Московский государственный университет имени М.В. Ломоносова
Факультет Вычислительной Математики и Кибернетики
Кафедра Автоматизации Систем Вычислительных Комплексов
Лаборатория Безопасности Информационных Систем

Магистерская программа «Распределённые системы и сети»

Магистерская диссертация

Построение модели веб-приложения для обнаружения уязвимостей авторизации

Выполнил:

студент 621 группы

Василенко Анатолий Эдуардович

Научный руководитель:

с.н.с, к.ф.-м.н.

Гамаюнов Денис Юрьевич

Москва
2017

Аннотация

В работе исследуется вопрос создания системы автоматического обхода веб-приложения и построения его модели, отражающей возможный функционал, для дальнейшего использования с целью детектирования таких уязвимостей, как уязвимости авторизации. Проводится обзор существующих веб-краулеров и описывается задача построения модели. В работе вводится математическая модель веб-приложения и описывается алгоритмический подход к построению модели и выделению шаблонов, позволяющих обнаруживать одинаковые функциональные элементы веб-приложения. По мере описания модели формулируются различные ограничения, накладываемые на веб-приложение, для которого доступно построение модели на основе представления веб-приложения. Далее описывается, как созданная модель может быть использована для детектирования уязвимостей авторизации. В завершение описывается реализованное средство, созданное с целью построения описанной модели веб-приложения с механизмом выделения шаблонов.

1. Оглавление

2. Введение

В современном мире широко распространены клиент-серверные технологии. Их активное развитие и массовое использование способствует появлению различных недостатков, приводящих к уязвимостям в веб-приложениях.

При анализе веб-приложений на наличие уязвимостей часто используются полуавтоматические методы. Оператор вручную исследует веб-приложение с целью выявления ключевых особенностей, функционала и точек входа в веб-приложение и в дальнейшем проверяет их вручную или запускает для них определённые автоматические средства с предварительной настройкой. Однако именно оператор производит первичный анализ веб-приложения для выделения функционала, который необходимо проверить[1].

Таким образом, важна задача ***веб-краулинга***, т.е. *автоматического обхода веб-приложения*. Современные AJAX (Asynchronous JavaScript and XML) технологии позволяют создавать сложные динамические веб-сайты, контент которых может динамически изменяться и становиться доступным или не доступным в зависимости от разнообразных пользовательских действий (клики пользовательской мыши, нажатие клавиш, наведение курсора, …), приводящих к обновлению или изменению пользовательского представления веб-приложения. В следствие сложности создаваемых веб-приложений задача веб-краулинга существенно затруднена.

При анализе веб-приложения на наличие уязвимостей оператора интересует не контент веб-приложения, а его функционал, т.е. уникальные действия, которые могут совершать различные пользователи в рамках веб-приложения. Необходимо зафиксировать *логику работы* веб-приложения, т.е. при автоматическом обходе веб-приложения цель не в построении копии веб-приложения и фиксации контента, а в построении некоторой ***функциональной модели***, отражающей, действия, которые могут быть совершены в веб-приложении. Доступный функционал веб-приложения в отличие от контента, который может в нём содержаться, ограничен, так как на его создание было затрачено ограниченное количество ресурсов ограниченного числа разработчиков, в связи с этим задача построения функциональной модели веб-приложения ограничена и зачастую существенно меньше задачи построения копии веб-сайта.

[1] Примером такого анализа может быть обход оператором веб-приложения через прокси-сервер и последующий анализ запросов и ответов к серверу зарегистрированных прокси-сервером, а также повторение определённых запросов с изменёнными параметрами для анализа поведения веб-приложения.

Как правило анализ происходит без возможности доступа к серверной части веб-приложения, что приводит оператора к необходимости анализа методом чёрного ящика, основываясь лишь на *представлении* веб-приложения.

Примером уязвимости, основанной на логике работы веб-приложения является уязвимость авторизации[1]. Данная уязвимость является следствием недостатка, связанного с ошибкой проверки правил разграничения доступа между пользователями, позволяющей некоторому пользователю совершить запрещённое действие.

3. Постановка задачи

3.1.Цель работы

Целью работы является разработка алгоритма автоматического построения модели, отражающей функциональные возможности веб-приложения, для дальнейшего обнаружения такого типа уязвимостей как уязвимости авторизации.

3.2.Задачи работы

Для достижения цели работы необходимо решить следующие задачи:

1. Исследование существующих методов автоматического построения моделей веб-приложений, использующих технологии AJAX.
2. Разработка метода построения модели веб-приложения, отражающей доступный функционал веб-приложения.
3. Описание метода обнаружения уязвимостей авторизации на основе разработанной модели.
4. Разработка средства, поддерживающего создание модели для веб-приложений.

4. Обзор

4.1.Общее устройство веб-краулеров

[1] Уязвимость авторизации (уязвимость контроля доступа) входит в OWASP Top 10 ([6]) уязвимостей (4-я позиция). OWASP Top 10 – список самых распространённых типов уязвимостей в веб-приложениях.

Существует множество веб-краулеров созданных для обхода статических веб-приложений. Их главной целью является сбор контента веб-приложения и они до сих пор необходимы (например, для поисковых движков). Однако с появлением Web 2.0, т.е. с появлением веб-приложений, использующих AJAX на стороне клиента, их интерактивность сильно возросла. Появилась необходимость в веб-краулерах, способных обходить функционал веб-приложений для таких целей как тестирование. В результате появилась необходимость создания веб-краулеров для новых видов веб-приложений.

Существует несколько различных подходов к созданию средств построения моделей веб-приложений. Как правило для моделей используются графы состояний и переходов. Все веб-краулеры вынуждены реализовывать следующие операции: фиксация текущего состояния веб-приложения (или веб-страницы как текущего представления веб-приложения), обнаружение активных элементов веб-страницы (возможных пользовательских действий), выбор следующего пользовательского действия для анализа, перевод веб-приложения в состояние, позволяющее совершить выбранное действие, сравнение текущего состояния веб-приложения с уже зафиксированными моделью состояниями веб-приложения и критерий останова алгоритма (на основе времени, размера модели или совершённого количества пользовательских состояний). При этом метод выбора следующего пользовательского действия для анализа определяет стратегию обхода, очевидными стратегиями являются обход в глубину, ширину или выбор случайного действия. Одной из особых стратегий обхода веб-приложения является подход с вычислением наиболее перспективного пользовательского действия (такой подход рассмотрен в работе [(15)]), что позволяет обнаруживать большее количество состояний за меньшее время. Сочетание некоторых подходов, например, критерия останова на основе времени и обхода в глубину может привести к тому, что веб-приложение будет исследовано неравномерно.

Каждый веб-краулер программно состоит из следующих частей: веб-браузер (или его аналог для взаимодействия с веб-приложением), средство управления веб-браузером (для совершения действий от имени пользователя или считывания параметров открытой веб-страницы), робот (для выбора следующего анализируемого действия и перевода веб-приложения в подходящее состояние), компонента, содержащая единую создаваемую модель, и контроллер управляющий остальными компонентами.

В зависимости от степени автоматизации, обход веб-приложения может не требовать участия оператора, может требовать некоторой подготовительной работы или нуждаться в непосредственном участии оператора в процессе веб-краулинга. Преимуществом ручного обхода является возможность оператора вводить корректные данные в веб-приложение и выполнять осмысленные трассы в рамках веб-приложения, однако при таком подходе требуются человеческие

ресурсы и время, которые жёстко ограничивают количество действий, которые будут совершены в процессе анализа.

Действия пользователя могут привести к совершению действия в рамках веб-приложения в случае, если пользователь совершит определённый вид действия над элементом веб-страницы. Действие может быть задано либо в рамках атрибута элемента:

- *link*

либо посредством назначения обработчика события

- *<script type="text/javascript">*
  document.getElementById("id1").onclick = doSomething;
  </script>
- *<script type="text/javascript">*
  document.getElementById("id1").addEventListener("click", doSomething);
  </script>

при этом в первом случае обработчик может быть обнаружен посредством изучения исходного кода страницы, во втором случае обработчик события доступен из JavaScript контекста, однако в третьем случае обнаружить обработчик события является нетривиальной задачей.

Также, помимо действий пользователя к действию в рамках представления веб-приложения может привести срабатывание JavaScript таймера. Такие функции, как setTimeout и setInterval позволяют выполнять определённые действия в зависимости от времени. Это затрудняет анализ веб-приложения так как позволяет вносить изменения не зависимо от пользовательских действий.

После запуска обработчика события веб-приложение переходит в некоторое промежуточное состояния, в ходе которого изменения в представлении веб-приложения могут отобразится через недетерминированный промежуток времени. Приостановка веб-краулинга в ходе промежуточного состояния представляет некоторую техническую проблему.

Все веб-краулеры основываются на предположении о детерминизме веб-приложения (результат выполнения одного и того же действия в рамках веб-приложения в известном состоянии всегда приводит к одинаковому результирующему состоянию), так как в процессе анализа неоднократно встаёт задача возврата в известное состояние веб-приложения для исследования оставшихся пользовательских действий.

4.2.Различные средства автоматического построения модели веб-приложения

Рассмотрим несколько работ по созданию веб-краулеров и их основные параметры, задающие их возможности по анализу веб-приложений.

4.2.1. crawljax

Одним из самых популярных средств на сегодняшний день является “crawljax”, его разработчики опубликовали множество статей, среди которых [2], [3]. Средство реализовано на языке Java, для взаимодействия с веб-приложением используется средство selenium 2.0 предоставляющее возможность программного управления браузером. Для средства crawljax доступен исходный код[1]. В зависимости от требований могут использоваться различные компоненты-плагины, позволяющие, например, фиксировать предварительные действия перед началом веб-краулинга или действия при нахождении нового состояния, устанавливать фильтры перед сравнением веб-страниц, подменять метод сравнения состояний веб-страниц.

В качестве состояний веб-страниц принимается состояние DOM-документа веб-страниц, при необходимости пропущенных через фильтры. Авторы опираются на предположении о неизменности состояния на стороне сервера веб-приложения, либо действия пользователя, изменяющие это состояние должны быть обозначены для их последующего игнорирования в процессе анализа. Таким образом, средство crawljax строит модель «среза» веб-приложения. Далее под состояниями веб-приложения в рамках исследования crawljax подразумевается состояние веб-страниц.

Обход веб-приложения начинается с некоторого корневого состояния, получаемого после загрузки веб-приложения по некоторому заданному url. После каждого пользовательского действия используется метрика Левенштейна для принятия решения о том, изменилось ли состояние веб-приложения и в случае успеха фиксируется переход в новое состояние, после чего детектируются изменения веб-страницы и новые пользовательские действия.

Основываясь на предположении о детерминированности веб-приложения, при необходимости перехода в другое известное состояние веб-приложения для дальнейшего анализа, посредством модели обнаруживается доступная последовательность пользовательских действий, начиная с перехода на некоторый url веб-приложения. Таким образом реализована процедура возврата в некоторое предыдущее состояние веб-приложения.

Так как веб-приложение может быть иметь такую структуру, что его модель бесконечна, создатели вводят ограничение на обход в глубину в количестве 3-х действий, по достижении которого происходит возврат к родительскому состоянию.

---

[1] Веб-сайт средства crawljax: http://crawljax.com/

Обнаружение пользовательских действий происходит посредством выделения элементов по их типу (тег) и атрибутам, при этом используется специальный язык CASL, позволяющий перед обходом веб-приложения специфицировать параметры интересующих элементов, которые могут оказаться пользовательскими действиями. Так как при таком подходе велик риск совершить действие над элементом, у которого отсутствует обработчик этого действия, то необходимо идентифицировать изменилось ли состояние веб-приложения, как это описано выше. Для минимизации объёма действий, необходимых для исследования вводится предположение, что действия над элементами, совпадающими между дочерними и родительскими состояниями веб-приложения функционально эквивалентны для веб-приложения, таким образом после каждого изменения веб-приложения дальнейшему анализу подвергнуться только действия, являющиеся вновь появившимися.

