**Лекция 9**. **Подходы к защите разрабатываемых программ от автоматической генерации инструментальными средствами программных закладок**

# **Способы внедрения разрушающих программных средств (РПС) посредством инструментальных средств**

Современный этап развития технологии создания программных комплексов характеризуется ориентацией практически всех специализированных организаций на применение методов проблемноориентированного программирования. При этом повышение качества программ основывается на использовании мощных инструментальных средств разработки и отладки программ. Использование для автоматизации процессов производства в области программотехники инструментальных средств автоматизации программирования значительно осложняет проблему выявления и устранения РПС. Это обусловлено тем, что программист практически не имеет возможности контролировать непосредственно создаваемые программы, так как работает на уровне логических конструкций языковых средств.

Если целью атаки является нанесение как можно большего вреда, то заманчивой целью для нарушителя, пытающегося внедрить РПС, являются программы, которые используют много различных пользователей, например, компиляторы [РПС]. Для того чтобы понять, как это можно сделать рассмотрим следующую упрощенную структуру компилятора, которая дает представление об общих принципах его работы:

compile:

get (line);

translate (line);

Согласно этой структуры компилятор сначала «получает строку», а затем транслирует ее. Конечно, настоящий компилятор устроен намного сложнее, чем эта схема, по этой иллюстрации отдано предпочтение потому, что она в виде некой модели разъясняет фазы лексического анализа трансляции компилятора. Целью РПС будет поиск новых текстовых участков во входных программах, которые будут транслироваться, и вставление в эти участки различного кода. В примере, представленном ниже, компилятор ищет текстовый участок наличие которого в функции входа в данную компьютерную систему известно, как мы предполагаем, нападающей программе. Когда этот участок будет найден, компилятор не будет транслировать «read pwd(p)», а вместо этого будет транслировать вставку из РПС, которая может устанавливать «лазейку», которая потом позволит злоумышленнику легко получить доступ к системе. Измененный код компилятора следующий:

compile;

get (line)

if line=«readpwd(p)» then

 translate (destructive means insertion);

else

translate(line);

fi;

В этом измененном коде, компилятор получает строку, ищет нужный код текст, и если находит, то транслирует код РПС. Код РПС может включать в себя простую проверку пароля «черного входа» (например, может признаваться правильный пароль «12345» для любого пользователя). Это особенно опасно, поскольку код источника больше не отражает того, что находится в объектном коде и просмотр кода источника (не смотря на то, что проверяется и компилятор) никогда не позволит уловить эту атаку (см рис. 1).

Заметим, что на рис.1 исходный текст или исполняемый код, включающий только те выражения, которые предлагались его разработчиком назван чистым, а код, содержащий РПС, - грязным. Далее заметим, что если компилируется атакованный компилятор и грязный исполняемый код устанавливается код в какой-либо рабочий каталог (так обычно и бывает), то компилятор с внедренным РПС может быть обнаружен, только если кто-нибудь вернется к источнику компилятора и проверит его (что редко случается), Но настоящий источник может быть восстановлен злоумышленником после компилирования грязного источника и создания грязного исполняемого компилятора. Это, вообще говоря, потом поможет восстановить настоящии исполняемыи компилятор при рекомпиляции источника, но это редкий случай.



Рис.1. Работа компилятора с привнесенным РПС

Более изощренный метод внедрения РПС заключается в реализации следующего алгоритма.

compile;

get (line)

if line-“pattern\_1”

then translate (destructive means insertion\_N1);

if line-“pattern\_2” then

translate (destructive means insertion\_N2); else

translate(line);

fi;

Сущность метода заключается в простом добавлении еще одного РПС к уже существующему. Второй образец настроен на сам компилятор. Заменяющим текстом может служить самовоспроизводящаяся программа, которая вставляет оба РПС в компилятор. В этом случае необходима также фаза обучения. Сначала компилируется модифицированный исходныи текст нормальным компилятором, когда получается готовая программа с РПС. Затем устанавливается эта готовая программа как официальный компилятор. Теперь можно удалить РПС из исходного текста компилятора, а новый компилятор будет воспроизводить РПС при каждой перекомпиляции. Команда login (например для ОС Unix), естественно, будет оставаться с РПС без всякого следа в каких-либо исходных текстах.

