ОБНАРУЖЕНИЕ И РАЗРЕШЕНИЕ КОНФЛИКТОВ В ПОЛИТИКАХ БЕЗОПАСНОСТИ КОМПЬЮТЕРНЫХ СЕТЕЙ

А. В. Тишков[1], И. В. Котенко[2], О. В. Черватюк[3], Д. П. Лакомов[4], С. А. Резник[5], Е. В. Сидельникова[6]

Санкт-Петербургский институт информатики и автоматизации РАН

СПИИРАН, 14-я линия ВО, д. 39, Санкт-Петербург, 199178

[1]<avt@iias.spb.su>, [2]<ivkote@iias.spb.su>, [3]<ovch@computer.edu.ru>, [4]<dmll@computer.edu.ru>, [5]<sergeyreznick@yandex.ru>, [6]<KittyKate137@yandex.ru>

УДК 681.3

Тишков А. В., Котенко И. В., Черватюк О. В., Лакомов Д. П., Резник С. А., Сидельникова Е. В. **Обнаружение и разрешение конфликтов в политиках безопасности компьютерных сетей** // Труды СПИИРАН. Вып. 3, т. 2. — СПб.: Наука, 2006.

Аннотация. *Предлагается подход к построению системы верификации политик безопасности, предназначенной для обнаружения и разрешения конфликтов в спецификациях политик безопасности компьютерных сетей. Рассмотрена архитектура предлагаемой системы верификации политик безопасности. Представлены модели реализации двух модулей верификации: модуля, основанного на теории доказательств, с применением исчисления событий и абдуктивного вывода, и модуля, использующего технологию верификации на модели. Описана текущая реализация программного прототипа системы верификации.* — Библ. 18 назв.

UDC 681.3

Tishkov A. V., Kotenko I. V., Chervatuk O. V., Lakomov D. P., Reznick S. A., Sidelnikova E. V. **Conflict Detection and Resolution in Security Policies of Computer Networks** // SPIIRAS Proceedings. Issue 3, vol. 2. — SPb.: Nauka, 2006.

Abstract. *We consider an approach for constructing the security policy verification system intended for detection and resolution of conflicts in computer network security policy specifications. The architecture of the security policy verification system suggested is considered. The models of two verification modules are proposed. The first one is based on proof theory, namely Event Calculus, and uses abductive reasoning. The second module uses model checking technique. The current implementation of the security policy verification system is described.* — Bibl. 18 items.

1. Введение

Программные системы защиты компьютерных сетей, основанные на политиках безопасности, получают все большее распространение благодаря гибкости в управлении и удобству администрирования. Политика безопасности описывает множество правил функционирования системы, в том числе правил обеспечения конфиденциальности, целостности и доступности ресурсов системы. Важным преимуществом таких систем является консолидация правил политики в едином репозитории.

При поддержке больших компьютерных систем (сетей) актуальной становится задача создания непротиворечивой политики безопасности. Создавая множество правил различных типов и категорий безопасности, администратор системы не в состоянии без соответствующего инструментария отследить возможные конфликты между различными правилами политики, особенно в ситуации взаимного влияния правил разных категорий, а также возможность реализации этих правил при использовании заданной конфигурации компьютерной сети.

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

Исследуя опубликованные работы по обнаружению и разрешению конфликтов в политиках безопасности, мы столкнулись с определенными трудностями в их использовании. Часть работ достаточно четко задает понятие конфликта, но в контексте языка политики безопасности, предлагаемого в настоящей работе, эти конфликты определяются лишь для одной из категорий. Так, в работах [1–4] исследуется конфликт авторизации. В работах [5–7] определяется несколько типов конфликтов, которые связаны с авторизацией и обязательным выполнением, близким к понятию операционного правила. Изложенные в [8, 9] решения, основанные на темпоральной логике, требуют дополнительных понятий, таких как состояние системы и активное, пассивное и потенциальное состояние конфликта. Другие работы, определяющие понятие конфликта, очень сложно или даже невозможно вырвать из контекста вводимого в них языка или системы. Так, интересный подход, связанный с деонтической логикой [10–12], требует достаточно специфического определения роли и ввода иерерхии ролей.

Предлагаемая в настоящей работе система верификации политик безопасности компьютерных сетей является своеобразным «отладчиком» политики, предоставляя администратору инструмент обнаружения и разрешения конфликтов между правилами политики. Она ориентирована на разрешение конфликтов как в рамках одной категории (аутентификации, авторизации, фильтрации, конфиденциальности передачи информации и данных, операционных правил), так и разных категорий, а также обнаружения невозможности реализации правил политики при использовании определенной конфигурации компьютерной сети. Входом для системы верификации являются XML-спецификации конфигурации компьютерной сети, в которые входит определение топологии, сервисов и ресурсов узлов сети, и XML-спецификации политики безопасности, как набора правил. В качестве результата функционирования системы верификации должны формироваться согласованные спецификации сети и политики безопасности.

