УДК 004.052.42:004.056.53

МОДЕЛЬ ОБНАРУЖЕНИЯ УЯЗВИМОСТЕЙ В WEB-ПРИЛОЖЕНИЯХ

В.В. БАХТИЗИН, Д.Е. ОНОШКО

Белорусский государственный университет информатики и радиоэлектроники
П. Бровки, 6, Минск, 220013, Беларусь

Поступила в редакцию 3 июня 2015

Предложена модель обнаружения уязвимостей в web-приложениях, основанная на статическом анализе исходных кодов. Приводится описание положенной в основу модели системы оценок. Рассматриваются способы расширения модели для некоторых сложных случаев ее применения.

Ключевые слова: web-приложение, статический анализ, SQL-инъекция, уязвимость.

Введение

В настоящее время в сфере информационных технологий наблюдается тенденция к переходу от классических desktop-приложений к web-приложениям. Обусловленный такими отличительными особенностями web-приложений, как доступность из любой точки мира, кроссплатформенность и более простой механизм обновления, этот процесс сопровождается также ростом требований к надежности таких приложений.

По данным Открытого проекта обеспечения безопасности web-приложений (OWASP) в настоящее время наиболее распространенной угрозой для различных типов приложений (включая web-приложения) являются SQL-инъекции [1]. Анализ свойств этого вида угроз показывает, что главной причиной уязвимости web-приложений к действиям злоумышленников остается некорректная обработка данных, поступающих извне. В зависимости от их характера и того, каким образом они используются в web-приложении, «сырые» данные (raw data) необходимо подвергать фильтрации (обработке), которая может как сводиться к надлежащему экранированию специальных символов или проверке числовых данных на вхождение в заданные диапазоны значений, так и представлять собой более сложные алгоритмы. Если фильтрация поступающих от пользователя данных производится неправильно, оказывается возможным встроить в них фрагменты, которые при обработке web-приложением ошибочно будут распознаны как управляющие, что позволит злоумышленнику заставить его выполнить действия, не предусмотренные разработчиками web-приложения.

Обнаружение и исправление ошибок такого рода (также называемых уязвимостями) является трудоемкой рутинной задачей, для успешного решения которой требуется не только повышенная концентрация внимания, но и хорошее знание архитектуры разрабатываемого web-приложения, характер взаимосвязей его отдельных модулей и т.д. При этом любое изменение в коде самого web-приложения или программного обеспечения, входящего в состав платформы (например, обновление web-сервера), может внести новые уязвимости, а также заставить проявиться уже существовавшие, но остававшиеся скрытыми уязвимости. Следовательно, необходимо осуществлять контроль web-приложения на наличие в нем уязвимостей на протяжении всего жизненного цикла с момента появления первого прототипа.

Для обеспечения такого контроля на различных этапах процессов разработки и сопровождения необходимо решить две задачи:

– выявить и формализовать причины появления уязвимостей в web-приложениях (разработать модель web-приложения в контексте обнаружения уязвимостей);

– автоматизировать обнаружение уязвимостей к SQL-инъекциям со сбором информации об их характере (разработать модель обнаружения уязвимостей).

Модель web-приложения в контексте обнаружения уязвимостей к SQL-инъекциям

Для построения модели обнаружения уязвимостей к SQL-инъекциям в web-приложениях необходимо обозначить причины возникновения таких уязвимостей и сформулировать критерии, по которым можно определить, содержит ли их тот или иной фрагмент исходного кода.

На рис. 1 показан принцип обработки запросов web-приложением. Браузер передает web-серверу информацию о действиях пользователя в форме HTTP-запросов. Web-сервер обеспечивает формирование соответствующих этим запросам ответов. В зависимости от собственных настроек и типа запроса web-сервер может сформировать ответ самостоятельно (запрос на получение файла, хранящегося на сервере) или делегировать формирование ответа серверу приложений. Как правило, web-сервер и сервер приложений реализуют в виде программных средств, выполняющихся одним и тем же физическим узлом.

Рис. 1. Принцип обработки запросов web-приложением

В случае делегирования задачи формирования ответа данные из HTTP-запроса передаются серверу приложений, который, в свою очередь, обеспечивает выполнение web-приложения с заданными параметрами. В большинстве случаев для формирования ответа web-приложению приходится обращаться к системам управления базами данных (СУБД), взаимодействие с которыми, как правило, происходит с помощью SQL-запросов.

