DOI: 10.15514/ISPRAS-2020-32(1)-2

Модель мандатного контроля целостности в операционной системе KasperskyOS

В. С. Буренков, ORCID: 0000-0002-0232-774X <Vladimir.Burenkov@kaspersky.com>
Д. А. Кулагин, ORCID: 0000-0001-5055-0826 <Dmitry.Kulagin@kaspersky.com>

АО «Лаборатория Касперского»,
125212, Россия, Москва, Ленинградское шоссе, д. 39А, стр. 3

Аннотация. Существующие модели мандатного контроля целостности в операционных системах накладывают ограничения на доступы активных компонент системы к пассивным и представляют эти доступы непосредственно. Такое представление можно использовать в случае операционных систем с монолитной архитектурой, где части системы, обеспечивающие доступ к ресурсам, входят в доверенную вычислительную базу. Однако эти части являются нетривиальными компонентами со сложной реализацией, в связи с чем доказательство отсутствия в них ошибок (уязвимостей) и, следовательно, соответствия модели реальной системе – чрезвычайно трудная задача, которая на практике, как правило, полностью не решается. В данной статье представлена модель мандатного контроля целостности для микроядерной операционной системы KasperskyOS. Базирование системы на микроядре предполагает минимизацию доверенной вычислительной базы. При таком подходе доступ к ресурсам предоставляется с помощью компонент, не входящих в доверенную вычислительную базу. Это создает предпосылки для гарантии соответствия системы определенным свойствам безопасности даже в случае некорректного функционирования этих компонент. В модели для представления частей системы, обеспечивающих доступ к ресурсам, введено понятие драйвера объектов, и операции получения доступа активных компонент (сущностей) к пассивным компонентам (объектам) выполняются посредством драйверов объектов. В статье определены требования, которым должны удовлетворять драйверы объектов и некоторые другие сущности. В модель также добавлены элементы, позволяющие провести ее анализ в случае нарушения данных требований. Сформулирована и доказана теорема о постоянном нахождении модели в целостном состоянии (об отсутствии в модели информационных потоков от менее целостных компонентов к более целостным либо несравнимым) в случае удовлетворения всеми сущностями сформулированных требований, а также об ограниченном характере распространения эффекта от нарушения целостности системы в случае наличия в системе компонентов, нарушающих требования. Корректная реализация модели гарантирует, что если компрометирована часть системы, уровень целостности компонентов которой ограничен некоторыми уровнями целостности, то это не приведет к компрометации части системы с более высокими или несравнимыми уровнями целостности. Описан язык спецификации политик мандатного контроля целостности, разработанный в соответствии с правилами модели, и приведен пример использования языка для задания политики безопасности системы, работающей под управлением KasperskyOS. Целью политики является обеспечение безопасности процесса обновления системы.

Ключевые слова: мандатный контроль целостности; операционная система; KasperskyOS.

Для цитирования: Буренков В.С., Кулагин Д.А. Модель мандатного контроля целостности в операционной системе KasperskyOS. Труды ИСП РАН, том 32, вып. 1, 2020 г., стр. 27-56. DOI: 10.15514/ISPRAS–2020–32(1)–2

A Mandatory Integrity Control Model for the KasperskyOS Operating System

V. S. Burenkov, ORCID: 0000-0002-0232-774X <Vladimir.Burenkov@kaspersky.com>
D. A. Kulagin, ORCID: 0000-0001-5055-0826 <Dmitry.Kulagin@kaspersky.com>

AO Kaspersky Lab.,
39A, building 3, Leningradskoe shosse, Moscow, 125212, Russia

Abstract. Existing models of mandatory integrity control in operating systems restrict accesses of active components of a system to passive ones and represent the accesses directly: subjects get read or write access to objects. Such a representation can be used in modeling of monolithic operating systems whose components that provide access to resources are part of the trusted computing base. However, the implementation of these components is extremely complex. Therefore, it is arduous to prove the absence of bugs (vulnerabilities) in them. In other words, proving such a model to be adequate to the real system is nontrivial and often left unsolved. This article presents a mandatory integrity control model for a microkernel operating system called KasperskyOS. Microkernel organization of the system allows us to minimize the trusted computing base to include only the microkernel and a limited number of other components. Parts of the system that provide resource access are generally considered untrusted. Even if some of them are erroneous, the operating system can still provide particular security guarantees. To prove that by means of a model, we introduce the notion of object drivers as intermediaries in operations on objects. We define the requirements that object drivers must satisfy. We also add the means for analysis of the consequences of violations of the requirements. We state and prove that the model either preserves integrity if all active components satisfy the requirements, or restricts the negative impact if some of the components are compromised. Correct implementation of the model guarantees that compromised components will not affect components with higher or incomparable integrity levels. We describe a policy specification language developed in accordance with the model. We provide an example of using it to describe a security policy that ensures a correct update of a system operated by KasperskyOS.

Keywords: mandatory integrity control; operating system; KasperskyOS.

For citation: Burenkov V.S., Kulagin D.A. A Mandatory Integrity Control Model for the KasperskyOS Operating System. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 1, 2020. pp. 27-56 (in Russian). DOI: 10.15514/ISPRAS–2020–32(1)–2

1. Введение

Контроль целостности компьютерных систем является одним из фундаментальных вопросов, рассматриваемых при разработке безопасных систем. Для обеспечения целостности в операционных системах реализуются механизмы *мандатного контроля целостности* [1]. Их реализация должна быть частью доверенной вычислительной базы.

Классический путь разработки механизмов безопасности начинается с создания модели системы [2]. Модель должна отражать существенные особенности разрабатываемой операционной системы. Достоинством наличия модели является возможность строгого доказательства того, что модель обладает определенными свойствами, которыми должна также обладать и сама система. Реализация модели приведет к получению требуемых механизмов безопасности. Если модель реализована корректно, то и система будет обладать доказанными ранее свойствами.

Существуют и официальные требования наличия формальной модели политики безопасности. Их предъявляет, например, ФСТЭК России [3].

В данной работе рассматривается разработка модели мандатного контроля целостности для ее реализации в новой микроядерной операционной системе KasperskyOS.

Концепция микроядерных операционных систем зародилась достаточно давно [4]. Также хорошо известны ее преимущества с точки зрения безопасности [5, 6, 7], главное из которых – уменьшение доверенной базы за счет вынесения большинства драйверов из

адресного пространства ядра в непривилегированный код. Несмотря на известность микроядерной архитектуры, существующие подходы к моделированию свойств безопасности операционных систем не учитывают ее специфические свойства. Так, в субъект-объектных [8] моделях управления доступом и информационными потоками получение доступов активных компонент системы к ресурсам представлено непосредственно (рис. 1). Это связано с тем, что данные модели разрабатываются для операционных систем с монолитной архитектурой, где части системы, обеспечивающие доступ к ресурсам (например, драйверы устройств и файловые системы), выполняются в адресном пространстве ядра. Это вынуждает считать их доверенными, и, как следствие, пренебрегать ими при моделировании. Однако эти части являются достаточно сложными программными компонентами, в которых высока вероятность наличия уязвимостей. По этой причине их включение в доверенную вычислительную базу означает использование предположений, которые в действительности могут быть не выполнены.

