ollydbg.de/

6	**Spike – тестирование переполнения буфера** https://www.immunitysec.com/downloads/SPIKE2.9.tgz
7	**Metasploit – тестирование переполнения буфера** https://www.metasploit.com/

Бесплатные анализаторы исходного кода

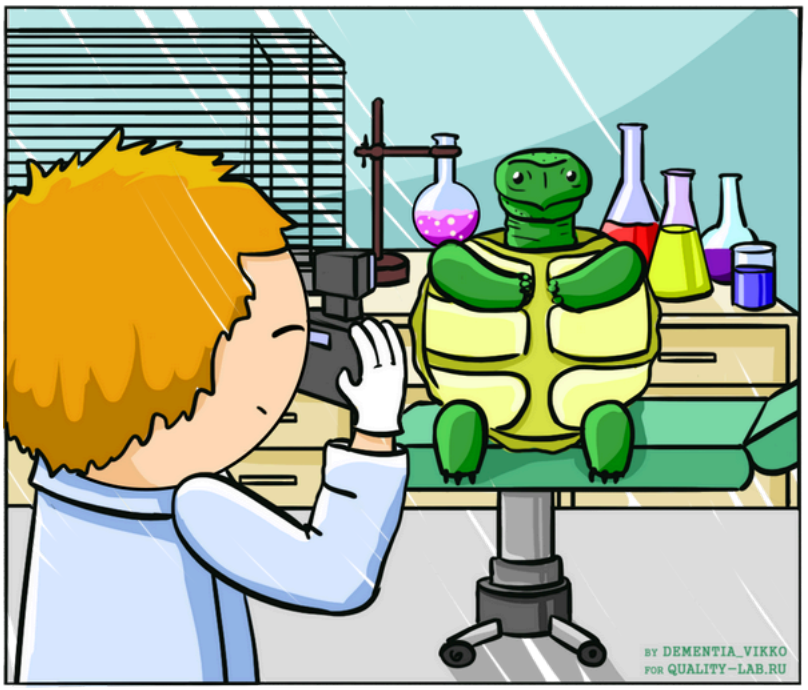
S.No	**Инструмент**
1	**Splint** http://splint.org/
2	**W3af** http://w3af.org/
3	**Flawfinder** https://www.dwheeler.com/flawfinder/
4	**FindBugs** http://findbugs.sourceforge.net/

Практическое задание

1. Проведите анализ предложенных информационных материалов.
2. Сформулируйте алгоритм – каким образом злоумышленники пользуются непроверенными перенаправлениями и переадресацией.
3. По каким признакам можно определить атаку отказа в обслуживании?

Тестирование безопасности: изнутри и снаружи

1. СНАРУЖИ



Ни для кого не секрет, что количество сетевых атак неуклонно растет. Каждый день их совершается около 1000 (устаревшие данные), подсчитать же их число за год практически невозможно. Атакам подвергаются самые разные сетевые ресурсы – от сайта местного провайдера до федерального размера торговой сети суши и атомных станций. Можно даже посмотреть на интерактивную карту кибератак в режиме онлайн.

И каждый уязвим. И каждый отдает себе отчет об этом риске. И каждый думает, что это не про него. Что уж там душой кривить – мы так привыкли. Нежелание признавать риски приводит к нежелательным последствиям.

1.1 *Что же такое тестирование? А тестирование безопасности? А тестирование безопасности веб-приложения или веб-сервера (нужное подчеркнуть)?*

Тестирование безопасности веб-приложения – это попытка найти все те места, в которых могли допустить ошибку разработчики или просто не предусмотреть / забыть (подчеркните ваш случай).

Веб-приложение и веб-сервер неразрывно связаны. Тестирование одного без другого не даст полной картины бедствия. Тестируя защиту веб-приложения, мы ищем уязвимые места для атаки на пользователей. Тестируя защиту веб-сервера, мы ищем уязвимые места для атаки на сервер, его инфраструктуру.

1.2 *Для чего все это нужно? Почему так много компаний заинтересовано в проведении тестирования безопасности?*

Нужно все это в первую очередь людям: простым смертным, решившим заказать пиццу, а не отправить данные своей карты непонятно куда; простым владельцам пиццерий, не беспокоящимся, что кто-то смог бесплатно научиться заказывать пиццу через их приложение; простым разработчикам, которым не придется править код в 3 часа утра.

