А. Н. Макаров
ВНИИПВТИ

МЕТОД АВТОМАТИЗИРОВАННОГО ПОИСКА ПРОГРАММНЫХ ОШИБОК

Тестирование является важной составляющей жизненного цикла программного обеспечения (ПО). Ошибки, допущенные на различных стадиях создания ПО, влияют не только на надежность функционирования, но и на безопасность ПО. Методы, позволяющие сократить время поиска ошибок, в конечном итоге позволяют повысить уровень безопасности ПО.

Рассмотрим разработанный автором метод автоматизированного поиска программных ошибок (АППО). Метод АППО основан на методе «черного ящика» и выполняет стрессовое тестирование [1, 4] исследуемого ПО. Основные особенности метода:

— метод ориентирован на применение в условиях отсутствия исходного кода исследуемого ПО;

— метод предназначен для выявления ошибок кодирования («программистских» ошибок) ПО, которые, как правило, трудно выявить, основываясь на формальных методах верификации ПО;

— метод является динамическим (анализ исследуемого ПО проводится во время его работы, т. е. исследованию подвергается не само ПО, а процесс, порожденный ПО);

— метод применим для тестирования широкого класса ПО, например, ПО, обрабатывающее файлы с нетривиальной внутренней структурой или программные компоненты операционной системы (ОС);

— метод не привязан к какой-либо существующей аппаратно-программной платформе.

Метод АППО не является альтернативой известным методам тестирования, которые применяются к ПО с исходным кодом. Вместе с тем опыт применения метода АППО показывает, что данный метод в состоянии найти программные ошибки, которые не были найдены на предыдущих стадиях тестирования ПО. Поскольку метод является динамическим, то реализация метода на практике не всегда применима к ПО, которое использует различные антиотладочные приемы.

Введем обозначения:

— $B^0 = \varnothing$ — последовательность нулевой длины;

— $B = B^1 = \{0, \ldots, 255\}$ — множество последовательностей единичной длины;

— B^n — множество последовательностей длины n, где n —натуральное число, при этом если $b \in B^n$, то $b=(b_0,..,b_{n-1})$, где для любого $0 \le i < n$ выполняется $b_i \in B$;

— $B^* = \{B^i$, где $i = 1, 2,\ldots\}$ — множество всех последовательностей произвольной конечной длины.

Определение. *Процессом ОС* будем называть совокупность следующих элементов: S, I, ρ, s_0, где $S \subset B^N$ — множество всех состояний процесса, где N фиксированное для каждого процесса, $I \subset B^*$ — множество всех входных данных, $\rho: I \times S \to S$ — функция перехода процесса из одного состояния в другое в зависимости от входных данных, $s_0 \in S$ — начальное состояние процесса. Процесс ОС обозначим через $P<S, I, \rho, s_0>$.

Замечание. Функцию перехода ρ следует рассматривать как последовательность исполняемых инструкций, которая задана разработчиками ПО.

Определение. *Аварийное завершение* процесса $P<S, I, \rho, s_0>$ — состояние $a \in S$ процесса P, вызванное обработкой входных данных, такое, что для любого $i \in I$ выполняется равенство $\rho(i,a) = a$. Если процесс переходит в аварийное состояние, то будем говорить, что процесс P *завершился аварийно*.

Замечание. Если происходит аварийное завершение процесса ОС, то дальнейшее функционирование процесса P без вмешательства других процессов ОС невозможно. При аварийном завершении процесса ОС, как правило, удаляет процесс из системы, освобождая занятые процессом ресурсы (если это возможно). Если в ОС функционируют средства отладки, то удаление процесса из системы может быть приостановлено, а последующее управление ходом выполнения процесса передается средствам отладки.

Определение. *Эталонные входные данные* — это входные данные, которые созданы штатными средствами исследуемого процесса, и их обработка, по определению, не приводит к аварийному завершению процесса.

Определение. *Программная ошибка* — это ошибка реализации ПО, допущенная разработчиками ПО на этапе кодирования, которая приводит к аварийному завершению исследуемого процесса.

