УДК 681.3

ПОДХОД К ТЕСТИРОВАНИЮ УЯЗВИМОСТИ WEB-ПРИЛОЖЕНИЙ ОТ АТАК ТИПА SQL-ИНЪЕКЦИЙ

Д.М. Рябко

Физико-технический учебно-научный центр НАНУ, кафедра Института кибернетики им. В.М. Глушкова
03680, Киев, проспект Глушкова, 40
тел.: (044)522 4741; e-mail: dryabko@topspin.kiev.ua

Статья посвящена проблеме тестирования уязвимости Web-приложений от атак типа SQL-инъекций. Рассматриваются методы написания тестов выявления уязвимостей Web-приложений и предлагаются подходы к организации автоматического тестирования.

This article is devoted to problem of Web application testing for SQL injection attacks. In the article are reviewed methods of writing tests to reveal Web application vulnerability and supposed approaches for organization automatic testing.

Введение

Роль Интернета значительно выросла за последнее время для среды программных продуктов. Приложения, основанные на Web-технологиях, стали востребованы в решении задач из разных отраслей и постепенно стали вытеснять приложения, основанные на других технологиях. Это привело к усложнению Web-приложений в плане структуры, архитектуры и реализации, к тому же в Web-приложениях стала использоваться распределенная архитектура. Такое усложнение выдвинуло новые требования к вопросам безопасности Web-приложений. Известно, что безопасность – это ключевой момент разработки Web-приложения и разработчики должны уделять достаточно внимания этому вопросу. Если меры безопасности будут заложены в приложение на раннем этапе разработки, то это значительно сэкономит время и средства, по сравнению с вариантом, когда безопасность будет добавляться в готовое или практически готовое приложение. Вопросы безопасности и надежности являются актуальными на всех этапах разработки. И хотя по этому поводу было написано множество документов и статей, но мало что изменилось – программисты продолжают писать программы, учитывая вопросы безопасности не в полной мере.

В данной статье предлагается подход к тестированию уязвимостей Web-приложений, основанный на методе их атаки, известной как SQL-инъекция. Рассмотрены следующие вопросы:

- варианты SQL-инъекций и способы проникновения;
- методы обеспечения безопасности от SQL-инъекций;
- методика организации автоматического тестирования;
- схема инструмента для генерации, запуска и анализа результатов тестов.

Описание SQL-инъекции как способа атаки

SQL-инъекция – это разновидность атаки для получения неавторизированного доступа к базе данных или для получения информации напрямую из базы данных. Такой вид атаки становится возможным благодаря общей грубой ошибке большинства программистов: их программы получают данные от клиента и исполняют SQL-запросы с этими данными без предварительного разбора и анализа полученных данных. Как результат, атакующий может выполнять любые операции над атакуемой базой данных. В некоторых случаях атакующий, через базу данных, может даже получить доступ к операционной системе или локальной сети, в которой находится компьютер с базой данных. SQL-инъекция работает даже в случае, когда база данных защищена файерволом (firewall), для операционной системы установлены все последние обновления и открыт только 80-й порт.

Принципы, которые лежат в основе SQL-инъекции, просты и нет ничего сложного в проведении этого вида атаки. С другой стороны, SQL-инъекция не требует знания об устройстве атакуемого приложения и может быть проведена на любой базе данных, что позволяет автоматизировать этот вид атаки. Атакующий может посылать приложению параметры, которые намеренно приведут к ошибке выполнения SQL-запроса и заставят сервер выдать сообщение об ошибке. По этим сообщениям об ошибке атакующий может собрать необходимую информацию для осуществления SQL-инъекции. Разработчик может перехватывать сообщения об ошибке, генерируемые сервером баз данных, но это не защищает приложение, а является лишь другим названием метода SQL-инъекции.