Работа структурирована следующим образом. Во втором разделе рассмотрена архитектура предлагаемой системы верификации политик безопасности. В третьем и четвертом разделе представлены модели реализации двух базовых модулей верификации: (1) основанном на теории доказательств, с применением исчисления событий и абдуктивного вывода, и (2) использующем технологию верификации на модели (model checking). В пятом разделе описана текущая реализация прототипа системы верификации. В заключении формулируются результаты работы и направления будущих исследований.

2. Архитектура системы верификации политик безопасности

Архитектура системы верификации политик безопасности является многомодульной. Специально выделенный менеджер верификации запускает модули верификации для проверки непротиворечивости политики и собирает от них информацию о найденных конфликтах и способах их разрешения. Модули верификации осуществляют эту проверку на основе различных математических методов. Ядро системы верификации политик безопасности содержит два основных класса: VerificationManager и VerificationModule.

Основными входными данными для системы верификации политик безопасности являются описание политики и системы. Для описания системы и по-

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

литик используется разработанный язык, представленный набором XML-схем и основанный на стандарте CIM.

В прототипе, на котором тестируется работа системы верификации политик безопасности, описания политики и системы принимаются в виде XML-файлов, валидных по отношению к их схемам. Политика представлена набором XML-документов и XML-схем, расположенных в отдельных подкаталогах в соответствии с категориями безопасности. Каждый подкаталог содержит XML-схему для соответствующей категории безопасности политики и набор файлов (XML-документов), представляющих правила этой категории. Описание системы также располагается в отдельном каталоге, содержащем XML-документ и XML-схему, реализующую язык описания системы.

Модули верификации транслируют XML-представление политик и системы в представление, необходимое для технологии, используемой каждым из модулей при верификации.

Помимо языка описания политик и языка описания системы в качестве входа можно рассматривать данные, необходимые для формальной модели того или иного модуля. Например, для модуля верификации ECVerificationModule необходима аксиоматика исчисления событий (Event Calculus).

Выходными данными работы системы верификации политик безопасности являются:

- информация об обнаруженном конфликте, включающая категорию конфликта;
- элементы компьютерной системы, корректная работа которых будет под угрозой в результате этого конфликта;
- правила, участвующие в конфликте;
- возможные стратегии разрешения.

3. Модуль верификации, основанный на исчислении событий

Исчисление событий является многосортной теорией предикатов первого порядка, впервые введенной Ковальским [13]. В основу исчисления событий заложен «принцип инерции» — свойства окружающего мира изменяются только под воздействием событий и остаются неизменными в промежутках между ними. Вводятся два основных сорта: свойство (fluent), представляющее некоторое изменяемое во времени состояние системы, и событие (event). В исчислении событий принцип инерции заключается в том, что свойство выполняется в некоторый момент времени, если оно было «инициировано» событием в более ранний момент времени и не было «остановлено» другим событием в промежутке между двумя этими моментами. Аналогично, свойство не выполняется в некоторый момент времени, если оно было «остановлено» до этого момента и не было «инициировано» в промежутке между этими двумя моментами.

Чтобы описать простейшее исчисление событий требуются следующие семь предикатов:

- **InitiallyTrue(f)** — свойство **f** выполняется в начальный момент времени;
- **InitiallyFalse(f)** — свойство **f** не выполняется в начальный момент времени;
- **Happens(e,t)** — событие **e** происходит в момент времени **t**;
- **Initiates(e,f,t)** — если **e** произошло в момент времени **t**, оно инициировало свойство **f**;

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

- **Terminates(e,f,t)** — если **e** произошло в момент времени **t**, оно терминировало свойство **f**;
- **HoldsAt(f,t)** — свойство **f** выполняется в момент времени **t**;
- **Clipped(t1,f,t2)** — свойство **f** было терминировано промежутке между моментами $\mathbf{t_1}$ и $\mathbf{t_2}$.

Для применения абдуктивного вывода [14–16] предметно-независимую аксиоматику удобно определять в форме тождеств:

- $\text{HoldsAt}(f,t) \equiv [\text{Happens}(e,t_1) \wedge \text{Initiates}(e,f,t_1) \wedge t_1 < t \wedge \neg\text{Clipped}(t_1,f,t)] \vee$ $\vee [\text{InitiallyTrue}(f) \wedge \neg\text{Clipped}(0,f,t)] \vee [t = 0 \wedge \text{InitiallyTrue}(f)]$;
- $\text{InitiallyTrue}(f) \equiv \neg\text{InitiallyFalse}(f)$;
- $\text{Clipped}(t_1,f,t_2) \equiv \text{Happens}(e,t) \wedge t_1 \le t < t_2 \wedge \text{Terminates}(e,f,t)$.