Веб-приложение может не удовлетворять условию детерминированности и возвращать различные веб-страницы на одно и тоже действие, в случае если имеются, например, рекламные баннеры или временные метки. Для решения этой проблемы в работе [4] было предложено использовать фильтры, пропуская через которые DOM-представление веб-страницы контент, не связанный непосредственно с веб-приложением и его функциональностью будет удаляться.

В рамках оптимизации производительности средство crawljax было частично распараллелено, а именно создаётся несколько экземпляров компонент робота и браузера, результаты работы которых объединяются в общую модель. Так как подготовительная работа по переводу веб-страницы в необходимое состояние и загрузки контента занимает значительную часть времени, создаётся непрерывно пополняемый пул браузеров, готовых к совершению одного из интересуемых пользовательских действий и дальнейшего анализа состояния.

4.2.2. htcap

Иным средством анализа веб-приложения является “htcap” ([5]), веб-краулер, рассчитанный для анализа “Single-page application” – веб-приложения, представляемого одной единственной веб-страницей, однако работающей на основе AJAX технологии и потому способной изменяться. Данный краулер не строит модель веб-приложения, а решает задачу сбора различных веб-форм и запросов, которые могут посылаться пользователем для дальнейшего их анализа другими средствами анализа уязвимостей.

Средство обнаруживает доступные пользовательские действия посредством анализа атрибутов элементов DOM-документа, а также обнаружением определённых пользовательских из JavaScript контекста. После каждого пользовательского действия, выделяются изменения в структуре веб-страницы, для обнаружения новых элементов и их дальнейшего исследования.

Средство htcap рассчитано именно для «одностраничных веб-приложений»[1], т.к. после совершения пользовательского действия все изменения в рамках веб-приложения становятся видны сразу после действия и регистрируются для дальнейшего анализа. Такой подход позволяет обнаружить и зафиксировать зависимости между пользовательскими действия и их последствиями.

4.2.3. ReRIA, CReRIA, CrawlRIA

Исследователи Amalfitano, etc. ([7], [8], [9], [10]) разработали свою линейку веб-краулеров. Для моделирования используется конечный автомат, состояния которого соответствуют состояниям веб-приложения, а переходы соответствуют событиям в рамках веб-приложения.

Состояние веб-страницы трактуется как состояние веб-приложения. Отличительной чертой является разбиение представления на «виджеты» - набор элементов DOM-документа, с которыми ассоциированы обработчики событий (например, форма и её параметры это виджет). Для сравнения состояний веб-приложения вводятся различные метрики, сравнивающие типы присутствующих виджетов, их расположение и количество.

Исследователи допускают возможность изменения состояния веб-приложения по мере совершения пользовательских действий и поэтому основываются на стратегии обхода в глубину исследующей веб-приложение трассами (связными последовательностями действий) и требуют возможности перезагрузки веб-приложения в исходное состояние после исследования каждой трассы. При этом было разработано несколько версий программных средств:

- ReRIA – средство работающее под управлением оператора, совершающего действия в веб-приложении вручную. После сбора различных трасс строится модель веб-приложения.
- CReRIA – средство совмещающее сбор трасс под управлением оператора и немедленное внедрение новых трасс в модель. Это позволяет оператору обходить веб-приложение одновременно наблюдая создаваемую модель.
- CrawRIA – средство автоматически перебирающее трассы и составляющее модель, перезагружая каждый раз веб-приложение перед выполнением очередной трассы.

4.2.4. «Prototype crawling tool» от IBM

[1] «Single-page web-application»

В рамках совместного исследования между университетом и подразделением IBM ([11]) исследовались различные методы построения модели веб-приложений. Однако и здесь авторы ограничиваются предположением о неизменности состояния веб-приложения на стороне сервера, тем самым анализируя лишь некоторый «срез» веб-приложения, или предполагают доступной возможность перезагрузки веб-приложения, для возможности перевода веб-приложения в необходимое состояние.

Обнаружение пользовательских действий осуществляется на основе средства AppScan, реализованном компанией IBM, позволяющим отслеживать регистрацию всех событий независимо от их способа объявления. Для идентификации действия как уникальный ID используется комбинация текстового представления DOM-элемента, для которого установлен обработчик, типа доступного события и хеш-функции от обработчика события. Аналогично средству crawljax, перед анализом DOM-представления веб-страницы происходит фильтрация не релевантного содержимого, например, баннерной рекламы или временных меток. Состоянием веб-приложения является очищенное DOM-представление веб-страницы и список доступных пользовательских действий.

На основе базового исследования были рассмотрены подходы, позволяющие адаптироваться к некоторым типам веб-приложений: модель «меню» и модель разделения веб-приложения на независимые компоненты.

Модель «меню» ([12]) подразумевает наличие в веб-приложении множества различных виджетов, действия в рамках, которых всегда приводят в одинаковое состояние веб-приложения. Подобные «меню»-действия удобны для возврата в известные состояния веб-приложения и не требуют повторного выполнения из других состояний, в силу известности результата. Веб-краулинг такого типа веб-приложений предлагается делать в 2 этапа: этап выявления состояний, во время которого происходит обнаружение и исследование новых состояний и этапа проверки корректности, во время которого совершаются действия в рамках «меню»-виджетов, из различных состояний веб-приложения для проверки корректности предположения, что меню-виджет был верно выделен. На втором этапе могут случиться 2 вида ошибок: во-первых, действие может привести в иное известное состояние, чем ожидалось, в таком случае модель веб-приложения следует исправить, во-вторых, действие может привести в новое состояние, в таком случае могут быть обнаружены дополнительные пользовательские действия и веб-краулинг вновь переходит на первую фазу для продолжения анализа веб-приложения.

Модель разделения веб-приложения на независимые компоненты ([13]) предполагает полное отсутствие взаимозависимостей между данными компонентами, это позволяет обходить их по отдельности. Такой подход позволяет бороться с комбинаторным взрывом, образующимся из-за возможности совершения действия 1, а затем 2 или 2, а затем 1 и в обоих случаях прихода в одно и тоже состояние. Такое происходит при наличии в веб-приложении «ортогональных» событий.

Независимые компоненты предлагается выделять на основе следующей эвристики: компонентой веб-приложения является такое минимальное поддерево DOM-документа, которое изменяется после выполнения события.

Исследователи также рассматривали экстенсивный путь ускорения веб-краулинга посредством распараллеливания. Первым подходом является централизованное распределение задач с одним координатором и множеством узлов, работающих с собственным экземпляром веб-приложения. Такой подход действенен в случае, если затраты на обработку моделей существенно меньше затрат на взаимодействие с веб-приложением. Вторым описанным подходом является децентрализованное распределение, основанное на P2P сети с топологией кольца, для исследования веб-приложения таким образом необходимо разбить множество состояний веб-приложения на группы, которые достанутся узлам P2P сети.

4.2.5. WebMate

Средство WebMate ([14]) было разработано с целью использования его для тестирования AJAX веб-приложений. В процессе веб-краулинга постепенно покрывается код веб-приложения на стороне сервера, также веб-приложение можно открывать в нескольких веб-браузерах и сравнивать получаемые состояния веб-страниц для анализа меж-браузерной совместимости.

Исследователи указывают на то, что, либо состояние веб-приложения на стороне сервера не изменяется, либо есть возможность перезагружать веб-приложения, при необходимости перехода в определённое известное состояние. В качестве состояния веб-приложения принимается некоторая абстракция веб-страницы, позволяющая идентифицировать незначительно отличающиеся веб-страницы (например, временной меткой) как одинаковые состояния веб-приложения.

Для обнаружения доступных пользовательских действий, предлагается выделять такие элементы как «a», «button», «input», а также обнаруживать из JavaScript контекста объявленные обработчики событий.

4.3.Основные недостатки рассмотренных средств

Исследователями отмечалось, что большое количество ресурсов отнимает взаимодействие с веб-приложением и как следствие перевод веб-приложения в интересуемое состояние. Для решения этой проблемы предлагалось использовать распараллеливание подзадач веб-краулинга, использование кеша и создание пула веб-браузеров, заранее подготовленных к нужному пользовательскому действию.

Обнаружение доступного пользовательского действия в виде JavaScript обработчика, объявленного при помощи метода «*addEventListener*», до сих пор является затруднительным. Из

рассмотренных средств с этим справились только представители компании IBM посредством использования средства AppScan, позволяющим обнаруживать все объявления обработчиков событий.

Предпринимались попытки обнаружения особых частей DOM-представления веб-страниц, для дальнейшей оптимизации анализа веб-приложения: подход к веб-приложению с точки зрения «меню»-модели, разбиение на виджеты, выделение изменившихся частей DOM-представления как отдельных сегментов.

Все исследователи, создававшие автоматическое средство построения модели веб-приложения основывались на предположении о неизменности состояния веб-приложения на стороне сервера, либо требовали возможности перезагрузки веб-приложения. Подобные веб-краулеры не могут применяться к промышленным, динамическим веб-сайтам, позволяющим пользователю активно изменять веб-приложение и работающим в одном экземпляре в интернете.

Исследователи не рассматривали вопрос анализа веб-приложения от имени нескольких пользователей, что приводит к невозможности анализа случаев, когда действие одного пользователя приводит к некоторому результату у другого пользователя (например, посылка сообщения).

5. Модель веб-приложения

5.1. Введём общую терминологию

Веб-приложение – клиент-серверное приложение, в котором в качестве клиента используется веб-браузер. На стороне сервера могут использоваться любые технологии, взаимодействие между клиентом и сервером происходит посредством протокола HTTP.

URL (Uniform Resource Locator) – единый указатель ресурса.

DOM (Document Object Model) – объектная модель документа загруженной веб-страницы в браузере, является упорядоченным корневым деревом.

В данной работе при работе с деревьями всегда будет подразумеваться, что это упорядоченные корневые деревья.

Пользователь веб-приложения – это активный агент, использующий веб-приложение посредством веб-браузера.

Состояние веб-приложения в строгом понимании определяется в виде композиции состояний как на стороне сервера (например, состояние баз данных или состояние исполняемых процессов), так и на стороне клиентов (содержимое локальных хранилищ данных и текущее состояние контекста открытых пользователями веб-страниц). Однако такое определение состояния веб-приложения затрудняет возможности по дальнейшему его анализу, так как подразумевает учёт контекста на стороне всех клиентов и наличие специфических знаний о функционировании

конкретного веб-приложения. Состояние веб-приложения в рамках описываемой модели веб-приложения будет определено далее.

Дальнейшая задача заключается в построении модели веб-приложения методом чёрного ящика, основываясь лишь на представлении веб-приложения с точки зрения нескольких пользователей. При этом для определения состояния веб-страницы будем учитывать состояние DOM в некотором браузере, используемом пользователем, а для определения возможных действий пользователя будем дополнительно учитывать обработчики событий из JavaScript контекста и различные url, на которые пользователь может перейти. Для предотвращения повторного выполнения одного и того же типа действий в процессе автоматического обхода веб-приложения, будем выделять шаблоны[1], на основе которых создаётся представление веб-приложения, основываясь на предположении, что действия в рамках одного шаблона приводят к одинаковому функциональному поведению исследуемого веб-приложения.

Примеры возможного поведения различных веб-приложений, затрудняющих процесс веб-краулинга:

1. Действие пользователя на одной веб-странице может привести к изменению представления веб-приложения на некоторой веб-страницу другого пользователя, позволяя ему совершить новые виды действий.
2. Незначительные изменения могут привести к изменению семантики и результата действий пользователя (например, подпись кнопки «ок» или «отмена»), с другой стороны большинство текста веб-приложения является контентом и не влияет на поведение веб-приложения.
3. Действие пользователя может привести к изменениям на веб-странице, однако не совершить обмена с веб-сервером, что затрудняет подход к анализу веб-приложения посредством только http-запросов.
4. Действие пользователя при повторном совершении может приводить каждый раз к разным последствиям.
5. Могут присутствовать различного рода зависимости, например, взаимные зависимости в рамках веб-приложения, в результате которых одно действие пользователя влияет на доступность другого и наоборот.