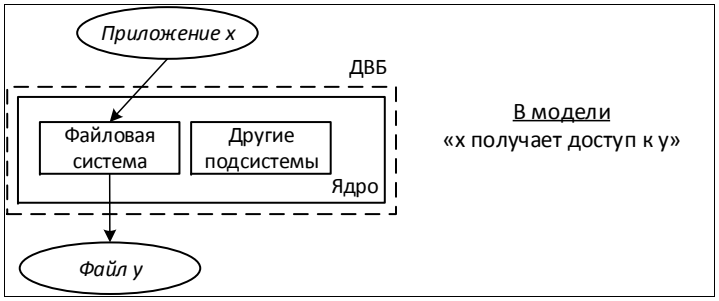

Рис. 1. Схема получения доступа к ресурсу (файлу) в монолитной операционной системе и ее трактовка в существующих формальных моделях

Fig. 1. Resource access scheme in a monolithic operating system and its interpretation in existing formal models

В отличие от монолитных операционных систем, доступ к ресурсам в рамках микроядерной операционной системы KasperskyOS происходит посредством отдельных процессов – драйверов ресурсов. Поскольку это обычные пользовательские процессы, отпадает необходимость считать их доверенными.

Рассмотрение драйверов ресурсов как обычных процессов позволяет также минимизировать начальное состояние модели системы, не включая в него большинство драйверов. Это дает возможность моделирования безопасности системы, начиная с самых ранних этапов ее загрузки: с того момента, когда загружены только ядро, модуль безопасности и очень ограниченный набор доверенных драйверов. В моделях безопасности для монолитных операционных систем достаточно сложный этап начальной загрузки системы не рассматривается, а созданные в процессе загрузки компоненты считаются частью начального состояния модели.

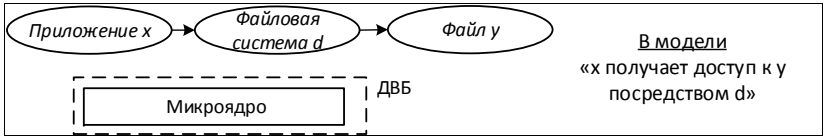

Рис. 2. Схема получения доступа к ресурсу (файлу) в микроядерной системе и необходимый вид ее трактовки в соответствующих формальных моделях

Fig. 2. Resource access scheme in a microkernel operating system and the way it should be interpreted in formal models

В то же время наличие недоверенных драйверов ресурсов означает, что при моделировании доступов к ресурсам необходимо всегда принимать во внимание, кроме

субъекта-инициатора запроса и объекта-ресурса, еще и субъект-драйвер – посредник при любом доступе к ресурсам определенного типа (рис. 2). Следовательно, чтобы в полной мере воспользоваться преимуществами микроядерной архитектуры, нужно расширить и уточнить существующие подходы к моделированию информационных потоков.

Несмотря на то, что драйверы рассматриваются как недоверенные приложения, они должны выполнять функции, присущие драйверам, и, следовательно, удовлетворять определенным требованиям. Однако если нет уверенности в том, что используемый драйвер будет отвечать этим требованиям, и требования заложены в основе модели мандатного контроля целостности, то невозможно сделать заключение о соответствии реальной системы и модели. В качестве решения этой проблемы в работе предлагается учитывать в модели возможность нарушения требований с тем, чтобы провести детальный анализ последствий такого нарушения.

Основными результатами данной работы являются следующие.

1. Разработана модель политики безопасности управления доступом и информационными потоками в KasperskyOS, описывающая мандатный контроль целостности. В отличие от существующих моделей, детально проработаны правила доступа к ресурсам посредством драйверов ресурсов, которые не обязательно являются доверенными.
2. Несмотря на то, что драйверы ресурсов и большая часть других сущностей не вносятся в доверенную вычислительную базу, свойства безопасности, обеспечиваемые моделью, зависят от конкретных аспектов поведения этих сущностей. Для учета этого обстоятельства в работе определены требования, которым должны удовлетворять драйверы ресурсов и другие сущности, участвующие в межпроцессном взаимодействии. Для анализа последствий нарушения этих требований определены дополнительные информационные потоки, возникающие в случае невыполнения требований.
3. Сформулирована и доказана теорема, утверждающая, что
   - если все сущности удовлетворяют сформулированным требованиям, то отсутствуют запрещенные информационные потоки от менее целостных компонент к более целостным или несравнимым,
   - либо, в случае присутствия сущностей, не удовлетворяющих сформулированным требованиям, характер нарушения целостности системы ограничен максимальными уровнями целостности компонентов, нарушающих требования.

   Таким образом, корректная реализация модели гарантирует, что если компрометирована часть системы вплоть до некоторого уровня целостности, то это не приведет к компрометации части системы с большими или несравнимыми уровнями целостности.
4. Помимо этого, описан язык спецификации политик мандатного контроля целостности, разработанный в соответствии с правилами модели, и приведен пример использования языка для задания политики безопасности системы с безопасным процессом обновления.

2. Существующие модели мандатного контроля целостности

Мандатный контроль целостности впервые был теоретически описан в работе Биба (K. Biba) [9]. Липнер (S. Lipner) [10]. Кларк и Уилсон (D. Clark, D. Wilson) [11], Ли (T. Lee) [12] представили ранние рассуждения о контроле целостности в коммерческих системах обработки данных. Брювер и Нэш (D. Brewer, M. Nash) [13] разработали другую классическую модель безопасности коммерческих систем, затрагивающую вопросы целостности и конфиденциальности.

Современные модели зачастую основаны на классических моделях целостности и информационных потоков. Так, Лиу (Z. Liu) и др. [14] описывают простую модель контроля целостности в стиле Белла-ЛаПадулы (D. Bell, L. LaPadula) [15].

Модели целостности Usable Mandatory Integrity Protection (UMIP) [16] и SecGuard [17] разработаны с целью простоты использования и реализованы в прототипах, основанных на ядре Linux. Описание моделей не формализовано в достаточной степени. Помимо этого, в них применяются потенциально опасные решения. Модели позволяют понижать уровень целостности субъекта, однако не рассматривают случаи, когда у такого субъекта есть открытые на запись высокоцелостные объекты. Авторы этих моделей также предлагают использовать дискреционные права доступа для задания уровней целостности. Однако при таком подходе ошибка в конфигурации дискреционных прав доступа приведет к бесполезности мандатного контроля целостности.

Модель, разработанная П.Н. Девяниным и реализованная в операционной системе Astra Linux [18], достаточно проработана теоретически. Имеются математические доказательства свойств модели. Более того, модель формализована и верифицирована в системе Event-B [19, 20]. Модель была разработана применительно к ядру Linux и подразумевает, что ядро, которое включает в себя большое количество компонент, включая драйверы и файловые системы, является доверенным. Это привело к достаточно большому количеству предположений о начальном состоянии модели, что не отвечает целям настоящей работы.

