DOI: 10.15514/ISPRAS-2020-32(1)-1

CC BY

Интеграция мандатного и ролевого управления доступом и мандатного контроля целостности в верифицированной иерархической модели безопасности операционной системы

[1] *П.Н. Девянин, ORCID: 0000-0003-2561-794X <devyanin.peter@yandex.ru>*
[2,3,4] *В.В. Кулямин, ORCID: 0000-0003-3439-9534 <kuliamin@ispras.ru>*
[2,3,4] *А.К. Петренко, ORCID: 0000-0001-7411-3831 <petrenko@ispras.ru>*
[2,3,4,5] *А.В. Хорошилов, ORCID: 0000-0002-6512-4632 <khoroshilov@ispras.ru>*
[2] *И.В. Щепетков, ORCID: 0000-0002-5794-004X <shchepetkov@ispras.ru>*

[1] *РусБИТех,*
117105, Россия, Москва, Варшавское шоссе, д. 26, стр.11
[2] *Институт системного программирования имени В.П. Иванникова РАН,*
109004, Россия, г. Москва, ул. А. Солженицына, д. 25
[3] *Московский государственный университет имени М.В. Ломоносова,*
119991, Россия, Москва, Ленинские горы, д. 1
[4] *НИУ Высшая школа экономики,*
101978, Россия, г. Москва, ул. Мясницкая, д. 20
[5] *Московский физико-технический институт,*
141701, Россия, Московская область, г. Долгопрудный, Институтский пер., 9

Аннотация. Проектирование механизма управления доступом в операционной системе (ОС), требующей высокого уровня доверия, является сложной задачей. Еще более сложной она стан овится, если требуется интеграция нескольких разнородных механизмов, таких как ролевое управление доступом (role-based access control, RBAC) и мандатное управление доступом (mandatory access control, MAC). Данная статья представляет результаты разработки иерархической интегрированной модели управления доступом и информационных потоков (hierarchical integrated model of access control and information flows, HIMACF), интегрирующей механизмы RBAC, MAC и мандатного контроля целостности (mandatory integrity control, MIC) с сохранением их ключевых свойств безопасности. Эта модель является дальнейшим развитием МРОСЛ-ДП-модели, она формализована на языке Event-B и ее корректность доказана формально. В иерархическом представлении модели каждый ее уровень (или модуль) представляет отдельный механизм управления доступом, благодаря чему модель может быть верифицирована помодульно, что снижает общую трудоемкость ее верификации за счет переиспользования при доказательстве корректности очередного уровня результатов верификации нижележащих уровней. Данная модель реализована в ОС Astra Linux Special Edition на основе инфраструктуры Linux Security Modules (LSM).

Ключевые слова: формальная модель управления доступом; управление доступом на основе ролей; мандатный контроль целостности; мандатное управление доступом; формальная верификация модели; Event-B; Astra Linux

Для цитирования: Девянин П.Н., Кулямин В.В., Петренко А.К., Хорошилов А.В., Щепетков И.В. Интеграция мандатного и ролевого управления доступом и мандатного контроля целостности в верифицированной иерархической модели безопасности операционной системы. Труды ИСП РАН, том 32, вып. 1, 2020 г., стр. 7-26. DOI: 10.15514/ISPRAS–2020–32(1)–1

Благодарности. Данная работа поддержана грантом РФФИ по проекту 18-01-00378.

Integrating RBAC, MIC, and MLS in Verified Hierarchical Security Model for Operating System

[1] *P.N. Devyanin, ORCID: 0000-0003-2561-794X <devyanin.peter@yandex.ru>*
[2,3,4] *V.V. Kuliamin, ORCID: 0000-0003-3439-9534 <kuliamin@ispras.ru>*
[2,3,4] *A.K. Petrenko, ORCID: 0000-0001-7411-3831 <petrenko@ispras.ru>*
[2,3,4,5] *A.V. Khoroshilov, ORCID: 0000-0002-6512-4632 <khoroshilov@ispras.ru>*
[2] *I.V. Shchepetkov, ORCID: 0000-0002-5794-004X <shchepetkov@ispras.ru>*

[1] *RusBITech*
26, Warsaw highway, Moscow, 117105, Russia
[2] *Ivannikov Institute for System Programming of the Russian Academy of Sciences,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia
[3] *Lomonosov Moscow State University,*
GSP-1, Leninskie Gory, Moscow, 119991, Russia
[4] *National Research University, Higher School of Economics*
20, Myasnitskaya Ulitsa, Moscow, 101978, Russia
[5] *Moscow Institute of Physics and Technology (State University),*
9 Institutskiy per., Dolgoprudny, Moscow Region, 141701, Russian Federation

Abstract. Designing a trusted access control mechanism of an operating system (OS) is a complex task if the goal is to achieve high level of security assurance and guarantees of unwanted information flows absence. Even more complex it becomes when the integration of several heterogeneous mechanisms, like role-based access control (RBAC), mandatory integrity control (MIC), and multi-level security (MLS) is considered. This paper presents results of developement of a hierarchical integrated model of access control and information flows (HIMACF), which provides a holistic integration of RBAC, MIC, and MLS preserving key security properties of all those mechanisms. Previous version of this model is called MROSL~DP-model. Now the model is formalized using Event-B formal method and its correctness is formally verified. In the hierarchical representation of the model, each hierarchy level (module) corresponds to a separate security control mechanism, so the model can be verified with less effort reusing the results of verification of lower level modules. The model is implemented in a Linux-based operating system using the Linux Security Modules infrastructure.

Keywords: Formal access control model; role-based access control; mandatory integrity control; multi-level security; formal verification; Event-B; Astra Linux

For citation: Devyanin P.N., Kuliamin V.V., Petrenko A.K., Khoroshilov A.V., Shchepetkov I.V. Integrating RBAC, MIC, and MLS in Verified Hierarchical Security Model for Operating System. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 1, 2020. pp. 7-26 (in Russian). DOI: 10.15514/ISPRAS–2020–32(1)–1

Acknowledgments. This study is supported by the Russian Foundation for Basic Research grant #18-01-00378.

1. Введение

Современные операционные системы (ОС) управляют достаточно сложно организованными массивами информации, выполняют множество гетерогенных функций и часто должны иметь дело с разветвленными иерархиями прав на выполнение разнообразных действий. В таком окружении использование обычного *дискреционного управления доступом* (discretionary access control, DAC) становится неудобным и требует чрезвычайно сложных действий для осуществления строгого контроля за потоками информации. Более гибким и масштабируемым с точки зрения администрирования механизмом является *управление доступом на основе ролей* (role-based access control, RBAC) [1]. Более строгие гарантии изоляции информации с разными правами доступа предоставляет *мандатное управление доступом* (mandatory access control, MAC). Еще

одной важной техникой изоляции информации и предотвращения ущерба от успешных атак является *мандатный контроль целостности* (mandatory integrity control, MIC).

Все три указанных механизма в различных комбинациях используются в современных ОС. RBAC и MAC (в виде так называемой многоуровневой защиты, multi-level security, MLS) могут быть прямо реализованы в рамках широко известной инфраструктуры защиты ОС Linux, SELinux [2]. Эта инфраструктура используется, например, в мобильной ОС Android, начиная с версии 5.0 [3]. Реализация многоуровневого MIC имеется в ОС Windows, начиная с Windows Vista [4]. Специфическая реализация контроля целостности поддерживается в macOS [5].

Каждый из этих механизмов имеет формальную модель, задающую его строгую семантику и обеспечиваемые свойства безопасности. Для MAC такой моделью является знаменитая модель Белла-ЛаПадулы (D.E. Bell, L.J. LaPadula, BLP) [6], для MIC – модель Биба (K.J. Biba) [7]. Разнообразные модели RBAC описаны в работах Сандху (R.S. Sandhu) с соавторами [1].

Однако, в опубликованных работах, кроме публикаций о предыдущих версиях HIMACF, не было представлено формальной модели безопасности, явным образом объединяющей механизмы RBAC, MAC и MIC, с сохранением их ключевых свойств безопасности, подходящей для реализации в рамках ОС. Работа [8], например, описывает, как смоделировать MAC и MIC с помощью механизма RBAC, пополненного специальными ограничениями, однако представленная в ней техника достаточно неудобна для администрирования в терминах уровней целостности или конфиденциальности. Классические простые модели недостаточны как формальное представление такого интегрированного механизма, поскольку современные ОС управляют большим разнообразием объектов с различными требованиями к ограничениям доступа к ним. Простое механическое объединение нескольких разнородных механизмов может приводить к нарушению их ключевых свойств безопасности и возникновению нежелательных потоков информации, например, использование специальных ролей для доступа к высококонфиденциальной информации может привести к ее утечке, если сами эти роли доступны для анализа пользователям с низкой конфиденциальностью.