Для конкретной задачи следует ввести предметно-зависимую (или предметную) аксиоматику. В качестве примера работы абдуктивного вывода в системе верификации рассмотрим конфликты для правил авторизации и аутентификации.

Введем соответствующие свойства и события:

- Аутентификация:
  - Fluent: **Authenticated (?user, ?auth_type, ?target)**;
  - Event: **RequestAuthentication (?user, ?auth_type, ?target)**;
- Авторизация:
  - Fluents: **Authorized (?user, ?activity, ?target)**, **AllowAuthorization (?user, ?activity, ?target)**, **DenyAuthorization (?user, ?activity, ?target**);
  - Event: **RequestAuthorization (?user, ?activity, ?target)**.

Приведенные свойства определяют состояние пользователя (**?user**) — аутентифицирован ли он определенным узлом сети (**?target**), и при помощи какого метода (**?auth_type**), а также авторизован ли он на выполнение действий (**?activity**) над объектом (**?target**).

События, относящиеся к политикам авторизации или аутентификации, инициируют (либо останавливают) соответствующие свойства. Таким образом, получаем следующую предметную аксиоматику:

- $\text{Initiates}(\text{RequestAuthentication}(?user, \text{AuthenticationMethod}, \text{Target}),$ $\text{Authenticated}(?user, \text{AuthenticationMethod}, \text{Target}), ?t)$;
- $\text{HoldsAt}(\text{Authenticated}(?user, \text{AuthenticationMethod}, \text{Target}), ?t) \wedge$ $\wedge\ \textit{SubjectRole}(?user, \text{Permitting Role})] \rightarrow$ $\text{Initiates}(\text{RequestAuthorization}(?user, \text{Activity}, \text{Target}),$ $\text{Authorized}(?user, \text{Activity}, \text{Target}), ?t)$;
- $\text{HoldsAt}(\text{Authenticated}(?user, \text{AuthenticationMethod}, \text{Target}), ?t) \wedge$ $\wedge\ \text{SubjectRole}(?user, \text{Permitting Role}) \rightarrow$ $\text{Initiates}(\text{RequestAuthorization}(?user, \text{Activity}, \text{Target}),$ $\text{AllowAuthorization}(?user, \text{Activity}, \text{Target}), ?t)$;
- $\text{HoldsAt}(\text{Authenticated}(?user, \text{AuthenticationMethod}, \text{Target}), ?t) \wedge$ $\wedge\ \text{SubjectRole}(?user, \text{Forbidding Role}) \rightarrow$ $\text{Initiates}(\text{RequestAuthorization}(?user, \text{Activity}, \text{Target}),$ $\text{DenyAuthorization}(?user, \text{Activity}, \text{Target}), ?t)$.

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

Рассмотрим пример (табл. 1), в котором задаются правила аутентификации на один объект (`SMTP Server`), требующие различного типа аутентификации для одного субъекта (пользователя, принадлежащего роли `Administrator`).

Таблица 1

Правила примера 1

№	Объект	Действие	Субъект	Тип аутентификации	Тип политики
1	SMTP Server	Configure	Administrator	Account	Аутентификация, Авторизация
2	SMTP Server	Send, Relay	Administrator, Student, Professor, User	Shared secret	Аутентификация, Авторизация

Конфликт возникает, когда пользователь пытается аутентифицироваться при помощи обоих типов аутентификации. В представлении исчисления событий эти два правила выглядят следующим образом:

- $\mathrm{Initiates}(\mathrm{RequestAuthentication}(?user, \mathrm{Account}, \mathrm{SMTPServer}),$ $\mathrm{Authenticated}(?user, \mathrm{Account}, \mathrm{SMTPServer}), ?t)$;
- $\mathrm{Initiates}(\mathrm{RequestAuthentication}(?user, \mathrm{SharedSecret}, \mathrm{SMTPServer}),$ $\mathrm{Authenticated}(?user, \mathrm{SharedSecret}, \mathrm{SMTPServer}), ?t)$.

Введем предикат **AuthenticationConflict** для выявления конфликтующих условий пользователь-объект:

- $\mathrm{AuthenticationConflict}(?user, ?target) \equiv$ $\equiv \mathrm{HoldAt}(\mathrm{Authenticated}(?user, ?authType_1, ?target), ?t) \wedge$ $\wedge \mathrm{HoldsAt}(\mathrm{Authenticated}(?user, ?authType_2, ?target), ?t) \wedge$ $\wedge (?authType_1 \neq ?authType_2)$.