Операционная система Microsoft Windows реализует мандатный контроль целостности [21], основанный на модели Биба, начиная с версии Windows Vista (2007). Имеются положительные оценки соответствующих механизмов.

Таким образом, в литературе отсутствуют модели мандатного контроля целостности, позволяющие минимизировать доверенную вычислительную базу, в частности, вынести драйверы из пространства ядра. Данная работа восполняет этот пробел.

3. Архитектура операционной системы KasperskyOS

Модель, представленная в данной работе, была разработана для реализации в микроядерной операционной системе KasperskyOS. Основными компонентами операционной системы являются микроядро, монитор безопасности и множество драйверов, реализованных в виде непривилегированных приложений. Главными функциями микроядра являются разделение ресурсов между процессами (которые в рамках операционной системы именуются *сущностями*), предоставление механизма межпроцессного взаимодействия IPC (interprocess communication) и запуск монитора безопасности для проверки этих взаимодействий. Операционная система разрабатывалась с учетом принципа разделения ресурсов и приложений посредством минимальной доверенной вычислительной базы, направленного на поддержку политик безопасности. Этот подход известен с 1980-х годов [22]. Его современным воплощением является архитектура MILS [23].

Важным компонентом архитектуры MILS является *ядро разделения*. Если операционная система корректно реализует ядро разделения, то она гарантирует изоляцию приложений (возможно, с различными уровнями доверия) друг от друга. Изоляция приложений очень важна с точки зрения безопасности, так как она позволяет ограничить нежелательное воздействие на систему в целом, которое может оказать отдельное приложение.

Однако, во многих случаях одного только ядра разделения недостаточно, особенно, когда взаимодействуют сложные компоненты с разными уровнями доверия. В такой ситуации требуется убедиться, что взаимодействия в системе удовлетворяют требованиям безопасности (часто сложным и разнообразным). В KasperskyOS для решения этой проблемы существует Kaspersky Security System – набор инструментов для спецификации

самых разных свойств безопасности и синтезирования монитора обращений, проверяющего любые взаимодействия внутри системы на предмет соответствия политике. Для спецификации свойств (политики) используется декларативный язык Policy Specification Language (PSL). Описанная на нем политика транслируется в запускаемый образ монитора безопасности, которому ядро и другие сервисы могут направлять свои запросы.

Подход имеет много общего с архитектурой FLASK [24]. Прежде всего это:

- абстрагирование ресурсов и процессов как доменов безопасности; это позволяет монитору применять общие правила по контролю, не обращая внимание на их различную природу (когда это допустимо);
- разделение вычисления политики и механизма, инициирующего проверку события, передающего связанный с событием контекст и интерпретирующего результат проверки. Это одна из ключевых идей FLASK, позволяющая контролировать не только системные события, но и события прикладного уровня.

В KasperskyOS, кроме этого, каждый сервис должен статически декларировать свои интерфейсы. Это делает структуру передаваемых сообщений прозрачной для монитора безопасности, существенно расширяя выразительные возможности языка спецификации политик: монитор безопасности может проверить, что структура сообщения соответствует объявленному интерфейсу, проанализировать части сообщения и связать определенные правила с данным взаимодействием. Для спецификации свойств безопасности можно одновременно использовать разные модели/формализмы. Например, ролевой доступ или политики на основе конечных автоматов. Разные правила комбинируются всегда с помощью конъюнкции – чтобы какое-либо событие было разрешено, необходимо, чтобы оно было разрешено всеми правилами, ассоциированными с этим событием. В данной работе рассматривается мандатный контроль целостности, который реализован как формализм, который можно использовать совместно с другими для спецификации политики безопасности решения.

Как было отмечено, каждая сущность или ресурс является *доменом безопасности*, с которым ассоциирован *контекст безопасности*, определяемый *идентификатором*. Таким образом, с точки зрения политики безопасности, программно-аппаратная система, управляемая KasperskyOS, представляет собой множество доменов безопасности и информационных потоков между ними.

3.1 Взаимодействие между процессами

В KasperskyOS сущности обмениваются друг с другом сообщениями посредством механизма синхронного межпроцессного взаимодействия. Выделяются две роли: клиент (сущность, инициирующая взаимодействие) и сервер (сущность, обрабатывающая обращение). Клиент направляет серверу сообщение-запрос, при этом поток исполнения, из которого производится обращение, блокируется до получения ответа от сервера или ядра (в случае, например, ошибки). Серверный поток исполнения находится в ожидании сообщений. При получении запроса, этот поток получает управление, обрабатывает запрос и отсылает сообщение-ответ. При получении сообщения-ответа клиентский поток разблокируется и может продолжать исполнение.

Предположим, что поток T_1 сущности P_1 вызывает метод, объявленный в интерфейсе другой сущности P_2 (рис. 3). В этом случае P_1 отправляет *запрос* сущности P_2. В этот момент T_1 приостанавливает свое исполнение, ожидая ответа от P_2. Запрос проходит через механизм IPC, реализованный в микроядре. Механизм IPC отправляет сообщение монитору безопасности, который обрабатывает его согласно политике безопасности. Если политика безопасности разрешает выполнение запроса, то запрос направляется сущности P_2. P_2 затем обрабатывает запрос (в соответствии с реализацией метода) и отправляет

ответ сущности P_1. Ответ может содержать данные, вычисленные сущностью P_2 или просто сигнал о завершении обработки запроса. Ответ затем проходит путь через механизм IPC и монитор безопасности. Монитор проверяет, может ли ответ быть доставлен согласно политике безопасности. Если да, то P_1 получает ответ и T_1 продолжает свое исполнение. Если отправка запроса или ответа запрещена монитором безопасности, то попытка взаимодействия между P_1 и P_2 прерывается и T_1 продолжает свое исполнение.

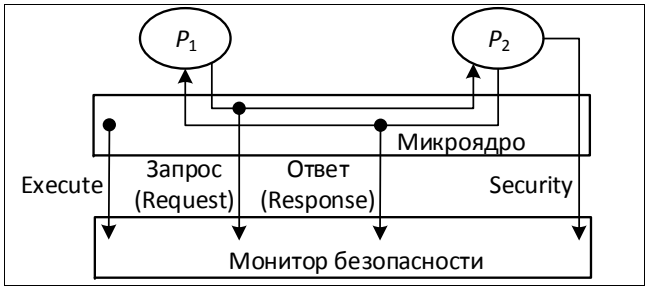

Рис. 3. Схема межпроцессного взаимодействия
Fig. 3. Interprocess communication scheme

Монитор безопасности имеет возможность контролировать любые IPC сообщения. С этой целью ядро передает монитору на проверку как сообщение-запрос, так и ответ. Передача сообщения происходит только в случае, если монитор дал разрешение. Кроме операций по контролю запросов и ответов, монитор также поддерживает две специальных операции – execute и security.

Операция *execute* используется ядром для сообщения монитору о создании новой сущности. Операция *security* позволяет сущностям отправлять сообщения монитору безопасности напрямую. Данная операция является основой для реализации сервисов (драйверов), которые предоставляют контролируемый доступ к ресурсам.

