УДК 004.052.42:004.056.53

МЕТОД ОБНАРУЖЕНИЯ УЯЗВИМОСТЕЙ К SQL-ИНЪЕКЦИЯМ В WEB-ПРИЛОЖЕНИЯХ

Д.Е. Оношко, В.В. Бахтизин

Рассмотрены основные свойства уязвимостей к SQL-инъекциям в web-приложениях. Предложена модель обнаружения уязвимостей, основанная на статическом анализе исходных кодов. Предлагается метод, основанный на модели обнаружения уязвимостей, позволяющий автоматизировать поиск уязвимостей и предоставляющий исходные данные для оценки качества web-приложений.

Введение

По данным Открытого проекта обеспечения безопасности web-приложений (OWASP) по состоянию на 2013 год наиболее распространённым семейством угроз для web-приложений являются инъекции кода и, в частности, SQL-инъекции [1]. Возможные последствия эксплуатации уязвимостей web-приложений к подобным атакам варьируются в зависимости от назначения уязвимой части web-приложения, специфики обрабатываемых приложением данных, а также целей злоумышленника.

Особую значимость данной проблеме придаёт тот факт, что задачи, решаемые с использованием web-приложений, как правило, предполагают возможность доступа к ним посредством сети Internet из любой точки мира, а значит, имеет место повышенная доступность web-приложения по сравнению с классическими desktop-приложениями, в том числе для злоумышленника. При этом следует отметить, что уязвимость к SQL-инъекции может относиться в том числе и к части кода web-приложения, отвечающей за аутентификацию и авторизацию пользователей, что зачастую упрощает проведение атаки.

По названным причинам своевременное обнаружение уязвимостей к инъекциям кода и, в особенности, SQL-инъекциям, представляет собой одну из важнейших задач при обеспечении качества web-приложений. При этом постоянный рост сложности web-приложений создаёт потребность в автоматизации обнаружения таких уязвимостей, а значит, и разработке методов автоматизированного обнаружения уязвимостей.

Требования к методу обнаружения уязвимостей

Результаты изучения проблемы SQL-инъекций (а также других видов инъекций кода) говорят о том, что причиной их возникновения в коде web-приложений являются допущенные разработчиками ошибки в обработке данных, поступающих от пользователя. Такие ошибки позволяют путём подачи на вход web-приложения специальным образом сформированных данных привести к изменению логики его работы, в частности, в случае SQL-инъекций — к изменению логики запросов к системе управления базами данных (СУБД).

Наиболее распространённым техническим решением, позволяющим избежать уязвимости web-приложения к SQL-инъекциям, является использование так называемых *подготовленных выражений* (prepared statements). Несмотря на гарантированное отсутствие уязвимостей к SQL-инъекциям при правильном использовании этого средства проблема обнаружения уязвимостей по-прежнему остаётся актуальной, поскольку даже при использовании выражений происходит формирование запроса к СУБД. В случае если при формировании запроса была допущена подстановка данных, поступивших с запросом пользователя, без надлежащей фильтрации, механизм подготовленных выражений не сможет предотвратить SQL-инъекции. Другие технические решения, часто используемые для предотвращения уязвимостей (например, *хранимые процедуры* или *автоматическое экранирование*), показывают ещё более низкую эффективность.

К настоящему времени предприняты многочисленные попытки классификации существующих методов обнаружения уязвимостей к SQL-инъекциям [2]. В общем случае все эти методы, в зависимости от положенного в основу принципа анализа web-приложения, как показано на рис. 1, разделяют на 3 группы: *статические*, *динамические* и *смешанные*. По ряду причин наибольшей популярностью пользуются динамические методы, которые в свою очередь условно можно разделить на *внутренние*, предполагающие внесение в код web-приложения изменений, и *внешние*, основанные на внедрении программных модулей без модификации исходных кодов самого web-приложения.