Таким образом, в контексте обработки запросов web-приложение может рассматриваться как некоторое ПС, выполняющее две основные функции:

– преобразование запросов клиента (браузера) в запросы к СУБД;

– преобразование ответов СУБД в ответы для клиента.

Данные для построения ответа могут быть получены web-приложением не только из баз данных, но и из других источников: сессий, файлов, переменных окружения и т.д. С точки зрения web-приложения эти источники фактически отличаются только способом взаимодействия с ними, поэтому в дальнейшем для обозначения всех возможных источников данных целесообразно применять более общее понятие – хранилище данных.

Современные web-приложения в зависимости от своего назначения могут являться достаточно сложными ПС с большими объемами кода и сложной логикой вышеуказанных преобразований. Для решения задачи обнаружения уязвимостей целесообразно рассматривать структуру web-приложений в соответствии с моделью, проиллюстрированной рис. 2.

В роли хранилища данных, как было отмечено ранее, чаще всего выступают системы управления базами данных. Хотя по отношению к web-приложению хранилище данных является внешним компонентом, доступ к нему (в т.ч. попадание данных, полученных от пользователя) возможен только через код web-приложения, поэтому данный слой на рисунке является центральным.

К ядру приложения в данной модели следует отнести код, составляющий основу web-приложения: код используемого в качестве основы фреймворка, набор вспомогательных процедур и классов, предоставляющих внешним слоям абстрактный интерфейс для доступа к хранилищу данных.

Слой бизнес-логики представляет собой код, отвечающий непосредственно за решение web-приложением возложенных на него задач. Размеры и состав этого слоя существенно варьируются в зависимости от предметной области.

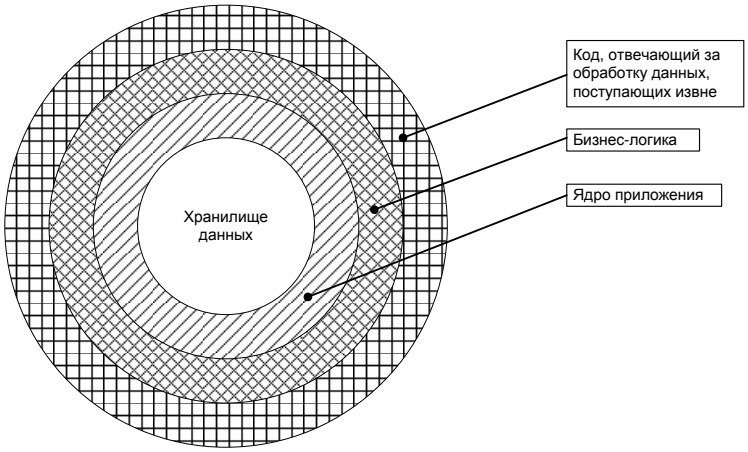

Рис. 2. Модель web-приложения в контексте обнаружения уязвимостей

Слой, который соответствует коду, отвечающему за обработку данных, поступающих извне, является уровнем абстракции, в задачи которого входит обеспечение взаимодействия кода бизнес-логики web-приложения с внешним миром.

Состав и размеры каждого слоя могут существенно различаться (вплоть до полного отсутствия) в зависимости от характера решаемых задач. Тем не менее, отсутствие какого-либо из предложенных слоев, как правило, говорит либо о невысокой сложности приложения, либо о недостаточной продуманности его архитектуры (некорректной или недостаточной декомпозиции задачи).

Такая модель web-приложения является развитием трехслойной модели из [2], где код, отвечающий за обработку данных, поступающих извне, назван слоем представления, слою бизнес-логики соответствует понятие домена, а ядро системы и хранилище данных представлены единым слоем «источник данных». Тем не менее, выделение хранилища данных как отдельного компонента является важным расширением модели, поскольку разработчик web-приложения использует этот компонент, как «черный ящик», не имея возможности изменить его поведение, а значит, ошибки в этом слое не являются ошибками web-приложения. Следовательно, исключение этого компонента из рассмотрения в рамках модели позволяет понизить ее сложность, не снижая эффективности.

SQL-инъекции, представляющие наибольший интерес ввиду их широкой распространенности, основываются на внедрении в SQL-запросы, посредством которых происходит взаимодействие web-приложения с СУБД (хранилищем данных), фрагментов кода, не предусмотренных разработчиками web-приложения. Причиной, по которой возникает такая возможность, является отсутствие, недостаточность или некорректность обработки (фильтрации) поступающих в web-приложение данных.