Таким образом, по мнению лауреата премии Тьюринга К. Томпсона «Нельзя доверять программе, которую вы не написали полностью сами. Сколько бы вы не исследовали и не верифицировали исходный текст - это не защитит вас от троянской программы. Для демонстрации атаки такого рода возьмем компилятор Си. Можно выбрать любую программу, обрабатывающую другие программы - ассемблер, загрузчик или даже микропрограмму, зашитую в аппаратуру. Чем ниже уровень программы, тем труднее и труднее обнаруживать подобные «жучки», Мастерски встроенный «жучок» в микропрограмме будет почти невозможно обнаружить».

**Возможные методы защиты программ от потенциально опасных инструментальных средств**

Кроме трансляторов, компиляторов и интерпретаторов и целый ряд других инструментальных средств автоматизации программирования могут иметь встроенные средства автоматического генерирования программных закладок, которые включаются в текст производимой программы в ключевых местах, например, перед анализом условии прерывания, в блоках-переключателях, в блоках управления памятью и т.п. Следовательно, одной из задач обеспечения технологической безопасности программ является диагностический контроль инструментальных средств, позволяющий убедиться в отсутствии в них средств генерации преднамеренных программных дефектов.

Решение данной задачи может основываться на применении функциональных диаграмм, позволяющих строить тестовые последовательности по спецификации программной продукции. При этом спецификация исследуемых инструментальных средств разбивается на сегменты приемлемой сложности, в каждом из которых выделяются входные данные (команды инициализации операций) и события, представляющие собой реакцию программных средств на поступившую команду. Каждая команда может рассматриваться как отдельное входное условие или класс эквивалентности входных условий. Реакция программных средств наступает в виде следствия одного или нескольких входных условий. Для удобства анализа программных средств все команды и события получают уникальные номера, а связь команд с событиями определяется матрицей из нулей и единиц. На основании анализа семантики спецификаций инструментальных средств устанавливаются функциональные связи между командами и событиями, которые описываются булевыми функциями в базисе «И,ИЛИ,НЕ». В более сложных случаях могут использоваться расширения этого базиса за счет функций «И-НЕ», «ИЛИ-НЕ». Ограничения на возможные комбинации входных команд также задаются на основе анализа спецификаций.

Наиболее сложным и слабо формализованным этапом построения тестов диагностического контроля технологической безопасности инструментальных средств является этап нахождения по функциональным диаграммам множества наборов входных условий (тестовых наборов), удовлетворяющих входным ограничениям, при которых имеет место декларированная в спецификации комбинация событий. Описываемый ниже метод предназначен для упорядочения данной процедуры, а также для предотвращения возможности частичного пропуска функциональных участков тестируемых программ, слабо отраженных в спецификациях.

Сущность метода заключается в предоставлении функциональной диаграммы в виде логической сети без обратных связей, а также использовании специально введенных операций «прямого продвижения» и «обратного продвижения», часто применяемых в технической диагностике. В данном случае в логической сети, задающей функциональную диаграмму, входные переменные соответствуют командам, а выходные - следствиям. Все логические элементы сети соответствуют функциональным связям. В дальнейшем для упрощения вместо термина «логическая сеть», задающая функциональную диаграмму, будет использоваться просто «функциональная диаграмма».

Для упорядочения операций с функциональными диаграммами вводится понятие ранга функциональных связей. Выходная функциональная связь (ФС) относится к рангу r=1. К рангу (i+1) относится ФС, выходы которых являются входами ФС ранга i. Ранжирование ФС завершается после определения рангов для всех связей. Необходимо отметить, что если в функциональной диаграмме выход некоторой ФС разветвляется, то может оказаться, что ФС, уже отнесенная к некоторому рангу в соответствии с описанным правилом должна быть отнесена к другому, большему рангу. В этом случае ФС относится к большему рангу. Максимальное значение ранга ФС называется *рангом* *функциональной диаграммы*.

Операция «прямого продвижения» позволяет определить комбинацию событий, вызываемую заданной комбинацией команд. Операция «обратного продвижения» заключается в определении множества команд, вызывающих заданную совокупность событий. При описании этих операций используется понятие «вырожденное покрытие» булевой функции, описывающей ФС.