В данной статье представлены текущие результаты разработки и верификации формальной модели, называемой иерархической интегрированной моделью управления доступом и информационных потоков (hierarchical integrated model of access control and information flows, HIMACF, ранние версии ее имели название МРОСЛ ДП-модель). [9-10]. Эта модель корректно интегрирует механизмы RBAC, MIC и MAC, сохраняя их ключевые свойства безопасности. Под ключевыми свойствами безопасности имеются в виду базовые свойства, обеспечиваемые проектными решениями в рамках рассматриваемого механизма, например, возможность доступа субъекта к объекту только через подходящую роль в RBAC, или возможность чтения субъектом объекта только при наличии у первого более высокого или равного уровня доступа в MAC. Наша модель также поддерживает тонко настраиваемое управление доступом к данным сложной структуры, которые встречаются в современных ОС. Она имеет описание на языке Event-B [11] и формально верифицирована с помощью поддерживающей этот язык среды Rodin [12]. Все ее свойства безопасности представлены как инварианты модели и строго доказаны. Представление модели иерархично, она состоит из нескольких модулей, уточняющих друг друга, причем каждый механизм описан в отдельном модуле. Такая структура позволяет представить свойства отдельного механизма в наиболее ясном виде и облегчает верификацию модели за счет того, что свойства, верифицированные в одном модуле, переиспользуются в уточняющих его модулях (не требуется их повторное доказательство). Представленная модель предназначена для использования как модель политик безопасности ОС, наличие

которой требуется стандартом Common Criteria [13-14] и нормативными документами ФСТЭК России для систем с высокими уровнями доверия. Модель частично реализована в построенной на основе ядра Linux ОС Astra Linux Special Edition [15]. Хотя в настоящее время формальное соответствие между моделью и ее реализацией в коде не может быть строго доказано, мы продолжаем исследования, нацеленные на достижение этой цели.

Статья организована следующим образом. В следующем разделе рассматриваются механизмы RBAC, MAC и MIC, их формальные модели, а также некоторые модификации, представленные в литературе. Общее описание HIMACF дано в разд. 3. Детальное формальное представление некоторых элементов модели представлено в разд. 4. Разд. 5 описывает особенности верификации модели, а заключение завершает статью и представляет некоторые дальнейшие расширения и доработки модели.

2. Базовые механизмы

В данном разделе мы напоминаем основные свойства RBAC, MIC, и MAC, а также формализующие их модели. Некоторые понятия и обозначения из этих моделей используются нами в модели HIMACF далее.

Далее процессы в ОС, способные получать доступ к данным, называются *субъектами* или *сессиями* (предпочтение отдается термину, используемому в оригинальном изложении соответствующей модели). Множество субъектов обозначается S. Данные, к которым может быть получен доступ, считаются размещенными в *объектах*, или *сущностях* системы, множество всех объектов обозначается O. Все рассматриваемые далее модели накладывают определенные ограничения на возможные виды доступа субъектов к объектам.

2.1 Модели управления доступом на основе ролей

Управление доступом на основе ролей [1] ограничивает доступ сессий к объектам с помощью *ролей* R, приписываемых сессиям, и *прав доступа* P, определяющих возможные доступы к объектам (на чтение, запись, исполнение или др. определенного объекта). Каждая роль имеет множество прав доступа, задаваемое отношением $PA \subseteq R \times P$. Пользователи системы образуют множество U. Каждый пользователь имеет множество ролей, задаваемое отношением $UA \subseteq U \times R$. Каждая сессия приписана к определенному пользователю с помощью функции $user: S \to U$ и имеет множество ролей, задаваемое отображением $roles: S \to 2^R$. На множестве ролей задан частичный порядок $RH \subseteq R \times R$, определяющий наследование ролей, роль r *наследует* r' тогда и только тогда, когда $(r, r') \in RH$. Иерархия ролей позволяет более удобно организовать права доступа сессий. Множество ролей сессии ограничено множеством ролей ее пользователя и теми, которые наследуются ими, $s \in S \Rightarrow roles(s) \subseteq \{r \mid \exists r'(r, r') \in RH \wedge (user(s), r') \in UA\}$. Состояние системы описывается кортежем $(U, R, P, S, UA, PA, RH, user, roles)$.

Модель ARBAC [18] расширяет RBAC при помощи *административных ролей* AR, которые могут иметь права на модификацию отношений UA, PA и RH. Множество административных прав обозначается AP. Предполагается, что административные и обычные права, так же, как административные и обычные роли, не пересекаются $AR \cap R = AP \cap P = \emptyset$. Привязка прав к административным ролям обозначается $APA \subseteq AR \times AP$. Привязка административных ролей к пользователям задается отношением $AUA \subseteq U \times AR$. На административных ролях также определена иерархия $ARH \subseteq AR \times AR$, являющаяся частичным порядком. Привязка ролей к сессиям переопределяется как $roles: S \to 2^{R \cup AR}$, и ограничение выглядит так: $s \in S \Rightarrow roles(s) \subseteq \{r \mid \exists r'(r, r') \in RH \cup ARH \wedge (user(s), r') \in UA \cup AUA\}$. Состояние системы задается кортежем $(U, R, AR, P, AP, S, UA, \$\$AUA, PA, APA, RH, ARH, user, roles)$.

Более детальная модель RBAC, включающая ограничения на возможные переходы между состояниями системы, описана стандартом ANSI RBAC [16].

2.2 Модель Белла-ЛаПадулы

Первая и наиболее известная модель MAC предложена Беллом и ЛаПадулой [6, 17]. В ней имеются множества субъектов S, объектов O и атрибутов доступа A. Объекты могут участвовать в отношении иерархии H. Для управления доступом используются *уровни защиты* L, имеющие частичный порядок на них. Уровень защиты объекта задается функцией $f_o: O \to L$, максимальный уровень доступа субъекта задается функцией $f_s: S \to L$, а текущий уровень доступа субъекта задан с помощью функции $f_c: S \to L$. Предполагается, что $f_c(s) \leq f_s(s)$ для всех $s \in S$. Права доступа определены с помощью матрицы прав доступа $M: S \times O \to 2^A$. Текущие доступы субъектов к объектам заданы отношением $B \subseteq S \times O \times A$. Состояние системы моделируется как (B, M, f_s, f_c, f_o, H) и называется безопасным, если выполнены следующие правила.

- Правило простой безопасности (ss-правило) требует, чтобы субъект s имел доступ на чтение к объекту o, только если $f_s(s) \geq f_o(o)$. Для доступа на запись (рассматриваемого здесь как возможность и читать, и модифицировать информацию), это тоже должно быть выполнено.
- *-правило требует, чтобы субъект s мог иметь доступ на чтение к объекту o, только если $f_c(s) \geq f_o(o)$, и s мог иметь доступ на модификацию (без чтения) к o, только если $f_c(s) \leq f_o(o)$. Для получения доступа на запись (чтение с модификацией), требуется $f_c(s) = f_o(o)$.
- Правило дискреционной безопасности (ds-правило) требует, чтобы субъект s мог иметь доступ a к объекту o, только если $a \in M(s, o\}$

Система называется безопасной, если все ее достижимые состояния безопасны. В работе [6] даны ограничения на переходы между состояниями, достаточные для того, чтобы система с безопасным начальным состоянием была безопасной.

2.3 Модель Биба

Наиболее известной моделью MIC является модель Биба [7]. Она использует упорядоченные уровни целостности I и функцию $i: O \cup S \to I$, приписывающую уровень целостности каждому объекту и субъекту системы. Модель рассматривает различные политики, ограничивающие возможные переходы между состояниями системы. Широко известна политика строгой целостности, требующая сохранения уровней целостности объектов и субъектов, а также выполнения следующих правил.

1. Субъект s может иметь доступ на чтение к объекту o, только если $i(s) \leq i(o)$.
2. Субъект s может иметь доступ на модификацию к объекту o, только если $i(o) \leq i(s)$.
3. Субъект s может иметь доступ на обращение (управление) к другому субъекту s', только если $i(s') \leq i(s)$.