В качестве простого примера SQL-запроса к БД можно рассмотреть процедуру авторизации пользователя по переданной им паре значений «Логин-Пароль». Пусть web-приложение хранит информацию о пользователях и их правах доступа Users в таблице.

Пример таблицы БД с информацией о пользователях (Users)

ID	Login	PasswordHash	Access	…
1	Admin	fca1b7589df…	Full	…
2	JohnSmith	d1ba142e45…	Limited	…
…	…	…	…	…
89	BillGates	59df8512a1…	Limited	…
90	JoelSpolsky	2f135e15ad…	Full	…
…	…	…	…	…

Проверка соответствия переданных пользователем логина и пароля в данном случае сводится к проверке наличия в таблице записи со значениями соответствующих полей, равными переданным значениям. При наличии такой записи можно осуществить ее выборку и использовать полученную информацию для дальнейшей работы web-приложения (определение

прав доступа и т.п.). Таким образом, в общем виде запрос к СУБД при авторизации будет следующим:

```
SELECT * FROM `Users`
WHERE Login = '<Логин>'
 AND PasswordHash = SHA1('<Пароль>');
```

Web-приложение формирует запрос путем подстановки полученных от пользователя данных (логина и пароля) в соответствующие им части шаблона запроса и получает полноценный синтаксически корректный запрос на языке SQL. Например, если пользователь вводит в качества логина и пароля `JohnSmith` и `P@$$w0rd`, SQL-запрос примет вид:

```
SELECT * FROM `Users`
WHERE Login = 'JohnSmith'
 AND PasswordHash = SHA1('P@$$w0rd');
```

При очевидной простоте данного подхода, основанного на конкатенации фрагментов шаблона и полученных от пользователя данных, у него есть существенный недостаток. Пусть злоумышленник в качестве логина вводит значение `' OR 1=1; --`. В этом случае запрос примет следующий вид:

```
SELECT * FROM `Users`
WHERE Login = '' OR 1=1; --'
 AND PasswordHash = SHA1('P@$$w0rd');
```

Последовательность символов `--` в наиболее часто применяемой для web-приложений СУБД MySQL является признаком начала комментария, поэтому запрос, который фактически будет выполнен, окажется следующим:

```
SELECT * FROM `Users` WHERE Login = '' OR 1=1;
```

Благодаря второй части условия (`1=1`), всегда принимающей значение «Истина», результатом данной выборки будут все записи таблицы. Продолжая варьировать вводимое в качестве логина значение, злоумышленник может сформировать условие, которому будет соответствовать учетная запись с административными правами, и тем самым повысить свои привилегии, получив больший (а возможно, и почти неограниченный) доступ к web-приложению и обрабатываемым им данным. Данный пример иллюстрирует простейшие уязвимость к SQL-инъекции и способ ее эксплуатации.

Модель обнаружения уязвимостей

Как было показано ранее, причиной уязвимостей к SQL-инъекциям является попадание полученных от пользователя данных, не прошедших надлежащую обработку (фильтрацию), в запросы к СУБД. Также было отмечено, что все действия, выполняемые кодом web-приложения, можно свести к двум преобразованиям:

– преобразованию данных HTTP-запроса в запросы к хранилищу данных;

– преобразованию данных HTTP-запроса и данных, полученных из хранилища данных, в HTTP-ответ (HTML-страницу, JSON- или XML-данные и т.д.).

Исходя из этого, в рамках модели обнаружения уязвимостей web-приложение можно рассматривать, как конечный автомат Мили (рис. 3).

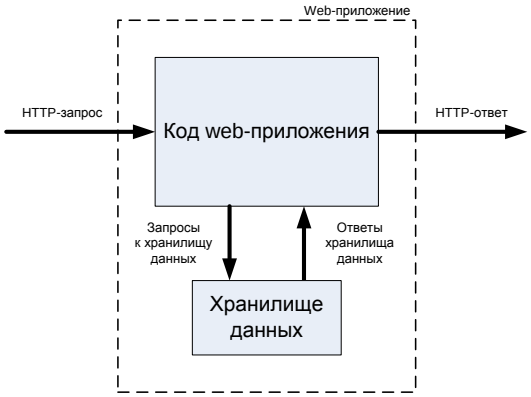

Рис. 3. Представление web-приложения как автомата Мили

Входным сигналом является HTTP-запрос, поступающий от клиента; выходной сигнал – ответ на этот запрос. Ответ зависит от параметров, переданных с запросом, и данных, полученных из хранилища данных. При этом, основываясь на предложенной модели web-приложения в контексте обнаружения уязвимостей, этот автомат следует разделить на две части: хранилище данных (внутренний слой модели web-приложения) и собственно код web-приложения (внешние слои).

