УДК 004.052.42:004.056.53

ПРИМЕНЕНИЕ СТАТИЧЕСКОГО АНАЛИЗА ИСХОДНЫХ КОДОВ В ОЦЕНКЕ НАДЁЖНОСТИ WEB-ПРИЛОЖЕНИЙ

Д.Е. ОНОШКО[1], В.В. БАХТИЗИН[2]

Белорусский государственный университет информатики и радиоэлектроники
ул. П. Бровки, 6, г. Минск, 220013, Республика Беларусь
[1]dimonsoft@sa-sec.org, [2]bww@bsuir.by

Исследованы существующие подходы к оценке надёжности и безопасности web-приложений с точки зрения их устойчивости к SQL-инъекциям. Предложен способ применения статического анализа исходных кодов web-приложений для обнаружения в них уязвимостей к SQL-инъекциям. Предложены числовые характеристики, основанные на данных, собранных в результате статического анализа, которые могут использоваться для оценки надёжности web-приложений.

Ключевые слова: надёжность, безопасность, web-приложение, SQL-инъекция, статический анализ исходных кодов, фильтрация данных.

Одной из основных тенденций в области разработки программных средств (ПС) в настоящее время является тенденция к переходу от классических desktop-приложений к web-приложениям. Между тем, размещение таких приложений на общедоступных серверах повышает требования к качеству этих приложений и, в частности, к корректности обработки поступающих от пользователей данных.

С точки зрения выполняемых функций непосредственно web-приложение представляет собой программный код, обеспечивающий преобразование запросов пользователей, как правило, по протоколу HTTP, в запросы к базе данных на языке запросов, в роли которого чаще всего выступает SQL. Анализ угроз безопасности, проведённый в рамках Открытого проекта обеспечения безопасности web-приложений (OWASP) в 2013 году показал, что наиболее распространённой угрозой для различных типов приложений и в том числе web-приложений являются SQL-инъекции [1].

Причиной уязвимости web-приложения к SQL-инъекции является наличие в нём ошибок, вследствие которых полученные от пользователя данные полностью или частично подставляются в текст запроса к системе управления базами данных (СУБД) без необходимой фильтрации. Это может быть использовано атакующим для внедрения таких специально выбранных последовательностей символов, что сгенерированный web-приложением запрос будет содержать необходимые атакующему команды SQL. Таким образом, недостаточная обработка (фильтрация) пользовательских данных открывает возможность выполнения произвольных действий на сервере приложений, а впоследствии и на компьютерах других пользователей web-приложения.

Для упрощения разработки web-приложений, не уязвимых к SQL-инъекциям, были предложены такие технические решения, как хранимые процедуры и подготовленные выражения. Между тем, они, хотя и имеют высокую эффективность, не являются достаточными, поскольку лишь обеспечивают безопасный механизм взаимодействия с СУБД, тогда как разработчик по тем или иным причинам может отказаться от их применения.

Несмотря на высокую популярность методов обнаружения ошибок посредством динамического анализа поведения web-приложения в условиях эксплуатации, их реальная эффективность остаётся крайне низкой, поскольку отсутствие проблем с точки зрения систем или компонентов, реализующих данные методы, не означает фактического

отсутствия уязвимостей в web-приложении. Единственным способом гарантированного обнаружения всех подобных ошибок, способных открыть возможность проведения SQL-инъекции, остаётся анализ исходных кодов, который является рутинной процедурой, трудоёмкость которой очевидно выше трудоёмкости разработки анализируемого web-приложения. Поэтому целесообразна её автоматизация с помощью ПС контроля качества кода, ориентированных на обнаружение потенциальных уязвимостей.

Для разработки подобных ПС предлагается подход, основанный на анализе потоков данных. Для этого анализируемое web-приложение рассматривается как множество $P = \{P_1, P_2, \ldots, P_N\}$ процедур, вызывающих друг друга с некоторыми параметрами. Формальным параметрам процедур назначаются оценкиэ, в простейшей реализации имеющие бинарный характер: «опасный» — данные, соответствующие параметру, будут корректно обработаны даже при отсутствии должной фильтрации, «безопасный» — данные должны пройти предварительную фильтрацию. Аналогичным образом назначаются оценки для возвращаемых процедурой значений: «опасный» — возвращаемые данные должны пройти фильтрацию перед подстановкой в текст запроса, «безопасный» — возвращаемые данные могут подставляться в запрос без дополнительной обработки.

Первоначально ПС известны оценки только для операторов языка программирования (также рассматриваются как процедуры) и некоторых стандартных процедур для взаимодействия с СУБД. Пусть на i-м шаге известны оценки для параметров и возвращаемых значений процедур $P'(i) = \{P_1, P_2, \ldots, P_{C(i)}\}$, где $C(i)$ — количество таких процедур на i-м шаге, а процедурой $P_{C(i)+1}$ используются только процедуры из $P'(i)$. Тогда, анализируя операторы $P_{C(i)+1}$, можно получить необходимые оценки для её параметров и возвращаемых значений: оценка фактического параметра совпадает с оценкой формального параметра. Последней анализируемой процедурой является процедура P_N, представляющая основную программу (главный блок begin…end, функцию main() и т.п.). Вычисленные оценки формальных параметров P_N (им соответствуют поступающие от пользователя данные) должны иметь значение «опасный». Параметры, для которых это не выполняется, являются потенциально уязвимыми. Анализируя путь, по которому была получена оценка, можно выявить причину возникновения уязвимости и предложить способы её устранения.

Пусть A — количество потенциально уязвимых параметров P_N, B — их общее количество. Тогда величина $X = 1 - \frac{A}{B}$ может использоваться в качестве оценки, характеризующей устойчивость web-приложения к SQL-инъекциям. Данная характеристика может отслеживаться на протяжении процесс разработки web-приложения с момента появления прототипа и использоваться для принятия проектных решений.

Список литературы

1. OWASP Top 10-2013. The Ten Most Critical Web Application Security Risks [Электронный ресурс]. — Режим доступа: http://owasptop10.googlecode.com/files/OWASP%20Top%2010%20-%202013.pdf. — Дата доступа: 31.10.2013.