[1] Примерами шаблонов могут являться иерархические комментарии на форуме, или объявления новостной ленты, или заголовок сайта, одинаковый для всех веб-страниц.

Таким образом, при создании модели необходимо учитывать различные возможности веб-приложения и накладывать ограничения на некоторые виды функционала, не позволяющие автоматизировать анализ веб-приложения.

5.2. Введём модель состояния веб-приложения

5.2.1. Пользовательское состояние веб-приложения

Допустим задана группа пользователей $users = \{u_1, \ldots, u_n\}$, от имени которых строится модель веб-приложения.

Введём ***состояние веб-страницы*** $S_{webPage}$ – некоторое отображение открытой в браузере веб-страницы. Состояние веб-страницы всегда рассматривается в контексте того пользователя, который её открыл.

Каждому состоянию веб-страницы $S_{webPage}$ соответствует набор ***возможных действий пользователя на веб-странице*** (нажатие кнопки клавиатуры, использование мыши, …) $clickables<S_{webPage}> = \{clk^i \mid i=0, \ldots, n\}$, каждое из которых переводит веб-страницу из состояния $S_{webPage}$ в новое состояние $S^i_{webPage}$:

$$S^i_{webPage} = clk^i(S_{webPage}),\ i=0, \ldots, n.$$

Допустим задан некоторый критерий, в соответствии с которым для каждой веб-страницы $S_{webPage}$ набор соответствующих ей clickables разбивается на два непересекающихся множества, соответствующих действиям пользователя, *изменяющим состояние веб-приложения* - pclk и *не изменяющим* это состояние - gclk.

$$clickables<S_{webPage}> = pclk<S_{webPage}> \cup gclk<S_{webPage}>,$$
$$pclk<S_{webPage}> \cap gclk<S_{webPage}> = \emptyset$$

Обозначим множество всех url, относящихся к данному веб-приложению, которые может ввести пользователь в адресную строку браузера: SURL. Переход по одному из возможных url пользователем u_1 в браузере будем обозначать:

$$S_{webPage} = uclk<u_1>(url),\ url \in SURL.$$

При этом множество url, изменяющих состояние веб-приложения будем обозначать SPURL, а не изменяющих состояние SGURL.

$$SURL = SPURL \cup SGURL;\ SPURL \cap SGURL = \emptyset$$

Таким образом, модель допускает следующие действия пользователя: действие на открытой ранее веб-странице и открытие новой веб-страницы.

Введём ограничение на веб-приложение фиксирующее, что предыстория действий пользователя u_p, не изменяющих текущее состояние веб-приложения не влияет на результат следующего действия пользователя в веб-приложении. ***Ограничение об отсутствии влияния предыстории***:

$$\forall\, S^{\kappa}_{webPage}\ \forall\, url \in SGURL\ \forall\, gclk^{\kappa} \in (gclk\langle S^{\kappa}_{webPage}\rangle \cup \{uclk\langle u_1\rangle(url)\})\ \exists\, S^{\kappa+1}_{webPage} : S^{\kappa+1}_{webPage} = gclk^{\kappa}(S^{\kappa}_{webPage})$$

$$\forall\, S^{0}_{webPage}\ \forall\, i \in \mathbb{N}\ \forall\, \{\forall\, url \in SGURL, gclk^{j} \in (gclk\langle S^{j}_{webPage}\rangle \cup \{uclk\langle u_p\rangle(url)\}) \mid j=0, \ldots, i-1;$$

$$S^{j+1}_{webPage} = gclk^{j}(S^{j}_{webPage})\} :$$

$$gclk^{\kappa} \in gclk\langle S^{i}_{webPage}\rangle \wedge S^{\kappa+1}_{webPage} = gclk^{\kappa}(S^{i}_{webPage})$$

Будем называть ***пользовательским состоянием веб-приложения,*** ориентированный граф $US_{webApp}\langle u_m\rangle = (G, V)$, где

1) G - множество всех состояний веб-страницы, которые могут быть достигнуты пользователем u_m, посредством действий, не изменяющих состояние веб-приложения:

$$G = \{S^{i}_{webPage} \mid (\exists\, S^{j}_{webPage}\ \exists\, gclk^{jk} \in gclk\langle S^{j}_{webPage}\rangle : S^{i}_{webPage} = gclk^{jk}(S^{j}_{webPage})) \vee (\exists\, url \in SGURL: S^{i}_{webPage} = uclk\langle u_m\rangle(url))\}$$

2) V - множество упорядоченных пар $(S^{j}_{webPage}, S^{i}_{webPage})$. Каждой паре соответствует действие пользователя $gclk^{jk}$ или $uclk\langle u_m\rangle(url)$, переводящее состояние веб-страницы от одного к другому. Данное соответствие биективно в силу условия о независимости результата действия пользователя от предыстории.

5.2.2. Модель веб-приложения

Будем называть ***состоянием веб-приложения*** с точки зрения группы пользователей $users = \{u_i \mid i=1, \ldots, n\}$ совокупность всех пользовательских состояний веб-приложения:

$$S_{webApp}\langle users\rangle = \{US_{webApp}\langle u_i\rangle \mid i=1, \ldots, n\}$$

Будем обозначать совокупность всех возможных действий пользователя u_i в состоянии веб-приложения $US_{webApp}\langle u_i\rangle$:

$$clickables\langle US_{webApp}\langle u_i\rangle\rangle = \bigcup_{S^{i}_{webPage} \in US_{webApp}\langle u_i\rangle} clickables < S^{i}_{webPage} >,$$

аналогично изменяющие и не изменяющие состояния веб-приложения будем обозначать:

$$pclk\langle US_{webApp}\langle u_i\rangle\rangle = \bigcup_{S^{i}_{webPage} \in US_{webApp}\langle u_i\rangle} pclk < S^{i}_{webPage} >$$

$$gclk\langle US_{webApp}\langle u_i\rangle\rangle = \bigcup_{S^{i}_{webPage} \in US_{webApp}\langle u_i\rangle} gclk < S^{i}_{webPage} >$$

При этом должно выполняться условие, что все возможные действия пользователя разбиваются на два непересекающихся множества изменяющих и не изменяющих состояние веб-приложения:

$$clickables<US_{webApp}<u_i>> = pclk<US_{webApp}<u_i>> \cup gclk<US_{webApp}<u_i>>,$$
$$pclk<US_{webApp}<u_i>> \cap gclk<US_{webApp}<u_i>> = \emptyset$$

Изменение состояния веб-приложения в следствие действия пользователя u_i на веб-странице $S_{webPage}$, трактуемого как изменяющее состояние веб-приложения, переводящее состояние из $S^н_{webApp}$ в $S^к_{webApp}$, будем обозначать:

$$S^к_{webApp} = pclk'(S^н_{webApp}), \text{ где}$$
$$pclk' \in (pclk<S_{webPage}> \cup \{uclk<u_i>(url) \mid url \in SPURL\}),$$
$$u_i \in users, S_{webPage} \in US^н_{webApp}<u_i>$$

В начальный момент времени веб-приложение находится в некотором состоянии:

$$S^0_{webApp}<users>$$

Введём ограничение на веб-приложение фиксирующее, что предыстория действий пользователей, изменяющих состояние веб-приложения не влияет на результат следующего действия пользователя, изменяющего состояние. ***Ограничение об отсутствии влияния предыстории*** (для действий пользователя, изменяющих состояние веб-приложения):

$$\forall S^к_{webApp} \forall u_m \in users \forall url \in SPURL \forall pclk^к \in (pclk<S^к_{webApp}> \cup \{uclk<u_m>(url)\}) \exists S^{к+1}_{webApp}:$$
$$S^{к+1}_{webApp} = pclk^к(S^к_{webApp})$$
$$\forall S^0_{webApp} \forall i \in \mathbb{N} \forall \{\forall u_p \in users \forall url \in SPURL, pclk^j \in (pclk<S^j_{webApp}> \cup \{uclk<u_p>(url)\}) \mid j=0,$$
$$\ldots, i\text{-}1; S^{j+1}_{webApp} = pclk^j(S^j_{webApp})\}:$$
$$pclk^к \in pclk<S^i_{webApp}> \wedge S^{к+1}_{webApp} = pclk^к(S^i_{webApp})$$

Будем обозначать ***моделью веб-приложения*** для группы пользователей users ориентированный граф $M_{webApp} = (G, V)$, где

1) G – множество всех состояний веб-приложения, которые могут быть достигнуты действиями пользователя, изменяющими состояние веб-приложения:

$$G = \{S^i_{webApp}<users> \mid [\exists S^j_{webApp}<users> \in G \exists pclk': S^i_{webApp} = pclk'(S^j_{webApp})] \vee$$
$$[S^i_{webApp}<users> = S^0_{webApp}<users>]\}$$

2) V – множество упорядоченных пар ($S^i_{webApp}<users>$, $S^j_{webApp}<users>$). Каждой паре соответствует действие пользователя pclk' в состоянии веб-приложения $S^i_{webApp}<users>$. Соответствие упорядоченных пар и действий пользователя является биективным, в силу

предположения о независимости результата пользовательского действия от предыстории действий других пользователей, изменяющих состояние веб-приложения.

5.2.3. Связь результата пользовательских действий и изменения представления веб-приложения

Введём ограничение на веб-приложение об отсутствии неконтролируемого изменения состояния веб-приложения и веб-страниц:

Любые изменения на веб-страницах должны быть обусловлены предварительным совершением некоторого действия от имени одного из пользователей.

Примером запрещаемых изменений на веб-страницах могут служить изменения на основе счётчиков времени (например, функций JavaScript “setInterval” и “setTimeout”).

Состоянию веб-приложения $S_{webApp}<users>$ соответствует множество доступных пользовательских действий, не изменяющих состояние веб-приложения $gclk<S_{webApp}<users>>$. Аналогично модели веб-приложения M_{webApp} соответствует множество пользовательских действий, изменяющих состояние веб-приложения $pclk<M_{webApp}>$, которые могут быть доступны пользователю в различных пользовательских состояниях веб-приложения на различных веб-страницах в этом состоянии. Процесс построения модели веб-приложения зависит от процесса автоматической смены состояния веб-страниц и веб-приложения.

Введём ***ограничение*** на веб-приложение ***об отсутствии отложенных эффектов*** у доступных пользователю действий:

Для любого обнаруженного пользовательского действия gfclk для пользователя u_k существует набор последовательностей пользовательских действий, не изменяющих состояние веб-приложения, начинающихся с действия открытия некоторого url и приводящих в состояние веб-страницы, позволяющее совершить действие gfclk, обозначим оператор получения набора последовательностей *E*:

$$\forall \{gclk^i \mid i=0, \ldots, n\} \in E(gfclk), \text{такова что}$$

$$gclk^0 := uclk<u_k>(url),\ url \in SURL,\ S^1_{webPage} = uclk<u_k>(url)$$

$$\forall i \in [1, n],\ gclk^i \in gclk<S^{i-1}_{webPage}>$$

$$\forall i \in [2, n],\ S^i_{webPage} = gclk^{i-1}<S^{i-1}_{webPage}>$$

$$gfclk \in gclk<S^n_{webPage}>$$

Введём оператор F над произвольной последовательностью действий $\forall \{gclk^i \mid i=0, \ldots, n\} \in E(gfclk)$, как оператор, удаляющий из последовательности все действия пользователя, которые могут быть опущены, так как не порождают дополнительных частей представления веб-страницы,

которые будут использованы в последующих действиях пользователя. Оператор F может быть реализован итеративно на основе последовательного удаления:

$$Если\ \exists j \in [1, n-1]: gclk^{j+1} \in gclk\langle S^{j-1}_{webPage}\rangle,$$

$$то\ gclk^{j}\ и\ S^{j}_{webPage}\ исключаются\ из\ последовательности\ \{gclk^{i}\}$$

Таким образом, для любого набора, возвращаемого оператором E, оператор F возвращает последовательность действий пользователя, в котором каждое действие добавляет в состояние веб-страницы DOM-элемент, позволяющий совершить следующее действие.