Как правило, небольшие организации, имеющие свой сайт и небольшой сервер с битриксом, думают, что они слишком малы для того, чтобы стать мишенью для атаки. И в этом они заблуждаются. Безусловно, вероятность таргетированной атаки на них ниже, чем на финансовых гигантов, но все-таки не равна нулю. В нынешнее время нейронных сетей и повальной автоматизации никто не будет выяснять, большой ли денежный оборот у фирмы. Главное – это количество уникальных посетителей, ведь суммарно в их карманах может оказаться больше «фишек», чем компания получает за год.

Существуют компании, которые уже взломали, и компании, которые еще не взломали. На практике это вопрос времени. Правильные компании, которые тратят на безопасность значительную часть прибыли, – это нормальный порядок вещей для всего цивилизованного мира. В нашей стране, к сожалению, пока к этому не пришли, считая, что студент-старшекурсник вполне способен настроить одну «циску» и запретить доступ к внутренним ресурсам извне (мол, «денег нет, но вы держитесь»). Ведь мы не привыкли тратить деньги на свою безопасность: бюджет отделов по информационной безопасности (ИБ) обычных компаний только сокращается. А ведь ежегодный ущерб от атак насчитывает более 85 млрд. $. С другой стороны, *многие адекватные* компании (читай, иностранные) формируют собственный штат по информационной безопасности, другие *многие* нанимают специалистов на аутсорс, *немногие* запускают программы баг-баунти, где любой желающий может принять участие в поиске уязвимостей. Безусловно, все они хотят сохранить свои доходы и свою репутацию.

1.3 *На что и на кого ориентировано тестирование безопасности? Кто получит выгоду от этого в первую очередь?*

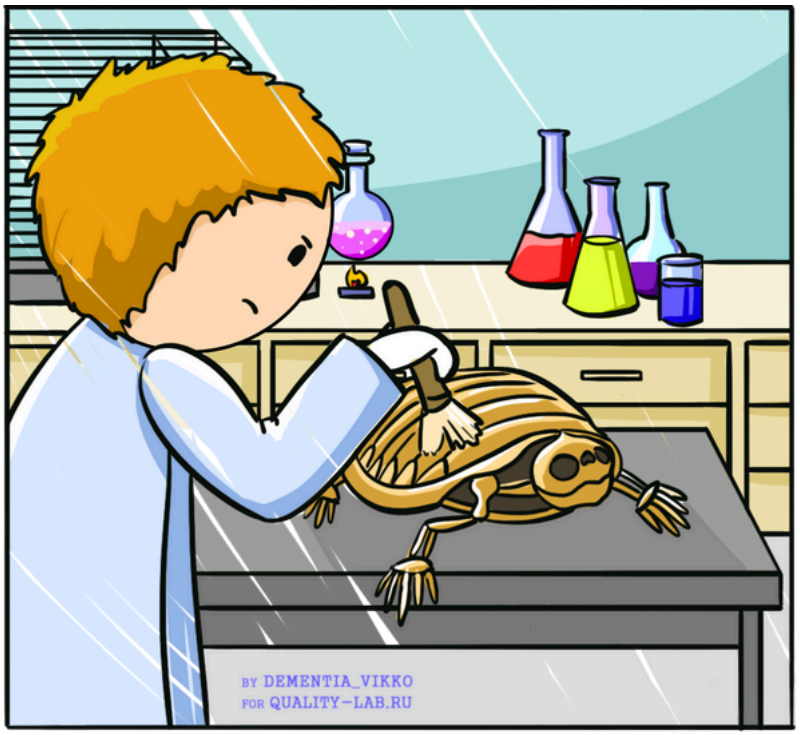
Конечно же, безопасность в первую очередь ориентирована на $. Кто говорит, что это не так, – нагло врет вам в лицо :).

Естественно, в первую очередь от безопасного серфа сайта выигрывает пользователь. Если нет нужды беспокоиться о том, что Ваши персональные данные могут утечь в сеть, то и доверять ресурсу Вы будете больше. А если счастлив пользователь, то счастлив и владелец веб-ресурса (и тем более он счастлив, чем меньше у него рисков потерять финансы).