При таком разделении существует четкое соответствие: состояние автомата представлено хранилищем данных, а код web-приложения выступает в роли функции, преобразующей аргументы (входной сигнал и состояние автомата) в результирующее значение (выходные сигналы и новое состояние автомата).

Преимущество такого разделения заключается в том, что код web-приложения не содержит запоминающих элементов, а значит, его поведение будет постоянным для одних и тех же исходных данных. Следовательно, для оценки этого поведения не требуется подготовка тестовых данных, а значит, и запуск web-приложения. Другими словами, возможна формальная верификация web-приложения с точки зрения его уязвимости к SQL-инъекциям путем статического анализа его исходных кодов.

В рамках предлагаемой модели обнаружения уязвимостей для формальной верификации используется абстрактная интерпретация [3]. При этом предлагается рассматривать web-приложение как множество процедур, связь между которыми устанавливается путем их вызова друг другом. Это целесообразно, т.к. работа с хранилищем данных, так или иначе, сводится к вызову стандартных процедур языка программирования или процедур стандартной библиотеки. Кроме того, вывод ответа на HTTP-запрос также можно выразить в терминах вызова стандартных процедур. При этом все свойства этих процедур известны заранее.

В соответствии с принципами восходящего проектирования точка входа (или главный блок) web-приложения также является процедурой, параметры и возвращаемое значение которой совпадают с входными и выходными данными web-приложения, а логика реализована с использованием иерархии вызовов более простых процедур, на нижнем уровне которой находятся стандартные процедуры.

Для удобства дальнейшего анализа необходимо изменить подход к рассмотрению параметров процедур. Во многих языках программирования помимо параметров, передаваемых по значению, существует возможность передавать параметры по указателю (ссылке), что позволяет вызываемой процедуре изменять значение переменной, использованной в качестве фактического параметра. Для того, чтобы обеспечить применимость модели к любому из существующих императивных языков программирования, предлагается выделить два вида параметров процедур:

– in-параметры – для описания данных, передаваемых в процедуру;

– out-параметры – для описания данных, возвращаемых из процедуры.

Возвращаемое значение функции может рассматриваться как особый случай out-параметра.

Следует отметить, что in- и out-параметры используются для обозначения передачи данных только в одном направлении (либо в процедуру, либо из нее). Параметры, позволяющие выполнять передачу в обоих направлениях, необходимо при анализе заменять сочетанием in- и out-параметров.

В [4] рассматривается подход, заключающийся в использовании венгерской нотации при написании кода web-приложений для обозначения того, является ли содержимое той или иной переменной потенциально опасным при подстановке в SQL-запрос. Применение данного подхода позволяет разрабатывать web-приложения, не содержащие уязвимостей к SQL-инъекциям. Однако при этом важно, чтобы программист обладал достаточной квалификацией для правильного выбора префиксов. Кроме того, при изменении кода web-приложения может возникнуть ситуация, когда должен быть выбран другой префикс, и отслеживание подобных ситуаций в больших проектах является проблематичным. Наконец, для ответа на вопрос о наличии или отсутствии уязвимостей в web-приложении, написанном без использования данного подхода, необходимо произвести переименование идентификаторов с правильным выбором префиксов, что может оказаться трудоемкой задачей. Между тем, [4] не предлагает

способов применения рассматриваемого подхода для автоматизированного обнаружения уязвимостей в web-приложениях, которые уже полностью или частично разработаны.

Предлагаемая в статье модель обнаружения уязвимостей в отличие от указанного подхода [4] основана на автоматизированном назначении оценок отдельным элементам web-приложений. В рамках модели предлагается выделить оценки двух видов – оценки для данных и оценки для параметров процедур.

Система оценок строится таким образом, чтобы ответ на вопрос о правильности обработки данных web-приложением можно было дать путем сравнения оценок in-параметров (формальных) с оценками фактически передаваемых значений. В простейшем варианте модели используется бинарная система оценки. В дальнейшем, при реализации основанных на предлагаемой модели программных средств, возможно расширение системы оценок.