Рассмотрим пример простой SQL-инъекции. Предположим, у нас есть Web-приложение, которое предлагает получить какую-то информацию по введенному ключу:

```
http://www.somecompany.com/someInfo.jsp?someID=2
```

На стороне `someInfo.jsp` код, обрабатывающий введенные параметры, может быть следующим:

```
String query = “SELECT id, body FROM someInfo” +
“ WHERE someID = “ + request.getParameter(“someID”);
Statement stmt = dbConnection.createStatement();
ResultSet rs = stmt.executeQuery(query);
```

SQL-запрос, который Web-приложение составит и отправит на выполнение в базу данных, будет выглядеть следующим образом:

```
SELECT id, body FROM someInfo WHERE someID = 2
```

Сервер баз данных возвратит множество значений, удовлетворяющих условию `someID=2`. В дальнейшем оно будет преобразовано Web-приложением и результат отослан Web-клиенту. Чтобы определить, подвержено ли Web-приложение уязвимости SQL-инъекцией через этот параметр, попробуем вставить такое выражение в условие WHERE, которое бы имело значение true все время. Например, это можно сделать, отправив Web-приложению вот такой URL:

```
http://www.somecompany.com/someInfo.jsp?someID=2 and 1=1
```

Тогда на сервер баз данных прийдет такой SQL-запрос:

```
SELECT id, body FROM someInfo WHERE someID = 2 and 1=1
```

В случае, когда на Web-клиенте откроется страница с теми же значениями, что и в прошлый раз, то есть подозрение, что через этот параметр проникает SQL-код, и есть подозрение на SQL-инъекцию.

Методы и средства обеспечения безопасности от атак SQL-инъекций

Рассмотрим методы обеспечения безопасности от атак посредством SQL-инъекций. В работах, посвященных безопасности Web-приложений, рекомендуется, чтобы разработчики никогда не использовали введенные пользователем данные напрямую для формирования SQL-запросов к базе данных. Самым лучшим решением будет, конечно, полная изоляция Web-приложения от формирования SQL-запросов, т.е. перенос всей функциональности, связанной с их формированием, на уровень хранимых процедур на сервер баз данных. И эти хранимые процедуры следует вызывать, используя безопасные интерфейсы, например CallableStatement в JDBC или Command Object в ADO. Такой вариант решения проблемы безопасности подходит не для всех приложений, в этом случае стоит рассмотреть следующие подходы.

Использование связанных параметров. Этот метод поддерживается практически интерфейсами всех баз данных. Согласно этой технологии, SQL-запрос создается вместе с «заполнителями» (ставится знак вопроса на месте каждого параметра) и затем запрос компилируется (или «prepared»(«подготавливается») на языке SQL) во внутреннюю форму. В дальнейшем эта форма исполняется вместе с реальными параметрами. В данном случае одинарные кавычки, точка с запятой, обратная косая черта, знак комментария в SQL не будут влиять на SQL-запрос, направляемый в базу данных для исполнения, потому что введенные пользователем значения будут пониматься как данные.

Разделение и ограничение прав пользователей базы данных. Каждому пользователю базы данных следует выделить только те права, которые нужны Web-приложению для осуществления его функциональности. Если несколько Web-приложений используют базу данных для разных целей, рекомендуется определить для них отдельно пользователей с теми правами, которые необходимы каждому Web-приложению для работы. В противном случае атакующий может воспользоваться дополнительными правами пользователя базы данных.

Использование хранимых процедур. Если сервер баз данных поддерживает использование хранимых процедур, есть возможность устранить SQL полностью (в случае, если сами процедуры написаны правильно). Возможно, для простых очередей (query) использование хранимых процедур и будет выглядеть как незначительный выигрыш, но с усложнением операций преимущества хранимых процедур становятся все очевиднее.

Замечание. Можно писать хранимые процедуры, которые бы сами составляли SQL-запросы динамически, но тогда они не будут предоставлять никакой защиты от SQL-инъекций.

Обработка сообщений об ошибках. Сообщения об ошибках, которые используются разработчиком и содержат отладочную информацию, не должны показываться пользователю. Атакующий может почерпнуть очень много полезной ему информации из этих сообщений. Полная информация об ошибке полезна разработчику, для пользователя же она абсолютна бесполезна.