Ограничение об отсутствии отложенных эффектов: $\forall\ u_k \in users\ \forall\ gfclk \in US_{webApp}\langle u_k\rangle\ \forall\ \{gclk^i \mid i=0, \ldots, n\} \in E(gfclk)$, если совершить последовательность действий $F(\{gclk^i\})$ из произвольного начального состояния веб-станицы, то состояние веб-страницы изменится на состояние, позволяющее совершить действие gfclk пользователю u_k.

Ограничение об отсутствии отложенных эффектов позволяет зафиксировать, что для перевода веб-страницы в интересуемое положение отсутствует необходимость в совершении пользовательских действий, не приводящих к немедленным изменениям в представлении веб-страницы, необходимых для совершения последующего действия[1]. Описанное ограничение является *необходимым* для успешного автоматического анализа веб-приложения, так как в случае его невыполнения будет невозможно определить взаимное влияние различных действий пользователя основываясь лишь на представлении веб-приложения, однако ограничение *не* является *достаточным* для этого.

Аналогичным образом можно ввести ограничение об отсутствии отложенных эффектов для действий пользователя изменяющих состояние веб-приложения. Аналогично оператор E будет возвращать последовательность действий (различных пользователей) (изменяющих и не изменяющих состояние веб-приложения) позволяющих перевести состояние веб-приложения в такое, в котором доступно состояние веб-страницы в котором доступно некоторое выбранное действие pfclk. Аналогичным образом оператор F итеративно удаляет действия пользователя, не приводящие к изменению веб-представления необходимого для следующего действия.

---

[1] Иными словами, любое действие пользователя, изменяющее состояние веб-страницы приводит к немедленному изменению представления веб-приложения и фиксируется моделью.

Ограничение об отсутствии отложенных эффектов (для действий, изменяющих и не изменяющих состояние веб-приложения): $\forall u_k \in users \; \forall pfclk \in M_{webApp} \; \forall \{gpclk^i \mid i=0, \ldots, n\} \in E(pfclk)$, если совершить последовательность действий $F(\{gpclk^i\})$ из начального состояния веб-приложения, то состояние веб-приложения и веб-страницы изменится на состояние, позволяющее совершить действие pfclk пользователю u_k.

5.3.Адаптация модели для реальных условий

Тривиальным способом определения отображения веб-страницы браузера в модельное состояние веб-страницы $S_{webPage}$ является фиксация структуры DOM. Однако при таком подходе модель M_{webApp} для многих веб-приложений будет неограничена, так как многие современные веб-приложения позволяют пользователю создавать дополнительный контент, таким образом модель M_{webApp} будет содержать действия пользователя по созданию контента, появление которого в свою очередь будет снова порождать новые действия по созданию контента.

Для ограничения создаваемой модели веб-приложения, необходимо ограничить рассматриваемые действия пользователя. Для этого предлагается анализировать структуры DOM веб-страниц для выделения схожих частей - *шаблонов*.

Разработчики при конструировании большинства веб-приложений используют ограниченное число созданных ими частичных заготовок DOM - *шаблонов*, используя которые в дальнейшем конструируются веб-страницы для пользователей[1], таким образом нашей целью является выделение этих шаблонов.

Предположение о шаблонах: Будем предполагать, что действиям пользователя на веб-страницах в рамках одного шаблона для отдельно взятого пользователя[2] соответствует одинаковый функционал по обработке данного события со стороны веб-приложения.

Допустим задана некоторая ***метрика схожести*** 2-х упорядоченных изоморфных деревьев: $CS(T_1, T_2)$, – данная метрика истинна, если поданные на вход 2 дерева схожи и могут быть выделены в отдельный шаблон. Также метрика может применяться к одному и тому же дереву $CS(T_1, T_1)$ для определения, можно ли выделить данное дерево в отдельный шаблон (так, например, слишком

[1] Примером технологии поддерживающей такой метод разработки веб-приложения является «liquid-разметка».

[2] Допускается ситуация, когда в рамках одного шаблона действие разных пользователей может приводить к разному результату.

маленькие деревья, состоящие из 1-2 вершин, не следует выделять в отдельный шаблон, их схожесть может быть лишь совпадением).

Допустим метрика *CS* удовлетворяет условиям[1]:

- транзитивности: $\forall$ T1 $\forall$ T2 $\forall$ T3 CS (T1, T2) $\wedge$ CS (T2, T3) => CS (T1, T3)
- симметричности: $\forall$ T1 $\forall$ T2 CS (T1, T2) => CS (T2, T1)

Эти условия необходимы для снижения зависимости при выделении шаблонов от порядка обнаружения новых состояний веб-страниц при построении модели (независимость от порядка пользовательских действий)[2].

В рамках модели, будем сопоставлять ***шаблону*** T пару (T, TP), где T = (V, E, v_0) – есть упорядоченное корневое дерево (V – множество вершин, E – множество рёбер, $v_0 \in$ V выделена как корневая вершина). TP есть множество DOM поддеревьев $\{SD_i \mid i = 0, \ldots, n\}$ некоторых веб-страниц, соответствующих поддереву T, т.е. с которыми оно изоморфно и состоит в отношении $CS\ (T, SD_i)$. Будем называть TP множеством ***прообразов*** шаблона T, а T – образом элементов множества TP.

Таким образом, при построении модели веб-приложения в соответствии с предположением о шаблонах, достаточно ограничиться исследованиями действий пользователя лишь одного из прообразов каждого из шаблонов T, а также действиями пользователя, не относящимися ни к одному из прообразов шаблонов.

В качестве критерия для разделения действий пользователя изменяющих состояние от не изменяющих будем использовать следующее предположение:

Предположение об изменяющих состояние действиях пользователя: Все действия пользователя, приводящие к запросу на сервер с HTTP методом “POST”, “PUT”, “DELETE” или “PATCH” являются изменяющими состояние веб-приложения, остальные методы запроса HTTP протокола являются не изменяющими.[3] Открытие любого url также не изменяет состояния веб-приложения.

Для обнаружения возможных действий пользователя на веб-странице будем выделять JavaScript обработчики событий, установленные на узлы DOM (html-теги) после загрузки веб-

---

[1] Условие рефлексивности для описанной метрики не выполнено.

[2] Ввести класс эквивалентности на основе метрики CS нельзя, т.к. не удовлетворяется условие рефлексивности.

[3] Аналогичное предположение было предложено в работе [1], для восстановления модели веб-приложения по указанным оператором веб-страницам.

страницы (например, onclick, onkeypress, onsubmit, …), а также заполнение веб-форм и переход по ссылкам (тег «a»).

Для некоторых веб-приложений повторное действие пользователя после некоторого количества раз может привести к иному результату. В связи с этим, если после действия пользователя на веб-странице остаётся часть представления, позволяющая совершить действие повторно, то это действие следует повторить несколько раз (максимальное число раз является параметром системы), и в случае получения уникального результата после k-го раза, зарегистрировать для модели последовательное совершение одного и того же действия k раз как отдельный вид действия.

Следствие предположения о разделении доступных действий на изменяющие и не изменяющие состояние веб-приложения, а также следствием предположения о возможности различного результата при повторном выполнении одного и того же действия является необходимость совершения возможных пользовательских действий и уже последующей трактовки относительно типа совершённого действия. Это обусловлено тем, что, основываясь на представлении веб-приложения, невозможно заранее определить результат действия, для решения этой проблемы в будущем может быть использовано машинное обучение.

5.4. Алгоритмическая реализация построения модели веб-приложения.

Построение модели веб-приложения состоит из последовательных этапов выбора некоторого пользовательского действия, перевода веб-приложения и веб-страницы в состояние, позволяющее совершить выбранное действие и фиксации полученного состояния веб-приложения с выделением шаблонов в результате совершения интересуемого действия.

5.4.1. Выделение шаблонов.

При фиксации состояния веб-страницы в процессе веб-краулинга необходимо выделять шаблоны. Введём ограничение на выделяемые шаблоны, которое будет истинно на каждом этапе процесса построения модели веб-приложения:

фиксируемые прообразы всех шаблонов $TMPL = \{T_i, TP_i\}$, $i = 0, \ldots, n$, не имеют пересечений:

$$\forall\, i, j = 0, \ldots, n;\ i \neq j,\ \forall\, SD_{k1} \in TP_i\ \forall\, SD_{k2} \in TP_j,\ \Rightarrow V(SD_{k1}) \cap V(SD_{k2}) = \emptyset$$

Опишем процедуру анализа новой веб-страницы $S^n_{webPage}$ (на вход подаётся DOM дерево веб-страницы) и изменения множества шаблонов на основе покраски деревьев (см. алгоритм 1):

function ($S^n_{webPage}$)

begin

// Изначально все вершины покрашены пустым множеством цветов.

$SnT_{webpage}$ = множество всех веб-страниц, входящих в модель {$S_{webPage} \in M_{webApp}$}

foreach($S \in SnT_{webpage}$) {

Удаление из дерева S, всех поддеревьев, являющихся прообразами какого-либо из шаблонов T. При удалении вершин и инцидентных им рёбер из дерева S, остаётся несколько несвязных поддеревьев {S_1, …, S_m}. Вставим их вместо множества S:

$SnT_{webpage} = SnT_{webpage} \setminus \{S\} \cup \{S_1, \ldots, S_m\}$

}

// $SnT_{webpage}$ – есть множество веб-страниц модели без прообразов выявленных шаблонов

// Этап покраски деревьев

foreach ($D \in (SnT_{webpage} \cup \{T_i \in TMPL\})$) {

{T^1_i, T^2_i} = поиск всех пар максимальных изоморфных поддеревьев $T^1 \in S^n_{webPage}$, $T^2 \in D$.

// Максимальных означает таких, что $\nexists T^1{}' \in S^n_{webPage}$, $T^2{}' \in D$: T^1 является поддеревом $T^1{}'$ и T^2 является поддеревом $T^2{}'$

foreach((T^1, T^2) $\in$ {T^1_i, T^2_i}) {

if ($CS(T^1, T^2)$)

Для каждой вершины деревьев T^1, T^2 добавим в множество цветов уникальный цвет.

}

// Этап выделения шаблона

foreach(все максимальные поддеревья T^1 дерева $S^n_{webPage}$, вершины которых покрашены в одинаковое множество цветов){

if ($CS(T^1, T^1)$) {

Сформировать новый шаблон {T^1, TP}, где TP множество прообразов шаблона T^1, зафиксированных на стадии покраски вершин деревьев. Данные прообразы являются подграфами деревьев T^2 являющихся изоморфными парами к дереву T^1, выделяемыми на этапе покраски деревьев.

foreach($SD_j \in TP \wedge SD_j$ есть подграф некоторого шаблона $T_i \in TMPL$) {

Разбить дерево шаблона T_i на поддеревья {T^j_i}, удалив подграф SD_j из дерева T_i

Сформировать новое множество шаблонов на основе поддеревьев {T^j_i} без фиксации прообразов.

// Фиксация прообразов:

foreach($SD_{ik} \in TP_i$ – множество прообразов шаблона T_i) {

Разбить прообраз SD_{ik}, на прообразы, соответствующие поддеревьям $\{T^j_i\}$ и T^1, и добавить их к множеству прообразов шаблонов, основанных на $\{T^j_i\}$ и T^1

}

Удалить шаблон T_i (так как он был разбит на меньшие шаблоны).