*Вырожденное покрытие* (ВП) булевой функции представляет собой сжатую таблицу истинности данной функции. При этом вырожденное покрытие булевой функции *f* от *n* аргументов есть совокупность (n+1)-разрядных строк, называемых кубами, в которых первые слева n разрядов представляют набор значений аргументов, а (n+1)- й разряд - значение булевой функции на этом наборе. В отличие от обычной таблицы истинности булевой функции, в которой аргументы принимают значение 0 или 1, аргументы в ВП могут принимать значение х (неопределенное). Если значение аргумента у равно х в некотором кубе ВП, то это значит, что значение булевой функции на наборах аргументов, задаваемых кубом не зависит от значения у, т.е. у может принимать как значение 0 так и значение 1 и задавать поэтому две строки таблицы истинности булевой функции.

Функциональная диаграмма характеризуется вектором состояния Р, размерность которого равна сумме числа ФС и числа входных условий. Каждый разряд вектора Р представляет значение либо команды, либо ФС и может принимать одно из 3-х значений: 0, 1 или х. Считается, что значение разряда вектора Р определено, если в нем записан 0 или 1.

Операционная схема метода состоит из этапов, каждый их которых реализуется с использованием собственного алгоритма, наиболее значимыми из которых являются следующие:

 алгоритм построения вырожденного покрытия функциональной диаграммы (алгоритм А);

 алгоритм определения множества комбинаций входных команд заданной совокупности событий (алгоритм Б);

 алгоритм вычисления эталонных значений событий для заданных тестовых наборов команд (алгоритм В).

Алгоритм А

Вырожденное покрытие функциональной диаграммы строится из ВП булевых функций, описывающих ФС Алгоритм определения ВП булевой функции *f* состоит из следующих процедур.

1. Любым известным методом находится минимальная дизъюнктивная нормальная форма (МДНФ) функции и ее отрицания.

2. Находятся кубы ВП, соответствующие значениям f=1. Для этого каждый простой импликанте ставится в соответствие куб, значения разрядов которого формируются по следующим правилам:

2.1. Если переменная входит в импликанту без отрицания, то в кубе, в котором *f*=1, ей соответствует 1, если же переменная входит с отрицанием, то значение 0;

2.2. Если переменная не входит в импликанту, то в кубе ей соответствует х.

3. Находятся кубы ВП, соответствующие значениям *f=*0. Для этого к МДНФ отрицания функции *f* применяют правила, определенные пунктом 2 алгоритма А.

Вырожденные покрытия булевой функции, описывающих ФС и применяемых при построении фушщиопальной диаграммы имеют вид, показанный в табл. 1.

Таблица 1

|  |
| --- |
| Функции |
| и | или | НЕ |
| *y1* | *y2* | *f* | *y1* | *y2* | *f* | *y* |  | *f* |
| 0 | х |  | 1 |  | 1 |  |  | 1 |
| х |  | 0 | х | 1 | 1 | 1 |  |  |
| 1 | 1 | 1 |  |  |  |  |  |  |

Вырожденное покрытие функциональной диаграммы строится по следующей процедуре.

1. Функциональные связи, входящие в функциональную диаграмму, нумеруются по следующим правилам:

а) номера 1,...,а присваиваются входным командам в произвольном порядке;

б) помера a+t,...,b, t≥1 присваиваются фупкциопальпым связям так, что номер выходного условия ФС (события) всегда больше номера ее любого входного условия (команды).

2. Строится таблица, имеющая d столбцов (d - количество ФС).

З. Начиная с ФС с наименьшим номером, в таблицу заносятся вырожденные покрытия булевых функций, описывающих ФС с учетом принятой нумерации.

Для реализации алгоритма Б выполняется операция «обратного продвижения», служащая для определения множества комбинаций входных условий (команд), при реализации которых имеет место заданная комбинация событий, соответствующая вектору Р состояния функциональной диаграммы. При этом в векторе Р должны быть определены все события, являющиеся следствиями из заданной комбинации команд, а остальные разряды вектора Р должны быть равны х. На комбинацию событий может быть наложено ограничение М(у1, y2) запрещающее одновременное равенство единице у1 и y2

Продвижение от событий к командам в соответствии с вектором Р заключается в выполнении алгоритма Б. Исходными данными при этом являются: вектор состояния функциональной диаграммы (совокупность значений реакций исследуемых инструментальных средств на входные команды) и вырожденное покрытие функциональной диаграммы.

