Также производим декомпозицию отношения «Блюда» в два отношения «Вид блюда» и «Блюда».

Приложение базы данных было выполнено в среде Access 2007. Определив все взаимосвязи между отношениями, получим окончательную схему данных (рис. 2).

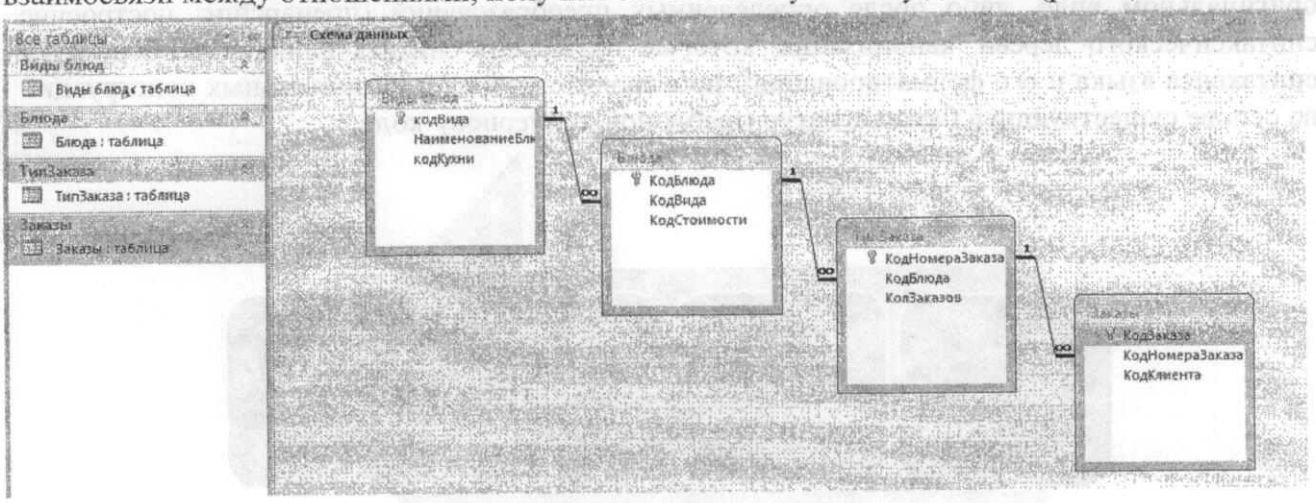

Рисунок 2 – Схема данных БД

Библиография

1. Клименко И.В., Лозовский А.В. Метод формальной нормализации отношений реляционной модели // Науч. мысль Кавказа. Прил. № 5. – 2004. – С. 115–119.
2. Мейер М. Теория реляционных баз данных. – М.: Мир, 1987. – 608 с.
3. Кузнецов С.Д. Основы баз данных. – М.: БИНОМ, 2007. – 484 с.
4. Фаворская М.Н. Распределенные базы данных: учеб. пос. – Красноярск: СибГАУ, 2003. – 116 с.

ВОПРОСЫ ВЫЯВЛЕНИЯ ДЕФЕКТОВ БЕЗОПАСНОСТИ КОДА МЕТОДОМ СТАТИЧЕСКОГО СИГНАТУРНОГО АНАЛИЗА

Марков А.С., Фадин А.А., Рауткин Ю.В.
НПО «Эшелон»
Москва, mail@cnpo.ru

Методы статического анализа исходных текстов программ в отличие от динамического тестирования позволяют избежать проблему проклятья размерности области тестовых данных и добиться большей степени автоматизации проверок на наличие дефектов безопасности, исходя из их конструктивных признаков. В настоящее время известен ряд техник статического синтаксического анализа, позволяющих эффективно определять некорректности кодирования (нефункциональные ошибки) [1]. Однако некоторые дефекты безопасности ПО могут иметь семантически или синтаксически корректную структуру, например, логические бомбы, ошибки конфигурирования, генераторы парольной информации и др. Для поиска подобных ошибок требуется привлечение эксперта, который исследует фрагменты кода повышенного риска, определяемые по некоторому шаблону [2]. В работе будут рассмотрена возможность использования сигнатурного анализа для выявления дефектов безопасности.

Работа была проведена при финансовой поддержке Минобрнауки.

Фактически под понятие сигнатурного анализа подпадает общий подход поиска тех или иных признаков объекта на основе сопоставления его содержимого с образцами из базы уже имеющихся признаков. Перспективным представляется использование данного подхода к

