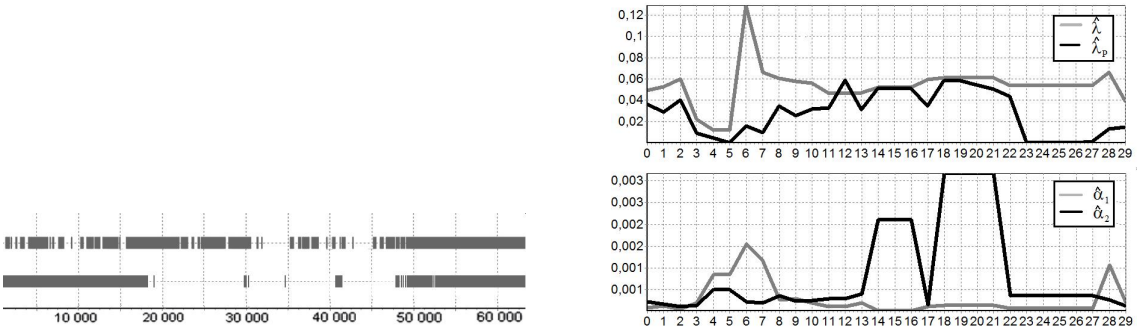
для эксп. 1: $\hat{\lambda} = 0{,}0458$, $\hat{\alpha}_1 = 0{,}0004$, $\hat{\alpha}_2 = 0{,}0011$, $\hat{\lambda}_P = 0{,}0332$. Для эксп. 2: $\hat{\lambda} = 0{,}0522$, $\hat{\alpha}_1 = 0{,}0002$, $\hat{\alpha}_2 = 0{,}0003$, $\hat{\lambda}_P = 0{,}0338$. Оценка $\hat{\lambda}_P$ для обоих случаев близка, с точки зрения простейшего потока эти случаи неразличимы. Оценки параметров альтернирующего потока отличаются существенно, эта модель — более гибкая.



а *б*

Рис. 1. Моменты поступления пакетов данных (2 эксперимента) (*а*). Динамика оценок (*б*)

На рис. 1,*а* представлена динамика оценок параметров трафика пользователя, работавшего с сетевыми сервисами и посещавшего сайты, в течение часа. Средние оценок, представленных на рис. 1,*б*, составили: $\overline{\hat{\lambda}}$=0,0534, $\overline{\hat{\alpha}}_1$=0,0003, $\overline{\hat{\alpha}}_2$=0,0009, $\overline{\hat{\lambda}}_P$=0,0281. Выборочные дисперсии оценок: $D_{\hat{\lambda}}$=0,0003, $D_{\hat{\alpha}_1}$=0,00001, $D_{\hat{\alpha}_1}$=0,00001, $D_{\hat{\lambda}_P}$=0,0004. Анализ результатов позволяет сделать следующие выводы: 1) стабильность оценок говорит о том, что модель альтернирующего потока более адекватно описывает трафик пользователя, чем модель пуассоновского; 2) резкое изменение оценок для альтернирующего потока позволяет использовать их для анализа сетевой активности.

ЛИТЕРАТУРА

1. *Головко Н. И., Каретник В. О., Танин В. Е., Сафонюк И. И.* Исследование моделей систем массового обслуживания в информационных сетях // Сиб. жур. индустр. матем. 2008. Т. XI. № 2(34). С. 50–58.
2. *Васильева Л. А., Горцев А. М.* Оценивание параметров дважды стохастического потока событий в условиях его неполной наблюдаемости // Автоматика и телемеханика. 2002. № 3. С. 179–184.
3. *Вентцель Е. С., Овчаров Л. А.* Теория случайных процессов и ее инженерные приложения. М.: Высш. шк., 2000. 383 с.

УДК 004.412

К ОПРЕДЕЛЕНИЮ СТЕПЕНИ ИНТЕГРИРОВАННОСТИ ПРОГРАММНЫХ ПОДСИСТЕМ

Д. А. Стефанцов

При разработке защищённых систем обработки информации (СОИ) строится модель нарушителя в виде формального описания набора угроз и/или атак и формулируется политика безопасности (ПБ) в виде набора формальных требований [1]. С изменением модели нарушителя появляется необходимость внесения изменений в ПБ.

Примером подобных изменений может служить реализация политики мандатного разграничения доступа SELinux для операционной системы GNU/Linux, ранее обладавшей только политикой дискреционного разграничения доступа [2]. Тесная интеграция программных реализаций СОИ и ПБ является препятствием к внесению изменений в ПБ [3], в связи с чем возникает проблема выбора такого метода интегрирования программной подсистемы ПБ в программную подсистему СОИ, который обеспечивал бы наименьшую степень интегрированности (далее: СИ) первой во вторую. Решение этой проблемы предполагает наличие количественной характеристики СИ одной программной подсистемы в другую.

В данной работе предлагается вариант формального определения СИ произвольной программной подсистемы, называемой подчинённой, в произвольную программную подсистему, называемую главной, и показывается его применение к оценке СИ подсистемы ПБ в файловую подсистему ядра операционной системы Linux.

Это определение удовлетворяет всем критериям метрики программного обеспечения, сформулированным в [4]. В его основе лежит подсчёт количества идентификаторов (имён переменных, функций, констант и т. п.) подчинённой подсистемы, встречаемых в исходном тексте главной подсистемы. Предполагается, что при модификации подчинённой подсистемы на обработку идентификаторов тратятся некоторые (вычислительные, ёмкостные, временные и др.) ресурсы, совокупность которых характеризует сложность этой обработки, и СИ определяется в зависимости от её величины.