Алгоритм Б

1. Начальная установка номера шага ј:=0.

2. Из вектора Рj состояния ФД выбирается ФС Sh с наибольшем номером, ранее не рассматривавшаяся, значение которой в Рj определено (0 или 1).

3. Если Sh есть входное условие ФД, то алгоритм завершается, т.е. значения входных условий, определенных в Рj, являются искомыми.

В противном случае выполняется п.4.

4. Из таблицы вырожденных покрытий ФС выбираются кубы $K\_{h}^{(1)},…,K\_{h}^{(k)}$, в которых значения ФС совпадают со значениями, определенными в Рj, а значения входных условий ФС не противоречат их значениям в Рj. Противоречие имеет место тогда и только тогда, когда значение входного условия в кубе $K\_{h}^{(j)}$ равно 0(1), а в Рj противоположное 1(0). Если h больше наименьшего номера ФС, значение которого определено в Рј, но из вырожденного покрытия ФС Sh нельзя выбрать ни одного куба, удовлетворяющего указанным условиям, то Рj не реализуем, т.е. не существует комбинации входных условий, при которой значение следствии соответствует Рj.

5. Находится множество векторов $P\_{j}^{(1)},…,R\_{j+1}^{(k)}$ поразрядного пересечения расширенных кубов $K\_{h}^{(1)},…,K\_{h}^{(k)}$ вырожденного покрытия Sh с вектором Рј: $P\_{j+1}^{(i)}=P\_{j}P\left\{K\_{h}^{(i)}\right\}$, i=1,...,k, причем операция П выполняется по следующим правилам: выбраппые кубы расширяются за счет присвоения всем неопределенным ФС значения *х*, 1П1=1, 0П0=0, 1П*х*=1, 0Пх=хП0=0, 1П0=0П1=d.

6. j:=j+1; пункты 2-5 выполняются для каждого из найденных векторов, $P\_{g}^{(j)}$, i=1,...,k.

В результате работы этого алгоритма заполняются соответствующие строки таблицы решений. Для получения полной таблицы решений необходимо выполнить алгоритм для всех комбинаций событий, включаемых в таблицу 1.

Рассматриваемые комбинации входных команд могут содержать некоторые неопределенные значения (х). Предположим, что таких значений l. Тогда существует 21 полностью определенных комбинаций входных команд, причем доопределение комбинации для получения тестовых наборов должно осуществляться с учетом ограничений, полученных при анализе спецификаций исследуемых на технологическую безопасность инструментальных средств. Возможны следующие ограничения на входные условия:

  - запрещены комбинации входных условии, в которых более одной переменной равны 1;

  - запрещена комбинация входных условии, в которых все xi=0;

 *O(x1,x2)-* одно и только одно из значений *х*1 и *x*2 должно иметь место, т.е. запрещены комбинации (*x1=*1, *x2=*1).или (*x1=*0, *x2=*0).

 *R(x1,х2)* - запрещена комбинация (*x1=*0, *x2=*0).

Возможны комбинации ограничений, например,  (i,j)=1,...,n. В общем случае ограничение может быть задано указанием произвольного множества запрещенных комбинаций команд (входных условий) Учет ограничений может быть осуществлен следующими способами.

Первый заключается в генерации комбинаций, удовлетворяющих ограничениям, второй основан на генерации всех возможных вариантов доопределения с последующим отбором тех вариантов, которые удовлетворяют системам ограничений. Исследования показали, что в алгоритмическом и программном отношениях первый способ оказывается значительно сложнее второго.

Особенно возрастает сложность программ тестирования в связи с необходимостью генерации тестовых наборов при наличии комбинации ограничений. Кроме того, при этом надо определить, все ли допустимые тестовые наборы построены, а также после генерации очередного тестового набора проверить, не содержится ли он в уже сформированном множестве.