Рассмотрим пример конфликта авторизации. В таблице 2 определяются два правила авторизации — разрешающее и запрещающее — для конфигурации `SMTP Server`-а.

Таблица 2

Правила примера 2

№	Объект	Действие	Субъект	Allow/ Deny	Тип политики
1	SMTP Server	Configure	Administrator	Allow	Аутентификация Авторизация
2	SMTP Server	Configure	Internet User	Deny	Аутентификация Авторизация

Конфликт возникает, если пользователь принадлежит обеим ролям: `Administrator` и `InternetUser`. На языке исчисления событий данные правила формализуются следующим образом:

- $\mathrm{SubjectRole}(?user, \mathrm{Administrator}) \rightarrow$ $\mathrm{Initiates}(\mathrm{RequestAuthorazation}(?user, \mathrm{Configure}, \mathrm{SMTPServer}),$ $\mathrm{AllowAuthorization}(?user, \mathrm{Configure}, \mathrm{SMTPServer}), ?t)$;

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

- $\text{SubjectRole}(?user, \text{InternetUser}) \rightarrow$
  $\text{Initiates}(\text{RequestAuthorization}(?user, \text{Configure}, \text{SMTPServer}),$
  $\text{DenyAuthorization}(?user, \text{Configure}, \text{SMTPServer}), ?t)$.

Предикат **AutorizationConflict**, определяет тройки субъект-действие-объект, которым даны противоположные полномочия:

- $\text{AuthorizationConflict}(?user, ?activity, ?t\,\text{arget}) \equiv$
  $\equiv \text{HoldsAt}(\text{DenyAuthorization}(?user, ?activity, ?t\,\text{arget}), ?t) \wedge$
  $\wedge \text{HoldsAt}(\text{AllowAuthorization}(?user, ?activity, ?t\,\text{arget}), ?t)$.

Для обнаружения конфликта используется абдуктивный вывод. В общем случае, абдуктивная логическая программа представляет собой тройку $(\text{Th}, \text{IC}, Q)$, в которую входят **теория** Th, конечное множество **ограничений целостности** IC и запрос Q. Теорией называется множество так называемых `iff`-определений:

$$p(X_1, \ldots, X_k) \equiv D_1 \vee \ldots \vee D_k .$$

Предикат p не может быть специальным предикатом (сравнения, $=$, *TRUE* и *FALSE*). Каждый из дизъюнктов D_i является конъюнкцией литерал. Предикат p называется **определенным.** Предикат, который не является ни определенным, ни специальным называется **абдуктивно-выводимым**.

Ограничения целостности, из которых состоит множество *IC*, являются импликациями следующего вида:

$$L_1 \wedge \ldots \wedge L_m > A_1 \vee \ldots \vee A_n ,$$

где для любого i L_i является литералом, а A_i атомом

Запрос Q — это конъюнкция литералов. Можно ввести следующее определение абдуктивного метода: ответом на запрос Q соответствующей абдуктивной программы $(\text{Th}, \text{IC}, Q)$ называется пара (Δ, σ), где Δ — конечное множество абдуктивно-выводимых атомов, а σ — подстановка свободных переменных, встречающихся в Q, такие что

$$\text{Th} \vee \text{Comp}(\Delta) \models \text{IC} \vee Q_\sigma .$$

Рассмотрим первый пример из предыдущего раздела. Вход для этого примера будет состоять из шести следующих `iff`-определений:

```
clipped(T1,F,T2) iff
     [[happens(A,T), T1#<T, T#<T2, terminates(A,F,T)]].
holds_at(F,T) iff
     [[happens(A,T1), initiates(A,F,T1), T1#<T, not(clipped(T1,F,T))],
     [initially_true(F), not(clipped(0,F,T))],
     [T=0, initially_true(F)]].
initially_true(F) iff
     [[not(initially_false(F))]].
initially_false(F) iff
     [[F=authenticated(User,AType,Target)]].
initiates(E,F,T) iff
     [[E=request_authentication(User,account,smtp_server),
     F=authenticated(User,account,smtp_server)],
     [E=request_authentication(User,shared_secret,smtp_server),
     F=authenticated(User,shared_secret,smtp_server)]].
```


Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

Множество ограничений целостности в этом примере пусто. Запрос выглядит следующим образом:

```
[holds_at(authenticated (User,Auth_type_1,Target),T),
holds_at(authenticated (User, Auth_type_2,Target),T), Auth_type_1 \=
Auth_type_2]
```

Применение алгоритма CIFF [17] для данного примера выдает следующий результат:

```
[happens (request_authentication (_A, shared_secret, smtp_server),_B), hap-
pens(request_authentication (_A, account, smtp_server), _C)]:[_D/_A, Tar-
get/smtp_server, User/_A]:[_B#=<_C, _B#=<_C-1, _B#=<T-1, _C#=<T-1]
```

Этот ответ является сценарием, приводящим к конфликтной ситуации: конфликт выявлен.