Введем обозначения:

— $A \subset S$ — множество аварийных состояний процесса P;

— $I_S \subset I$ — множество эталонных входных данных для процесса P, при этом для любого $i \in I_S$ и $s \in S \setminus A$ выполняется условие $\rho(i, s) \notin A$;

— $I_F \subset I$ — множество входных данных, при обработке которых происходит аварийное завершение процесса P, при этом, по определению, выполняется $I_F \cap I_S = \varnothing$;

— F — множество программных ошибок, содержащихся в процессе P, при этом будем считать, что $F=\{f \mid f=(a, i_1, \ldots, i_n)\}$, где n — число входных данных, подаваемых на вход исследуемому процессу, $a \in A$, для любого $1 \le k \le n$ выполняется условие $i_k \in I$, $(i_1, \ldots, i_n)$ — упорядоченная последовательность входных данных, передаваемых на обработку процессу P, такая, что выполняется равенство:

$$\rho(i_n, \rho(i_{n-1}, \ldots \rho(i_2, \rho(i_1, s_0)) \ldots)) = a;$$

— G — процедура внесения искажений в эталонные входные данные, которую будем называть процедурой формирования входных данных (ПФВД).

Метод АППО

Пусть имеется программное средство (ПС), которое реализует метод АППО, тогда применение метода заключается в выполнении следующих трех этапов.

Этап 1. Проводится анализ исследуемого ПО с целью выяснить, каким образом оно может быть активировано (запущен исследуемый процесс) ПС и каким образом исследуемому процессу будут передаваться входные данные. Определяется вид входных данных, которые могут быть, например, файлом, сетевым пакетом, последовательностью событий нажатия клавиш на клавиатуре.

Этап 2. ПС осуществляет автоматизированный поиск программных ошибок. Поиск заключается в том, что ПС запускает исследуемый процесс. Каждый запуск процесса назовем тестом. При выполнении одного теста исследуемому процессу передаются специально сформированные для этого теста входные данные. Если в результате обработки входных данных процесс завершился аварийно (найдена программная ошибка), то управление возвращается ПС, которое фиксирует состояние процесса и входные данные, обработка которых стала причиной аварийного завершения процесса.

Этап 3. Проводится анализ найденных на втором этапе программных ошибок. Цель анализа — определить, какие из программных ошибок являются наиболее «критическими».

Основным при реализации является второй этап, поэтому рассмотрим его подробней. Второй этап полностью автоматизирован. Поскольку на большинстве тестов исследуемый процесс не будет завершаться аварийно, целесообразно, чтобы число проведенных тестов было как можно больше (десятки тысяч тестов для одного исследуемого ПО).

Каждый тест выполняется за четыре шага.

Шаг 1. На первом шаге с помощью специально разработанных процедур формирования входных данных (ПФВД) подготавливаются входные данные, которые будут переданы исследуемому процессу. Подробнее ПФВД будут рассмотрены далее.

Шаг 2. На втором шаге ПС запускает исследуемый процесс и передает ему сформированные на первом шаге входные данные.

Шаг 3. На третьем шаге ПС регистрирует состояние исследуемого процесса с использованием специально разработанных процедур регистрации и интерпретации ошибок. Если исследуемый процесс на переданных ему входных данных завершает работу аварийно, то данный факт регистрируется. В

простейшем случае (это минимум того, что необходимо делать) регистрируется, на каких входных данных процесс завершился аварийно.

Шаг 4. На четвертом шаге теста в случае аварийного завершения исследуемого процесса восстанавливается корректность его последующих запусков для независимого выполнения очередного теста.

Опишем алгоритм выполнения тестов, который назовем алгоритмом АППО.

Алгоритм АППО. Пусть даны: процесс $P<S, I, \rho, s_0>$, конечное подмножество множества эталонных входных данных $I_{S1} \subset I_S$: $|I_{S1}| = n$ — мощность множества I_{S1}, G — ПФВД. Через s обозначим текущее состояние процесса P. Тогда алгоритм АППО осуществляет поиск программных ошибок $F_1 \subset F$ в результате выполнения следующей последовательности шагов:

1) $F_1 := \varnothing$, $s := s_0$;

2) если $I_{S1} \neq \varnothing$, то выбираются входные эталонные данные $i \in I_{S1,}$ и удаляется i из I_{S1}: $I_{S1} := I_{S1} \setminus \{i\}$, иначе поиск завершен;

3) с помощью ПФВД G формируется множество входных данных $I_T := G(i)$, $I_T = \{i_1,\ldots,i_m,\ldots,i_k\}$, где $I_T \in I$;

4) если $I_T \neq \varnothing$, то выбираются входные данные $i_m \in I_T$, где $m \in \{1,\ldots,k\}$, и удаляется i_m из I_T: $I_T := I_T \setminus \{i_m\}$, иначе осуществляется переход на шаг 2;

5) процессу P передаются входные данные i_m: $s := \rho(i_m, s)$;

6) проверяется, завершился ли процесс P аварийно $s \in A$, если да, то запоминаются входные данные $i_1,\ldots,i_m$ и состояние процесса s во множестве программных ошибок $F_1 = F_1 \cup \{(s, i_1,\ldots,i_m)\}$, а состояние процесса P переводится в начальное состояние $s := s_0$;

7) осуществляется переход на шаг 4.

