анализу на оптимальность работы в условиях многопоточной среды.

Обработка пакетов в системе банковских расчетов может быть представлена в виде последовательности отдельных операций:

считывание пакетов из очереди;
анализ содержимого пакета;
построение входной модели формата;
разбиение пакета на транзакции;
построение выходной модели формата;
отправка пакета получателю.

В классической модели все эти операции выполняются последовательно для каждого пакета. К достоинствам этой модели относится ее простота, поскольку все этапы выполняются один за одним и не требуют никакой дополнительной синхронизации. Недостаток данной модели: она практически не масштабируема, что проявляется в невозможности использования инструментов многопоточной среды.

Первый этап на пути модернизации модели системы банковских расчетов - внедрение параллельной обработки нескольких пакетов. Вместо ожидания в очереди пакеты будут параллельно обрабатываться при наличии свободных ресурсов.

Современные языки программирования предоставляют богатый набор средств для работы в многопоточной среде. Рассматриваемая в докладе система банковских платежей разработана на языке JAVA SE 6. В пятой версии языка был добавлен специальный «Executor Framework». Он предоставляет удобный интерфейс для использования многопоточного программирования, позволяя абстрагироваться от низкоуровневого понятия «Поток» (анг. «Thread»), представляющего собой класс для выполнения операций в отдельном потоке процессора.

Введение многопоточной обработки неизбежно приведет к усложнению работы системы и возможному появлению ошибок, которые связаны с неправильным использованием синхронизации данных. Такие ошибки считаются одними из самых сложных в обнаружении, т. к. они могут не проявляться на протяжении длительного периода времени до наступления каких-либо критических условий. Однако нельзя недооценивать всех тех преимуществ, которые дает параллельная обработка (время выполнения, возможность масштабирования).

Анализируя результаты параллельной обработки пакетов, можно прийти к выводам, что распараллеливание всего процесса обработки является не самым лучшим решением.

Параллельное выполнение каждого этапа обработки пакета по отдельности является предпочтительным решением, поскольку позволит распределять вычислительные ресурсы системы более рационально. Такое решение будет эффективным, если этапы обработки пакета выполняются за разные промежутки времени. Однако такое решение требует введения нового понятия «задача». Она будет предоставлять отдельный этап работы по обработке пакета, который может быть выполнен независимо. Ресурсы системы можно будет переключать на те задачи, которые требует немедленного исполнения или уменьшать ресурсы для тех задач, которые выполняются быстро.

Анализируя преимущества параллельного выполнения каждого этапа обработки пакета по отдельности, можно прийти к выводу, что система становится масштабируемой и легко настраиваемой под нужды конкретной ситуации, а также более производительной. Сложности, связанные с этим подходом, проявляются в том, что нужны дополнительный модуль мониторинга исполнения всех задач и контроль целостности обработки пакета. Использование модели параллельного выполнения каждого этапа обработки пакета по отдельности является оптимальным решением для высоконагруженной системы банковских платежей, т. к. производительность является одним из ключевых показателей для этой системы. Использование многопоточности сопряжено с возможными ошибками в синхронизации объектов, что требует дополнительных средств по контролю качества программного продукта.

Список использованных источников:

1. Васильев, А. Н. Java.. Объектно-ориентированное программирование / А. Н. Васильев. – СПб : Питер, 2011.- 400 с.
2. Колесов, Ю. Б. Объектно-ориентированное моделирование сложных динамических систем / Ю. Б. Колесов. – СПб : Изд-во СПбГПУ, 2004

АВТОМАТИЗАЦИЯ СТАТИЧЕСКОГО АНАЛИЗА ИСХОДНЫХ КОДОВ ПРИ ОЦЕНКЕ НАДЕЖНОСТИ WEB-ПРИЛОЖЕНИЙ

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

Оношко Д.Е.

Бахтизин В.В. — к.т.н., профессор

Наблюдающаяся в настоящее время тенденция к переходу от классических desktop-приложений к web-приложениям сопровождается ростом значимости вопросов качества разрабатываемого программного обеспечения

(ПО), в том числе — надёжности и безопасности.

По данным Открытого проекта обеспечения безопасности web-приложений (OWASP) в 2013 году наиболее распространённой угрозой для различных типов приложений (включая web-приложения) являются SQL-инъекции [1]. Анализ их свойств показывает, что причиной уязвимости web-приложений к SQL-инъекциям является наличие в них ошибок, вследствие которых полученные от пользователя данные полностью или частично подставляются в текст запроса к системе управления базами данных (СУБД) без необходимой фильтрации.

Несмотря на высокую популярность методов обнаружения подобных 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 ошибок, способных открыть возможность проведения SQL-инъекции, остаётся полный 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.

Анализируемое web-приложение рассматривается как множество $P = \{P_1, P_2, \ldots, P_N\}$ процедур (в т.ч. операторов языка), вызывающих друг друга с некоторыми параметрами. Формальным параметрам и возвращаемым значениям процедур назначаются оценки, в простейшей реализации имеющие бинарный характер: «опасный» или «безопасный».

Первоначально ПС известны оценки только для операторов языка программирования и некоторых стандартных процедур взаимодействия с СУБД. Пусть на i-м шаге известны оценки параметров и возвращаемых значений для 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 оценки 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. Ряд других числовых характеристик, получаемых в ходе поиска потенциальных уязвимостей по предлагаемому алгоритму, может использоваться также для оценки других свойств разрабатываемого web-приложения, например, сложности.

Оценка надёжности разрабатываемых программных средств на различных этапах процессов разработки и последующего сопровождения позволяет отслеживать динамику изменения данной характеристики качества web-приложения и может выступать в качестве одного из критериев для принятия управленческих проектных решений.

Список использованных источников

1. [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.