

*Д. В. Анисимов, В. А. Петров (к. т. н., доцент)*
Московский инженерно-физический институт (государственный университет)

# РЕАЛИЗАЦИЯ АНАЛИЗА ПОЛНОТЫ И ОТСУТСТВИЯ ИЗБЫТОЧНОСТИ ИСХОДНЫХ ТЕКСТОВ НА УРОВНЕ ФАЙЛОВ

*Описывается методика проведения анализа полноты и отсутствия избыточности в исходных текстах на уровне файлов программного обеспечения, подлежащего сертификации по требованиям ФСТЭК.*

При сертификации программных продуктов по требованиям ФСТЭК [1] одним из главных и первоочередных требований является проведение контроля полноты и отсутствия избыточности исходных текстов на уровне файлов. В настоящее время данный вид контроля проводят многие сертификационные лаборатории, в основном проводя компиляцию представленных исходных текстов при определении полноты и отслеживая связь файлов по служебным заголовкам (include) при определении избыточности, однако этого мало для получения реальных данных по этому виду контроля.

Цель данной статьи — дать подробное описание процесса проведения проверки на полноту и отсутствие избыточности на уровне файлов, описать возможные проблемы и предложить пути их решения. При этом необходимо решить следующие основные задачи:

— обеспечить гарантии полноты представленных исходных текстов (присутствия всех необходимых для функционирования исследуемого программного продукта файлов);

— исключить присутствие файлов, не содержащих необходимой информации для корректного (декларированного в документации на программный продукт) функционирования исследуемого программного обеспечения (ПО).

Рассмотрим основные способы решения этих задач.

На первом этапе необходимо провести вычисление контрольных сумм для файлов проекта и составление списка файлов проекта.

Общей проверкой для проведения контроля полноты и отсутствия избыточности исходных текстов является сравнение полученного списка файлов и контрольных сумм со списком, представленным в документации к исследуемому программному продукту.

Полнота исходных текстов может быть определена путем сборки (компиляции) исследуемого продукта, проверкой контрольных сумм получившихся исполняемых файлов с контрольными суммами, указанными в документации к проекту, а также проверкой работоспособности полученного программного продукта.

Выбор методики проведения контроля избыточности исходных текстов на уровне файлов зависит от используемого при разработке языка программирования. Рассмотрим случаи использования языков C и C++. В случае применения этих языков программирования задача сводится к выполнению следующих проверок:

— анализируются файлы сборки (сценарии компиляции «Makefile»), определяются файлы, участвующие в компиляции;

— проводится анализ служебных заголовков (include), по которым определяются подключаемые файлы;

— проводится анализ содержимого исходных текстов на наличие функций работы с файлами (fopen, fread, freopen и др.) и файловых потоков (ifstream), выявляя файлы, вызываемые в процессе работы программы. При этом учитываются существующие файлы, а создаваемые при вызове функций не учитываются;

— по полученным данным строится следующая матрица (граф) связанности (вызовов) файлов $M=(N,Z)$, где $N$ — множество файлов проекта, а $Z$ — множество связей между ними:

$$M=\begin{vmatrix} 0 & k_{1,2} & k_{1,3} & \dots & k_{1,j} & \dots & k_{1,n} \\ k_{2,1} & 0 & k_{2,3} & \dots & k_{2,j} & \dots & k_{2,n} \\ k_{3,1} & k_{3,2} & 0 & \dots & k_{3,j} & \dots & k_{3,n} \\ \dots & \dots & \dots & 0 & \dots & \dots & \dots \\ k_{i,1} & k_{i,2} & k_{i,3} & \dots & 0 & \dots & k_{i,n} \\ \dots & \dots & \dots & \dots & \dots & 0 & \dots \\ k_{n,1} & k_{n,2} & k_{n,3} & \dots & k_{n,j} & \dots & 0 \end{vmatrix}, \quad (1)$$

где $n$ — количество файлов проекта, между которыми определяется связь, $k_{i,j}$ — результат связи между $i$ и $j$ файлами, $k_{i,j} \in Z$. Если $i = j$, то $k_{i,j} = 0$ (рекурсия на уровне файлов не применима). Значения $k_{i,j}=[0, 1]$ имеют следующий смысл:

— 0 — отсутствует связь между файлами;

— 1 — $i$-й файл производит вызов или вызывается $j$-м файлом.

