Проблемы обнаружения логических уязвимостей в современных веб-приложениях

Раздобаров А.В., Гамаюнов Д.Ю.,
Лаборатория интеллектуальных систем кибербезопасности
Факультет ВМК МГУ имени М. В. Ломоносова
korse@seclab.cs.msu.su, gamajun@seclab.cs.msu.su

Доклад посвящен проблемам обнаружения уязвимостей в современных веб-приложениях. В докладе описываются трудности, которые возникают при анализе интерфейсов в связи с переходом от статических интерфейсов веб-приложений к динамическим. Рассмотрены требования, которым должны отвечать автоматические средства обнаружения уязвимостей для полноценной работы с динамическими веб-интерфейсами.

В докладе рассматриваются вопросы обнаружения уязвимостей в веб-приложениях, построенных на основе динамических технологий. Исследования по данному направлению ведутся в рамках проекта Фонда перспективных исследований на базе специально созданной на факультете ВМК МГУ лаборатории. Данный проект направлен на создание перспективных технологий оценки защищенности отечественного сегмента интернета от угроз, связанных с возможностью эксплуатации логических уязвимостей в веб-приложениях.

Задача автоматического обнаружения уязвимостей в веб-приложениях предполагает решение следующих подзадач:

- первичная навигация по веб-приложению и поиск входных точек, которые далее будут исследоваться. Под входными точками обычно понимается “параметризованный” HTTP-запрос, параметры которого будут заполняться на этапе подтверждения уязвимости;
- поиск скрытых функций. Помимо сценариев взаимодействия, предусмотренных пользовательским интерфейсом, приложение может содержать “скрытые” функции, доступ к которым нельзя получить непосредственно. Наиболее распространенные примеры таких функций - это панель администратора и различные отладочные интерфейсы.
- генерация и воспроизведение сценариев взаимодействия с найденными входными точками. Так как для обнаружения уязвимостей может потребоваться несколько вариантов сценариев взаимодействия с различными тестовыми данными, то эта подзадача добавляет дополнительное требование к подзадаче навигации - воспроизводимость активации входных точек. В простейшем случае активация – это посылка HTTP-запроса, в более сложных случаях в процесс активации могут входить дополнительные шаги, например, получение валидного CSRF-токена;
- анализ ответов веб-приложения. Некоторые техники обнаружения уязвимостей предполагают, что подтверждение уязвимости производится на основе отклонения поведения веб-приложения от “нормального” (например, т.н. слепые SQL-инъекции). Определение таких отклонений также можно отнести к требованиям к решению подзадачи навигации.

Первую подзадачу (без учета требования воспроизводимости) можно считать решенной для статических интерфейсов веб-приложений, т.к. входные точки полностью определяются по HTML-коду. В статических интерфейсах семантика и поведение элементов страницы являются фиксированными, таким образом, при клике пользователя по ссылке реакция интерфейса всегда будет одинаковой. В динамических интерфейсах поведение элементов может быть изменено программно (с помощью JavaScript-кода), что делает невозможным определение входных точек на основе HTML-кода в общем случае. Поэтому для решения задачи навигации для динамических интерфейсов большинство исследователей предлагают решения, основанные на движках браузеров или их эмуляторах, которые способны исполнять JavaScript-код [1, 2, 3]. Стоит отметить, что напрямую ни один из этих подходов не гарантирует выполнение требований предъявляемых к подзадаче навигации, но там присутствуют “сходные” задачи:

- определение и различение состояний интерфейса веб-приложения. Описанные подходы представляют веб-приложение как ориентированный граф, вершинами этого графа являются т.н. “состояния веб-приложения”, для построения такой модели веб-приложения необходимо отображение страниц веб-приложения на “состояния”;
- выбор стратегии обхода. Т.к. приложение представляется графом, то требование воспроизводимости активации входных точек переходит в возможность прохождения по пути в графе несколько раз (что является необходимым условием, для полного обхода графа).

В статических интерфейсах подзадачу поиска скрытых функций можно решать только путем перебора (fuzzing), таким образом, например, действует средство DirBuster. Базы перебора для таких средств обычно составляются на основе уже известных URL-адресов из других приложений. В динамических интерфейсах, помимо описанного метода, данная задача может решаться путем анализа JavaScript-кода, так как в коде могут быть содержаться функции, не используемые текущим интерфейсом. Помимо “раскрытия” информации, такие ситуации могут приводить к нарушению или отсутствию проверки прав доступа к функциям приложения. Предпосылками к такого рода ситуациям являются:

- количество JavaScript-файлов обычно намного меньше, чем количество различных состояний интерфейса, поэтому в файлах, загруженных для одного интерфейса, могут содержаться “лишние” функции;
- практики обслуживания запросов к статическому контенту. В целях повышения производительности веб-приложений запросы на статический контент могут обслуживаться отдельным сервером, который находится вне контекста исполняемого приложения, таким образом, отсутствует проверка прав доступа к JavaScript-файлам как таковая;
- практики сборки JavaScript-кода. Для уменьшения количества HTTP-запросов, JavaScript-код веб-приложения может собираться в один или несколько JavaScript-файлов, что также может приводить к отсутствию проверки прав доступа;
- шаблонная структура веб-приложений. Многие веб-фреймворки (django, ruby on rails, express, и т.п.) для рендеринга страниц используют шаблонизаторы, что позволяет переиспользовать блоки кода для различных страниц (таким блоком может быть подключение JavaScript и CSS-файлов). Возможны ситуации когда такие блоки минуют проверку прав доступа, например, блок из панели администратора будет включен во все страницы, относящиеся к административному интерфейсу, в том числе и в страницу аутентификации, доступ к которой можно получить не обладая административными правами;
- автоматическая генерация JavaScript-кода. Некоторые веб-фреймворки, такие как ASP .NET и Vaadin, предоставляют возможность автоматической генерации клиентского кода, на основе кода на стороне сервера. Что также может приводить к раскрытию информации или нарушению проверки прав доступа, например, если в клиентский код попадет переход на какой-нибудь адрес в зависимости от роли пользователя. Частично данная проблема решается путем обфускации клиентского JavaScript-кода.

Преимуществом анализа JavaScript-кода по сравнению с fuzzing’ом в статических интерфейсах является то, что код помимо URL-адресов может содержать и дополнительную информацию (например, имена и типы HTTP-параметров, значения HTTP-заголовков и т.п.).

Решения подзадач применения полезной нагрузки и анализа результатов должны быть расширены на уязвимости, характерные только для динамических интерфейсов, такие как:

- DOM XSS;
- уязвимости, связанные с неправильным использованием механизмов HTML5, например, window.postMessage или cross-origin resource sharing.

Таким образом, для анализа современных веб-приложений средства автоматического обнаружения уязвимостей должны уметь:

- осуществлять навигацию по веб-приложению, запоминая каким образом активируются различные входные точки;
- воспроизводить активацию входных точек используя полезную нагрузку, необходимую для обнаружения уязвимости;
- определять изменения в поведении веб-приложения, вызванные полезной нагрузкой;
- анализировать загружаемый JavaScript-код на предмет входных точек, активация которых невозможна через веб-интерфейс.

Литература

1. 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.
2. 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.
3. Amalfitano D. Reverse engineering and testing of rich internet applications. – 2011.