Подход к организации тестирования Web-приложений от атак типа SQL-инъекции

Определим конкретнее, к каким Web-приложениям будет применим данный метод тестирования.

Архитектура Web-приложений. Для начала построим цепь, по которой связываются пользователь с базой данных: пользователь заходит на страницу, вводит значения в поля ввода, после нажатия на кнопки типа Submit страница проверяет правильность данных и отправляет данные Web-серверу, который формирует SQL-запрос и передает его базе данных для проверки запроса и возврата результата скрипту, который выдает пользователю ответ. Иными словами, архитектуру Web-приложения можно представить следующим образом (рис.1.).

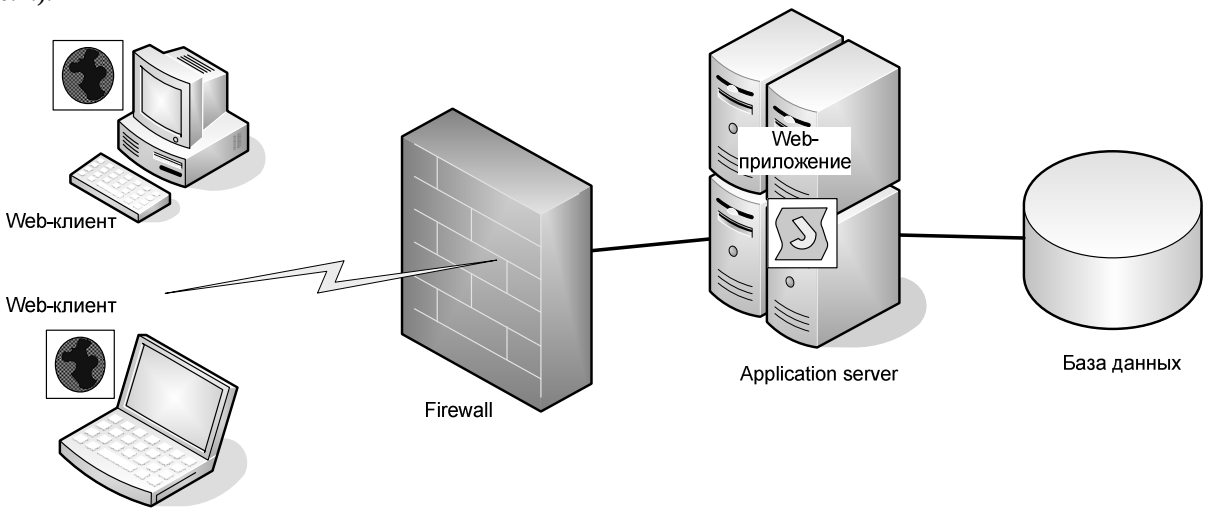

Рис. 1. Архитектура Web-приложения

Отметим, что Web-клиент – это любой клиент, который может взаимодействовать с Web или Application-сервером. Application-сервер – любой Web или Application-сервер из известных на данное время.

Критические места Web-приложения, уязвимые для SQL-инъекций. Существует несколько элементов Web-приложений, через которые возможно проникновение SQL-инъекций. Их можно классифицировать следующим образом:

- параметры в URL;
- поля input HTML-формы;
- hidden поля HTML-формы;
- cookies.

Рассмотрим каждый из этих элементов более подробно.

SQL-инъекция, осуществляемая через параметры URL. Такой вид инъекции возможен, если параметры передаются на сервер методом GET. Этот случай был рассмотрен, когда приводился пример простой SQL-инъекции.

SQL-инъекция, осуществляемая через поля input HTML формы. Этот вид инъекции возможен, если атакующий попытается ввести свой SQL-код, используя поля input на Web-странице. Механизм SQL-инъекции через поля input похож на механизм SQL-инъекции через параметры URL. Различие в том, что в первом случае данные с Web-страницы попадают Web-приложению методом POST, а во втором – методом GET. Формирование SQL-запроса может проводиться одним методом.