3.2 Управление доступом к ресурсам

В KasperskyOS большинство сервисов реализуется как непривилегированные приложения. Такие сервисы часто предоставляют доступ к ресурсам определенного вида. Смысл понятия «ресурс» определяется самим сервисом. Например, для файловой системы ресурсами могут быть файлы и каталоги. Для сетевой подсистемы – интерфейсы или сокеты. Поскольку данные сервисы не входят в ядро операционной системы, то контролировать доступ к ним только возможностями ядра невозможно – ядро операционной системы ничего не знает ни о семантике этих ресурсов, ни о том, какие операции над ними возможны. Тем не менее, задача разграничивать доступ к таким ресурсам возникает, и для ее решения в KasperskyOS выработан следующий поход.

Ключевым понятием является *драйвер* – это процесс-сервис, который вводит в систему новый тип ресурсов. Например, драйвер файловой системы вводит тип ресурсов «файл». Доступ к ресурсам определенного типа возможен только путем обращения к соответствующему драйверу. С каждым ресурсом драйвер ассоциирует системный дескриптор, с которым подсистема безопасности связывает информацию, необходимую для контроля доступа (например, метку целостности или роль).

Во время каждого доступа к ресурсам драйвер ресурсов должен предоставлять монитору безопасности необходимый контекст. Монитор безопасности затем разрешает или запрещает доступ. При доступе к ресурсу, драйвер использует операцию security для обращения к подсистеме безопасности. В этом обращении драйвер передает дескриптор ресурса и любую дополнительную информацию, которую посчитает необходимой для

осуществления контроля. Например, при чтении файла передается информация о том, кто (дескриптор клиента) хочет выполнить операцию (чтение файла), над чем (дескриптор файла). Монитор безопасности применяет ассоциированные с этим обращением правила и возвращает результат проверки. Ответственность драйвера – правильно проинтерпретировать ответ монитора: заблокировать операцию, если она не была разрешена. Таким образом, для реализации контроля над ресурсами, драйвер должен предоставить механизм, а политика определяется монитором безопасности. Такая архитектура позволяет подсистеме безопасности единообразно трактовать как системные ресурсы (в первую очередь, сами процессы-сущности и IPC-каналы), так и прочие ресурсы, введенные в систему драйверами. Само ядро операционной системы при этом абстрагировано от таких ресурсов и занимается только управлением (выдачей и квотированием) системных дескрипторов.

Ядро операционной системы гарантирует разделение адресных пространств сущностей, поэтому драйвер может управлять только теми ресурсами, системные дескрипторы на которые были выданы именно этому драйверу.

Безусловно, уровень доверия (в самом общем смысле) некоторого ресурса не может быть выше, чем уровень доверия драйвера этого ресурса. Любая политика безопасности должна учитывать это обстоятельство. Это становится важным в случае, когда драйвер является отдельным приложением, не входящим в доверенную вычислительную базу.

Рис. 4 иллюстрирует типовой случай выполнения доступа к ресурсу (цифрами обозначен порядок выполнения операций). Здесь сущность *P*, Драйвер и его Ресурс являются отдельными доменами безопасности. Сущность *P*, которая хочет получить доступ к ресурсу отправляет соответствующее сообщение драйверу этого ресурса. В этом сообщении ресурс может быть идентифицирован различными способами, например, по имени. Драйвер ресурса вычисляет идентификатор ресурса. Затем драйвер ресурса отправляет сообщение монитору безопасности, предоставляя контекст: дескриптор инициатора запроса *P*, дескриптор ресурса и другую информацию. Монитор безопасности определяет, может ли *P* получить доступ к ресурсу согласно политике безопасности. Если доступ разрешен, то драйвер получает доступ к ресурсу и отправляет ответ сущности *P*. Ответ содержит данные, полученные по результатам доступа к ресурсу, и идентификатор ресурса, который может быть использован сущностью в дальнейшем.

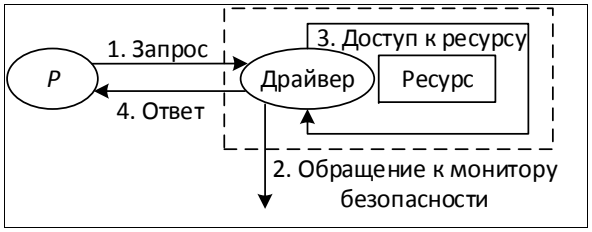

Рис. 4. Схема обращения к ресурсу посредством его драйвера
Fig. 4. Resource access via its driver

4. Модель мандатного контроля целостности

4.1 Информационные потоки

Нахождение системы в целостном состоянии означает, что сущности системы работают корректно. Для обеспечения их корректной работы механизмы мандатного контроля целостности должны обеспечить независимость сущностей от данных с меньшим уровнем доверия (за исключением особенных, специально определенных случаев).

Для моделирования влияния данных в системе на сущности и объекты, а также для формализации свойств мандатного контроля целостности, в работе используется понятие информационного потока.

Информационным потоком от источника к приемнику называется преобразование данных в приемнике, зависящее от данных, находящихся в источнике [8]. В качестве как источника, так и приемника может выступать сущность или объект. Информационный поток от источника x к приемнику y может возникнуть в результате выполнения различных операций, например, когда сущность x получает доступ на запись в объект y или когда сущность y получает доступ на чтение к объекту x. Во всех случаях поток будем обозначать $(x, y, write_m)$ – информация передается от x к y (x «записывает» в y, отсюда название потока $write$). Индекс «m» подчеркивает, что рассматриваются информационные потоки по памяти.

4.2 Основные компоненты модели

4.2.1 Сущности и объекты

S – множество *сущностей* (субъектов) системы, представляющее активные компоненты операционной системы. O – множество *объектов* системы, представляющее пассивные компоненты операционной системы.

Для целей моделирования иерархии объектов введена *функция иерархии объектов* $HO: O \to 2^O$, которая ставит в соответствие каждому объекту $c \in O$ множество объектов $HO(c)$. Будем говорить, что объекты из $HO(c)$ непосредственно находятся в *контейнере* c.

Выделяется множество *корневых объектов* $CR \subseteq O$, не находящихся ни в каком контейнере.

Для целей моделирования управления работой с ресурсами системы посредством драйверов ресурсов, в модели введено понятия *драйвера объектов*. Драйвер объектов – сущность, посредством которой выполняются операции с объектами. Будем говорить, что драйвер объектов управляет подмножеством объектов.

Функция $OD: O \to S$ ставит в соответствие любому объекту системы его драйвер. Обратное отношение $OD^{-1} \subseteq S \times O$ позволяет определить объекты, управляемые данным драйвером.

4.2.2 Доступы и информационные потоки

$R_a = \{read_a, write_a\}$ – множество *видов доступа*, где $read_a$ обозначает доступ на чтение, $write_a$ обозначает доступ на запись.

$R_f = \{write_m\}$ – множество *видов информационных потоков*, где $write_m$ обозначает информационный поток по памяти на запись в сущность или объект.

$A \subseteq S \times O \times R_a$ – множество *доступов* сущностей к объектам.