Для данных (переменных, констант) предлагается использовать следующие оценки:

– оценку S (safe) получают данные, которые могут быть подставлены в текст SQL-запроса и при этом не приведут к возникновению уязвимости;

– оценку U (unsafe) получают данные, которые при подстановке в текст SQL-запроса могут изменить его логику, т.е. привести к SQL-инъекции.

Наилучшей оценкой считается оценка S, наихудшей – оценка U.

Оценка in-параметра равна наихудшей возможной оценке фактического параметра (данных), при которой его передача в процедуру не приведет к возникновению уязвимости:

– оценку S (safe) получают in-параметры, для которых передаваемое значение должно иметь оценку не ниже S, т.е. быть надлежащим образом обработано;

– оценку U (unsafe) получают in-параметры, для которых передаваемое значение должно иметь оценку не ниже U, т.е. может быть любым.

Оценки для out-параметров получаются в ходе абстрактной интерпретации кода процедуры. Значение оценки out-параметра – это значение наихудшей оценки для данных, которые процедура может возвращать через этот out-параметр. При анализе вызывающей процедуры эта оценка становится оценкой переменной, переданной в качестве out-параметра:

– оценку S (safe) получают out-параметры, через которые возвращаются данные с оценкой не ниже S;

– оценку U (unsafe) получают out-параметры, через которые возвращаются данные с оценкой не ниже U, т.е. которые следует в дальнейшем считать потенциально опасными.

Предложенная модель может быть расширена в соответствии со спецификой решаемой задачи по обнаружению уязвимостей.

Так, например, предложенная в качестве базовой бинарная система оценки может оказываться неэффективной из-за большого количества ложноположительных срабатываний. Они могут возникать в ряде случаев, например:

– web-приложение в настоящее время не имеет уязвимости, однако она может возникнуть в результате внесения изменений в код приложения;

– алгоритмы экранирования данных, созданные разработчиками web-приложения, оказываются слишком сложными для автоматического распознавания моделью обнаружения уязвимостей.

Следует понимать, что модель предполагает обнаружение не только уязвимостей, позволяющих успешно провести атаку, но и потенциальных уязвимостей, которые могут стать эксплуатируемыми при изменении кода web-приложения в ходе его дальнейшей разработки и/или сопровождения. Другими словами, наличие ложноположительных срабатываний в первую очередь говорит об общем качестве кода приложения, включая не только его надежность, но и сопровождаемость, и ряд других характеристик качества.

Тем не менее, на практике целесообразно расширить систему оценки третьим значением – UDS (user-defined safe, безопасное по определению пользователя), позволяющим пользователю программного средства обнаружения уязвимостей явно указать, что некоторые параметры процедур, получившие при автоматическом анализе с использованием модели обнаружения уязвимостей оценку U, на самом деле должны иметь оценку S.

Предлагаемая модель разработана в первую очередь для обнаружения уязвимостей к SQL-инъекциям, однако может быть адаптирована и для выявления других видов уязвимостей.

Заключение

Предложенная в статье модель обнаружения уязвимостей предоставляет возможность автоматизации трудоемкого процесса анализа исходных кодов web-приложений на наличие уязвимостей к SQL-инъекциям. Результаты, получаемые при использовании модели, могут в дальнейшем быть использованы для отслеживания динамики изменений уровня качества web-приложений на различных этапах процессов разработки и сопровождения.

A WEB-APPLICATION VULNERABILITY DETECTION MODEL

V.V. BAKHTIZIN, D.E. ONOSHKO

Abstract

A web-application vulnerability detection model based on static analysis of source codes is proposed. Evaluation system used by the vulnerability detection model is described. Several ways of extending the model for certain difficult cases are shown.

Keywords: web-application, static analysis, SQL-injection, vulnerability.

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.
2. *Фаулер М.* Архитектура корпоративных программных приложений. М., 2006.
3. *Patrick Cousot, Radhia Cousot* // Conference Record of the Fourth ACM Symposium on Principles of Programming Languages. Los Angeles, California, USA, January, 1977. P. 238–252
4. Making Wrong Code Look Wrong – Joel on Software. [Электронный ресурс]. – Режим доступа: http://www.joelonsoftware.com/articles/Wrong.html. – Дата доступа: 21.12.2014.