В современных ОС обычно реализуется кольцевая политика, требующая выполнения правил 2 и 3.

2.4 Детализация ограничений MAC и анализ информационных потоков

В этом подразделе дается краткий обзор развития и детализации моделей мандатного управления доступом, которые делают этот механизм более практичным при использовании в современных ОС, управляющих сложными данными. Некоторые

элементы этих моделей использованы в HIMACF. Сущностями далее называются субъекты и объекты системы.

Распространение информации при работе системы недостаточно аккуратно отражается только операциями по контролю доступа. Для более точного его моделирования можно использовать понятие *информационных потоков*, в том виде, как это сделано в известной модели Take-Grant (TGM) [18-19]. Эта модель рассматривает непрямой доступ субъектов к информации через объекты и другие субъекты, который возникает в результате последовательных чтений и модификаций. За счет этого TGM полезна для анализа распространения информации и ее возможного перехода между уровнями защиты. В TGM обычные ограничения контроля доступа (как ss-правило или *-правило) названы *правилами de jure*, и они дополнены правилами *de facto*, описывающими распространение информационных потоков. Информационный поток на запись от сущности x к сущности y обозначает любой возможный способ распространения информации от x к y, например, если субъект x получает доступ на запись к y, или субъект y получает доступ на чтение к x, или другой субъект z имеет доступ на чтение к x и доступ на запись y, возможно, перенося часть данных из x к y, и т.п. Наличие информационного потока на запись от x к y эквивалентно наличию информационного потока на чтение от y к x.

- *Post-правило*: если субъект x имеет доступ или информационный поток на чтение к сущности (объекту или субъекту) z, и субъект y имеет доступ или информационный поток на запись к z, то x имеет поток на чтение к y.
- *Pass-правило*: если субъект y имеет доступ или поток на чтение к сущности x и доступ или поток на запись к сущности z, то x имеет поток на чтение к z.
- *Spy-правило*: если субъект x имеет доступ или поток на чтение к субъекту y, имеющему доступ или поток на чтение к z, то x имеет поток на чтение к z.
- *Find-правило*: если субъект z имеет доступ или поток на запись к y, имеющему доступ или поток на запись к x, то x имеет поток на чтение к z.

Видно, что эти правила эквивалентны созданию потока на чтение из каждого доступа на чтение, созданию обратного потока на чтение из каждого доступа на запись, и транзитивному замыканию получаемых потоков на чтение.

Иерархичность данных под управлением ОС также влияет на правила управления доступом. Такое влияние аккуратно описано в модели военных сообщений (Military Message System, MMS) [20], расширяющей модель Белла-ЛаПадулы. В этой модели *контейнеры* являются объектами, которые используются для доступа к другим объектам, поэтому правила контроля доступа к объекту должны учитывать уровень защиты содержащего его контейнера. В MMS требуется, чтобы контейнер имел уровень защиты больший или равный максимальному уровню защиты содержащихся в нем объектов. При этом доступ к содержащимся в контейнере объектам может быть организован по-разному. При выставленном флаге CCR (*container clearance required*) у контейнера, доступ субъекта к содержимому контейнера возможен только при возможности доступа к самому контейнеру. При опущенном флаге CCR у контейнера, доступ субъекта к содержащемуся в контейнере объекту возможен при соблюдении лишь ограничений доступа к самому этому объекту. Помимо этого, модель MMS позволяет доступ к объекту через ссылки на него при выполнении тех же ограничений, которые должны выполняться при прямом доступе к этому объекту. В этой модели пользователь может обладать множеством ролей с различными правами доступа. Модификация ролей пользователя и прав ролей возможна только от имени специальной роли *офицера защиты*. Кроме того, введена особая роль, позволяющая снижать уровень защиты объектов при необходимости. Однако, помимо перечисленных правил, MMS не специфицирует других ограничений на администрирование пользователей, ролей и объектов.

2.5 Обзор SELinux

Security-Enhanced Linux (SELinux) [2, 21] является инфраструктурой управления доступом, реализованной в рамках ядра ОС Linux на основе Linux Security Modules (LSM). LSM определяет набор функций-перехватчиков, которые могут выполнять проверки прав доступа, дополнительные к стандартному для Unix механизму дискреционного управления доступом. Вызовы этих функций-перехватчиков распределены по коду ядра ОС Linux так, что каждый раз при выполнении операции доступа выполняется обращение к нужному перехватчику (за корректность размещения вызовов отвечают разработчики ядра, так что доверие к этой инфраструктуре основано на доверии к компетентности и аккуратности группы поддержки ядра ОС Linux). SELinux предоставляет реализацию нескольких механизмов контроля доступа, а именно, RBAC, MAC, а также контроля доступа на основе типов (type enforcement, TE), в виде общего механизма контроля выполнения правил политики, при этом политика задается извне, сам по себе механизм SELinux никаких конкретных правил не реализует.

Политика безопасности описывается на специализированном языке, каждое правило политики может задавать ограничение на выполнение определенных операций над определенному классами сущностей ОС на основе атрибутов, входящих в метки безопасности. В последних версиях SELinux определено примерно 50 классов сущностей, включающих процессы, директории, файлы и сокеты различных типов, сетевые интерфейсы, семафоры, очереди и пр. Для каждого класса определено от 5 до 20 операций. Метка безопасности сущности включает идентификатор пользователя, роль, тип, уровень и множество категорий. Помимо ограничений на выполнение операций могут быть заданы правила присваивания и трансформации меток.

В дополнение, SELinux поддерживает контроль доступа к данным внутри приложений, если приложение позволяет осуществлять такой контроль в дополнение к обычному контролю доступа в ядре ОС. Примерами подобных приложений являются системы управления базами данных (СУБД), менеджеры графического интерфейса, разнообразное программное обеспечение промежуточного уровня. На текущий момент SELinux поддерживает набор специальных классов для сущностей, связанных с внутренними объектами PostgreSQL [22], D-Bus [23] и X Window System [24].

Язык описания политик SELinux достаточно выразителен и позволяет использовать совместно атрибуты, относящиеся к механизмам RBAC, MAC или TE, в произвольных выражениях, которые можно построить из логических связок, операторов сопоставления атрибутов с константами, друг с другом, операторов сравнения для уровней, значения которых считаются упорядоченными, и операторов включения для множеств категорий. Поэтому анализ и валидация политик безопасности SELinux являются очень сложными задачами. Существуют некоторые инструменты для этого (например, [25]), но они поддерживают только подмножества языка политик и недостаточны для верификации специфических свойств безопасности, которые политика должна обеспечивать.

В нескольких работах рассматривается формальное моделирование и дальнейший анализ свойств политик SELinux. Работа [26] представляет формальный язык (имеющий строго определенную семантику) для описания политик с использованием элементов TE и RBAC, но не MAC. В статье [27] этот язык использован для построения алгоритмов и инструментов анализа политик. В работе [28] механизм SELinux вкладывается в известную модель безопасности Харрисона-Руззо-Ульмана (M.A. Harrison, W.L. Ruzzo, J.D. Ullman) [29], однако, поскольку анализ достижимости небезопасных состояний в этой модели неразрешим алгоритмически, практической пользы от этого не много.

Работа [30] предлагает формальную модель для анализа политик SELinux, включающих ограничения MAC. Эта модель использует два вида правил: ограничивающие доступ

субъектов к объектам и ограничивающие модификацию меток безопасности объектов субъектами. В указанной работе эта модель используется для анализа информационных потоков между различными уровнями защиты с помощью среды языка Prolog.

Обзорная статья [25] рассматривает свойства языка описания политик SELinux на основе общих свойств таких языков, проанализированных в работе [31]. Язык называется *безопасным*, если на нем запрос с меньшими входными данными приводит к более слабым решениям (относительно некоторого естественного порядка на решениях), чем запрос с большими входными данными. Язык обладает свойством *независимой композиции*, если результат, полученный при использовании всех правил политики такой же, как и результат последовательного применения каждого из правил в некотором порядке. Язык *монотонный*, если добавление правила в политику не изменяет решения с разрешения на запрет доступа. Язык политик SELinux в этих терминах обладает свойством независимой композиции, но является немонотонным и небезопасным.