SQL-инъекция, осуществляемая через hidden поля HTML-формы. Такой вид SQL-инъекции возможен, если атакующий попробует изменить hidden поля в Web-формы, а затем передать эту форму на сервер для выполнения. Механизм инъекции через этот элемент похож на механизм инъекции через поля input. Различие заключается в том, что данные в hidden-поле нельзя изменить через Web-страницу напрямую, а только меняя код Web-страницы.

SQL-инъекция, осуществляемая через cookies. Этот вид возможен, если атакующий попытается изменить значения cookies, которые хранятся на машине пользователя, а затем открыть Web-страницу, которая использует эти cookies. Если разработчик не проверяет значения, полученные из cookies, то велика вероятность, что может произойти SQL-инъекция. Данные из cookies формируются в SQL-запрос Web-приложением и отправляются базе данных. Таким образом может быть сформирован не ожидаемый SQL-запрос.

Схема тестирования. Схема тестирования, приведенная на рис.2, включает в себя тестирующее приложение, приложение, создающее и запускающее тесты, и Web-приложение, которое тестируется. Если в схеме присутствует прокси (proxy), эмулирующее базу данных, то взаимодействие Web-приложения с базой данных идет через этот компонент, если прокси отсутствует – то напрямую. В первом случае прокси получает SQL-запрос от Web-приложения, оценивает и отдает серверу баз данных. Во втором случае о результате выполнения теста можно судить только по косвенным признакам: возвращаемым ошибкам; по реакции на заведомо верные и неверные данные. В обоих случаях тестирующее приложение заменяет собой Web-клиента и эмулирует его работу, т.е. реальное поведение пользователя.

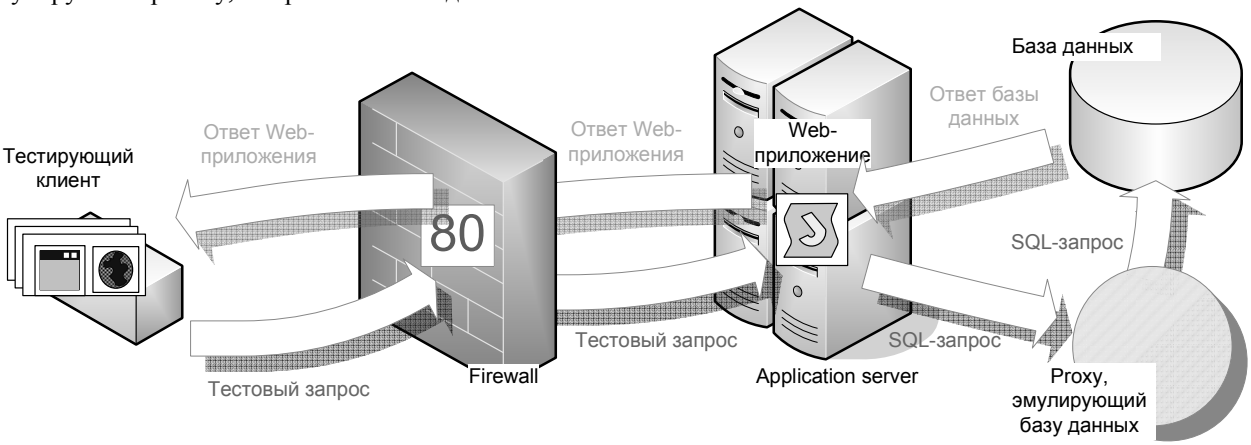

Рис.2. Схема тестирования

Методология тестирования. Представим методику создания тестов, позволяющих оценить уязвимость Web-приложения перед атакой методом SQL-инъекции.

Примечание. Во всех тестовых сценариях понимается:

- под истинным выражением – любое истинное выражение, например (1, 'a'='a', 1<>2, 3>2, 1+1, ISNULL(NULL), 2 IN (0,1,2), 2 BETWEEN 1 AND 3);
- под системными таблицами баз данных – таблицы, содержащие метаданные для тестируемой базы данных;
- под именем существующего пользователя – имя пользователя, подобранное по словарю;
- под знаком комментария в SQL – знаки (#, /**/, --).