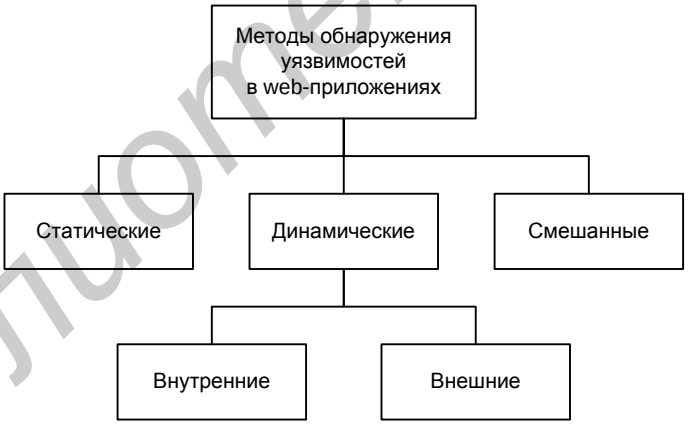

Рис. 1. Классификация методов обнаружения уязвимостей в web-приложениях

В зависимости от того, на каком этапе происходит анализ данных web-приложения, все динамические методы также можно разделить на три группы:

– динамические методы с предобработкой — анализируют данные, поступающие от пользователя на наличие в них потенциально опасных последовательностей символов;

– динамические методы с постобработкой — анализируют данные, поступающие от web-приложения к СУБД на соответствие определённым требованиям к их структуре;

– динамические методы с двойной обработкой — совмещают в себе признаки первых двух групп.

Поскольку уязвимости к SQL-инъекциям представляют собой ошибки в логике работы web-приложения, а корректной признаётся логика работы программных средств (ПС) и отдельных составляющих их модулей, задаваемая спецификациями требований к ним, для обнаружения подобных ошибок информации, которую можно получить из исходных кодов или результатов работы web-приложения недостаточно. Например, непосредственная передача полученных от пользователя строковых данных в качестве запроса к СУБД является ошибкой в обработке данных для большинства web-приложений, однако может являться корректным поведением для web-приложений, предназначенных для управления СУБД. По этой причине методы, используемые для обнаружения уязвимостей к SQL-инъекциям, могут давать так называемые *ложноположительные* и *ложноотрицательные* результаты.

Таким образом, важным свойством метода обнаружения уязвимостей является относительное количество ложных результатов, получаемых при его использовании. При этом, поскольку в данном случае речь идёт о выявлении ошибок, наличие ложноположительных результатов является приемлемым (обнаружена ошибка, не являющаяся таковой), в то время как наличие ложноотрицательных результатов крайне нежелательно (ошибки присутствуют, но не обнаруживаются методом).

Из приведённых рассуждений следует, что наилучшими характеристиками обнаружения обладают статические методы. Динамические методы по своей сути представляют собой тестирование web-приложения путём передачи ему конкретных значений исходных данных, а значит, доказать отсутствие уязвимостей при использовании динамических методов обнаружения возможно только путём исчерпывающего тестирования, которое не производится ввиду потенциально бесконечного количества тестовых данных. При этом корректность полученных результатов для всех тестовых данных не гарантирует отсутствия потенциальных проблем, а обнаружение уязвимости во время работы web-приложения позволяет в лучшем случае предотвратить атаку, при этом выполнение web-приложением соответствующей функции при определённых исходных данных становится невозможным. Между тем статические методы потенциально могут обходиться без ложноотрицательных результатов.

Поскольку уязвимости к SQL-инъекциям являются ошибками в логике web-приложения, их наличие желательно отслеживать на всех этапах жизненного цикла начиная с момента появления первого прототипа с целью своевременного принятия управленческих решений (например, о необходимости повышения квалификации программистов или корректировке сроков завершения проекта). Следовательно, метод обнаружения уязвимостей должен предусматривать не только обнаружение реальных и потенциальных проблем, но и предоставлять возможность получения числовых оценок качества web-приложения с точки зрения уязвимости к SQL-инъекциям.

Модель web-приложения в контексте обнаружения уязвимостей

В рамках задачи обнаружения уязвимостей целесообразно рассматривать web-приложение как автомат Мили, как показано на рис. 2.

Ответ web-приложения на поступивший от пользователя запрос определяется данными, поданными на вход, а также внутренним состоянием web-приложения, которое представлено *хранилищем данных* — комплексом программных и аппаратных средств, используемых для сохранения данных между запросами пользователя. В зависимости от особенностей web-приложения в качестве хранилища данных могут выступать базы данных, файлы, сессии и т.п.

Важно заметить, что внутреннее состояние отделено от самого web-приложения, а доступ к нему осуществляется посредством программных интерфейсов (API), предоставляемых хранилищем данных. Вследствие этого непосредственно код web-приложения не содержит в себе запоминающих элементов, что позволяет существенно упростить проверку его корректности.

