ПРОБЛЕМЫ АВТОМАТИЗАЦИИ ВЫЯВЛЕНИЯ ПРОГРАММНЫХ ЗАКЛАДОК И ДЕФЕКТОВ БЕЗОПАСНОСТИ КОДА МЕТОДОМ СТАТИЧЕСКОГО СИГНАТУРНОГО АНАЛИЗА

А.С. Марков, А.А. Фадин
ЗАО «НПО «Эшелон», Москва, 107023, ул. Электрозаводская, д. 24
code_audit@cnpo.ru

Ключевые слова: дефекты, уязвимости, безопасность программ, сигнатурный анализ, статический анализ

Аннотация. *Рассмотрен сигнатурный анализ безопасности программного кода. Приведены примеры выявляемых дефектов безопасности. Предложена методика выполнения сигнатурного анализа кода с целью выявления программных закладок и дефектов безопасности.*

Введение

Методы статического анализа исходных текстов программ в отличие от динамического тестирования позволяют избежать проблемы 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.

Место сигнатурного подхода в статическом анализе программ

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 выявлению ошибок в конфигурации приложений, а также потенциально опасных конструкций в коде приложений, в том числе программных закладок и дефектов кода.

Рассмотрим общую схему проведения сигнатурного анализа (рис.1). В полученном на входе анализатора исходном тексте ПО, либо в оригинальном виде, либо после определенных преобразований (препарсинг, построение синтаксического дерева, канонизация, которые позволяют снизить влияние особенностей синтаксиса языка и его форматирования), производится поиск подозрительных конструкций на основе существующей базы сигнатур (шаблонов, паттернов) кода.

В процессе работы анализатора выполняются сопоставления участков кода всем находящимся в базе паттернам (сигнатурам) потенциально опасных конструкций.

Результатом работы сигнатурного анализатора как правило является набор найденных подозрительных участков кода с указанием типа вероятного дефекта или программной закладки).

К примеру, сигнатурный анализатор безопасности кода можно представить кортежем [3]:

CSA = < SDB, HDB, MLA, MSA, MSY, MLO, MRP >,

где: *SDB* – база данных сигнатур (паттернов) ПОК, *HDB* – база данных структурной информации о коде (иерархического представления программы), *MLA* – программный модуль лексического анализа, *MSA* – программный модуль синтаксического анализа, *MSY* – программный модуль сигнатурного анализа, *MLO* – программный модуль логического вывода, *MRP* - программный модуль построения отчетов.

Сигнатурный анализатор осуществляет сквозной поиск в исходных текстах программы паттернов ПОК, используя базу сигнатур, которая представляет собой набор следующих кортежей:

SDB = <PE, DL,TP>,

где: *PE* – описание фрагмента ПОК (например, используя регулярные выражения), *DL* – оценка степени их опасности (например, числовая шкала 1–10), *TP* – тип (класс, категория) найденного дефекта или потенциальной уязвимости.