Шаблон тестовых сценариев. Он используется для удобства описания повторяющихся элементов в определении тестовых сценариев.

Исходные данные: уникальны в каждом тестовом сценарии.

Способ тестирования: добавление к ожидаемому Web-приложением значению параметра символов, которые не ожидаются.

Алгоритм тестирования:

1. Оценка ожидаемого значения параметра.
2. Добавление набора символов, приводящих к SQL-инъекции.
3. Оценка ответов Web-приложения или сообщения об ошибке.

Набор символов, используемый для создания тестов (добавочные символы): уникальны для каждого тестового сценария.

Обработка результатов. Случай тестового инструмента с прокси (proxy) между Web-приложением и базой данных. Результаты выполнения тестов определяются из анализа, отправленного Web-приложению параметра, и параметра, который пришел из Web-приложению в прокси. Положительным результатом считается наличие в пришедшем в прокси параметре добавочных символов.

Случай тестового инструмента без прокси. В этом случае результат теста оценивается по косвенным признакам. Положительным результатом считается сообщение об ошибке базы данных, сообщение о внутренней ошибке сервера, об ошибке Web-приложения, получение от Web-приложения результатов, отличных от результатов, полученных при запросе с тем же параметром, только без добавочных символов. Если Web-приложение на тест возвращает ту же страницу, что и при запросе с параметром без добавочных символов, то этот случай требует дальнейшего исследования.

Тестовый сценарий 1. Уязвимость символьных параметров. Тест предназначен для определения степени уязвимости символьных параметров, попадающих в Web-приложение.

Исходные данные: любой символьный параметр.

Набор символов, используемый для создания тестов (добавочные символы):

1. «» (пустое значение).
2. «’» (одинарная кавычка) .
3. «’’» (две одинарных кавычки).
4. «\’’» (две одинарных кавычки, первая из которых экранирована символом бэкслеш(back slash)).
5. «--» или «#» или «/*», «*/» (знак комментария в SQL).

6. «”» (двойная кавычка).
7. «’ or ‘1’=’1» (комбинация одинарной кавычки с истинным выражением).
8. «” or “1”=”1» (комбинация двойной кавычки с истинным выражением).
9. «') or ('1'='1» (комбинация одинарной кавычки со скобочным выражением).
10. Составление тестового параметра по формуле
<тестовый параметр>:=«; SELECT 0{,0}(возрастающее с каждой итерацией количество параметров)> FROM { системные таблицы различных баз данных }».
11. Составление тестового параметра по формуле
<тестовый параметр>:=«UNION SELECT 0{,0}(возрастающее с каждой итерацией количество параметров)> FROM { системные таблицы различных баз данных }<знак комментария в SQL>».
12. Вставка «/**/» на месте пробелов в предыдущих случаях.
13. Вставка «+»или ASCII-кода этого символа на месте пробелов в предыдущих случаях.
14. Рассмотренные выше наборы символов с произвольной комбинацией символов и ASCII-кодов этих символов.

Тестовый сценарий 2. Уязвимость числовых параметров. Тест предназначен для определения уязвимости числовых параметров, попадающих в Web-приложение.

Исходные данные: любой численный параметр.

Набор символов, используемый для создания тестов (добавочные символы):

1. «» (пустое значение).
2. « or 1=1» (пробел и истинное выражение).
3. «--» или «#» или «/*», «*/» (знак комментария в SQL).
4. Рассмотренные выше наборы символов с произвольной комбинацией символов и ASCII кодов этих символов.
5. Вставка «/**/» на месте пробелов в предыдущих случаях.
6. Составление тестового параметра аналогично случаю 9 в тестовом сценарии 1.
7. Составление тестового параметра аналогично случаю 10 в тестовом сценарии 1.
8. Вставка «+»или ASCII кода этого символа на месте пробелов в предыдущих случаях.
9. Составление тестового параметра по формуле

<тестовый параметр> := <правильный параметр+1>-1 .