Замечание. Описанный алгоритм АППО изложен для произвольного ПО. В то же время, для применения его при исследовании различных типов ПО (например, текстового редактора или почтового клиента), требуется соответствующая модификация алгоритма АППО, учитывающая особенности исследуемого ПО. В зависимости от типа исследуемого ПО алгоритм АППО может быть упрощен.

Процедуры формирования входных данных

Рассмотрим назначение ПФВД в алгоритме АППО. ПФВД должны сформировать входные данные, которые пройдут проверку исследуемым процессом и будут приняты к обработке.

Знание формата кодирования входных данных существенно облегчает задачу подготовки корректных входных данных. Но, как правило, формат входных данных может быть полностью или частично неизвестен, в связи с чем сформировать входные данные, которые пройдут проверку на корректность, но вместе с тем выявят программную ошибку, на которой процесс завершится аварийно, — задача нетривиальная.

Основная особенность ПФВД в том, что они формируют входные данные из эталонных входных данных, т. е. входных данных, которые были сформированы штатными средствами исследуемого ПО и, следовательно, принимаются к обработке исследуемым процессом. ПФВД осуществляют поиск строк или длин во входных эталонных данных и модифицируют их с таким расчетом, чтобы исследуемый процесс при обработке модифицированных структур завершился аварийно. Подобный подход к тестированию ПО приведен в работе [3] и получил название «зашумление» (fuzzing).

Разумеется, что ПФВД зависит от строения входных данных исследуемого ПО. Приведем пример ПФВД для поиска 32-битовых целочисленных переменных для бинарных входных данных (например, для бинарных OLE документов), но, прежде чем описывать ПФВД, дадим определение.

Определение. Определим операцию конкатенации "+" над последовательностями из B^*: для любых $a \in B^n$ и $b \in B^k$ по определению $a + b = (a_0,\ldots,a_{n-1},b_0,\ldots,b_{k-1}) \in B^{n+k}$. При этом B^4 — множество всех 32-битовых целочисленных переменных.

Процедура ПФВД 1. Пусть даны: эталонные входные данные длины n: $i \in B^n$, где $n \geq 0$ — фиксированное число, $i = (i_0,\ldots,i_{n-1})$; k — неотрицательный индекс: $0 \leq k \leq n-4$; m — число

«преобразуемых» бит, $m \in \{0,\ldots,31\}$; $f: B^4 \to B^4$ — параметр-функция, преобразующая 32-битовые целочисленные переменные. Тогда ПФВД G формирует множество входных данных $I_T \subset I$ в результате выполнения следующей последовательности шагов:

1) $I_T := \varnothing$, $k := 0$;

2) если $k > n-4$, то ПФВД G завершает работу;

3) последовательность i представляется как конкатенация трех подпоследовательностей: $i=i^{(1)}+i^{(2)}+i^{(3)}$, где $i^{(1)}=(i_0,\ldots,i_{k-1}) \in B^k$, $i^{(2)}=(i_k,\ldots,i_{k+3}) \in B^4$, $i^{(3)}=(i_{k+4},\ldots,i_{n-1}) \in B^{n-k-4}$;

4) если старшие m бит 32-битовой целочисленной переменной $i^{(2)}$ не равны 0, то осуществляется переход на шаг 6;

5) формируется новая последовательность и добавляется ко множеству $I_T := I_T \cup \{i^{(1)} + f(i^{(2)}) + i^{(3)}\}$;

6) $k := k + 1$ — увеличивается индекс;

7) осуществляется переход на шаг 2.

Замечание о выборе параметра m. Параметр m в ПФВД 1 выбирается в зависимости от длины n эталонных входных данных i. Из предположения о том, что в эталонных входных данных целочисленные переменные содержат длины и смещения строк или структур, следует, что значения целочисленных переменных не могут быть больше, чем значение длины эталонных входных данных.

Замечание о выборе параметра f. Параметр-функция f в ПФВД 1 выбирается, исходя из принципа, которым руководствуются, выбирая данные для тестирования с применением метода эквивалентных разбиений на основе результатов анализа граничных значений [2].

Процедура ПФВД 2 (неформальное описание). Если входные данные представлены в формате, например, RTF или HTML, то в качестве примера рассмотрим ПФВД, реализующую следующий алгоритм:

1) исходные данные преобразуются в промежуточное представление — дерево разбора;

2) берется очередной элемент дерева разбора и модифицируется (размер элемента многократно увеличивается, в значение элемента вносятся недопустимые символы и т. д.);

3) после изменения одного элемента из дерева разбора собираются «новые» входные данные;

4) модифицированный элемент восстанавливается, после чего осуществляется переход на шаг 2.

