ВЕРИФИКАЦИЯ НА МОДЕЛИ В ЗАДАЧЕ ДИНАМИЧЕСКОГО ОБНАРУЖЕНИЯ КОНФЛИКТОВ В ПОЛИТИКЕ БЕЗОПАСНОСТИ КОМПЬЮТЕРНЫХ СЕТЕЙ

О. В. Черватюк, И. В. Котенко, А. В. Тишков (Санкт-Петербург)

Введение

При построении основанных на правилах систем безопасности больших вычислительных сетей важным вопросом является проверка непротиворечивости создаваемой политики. В условиях частой реконфигурации системы, периодической настройки политики и изменяющегося набора ролей и пользователей, задача поиска существующих и потенциальных конфликтов в правилах может быть эффективно решена при помощи моделирования поведения защищенной системы. В докладе предлагается два подхода к моделированию: на основе технологии верификации на моделях (model checking) [1] и с использованием методов доказательства теорем на базе исчисления событий [2].

Оба подхода осуществляют верификацию 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. Перебор управляется условиями, которые сформулированы на языке темпоральной логики и выражают корректные состояния системы. Состояние системы определяется набором значений переменных, а изменение состояния вызывается 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 [3].

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 [4] позволяет найти последовательность событий, которая приведет систему к конфликтному состоянию. В общем случае эта последовательность значительно короче, чем трасса к некорректному состоянию, полученная при помощи средств проверки на модели.

Архитектура системы

Общая архитектура системы безопасности вычислительных сетей, основанной на политиках, приведена на рис. 1.

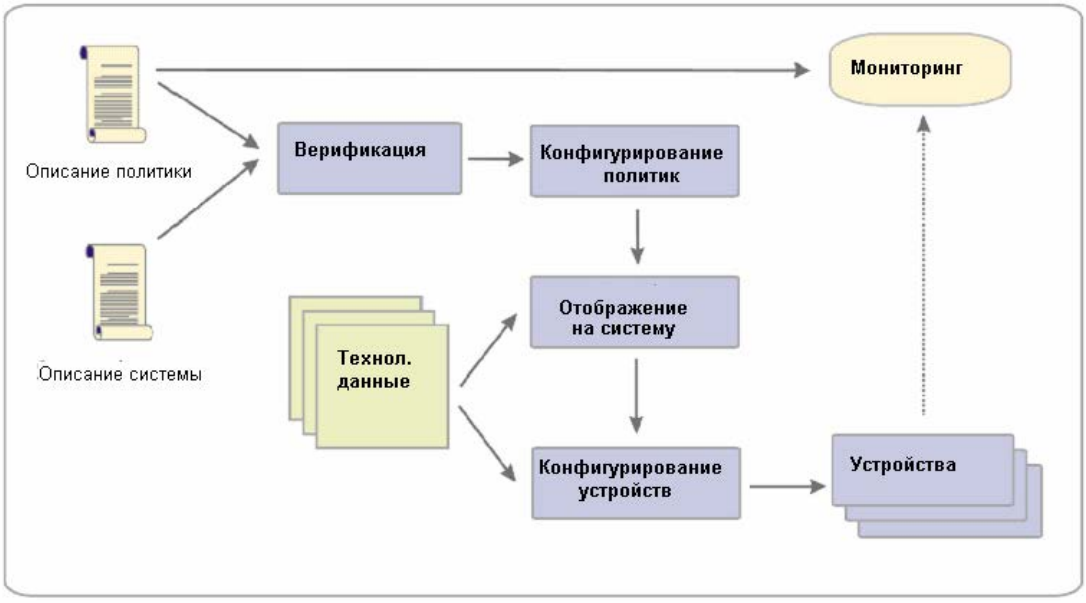

Рис. 1. Архитектура системы безопасности, основанная на политиках

Компонент верификации проверяет непротиворечивость правил политики, а также может ли определенная администратором политика применяться к заданному описанию системы. Верификатор играет роль отладчика правил политики, сообщая администратору о противоречиях и указывая фрагмент описания системы, в котором они возникли. *Компонент конфигурирования политик* распределяет и преобразует правила политики в зависимости от категорий устройств, которые будут эти правила поддерживать. Компонент отображения преобразованной политики на систему трансформирует полученные правила в соответствии с языком конфигурации конкретного типа устройства, используя сведения о таком языке, поставляемые производителем устройства. Последним компонентом, участвующим в процессе внедрения правил политики, является компонент конфигурирования устройств, который рассылает информацию, полученную на предыдущем шаге, по сети. *Проактивный монитор безопасности* является средством обнаружения уязвимостей и вторжений, с одной стороны, анализируя сетевой трафик, проверяя его соответствие заданной политике, а с другой стороны, обнаруживая потенциальные уязвимости.

Компонент верификации решает три основные задачи: 1) проверка непротиворечивости правил политики безопасности; 2) проверка применимости политики к описанию системы; 3) вычисление уровня защищенности сети. Для решения первых двух задач используется моделирование поведения системы.