Тестовый сценарий 3. Уязвимость полей поиска. Тест предназначен для определения уязвимости полей поиска, предоставляющих возможность поиска данных в базе данных.

Исходные данные: параметр, передаваемый в поле поиска.

Набор символов, используемый для создания тестов (добавочные символы):

1. «» (пустое значение).
2. «’#» (одинарная кавычка и знак комментария в SQL).
3. «%» (знак произвольного выражения при поиске совпадений оператором LIKE в SQL).
4. «%’#».
5. «’/*» и «*/» для параметра sort (ссортировка), если есть возможность устанавливать значение параметру sort.

Тестовый сценарий 4. Уязвимость полей авторизации. Тест предназначен для определения уязвимости полей авторизации.

Исходные данные: параметры, передающиеся в поля авторизации.

Способ тестирования: добавление к имени пользователя(login) и паролю(password) неожидаемых данных.

Набор символов, используемый для создания тестов (добавочные символы):

1. «<имя существующего пользователя>’ or ‘1’=’1» (имя существующего пользователя с истинным выражением).
2. «<имя существующего пользователя>’#» (имя существующего пользователя и знак комментария в SQL).
3. Подстановка вышеприведенных значений в поле пароля и комбинация этих значений в поле логина и пароля.
4. «%» в поле пароля.

Тестовый сценарий 5. Уязвимость полей регистрации. Тест предназначен для определения уязвимости полей, используемых для регистрации пользователей.

Исходные данные: параметры, передающиеся в поля регистрации.

Способ тестирования: добавление к регистрационным данным неожидаемых символов.

Алгоритм тестирования:

Первые три пункта, как в шаблоне, и четвертый пункт:

4. Попытка авторизироваться пользователем, зарегистрированным с верными параметрами, а затем пользователем с тестовыми параметрами.

Набор символов, используемый для создания тестов (добавочные символы):

составление тестового параметра по формуле

<тестовый параметр>:={, <значения, подобранные эвристическим образом>})<знак комментария в SQL>.

Примечание. В данной формуле заложено предположение, что следование сохраняемых параметров в SQL-запросе, такое же, как и порядок полей на форме. Значения в {} скобках повторяются столько раз, сколько полей, начиная от поля, принимающего текущий параметр, до конца списка полей. Подбор эвристическим образом означает, что тип параметров оценивается эвристическим образом.

Организация и проведение тестирования

Схемы тестирования состоит из следующих шагов (рис.3.):

- оценка тестируемого приложения;
- создание базовой среды для тестирования;
- поиск критических Web-страниц;
- определение критических значений для параметров и их типов;
- подготовка и запуск тестов для найденных параметров;
- оценка результатов тестов и фиксация уязвимостей;
- составление итогового отчета.