Заключение

Отсутствие найденных ошибок в исследуемом ПО при применении метода АППО не является основанием для того, чтобы считать, что ПО не содержит программных ошибок. Практическое применение метода АППО выявило, что результативность метода (оцененная количеством найденных программных ошибок) существенно зависит от реализации ПФВД.

Для эмпирической оценки результативности тестирования введен параметр K — относительное число аварийных завершений, который является отношением числа тестов, завершившихся аварийно, к общему числу проведенных тестов. Опыт применения описанного метода говорит о том, что при значении K около 0,1% (на 1000 тестов один тест завершается аварийно) и выше результативность метода для данного ПО можно признать удовлетворительной. Поскольку время проведения одного теста, как правило, занимает не более 10 секунд (на типовом компьютере), то при значении K порядка 0,1% и выше программные ошибки будут найдены за приемлемое время. Эксперименты показали, что существует ПО, для которого значение K достигает 5–6%.

Таким образом, метод АППО может успешно дополнить методы тестирования с доступным исходным кодом ПО, а также успешно осуществить поиск программных ошибок в ПО, когда его исходные коды отсутствуют.

СПИСОК ЛИТЕРАТУРЫ

1. *Бейзер Б.* Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем. СПб.: Питер, 2004.
2. *Калбертсон Р., Браун К., Кобб Г.* Быстрое тестирование. М.: Издательский дом «Вильямс», 2002.
3. *Козиол Д., Личфилд Д., Эйтэл Д., Энли К., Эрен С., Мехта Н., Хассель Р.* Искусство взлома и защиты системы. СПб.: Питер, 2006.
4. *Соммервилл И.* Инженерия программного обеспечения. 6-е изд. М.: Издательский дом «Вильямс», 2002.

И. В. Машкина (к. т. н., доцент), М. Б. Гузаиров (д. т. н., профессор)
Уфимский государственный авиационный технический университет

МЕТОДЫ РАЗРАБОТКИ ФУНКЦИОНАЛЬНОЙ МОДЕЛИ УПРАВЛЕНИЯ ЗАЩИТОЙ ИНФОРМАЦИИ

На сегодняшний день реальная практика менеджмента информационной безопасности (ИБ) в большинстве случаев сводится к наборам правил, в соответствии с которыми функционирует отдел ИБ, наборам положений и инструкций, которые слабо структурированы и взаимосвязаны.

Это обусловлено отсутствием понятия функциональной модели управления ИБ.

С помощью моделирования менеджмента ИБ можно эффективно анализировать проблемы управления и оптимизировать затраты на защиту информации. В работе предлагается подход к разработке математического метода принятия решений по организационно-техническому управлению защитой информации (ЗИ), построению модели системы управления (СУ) ЗИ с использованием IDEF-BPWin технологии.

1. Задача принятия решений по управлению модульным составом СЗИ и метод ее решения

В течение периода эксплуатации объекта информатизации (ОИ), как правило, неоднократно изменяются планы обработки информации и соответствующие им требования к уровню защищенности [1]. К тому же постоянно происходит обновление информационной системы, изменяется ее структура, используются новые ИТ, организуются новые подключения к системе. Учет этих факторов требует разработки новых подходов к обеспечению ЗИ.

Таким подходом может быть проектирование системы защиты информации (СЗИ), свойства и параметры которой динамично изменяются в зависимости от уровня критичности информации, обрабатываемой в данный период времени на объекте защиты.

Для этого должны быть реализованы механизмы *организационно-технического управления* (ОТУ) ЗИ, одной из задач которых является *обоснование наборов средств защиты*, используемых в СЗИ в определенные периоды функционирования ОИ.

Для совершенствования, развития и повышения эффективности СЗИ необходимы разработка и практическое применение методического обеспечения, связанного с решением задач проектирования СЗИ и организационно-технического управления защитой информации: синтез структуры и разработка алгоритмов функционирования системы поддержки принятия решений (ПР) для системы ОТУ ЗИ.

В работе используется идея морфологического подхода для моделирования и синтеза рациональных наборов средств защиты (СрЗ) для СЗИ.

Морфологический метод синтеза целесообразно использовать при проектировании СЗИ и в процессе ОТУ ЗИ [2], поскольку данный метод позволяет реализовать многоальтернативный и многокритериальный выбор, когда система представляет собой сложный комплекс СрЗ, включающий в себя наборы средств защиты для определенных точек их установки на объекте защиты; каждый набор