4. Модуль верификации на модели

Методы проверки на модели основаны на переборе состояний, в которые может перейти система в зависимости от запросов пользователей и ответов компонента, принимающего решения о разрешении или отклонении такого запроса. Перебор управляется условиями, которые сформулированы на языке темпоральной логики и выражают корректные состояния системы. Состояние системы определяется набором значений переменных, а изменение состояния вызывается некоторыми параллельными процессами. При этом выбор процесса, который должен сделать шаг в очередной момент времени, происходит случайно. Система рассматривает все возможные последовательности шагов для заданных процессов, и сигнализирует о некорректном состоянии, если в таковое возможно прийти. Пользователю выдается трасса — последовательность шагов, приведшая к некорректному состоянию системы относительно заданных условий. Методы проверки на модели имеют существенное ограничение: рассматриваются лишь системы с конечным набором состояний (хотя сам моделируемый процесс может быть бесконечным). Для реализации соответствующего модуля верификации использовалось программное средство SPIN [18], использующее язык спецификации Promela.

SPIN представляет собой средство для верификации моделей динамических систем, поэтому его использование в верификации политик уместно в тех случаях, когда конфликты не могут быть выявлены путем статического анализа описаний политики и системы. Другими словами, речь идет о ситуации, когда конфликт происходит только при условии того, что выполнено некоторое динамическое предусловие.

В статье рассмотрен пример верификации политики, где возможен только один конфликт такого типа. Это конфликт типа «авторизация+аутентификация», который может произойти при выполнении следующих условий: есть два правила с совпадающей парой «объект–действие»; роли (объекты), используемые в этих правилах, различны; права доступа взаимно противоположны.

Рассматривается динамическое поведение пользователя. Предполагается, что пользователю можно присвоить несколько ролей. К политикам могут быть добавлены некоторые ограничения на присваивание ролей. Эти ограничения, равно как и собственно правила политик, должны быть отображены в Promela-описания, используемые в качестве входа для верификатора SPIN.

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

Для описания конфликта, который может случиться в условиях, когда какой-либо пользователь может присвоить роли из обоих правил, вовлеченных в вышеописанный конфликт, вводится понятие потенциального конфликта.

Потенциальные конфликты выявляются статически, и они включаются в Promela-модель.

Рассмотрим далее вводимые синтаксические конструкции этой модели для представления сущностей верифицируемой системы и политики.

Тип `PotentialConflict` в строке 1–3 листинга 1 описывает потенциально конфликтующие роли. Он независим от конкретной политики.

```
1    typedef PotentialConflict  {
2       mtype roles[2]
3    };
4    PotentialConflict conflicts[1];
5    chan req = [0] of {mtype};
6    mtype = {FTPWriteAllowedRole, FTPWriteDeniedRole};
7    bool userRoles[2];
```

Листинг 1. Определение типов данных.

Массив `conflicts` в строке 4 листинга 1 должен быть инициализирован потенциально конфликтующими парами ролей. Его размер равен количеству таких пар.

Канальный тип, используемый для взаимодействия между процессом, описывающим пользователя, и процессом, описывающим виртуальный сервис `RoleAssigner`. Он не зависит от политики. В строке 5 листинга 1 приводиться пример определения канального типа.

В строке 6 листинга 1 описывается тип, содержащий роли, вовлеченные в потенциальные конфликты. Массив, заданный в строке 7 листинга 1, представляет собой роли присвоенные пользователю. Его размер равен количеству ролей, вовлеченных в потенциальные конфликты.

Следующие структуры предназначены для инициализации и активации процессов, моделирующих работу верифицируемой системы с применением политики безопасности.

Часть модели в листинге 2 инициализирует массив конфликтов и запускает процессы `RoleAssigner` и `user`. Массив конфликтов заполняется ролями, вовлеченными в потенциальные конфликты.

```
init {
  printf("STARTED\n");
  d_step {
     conflicts[0].roles[0] = FTPWriteAllowedRole;
     conflicts[0].roles[1] = FTPWriteDeniedRole;
  }
  run RoleAssigner();
  run user();
}
```

Листинг 2. Процесс инициализации и активации основных процессов.

В листинге 3 пользовательский процесс случайным образом пытается присвоить две роли из списка потенциально конфликтующих. Успех такой попытки

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

зависит от поведения процесса RoleAssigner. Если нет ограничений на присваивание ролей, его поведение тривиально:

```
proctype user () {
  printf("user process started!\n");

  if
    :: req ! FTPWriteAllowedRole;
    :: req ! FTPWriteDeniedRole;
  fi;
  if
    :: req ! FTPWriteAllowedRole;
    :: req ! FTPWriteDeniedRole;
  fi
}
```

Листинг 3. Процесс, моделирующий динамическое назначение ролей пользователю.