3. Общее описание модели HIMACF

Модель HIMACF (рис. 1) интегрирует механизмы RBAC, MAC и MIC. Она также включает аналогичные TGM правила построения информационных потоков, которые позволяют анализировать последствия наличия скрытых каналов передачи информации или информационных атак и обеспечивают проверку выполнения определенных гарантий безопасности даже в случае успешных атак на систему на уровне низкоцелостных процессов. Модель формализована на языке Event-B и состоит из четырех модулей, соответствующих описываемым механизмам. Первый модуль или базовый уровень модели описывает RBAC, второй – MIC, третий – MAC, четвертый – информационные потоки. Каждый следующий модуль уточняет предыдущий.

Event-B [11] позволяет описывать моделируемую систему как абстрактный *автомат* с помощью *переменных*, комбинации значений которых соответствуют состояниям автомата, а типы могут быть описаны в обычной теории множеств, операций или *событий*, соответствующих переходам между состояниями и изменяющим значения переменных, и *инвариантов*, описывающих свойства переменных, сохраняемые при переходах. Событие определяется при помощи его имени, параметров, *предусловия* и *постусловия*. Предусловие и постусловие вместе определяют *контракт события*. Предусловие описывает ограничения на значения параметров и переменных состояния, которые должны выполняться при срабатывании данного события. Одно такое ограничение называется *охранным условием*. Постусловие описывает изменения переменных состояния при срабатывании события. Одно такое правило, задающее изменение одной переменной, называется действием.

Рис. 1. Структура HIMACF
Fig. 1. HIMACF Structure

Уточнение одной моделью другой означает, что переменные, инварианты, события и контракты событий уточняемой модели наследуются уточняющей. При этом в уточняющей модели могут быть добавлены новые переменные, инварианты и события. Кроме того, в постусловиях унаследованных событий могут быть добавлены новые действия, описывающие изменения новых переменных, а также могут быть добавлены новые параметры и в предусловиях могут быть добавлены новые охранные условия. Корректно реализованное уточнение обеспечивает выполнение в уточняющей модели всех инвариантов уточняемой, поскольку их доказательства остаются валидными в уточняющей модели.

3.1 Основные идеи

Описание RBAC на базовом уровне HIMACF похоже на модель ARBAC [32]. Оно использует множества *пользователей*, *субъектов* (представляющих процессы ОС), *сущности* (представляющие любые объекты доступа, включая файлы, сокеты, устройства, очереди, семафоры, и пр.), *роли*. Сущности могут быть *контейнерами*, т.е., сущностями, способными содержать другие сущности, либо простыми *объектами*. объекты и контейнеры не пересекаются. Роли могут быть *обычными*, имеющими права обычных типов ($read, write, execute, own$), или *административными*, имеющие права на приписывание или удаление ролей, а также на модификацию прав ролей. Множества обычных и административных ролей не пересекаются.

Права доступа определенного типа или права владения представлены как *обычные права*. *Владельцем* сущности является роль, имеющая права менять права доступа к этой сущности. Владение моделируется при помощи права own. У сущности может быть не более одного владельца. *Административные права* представляют возможность приписывать обычную или административную роль субъекту или возможность модифицировать права роли. Субъект, имеющий (через административную роль) доступ $read$ к роли, может использовать ее права, т.е., наличие этого доступа к роли моделирует привязку роли к субъекту. Субъект, имеющий доступ $write$ к роли, может модифицировать ее права.

Каждый контейнер имеет строковое имя. Объект может иметь различные имена в разных контейнерах, но имя объекта внутри контейнера уникально. Иерархия сущностей ациклична, т.е., контейнер не может прямо или косвенно содержаться в себе самом. Есть единственный корень иерархии сущностей, который не содержится ни в каком контейнере.

Субъекты также имеют иерархию, субъект может иметь не более одного *родителя*, что моделирует соответствующее отношение между процессами в Unix. Эта иерархия ациклична, но может иметь несколько корневых субъектов. Для каждого субъекта имеется *роль-владелец*, имеющая права на любые операции над этим субъектом.

Роли имеют свою иерархию, иерархии обычных и административных ролей не пересекаются. У роли может быть несколько *родителей*. Иерархия ролей ациклична и может иметь несколько корней. Субъект, к которому приписана некоторая роль, имеет права этой роли, а также ее родителей и всех более далеких предков. Роли имеют строковые имена, имя роли уникально среди всего множества ролей. Есть выделенное множество *специальных административных ролей*, включающее роли, имеющие права администрирования пользователей, сущностей, субъектов и ролей. Специальные административные роли не могут быть созданы или удалены, переименованы, их права не могут быть модифицированы. Также постулируется существование специальных пользовательских ролей, имеющих права на операции, специфичные для данного

пользователя, и общих ролей, имеющих права на действия, которые может выполнить любой пользователь.

Состояние модуля RBAC задается множествами пользователей, сущностей, субъектов, ролей, привязкой (обычных и административных) ролей к субъектам, иерархиями сущностей, субъектов и ролей, отображением владения субъектов в пользователей, а также текущими (обычными и административными) доступами субъектов к сущностям и ролям.

В модуле RBAC определены события создания и удаления пользователей, сущностей, субъектов и ролей, добавления имеющейся сущности в контейнер (создания жесткой ссылки), переименования сущности или роли, модификации множества ролей, приписанных к субъекту, модификации прав роли, получения обычного доступа субъекта к сущности или административного доступа субъекта к роли.

Ключевое свойство безопасности механизма RBAC состоит в том, что для получения доступа к чему-либо субъект должен иметь привязанную роль, имеющую права на этот доступ. Это ограничение зафиксировано в предусловиях событий получения доступа, оно не может быть оформлено в виде инварианта, поскольку субъект может получить доступ к сущности через определенную роль, а затем потерять эту роль, не теряя доступ. Аналогично, после получения доступа через роль эта роль может потерять право на этот доступ.

Модуль, моделирующий механизм MIC, определяет *уровни целостности* (включающие, по крайней мере, low и $high$) и добавляет в состояние отображение пользователей, сущностей, субъектов и ролей в уровни целостности. Высокоцелостные пользователи (имеющие уровень целостности $high$) могут запускать от своего имени как высокоцелостные, так и низкоцелостные (с уровнем low) субъекты. Низкоцелостные пользователи могут запускать только низкоцелостные субъекты. Кроме того, низкоцелостный субъект может быть родителем только низкоцелостных субъектов.

Для каждого пользователя есть специальные низкоцелостные обычная и административная роли, имеющие права на все операции, которые этот пользователь может выполнить на низком уровне целостности. Для высокоцелостных пользователей, кроме того, имеются высокоцелостные обычная и административная роли для тех действий, которые он может выполнить на высоком уровне целостности. Поэтому в модели ни одно действие не может быть выполнено без использования роли с правом на это действие, что позволяет сохранить ключевое свойство безопасности RBAC.

Моделирующий MIC модуль добавляет события, позволяющие устанавливать уровни целостности пользователей, сущностей, субъектов и ролей. Все имеющиеся на базовом уровне события уточнены, в них добавлены охранные условия, говорящие, что только субъект уровня $high$ может получить доступ на модификацию чего-либо с уровнем $high$.

Основным свойством безопасности этого механизма является требование к субъекту иметь уровень $high$ для получения $write$-доступа к чему-либо с уровнем $high$, помимо предусловий оно зафиксировано в виде инвариантов. Поскольку все в системе имеет уровни целостности, все прямые или косвенные попытки нарушить это требование могут быть отслежены и предотвращены.

Модуль, моделирующий механизм MAC, добавляет частично упорядоченное множество *уровней конфиденциальности*, а в состояние – отображения пользователей, сущностей, субъектов и ролей в уровни конфиденциальности. Уровень конфиденциальности рассматривается как составная метка, содержащая всю информацию, необходимую для контроля конфиденциальности. В обычных терминах, она содержит и множество категорий, и число, соответствующее обычному уровню. Пользователь может запускать субъекты любого уровня конфиденциальности, меньше или равного уровню

пользователя. Специальные обычные и административные роли создаются для любой комбинации из уровня целостности и уровня конфиденциальности, доступной данному пользователю, эти роли обладают правами на все действия этого пользователя на данных уровнях конфиденциальности и целостности.

Основным свойством безопасности механизма MAC является требование к субъекту иметь больший или равный уровень конфиденциальности для доступа на чтение к чему-либо, и равный уровень конфиденциальности для доступа на запись. Это свойство оформлено в виде инвариантов.