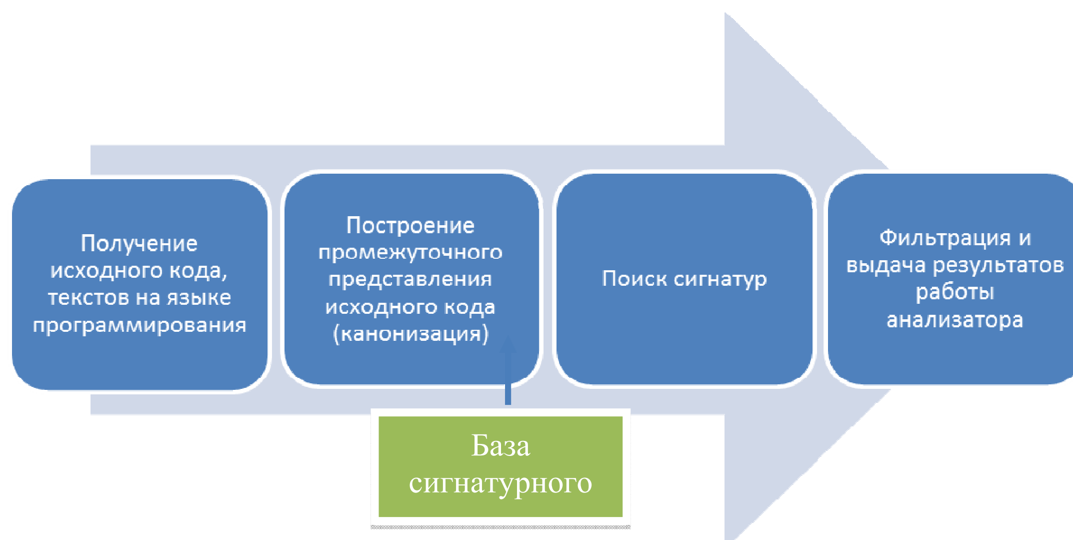

Рис. 1. Общая схема выполнения сигнатурного анализа

Типовые дефекты, выявляемые методом сигнатурного анализа

В статье для демонстрации возможности выявления дефектов выбрана система классификации дефектов CWE (Common Weakness Enumeration) [4]. На сегодняшний день последняя версия стандарта 2.1 доступна по адресу cwe.mitre.org. Примеры дефектов, их признаки и способы выявления представлены в таблице 1.

Таблица 1.

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

Элемент CWE верхнего уровня	Элемент CWE нижнего уровня	Признаки наличия	Способ обнаружения
			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 инициализации (левая часть в присваивании, функции использующие их в виде выходных значений)
Недостаточная инкапсуляция (Insufficient Encapsulation) (485)	Остаточный отладочный код (Leftover Debug Code) (489)	<наличие отладочного кода>	Поиск присутствия вызовов функций из списка отладочных Поиск подозрительных правил в названия отладочных функций (ключевые слова по debug)
Проблемы указателей (Pointer Issues) (465)	Разыменование нулевых указателей (NULL Pointer Dereference) (476)	<вызов операций по освобождению памяти для указателей, которые равны NULL>	Поиск вызовов функций, возвращающих указатель (на которую может повлиять пользователь), при отсутствии проверки ее значения следом за этим
Механизмы безопасности (Security Features) (254)	Использование жестко прописанных паролей (Use of hard-coded password) (259)	<наличие паролей и других данных авторизации непосредственно в коде приложения>	Поиск вызовов функций с параметрами авторизации, использующих аргументы со строковыми константами Поиск функций, требующих авторизации вместе с переменными, которым были присвоены строковые константы
Время и состояние (Time and State) (361)	Внешнее влияние на сферу определения (External Influence of Sphere Definition) (673)	<проверка наличия функций, использующих значения переменных окружения>	Поиск вызовов функций, читающих содержимое переменных окружения
Ошибки интерфейса пользователя (User Interface Errors) (445)	Нереализованная или неподдержимая функция в GUI (Unimplemented of unsupported feature in UI) (447)	<наличие скрытых элементов GUI, а также некорректностей в обработчиках событий>	Поиск наличия disabled=true hidden=true и других подобных свойств в файле GUI> Поиск отсутствия кода в функции-обработчике (Qt, Builder)

Разработка методики проведения сигнатурного анализа

Попробуем обобщить перечисленные выше подходы к выявлению дефектов безопасности.

Для выявления потенциально опасных конструкций необходимо решать задачи следующего рода:

1. Определение имени информационного объекта (как правило, переменной), которому происходит передача значения (как правило, присваивания) заданного потенциально опасного вызова (вызов функции, обращение к ассоциативному массиву значений).
2. Определение внешнего вызова (прежде всего функции) из списка потенциально опасных для данного языка, платформы или при условии подключения/не подключения заданного модуля/библиотеки и другого подобного контекста.
3. Определение внешнего вызова (прежде всего функции) из списка потенциально опасных, при условии использования в нем информационного объекта с известным именем.
4. Определение внешнего вызова (прежде всего функции) из списка потенциально опасных, при условии использования в нем аргумента (строкового параметра или информационного объекта) содержащего заданное строковое регулярное выражение.
5. Определение внешнего вызова из списка потенциально опасных, который использует тот же информационный объект, что уже был использован потенциально опасным вызовом.
6. Поиск всех определений блочных конструкций языка (например, перехват исключений или определение класса) использующих объект заданного типа (класс-родитель, аргумент перехвата исключения).
7. Определение для каждой блочной конструкции (например, класса) наличия заданного вложенного объекта (метода, другого класса) при условии выполнения условия (5).

Одним из решающих преимуществ сигнатурного анализа является его быстрота работы и легкость конструирования сигнатур (по сути, их можно оперативно дорабатывать и 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 (атомы) 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:

$$CD = \langle C, I, O \rangle,$$

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);
– атом (в данном выражении находится непосредственное значение элемента – строка, число, примитивный тип языка и т.п.).