Листинг 4 демонстрирует реализацию присвоения роли пользователям. Любой запрос на присвоение роли удовлетворяется. Если есть ограничение на присвоение ролей, то поведение RoleAssigner должно быть более сложным.

Возможны различные виды ограничений на присвоение ролей: приоритеты ролей, деактивация текущей роли при присвоении другой, запрет на одновременное присвоение некоторых ролей.

```
proctype RoleAssigner () {
  mtype r;

  printf("RoleAssigner process started!\n");
  do
    ::req ? r -> printf("received: %d\n", r);
                 userRoles[r-FTPWriteDeniedRole] = 1;
  od
}
```

Листинг 4. Метод назначения роли пользователю.

Следующий пример в листинге 5 демонстрирует разрешение обнаруженного конфликта путем введения метаправил, представляемых в Promela-модели усложнением процесса RoleAssigner.

Для этого дополнительно вводится условие, определяющее, что две роли не могут быть одновременно присвоены одному пользователю.

Если роль FTPWriteDeniedRole уже присвоена и пользователь запрашивает присвоение роли FTPWriteAllowedRole (или наоборот), то ничего не произойдет. В противном случае запрос будет удовлетворен.

```
proctype RoleAssigner () {
  mtype r;
  printf("RoleAssigner process started!\n");
  do
    :: req ? r ->
       printf("received: %d\n", r);
       if
         ::(r==Student && userRoles[FTPWriteDeniedRole-FTPWriteDeniedRole]
              == 1) -> skip;
```


Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

```
          ::(r==FTPWriteDeniedRole && userRoles[FTPWriteAllowedRole-
              FTPWriteDeniedRole] == 1) -> skip;
          ::else -> userRoles[r-FTPWriteDeniedRole] = 1;
        fi;
  od
}
```

Листинг 5. Разрешение конфликта введением дополнительных условий при назначении роли.

И последней синтаксической конструкцией, заданной в листинге 6, является оператор `never`, проверяющий возникновение конфликта.

```
#define p0 (userRoles[conflicts[0].roles[0]-FTPWriteDeniedRole] &&
userRoles[conflicts[0].roles[1]-FTPWriteDeniedRole])
never {    /* <>p */
  skip;
  skip;
  T0_init:
    if
      ::((p0)) ->
          printf("# of condition: 0\n");
          goto accept_all
      ::(1) -> goto T0_init
    fi;
  accept_all:
    skip;
}
```

Листинг 6. Структура, фиксирующая нарушение проверяемых свойств системы.

`SpinVerificationModule` реализован как Java-класс, способный разбирать правила, порождать соответствующие Promela-описания, подавать их на вход SPIN для порождения верификатора, компилировать верификатор, запускать его и анализировать его выход с тем, чтобы определить результат верификации.

В случае конфликта, обнаруженного путем верификации, порожденная SPIN трасса анализируется с тем, чтобы определить конфликтующие правила.В случае отсутствия конфликтов, сообщается об успешной верификации.

5. Реализация прототипа системы верификации

Разработан программный прототип системы верификации политики безопасности. Ядро системы представляет собой Java-библиотеку, на основе которой создано web-приложение и приложение, запускаемое на отдельной рабочей станции. Оба приложения поддерживают функциональность, кратко описанную в разделе 2:

- выбор политики безопасности для верификации;
- загрузка и запуск зарегистрированных модулей верификации;
- отчет о результатах верификации, содержащий перечень конфликтов (если найдены) и набор стратегий разрешения для каждого конфликта;
- разрешение конфликта при помощи применения одной из стратегий разрешения;
- просмотр журнала работы модуля верификации, включающего внутреннее представление политики в соответствии с формализмом, при-

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

меняемым конкретным модулем, и результаты верификации.

На рис. 1 представлены основные окна интерфейса приложения для отдельной рабочей станции.

В левой части рис. 1 изображены следующие окна:

- основное окно системы (вверху),
- окно результата верификации (посередине),
- окно выбора стратегии разрешения конфликта (внизу справа) и
- окно реализации одной из стратегий разрешения — деактивации конфликтующих правил вручную (внизу слева).

В правой части рисунка изображены журналы работы модулей верификации: для исчисления событий (вверху) и проверки на модели (внизу). В нижней части каждого из журналов выделены отчеты о результатах верификации.

6. Заключение

В настоящей статье представлен общий подход к созданию системы верификации политик безопасности компьютерных сетей. Предложена архитектура системы верификации и два подхода реализации механизмов обнаружения и разрешения противоречий в политиках безопасности.