Практическое задание

1. Проведите анализ предложенных информационных материалов.
2. Кратко сформулируйте – для чего и с какой целью проводится тестирование безопасности.

2. ИЗНУТРИ



Исследование безопасности веб-ресурса – сложная и кропотливая работа, требующая внимательности, фантазии и творческого подхода. Исследователю безопасности необходимо глубокое понимание технической изнанки работы веб-приложения и веб-сервера. Каждый новый проект дает пищу для фантазии, каждый новый инструмент – просторы для творчества. Да и вообще, тестирование безопасности больше похоже на исследовательскую работу – это постоянный поиск и анализ.

2.1 *Как тестирование безопасности выглядит изнутри? Чем руководствуется инженер? Как выставляет приоритеты?*

Каждый новый проект требует применения новых инструментов, изучения новых технологий, поглощения множества книг и статей, которые достаточно трудно найти. Большая часть информации находится в англоязычном пространстве, что добавляет определенные трудности в освоении материала людям, языком английским не владеющим.

Если говорить о процессе тестирования безопасности, то в целом он не сильно отличается от тестирования обычного: поиск, локализация, воспроизведение, заведение, отчет. Приоритеты, безусловно, зависят от заказчика, от целей тестирования.

К сожалению, некоторые курсы *по тестированию безопасности / анализу защищенности / аудиту безопасности* в интернете внушают, что достаточно пройтись по веб-приложению каким-нибудь сканером безопасности – и все готово! Отчет есть, уязвимости – вот они! Что же еще вам нужно? Но не стоит быть таким наивным. Большинство уязвимостей

ищется и находится именно вручную, при внимательном изучении. Они могут быть совершенно несложными, но автоматические сканеры пока еще не способны их обнаружить.

2.2 *Несколько слов об OWASP TOP 10.*

Классификацией векторов атак и уязвимостей занимается сообщество OWASP (Open Web Application Security Project) – международная некоммерческая организация, сосредоточенная на анализе и улучшении безопасности программного обеспечения.

OWASP (https://www.owasp.org/) составил список из 10-и самых опасных уязвимостей, которым могут быть подвержены интернет-ресурсы. Сообщество обновляет и пересматривает этот список раз в три года, поэтому он содержит актуальную информацию. Последнее обновление было сделано 2017 году («Огласите весь список, пожалуйста!» (с)):

- Внедрение кода;
- Некорректная аутентификация и управление сессией;
- Утечка чувствительных данных;
- Внедрение внешних XML– сущностей (XXE);
- Нарушение контроля доступа;
- Небезопасная конфигурация;
- Межсайтовый скриптинг;
- Небезопасная десериализация;
- Использование компонентов с известными уязвимостями;
- Отсутствие журналирования и мониторинга.

2.3 *Методика тестирования. GuidLine от OWASP.*

Организация OWASP дополнительно к своему списку из 10 самых опасных уязвимостей разработала методические рекомендации (практикум) по тестированию безопасности веб-приложений. В них подробно, шаг за шагом, описано, как и что необходимо тестировать, на что обратить внимание в первую очередь, а на что – во вторую. Методика носит рекомендательный характер и, конечно, никого ни к чему не обязывает (инженер, проводящий тестирование, некоторые моменты может перенести, а другие – вообще опустить), но, тем не менее, позволяет сделать максимальное покрытие для веб-приложения.

Итак, весь процесс тестирования состоит из двух этапов:

- Пассивный, во время которого тестировщик пытается понять логику приложения и «играет» с ним. Могут использоваться инструменты для сбора информации. Например, с помощью HTTP прокси можно изучить все HTTP-запросы и ответы. В конце данного этапа тестировщик должен понимать все точки входа приложения (HTTP-заголовки, параметры, куки и пр.).
- Активный. Во время данного этапа тестировщик проводит тесты в соответствии с методологией. Все тесты разбиты на одиннадцать подразделов:
  - сбор информации;
  - тестирование конфигурации;

- тестирование политики пользовательской безопасности;
- тестирование аутентификации;
- тестирование авторизации;
- тестирование управления сессией;
- тестирование обработки пользовательского ввода;
- обработка ошибок;
- криптография;
- тестирование бизнес-логики;
- тестирование уязвимостей на стороне пользователя.