}

}

}

}

end

Алгоритм 1. Процедура выделения шаблонов

Метрика схожести *CS* является параметром алгоритма построения модели и для различных веб-приложений может отличаться. Метрика может учитывать такие параметры DOM вершин, как тип тега, тип обработчиков событий для этого тега, атрибуты тега и его содержимое, а также анализировать структуры поддеревьев. Не следует требовать в метрике CS полного совпадения параметров каждой пары изоморфных вершин сравниваемых деревьев для признания их прообразами одного шаблона. Разные экземпляры шаблонов могут отличаться в незначащих местах, например, CSRF-токенах[1] или разница в id элементов. Также не следует признавать слишком маленькие деревья прообразами одного шаблона, так как это может быть лишь совпадением, а функционал веб-приложения на стороне сервера для этих действий может отличаться.

В случае, если некоторые деревья, которые содержат различные действия пользователя, будут ошибочно признаны прообразами одного шаблона (*false positive*), то это может привести к следующим последствиям:

1. для дальнейшего исследования веб-приложения возможно будет использоваться только один из прообразов шаблона, таким образом другой тип действия никогда не будет проанализирован, в этом случае снижается качество веб-краулинга.
2. в случае, если действие пользователя, соответствующее данному шаблону, необходимо будет повторить, однако для этого будет использован иной прообраз шаблона, по сравнению с предыдущим разом, то действие пользователя приведёт к не ожидаемому результату, что может привести к проблеме выбора дальнейших совершаемых

[1] Механизм защиты от CSRF атак основанный на передаче клиенту одноразового токена, который клиенту необходимо передать обратно серверу в случае совершения операции.

действий пользователем. Подобная ситуация также снижает качество веб-краулинга, так как цель с которой было совершено неверное действие пользователя (переход в ожидаемое состояние) не будет достигнута и исследование веб-приложения из ожидаемого состояния будет невозможно.

В случае, если некоторые деревья, которые являются прообразом одного шаблона не будут выявлены (*false negative*), то согласно модели веб-приложения пользовательские действия, соответствующие этим деревьям, будут исследоваться как различные. Это приведёт к росту размера создаваемой модели и времени её построения.

На практике при построении предложенной модели после детектирования нескольких прообразов одного шаблона может сохраняться для дальнейшего использования и сравнения лишь один прообраз, однако для дальнейшего использования модели в целях поиска уязвимостей может иметься необходимость в возможности отличить различные и одинаковые прообразы одного шаблона. Для этого можно использовать любую хеширующую функцию, позволяющую получить хеш в зависимости от прообраза.

Некоторые веб-приложения (такие как веб-форумы или поисковики), позволяют создавать и открывать однотипные веб-страницы. Ссылки на такие веб-страницы сами по себе являются частью некоторых других шаблонов в веб-приложении, исследование которых будет ограничено основываясь на механизме выделения шаблонов.

5.4.2. Переход между состояниями веб-страниц и веб-приложения

При веб-краулинге необходимо переводить веб-приложение и веб-страницу в состояние, позволяющее выполнить интересующее действие. В случае если это необходимо при построении пользовательского представления веб-приложения, то действия ограничиваются множеством действий, не изменяющих состояние веб-приложения, в ином случае могут использоваться любые пользовательские действия.

Основываясь на ограничении об отсутствии отложенных эффектов и предположении о том, что действия для разных прообразов, но в рамках одного шаблона соответствуют одинаковому функционалу веб-приложения, будем связывать доступные действия пользователей и изменение представления веб-приложения, которые эти действия порождают. Таким образом, для перевода состояния к тому, в котором доступно искомое действие clk, необходимо предварительно совершить действие clk’, результатом которого является появление той части представления веб-приложения, которая содержит действие clk. Подобным образом можно выделить

последовательность действий {clk^i}, переводящее текущее состояние веб-приложения и веб-страницу в искомые.

Если принять в качестве обозначений u – открытие пользователем страницы по url, p – действие пользователя, изменяющее состояние, g – действие не изменяющее состояние, то последовательность переводящая изменяющая состояние веб-страницы, но не состояние веб-приложения, выраженная через регулярное выражение может иметь вид: «(u|g)(g)*», - последовательность действий также изменяющая состояние веб-приложения: «(u|g)(g)*(p(u|g)(g)*)*», - действия пользователя типа g перемежаются действиями пользователя типа p.

Множеством пользовательских действий, необходимых для исследования являются действия уникальные с функциональной точки зрения. В соответствии с предположением о шаблонах, действия в рамках экземпляров DOM-представлений, являющихся прообразами одного шаблона и для одного пользователя, считаются идентичными с функциональной точки зрения.

При выборе следующего интересуемого действия пользователя для построения модели веб-приложения предпочтительно выбирать действия доступные из текущего состояния веб-приложения и состояния веб-страницы, это способствует уменьшению переходов между состояниями для достижения следующего исследуемого действия пользователя. Обход в первую очередь пользовательских действий, ставших доступными после последнего действия, позволяет снизить смешение различных сценариев поведения пользователя, так как после совершённого пользователем действия разумно предположить, что его дальнейшие действия будут связаны с изменённой частью представления веб-приложения (пользователь, работая с веб-приложением преследует какую-то цель). Снижение степени смешения различных сценариев поведения позволяет уменьшить риск взаимного влияния этих сценариев в случае, если веб-приложение не полностью удовлетворяет ограничению об отсутствии отложенных эффектов, что может привести к проблемам при автоматическом обходе веб-приложения, такой подход можно назвать *обходом в глубину*.

В конечном итоге при построении модели веб-приложения мы преследуем цель совершить все возможные действия пользователей различного функционала, за счёт выделения интересующего действия, перевода состояния веб-приложения в подходящее и совершения действия, после чего снова изучаются возможные действия, подобный подход можно назвать *жадным*.

6. Детектирование уязвимости авторизации

Привилегии пользователя – это множество разрешённых пользователю операций.

Роль – класс эквивалентности по отношению равенства привилегий. Будем считать, что один пользователь может принадлежать только одной роли.

Контроль доступа – функция системы, обеспечивающая технологию безопасности, которая разрешает или запрещает доступ к определенным операциям (объектам доступа), основанную на идентификации субъекта и объекта.

Аутентификация – процедура проверки подлинности предъявляемого пользователем идентификатора.

Авторизация – предоставление пользователю прав на выполнение определённых действий и контроль доступа к этим правам.

Политика контроля доступа – политика, описывающая правила контроля доступа.

Уязвимость контроля доступа (или ***уязвимость авторизации***) – недостаток в системе, позволяющий конкретному пользователю иметь доступ к некоторому ресурсу или совершать действия, запрещённые политикой контроля доступа.[1]

Главная особенность задания политики контроля доступа в том, что как правило отсутствует её строгое формальное описание, которое можно было бы автоматизировано анализировать некоторой информационной системой. В связи с этим для определения политик прибегнем к предположению, описанному в работе [1]:

Действия, которые доступны пользователю из его веб-интерфейса являются разрешёнными, остальные действия считаются запрещёнными.[2]

Таким образом, действия, отсутствующие в качестве доступных действий в модели веб-приложения, считаются запрещёнными.

На основе сформулированного предположения, для поиска *воспроизводимых* уязвимостей авторизации, достаточно находить действия, доступные одному пользователю, но не доступные другому и совершать эти действия от имени другого пользователя. Данный подход в работе [1] был назван «***дифференциальным анализом***»[3].

---

[1] Следует отличать уязвимость авторизации от уязвимости аутентификации. Предполагается, что аутентификация пользователя (т.е. проверка подлинности предъявляемого пользователем идентификатора) проходит успешно и корректно.

[2] Следует отметить, что данный критерий не является гарантированно надёжным, однако согласно исследованию [1], предположение формализует политику контроля доступа в достаточной мере, чтобы обнаруживать новые уязвимости авторизации.

[3] Однако в работе [1] для построения модели веб-приложения и последующего поиска уязвимостей авторизации требуется большое количество предварительной работы оператора, по совершению различных действий в веб-приложении вручную и пометке взаимосвязей между сценариями поведения пользователей для возможности автоматического перевода веб-приложения из одного состояния в другое.

Для поиска уязвимостей авторизации одного пользователя u_1 относительно другого пользователя u_2, запустим процесс построения модели M_{webApp} относительно пользователей users = $\{u_1, u_2\}$. Пара пользователей, для которых выполняется поиск уязвимостей авторизации могут быть как из разных ролей пользователей в веб-приложении, так и из одной роли, однако модель пользователей веб-приложения не обязательно ролевая, анализ может применяться к любой интересующей паре пользователей. После построения модели, проведём процедуру поиска уязвимостей авторизации:

---

function moveState (S_{webApp})
begin
// Перевод веб-приложения в состояние, соответствующее S_{webApp}
foreach ($u_i \in users$, $clk \in clickables<US_{webApp}<u_i>>$) {
Перевести состояние веб-приложения в то, в котором присутствует часть представления, позволяющая совершить действие clk пользователю u_i. (смена состояния описана в параграфе «переход между состояниями веб-страниц и веб-приложения»)
}
end

function searchVulnerabilities (M_{webApp})
begin
foreach ($S_{webApp} \in M_{webApp}$) {
moveState(S_{webApp});

{$dclk_j$, u_{ij}} := множество пар действий пользователя u_1 или u_2 изменяющих или не изменяющих состояние веб-приложения, и при этом доступных одному пользователю но не другому
foreach ($dclk_j$, $u_{ij} \in$ {$dclk_j$, u_{ij}}) {
Совершить действие пользователя $dclk_j$ от имени другого пользователя – не u_{ij}
Проверить успешность совершённого действия, и в случае успеха зафиксировать уязвимость авторизации.
moveState(S_{webApp}); // Восстановление состояния веб-приложения
}
}
end

Алгоритм 2. Процедура поиска уязвимостей авторизации

Для совершения действия пользователя от имени другого пользователя необходимо подменить аутентификационные данные. Существует несколько способов передачи аутентификационных данных, например, cookies браузера (самый распространённый метод), http basic authentication.

Проверка успешности совершённого действия от имени поддельного пользователя не тривиальна, так как различные веб-приложения могут реагировать самыми разнообразными способами[1]. Опишем несколько методов, и если один из них сработает, значит действие пользователя было заблокировано, иначе следует фиксировать уязвимость авторизации:

1. HTTP ответ класса 400 или 500 или несовпадение классов ответов (если ожидался ответ из класса 200, а был возвращён 300, и наоборот)
2. Сравнение состояний веб-страницы, с учётом различий в шаблонах после поддельного действия пользователя и после эталонного действия, совершённого при построении модели веб-приложения (учёт не только факта появления либо удаления шаблона, но и факт изменения количества шаблонов[2]).
3. Расчёт некоторой метрики расстояния (например, расстояние Левенштейна) HTTP-ответов на эталонное действие пользователя и на поддельное действие. (для использования данного метода необходимо отключить кеширование при совершении пользовательских действий, что замедлит производительность анализа)

Приведённый подход для поиска уязвимостей авторизации позволяет обнаружить уязвимости, не зависящие от предыстории действий других пользователей. Это обусловлено тем, что для обнаружения уязвимостей с зависимостями, когда для успешного совершения действия от имени другого пользователя, необходимо чтобы предварительно были совершены другие легальные действия различными пользователями, необходимо перебирать все возможные комбинации действий пользователей, что приводит к экспоненциальному росту количества компьютерных вычислений.

В общем случае после обнаружения уязвимости авторизации, веб-приложение может перейти в неконтролируемое состояние, в котором дальнейшая работа веб-приложения не может быть

[1] Например, перенаправлять пользователя на главную веб-страницу, или возвращать HTTP ответ с кодом ошибки, …

[2] Изменение количества шаблонов сигнализирует об успешном изменении контента веб-приложения (например, добавление или удаление дополнительного пользовательского комментария)

продолжена. В таких случаях следует приостановить анализ и либо восстановить веб-приложение[1], либо устранить обнаруженные недостатки и запустить анализ веб-приложения заново. Частым последствием при успешном выполнении запроса от имени чужого пользователя является смена пользователя, от имени которого веб-браузер взаимодействует с веб-приложением.