Предложенные подходы верифицируют политики безопасности при помощи моделирования, однако каждый из них имеет свои особенности.

Методы проверки на модели основаны на переборе состояний, в которые может перейти система в зависимости от запросов пользователей и ответов компонента, принимающего решения о разрешении или отклонении такого запроса. Методы проверки на модели имеют существенное ограничение: рассматриваются лишь системы с конечным набором состояний.

Подход, основанный на исчислении событий, использует логический вывод для определения того, поддерживается ли некоторое свойство системы при условии возникновения определенной последовательности событий. Таким образом, если сформулировать конфликт как свойство системы и генерировать события случайно, интересуясь значениями свойства-конфликта, то работа такого верификатора аналогична работе модуля проверки на модели (уступая по скорости). Однако, при использовании исчисления событий возможно решение обратной задачи: абдуктивный вывод позволяет найти последовательность событий, которая приведет систему к конфликтному состоянию. В общем случае эта последовательность короче, чем трасса к некорректному состоянию, полученная при помощи средств проверки на модели.

В разработанном прототипе системы верификации данные подходы применены для обнаружения и разрешения политик аутентификации и авторизации. Прототип реализован на языке Java с использованием SICStus Prolog 3.12.5, CIFF 3.0, SPIN 4.2.6. В настоящее время ведутся работы по формализации конфликтов в правилах пяти категорий безопасности. В дополнение к аутентификации и авторизации рассматриваются правила фильтрации сетевого трафика, конфиденциальности данных и операционные правила.

Работа выполнена при финансовой поддержке РФФИ (проект №04-01-00167), программы фундаментальных исследований ОИТВС РАН (контракт №3.2/03) и при частичной финансовой поддержке, осуществляемой в рамках проекта Евросоюза POSITIF (контракт IST-2002-002314).