Входные языки описания системы (ЯОС) и политик (ЯОП) определены с помощью XML-схем. ЯОС позволяет задать элементы сети, связи между ними, а также сервисы, доступные извне для каждого элемента. Идентифицирующими сведениями элемента являются его сетевые интерфейсы, IP-адрес, маска подсети. Для сервисов задается имя приложения, порт, протокол и точка доступа в виде адреса. Для каждого сервиса определяется список действий, которые можно над ним произвести (по умолчанию – запуск и остановка). Определяется точка доступа к сети. Дополнительно ЯОС включает факты запуска правил политик, такие как «пользователь *U* аутентифицирован», «обнаружен вирус *V*» и др.

ЯОП позволяет определять правила для нескольких категорий безопасности: аутентификации, авторизации, конфиденциальности, фильтрации, защиты от вторжений и операционных правил. Последняя категория включает правила вида условие-

действие, в посылке которых задается паттерн состояния системы, а в заключение определяются действия над сервисами элементов сети.

Компонент верификации является многомодульным, управление запуском модулей осуществляет менеджер верификации. Архитектура компонента открыта, т. е., возможно подключение внешних модулей верификации.

В качестве иллюстрации работы модулей, основанных на проверке на модели и на исчислении событий, рассмотрим пример конфликта авторизации. Конфликт возникает в том случае, когда один из пользователей приписывается к двум ролям, имеющим противоречивые привилегии на одно и то же действие: для одной роли разрешение, для другой – запрет.

Моделирование конфликта с помощью проверки на модели

В настоящем пункте описана реализация моделирования конфликта авторизации при помощи SPIN. Ключевыми алгоритмами программы являются два процесса. Первый процесс случайным образом назначает и удаляет принадлежность пользователя к одной из двух ролей: R1 и R2. Следующий код соответствует приписыванию пользователя к роли:

```
active proctype userRoleAssignment()
{
…
    :: (r.q<max_q_roles-1)->
          atomic      {
                r.q++;
                if
                ::r.ar[r.q]=R1;
                ::r.ar[r.q]=R2;
                fi
                }
…
```

Пользователь в случайные моменты времени посылает запросы на печать. Процедура IsAssigned проверяет принадлежность пользователя к заданной роли. Следующий код, при получении запроса на печать, присваивает значение true переменной deny (если пользователь в текущий момент принадлежит роли R1) или переменной allow (если пользователь принадлежит роли R2):

```
::smtp_server_chan_in?action,rr-> atomic
                {
                deny=false;
                allow=false
                IsAssigned(rr,R1,R1Res);
                IsAssigned (rr,R2,R2Res);
                if
                ::R1Res->deny=true
                ::else
                fi
                if
                ::R2->allow=true
                ::else
                fi
…
```

Возникновение конфликта заключается в нарушении следующего условия корректности состояния системы: allow и deny не могут выполняться одновременно:

```
assert((allow && !(deny)) || (!(allow) && deny))
```

Моделирование конфликта в исчислении событий

Исчисление событий – теория, предназначенная для описания свойств (fluents) некоторой системы и событий (events), изменяющих эти свойства. Теория основана на «принципе инерции»: «если ничего не произошло, то ничего не изменилось», т. е., только события могут изменять свойства системы. Теория исчисления событий основана на логике предикатов первого порядка, включающей дополнительные сорта для свойств, событий и времени (целые или вещественные числа). Состояние системы описывается при помощи предиката *HoldsAt(f)*, параметром которого является некоторое свойство системы. Этот предикат истинен, если данное свойство выполнено. Возникновение событий описывает предикат *Happens(e,t)*, он истинен, если событие *e* произошло в момент времени *t*. Предикаты *Terminates(e,f,t)* и *Initiates(e,f,t)* служат для выражения факта прекращения или начала действия свойства *f* соответственно при возникновении события *e* в момент времени *t*. Последние два предиката определяются предметной аксиоматикой, в отличие от предиката *HoldsAt*, истинность которого выводится из предметных аксиом и аксиом исчисления событий.

Аксиоматика исчисления событий (EC1–EC7), используемая для большинства приложений, приведена ниже. Описание других аксиоматик можно найти в [5].

$$Clipped\ (t_1,f,t_2) \leftarrow Happens(a,t_1)\ \&\ t_1 \le t < t_2\ \&\ Terminates\ (f,t_2) \qquad (EC1)$$

$$Declipped\ (t_1,f,t_2) \leftarrow Happens(a,t_1)\ \&\ t_1 \le t < t_2\ \&\ Initiates(f,t_2) \qquad (EC2)$$

$$HoldsAt\ (f, t_2) \leftarrow Happens(a,t_1)\ \&\ Initiates(a,f,t_1)\ \&\ t_1 < t_2\ \&\ \neg Clipped\ (t_1,f,t_2) \qquad (EC3)$$

$$\neg HoldsAt(f, t_2) \leftarrow Happens(a,t_1)\ \&\ Terminates(a,f,t_1)\ \&\ t_1 < t_2\ \&\ \neg Declipped(t_1,f,t_2) \qquad (EC4)$$