Последний, четвертый модуль, добавляет переменные и события для построения информационных потоков. По аналогии с моделью Take-Grant, информационные потоки строятся между сущностями и субъектами путем построения транзитивного замыкания доступов. Вводятся события, являющиеся аналогами правил *take, pass, find*, и *spy*. Кроме того, определяется отношения контроля между субъектами. Субъект контролирует себя и любого другого субъекта, к исполнимым файлам которого он имеет доступ на запись или поток на запись, далее это отношение замыкается по транзитивности. Информационные потоки также пополняются потоками, которые могут создать контролирующие процессы через контролируемые. Таким образом, в рамках модели. включающей этот модуль можно анализировать информационные потоки в системе, а также возможные утечки информации при успешных атаках. Можно доказать, что информационные потоки не могут нарушить основные свойства безопасности – перенести информацию с более высокого уровня конфиденциальности на более низкий, даже при успешной атаке, компрометирующей субъекты на только низком уровне целостности.

3.2 Формальное представление модели

В данном подразделе дается более детальное описание формального представления базового уровня модели HIMACF, описывающего механизм RBAC. Кроме того, приведено несколько примеров спецификации событий и инвариантов других уровней для демонстрации важных свойств модели.

В моделях на языке Event-B статическая часть называется *контекстом* и содержит описание базовых типов (называемых множествами), констант и аксиом, аксиомы задают типы констант и другие ограничения. Динамическая часть модели, называемая *машиной*, содержит описание элементов автомата – переменных состояния, событий и инвариантов.

Контекст базового уровня модели HIMACF содержит следующие множества.

- *UserType* представляет тип пользователей. Это совокупность всех потенциально возможных пользователей, включая еще не зарегистрированных в системе.
- *SubjectType* – тип субъектов.
- *EntityType* – тип сущностей (объектов и контейнеров).
- *RoleType* – тип ролей.
- *Names* представляет все возможные имена сущностей и ролей.
- *Accesses* – возможные доступы к сущностям и ролям, включает доступ на чтение $ReadA$ и на запись $WriteA$.
- *AccessRights* – возможные права ролей, включает константы $Read$, $Write$, $Execute$, Own.

Корневой контейнер в иерархии сущностей соответствует константе $Root \in EntityType$. Множество специальных административных ролей *SpecialAdmRoles* тоже является константой, содержащей отдельные роли для администрирования сущностей, субъектов, пользователей, обычных ролей и административных ролей. Далее используется

конструкция $partition(U, A, B, C)$, в Event-B означающая $U = A \cup B \cup C$ и $A \cap B = A \cap C = B \cap C = \emptyset$, эта конструкция может иметь любое число аргументов, больше двух, с аналогичным смыслом. Выполнены свойства $partition\ (SpecialAdmRoles,\ \{EntitiesAR\},\ \{SubjectsAR\},\ \{UsersAR\},\ \{OrdRolesAR\},\ \{AdmRolesAR\})$, $partition\ (Accesses;\ \{ReadA\},\ \{WriteA\})$ и $partition\ (AccessRights,\ \{Read\},\ \{Write\},\ \{Execute\},\ \{Own\})$.

Базовый уровень модели HIMACF имеет следующие переменные состояния (определения типов переменных мы указываем здесь же, а не в отдельных инвариантах, как положено в Event-B).

- $Users \subseteq UserType$ – множество зарегистрированных пользователей.
- $Subjects \subseteq SubjectType$ – множество активных (работающих в данный момент) субъектов.
- $Entities \subseteq EntityType$ – множество существующих сущностей.
- $Roles \subseteq RoleType$ – множество всех существующих ролей, обычных и административных.
- $Objects$ – множество существующих объектов, сущностей, не содержащих другие сущности.
- $Containers$ – множество контейнеров, способных содержать другие сущности. Выполняются свойства $partition\ (Entities, Objects, Containers)$ и $Root \in Containers$.
- $RegRoles$ – множество обычных ролей.
- $AdmRoles$ – множество административных ролей. Выполнены свойства partition $(Roles, RegRoles, AdmRoles)$ и $SpecialAdmRoles \subseteq AdmRoles$.
- $EntityNames \in (Entities \setminus \{Root\}) \to (Containers \leftrightarrow Names)$ – отображение сущности в множество пар из контейнера и имени этой сущности в данном контейнере.
- $RoleName \in Roles \to Names$ – отображение ролей в их имена.
- $Parent \in (Containers \setminus \{Root\}) \to Containers$ – отображение контейнера в родительский контейнер. Контейнер может содержаться только в одном контейнере, для них жесткие ссылки запрещены. Объект может содержаться в нескольких контейнерах, которые можно вычислить из отображения $EntityNames$.
- $SParent \in Subjects \nrightarrow Subjects$ – частичное отображение субъектов в их родителей.
- $RParents \in Roles \to \mathcal{P}(Roles)$[1] – отображение роли в множество ее родительских ролей.
- $RoleRights \in Roles \to (Entities \leftrightarrow AccessRights)$ – отображение ролей в их права.
- $RoleAdmRights \in AdmRoles \to (Roles \leftrightarrow AccessRights)$ – отображение административных ролей в их права.
- $SubjectUser \in Subjects \to Users$ – отображение субъекта в активировавшего его пользователя.
- $SubjectOwner \in Subjects \nrightarrow Roles$ – частичное отображение субъектов в их роли-владельцы.
- $SubjectAccesses \in Subjects \to (Entities \leftrightarrow Accesses)$ – отображение субъектов в их текущие доступы к сущностям.
- $SubjectAdmAccesses \in Subjects \to (Roles \leftrightarrow Accesses)$ – отображение субъектов в их административные доступы к ролям.

[1] Здесь и далее $\mathcal{P}(X)$ используется как обозначение множества всех подмножеств X.

Инварианты, в дополнение к типовым ограничениям, содержат ограничения корректности значений переменных. Для отображений $EntityNames$ и $Parent$ должно быть выполнено следующее.

- $\forall e \cdot e \in \mathrm{dom}(EntityNames) \Rightarrow EntityNames(e) \neq \emptyset$. Каждая сущность, кроме $Root$, содержится в каком-то контейнере.
- $\forall e1, e2 \cdot \{e1, e2\} \subseteq \mathrm{dom}(EntityNames) \wedge e1 \neq e2 \Rightarrow EntityNames(e1) \cap EntityNames(e2) = \emptyset$. Имена сущностей в контейнерах уникальны.
- $\forall c1, c2 \cdot c1 \in Containers \wedge c1 \neq Root \wedge c2 \in \mathrm{dom}(EntityNames(c1)) \Rightarrow c2 = Parent(c1)$. Некорневой контейнер содержится только в одном, родительском, контейнере.
- $\forall C \cdot C \subseteq \mathrm{dom}(Parent) \wedge C \neq \emptyset \Rightarrow C \setminus Parent[C] \neq \emptyset$. В иерархии контейнеров нет циклов.

На иерархии субъектов и ролей наложены следующие ограничения.

- $\forall r \cdot r \in RegRoles \Rightarrow RParents(r) \subseteq RegRoles$.
- $\forall r \cdot r \in AdmRoles \Rightarrow RParents(r) \subseteq AdmRoles$. Иерархии обычных и административных ролей не пересекаются.
- $\forall R \cdot R \subseteq Roles \wedge R \neq \emptyset \Rightarrow (\exists r \cdot r \in R \wedge R \cap RParents(r) = \emptyset)$. В иерархии ролей нет циклов.
- $\forall S \cdot S \subseteq \mathrm{dom}(SParent) \wedge S \neq \emptyset \Rightarrow S \setminus SParent[S] \neq \emptyset$. В иерархии субъектов нет циклов.

У сущности или роли может быть не более одного владельца. $\forall r1, r2, e \cdot r1 \in Roles \wedge r2 \in Roles \wedge (e \mapsto Own) \in RoleRigths(r1) \cap RoleRights(r2) \Rightarrow r1 = r2$.

Административное право $read$ распространяется от ролей-родителей к детям. $\forall a, r, p \cdot a \in AdmRoles \wedge r \in Roles \wedge p \in RParents(r) \wedge (p \mapsto Read) \in RoleAdmRights(a) \Rightarrow (r \mapsto Read) \in RoleAdmRights(a)$.

Список операций модели (событий Event-B) на базовом уровне следующий.