Второй способ лишен этих недостатков. Действительно, для проверки соответствия комбинации некоторому ограничению строится относительно простая подпрограмма. При наличии некоторых ограничений, наложенных на одни и те же входные условия, каждый вариант доопределения комбинации входных условий последовательно и в произвольном порядке проверяется на выполнение каждого из наложенных ограничений, в то время как при генерации допустимого тестового набора эту последовательность приходится учитывать. Отпадает необходимость и проверки наличия очередного тестового набора, удовлетворяющего ограничениям в уже сформированном множестве, так как этот набор в принципе не может совпадать ни с одним из наборов, уже сформированных ранее, потому что эти наборы не генерируются, а отбираются из попарно различных вариантов доопределепия комбинаций. Недостатком второго способа является необходимость перебора всех вариантов доопределения комбинаций входных условий. Учитывая, что подпрограммы учета входных условий для типовых участков инструментальных средств автоматизации программирования достаточно просты, то при использовании современных средств вычислительнои техники этот недостаток не практике несущественен.

Алгоритм В производит вычисление эталонных значений реакции исследуемых инструментальных средств на заданный тестовый набор входных условий осуществляется с использованием операции прямого продвижения вектора Р к выходам функциональной диаграммы. В данном алгоритме XT определяет тестовый набор входных условий, а значения остальных разрядов вектора Р равны *х*. Необходимо заметить, что в векторе Р можно было бы определить те значения реакций на входные команды, при которых определялся тестовый набор ХT. Однако, если в векторе Р значения этих реакций положить равными х и выполнить продвижение к выходам, то появляется дополнительная возможность проверки правильности вычисления ХT. Кроме того, комбинация входных условий, доопределенная с учетом ограничений, как правило, порождает комбинации реакций, в которых наряду с определенными значениями, соответствующими набору команд ХТ определены и значения других реакций исследуемого участка программных средств.

Задача, решаемая данным алгоритмом, формируется следующ образом.

Дано:

 вырожденное покрытие функциональной диаграммы инструментальных средств (или их относительно автономного участка), исследуемых на отсутствие блоков формирования программных закладок диверсионного типа;

 вектор Р(О) состояния функциональной диаграммы, соответствующий тестовому набору ХТ.

Требуется: найти для исследуемых средств значения реакций соответствующие вектору Р(О).

Решение данной задачи осуществляется следующим образом.

Алгоритм В

1. Начальная установка номера шага алгоритма ј:=1.

2. В векторе Рј-1 определяется функциональная связь Sg с минимальным номером, значение которой не определено, а хотя бы одно входное условие этой ФС имеет значение 0 или 1. Так как в соответствии с принятым правилом нумерации номер ФС всегда больше номера любого ее входного условия, то g≥V+j, где  - число входных условий ФС. Поэтому при выполнении этой операции следует анализировать значения ФС с номерами где - число ФС.

3. Из вырожденного покрытия ФС Sg выбирается куб Кhg, значения компонентов которого не противоречат значениям соответствующих компонентов вектора состояния Рј-1 Противоречие отсутствует, если для всех компонентов куба Khg выполняются следующие условия:

* 1. Если в кубе компонент а имеет значение х, то в векторе Р(ј-1) значение компонента а может быть 0, 1 или х;

3.2.Если в кубе Khg компонент а имеет значение d, d=0,1 то и в векторе Рј-1 этот компонент должен иметь значение d.

1. Если не выбрано ни одного куба, то Рј=Рј-1 и осуществляется переход к п.6, в противном случае — к п.5.
2. Формируется вектор Рj состояния ФД, разряды которого образуются с помощью операций П поразрядного пересечения (см. п.5 алгоритма Б) компонентов выбранного куба Kgh с соответствующими компонентами вектора Рј-1 и заменой значений и этих компонентов результатами операции перечисления.
3. Если g<N+V, то осуществляется переход к п.7. В противном случае алгоритм завершается.
4. j:=j+1, переход к п.2.

Таким образом, рассмотренный метод построения тестов проверки технологической безопасности исследуемых инструментальных средств и алгоритмы его реализации позволяют сформировать и проанализировать таблицы решений для полпого тестового набора. В то же время, создание самих тестов представляет собой самостоятельную задачу, связанную с необходимостью разработки некоторого программного обеспечения, которое само по себе должно соответствовать логическим схемам проведения проверок.