РАСШИРЕНИЕ МОДЕЛИ ОБНАРУЖЕНИЯ УЯЗВИМОСТЕЙ В WEB-ПРИЛОЖЕНИЯХ, ОСНОВАННОЙ НА СТАТИЧЕСКОМ АНАЛИЗЕ ИСХОДНЫХ КОДОВ

Оношко Д.Е.

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Бахтизин В.В. – канд. техн. наук

Предлагается способ расширения системы оценки для ранее описанной модели обнаружения уязвимостей в web-приложениях, основанной на статическом анализе исходных кодов.

По данным Открытого проекта безопасности web-приложений, в настоящее время наиболее значимой угрозой безопасности web-приложений являются инъекции [1]. При этом наиболее ценной для злоумышленника целью атаки являются запросы к системам управления базами данных (СУБД), использующимся подавляющим большинством современных web-приложений.

Существующие технические решения, такие как объектно-реляционное отображение (ORM), хранимые процедуры (stored procedures) или подготовленные выражения (prepared statements) позволяют существенно усложнить проведение подобной атаки злоумышленником, однако, как и экранирование полученных извне данных, не предотвращают возникновение уязвимостей в полной мере, поскольку обеспечивают требуемый эффект исключительно при грамотном применении, т.е. отсутствии ошибок со стороны разработчика. Поскольку подобные решения накладывают ряд ограничений, связанных с границами их применимости, также оказывается невозможным исключить обращения к СУБД в обход подобных решений в нестандартных задачах, а значит, и сопутствующие ошибки их применения.

Таким образом, важной мерой обеспечения качества web-приложений оказывается контроль их исходных кодов на предмет наличия ошибок, обусловливающих уязвимость к SQL-инъекциям.

Основой предложенного в [2] метода оценки качества web-приложений является модель обнаружения уязвимостей, предполагающая разделение анализируемого исходного кода на множество обобщённых процедур $P = \{p_1, p_2, \ldots, p_n\}$, где p_i характеризуется множеством оценок $m_i = M(p_i)$ её формальных параметров с точки зрения производимых процедурой преобразований данных, с последующим анализом передачи данных между этими процедурами. Предлагаемая бинарная система оценки позволяет обеспечить назначение оценок всем элементам путём упрощённой абстрактной интерпретации [3] исходного кода при сохранении невысокой сложности программной реализации метода обнаружения. Основной проблемой практического применения модели с бинарной системой оценки является большое количество ложных срабатываний (ввиду принципов назначения оценок — преимущественно ложноположительных).

Одним из возможных подходов для снижения доли ложноположительных результатов является расширение системы оценки таким образом, чтобы значения оценок соответствовали не только предполагаемому подмножеству значений соответствующего элемента данных (что соответствует оценкам S и U в модели [2]), но и другим свойствам элементов данных. Помимо ручного выбора элемента данных как имеющего оценку S (оценка UDS в исходной модели), примером таких свойств может быть, например, происхождение элемента данных и его оценки:

– данные, полученные извне;
– данные, полученные как значение литерала в исходном коде;
– данные, полученные из результатов выполнения запроса к СУБД;
– данные, полученные в процессе обработки внутри web-приложения;
– и т.п.

В некоторых случаях учёт этих свойств можно осуществить путём расширения базовой бинарной шкалы оценок. В общем же случае целесообразно рассматривать оценку m_i элемента данных как вектор $\vec{m}_i = (m_{i1}, m_{i2}, \ldots)$, каждый элемент которого соответствует одному из свойств.

В качестве примера такого расширения системы оценки можно рассмотреть следующий набор свойств:

– **возможность прямой подстановки данных в запрос:** U — подстановка приводит к уязвимости, S — подстановка допускается;

– **источник элемента данных:** Request — данные, полученные от пользователя вместе с запросом, Literal — данные, представленные литералом в исходном коде web-приложения, Database — данные, полученные из базы данных web-приложения, Derived — данные, полученные их

других данных в ходе обработки.

Нетрудно заметить, что некоторые комбинации значений свойств могут не иметь смысла. Так, например, строковые литералы, записанные в исходном коде, как правило, могут передаваться в процедуры, выполняющие запрос к СУБД, без ограничений, поскольку при этом не происходит изменения логики запроса. Тем не менее, комбинация (U, Literal) возможна, если при программной реализации метода обнаружения следует учесть возможность использования строковых литералов из недоверенных вспомогательных модулей web-приложения. Другая подобная комбинация свойств — (S, Request).

Следует понимать, что ключевым свойством в такой расширенной системе оценки по-прежнему остаётся свойство, отражающее возможность или невозможность прямой подстановки данных в запрос: именно по сочетанию значений этого свойства для формального параметра обобщённой процедуры и для передаваемого фактически элемента данных можно судить о наличии или отсутствии уязвимости. Однако сведения об источнике данных могут использоваться для назначения более точных оценок результатам некоторых операций и стандартных процедур. Так, например, результат конкатенации двух строковых значений в общем случае имеет оценку вида (U, *), где * — произвольное значение свойства, однако при конкатенации, например, двух элементов с оценками (S, Literal) результат получит оценку (S, Derived).

Дальнейшее расширение модели за счёт перехода к оценкам-векторам может заключаться в увеличении количества учитываемых свойств, а также в поддержке одновременного обнаружения уязвимостей к различным видам инъекций, а не только к SQL-инъекциям. Однако на практике увеличение размерности оценочного вектора приводит к повышению сложности анализатора, в том числе в связи с тем, что некоторые добавляемые свойства могут потребовать реализации дополнительных настроек анализа.

Одним из возможных способов сохранения баланса между сложностью анализа и достоверностью получаемых результатов при использовании предлагаемого способа расширения системы оценки является отложенное вычисление значений дополнительных свойств по упрощённой схеме.

Так, например, при анализе web-приложения, использующего несколько баз данных, может оказаться полезной информация не только о том, что путь к эксплуатации уязвимости связан с использованием данных, полученных от СУБД, но и о том, какая именно база данных выступает в качестве источника этих данных. При этом пользователю анализатора может быть предоставлена возможность отметить одну из баз данных как доверенную. Добавление информации о том, какая именно база данных является источником элемента данных, в оценочный вектор будет в этом случае нецелесообразным, поскольку сам по себе идентификатор или условный номер базы данных не несёт полезной информации для анализа.

Для приведённой выше системы оценки в подобной ситуации более простым решением будет добавление специального значения Trusted database для оценки данных, поступающих из доверенной базы данных. При этом информация о том, какая именно база данных является источником, которая потребуется только при формировании отчёта для пользователя анализатора, может быть извлечена из структур данных, используемых для представления структуры анализируемого web-приложения при восстановлении пути инъекции.

Таким образом, предлагаемый способ расширения модели обнаружения уязвимостей при сохранении сравнительно невысокой сложности программной реализации обнаружения уязвимостей позволяет уменьшить долю ложноположительных результатов и упростить отслеживание путей эксплуатации найденных уязвимостей.

Список использованных источников:

1. OWASP Top 10 – 2017 [Электронный ресурс]. — Режим доступа: https://wiki.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf. — Дата доступа: 20.03.2020.

2. Оношко Д.Е., Бахтизин В.В. Метод оценки качества web-приложений, основанный на обнаружении уязвимостей // Цифровая трансформация. 2018. №1(2). С. 58–65.

3. Cousot P., Cousot R. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints // Conference record of the Fourth ACM symposium on principles of programming languages. — 1977. — p. 238–252.