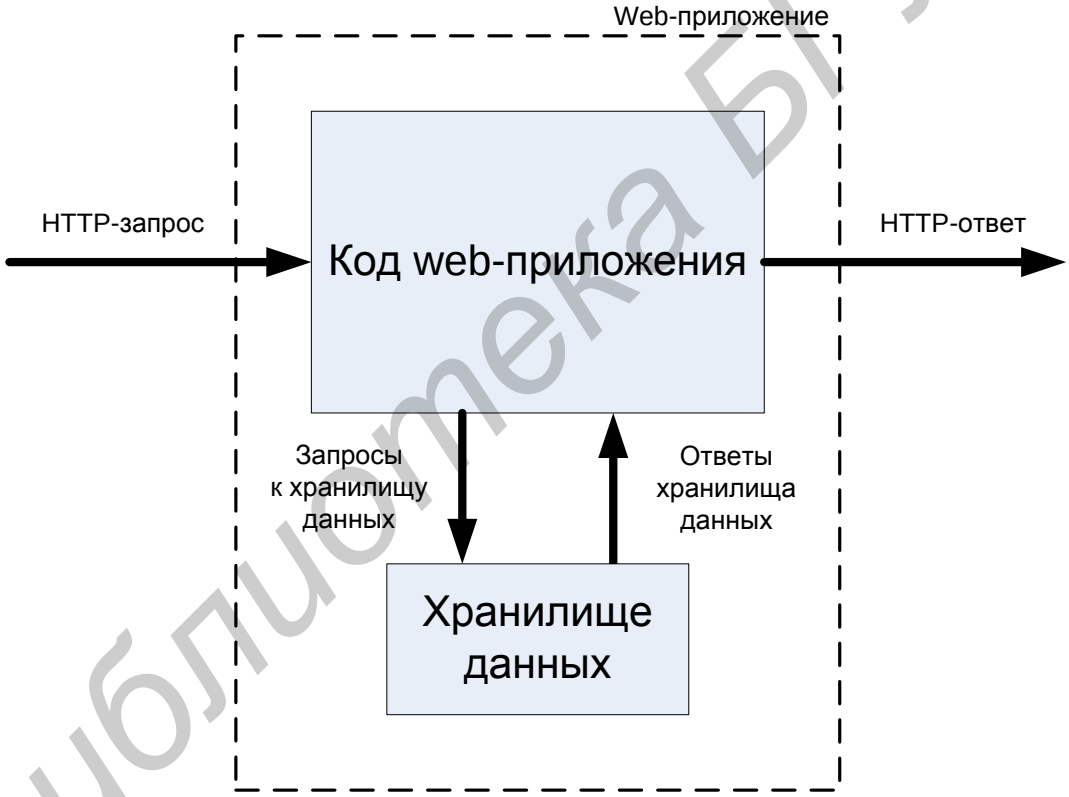

Рис. 2. Структура web-приложения в контексте обнаружения уязвимостей

Фактически код web-приложения представляет собой сложную функцию, аргументами которой являются данные, поступающие с запросом пользователя, и данные, получаемые из хранилища данных, а результатами — запросы к хранилищу данных и ответ на запрос пользователя. При этом уязвимости к SQL-инъекциям возникают при наличии ошибок именно в этой части web-приложения.

Следует также заметить, что при поиске уязвимостей к SQL-инъекциям наибольший интерес представляет та часть кода web-приложения, через которую данные, полученные от пользователя (и, возможно, производные от

них данные), могут попадать в места вызова функций, составляющих API СУБД. Это потенциально позволяет сократить объём анализируемого кода.

Модель обнаружения уязвимостей в web-приложениях

Для проверки web-приложения на наличие в нём уязвимостей к SQL-инъекциям целесообразна дальнейшая декомпозиция web-приложения.

Минимальным элементом, который предлагается использовать для анализа исходных кодов web-приложения, является *процедура*. В контексте автоматизированного обнаружения уязвимостей данное понятие целесообразно обобщить, поскольку в зависимости от используемого разработчиками web-приложения языка программирования ими могут применяться различные языковые средства, имеющие схожее назначение. В дальнейшем под процедурой будет пониматься некоторая часть кода web-приложения, для которой можно выделить входные и выходные данные.