- Управление пользователями: *create_user, delete_user, get_user_attr*.
- Управление доступами: *access_read_entity, access_write_entity, delete_access_entity, access_read_role, access_write_role, delete_access_role*.
- Управление правами: *grant_rights, remove_rights, set_entity_owner, set_subject_owner, grant_admin_rights, remove_admin_rights*.
- Управление сущностями: *create_object, create_container, delete_entity, create_hard_link, delete_hard_link, rename_entity, get_entity_attr, read_container, set_entity_labels, set_container_attr*.
- Управление ролями: *create_role, create_hard_link_role, delete_role, delete_hard_link_role, rename_role, get_role_attr*.
- Управление субъектами: *create_first_subject, create_subject, delete_subject, get_subject_attr*.

Например, спецификация события *access_write_entity* на базовом уровне такова.

event $access_write_entity$
  any
    $subject$
    $entity$
  where
    @grd1 $subject \in Subjects$
    @grd2 $entity \in Entities$
    @grd3 $\exists r \cdot r \in RegRoles \cup AdmRolesr$

$\wedge\, r \mapsto ReadA \in SubjectAdmAccesses(subject)$
$\wedge\, entity \mapsto Write \in RoleRights(r)$

@grd4 $\exists E, c \cdot E \subseteq \mathrm{dom}(Parent) \wedge (entity = Root$
$\wedge\, E = \emptyset) \vee (\, entity \neq Root$
$\wedge\, c \in \mathrm{dom}\big(EntityNames(entity)\big)$
$\wedge\, Parent[E] \cup \{c\} = E \cup \{Root\}))$
$\wedge\, (\forall o \cdot o \in E \cup \{entity\} \cup \{Root\} \Rightarrow$
$(\exists r \cdot r \in RegRoles \cup AdmRoles$
$\wedge\, r \mapsto ReadA \in SubjectAdmAccesses(subject)$
$\wedge\, o \mapsto Execute \in RoleRights(r)))$

then

@act1 $SubjectAccesses(subject) \coloneqq$
$SubjectAccesses(subject) \cup \{entity \mapsto WriteA\}$

end

Здесь параметр $subject$ – это субъект, пытающийся получить доступ, *entity* – сущность, доступ к которой запрошен. Два первых охранных условия задают типы параметров. Третье условие утверждает, что чтобы получить доступ на запись в сущность, субъект должен иметь связанную с ним роль r, имеющую право на запись в эту сущность. Четвертое условие означает, что для получения доступа субъект должен иметь доступ $Execute$ ко всем членам некоторой цепочки включающих друг друга контейнеров, начинающейся с $Root$ и заканчивающейся контейнером, содержащим нужную сущность (эта цепочка без корня обозначена как E).

Далее представлены уточненные спецификации этого же события в других модулях. В модуле, описывающем механизм MIC, к типам добавлено множество уровней целостности $Integrity$ и две константы *LowI* и $HighI$, принадлежащие этому множеству. Добавленные в переменные отображения $EntityInt: Entities \to Integrity$ и $SubjectInt: Subjects \to Integrity$ задают уровни целостности сущностей и субъектов. Отображение $CCRI: Containers \to BOOL$ задает CCRI-флаги контейнеров. Если такой флаг для контейнера выставлен в $TRUE$, то для доступа к сущностям внутри контейнера субъекту необходимо выполнить условия доступа к самому контейнеру. В этом модуле добавлены события *set_user_labels* и *set_role_labels*, позволяющие менять уровни целостности пользователей и ролей. Большая часть событий первого модуля уточнены при помощи добавления охранных условий, проверяющих соотношения уровней целостности параметров.

Инвариант, представляющий основное свойство безопасности механизма MIC, выглядит так.

$@SubjectAccesses1$
$\forall s, e \cdot s \in Subjects \wedge (e \mapsto WriteA) \in$
$SubjectAccesses(s) \Rightarrow EntityInt(e) \leq SubjectInt(s)$

В спецификацию события *access_write_entity* во втором модуле добавлены следующие два условия.

event $access_write_entity$ extends $access_write_entity$

where

$@grd5\ \exists E, c \cdot E \subseteq \mathrm{dom}(Parent) \wedge ((entity = Root$
$\wedge\, E = \emptyset) \vee (entity \neq Root$
$\wedge\, c \in \mathrm{dom}(EntityNames(entity))$
$\wedge\, Parent[E] \cup \{c\} = E \cup \{Root\}))$
$\wedge\, (\forall o \cdot o \in E \cup \{entity\} \cup \{Root\} \Rightarrow$
$(\exists r \cdot r \in RegRoles \cup AdmRoles$

$\wedge\, r \mapsto ReadA \in SubjectAdmAccesses(subject)$
$\wedge\, o \mapsto Execute \in RoleRights(r))$
$\wedge\, EntityInt(o) \leq SubjectInt(subject)$
$\vee\, CCRI(o) = FALSE))$
@grd6 $EntityInt(entity) \leq SubjectInt(subject)$
end

Пятое охранное условие требует существования цепочки включающих друг друга контейнеров, начинающейся с $Root$ и заканчивающейся контейнером, содержащим нужную сущность, для каждого из которых данный субъект должен иметь роль с правом доступа на $Execute$, кроме того, каждый из этих контейнеров должен либо иметь опущенный CCRI-флаг, либо уровень целостности меньше или равный уровня целостности субъекта, запрашивающего доступ. Шестое условие требует, чтобы субъект, запрашивающий доступ на запись к сущности, имел уровень целостности, больше или равный уровню целостности этой сущности.

Третий модуль описывает механизм MAC, добавляя в рассмотрение уровни конфиденциальности Cnf, определяемые как множество всех подмножеств некоторого непустого конечного множества $Confidentiality$. Поскольку каждое конечное частично упорядоченное множество может быть изоморфно вложено в решетку множества всех подмножеств некоторого конечного множества, такое описание не накладывает значимых ограничений на структуру возможных уровней конфиденциальности.

В качестве переменных добавлены отображения $EntityCnf: Entities \to Cnf$ и $SubjectCnf: Subjects \to Cnf$, задающие уровни конфиденциальности сущностей и субъектов. Также добавлено отображение $CCR: Containers \to BOOL$, описывающее CCR-флаги контейнеров. Если такой флаг для контейнера выставлен в $TRUE$, то для доступа к сущностям внутри контейнера субъекту необходимо выполнить условия доступа к самому контейнеру. Снова, большинство событий уточнены с помощью охранных условий, проверяющих соотношения между уровнями конфиденциальности их параметров.

Инварианты, описывающие основное свойство безопасности специфицируемого механизма, выглядят так.

$@SubjectAccesses2$
$\forall s, e \cdot s \in Subjects(e \mapsto ReadA) \in SubjectAccesses(s) \Rightarrow$
$EntityCnf(e) \subseteq SubjectCnf(s)$
$@SubjectAccesses3$
$\forall s, e \cdot s \in Subjects(e \mapsto WriteA) \in SubjectAccesses(s) \Rightarrow$
$EntityCnf(e) = SubjectCnf(s)$

Следующие условия добавлены в спецификацию $access_write_entity$ в третьем модуле.

event $access_write_entity$ $extends$ $access_write_entity$
$where$
@grd7 $\exists E, c \cdot E \subseteq \mathrm{dom}(Parent) \wedge ((entity = Root$
$\wedge\, E = \emptyset) \vee (entity \neq Root$
$\wedge\, c \in \mathrm{dom}(EntityNames(entity))$
$\wedge\, Parent[E] \cup \{c\} = E \cup \{Root\}))$
$(\forall o \cdot o \in E \cup \{entity\} \cup \{Root\} \Rightarrow$
$(\exists r \cdot r \in RegRoles \cup AdmRoles$
$\wedge\, r \mapsto ReadA \in SubjectAdmAccesses(subject)$
$\wedge\, o \mapsto Execute \in RoleRights(r))$
$\wedge\, (EntityInt(o) \leq SubjectInt(subject)$

$\vee\ CCRI(o) = FALSE)$
$\wedge\ (EntityCnf(o) \subseteq SubjectCnf(subject)$
$\vee$ CCR(o)=FALSE))
$@grd8\ EntityCnf(entity) = SubjectCnf(subject)$
end

Седьмое условие здесь снова требует существования цепочки включающих друг друга контейнеров, начинающейся с $Root$ и заканчивающейся контейнером, содержащим нужную сущность, для каждого из которых данный субъект должен иметь роль с правом доступа на $Execute$, кроме того, каждый из этих контейнеров должен либо иметь опущенный CCRI-флаг, либо уровень целостности меньше или равный уровню целостности субъекта, запрашивающего доступ, и должен либо иметь опущенный CCR-флаг, либо уровень конфиденциальности меньше или равный уровню конфиденциальности субъекта. Восьмое условие говорит, что для получения доступа $Write$ к сущности субъект должен иметь равный уровень конфиденциальности. Условия 5 и 7 вынуждено повторяют ограничения из условия 4, поскольку все эти ограничения должны быть выполнены для одной и той же цепочки контейнеров.