Для проведения контроля избыточности на уровне файлов введем параметр $K$, который будет описывать сумму значений (сумму результатов связей между файлами) матрицы $M$.

Следующее выражение описывает сумму результатов связей между файлами:

$$K = \sum_{i=1}^{n} \sum_{j=1}^{n} k_{i,j}, \text{ если } i=j, \text{ то } k_{i,j} = 0. \quad (2)$$

Проведение исследования требует определения взаимосвязи каждого файла с остальными файлами проекта, поэтому введем параметры: $K_i$ — сумма результатов связи между $i$-м файлом и остальными файлами проекта и $K_j$ — между $j$-м файлом и остальными файлами.

Для описания этих параметров выражение (2) принимает вид:

для $i$-ого файла $K_i = \sum_{j=1}^{n} k_{i,j}$, если $i = j$, то $k_{i,j} = 0$, (3)

или

для $j$-го файла $K_j = \sum_{i=1}^{n} k_{i,j}$, если $i = j$, то $k_{i,j} = 0$. (4)

Дальнейшее рассмотрение цепочек связанностей будет проводиться с $K_i$.

Вычислим $K_i$, тогда (3) для всех файлов принимает следующий вид:

$$\begin{cases} K_i = \sum_{j=1}^{n} k_{i,j}, \text{ если } i = j, \text{ то } k_{i,j} = 0. \\ i = 1..n \end{cases} \quad (5)$$

Полученные результаты представляем в виде матрицы $T$:

$$T = \begin{vmatrix} t_1 \\ t_2 \\ \dots \\ t_i \\ \dots \\ t_n \end{vmatrix}, \text{ где } \begin{cases} t_i = 1, \text{если } K_i > 0 \\ t_i = 0, \text{если } K_i = 0 \end{cases}. \quad (6)$$

Результаты, где $t_i=0$, указывают, что выявленные файлы не имеют цепочек связанностей, т. е. указывают на избыточность $i$-го файла, однако нельзя сразу утверждать, что все выявленные таким образом файлы избыточны.

Для проведения контроля избыточности на уровне файлов полученных результатов недостаточно, так как

— существует вероятность необнаружения связи с другими файлами (принятие файла за избыточный);

— один или несколько файлов проекта могут являться независимыми компонентами и не иметь связей с другими файлами;

— может присутствовать связь между файлами, но они, в свою очередь, не взаимодействуют с основными файлами исследуемого ПО и не влияют на работу программы в целом (файлы между собой передают информацию, а в проекте не задействованы);

— существует возможность присутствия связи между файлами по формальному признаку (случаи подключения неиспользуемых библиотек или файлов, используемых при тестировании на стадии разработки и отладки), при которой нет передачи информации между ними.

Имеется два варианта решения этих проблем:

— дополнительный ручной анализ полученных результатов. При этом необходимо проводить анализ выявленных избыточных файлов на возможную связанность с проектом, которая не была обнаружена при получении матрицы $T$. Данный вид анализа является сложным: требует длительного времени проведения и накладывает ограничения по уровню профессиональной подготовки на лиц, его проводящих. Прямо пропорционально росту объема исследуемых исходных текстов падает практическая возможность их обработки в таком режиме за определенный промежуток времени, что ведет к неприменимости данного варианта анализа при очень больших объемах исходных текстов;

— проведение анализа с помощью метода, позволяющего снизить временные затраты на анализ программного продукта и не вносить ограничений по объему исследуемого исходного текста.