Взаимодействия между процедурами могут быть описаны как обмен данными посредством параметров процедур. При этом можно выделить параметры двух типов:

– *in-параметры* — параметры, посредством которых процедура получает данные извне;

– *out-параметры* — параметры, посредством которых процедура передаёт данные (результаты своей работы) внешнему коду.

Обнаружение уязвимостей предлагается осуществлять посредством абстрактной интерпретации исходных кодов с учётом специфики SQL-инъекций. Для этого предлагается назначать отдельным элементам программы оценки, отражающие допустимость использования данных, соответствующих этим элементам, при построении запросов к СУБД.

В простейшем случае такая система оценок является бинарной:

– Оценка «U» (unsafe) используется для обозначения элементов, использование которых при подстановке в запрос к СУБД потенциально может приводить к эксплуатации уязвимости.

– Оценка «S» (safe) используется для обозначения элементов, использование которых при подстановке в запрос к СУБД гарантированно не приводит к эксплуатации уязвимости.

При такой системе оценок наихудшей (минимальной) считается оценка «U», наилучшей (максимальной) — оценка «S».

Оценки назначаются in- и out-параметрам процедур, а также переменным и константам, которые используются для хранения данных, передаваемых в качестве параметров.

Оценка для in-параметра определяется как наихудшая оценка фактически передаваемых через этот параметр данных, при которой не возникает уязвимости, т.е.:

– оценка «S» назначается in-параметрам, которые могут принимать на вход процедуры только данные с оценкой не ниже «S», т.е. предварительно обработанные надлежащим образом;

– оценка «U» назначается in-параметрам, которые могут принимать на вход процедуры данные с оценкой не ниже «U», т.е. любые данные.

Оценка для out-параметра определяется как наихудшая оценка фактически возвращаемых через этот параметр данных, т.е.:

– оценка «S» назначается out-параметрам, через которые процедура всегда возвращает надлежащим образом обработанные данные;

– оценка «U» назначается out-параметрам, через которые процедура может передавать не обработанные надлежащим образом данные.

В терминах in- и out-параметров могут быть описаны свойства не только таких возможностей современных языков программирования, используемых при разработке web-приложений, как процедуры, функции и методы, но и операторов языка: операнды в этом случае соответствуют in-параметрам, результат выполнения операции — out-параметру.

Оценки для in- и out-параметров стандартных процедур (в т.ч. относящихся к API СУБД, соответствующих операторам языка и т.п.) считаются известными заранее и назначаются вручную. При использовании для автоматизации обнаружения уязвимостей специализированного ПС анализа исходных кодов эта информация может включаться в состав ПС.

Оценки для процедур, написанных разработчиками web-приложения, могут быть вычислены путём абстрактной интерпретации исходных кодов этих процедур.

Из вышеприведённых принципов назначения оценок следует основное правило, которое должно соблюдаться для того, чтобы web-приложение не содержало уязвимостей: *оценка данных, которые фактически передаются в качестве любого in-параметра в любую из процедур, должна быть* **не хуже,** *чем оценка соответствующего in-параметра*.

Предложенное понятие обобщённой процедуры можно соотнести с понятием функционального блока в теории структурного программирования. В соответствии с принципом Бёма-Якопини [3] любая программа, состоящая из подобных функциональных блоков, может быть описана в терминах трёх базовых конструкций:

– конструкции следования;

– конструкции выбора;

– конструкции итерации.

Вызов процедуры с известными оценками параметров может рассматриваться в качестве функционального блока. Таким образом, любая процедура web-приложения может быть декомпозирована по правилам Бема-Якопини до отдельных таких вызовов.

Анализ кода процедуры должен начинаться с выявления идентификаторов, доступных в области видимости данной процедуры — переменных (в том числе созданных автоматически для in-параметров) и констант.

Поскольку константы являются значениями, которые явным образом записаны в исходном коде web-приложения разработчиками, и, кроме того,

могут использоваться для записи шаблонов SQL-запросов к СУБД, по умолчанию их оценку следует устанавливать равной «S».

Последовательный анализ вызовов, выполняемых процедурой, позволяет определить оценки для её in- и out-параметров. При этом:

- оценка in-параметра анализируемой процедуры определяется как наилучшая из оценок, которые имеют in-параметры вызываемых ею процедур, принимающие данный in-параметр;
- оценка out-параметра анализируемой процедуры определяется как наихудшая из оценок, которые имеют out-параметры вызываемых ею процедур, помещающие результаты своей работы в данный out-параметр.