$F \subseteq (S \cup O) \times (S \cup O) \times R_f$ – множество *информационных потоков*.

4.2.3 Уровни целостности

$(LI, \leq)$ – решетка *уровней целостности*. *Уровень целостности* сущностей и объектов определен с помощью функции $il: S \cup O \to LI$.

При работе операционной системы возникают ситуации, в которых возможно участие сущности в «опасных» взаимодействиях, то есть обращениях на чтение к объектам с более низким либо несравнимым уровнем целостности. Для разрешения таких взаимодействий в модели введена функция $ilr: S \to LI$, определяющая для каждой сущности минимальный

уровень целостности объектов, к которым эта сущность может обращаться на чтение. Модель построена таким образом, что для любой сущности $s \in S$ выполняется $ilr(s) \leq il(s)$.

Для элементов $x, y \in LI$ под $x < y$ будем иметь в виду $x \leq y \wedge x \neq y$.

$UP: S \to \{false, true\}$ – *функция привилегий*, служащая для обозначения сущностей, которым разрешено повышать уровень целостности объектов. Предполагается, что в качестве таких сущностей могут выступать доверенные сущности, для которых значение данной функции устанавливается равным $true$.

4.2.4 Система переходов

В соответствии с распространенным подходом [25], в качестве модели компьютерных систем, управляемых операционной системой, используется система переходов.

Системой переходов называется кортеж

$$TS = (States, OP, \to, G_0),$$

где

- $States$ – множество состояний,
- OP – множество правил преобразования состояний,
- $\to \subseteq States \times OP \times States$ – отношение переходов,
- $G_0 \in States$ – начальное состояние.

Для краткости под обозначением $G \xrightarrow{op} G'$ будем понимать $(G, op, G') \in \to$. Если правило преобразования состояний в данном контексте неважно, будем писать $G \to G'$.

Состояние системы переходов $TS = (States, OP, \to, G_0)$ определено как кортеж

$$G = (S, O, OD, A, F, HO, (il, ilr), UP).$$

Примечание. В рамках модели мы не рассматриваем процесс определения функции UP. Предполагается, что ее значение в каждом состоянии системы переходов определено корректно. Если $UP(s) = true$ для некоторой сущности s, то эта сущность действительно обладает привилегией повышения уровня целостности объектов. Если же сущность s не обладает такой привилегией, то $UP(s) = false$.

4.2.5 Захват контроля над сущностями и объектами

Минимизация доверенной вычислительной базы – в частности, не включение в нее драйверов ресурсов, – приводит к необходимости учета требований к сущностям, не обладающим максимальным уровнем целостности. Однако по каким-либо причинам (например, ошибка в исходном коде сущности) эти требования могут не выполняться. Описание таких причин может быть составлено на основе деталей реализации системы, что находится за рамками модели. При этом, как обсуждалось выше, целесообразно выполнить анализ последствий нарушения требований и возникновения непредвиденных событий в системе в рамках модели. Для этого в модели введено понятие *захвата контроля* над сущностью или объектом. Если над сущностью захвачен контроль, то предположения о ее поведении более не выполняются. Захват контроля над объектом означает появление в нем данных, которые он не должен содержать при корректном функционировании системы. Формально последствия захвата контроля над сущностями и объектами описаны далее в правилах преобразования состояний: если результат выполнения правила зависит от того, захвачен ли контроль над сущностью или объектом, являющимся параметром правила, то сформулирован результат такой зависимости. Далее предполагается, что в каждом состоянии модели определено множество $CS \subseteq S$ сущностей, над которыми захвачен контроль, и множество $CO \subseteq O$ объектов, над которыми захвачен контроль.

Как будет доказано в разделе 4.5, модель исключает неограниченный захват контроля частей системы: если захвачен контроль над частью системы, максимальный уровень целостности компонент которой ниже максимально возможного уровня целостности, то это не приводит к захвату контроля над более целостными компонентами системы и, как следствие, всей системы в целом.

4.3 Требования к начальному состоянию

Представленная модель разрабатывалась таким образом, чтобы к начальному состоянию

$$G_0 = (S_0, O_0, OD_0, A_0, F_0, HO_0, (il_0, ilr_0), UP_0)$$

системы переходов TS предъявлялось как можно меньше требований. Предполагается, что в начальном состоянии существует сущность $core \in S_0$, которая представляет ядро операционной системы, и определены значения уровней целостности $il_0(core)$ и $ilr_0(core)$. Значение $il_0(core)$ является максимально возможным значением уровня целостности: ядро операционной системы является наиболее доверенной сущностью. Остальные множества, определенные в модели, являются пустыми.

В случае необходимости в качестве начального можно использовать и состояние с большим количеством непустых компонент. В таком случае к начальному состоянию модели предъявляются следующие требования.

- Для каждого объекта определен его драйвер:
$$\forall o \in O_0. \exists d \in S_0. OD_0(o) = d.$$
- Для всех объектов $o \in O_0$ определено значение $il_0(o)$.
- Для всех сущностей $s \in S_0$ определены значения $il_0(s)$ и $ilr_0(s)$.
- Уровень целостности каждого объекта не выше уровня целостности его драйвера:
$$\forall o \in O_0. il_0(o) \le il_0\big(OD_0(o)\big).$$
- Для каждого объекта, не являющегося корневым, существует не менее целостный контейнер, в котором находится этот объект:
$$\forall o \in O_0 \setminus CR_0. \exists c. o \in HO_0(c) \wedge il_0(o) \le il_0(c).$$
- Для всех сущностей $s \in S_0$ определено значение $UP_0(s)$.
- Множество доступов является пустым: $A_0 = \emptyset$.
- Множество информационных потоков является пустым: $F_0 = \emptyset$.

4.4 Правила преобразования состояний

В результате анализа операций межпроцессного взаимодействия, возникающих в ряде приложений KasperskyOS, был определен набор правил преобразования состояний, которые могут быть использованы для контроля таких взаимодействий с целью обеспечения целостности системы.

Каждое правило преобразования состояний описывает переход из состояния $G = (S, O, OD, A, F, HO, (il, ilr), UP)$ в состояние $G' = (S', O', OD', A', F', HO', (il', ilr'), UP')$ и представлено с помощью пред- и постусловий. Если все части предусловия правила истинны в состоянии G, то постусловие этого правила определяет отношение между компонентами состояния G' и соответствующими компонентами состояния G. В определении правил указаны только те компоненты G', которые изменяются при осуществлении перехода.

Предусловия правил, моделирующих действия сущностей, должны быть реализованы в операционной системе. В связи с этим предусловия разрабатывались так, чтобы они были по возможности простыми. Постусловия в основном служат для анализа модели.

Правила преобразования состояний принимают параметры, список которых указывается в скобках после имени правила.