Последний модуль специфицирует модель информационных потоков между сущностями и субъектами. Для этого добавляются четыре переменных $EEMFlows: Entities \to \mathcal{P}(Entities)$, $ESMFlows: Entities \to \mathcal{P}(Subjects)$, $SEMFlows: Subjects \to \mathcal{P}(Entities)$ и $SSMFlows: Subjects \to \mathcal{P}(Subjects)$. Они представляют потоки на запись от сущностей к сущностям, от сущностей к субъектам, от субъектов к сущностям и от субъектов к субъектам. Отношение контроля между субъектами представлено отображением $DeFactoOwn: Subjects \to \mathcal{P}(Subjects)$. Правило $find$ из TGM трансформируется в два события, $find_entity$ и $find_subject$. Спецификация первого из них выглядит следующим образом.

event $find_entity$
  any
    x
    y
    z
  where
    @grd1 $x \in Subjects$
    @grd2 $y \in Subjects$
    @grd3 $z \in Entities$
    @grd4 $y \in SSMFlows(x)$
    @grd5 $z \in SEMFlows(y)$
  $then$
    $@act1\ SEMFlows(x) := SEMFlows(x) \cup \{z\}$
end

4. Верификация модели

Сложность модели HIMACF делает ее верификацию без поддержки каких-либо инструментов чрезвычайно сложной задачей. Ее описание на сочетании математического и естественного языков занимает около 300 страниц, и оно увеличивается, поскольку модель постоянно развивается и в нее включаются новые элементы. Некоторые числовые характеристики модели в текущей версии представлены в табл. 1, каждая строка которой показывает значения характеристик для отдельных модулей, а последняя строка – общие числа. Обозначения модулей сокращены. В третьем ее столбце находится количество новых событий, специфицированных в

определенном модуле, в четвертом – количество уточненных в данном модуле событий из предыдущих модулей.

Табл. 1. Числовые характеристики модели HIMACF
Table 1. Numerical characteristics of the HIMACF model

Модуль	Переменные	Нов. события	Уточн. события	Инварианты	Строки кода
RBAC	21	37	0	50	1480
MIC+	11	2	37	47	1120
MLS+	5	0	34	25	700
Information Flows+	15	36	39	74	1770
Total	52	75	—	196	5070

Автоматизированная верификация модели проведена с помощью инструмента Rodin [12], поддерживающего работу с моделями на языке Event-B. Общее число доказываемых утверждений (каждое из них утверждает корректность типа переменной, изменяемой в рамках некоторого события, сохранение инварианта в рамках некоторого события, и т.д.), сгенерированных в среде Rodin, оказывается около 3600. Из них примерно 75% доказывается автоматически, остальные доказываются интерактивно, с участием человека, и для этого нужно около одного человеко-месяца. Эти оценки применимы к текущей структуре модели и работающим с ней разработчикам. Первая версия модели [10] не использовала уточнения, все механизмы были специфицированы в одном модуле, и ее разработка и верификация на основе исходного представления на естественном ярыке потребовали примерно 2 человеко-года. Текущая структура модели более удобна как для восприятия, так и для внесения изменений и развития модели, поскольку трудоемкость внесения небольших изменений и повторного доказательства инвариантов, на которые они влияют, обычно существенно ниже [33].

5. Заключение и возможное развитие

В данной статье представлены текущие результаты по разработке и верификации интегрированной модели, специфицирующей композицию управления доступом на основе ролей (RBAC), мандатного контроля целостности (MIC), мандатного управления доступом (MAC), а также информационные потоки в возникающей системе, и названной HIMACF (hierarchical integrated model of access control and information flows). Модель специфицирована на формальном языке Event-B и формально верифицирована, все ее инварианты доказаны с помощью инструмента Rodin, поддерживающего работу с моделями на Event-B.

Модель построена из нескольких модулей, уточняющих друг друга. Это позволяет более ясно описать каждый из механизмов, четко выделяя их ключевые свойства безопасности, а также облегчает выполнение верификации. Первый модуль описывает механизм RBAC, второй – MIC, третий – MAC, и четвертый модуль специфицирует информационные потоки. Ключевым свойством безопасности RBAC является требование того, чтобы субъект получал доступ на выполнение некоторой операции, только если он обладает ролью, имеющей права на ее выполнение. Ключевое требование MIC – субъект может получить доступ на модификацию сущности или вмешательство в работу другого субъекта, только если он имеет более высокий или равный уровень целостности. Аналогично, MAC требует, чтобы субъект, получающий доступ на чтение или выполнение, имел более высокий уровень конфиденциальности, а получающий доступ на модификацию – равный уровень конфиденциальности. В модели HIMACF все эти ограничения выполняются, благодаря разметке всех субъектов, сущностей и ролей уровнями конфиденциальности и целостности, а также наличию специализированных ролей, через которых пользователи выполняют все обычные операции.

Поскольку основной целью разработки модели была аккуратная интеграция нескольких механизмов управления доступом, применимых в современных ОС, контролирующих сложные иерархии процессов и файлов, получившаяся модель достаточно сложна. Тем не менее, за счет использования языка формальных спецификаций, поддержанного инструментами интерактивного дедуктивного анализа, формальная верификация модели проводится с приемлемой трудоемкостью.

Иерархическая модульная структура самой модели, использующая уточнение, позволяет эффективно развивать модель и вносить новые элементы в нее, проводя верификацию измененных версий быстро и с небольшими трудозатратами. В настоящее время уже разработаны версии модели с косвенными метками защиты (помещаемыми не на самих файлах, а на корне некоторого дерева контейнеров), специализированными объектами с ослабленными ограничениями доступа (моделирующими специальные устройства Unix без возможности сохранения информации, как, например, dev/null), запрещающими ролями (ролями, обладание которыми запрещает выполнение некоторых операций) и более сложной структурой уровней целостности в виде произвольного частично упорядоченного множества. В процессе разработки находится расширение модели, применимое для использования в качестве модели безопасности СУБД.

Модель HIMACF частично реализована в построенной на основе ядра Linux ОС специального назначения Astra Linux Special Edition [15], использующей инфраструктуру Linux Security Modules (LSM) для дополнительного контроля доступа. Наличие формальной модели позволяет построить специализированную формальную модель реализации интерфейсов LSM в этой ОС и, в идеале, провести формальную верификацию соответствия реализации кода реализации ОС самой модели, например, с использованием инструментов [34], что в итоге позволит получить формально верифицированную реализацию механизма управления доступом в рамках современной ОС. Эта работа требует еще довольно много усилий и совершенствования имеющихся инструментов верификации, информацию о прогрессе в ее продвижении можно найти, например, в [35].

Список литературы / References

[1] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, C.E. Youman. Role-based access control models. Computer, vol. 29, no. 2, 1996, pp. 38-47.

[2] S. Smalley, C. Vance, W. Salamon. Implementing SELinux as a linux security module. Technical Report 01-043, NAI Labs, 2001.

[3] S. Smalley, R. Craig. Security Enhanced (SE) Android: Bringing Flexible MAC to Android. In Proc. of Network & Distributed System Security Symposium (NDSS), 2013, 18 p.

[4] M. Conover. Analysis of the Windows Vista security model. Technical Report, Symantec Corp., 2008.

[5] A. Cunningham, L. Hutchinson. OSX10.11 El Capitan: The Ars Technica Review. Available at: https://arstechnica.com/apple/2015/09/os-x-10-11-el-capitan-the-ars-technica-review/. Accessed 21 January 2019.

[6] D.E. Bell, L.J. LaPadula. Secure Computer Systems: Mathematical Foundations. Technical Report ESD-TR-73-278 v.~1, (also MTR-2547, v.~1), Electronic Systems Division, AFSC, Hanscom AFB, 1973.

[7] K.J. Biba. Integrity considerations for secure computer systems. Technical Report MTR-3153, The MITRE Corporation, 1977.