$$HoldsAt\ (f, t) \leftarrow InitiallyTrue(f)\ \&\ \neg Clipped(0,f,t) \qquad (EC5)$$

$$\neg HoldsAt\ (f,t) \leftarrow InitiallyFalse(f)\ \&\ \neg Declipped(0,f,t) \qquad (EC6)$$

$$InitiallyTrue(f)\ |\ InitiallyFalse(f) \qquad (EC7)$$

Для моделирования конфликта авторизации определяются следующие предикаты: *User(r) – r* является пользователем; *Action(a) – a* является действием, *Role(r) – r* является ролью. Вводится предикат *ContradictoryRoles(r_1,r_2,t,a)*, истинный в том случае, если роли r_1 и r_2 имеют противоположные привилегии относительно действия *a* в момент времени *t*. Вводятся следующие события: *AssignmentRequest(u,r)* – запрос для пользователя *u* на принадлежность роли *r*; *RolePermitActivity(r,a)* – действие *a* разрешено для роли *r*; *RoleDenyActivity(r,a)* – действие *a* запрещено для роли *r*. Определяются следующие свойства: *Assigned(u,r)* пользователь *u* приписан к роли *r*; *RoleHavePermission(r,a)* – пользователи роли *r* имеют разрешение на выполнение действия *a*; AuthorizationConflict(r_1,r_2) – возник конфликт авторизации для ролей r_1 и r_2.

Предметные аксиомы, описывающие разрешения для ролей и приписывание пользователя к роли следующие:

$$Initiates(RoleHavePermission(r,a), RolePermitActivity(r,a),t) \leftarrow$$
$$Happens(RolePermitActivity(r,a),t)\ \&\ (\neg HoldsAt(RoleHavePermission(r,a),t)) \qquad (AC1)$$
$$Terminates(RoleHavePermission(r,a), RoleDenyActivity(r,a),t) \leftarrow$$
$$Happens(RoleDenyActivity(r,a),t)\ \&\ HoldsAt(RoleHavePermission(r,a),t) \qquad (AC2)$$
$$Initiates(Assigned\ (u,r), AssignmentRequest(u,r),t) \leftarrow$$
$$Happens(AssignmentRequest(u,r),t)\ \&\ (\neg HoldsAt(AuthorizationConflict(r,r_1),t)) \qquad (AC3)$$

Четвертая предметная аксиома определяет конфликтующие роли:

ContradictoryRoles (r_1,r_2,t,a) ← (HoldsAt(RoleHavePermission(r_1,a),t) & (¬HoldsAt(RoleHavePermission(r_2,a),t))) | (HoldsAt(RoleHavePermission(r_2,a),t) & (¬HoldsAt (RoleHavePermission (r_1,a),t))) (AC4)

Теперь конфликт авторизации определяется как запрос к одной из конфликтующих ролей при том, что пользователь уже приписан ко второй:

Happens(conflictEvent,t) & Initiates (AuthorizationConflict (r, r_1), conflictEvent, t) ← HoldsAt(Assigned (u, r_1), t) & Happens(AssignmentRequest(r, u), t) & ContradictoryRoles (r, r_1, a, t) (AC5)

Выводы

В настоящей работе были представлены на примерах два метода моделирования поведения системы безопасности вычислительных сетей, основанной на политиках. Несмотря на то, что оба метода выполняют близкие задачи, они не конкурируют, а скорее дополняют друг друга. В рамках настоящего проекта видится целесообразным создание репозитория описаний конфликтов политик и применение обоих методов для обнаружения и разрешения конфликтов. Для реализации модулей верификации выбраны достаточно мощные и при этом свободно распространяемые программные средства. Проверка на модели осуществляется при помощи программного средства SPIN. Модуль для исчисления высказываний реализован на Jess, программном средстве доказательства теорем, поддерживающем язык Java.

Работа выполнена при поддержке «Фонда содействия отечественной науке», РФФИ (проект № 04-01-00167), программы фундаментальных исследований ОИТВС РАН (контракт № 3.2/03) и программы FP6 Европейского Сообщества, как часть проекта POSITIF (контракт № IST-2002-002314).

Литература

1. **Кларк Э. М., Грамберг О., Пелед Д.** Верификация моделей программ: Model Checking. – М.: МЦНМО, 2002. – 416 с.
2. **Kowalski R. A., Sergot M. J.** A Logic-Based Calculus of Events//New Generation Computing. 1986. 4. P. 67–95.
3. **Holzmann G. J.** The Spin Model Checker//IEEE Trans. on Software Engineering, 23(5):279–295, May 1997.
4. **Russo A., Miller R., Nuseibeh B. and Kramer J.** "An Abductive Approach for Analysing Event-Based Requirements Specifications//18th Int. Conf. on Logic Programming (ICLP), Copenhagen, Denmark, 2002.
5. **Miller R. and Shanahan M.** The Event Calculus in Classical Logic//Linkoping Electronic Articles in Computer and Information Science, 4(16), 1999.