Правила, описывающую работу с объектами, разработаны таким образом, что в предусловии каждого из них присутствует требование того, что уровень целостности объекта меньше либо равен уровню целостности драйвера этого объекта. В связи с этим информационный поток от драйвера к объекту не несет угрозы целостности. Информационный поток от объекта к драйверу может оказать влияние на драйвер только в случае захвата контроля над драйвером. Однако в этом случае уже захвачен контроль над не менее целостной сущностью, чем объект, в связи с чем информационный поток от объекта к этой сущности не приведет к расширению зоны захвата (подробнее это разобрано ниже). Эти особенности модели позволяют не рассматривать в правилах потоки от драйверов к их объектам и от объектов к их драйверам.

В постусловиях некоторых правил используются конструкции if-then-else (как правило, для описания эффектов от факта компрометации параметров правила), смысл которых такой же, как и во многих популярных языках программирования.

4.4.1 Получение доступа на чтение к объектам

Операции получения доступа к ресурсам операционной системы выполняются посредством драйверов ресурсов. В корректно работающей системе должно выполняться следующее свойство.

Свойство драйверов ресурсов 0

При доступе сущности к ресурсу драйвер этого ресурса должен инициировать сравнение уровней целостности сущности и ресурса, реализованное в механизме мандатного контроля целостности.

Таким образом, в модели предполагается, что при доступе сущности s к объекту o, если над драйвером объекта o не захвачен контроль, то будет выполнено сравнение уровней целостности сущности s и объекта o. Если над драйвером захвачен контроль, то будем считать, что сравнение уровней выполнено не будет. В связи с этим модель разработана таким образом, чтобы уровень целостности драйвера был не меньше уровня целостности объектов, которыми он управляет, а также выполнялись определенные гарантии по поводу соотношения уровней целостности драйвера объектов и сущности, получающей доступ к этим объектам.

Рассмотрим операцию получения сущностью x доступа на чтение к объекту y. Эта операция выполняется посредством драйвера d объекта y: драйвер получает доступ к объекту и передает данные из объекта в сущность x. Возникающие информационные потоки показаны на рис. 5.

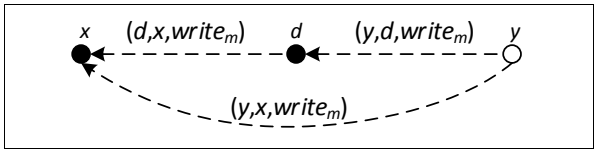

Рис. 5. Возможные информационные потоки при получении сущностью x доступа на чтение к объекту y посредством драйвера d

Fig. 5. Information flows that may arise if entity x has acquired read access to object y via driver d

В корректно работающей системе драйверы ресурсов должны обладать следующим свойством.

Свойство драйверов ресурсов 1

1. Драйвер ресурсов должен быть реализован таким образом, чтобы информационные потоки от ресурсов, которыми он управляет, не влияли на его корректную работу.

2. При доступе на чтение сущностей к ресурсу, управляемому драйвером, драйвер должен передавать данные из ресурса в неизменном виде.

Свойство 1.1 позволяет не учитывать поток $(y, d, write_m)$. Свойство 1.2 позволяет не учитывать поток $(d, x, write_m)$. Из этого свойства следует, что достаточно учитывать только информационный поток $(y, x, write_m)$ от объекта y к сущности x, получающей к нему доступ на чтение.

В случае, если над драйвером d захвачен контроль, d более не обладает свойством 1. Важным становится информационный поток $(d, x, write_m)$, поскольку d может передавать в x произвольные данные. Информационный поток $(y, d, write_m)$ можно не учитывать и в этом случае, поскольку над d и так уже захвачен контроль.

Получение доступа на чтение к данным с большим или равным уровнем целостности является стандартной операцией.

Случай, когда $\neg\big(il(x) \le il(y)\big) \wedge \big(ilr(x) \le il(y)\big)$, и над сущностью x не захвачен контроль, является особенным: предполагается, что сущность x может безопасно считывать менее целостные данные (либо данные, чей уровень целостности не сравним с $il(x)$, но сравним с $ilr(x)$; в дальнейшем будем называть такие данные просто менее целостными), то есть продолжать функционировать согласно своей спецификации. В этом случае информационных потоков к сущности x не возникает (в предположении, что над сущностью x не захвачен контроль).

В случае, если над драйвером d захвачен контроль, сравнение уровней целостности x и y может не выполняться. Однако ядром операционной системы обеспечивается аналогичное сравнение уровней целостности x и d. Ядро операционной системы также гарантирует выполнение условий $OD(y) = d$ и, следовательно, $il(y) \le il(d)$.

Таким образом, правило имеет вид:

$read(x, d, y)$: сущность x получает доступ на чтение к объекту y посредством драйвера объектов d.

Предусловие: $x, d \in S$; $y \in O$; $OD(y) = d$;

$\big(il(x) \le il(d)\big) \vee \Big(\neg\big(il(x) \le il(d)\big) \wedge \big(ilr(x) \le il(d)\big)\Big)$;

$d \notin CS \Rightarrow \big(il(x) \le il(y)\big) \vee \Big(\neg\big(il(x) \le il(y)\big) \wedge \big(ilr(x) \le il(y)\big)\Big)$;

$il(y) \le il(d)$.

Постусловие: $A' = A \cup \{(x, y, read_a)\}$;

if $d \notin CS$ then {

  if $il(x) \le il(y) \wedge il(x) \le il(d) \vee x \in CS$ then $F' = F \cup \{(y, x, write_m)\}$

}

else {

  if $il(x) \le il(d) \vee x \in CS$ then $F' = F \cup \{(y, x, write_m), (d, x, write_m)\}$

}.

4.4.2 Получение доступа на запись к объектам

Рассмотрим операцию получения сущностью x доступа на запись к объекту y. Эта операция выполняется посредством драйвера d объекта y: драйвер получает данные для записи от сущности x, доступ к объекту y и записывает полученные данные в объект. Возникающие информационные потоки показаны на рис. 6.

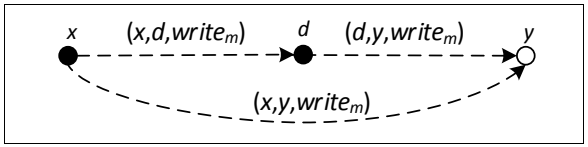

Рис. 6. Возможные информационные потоки при получении сущностью x доступа на запись к объекту y посредством драйвера d

Fig. 6. Information flows that may arise if entity x has acquired write access to object y via driver d

В корректно работающей системе драйвера ресурсов должны обладать следующим свойством.

Свойство драйверов ресурсов 2

1. Драйвер ресурсов должен быть реализован таким образом, чтобы информационные потоки от сущностей, использующих его сервисы по записи в ресурсы, которыми он управляет, не влияли на его корректную работу.
2. При доступе на запись сущностей к ресурсу, управляемому драйвером, драйвер должен записывать в ресурс те же данные, которые ему были переданы сущностью.

Свойство 2.1 позволяет не учитывать поток $(x, d, write_m)$. Свойство 2.2 позволяет не учитывать поток $(d, y, write_m)$.