[8] R. Sandhu. Role hierarchies and constraints for lattice-based access controls. Lecture Notes in Computer Science, vol. 1146, 1996, pp.65-79.

[9] П.Н. Девянин. Модели безопасности компьютерных систем. Управлне доступом и информационными потоками. Горячая линия-Телеком, 2013, 338 стр. / P.N. Devyanin. Security models of computer systems. Control for access and information flows. Hotline-Telecom, 2013, 338 p. (in Russian).

[10] P.N. Devyanin, A.V. Khoroshilov, V.V. Kuliamin, A.K. Petrenko, I.V. Shchepetkov. Formal

Verification of OS Security Model with Alloy and Event-B. Lecture Notes in Computer Science, vol. 8477, 2014, pp. 309-313.
[11] J.R. Abrial. Modeling in Event-B: System and Software Engineering. Cambridge University Press, 2010, 612 p.
[12] J.R. Abrial, M. Butler, S. Hallerstede, T.S. Hoang, F. Mehta, L. Voisin. Rodin: an open toolset for modelling and reasoning in Event-B. International Journal on Software Tools for Technology Transfer, vol. 12, no. 6, 2010, pp. 447-466.
[13] ISO/IEC 15408-1:2009. Information technology – Security techniques – Evaluation criteria for IT security – Part 1: Introduction and general model. ISO, 2009.
[14] ISO/IEC 15408-2:2008. Information technology – Security techniques – Evaluation criteria for IT security – Part 2: Security functional components. ISO, 2008.
[15] Astra Linux. Available at: https://en.wikipedia.org/wiki/Astra_Linux. Accessed 21 January 2019.
[16] American National Standard for Information Technology – Role Based Access Control. ANSI INCITS 359-2004, 2004.
[17] D.E. Bell, L.J. LaPadula. Secure Computer System: Unified Exposition and MULTICS Interpretation. Technical Report ESD-TR-75-306 (also MTR-2997), Electronic Systems Division, AFSC, Hanscom AFB, 1976.
[18] A.K. Jones, R.J. Lipton, L. Snyder. A linear time algorithm for deciding security. In Proc. of 17-th Annual Symposium on Foundations of Computer Science, 1976, pp. 33-41.
[19] 19. M. Bishop, L. Snyder. The Transfer of Information and Authority in a Protection System. In Proc. of 7th ACM Symposium on Operating System Principles, 1979, pp. 45-54.
[20] C.E. Landwehr, C.L. Heitmeyer, J. McLean. A security model for military message systems. ACM Transactions on Computer Systems, vol. 2, issue 3, 1984, pp. 198-222.
[21] Security-Enhanced Linux. Available at: https://www.nsa.gov/what-we-do/research/selinux/. Accessed 21 January 2019.
[22] PostgreSQL. Available at: https://en.wikipedia.org/wiki/PostgreSQL. Accessed 21 January 2019.
[23] D-Bus. Available at: https://en.wikipedia.org/wiki/D-Bus. Accessed 21 January 2019.
[24] X Window System. Available at: https://en.wikipedia.org/wiki/X_Window_System. Accessed 21 January 2019.
[25] A. Eaman, B. Sistany, A. Felty. Review of Existing Analysis Tools for SELinux Security Policies: Challenges and a Proposed Solution. Lecture Notes in Business Information Processing, vol. 289, 2017, pp. 116-135.
[26] G. Zanin, L.V. Mancini. Towards a Formal Model for Security Policies Specification and Validation in the Selinux System. In Proc. of 9th ACM Symposium on Access Control Models and Technologies, 2004, pp. 136-145.
[27] G. Zhai, T. Guo, J. Huang. SCIATool: A Tool for Analyzing SELinux Policies Based on Access Control Spaces, Information Flows and CPNs. Lecture Notes in Computer Science, vol. 9473, 2015, pp. 294-309.
[28] P. Amthor, W.E. Kühnhauser, A. Pölck. Model-based safety analysis of SELinux security policies. In Proc. of 5th International Conference on Network and System Security (NSS), 2011, pp. 208-215.
[29] M.A. Harrison, W.L. Ruzzo, J.D. Ullman. Protection in operating systems. Communications of the ACM, vol. 19, no. 8, 1976, pp. 461-471.
[30] B. Hicks, S. Rueda, L. St.Clair, T. Jaeger, P. McDaniel. A logical specification and analysis for SELinux MLS policy. ACM Transactions on Information and System Security, vol. 13, issue 3, 2010, article no. 26.
[31] M.C. Tschantz. The clarity of languages for access-control policies. PhD Thesis, Brown University, Providence, Rhode Island, USA, 2005.
[32] R. Sandhu, V. Bhamidipati, Q. Munawer. The ARBAC97 model for role-based administration of roles. ACM Transactions on Information and System Security, vol. 2, issue 1, 1999, pp. 105-135.
[33] П.Н. Девянин, В.В. Кулямин, А.К. Петренко, А.В. Хорошилов, И.В. Щепетков. Сравнение способов декомпозиции спецификаций на Event-B. Программирование, vol. 42, no. 4, 2016, pp. 17-26 / P.N. Devyanin, V.V. Kuliamin, A.K. Petrenko, A.V. Khoroshilov, I.V. Shchepetkov. Comparison of specification decomposition methods in Event-B. Programming and Computer Software, vol. 42, no. 4, 2016, pp. 198-205.
[34] J.C. Filliâtre, A.Paskevich. Why3 – Where Programs Meet Provers. Lecture Notes in Computer Science, vol. 7792, 2013, pp. 125-128.

[35] D. Efremov, M. Mandrykin, A. Khoroshilov. Deductive verification of unmodified Linux kernel library functions. Lecture Notes in Computer Science, vol. 11245, 2018, pp. 216-234.

Информация об авторах / Information about authors

Петр Николаевич ДЕВЯНИН – член-корреспондент Академии криптографии России, доктор технических наук, профессор, главный научный сотрудник ООО "РусБИТех-Астра" (ГК Astra Linux). Область интересов: теория информационной безопасности, формальные модели безопасности компьютерных систем.

Petr Nikolaevich DEVYANIN – Doctor of Technical Sciences, corresponding member of Russian Academy of Cryptography, professor, main researcher in RusBITech-Astra (Astra Linux). Field of Interest: information security theory, formal security models of computer systems.

Виктор Вячеславович КУЛЯМИН –кандидат физико-математических наук, ведущий научный сотрудник ИСП РАН, доцент кафедр системного программирования МГУ и ВШЭ. Область интересов: формальная дедуктивная верификация моделей, тестирование на основе моделей.

Viktor Vyacheslavovich KULYAMIN – Ph.D. in Physics and Mathematics, leading researcher at ISP RAS, associate professor of system programming departments at Moscow State University and the Higher School of Economics. Fields of Interest: formal deductive model verification, model-based testing

Александр Константинович ПЕТРЕНКО –доктор физико-математических наук, профессор, начальник отдела ИСП РАН, профессор МГУ и ВШЭ. Область интересов: формальные методы программной инженерии, тестирование программного и аппаратного обеспечения, формальная спецификация требований.

Alexander Konstantinovich PETRENKO – Doctor of Physical and Mathematical Sciences, Professor, Head of the Department of ISP RAS, Professor of Moscow State University and Higher School of Economics. Areas of interest: formal methods of software engineering, testing of software and hardware, formal specification of requirements.

Алексей Владимирович ХОРОШИЛОВ, ведущий научный сотрудник, кандидат физико-математических наук, директор Центра верификации ОС Linux в ИСП РАН, доцент кафедр системного программирования МГУ, ВШЭ и МФТИ. Основные научные интересы: методы проектирования и разработки ответственных систем, формальные методы программной инженерии, методы верификации и валидации, тестирование на основе моделей, методы анализа требований, операционная система Linux.

Alexey Vladimirovich KHOROSHILOV, Leading Researcher, Ph.D. in Physics and Mathematics, Director of the Linux OS Verification Center at ISP RAS, Associate Professor of System Programming Departments at Moscow State University, the Higher School of Economics, and Moscow Institute of Physics and Technology. Main research interests: design and development methods for critical systems, formal methods of software engineering, verification and validation methods, model-based testing, requirements analysis methods, Linux operating system.

Илья Викторович ЩЕПЕТКОВ – стажер-исследователь. Область интересов: разработка и верификация формальных моделей компьютерных систем

Ilya Viktorovich SHCHEPETKOV – intern researcher. Fields of Interest: development and verification of formal models of computer systems