7. Дальнейшие направления исследования

Создание средства автоматического обхода веб-приложения является сложной и многогранной задачей. В данной работе описана лишь основная модель, однако на этом исследования не исчерпываются. Дальнейшие направления исследований могут иметь следующие направления:

1. Распараллеливание задачи веб-краулинга, в частности распараллеливание процесса построения модели, выделения шаблонов и параллельное использование нескольких агентов взаимодействующих с веб-приложением (веб-браузеров).
2. Применение машинного обучения для распознавания шаблонов и их возможного назначения с целью улучшения механизма выделения шаблонов, а также для обнаружения действий, нарушающих ограничение об отсутствии отложенных эффектов, для их особой обработки, или выделение шаблонов представления веб-приложения с целью обнаружения взаимосвязанных пользовательских действий[2].
3. Разработка оптимальной метрики схожести *CS* двух деревьев для признания их прообразами одного шаблона.
4. Инструментирование JavaScript для отслеживания трассы выполнения JavaScript обработчиков с целью обнаружения различных видов пользовательских действий, скрытых за прообразами одного шаблона, при различных состояниях веб-приложения[3]. Примером JavaScript фреймворка с делегированными обработчиками, не

[1] Для восстановления веб-приложения можно использовать, например, технологию linux-контейнеров или виртуальные машины.

[2] Например, выбор различных опций на веб-странице непосредственно перед пользовательской кнопкой «ок»

[3] Например, проблема делегированных обработчиков, при использовании которых обработчики событий привязываются к некоторому общему родителю множества других элементов. В этом случае при совершении действия пользователя в рамках одного из потомков, действие делегируется обработчику родителя, и в JavaScript контексте определяется, является ли действие пользователя значимым и к какому результату это приводит. В результате по установленному обработчику пользовательского действия невозможно определить какие действия доступны и невозможно привязать различное поведение веб-приложения к представлению этого веб-приложения.

позволяющими связать функционал веб-приложения с его представлением, может быть “jquery” [1].

5. Обнаружение доступных пользовательских действий не на основе DOM веб-станицы, а на основе визуального представления пользователю, т.е. на основе размера, типа и местоположения различных элементов на веб-странице.
6. Стабилизация работы веб-краулера за счёт автоматического детектирования автоматических login и logout. В веб-приложениях может быть автоматический выход из системы по различным причинам (по времени, сбою сессии, специфичное пользовательское действие), после которых необходимо автоматически производить вход в систему от имени соответствующего пользователя, для продолжения краулинга.

Функциональная модель веб-приложения может быть использована для различных аспектов, помимо обнаружения уязвимостей авторизации:

1. Возможность выделить связь между действием пользователя и изменением в представлении веб-приложения может облегчить обнаружение Stored-XSS уязвимостей.
2. Проверка аналогичная проверке уязвимостей авторизации на основе попыток совершения пользовательского действия, которое не доступно пользователю в текущем состоянии веб-приложения, но было доступно в другом состоянии ранее, также позволяет обнаружить недостаток в веб-приложении, позволяющий атакующему дублировать действия в веб-приложении, которые возможно должны быть доступны на выполнение лишь один раз.
3. Сбор «точек входа» в веб-приложение, т.е. обнаружение различных видов запросов к веб-серверу, для дальнейшей проверки различного вида уязвимостей (например, SQL-injection, XML-injection, SSL misconfiguration, корректность HTTP заголовков, …) и фаззинга.

8. Рекомендации при построении описанной модели веб-приложения

Действие пользователя, изменяющее состояние, может приводить к изменениям пользовательского состояния веб-приложения как для этого пользователя, так и для другого, это

[1] В работе [1] рассказано о подходе, как можно преодолеть проблему, описанную для фреймворка jQuery, однако этот метод специально настраивается для данного фреймворка и не может быть распространён автоматически на все веб-приложения.

обуславливает необходимость построения пользовательского состояния веб-приложения от имени каждого пользователя после совершения любого изменяющего действия. Если добавить ограничение, что действия других пользователей веб-приложения, не входящих в исследуемую группу *users* не влияют на представление веб-приложения для исследуемых пользователей, то описанный подход по автоматическому обходу функциональных возможностей веб-приложения применим к уже используемым другими пользователями веб-приложениям[1].

Так как при выделении шаблонов на основе метрики схожести CS могут возникать ошибки, приводящие к распознаванию одинаковых пользовательских действий как различных, веб-краулинг пользовательских состояний может не завершиться, в связи с этим на практике необходимо ввести ограничение на количество исследуемых действий и глубину[2] веб-краулинга.

Веб-приложение может обладать некоторыми особыми свойствами, в результате которых определённые пользовательские действия невозможно исследовать. Приведём несколько таких особенностей:

1. При построении модели веб-приложения пользовательские действия выполняются ограниченное число раз, таким образом, если в связи с некоторыми счётчиками, встроенными в веб-приложение, пользовательское действие на k-ом шаге приведёт к другим функциональным изменениям в веб-приложении, это может быть не выявлено.
2. В начальный момент времени модель веб-приложения может находиться в состоянии, в котором доступны несколько уникальных пользовательских действий, которые не будут никогда доступны после совершения любого из них. Таким образом, при построении модели веб-приложения будет совершено одно из уникальных действий, в результате чего остальные действия веб-приложения совершить будет уже невозможно.

Для улучшения работы средства построения модели веб-приложения, необходимо, чтобы выполнялись следующие условия со стороны клиентов и сервера веб-приложения:

1. *Лояльность*. Сервер не должен блокировать клиентам доступ к веб-приложению, если поведение пользователя с точки зрения сервера трактуется, как подозрительная

[1] Таким образом, подход позволяет анализировать «production» веб-приложения доступные в интернете и уже содержащие некоторый контент.

[2] Под глубиной подразумевается описанный ранее процесс выделения независимых сценариев поведения пользователя. Спустя некоторый лимит количества совершённых действий, необходимо вернуться к обходу доступных пользовательских действий, обнаруженных на предыдущих этапах.

активность. Главной целью является построение модели для обнаружения уязвимостей в веб-приложении, дополнительные средства обнаружения вторжения могут лишь навредить.

2. Поддержка кеширования может существенно ускорить процесс обмена данными по сети, что приведёт к ускорению работы веб-краулера.[1]
3. При построении модели веб-приложения отсутствует необходимость загрузки некоторых разновидностей контента, таких как картинок или стилей. Их отключение ускорит процесс загрузки веб-страниц, что приведёт к ускорению работы веб-краулера.

9. Практическая реализация средства автоматического обхода веб-приложения

В рамках данного исследования было разработано средство, названное «jaxsnoop»[2], позволяющее создавать модель веб-приложения на основе описанного механизма выделения шаблонов, а также механизма разделения пользовательских действий на изменяющие и не изменяющие состояние веб-приложения. Средство jaxsnoop основано на использовании технологий NodeJS и selenium-webdriver:

1. NodeJS[3] – программная платформа, основанная на движке V8, транслирующем JavaScript в машинный код, и предоставляющая API для работы с устройствами ввода-вывода и подключения внешних библиотек.
2. Selenium-webdriver – набор библиотек, позволяющий автоматически управлять различными браузерами. Набор библиотек selenium существует для ряда языков, в средстве jaxsnoop используется язык JavaScript. Альтернативами использованию selenium являются такие системы управления браузером, как slimerjs, phantomjs, casperjs, однако им свойственны определённые недостатки:
    a. API средств slimerjs и phantomjs почти полностью совпадают, однако slimerjs основан на движке Gecko, а phantomjs основан на WebKit. Что ограничивает возможность смены используемого типа браузера, в отличие от selenium. Также JS код управления браузером в selenium-webdriver является контекстом

[1] В исследованиях по созданию веб-краулеров отмечается, что самым ресурсоёмким по времени процессом при веб-краулинге приложений является взаимодействие с веб-сервером.

[2] Средство доступно по адресу https://github.com/phonexicum/jaxsnoop

[3] В основе NodeJS лежит событийно-ориентированное и асинхронное программирование с неблокирующим вводом-выводом.

выполнения NodeJS, что позволяет подключать различные библиотеки для NodeJS, в свою очередь для slimerjs и phantomjs такое невозможно.

b. Casperjs является надстройкой над slimerjs и phantomjs, добавляющей различные возможности, однако важным недостатком является необходимость полного описания сценария поведения пользователя по отношению к веб-браузеру перед непосредственным выполнением заданного сценария. Это затрудняет применение casperjs для построения модели веб-приложения, так как дальнейшие действия пользователя могут быть неизвестны заранее.

Для использования предположения об изменяющих состояние пользовательских действиях, позволяющих отличить изменяющие действия пользователя от не изменяющих было реализовано средство «ClientProxy»[1], являющееся HTTP/HTTPS веб-сервером, позволяющим перехватывать и изменять запросы и ответы при общении браузера с веб-сервером. Данное средство основано на NodeJS библиотеке «http-proxy», являющейся веб-прокси, рассчитанным для использования на стороне сервера. Теоретически, для решения задачи контроля общения с веб-сервером может использоваться любой прокси-сервер с возможностью управляемого перехвата запросов и ответов. Средство ClientProxy может быть запущено отдельно как простой веб-прокси, при этом будет происходить лишь логгирование запросов, либо использовано как NodeJS модуль, позволяя задать произвольную функцию обработки пар запрос-ответ, в последнем режиме данное средство и используется в jaxsnoop. При установлении соединения с веб-сервером посредством HTTP, веб-прокси достаточно просто передавать получаемые данные через себя, обрабатывая их установленными обработчиками, которым доступны данные из незашифрованного канала. При установлении соединения посредством HTTPS протокола, через веб-прокси происходит специальная процедура установки соединения, для возможности дешифровки трафика в целях его анализа:

- Предварительно веб-прокси берёт некоторый корневой сертификат (или автоматически его генерирует), на основе которого будут генерироваться поддельные сертификаты от имени веб-сервера для обработки дальнейших соединений.
- Веб-браузер посылает HTTP запрос с методом CONNECT для веб-прокси, при этом запрос имеет несколько HTTP заголовков, среди которых есть “Host”, указывающий веб-сервер, с которым веб-браузер собирается установить соединение.

[1] Средство доступно по адресу https://github.com/phonexicum/ClientProxy

- В ответ веб-прокси возвращает HTTP ответ “200 Connection established”, а также генерирует поддельный сертификат веб-сервера, соответствующий “Host” к которому происходит соединение.
- Веб-прокси создаёт HTTPS сервер, работающий на основе созданного поддельного сертификата, и все данные из тела метода CONNECT передаются на порт созданного HTTPS сервера. Далее происходит инкапсуляция HTTPS трафика в тело HTTP соединения методом CONNECT.
- Веб-браузер после получения от веб-прокси сообщения об успешной установке соединения начинает процедуру передачи данных веб-серверу в соответствии с HTTPS протоколом. При этом всё общение с веб-сервером передаётся в теле запроса метода CONNECT.
- Веб-прокси отвечает на данные приходящие от веб-браузера позволяя установить шифрованное соединение, а в качестве сертификата предоставляет поддельный сертификат веб-сервера.
- HTTPS сервер, порождённый веб-прокси на основе известного сертификата расшифровывает данные, получаемые от веб-приложения, и выступая в роли клиента передаёт эти данные веб-серверу, создавая новое HTTPS соединение. Таким образом, веб-прокси доступно содержимое шифрованного соединения, что позволяет применять те же обработчики запросов и ответов, используемые для HTTP прокси.