Грубой оценкой СИ программных подсистем может служить выражение $M = {=}\alpha \cdot n$, где n — это количество идентификаторов подчинённой подсистемы, встречаемых в исходном тексте главной подсистемы, и α — средняя сложность обработки одного идентификатора.

Для более точного определения СИ рассмотрим сначала простейший случай главной подсистемы — последовательность f некоторых команд s_i без циклов и ветвлений, т. е. $f = (s_1,\ s_2, \ldots,\ s_k)$. Обозначим $n(s_i)$ количество идентификаторов подчинённой подсистемы в команде s_i. Пусть задана функция $\alpha(n)$ — сложность обработки команды, содержащей n идентификаторов подчинённой подсистемы. Тогда более точную оценку СИ подсистем даёт выражение $M(f) = \sum\limits_{i=1}^{k} \alpha(n(s_i))$.

Предполагается, что функция $\alpha(n)$ определена либо на всём множестве целых неотрицательных чисел, либо на множестве $\{0,\ 1,\ \ldots,\ N\}$ для достаточно большого N. Она должна быть также монотонно неубывающей и обладать свойством $\alpha(0) = 0$.

Рассмотрение случаев циклов и составных условных операторов может быть проведено аналогично случаю неполного условного оператора (НУО), поэтому далее предполагаем, что среди команд в программах могут встречаться только НУО и выражения (команды, не содержащие циклов и ветвлений). В свою очередь, НУО может быть рассмотрен как алгоритм $f = (a; s)$, где a — условие, $s = (s_1, \ldots, s_k)$ — последовательность команд, являющаяся телом условного оператора. Полагая идентификаторы подчинённой подсистемы, встречаемые в параметрах алгоритма и в одной из его команд, связанными, определим степень интегрированности $M(f)$ подчинённой подсистемы в реализацию алгоритма f рекурсивно как

$$M(f) \;=\; M(a;\; (s_1,\; \ldots s_k)) = \sum_{i=1}^{k} M(a; s_i),$$

$$M(a;\; s_i) \;=\; \begin{cases} \alpha(n(a) \;+\; n(s_i)),\; \text{если } s_i \text{ — выражение,} \\ M(a,\; a_i;\; (s_{i1},\; \ldots,\; s_{il})) \;= \sum\limits_{j=1}^{l} M(a,\; a_i;\; s_{ij}), \\ \text{если } s_i = (a_i;\; (s_{i1},\; \ldots,\; s_{il})) \text{ — НУО.} \end{cases}$$

Введённая мера СИ была применена к существующим программным системам — к версиям 0.01 и 2.6.24 ядра операционной системы Linux, являющимся соответственно самой первой и одной из самых последних версий ядра Linux. В качестве главной подсистемы была выбрана подсистема работы с файлами с помощью системных вызовов sys_open(), sys_mkdir(), sys_rmdir(), sys_link(), sys_unlink(), а в качестве подчинённой — реализация дискреционной политики безопасности.

Для Linux 0.01 получено значение СИ, равное $\alpha(1) \;+\; 5\alpha(2) + 5\alpha(3)$, а в случае Linux 2.6.24 СИ равна $3\alpha(1) \;+\; 2\alpha(2) \;+\; 3\alpha(3)$. Как было отмечено, функция α — монотонно неубывающая. Если принять $\alpha(n)$ за константу, то оценка для Linux 0.01 станет равной 26α, а для Linux 2.6.24 — равной 16α. Это уменьшение значения СИ является следствием уменьшения как общего количества идентификаторов подчинённой подсистемы в главной, так и тех из них, которые связаны (входят в один оператор).

ЛИТЕРАТУРА

1. *Landwehr C. E.* Formal models for computer security // ACM Comput. Surv. 1981. September. V. 13. No. 3. P. 247–278.
2. Security-Enhanced Linux / USA National Security Agency., Электрон. дан., 2009. Режим доступа: `http://www.nsa.gov/research/selinux/index.shtml`, свободный.
3. Grand Research Challenges in Information Systems / Computing Research Association., Электрон. дан., 2002. Режим доступа: `http://www.cra.org/reports/gc.systems.pdf`, свободный.
4. *Mills E. E.* Metrics in the software engineering curriculum // Ann. Softw. Eng. 1999. V. 6. No. 1–4. P. 181–200.

УДК 681.3.06:62-507

ТРАНСЛЯЦИЯ ОПИСАНИЙ АВТОМАТОВ, ПРЕДСТАВЛЕННЫХ В ФОРМАТЕ MICROSOFT VISIO, В ИСХОДНЫЙ КОД НА ЯЗЫКЕ C

Л. В. Столяров, И. Р. Дединский, А. А. Шалыто

Существует подход к созданию программ для систем логического управления с использованием конечных автоматов, названный в [1] автоматным программированием (АП). В последнее время этот подход быстро развивается [1, 2]. С его помощью удобно реализуются различные системы логического и событийного управления [3]. АП основывается на использовании конечных автоматов (КА) для описания логики работы программ. КА — это конечное множество состояний, в которых он может находиться, и переходов между этими состояниями. Подход основан на создании графов переходов КА (автоматных схем) [4] с последующей их трансляцией в исходный код программы. Автоматная схема является графическим отражением алгоритма в терминах состояний и переходов. Поэтому при разработке программ на основе АП автоматная схема