Для проведения такого анализа необходимо

— выделить файлы с исходными текстами, из которых непосредственно компилируются исполняемые файлы. Результатом этого исследования является множество (список) $S$ с их номерами в матрице $T$. $s_r \in S$, где $s_r$ — значение (номер файла) из полученного списка, $r = 1..z$, $z$ — количество файлов в списке;

— построить матрицу $T'$, такой же размерности, что и $T$, элементами которой являются значения $t'_i$ (где $i=1..n$), полученные следующим способом:

1) для каждого значения $s_r$ из списка проанализировать связи $k_{s_r,j}$ в матрице $M$, где $r = 1..z$, $j = 1..n$;

2) если $k_{s_r,j}=0$, перейти к пункту 4. Если $k_{s_r,j} \neq 0$, то $t'_{s_r}=1$. Перейти к пункту 3;

3) $s_r=j$, рекурсивно выполняем пункт 2 до тех пор, пока в цепочке связанности не останется строк, содержащих значения $k_{s_r,j} \neq 0$. После этого возвращаем первоначальные значения (до выполнения рекурсивного анализа) $s_r$ и $j$;

4) $j=j+1$; если $j \leq n$, выполняем пункт 2, иначе пункт 5;

5) $r=r+1$, если $r \leq z$ , $j=1$ и выполняем пункт 2.

После того, как произойдет заполнение матрицы $T'$, анализируются ее значения $t'_i$ и делается заключение о проведенных исследованиях. Если $t'_i=0$, то с определенной вероятностью можно утверждать, что $i$-й файл избыточен.

При использовании такого способа анализа избыточности исходных текстов ПО на уровне файлов получаются максимально достоверные результаты исследования. Данный метод сводит к минимуму возможность возникновения ошибок при проведении исследований и позволяет обрабатывать большие объемы исходных текстов за выделенное для сертификации время.

## СПИСОК ЛИТЕРАТУРЫ

1. Руководящий документ. Защита от несанкционированного доступа к информации. Государственная техническая комиссия при Президенте Российской Федерации. 1999 г. Часть 1. Программное обеспечение средств защиты информации. Классификация по уровню контроля отсутствия недекларированных возможностей.

*Е. А. Васильева, Н. В. Медведев (к. т. н., доцент)*
Московский государственный технический университет им. Н. Э. Баумана

# МЕТОДИКА ТЕСТИРОВАНИЯ МЕЖСЕТЕВЫХ ЭКРАНОВ

*Рассматриваются особенности реализации межсетевых экранов с инспекцией состояния, и предлагается подход к их тестированию.*

### Введение

На сегодняшний день одними из самых востребованных средств защиты сетевого взаимодействия являются межсетевые экраны. Межсетевой экран (МЭ) представляет собой программный, аппаратный или программно-аппаратный комплекс, реализующий функции фильтрации сетевого трафика (информационных потоков) между двумя или более автоматизированными системами по некоторому набору правил, определяемых политикой безопасности защищаемой сети. Существует множество различных классификаций МЭ [1–3]. Несмотря на это, можно выделить основные категории МЭ: пакетные фильтры (packet filter), прикладные посредники (application proxy), инспекторы состояния (stateful inspection firewalls).

В большинстве случаев проверить и сравнить заявленные возможности МЭ различных производителей не вызывает проблем [4]. Исследование (тестирование) же инспекторов состояния из-за особенностей функционирования ставит перед специалистами ряд вопросов. Основная сложность заключается в том, что подробные алгоритмы их функционирования являются авторскими разработками и не раскрываются производителями. Ситуация осложняется тем, что на сегодняшний день не существует специальных прикладных средств, предназначенных для проведения таких тестов.

Основной принцип работы инспектора состояния заключается в следующем [1]: ни один сетевой пакет не будет пропущен, если он не принадлежит к некоторому виртуальному соединению, ассоциированному МЭ с ранее установленным соединением (рис. 1). Исключение составляют пакеты, разрешенные политикой безопасности и принадлежащие текущей стадии установленного соединения. Информация обо всех виртуальных соединениях хранится в специальной таблице, называемой таблицей состояний.