Так как предоставляемый сертификат является поддельным, то для успешного установления соединения необходимо либо отключить в веб-браузере проверку корректности сертификатов, либо добавить используемый в веб-прокси корневой сертификат как доверенный. Для экономии ресурсов затрачиваемых на генерацию сертификатов для различных серверов, они сохраняются и переиспользуются при повторных соединениях к различным веб-серверам.

Средство jaxsnoop состоит из следующих компонент:

- Компонента копирования DOM, – данная компонента выполняется в контексте JavaScript веб-браузера и последовательно обходя элементы DOM-представления веб-страницы создаёт упорядоченное дерево, каждый элемент которого отражает тип элемента (тег), его атрибуты, доступные JS обработчики событий (onclick), а также содержимое элемента.
- Компонента управления веб-браузером, в задачи которой входит открытие веб-браузера и его инициализация, вход в веб-приложение от имени определённого пользователя, совершение действий от имени пользователя, а также запуск компоненты копирования DOM и передача результата шаблонизатору для дальнейшего анализа.

- Шаблонизатор – компонента, принимающая на вход копию DOM-представления, созданную компонентой копирования DOM. Шаблонизатор содержит в себе следующие этапы:
    - Обнаружение изоморфных поддеревьев максимального размера в рамках одного переданного DOM-представления (возможны пересечения рассматриваемых поддеревьев) и при положительном результате метрики CS, покраска элементов этих деревьев в уникальные цвета (отдельный цвет для каждого поддерева).
    - Обнаружение изоморфных поддеревьев максимального размера между переданным DOM-представлением и всеми выделенными ранее шаблонами. Для подходящих пар поддеревьев, применяется метрика CS и в случае успеха поддерево DOM-представления окрашивается в уникальный цвет.
    - Аналогично предыдущему пункту обнаружение изоморфных поддеревьев максимального размера между переданным DOM-представлением и всеми DOM-представлениями, которые были обнаружены веб-краулером за всё время анализа.

  В результате этих 3-х этапов элементы различных поддеревьев обрабатываемого DOM-представления могут быть окрашены в различное множество цветов.
    - Обнаружение максимальных поддеревьев, все элементы которых окрашены в множество одинаковых цветов и применение к ним метрики CS. Если метрика истина, это означает, что было обнаружено поддерево-шаблон, которое изоморфно некоторым другим поддеревьям из того же DOM-представления или уже изученных представлений или известных шаблонов.
    - Происходит фиксация новых шаблонов, обнаруженных на предыдущем этапе, а также удаление из других DOM-представлений и шаблонов изоморфных поддеревьев. При удалении поддерева, оно заменяется на специальные элементы-ссылки, означающие, что на данном месте находится некоторый обнаруженный шаблон. Таким образом DOM-представления содержат в качестве элементов дерева, либо элементы DOM с параметрами, либо узлы, ссылающиеся на некоторый шаблон.

      При этом в процессе удаления поддерева из другого шаблона, он разбивается на части – новые шаблоны, на которые заменяются все ссылки в DOM-поддеревьях, таким образом DOM-представления имеют элементы-ссылки на шаблоны, а шаблоны состоят исключительно из образов DOM элементов веб-страниц.

Такой подход необходим для возможности однозначного восстановления состояния веб-страницы, так как если позволить шаблонам иметь в качестве элементов ссылки на другие шаблоны, затрудняется хранение модели приведённой на рис. 1. DOM-представления веб-страниц 1 и 2 будут иметь в качестве элементов-ссылок указатели на шаблон 1, а в свою очередь шаблон 1 будет иметь элемент-ссылку на шаблон 2, однако необходимо дополнительно сохранять “условную” информацию о том, что шаблон 2 отсутствует в шаблоне 1, в случае если шаблон 1 рассматривается, как дочерний для веб-страницы 2. При увеличении вложенности шаблонов и различных комбинаций подстановок шаблонов, а также использовании различных комбинаций шаблонов для различных веб-страниц, такое представление в рамках модели будет иметь сложную структуру зависимостей “условной” информации.

Рис. 1

Поэтому в соответствии с моделью в jaxsnoop, веб-страница 1 (рис. 2) будет содержать два элемента-ссылки: на шаблон 1 и на шаблон 2 соответственно. При этом элемент-ссылка на шаблон 2 будет дочерним элементом элемента-ссылки на шаблон 1, а также будет содержать в себе ссылку на DOM-элемент в шаблоне 1 (серая стрелка на рис. 2), для которого DOM-представление шаблона 2 является дочерним в исходной веб-странице.

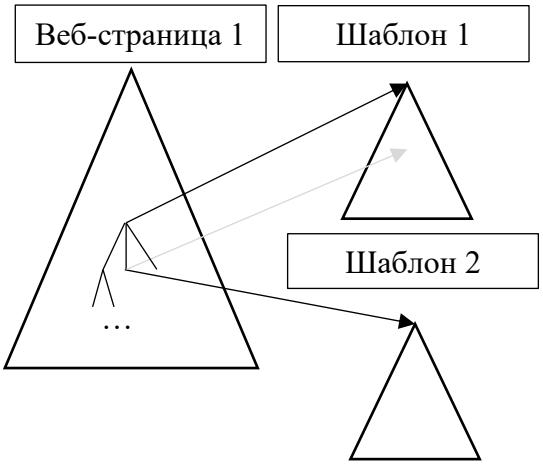


Рис 2. Структура с вложенными шаблонами.

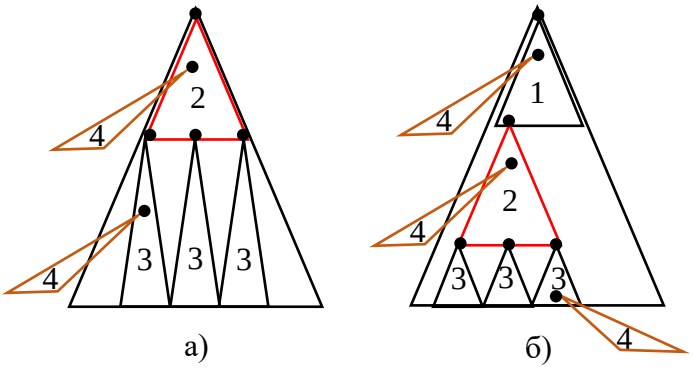
- На предыдущем шаге, в процессе удаления поддеревьев изоморфных обнаруженному шаблону, в случае если это происходит для некоторого DOM-представления, вырезанное поддерево заменяется на элемент-ссылку. В случае если это происходит для некоторого шаблона (красное поддерево на рис. 3), то в зависимости от расположения вырезаемого поддерева, шаблон может распасться на меньшие шаблоны 2-мя разными способами, в зависимости от того, является ли корень вырезаемого поддерева корнем шаблона (рис. 3а) или нет (рис. 3б). В обоих случаях присутствуют дочерние поддеревья для вырезаемого поддерева (поддеревья группы 3 на рис. 3), которые становятся самостоятельными шаблонами.
- 

Рис 3. Разделение шаблона на меньшие части

После разбиения шаблона на поддеревья, являющиеся отдельными шаблонами необходимо корректировать DOM-представления, ссылающиеся на исходные шаблоны, необходимо исправить элемент-ссылку (необходимо ссылаться на шаблон под номером 2 (рис. 3а) или 1 (рис 3б) в зависимости от случая) и добавить к дочерним элементам шаблоны, соответствующие поддеревьям, принадлежащим на рис. 3 к категории 3 (дочерние поддеревья). В случае рис. 3б, элементы-ссылки на шаблоны категории 3 являются дочерними к элементу-ссылке на шаблон 2, а элемент-ссылка на шаблон 2 является дочерним к элементу ссылке на шаблон 1.

Также родительское DOM-представление могло содержать дополнительные дочерние элементы, представляющие части DOM-представления, являющиеся дочерними для различных элементов разбиваемого шаблона (элементы на рис. 3 категории 4). В зависимости от принадлежности этих элементов к различным частям разбиваемого шаблона, их необходимо скорректировать как дочерние DOM-представления, прикреплённые к новым шаблонам, полученным из разбиения исходного шаблона.

- Обнаружение изоморфных поддеревьев двух деревьев происходит посредством последовательного перебора вершин заданных деревьев (вершинами являются

образы вершин DOM-дерева веб-страницы). В случае если вершины эквивалентны, то запускается процесс рекурсивного поиска дочерних эквивалентных вершин. При этом учитывается упорядоченность дочерних вершин, однако определённые вершины могут быть пропущены без нарушения изоморфизма упорядоченных поддеревьев, в случае если для них не найдено парной вершины, которой они эквиваленты.

Почленное сравнение на исходном этапе требует $O(x^2)$ количества сравнений (где x – среднее количество вершин дерева). Однако рекурсивное продолжение обнаружения дочерних эквивалентных вершин потребует более нескольких сравнений лишь при обнаружении действительно изоморфных поддеревьев, в остальных случаях количество сравнений будет ограничиваться средним числом дочерних элементов у отдельно взятой вершины. Таким образом, зависимость при обнаружении изоморфных поддеревьев является квадратичной.

Благодаря описанному методу обнаружения изоморфных поддеревьев, доступна возможность обнаружения шаблонов не только являющихся изолированной DOM-структурой, но и также являющихся DOM-структурой родительской для других DOM-структур. Аналогичный результат не может дать ни сегментирование снизу-вверх, ни сверху-вниз. Алгоритмы поиска максимально возможной подстроки двух строк, которые адаптируются для использования на деревьях при помощи «*Prufer*» кодирования ([16]) также не подходят. Они ищут максимальную подстроку, в то время как необходимо находить все возможные изоморфные поддеревья. Также недоступен поиск изоморфных поддеревьев с вырезанными DOM-структурами (пропуск не эквивалентных дочерних вершин).

- В результате состояние веб-страницы характеризуется DOM-представлением и множеством шаблонов, задействованных в нём.

  Состояние веб-приложения характеризуется множеством DOM-представлений и множеством шаблонов, задействованных в них.

- Компонента направленного изменения состояния веб-приложения (так называемый «робот»).

  В качестве стратегии обхода веб-приложения принята стратегия обхода в глубину с ограничением. По достижении ограничения следующее действие выбирается из множества обнаруженных на ранних этапах и веб-приложение переводится в состояние, позволяющее совершить данное действие. Для этого веб-приложение необходимо перевести в состояние, в котором доступна DOM-структура, в рамках

которой расположено интересуемое действие. Данная DOM-структура может быть доступна в других состояниях веб-приложения, в которые можно перейти при помощи определённых пользовательских действий. Такой процесс позволяет построить цепочку пользовательских действий, изменяющих состояние веб-приложения $\{pclk^i\}$ и переводящих его в состояние, в котором доступно исходное целевое пользовательское действие ($pclk^0$ должно являться действием, доступным из текущего состояния веб-приложения). Для каждого действий $pclk^i$ существует некоторый шаблон, в рамках которого это действие может быть совершено. Аналогичным образом строится последовательность действий пользователя $\{gclk^i\}$, переводящих состояние веб-страницы в необходимое для совершения действия $pclk^i$.

В случае, если создание цепочки невозможно, так как не известно действие переводящее веб-приложение в состояние, в котором доступен необходимый шаблон для дальнейших действий, целевое действие пользователя помечается, как временно недоступное и на его место выбирается другое действие пользователя.

В случае, если остались лишь действия пользователя, помеченные как недоступные или уже совершённые, все недоступные пользовательские действия переходят в категорию не совершённых. Если после этого снова невозможно совершить ни одно из интересующих действий, то процесс анализа веб-приложения останавливается.

Для поддержки работы с деревьями используется функция позволяющая рекурсивно обходить дерево в различных режимах (с заходом в каждую вершину последовательно или с заходом в каждую вершину, однако фиксацией каждой вершины лишь один раз) слева-направо и сверху-вниз.