В случае, если над драйвером d захвачен контроль, d более не обладает свойством 2. Важным становится информационный поток $(x, d, write_m)$, поскольку данные от x могут привести к неспецифицированному поведению драйвера d. Информационный поток $(d, y, write_m)$ можно не учитывать и в этом случае, поскольку уровень целостности объекта y не больше уровня целостности сущности d, и в рамках модели можно описать захват контроля над y: с помощью последовательности правил $write(d, d, y)$ и $control(y)$, которые будут описаны далее.

Таким образом, правило имеет вид:

$write(x, d, y)$: сущность x получает доступ на запись к объекту y посредством драйвера объектов d.

Предусловие: $x, d \in S$; $y \in O$; $OD(y) = d$;

$d \notin CS \Rightarrow il(y) \le il(x)$;

$il(y) \le il(d)$.

Постусловие: $A' = A \cup \{(x, y, write_a)\}$;

if $d \notin CS$ then $F' = F \cup \{(x, y, write_m)\}$

else $F' = F \cup \{(x, y, write_m), (x, d, write_m)\}$.

4.4.3 Создание сущностей и объектов, удаление, перемещение, повышение уровня целостности объектов

$execute(x, y, s, ils, ilsr)$: сущность x создает (запускает) сущность s из объекта y с установлением уровней целостности сущности s.

Предусловие: $x \in S$; $y \in O$; $s \notin S \cup O$; $ils \le il(y)$; $ilsr \le ils$.

Постусловие: $S' = S \cup \{s\}$; $il'(s) = ils$; $ilr'(s) = ilsr$;

if $y \in CO$ then $F' = F \cup \{(y, s, write_m)\}$.

Если над объектом y захвачен контроль, то возникает информационный поток от y к сущности s с тем, чтобы в дальнейшем можно было моделировать распространение захвата контроля с помощью правила $control(s)$.

В правилах $create$, $create\_root$, $move$ и $delete$ возникновение информационных потоков объясняется аналогично правилу $write$.

$create(x, y, z, d, ily)$: сущность x создает объект y посредством драйвера d, включает y в состав контейнера z и устанавливает уровень целостности y в ily.

Предусловие: $x, d \in S$; $y \notin S \cup O$; $z \in O$; $(x, z, write_a) \in A$; $(d, z, write_a) \in A$;
$ily \le il(x)$; $ily \le il(z)$; $ily \le il(d)$.
Постусловие: $O' = O \cup \{y\}$; $HO'(z) = HO(z) \cup \{y\}$; $HO'(y) = \emptyset$; $OD'(y) = d$; $il'(y) = ily$;
if $d \in CS$ then $\{ F' = F \cup \{(x, y, write_m), (x, d, write_m)\}$; $CO' = CO \cup \{y\}$ }.

$create\_root(x, y, d, ily)$: сущность x создает корневой объект y посредством драйвера d и устанавливает уровень целостности y в ily.

Предусловие: $x, d \in S$; $y \notin S \cup O$; $ily \le il(x)$; $ily \le il(d)$.
Постусловие: $O' = O \cup \{y\}$; $CR' = CR \cup \{y\}$; $HO'(y) = \emptyset$; $OD'(y) = d$; $il'(y) = ily$;
if $d \in CS$ then $\{ F' = F \cup \{(x, y, write_m), (x, d, write_m)\}$; $CO' = CO \cup \{y\}$ }.

$move(x, y, d, from, to)$: сущность x перемещает объект y из контейнера $from$ в контейнер to посредством драйвера d.

Предусловие: $x, d \in S$; $y \in O$; $OD(y) = d$; $from, to \in O$; $from \ne to$; $y \ne from$; $y \ne to$;
$y \in HO(from)$;
$(x, from, write_a) \in A$; $(x, to, write_a) \in A$; $(d, from, write_a) \in A$; $(d, to, write_a) \in A$; $d \notin CS \Rightarrow il(y) \le il(x)$;
$il(y) \le il(d)$; $il(y) \le il(to)$.
Постусловие: $HO'(from) = HO(from) \setminus \{y\}$;
$HO'(to) = HO(to) \cup \{y\}$.
if $d \in CS$ then $F' = F \cup \{(x, y, write_m), (x, d, write_m)\}$.

$delete(x, y, d, z)$: сущность x удаляет объект y из контейнера z посредством драйвера d.

Предусловие: $x, d \in S$; $y \in O$; $z \in O$; $y \in HO(z)$; $OD(y) = d$;
$(x, z, write_a) \in A$; $(d, z, write_a) \in A$; $HO(y) = \emptyset$; $il(y) \le il(x)$; $il(y) \le il(d)$.
Постусловие: $O' = O \setminus \{y\}$;
if $y \in CO$ then $CO' = CO \setminus \{y\}$;
$A' = A \setminus \{(s, y, \alpha_a) : s \in S, \alpha_a \in R_a\}$;
$F' = F \setminus (\{(x, y, write_m) : x \in S \cup O\} \cup \{(y, x, write_m) : x \in S \cup O\})$;
$HO'(z) = HO(z) \setminus \{y\}$;
if $d \in CS$ then $F' = F' \cup \{(x, d, write_m)\}$.

$upgrade(x, y, z, d, ily)$: сущность x изменяет уровень целостности объекта y посредством драйвера d, объект y находится в контейнере z, новое значение уровня целостности объекта y равно ily.

Предусловие: $x \in S$; $y \in O$; $OD(y) = d$; $z \in O$; $UP(x) = true$;
$il(y) \le il(x)$; $ily \le il(x)$; $y \in HO(z)$; $ily \le il(z)$; $ily \le il(d)$; $il(y) < ily$.
Постусловие: $il'(y) = ily$.

4.4.4 Взаимодействие сущностей

Наиболее распространенным случаем межпроцессного взаимодействия является вызов одной сущностью метода другой сущности с тем, чтобы вторая сущность выполнила определенное действие и вернула результат. Если сущность x вызывает метод сущности y, то в рамках такого взаимодействия возможны два информационных потока:

- $(x, y, write_m)$ – данные, передаваемые сущностью x при обращении к сущности y за сервисом сущности y (параметры вызываемого метода сущности y).
- $(y, x, write_m)$ – данные, возвращаемые сущности x после того, как вызванный метод сущности y завершил свою работу.

В корректной системе должно выполняться следующее предположение.

Предположение 1. Поведение сущности не должно приводить к некорректному состоянию системы независимо от того, какие данные ей переданы в результате вызова ее метода.

Таким образом, в случае, если над сущностью y не захвачен контроль, то поток $(x, y, write_m)$ учитывать не нужно.

Соотношения уровней целостности сущностей x и y аналогичны случаю получения сущностью доступа на чтение к объекту.

Описанный случай взаимодействия сущностей представлен в модели с помощью правила $call$:

$call(x, y)$: сущность x вызывает метод сущности y с целью получения данных от сущности y.

Предусловие: $x, y \in S$; $\big(il(x) \le il(y)\big) \vee \Big(\neg\big(il(x) \le il(y)\big) \wedge \big(ilr(x) \le il(y)\big)\Big)$.

Постусловие: if $il(x) \le il(y) \vee x \in CS$ then {
if $y \notin CS$ then $F' = F \cup \{(y, x, write_m)\}$
else $F' = F \cup \{(y, x, write_m), (x, y, write_m)\}$ }.