В ходе анализа web-приложения на наличие уязвимостей к SQL-инъекциям оно рассматривается как множество (в том числе стандартных) обобщённых процедур $Q = \{P_1, P_2, \ldots, P_N\}$, где N — общее количество процедур в web-приложении. Пусть на i-м шаге анализа известны оценки для параметров процедур $Q_i = \{P_1, P_2, \ldots, P_{C_i}\}$, $Q_i \subseteq Q$, где $C_i \leq N$ — количество таких процедур (зависит от количества стандартных процедур с заранее известными оценками). Тогда в соответствии с принципами восходящего проектирования, поскольку все процедуры из множества Q принадлежат одному и тому же web-приложению, существует процедура P_{C_i+1}, зависящая только от процедур из множества Q_i. Анализируя исходный код процедуры P_{C_i+1}, можно получить оценки для её параметров.

Последней процедурой, которая будет подвергнута анализу, является процедура P_N, которая соответствует *точке входа* web-приложения — блоку с наименьшей вложенностью. In-параметрам этой процедуры будут соответствовать данные, поступающие от пользователя. Поскольку все они в момент передачи P_N не прошли надлежащей обработки, в соответствии с вышеприведённым основным правилом в результате анализа исходных кодов web-приложения все оценки in-параметров процедуры P_N должны быть не выше «U». В противном случае web-приложение содержит уязвимости (возможно, потенциальные), причём для их эксплуатации могут быть использованы in-параметры, имеющие оценки выше «U».

Адаптация метода к автоматизации анализа web-приложений

Приведённая выше бинарная система оценки при применении на практике оказывается недостаточно эффективной и требует дальнейшего расширения.

Наиболее существенная проблема двухбалльной шкалы заключается в высокой вероятности получения **ложноположительных** результатов. Это происходит, например, в тех случаях, когда в web-приложении осуществляется надлежащая обработка данных, поступающих от пользователя, но её реализация не распознаётся предложенным методом обнаружения уязвимостей. Это может происходить, поскольку предлагаемый метод не выполняет

детального анализа диапазонов значений, которые могут принимать переменные в web-приложении.

Примером такой обработки может быть экранирование строковых данных, полученных от пользователя, вручную, т.е. собственная реализация функции наподобие mysql_escape_string() или mysql_real_escape_string(). Несмотря на низкую эффективность данного подхода [4] подобный алгоритм действительно может обеспечивать достаточную обработку данных для предотвращения SQL-инъекций.

Изменения в составе и назначении специальных символов при развитии диалекта языка SQL, используемого СУБД, могут привести к тому, что такая обработка окажется недостаточной, поэтому данное ложноположительное срабатывание метода в данном случае является скорее желательным поведением. Тем не менее, в некоторых случаях (например, при анализе web-приложения с большой долей унаследованного или стороннего кода) необходима возможность независимо от оценок, получаемых in-параметрами такой процедуры в рамках предлагаемого метода, назначить им оценку «S» по результатам ручного анализа.

Поскольку оценки, назначенные параметрам вручную, могут изменяться при изменении исходных кодов соответствующей процедуры, в подобных случаях при программной реализации предлагаемого метода в виде ПС анализа исходных кодов необходимо использование специального значения оценки — «UDS» (user-defined safe). При этом в ПС анализа исходных кодов потребуется поддержка двух режимов анализа: *обычного* и *параноидального*. В обычном режиме оценка «UDS» эквивалентна оценке «S», в параноидальном — игнорируется.

Ещё одной проблемой использования предлагаемого метода обнаружения уязвимостей к SQL-инъекциям является возможное использование анализируемым web-приложением неинициализированных переменных. Несмотря на то, что сами по себе значения неинициализированных переменных, как правило, оказываются псевдослучайными, а в некоторых языках программирования инициализируются нулевыми значениями, в зависимости от особенностей конкретной программной реализации ПС анализа исходных кодов наличие такого поведения в анализируемом web-приложении может создавать проблемы для назначения оценок.

В данной ситуации рекомендуется введение дополнительного особого значения «–» для оценки значений неинициализированных переменных. При использовании такого подхода результаты вызова (out-параметры) любой процедуры с передачей в качестве in-параметра переменной с такой оценкой должны также получать оценку «–». Кроме того, допускается выдача сообщения об ошибке с указанием причины, поскольку в ряде случаев использование неинициализированных переменных может становиться основой для проведения атак других семейств [5, 6].