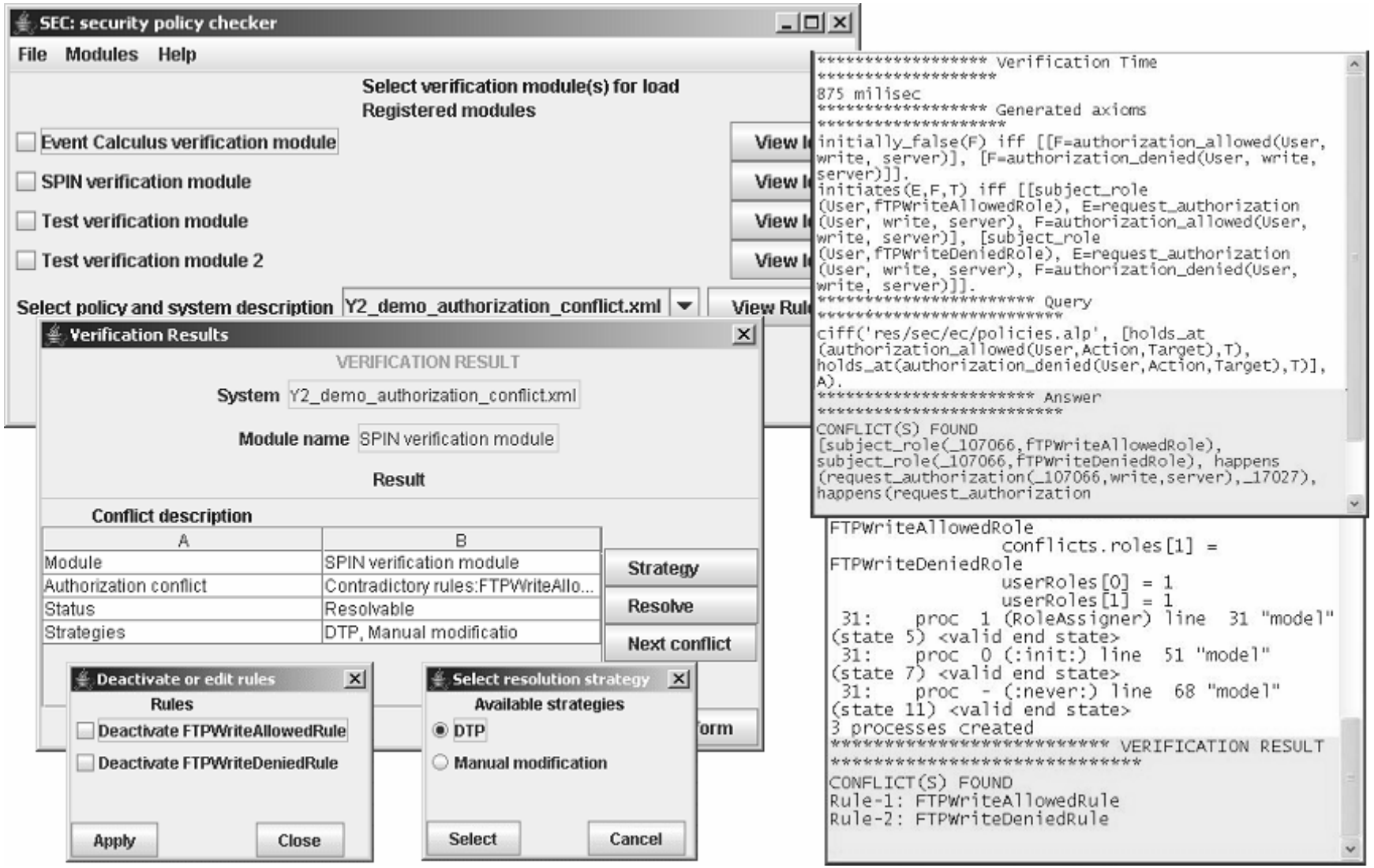

Рис. 1. Основные окна интерфейса системы верификации политики безопасности.

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

Литература

1. *De Capitani di Vimercati S., Paraboschi S., Samarati P.* Access Control: Principles and Solutions // Software — Practice and Experience. 2003. Vol. 33, no. 5. P. 397–421.
2. *Jajodia S., Samarati P., Sapino M. L., Subrahmanian V. S.* Flexible Support for Multiple Access Control Policies // ACM Trans. Database Systems. 2001. Vol. 26, no. 2. P. 214–260.
3. *Jajodia S., Samarati P., Subrahmanian V. S.* A Logical Language for Expressing Authorizations // IEEE Symposium on Security and Privacy. 1997. P. 31–42.
4. *Zhang Nan, Ryan Mark D., Guelev Dimitar* Evaluating Access Control Policies Through Model Checking // Eighth Information Security Conference (ISC'05). Lecture Notes in Computer Science. New York: Springer-Verlag, 2005. Vol. 3650. P. 446–460.
5. *Lupu E., Sloman M.* Conflict Analysis for Management Policies // Fifth IFIP/IEEE International Symposium on Integrated Network Management (IM'97), Chapman & Hall Publishers. San-Diego, 1997. P. 430–443.
6. *Lupu E., Sloman M.* Conflicts in Policy-based Distributed Systems Management // IEEE Transactions on Software Engineering — Special Issue on Inconsistency Management. Nov. 1999. Vol. 25, no. 6. P. 852–869.
7. *Lymberopoulos L., Lupu E., Sloman M.* Ponder Policy Implementation and Validation in a CIM and Differentiated Services Framework // IFIP/IEEE Network Operations and Management Symposium (NOMS 2004). Seoul, Korea. April 2004, Vol. 1. P. 31–44.
8. *Dunlop N., Indulska J., Raymond K.* Methods for Conflict Resolution in Policy-Based Management Systems // EDOC. 2003. P. 98–111.
9. *Dunlop N., Indulska J., Raymond K.* Dynamic Conflict Detection in Policy-Based Management Systems // EDOC. 2002. P. 15–26.
10. *Cholvy L., Cuppens F.* Solving Normative Conflicts by Merging Roles // Proceedings of the fifth International Conference on Artificial Intelligence and Law. Washington. May 1995. P. 201–209.
11. *Cholvy L., Cuppens F.* Analysing Consistency of Security Policies // Proceedings of IEEE Symposium on Security and Privacy. Oakland, USA. May 1997. P. 103–112.
12. Common Information Model (CIM) Standards [Электронный ресурс] // <http://www.dmtf.org/standards/cim> (по состоянию на 10.04.2006).
13. *Kowalski R. A, and Sergot M. J.* A Logic-Based Calculus of Events // New Generation Computing. 1986. no. 4. P. 67–95.
14. *Bandara A., Lupu E., Russo A.* Using Event Calculus to Formalize Policy Specifications and Analysis // IEEE 4th International Workshop on Policies for Distributed Systems and Networks. June 2003. P. 26–39.
15. *Kakas A. C., Kowalski R. A., Toni F.* Abductive Logic Programming // Journal of Logic and Computation. 1993. Vol. 2, no. 6. P. 719–770.
16. *Fung T. H., Kowalski R. A.* The IFF Proof Procedure for Abductive Logic Programming // Journal of Logic Programming. 1997. Vol. 33, no. 2. P. 151–165.
17. *Endriss U., Mancarella P., Sadri F., Terreni G., Toni F.* The CIFF Proof Procedure: Definition and Soundness Results // Technical Report 2004/2, Department of Computing, Imperial College London, May 2004. P. 31–43.
18. *Holzmann Gj.* The Spin Model Checker // IEEE Trans. on Software Engineering, May 1997. Vol. 23, no 5. P.279–295.