Рассмотрим работу данного похода на практике, построив простой сигнатурный анализатор, способный обнаруживать SQL-инъекцию с учетом потоков данных программы.

Пусть у нас есть фрагмент программы на языке PHP с потенциально реализуемой SQL-инъекцией СУБД PostgreSQL на основе содержимого cookies клиентского браузера.

```
<?php
...
$id = $_COOKIE["mid"]; //получение данных
...
$query = "SELECT MessageID FROM messages WHERE
MessageID = '$id'"; //формирование запроса к БД
...
$result = pg_query($conn, $query); //выполнение запроса

?>
```

Для обнаружения этой конструкции, анализатору понадобятся следующие сигнатуры:

Сигнатуры синтаксиса:

1. Выделение названия функции из ее вызова (тип «ссылка»);
2. Выделение аргументов функции из ее вызова (тип «ссылка» или «атом»);
3. Определение источников и приемников данных во время присваивания (тип «ссылка» или «атом»).

Сигнатуры семантики:

1. Сигнатура выявления модификаций объектов критичных для программной системы (пример: выполнение запроса к СУБД, вызов функции pg_query);
2. Сигнатура выявления непроверенных источников данных (пример: обращение к ассоциативному массиву, содержащему значения cookies полученных от браузера клиента $_COOKIE).

Работа сигнатурного анализатора будет состоять в последовательном проходе по исходному тексту программы, заполнению кортежа CD элементами найденными с помощью синтаксических сигнатур, а также проверкой срабатывания сигнатур связанных с потенциально опасными операциями.

Шаг 1:

CD=(0,(”_COOKIE”),”$id”)

Анализатор считывает первую строку и определяет то, что C=0 (нет внешних вызовов) I=_COOKIE (входные данные для оператора массив пользовательских данных COOKIE), O=переменной $id, т.е. она получает значения от _COOKIE. При этом сработала сигнатура того, что _COOKIE – это критичная операция (поскольку имеет недоверенный источник данных, браузер на стороне клиента).

Шаг 2:

CD=(0, ”$id”, “$query”)

Определяем что вызовов нет (C=0), а $query (O) получает значения от $id (I). В анализаторе установлено правило транзитивности: если $id – получил значение _COOKIE (критичной операции), а $query получила значения от $id – она тоже содержит значения критичной операции.

Шаг 3:

CD=(“pg_query”, “$query” ,“$result”)

Определяем что в третьей строке кода идет внешний вызов pg_query, полученние входных данных от $query и запись в $result

Шаг 4:

4:(“pg_query”, “_COOKIE”,“$result”)

После шага №4 у нас срабатывают одновременно две сигнатуры критичных операций (получение данных из COOKIE и выполнение запроса к СУБД) и мы можем сигнализировать о наличии потенциально опасной конструкции: использование данных веденных от пользователя в SQL-запросах без предварительной фильтрации.

Заключение

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) [5].

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.

ЛИТЕРАТУРА

1. **Глухих М.И., Ицыксон В.М**. Программная инженерия. Обеспечение качества программных средств методами статического анализа. СПб.: Изд-во Политехн. ун-та, 2011. 150 с.
2. **Марков А.С., Миронов С.В., Цирлов В.Л.** Выявление уязвимостей программного обеспечения в процессе сертификации // Известия Южного федерального университета. Технические науки. 2006. Т. 62. №7. С. 82–87.
3. **Марков А.С., Фадин А.А., Цирлов В.Л**.. Концептуальные основы построения анализатора безопасности программного кода // Программные продукты и системы. 2012. №2.
4. Систематика дефектов и уязвимостей программного обеспечения / **А.С.Марков, А.А.Фадин, В.Л.Цирлов** // Безопасные информационные технологии. Сборник трудов II НТК. М.: МГТУ им.Н.Э.Баумана. 2011. С. 83–87.
5. Патент 114799 Российская Федерация. Система для определения программных закладок. / В.В.Вылегжанин, **А.Л.Маркин, А.С.Марков, Р.А.Уточка, А.А.Фадин, А.К.Фамбулов, В.Л.Цирлов** – № 2011153967/08(081180); заявл. 29.12.11; опубл. 10.04.12, Бюл. № 10. – 2 с.