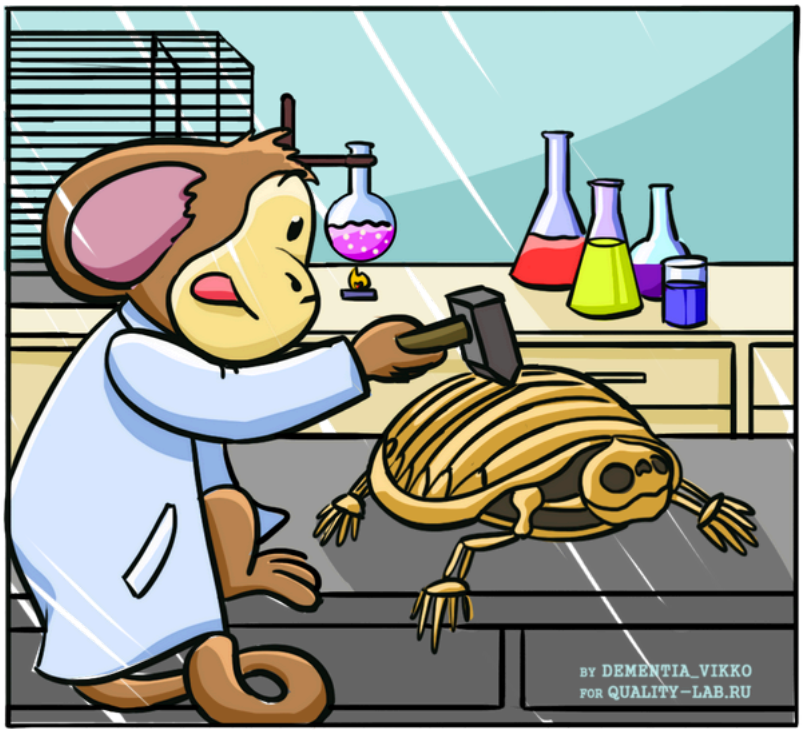
2.4 *Переход к технической стороне вопроса. Инструментарий – ручной или автоматизированный? Какой для чего?*

Какие инструменты используются для анализа безопасности? Оценив объемы бедствия, следует рассмотреть существующие инструменты. Конечно же, главные Ваши аргументы против несправедливости в сетевых именах – это глаза и мозг. На самом же деле, добрые люди разработали огромный инструментарий – начиная от специализированных скриптов, «заточенных» для какой-то одной конкретной цели, и заканчивая целыми комбайнами — готовыми выжать максимальные выводы из минимума вводных. К сожалению, часто такой результат оказывается лже-срабатыванием. Как правило, инженер по тестированию при выборе инструментов основывается на приоритетах: что важнее – время или область покрытия? Современные разработчики довели автоматизацию до небывалых высот, поэтому можно смело следовать принципу Парето: 80% работы скармливать автоматизированным анализаторам, а все оставшееся «проходить руками». Но, честно говоря, результаты автоматизированных средств все равно придется изучать и проверять.

Приведем небольшой список категорий инструментов:

- сканеры веб-уязвимостей;
- инструменты для эксплуатации уязвимостей;
- инструменты криминалистики;
- сканеры портов;
- инструменты мониторинга трафика;
- отладчики;
- руткит детекторы;
- инструменты шифрования;
- инструменты для брутфорса.

2.5 *Откуда же возникают все эти уязвимости?*



Из-за разгильдяйства разработчиков. Из-за невнимательности. Из-за халтуры. Из-за лени. Из-за-за…

Но чаще всего уязвимости возникают из-за неопытности. Не все разработчики представляют себе, как злоумышленник будет атаковать их продукт. Некоторые считают, что достаточно экранировать кавычку в пользовательском вводе или спастись «magic_quotes» – и можно будет избежать SQL-инъекции. Отсюда и получается, что превентивные меры мы принимаем, а что делать дальше – не знаем. И WAF'ы нас не спасут.