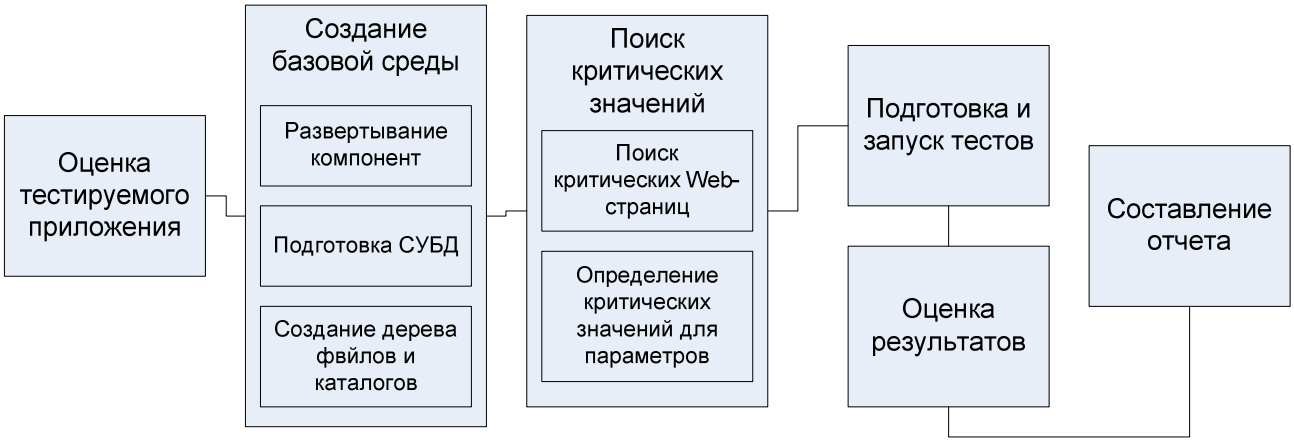

Рис. 3. Диаграмма работы схемы тестирования

Оценка тестируемого приложения. На данном этапе проводится оценка таких особенностей тестируемого Web-приложения, как операционная среда, Web-сервер, СУБД, на которых работает Web-приложение и язык программирования, на котором оно написано.

Создание базовой среды для тестирования. На этом этапе выполняется развертывание компонентов тестовой среды, подготовка СУБД в зависимости от особенностей Web-приложения, выявленных на предыдущем этапе. Затем создается дерево файлов и каталогов, т.е. определяется внутренняя структура Web-приложения.

Поиск критических Web-страниц. Этот шаг используется для поиска, анализа и выявления Web-страниц, которые могут содержать параметры, критические по отношению к SQL-инъекции.

Определение критических значений для параметров. На этом этапе осуществляется анализ и определение ожидаемых параметров и их типов для выявленных Web-страниц.

Подготовка и запуск тестов. Используя ожидаемые параметры, согласно методологии тестирования, происходит создание и запуск тестов для выявленных критических параметров.

Оценка результатов и фикция уязвимостей. После каждого запущенного теста происходит оценка результатов тестирования и найденные уязвимости заносятся в журнал.

Составление итогового отчета. Анализируя журнал тестирования, происходит составление итогового отчета о тестировании.

Выводы

Данная статья является итогом исследования определения уязвимостей Web-приложений перед атаками типа SQL-инъекций. Рассмотрен подход к тестированию Web-приложений, особенности атак типа SQL-инъекций, предложена методика организации тестирования. Приведены отдельные шаги в схеме тестирования и особенности их выполнения. Представленная методология и схема тестирования могут быть реализованы разработчиками для выявления и устранения уязвимостей при тестировании Web-приложений.

Автор использовал эту методологию при разработке Web-приложений. Результаты применения методологии позволили повысить надежность данных приложений.

1. *Kevin Spett.* Blind SQL Injection.—SPI Dynamics.—2005. – 11 p. – URL: http://www.spidynamics.com/whitepapers/Blind_SQLInjection.pdf
2. *Stephen Kost.* An Introduction to SQL Injection Attacks for Oracle Developers. – Integrigy. – 2004. – 24 p. – URL: http://www.integrigy.com/info/IntegrigyRethinkingSQLInjectionAttacks.pdf
3. *Ofer Maor, Amichai Shulman.* SQL Injection Signatures Evasion. –Imperva. – 2004. – 16 p. – URL: http://www.imperva.com/application_defense_center/white_papers/sql_injection_signatures_evasion.html
4. *Ofer Maor, Amichai Shulman*. Blindfolded SQL Injection. – Imperva. – 2003. – 16 p. – URL: http://www.webcohort.com/Blindfolded_SQL_Injection.pdf
5. *Cesar Cerrudo*. Manipulating Microsoft SQL Server Using SQL Injection. – Application Security, Inc. – 2003. – 14 p. – URL: http://www.appsecinc.com/presentations/Manipulating_SQL_Server_Using_SQL_Injection.pdf
6. *Chris Anley*. Advanced SQL Injection In SQL Server Applications. – Next Generation Security Software Ltd. – 2002. – 25 p. – URL: http://www.nextgenss.com/papers/advanced_sql_injection.pdf