Таблица – Примеры способов выявления дефектов безопасности

Элемент CWE верхнего уровня	Элемент CWE нижнего уровня	Признаки наличия	Способ обнаружения
Проблемы поведения (Behavioral Problems) (438)	Изменение поведения в новой версии окружения (Behavioral Change in New Version or Environment) (438)	<совпадение вызова со списком проблемных для зависимости, используемой в проекте>	Определить версию среды функционирования Сравнить вызов со списком вредоносных функций
	Неверный контроль частоты взаимодействия (Improper Control of Interaction Frequency) (799)	<нет таймаута или запрета попыток> в интерактивной функции (взаимодействующей с пользователем или внешней стороной).	Определить интерактивность функционального объекта (по наличию API заданного типа) Определить внутри интерактивного объекта наличие API по обработке задержки
Ошибки каналов и путей (Channel and path errors) (417)	Неверная защита альтернативного пути Improper Protection of Alternate Path - (424)	<отсутствие проверки при показе закрытой страницы>	Определить вызов функционала, отображающего содержимое страницы Удостовериться в наличии функционала контроля сессии доступа пользователя
	Скрытый канал памяти (Covert storage channel) (515)	<обращение к нестандартным информационным объектам (файл подкачки, реестр, прямой доступ к диску, операции с кучей)>	Поиск вызовов функционала из списка объектов потенциально скрытых каналов
	Скрытый канал времени (Covert timing channel) (385)	<обращение к нестандартным информационным объектам (бесконечные циклы, работа с очередью сообщений, таймером, высокопроизводительным таймером)>	Поиск вызовов функционала из списка объектов потенциальных скрытых каналов
Обработка данных (Data Handling) (19)		<поиск включения введенных пользователем данных при формировании строк>	Поиск вызовов получения пользовательских данных Поиск вызовов потенциально опасных функций с использованием пользовательских данных
Обработка ошибок (Error Handling) (388)		наличие в блоке catch-операции вывода в журнал	Поиск обработчиков исключений и определение внутри вызовов вывода в журнал
Ошибки обработчика (Handler Errors) (429)		<выброс исключения, у которого нет обработчика>	Создание списка всех выбросов исключений Создание списка всех обработчиков Сравнение двух списков
Злоупотребление API (API Abuse) (227)		<использование внешних компонент (библиотек) с некорректными версиями>	Составление списка всех внешних зависимостей с версиями Поиск потенциально опасных вызов из этих библиотек
Индикатор плохого качества кода (Indicator of Poor Code Quality) (398)	Присваивание вместо (Assigning instead of Comparing) (481)	<наличие «=» вместо «==» в блоке if, где сравниваются лишь переменные>	Проверка заданного списка регулярных выражений для содержимого заданных типов блоков
Ошибки очистки и инициализации(Initialization and Cleanup Errors) (452)	Неверная инициализация (Improper Initialization) (665)	Отсутствие инициализации значений объектов (например, строк) перед их использованием	Поиск объявлений объектов, которые требуют инициализации перед своим использованием (например, статический массив строки) Поиск первого использования объекта в вызовах и операциях (например, правая часть в присваивании; функции, использующие их в качестве входных значений), при отсутствии их

настраивать для решения любой узкой задачи распознавания наборов). В дальнейшем на основе распознанных конструкций можно построить любую, самую сложную логику работы анализатора.

Чтобы решить задачи, описанные выше и построить базис для подобной логики, для каждого из поддерживаемых анализатором языков программирования требуется решить следующие задачи:

1) выполнить разбиение исходных текстов на строки операций с удалением комментариев, избыточных символов пробелов, переносов строк и другой незначимой для данного вида анализа информации;

2) выделить в строке операции вызова функционального объекта и извлечь литерала его названия (для повышения точности операции имеет смысл добавить небольшой список ключевых слов-исключений, представляющий зарезервированные слова этого языка, например: for, while, if);

3) выделить в строке операции передачу значения (в первую очередь, присваивание);

4) выделить литералы имен объектов, предположительно получающих значения, и литералов имен объектов, служащих источником данных значений.

Данный вид анализа в литературе по созданию компиляторов получил название «анализ потоков данных» (data-flow analysis). Рассмотрим схему работу анализатора, использующего сигнатурный анализ и анализ потоков данных для выявления дефектов программного кода.

Предложим следующее представление кода. По сути, каждая значимая строка конструкций исходного кода (за исключением специфических макросов или других директив компилятора) может быть представлена в виде следующего кортежа:

$$CD=<C, I, O>,$$

где C – это вызываемый элемент, I – это перечень элементов, чьё значение считывается, O – это перечень элементов, чье значение изменяется на основе считанных элементов.

Каждое поле типа «элемент» может быть одного из трех типов:

– объект (в данном элементе объявляется или создается объект, использующий память);

– ссылка (в данном выражении находится лишь имя или косвенная ссылка на имя существующего объекта);

– литерал (в данном выражении находится непосредственное значение элемента – строка, число и т.п.).

Исследование существующих баз дефектов кода ПО и изучение реальных примеров кода показали, что достаточно широкий спектр задач, связанных с выявлением потенциально опасных конструкций, можно решить с помощью сигнатурного анализа с внутрипроцедурным анализом потока данных (intro-procedural data-flow).

Среди достоинств сигнатурного метода статического анализа можно назвать относительную простоту реализации, очень высокую скорость работы, а также легкость портирования сигнатур на различные платформы и языки программирования.

Недостатком сигнатурного подхода является необходимость учитывать различные «вариации» конструкций и блоков кода, которые допускает синтаксис языка программирования и логика выполнения программы.

В связи с этим наиболее эффективной роль сигнатурного анализа видится в исследовании зависимостей модулей программ и внешних компонент, поиске вызовов функциональных объектов, а также проверке содержимого информационных объектов программы. Для класса ошибок, связанных с синтаксисом программы и достижимостью кода, его следует применять совместно с другими методами.