Использование метода в обеспечении качества web-приложений

Предлагаемый метод основан на статическом анализе исходных кодов web-приложения. В результате его применения может быть собрано значительное количество информации о структуре web-приложения, характере взаимосвязей между отдельными его частями и т.п. Подобная информация может использоваться для оценки и обеспечения качества web-приложения на всех этапах жизненного цикла начиная с момента появления первого прототипа.

Наиболее полезными числовыми оценок качества web-приложения, которые могут быть получены в результате применения метода, являются те из них, которые позволяют оценить общее качество кода. Так, например, близкая к 100% доля in-параметров с оценкой «U» среди in-параметров процедур, написанных разработчиками, свидетельствует о значительном запасе *устойчивости web-приложения к изменениям* в исходных кодах с точки зрения уязвимости к SQL-инъекциям. Другими словами, при высоком значении этого показателя повышается вероятность того, что в случае внесения в код web-приложения ошибок обработки данных, полученных от пользователя, в web-приложении не появится эксплуатируемых уязвимостей за счёт того, что внесённая ошибка будет «замаскирована» избыточной обработкой поступающих извне данных.

Кроме того, в зависимости от особенностей реализации предложенного метода возможно обнаружение не только факта наличия уязвимостей в web-приложении, но и, с достаточно высокой точностью, пути, по которому полученные от пользователя данные могут попасть в запрос к СУБД без надлежащей обработки. Большое количество таких путей, а также большое количество нарушений основного правила соответствия оценок, приведённого выше, может свидетельствовать о необходимости повышения квалификации разработчиков.

Получая подобные оценки для различных промежуточных версий web-приложения, можно также отслеживать динамику изменения его качества в контексте уязвимости к SQL-инъекциям. Эти данные в дальнейшем могут использоваться для более точной оценки сроков завершения работы над проектом.

Заключение

Предлагаемый метод обнаружения уязвимостей в web-приложениях, основанный на статическом анализе исходных кодов путём абстрактной интерпретации, может быть использован при реализации ПС анализа исходных кодов для автоматизации контроля web-приложения на наличие уязвимостей к SQL-инъекциям. Важным достоинством метода является отсутствие ложноотрицательных результатов и широкие возможности по снижению количества ложноположительных срабатываний за счёт расширения предлагаемой системы оценок данных и параметров процедур анализируемого web-приложения.

Собираемые в ходе применения метода сведения об исходных кодах web-приложения могут быть использованы для оценки и обеспечения его качества.

Список литературы

1. OWASP Top 10-2013. The Ten Most Critical Web Application Security Risks. [Электронный ресурс] / Google Code Archive. — Режим доступа: http://owasptop10.googlecode.com/files/OWASP%20Top%2010%20-%202013.pdf. — Дата доступа: 04.03.2016.
2. Shakya, A. A Taxonomy of SQL Injection Defense Techniques / A. Shakya, D. Aryal — Master's Thesis, School of Computing, Blekinge Institute of Technology. — 2011. — 134 p.
3. Böhm, C. Flow Diagrams, Turing Machines And Languages With Only Two Formation Rules / C. Böhm, G. Jacopini // Communications of the ACM. — 1966. — №9. — С. 366–371.
4. SQL-инъекции [Электронный ресурс] / PHP: Hypertext Preprocessor. — Режим доступа: http://php.net/manual/ru/security.database.sql-injection.php. — Дата доступа: 21.03.2016.
5. Uninitialized Variable [Электронный ресурс] / OWASP. — Режим доступа: https://www.owasp.org/index.php/Uninitialized_Variable. — Дата доступа: 21.03.2016.
6. Основы [Электронный ресурс] / PHP: Hypertext Preprocessor. — Режим доступа: http://php.net/manual/ru/language.variables.basics.php. — Дата доступа: 21.03.2016.

Оношко Дмитрий Евгеньевич, ассистент кафедры программного обеспечения информационных технологий Белорусского государственного университета информатики и радиоэлектроники, магистр технических наук, onoshko@bsuir.by

Бахтизин Вячеслав Вениаминович, профессор кафедры программного обеспечения информационных технологий Белорусского государственного университета информатики и радиоэлектроники, кандидат технических наук, доцент, bww@bsuir.by