Базовое ядро средства jaxsnoop содержит множество изменяемых параметров, среди которых:

- Фиксация для каждого элемента DOM-представления его определённых свойств.
- Сравнение двух образов DOM-элемента на эквивалентность.
- Метрика проверки эквивалентности двух изоморфных деревьев, а также метрика проверяющая возможность выделения дерева, все вершины которого покрашены в одно множество цветов как отдельный шаблон. По умолчанию обе метрики проверяют лишь количество вершин дерева, которое должно быть больше 3-х.

Для каждого пользователя должна быть задана функция входа в систему от имени соответствующего пользователя, которая будет выполняться при старте веб-браузера. Предполагается, что случайных выходов из системы нет, либо соответствующие действия заблокированы при помощи чёрного списка. Каждому пользователю соответствует свой веб-браузер с его контекстом выполнения. Каждый веб-браузер, управляемый посредством selenium, взаимодействует с веб-приложением через отдельный прокси сервер ClientProxy. Веб-браузеры

настроены таким образом, чтобы не проверять сертификаты и не загружать такой вид контента, как картинки.

Для проверки некоторого функционала средства jaxsnoop созданы unit-тесты.

Средство jaxsnoop применялось к следующим веб-приложениям: “pyforum”[1] и “easy JSP forum”[2]. Данные веб-приложения имеют несколько ролей пользователей и позволяют динамически создавать контент.

При обходе веб-приложения pyforum в качестве предварительных шагов необходимо создать функцию входа в систему от имени пользователя в веб-модель. Для ограничения некоторых действий в рамках веб-приложения, а именно запрета возможных специфических действий для администратора системы, введём ограничение и не будем исследовать действие пользователя, ведущее к открытию администрирования сайта, откуда можно повредить функциональность веб-сайта или сменить роли пользователей или изменить пароль пользователя. Также заранее создадим от имени администратора несколько тематик, в рамках которых пользователи могут создавать топики, содержащие сообщения. В качестве входной точки в веб-приложение обозначим главную страницу веб-приложения «/pyforum/default/index». Обход веб-приложения будем вести от имени обычного пользователя и пользователя с привилегиями администратора. После анализа первой же страницы веб-приложения, шаблонизатор выделяет DOM-структуры, соответствующие заголовкам существующих тематик. В рамках выделенного шаблона присутствует ссылка открывающая содержимое тематики, при совершении данного действия не происходит POST запросов, поэтому запросы не прерываются и открывается вторая веб-страница. При анализе второй веб-страницы большая часть DOM-структуры, включающая заголовок, нижнюю часть веб-страницы и панель справа выделяется как отдельный шаблон, так как эта часть неизменна для веб-приложения. В средней, отличающейся части DOM-структуры присутствуют действия создания нового топика, и подписки на содержимое данной тематики. Далее происходит анализ из текущего состояния первого выделенного шаблона на текущей веб-странице. Оно содержит различные виды действий (переход на главную веб-страницу, просмотр сообщений). Далее анализ происходит аналогично описанию первых двух шагов. Спустя некоторое время совершается действие по созданию нового топика, открывающее веб-страницу с веб-формой. Веб-форма заполняется случайными уникальными значениями и со всеми включёнными галочками (заполнение всех checkbox является поведением по умолчанию), после чего сначала будет совершено действие preview, создающее дополнительную DOM-структуру, не содержащую каких-либо действий, в следствие чего не

[1] Средство pyforum доступно по адресу https://bitbucket.org/speedbird/pyforum
[2] Средство easyJSPforum доступно по адресу https://sourceforge.net/projects/easyjspforum/

исследуемой в дальнейшем, а далее будет совершено действие add, создающее топик и перенаправляющее веб-приложение на страницу с топиками, действие будет отмечено, как изменяющее состояние веб-приложения. В результате будет создана форма, в которой будет доступно удаление существующих топиков. Аналогично, после прохода по различным состояниям, не изменяющим веб-приложение, данная форма будет заполнена, и будет совершено действие по удалению топиков. Отметим, что до сих пор происходил сбор состояний веб-страниц в каждом состоянии веб-приложения от имени двух пользователей. Однако после создания топика, у второго пользователя действие по удалению топика не существует. Таким образом после действия создания топика веб-приложение перешло в состояние, в котором множество доступных пользовательских действий не совпадает. В дальнейшем, при поиске уязвимости авторизации, веб-приложение достаточно перевести в состояние, в котором существует шаблон топика, который можно удалить и действие по удалению этого топика будет совершено от имени простого пользователя при помощи подмены аутентификационных данных cookies. Запрос завершится успешно, будет возвращён http ответ со статусом “200 OK”, а состояние веб-приложения перейдёт в состояние, в котором отсутствует шаблон, отвечающий наличию топиков. Таким образом, поддельное действие будет завершено успешно и уязвимость авторизации будет найдена.

Для анализа веб-приложения “easy JSP forum” аналогично напишем функцию входа в систему от имени пользователей. В качестве пользователей выберем администратора и обычного пользователя. Также исключим из рассмотрения действия, ведущие к изменению пароля пользователя, изменения ролей других пользователей или выходу пользователя из системы (logout). В веб-приложении присутствуют действия, не удовлетворяющие гипотезе о разделении изменяющих и не изменяющих пользовательских состояниях. Этим типом действий является действие по удалению комментария на форуме, оно приводит к HTTP запросу на сервер с методом GET, в связи с этим внесём некоторые изменения в средство jaxsnoop и будем трактовать элементы DOM-дерева, содержащие атрибут alt="Delete" как действия, изменяющие состояние веб-приложения.

Изначально в веб-приложении отсутствуют какие-либо форумы, поэтому изначально доступными действиями являются просмотр своего профиля и переход на домашнюю страницу, однако пользователю-администратору доступно действие создания нового форума, посредством заполнения веб-формы. После совершения этого действия будет создан форум, доступный всем пользователям. DOM-представление, отображающее новый созданный форум будет обнаружено и выделено в отдельный шаблон, так как сначала будет открыта веб-страница с форумом для одного пользователя, а далее веб-страница будет открыта другим пользователем, и после запуска процесса выделения шаблонов, будет обнаружен шаблон отвечающий за отображение форума. После открытия данного форума, будет открыта веб-страница со списком доступных обсуждений, и аналогичным образом список обсуждений будет выделен в отдельный шаблон. Однако в силу

предположения о том, что пользовательское действие в рамках одного шаблона, но для разных пользователей может приводить к разным результатам, открытие обсуждения произойдёт для обоих пользователей. В результате будет выведен список сообщений, который снова будет выделен в отдельный шаблон, однако у шаблона одного пользователя действия будут отсутствовать, а для пользователя-администратора будут обнаружены два действия: изменение сообщения и удаление сообщения. При дальнейшем анализе эти действия будут выполнены и функциональная модель веб-приложения будет построена.

Построенную модель веб-приложения «easy JSP forum» можно использовать для поиска уязвимостей авторизации. В изначальном состоянии веб-приложения пользователю-администратору доступна возможность создания новых форумов. Попытка создания форума от имени обычного пользователя (посредством замены cookies) не переведёт веб-приложение в новое состояние, в котором доступен ещё один форум (не будет обнаружен дополнительный экземпляр шаблона форума), таким образом в данном функционале уязвимость авторизации отсутствует. При дальнейшем рассмотрении модели, присутствует состояние веб-приложения, в котором администратору доступно действие удаления некоторого комментария с форума, при попытке совершения этого действия от имени обычного пользователя будет возвращён статус сервера «200 OK», и веб-приложение перейдёт в новое состояние, в котором будет на один шаблон меньше (шаблон сообщения веб-приложения), что означает, что состояние веб-приложения действительно изменилось. Таким образом, будет обнаружена уязвимость авторизации, позволяющая обычному пользователю удалять комментарии на форуме.

В заключение можно сказать, что практическая реализация веб-приложения основывается на механизме выделения шаблонов и разделении пользовательских действий на изменяющие и не изменяющие состояние веб-приложения. При некоторых ограничениях, это позволяет строить модели веб-приложений с динамически изменяемым контентом.

10. Заключение

В рамках данной работы были исследованы различные методы автоматического построения моделей веб-приложений и найдены недостатки, не позволяющие применять их к определённому классу веб-приложений.

Был предложен метод построения модели веб-приложений, учитывающий возможные изменения состояния веб-приложения на стороне сервера в следствие выделенных в отдельный класс пользовательских действий. Изменение состояния веб-приложения способно влиять на представление веб-приложения как для того же пользователя, так и для другого. Метод основан на выделении общих частей DOM-структур веб-страниц для обнаружения шаблонов содержащих

одинаковые пользовательские действия. На основе зависимости в появлении шаблонов при определённых действиях в веб-приложении и разделении пользовательских действий на изменяющие и не изменяющие, доступна возможность перевода веб-приложения в требуемое состояние для дальнейшего анализа без требования перезагрузки веб-приложения.

Для разработанной модели описаны доступные преимущества в анализе уязвимостей веб-приложений, в частности описан процесс по дальнейшему обнаружению уязвимостей авторизации при существующей модели.

Описано средство jaxsnoop, созданное для построения модели веб-приложения, содержащее такой базовый функционал, как выделение шаблонов и обнаружение доступных пользовательских действий.

Поставленные в работе задачи выполнены, цель по созданию алгоритма построения функциональной модели веб-приложения для дальнейшего обнаружения уязвимостей достигнута.

11. Список литературы

1. Носеевич Г.М. Обнаружение уязвимостей авторизации в веб-приложениях // – 2012
2. Mesbah A., Bozdag E., Van Deursen A. Crawling Ajax by inferring user interface state changes //Web Engineering, 2008. ICWE'08. Eighth International Conference on. – IEEE, 2008. – С. 122-134.
3. Mesbah A., Van Deursen A., Lenselink S. Crawling Ajax-based web applications through dynamic analysis of user interface state changes //ACM Transactions on the Web (TWEB). – 2012. – Т. 6. – №. 1. – С. 3.
4. Roest D., Mesbah A., Van Deursen A. Regression testing ajax applications: Coping with dynamism //Software Testing, Verification and Validation (ICST), 2010 Third International Conference on. – IEEE, 2010. – С. 127-136.
5. Filippo Cavallarin. “htcap” [Электронный ресурс]. – Режим доступа: http://www.htcap.org
6. OWASP. “Top 10 2017-Top 10” [Электронный ресурс]. – Режим доступа: https://www.owasp.org/index.php/Top_10_2017-Top_10
7. Amalfitano D., Fasolino A. R., Tramontana P. Experimenting a reverse engineering technique for modelling the behaviour of rich internet applications //ICSM. – 2009. – Т. 9. – С. 571-574.
8. Amalfitano D., Fasolino A. R., Tramontana P. Techniques and tools for rich internet applications testing //Web Systems Evolution (WSE), 2010 12th IEEE International Symposium on. – IEEE, 2010. – С. 63-72.
9. Amalfitano D. Reverse engineering and testing of rich internet applications. – 2011.
10. Amalfitano D., Fasolino A. R., Tramontana P. Using dynamic analysis for generating end user documentation for web 2.0 applications //Web Systems Evolution (WSE), 2011 13th IEEE International Symposium on. – IEEE, 2011. – С. 11-20.

11. Benjamin K. et al. A strategy for efficient crawling of rich internet applications //International Conference on Web Engineering. – Springer Berlin Heidelberg, 2011. – С. 74-89.
12. Choudhary S. M-crawler: Crawling rich internet applications using menu meta-model : дис. – Université d'Ottawa/University of Ottawa, 2012.
13. Moosavi A. Component-based crawling of complex rich internet applications : дис. – University of Ottawa, 2013.
14. Dallmeier V. et al. Webmate: a tool for testing web 2.0 applications //Proceedings of the Workshop on JavaScript Tools. – ACM, 2012. – С. 11-15.
15. Fard A. M., Mesbah A. Feedback-directed exploration of web applications to derive test models //ISSRE. – 2013. – Т. 13. – С. 278-287.
16. Huang W. Sequence-based Web Page Template Detection : дис. – Arizona State University, 2011.