В рамках особого случая межпроцессного взаимодействия сущность передает данные менее целостной сущности. Например, сущность может быть подписана на получение данных от более доверенной сущности. В модели такое взаимодействие задано с помощью правила $invoke$.

При моделировании такого взаимодействия необходимо всегда учитывать информационный поток от сущности, передающей данные к сущности, получающей их. При этом в корректной системе должно выполняться следующее предположение.

Предположение 2. Если сущность вызывает метод другой сущности посредством $invoke$, то данные-ответ, полученные от последней в ходе IPC-взаимодействия, не влияют на корректность работы первой.

Таким образом, если над сущностью x не захвачен контроль, то поток $(y, x, write_m)$ учитывать не нужно.

$invoke(x, y)$: сущность x вызывает метод сущности y с целью передачи данных сущности y.

Предусловие: $x, y \in S$; $il(y) \le il(x)$.
Постусловие: if $x \notin CS$ then $F' = F \cup \{(x, y, write_m)\}$
else $F' = F \cup \{(x, y, write_m), (y, x, write_m)\}$.

4.4.5 Возникновение неявных информационных потоков

Ранее представленные (де-юре) правила моделируют действия сущностей в системе и определяют информационные потоки, возникающие в результате таких действий. Однако информационные потоки также могут возникать неявно, в результате ранее совершенных действий. Для моделирования возникновения таких потоков в модель добавлены правила, основанные на идее *де-факто передач* [26]. Следуя [8, 27], будем называть такие правила де-факто правилами.

В отличие от де-юре правил, де-факто правила не предназначены для реализации в операционной системе и используются для целей, связанных с синтезом и анализом модели.

$pass(x, z, y)$: «z передает (passes) данные из x в y».

Предусловие: $z \in S$; $x \in O$; $y \in S \cup O$; $x \neq y$; $(z, x, read_a) \in A$; $(z, y, write_m) \in F$;

$\neg\left(\neg\left(il(z) \leq il(x)\right) \wedge ilr(z) \leq il(x) \vee \neg\left(il(z) \leq il\left(OD(x)\right)\right) \wedge ilr(z) \leq il\left(OD(x)\right)\right) \vee z \in CS$.

Постусловие: $F' = F \cup \{(x, y, write_m)\}$.

Если сущность z может безопасно считывать менее целостные данные с меньшим либо равным уровнем целостности (из объекта x) и над ней не захвачен контроль, то сущность z безусловно не передает эти данные в сущность или объект y, следовательно, информационный поток от x к y не возникает. В остальных случаях передача данных происходит.

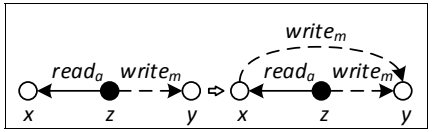

Рис. 7. Реализация информационного потока с помощью правила pass
Fig. 7. Realization of information flow via the pass de facto rule

$post(x, z, y)$: «x отправляет (posts) данные y через z».

Предусловие: $x, y \in S$; $z \in O$; $x \neq y$; $(y, z, read_a) \in A$; $(x, z, write_m) \in F$;

$\neg\left(\neg\left(il(y) \leq il(z)\right) \wedge ilr(y) \leq il(z) \vee \neg\left(il(y) \leq il\left(OD(z)\right)\right) \wedge ilr(y) \leq il\left(OD(z)\right)\right)$.

Постусловие: $F' = F \cup \{(x, y, write_m)\}$.

Элемент предусловия $\neg\left(\neg\left(il(y) \leq il(z)\right) \wedge ilr(y) \leq il(z) \vee \neg\left(il(y) \leq il\left(OD(z)\right)\right) \wedge ilr(y) \leq il\left(OD(z)\right)\right)$ говорит о том, что сущность y получила доступ на чтение к объекту z не как сущность, которая может безопасно считывать менее целостные данные. В ином случае, аналогично правилам $read$ и $call$, поток от x к y не возникает.

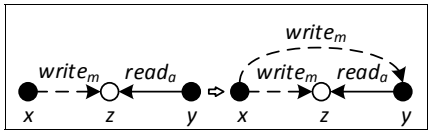

Рис. 8. Реализация информационного потока с помощью правила post
Fig. 8. Realization of information flow via the post de facto rule

$find(x, z, y)$: «y находит (finds) данные из x через z».

Предусловие: $x, z \in S$; $y \in S \cup O$; $x \neq y$; $\{(x, z, write_m), (z, y, write_m)\} \subseteq F$.

Постусловие: $F' = F \cup \{(x, y, write_m)\}$.

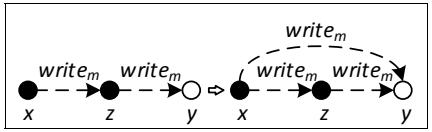

Рис. 9. Реализация информационного потока с помощью правила find
Fig. 9. Realization of information flow via the find de facto rule

4.4.6 Распространение зоны захвата контроля

Для моделирования распространения зоны захвата контроля в модели введено де-факто правило, описывающее изменения множества $CS \cup CO$: если к сущности или объекту имеется поток от сущности или объекта из $CS \cup CO$, то эта сущность или объект попадает в множество сущностей и объектов, над которыми захвачен контроль. Предусловие этого правила проверяет наличие потока от элемента множества $CS \cup CO$, а постусловие добавляет того, к кому этот поток, в множество $CS \cup CO$. Таким образом, наличие потока в текущем состоянии приведет к попаданию во множество захваченных в следующем состоянии. В случае, если захват контроля осуществляется над драйвером объектов, контроль также захватывается и над всеми объектами, которыми управляет этот драйвер.

$control(x)$: захват контроля над сущностью или объектом x.

Предусловие: $x \in (S \cup O) \setminus (CS \cup CO)$; $\exists y \in CS \cup CO.\, (y, x, write_m) \in F$.

Постусловие: $CS' \cup CO' = CS \cup CO \cup \{x\}$;

if $x \in S$ then $CO' = CO' \cup OD^{-1}(x)$.

4.5 Свойства безопасности, обеспечиваемые моделью

Операция $upgrade$, которая должна выполняться в реальной системе только доверенными сущностями, подразумевает отсутствие информационных потоков к объекту, уровень целостности которого предполагается повысить. В противном случае источник такого информационного потока может стать менее целостным, чем приемник. В реальной системе для гарантии отсутствия таких потоков используются специальные средства (например, криптографические). Такие средства не рассматриваются в рамках модели. Поэтому при анализе модели на предмет свойств безопасности, которые она обеспечивает, мы рассматриваем только такие вычисления системы переходов, на которых доверенные сущности не выполняют операции, которые могут нарушить целостность системы.

Назовем *вычислениями без кооперации доверенных и недоверенных сущностей для реализации информационных потоков* такие вычисления системы переходов TS, на которых доверенные сущности не инициируют применение правила $upgrade$.

Теорема 1 показывает, что либо возникающие информационные потоки направлены от не менее целостных сущностей или объектов, либо расширение зоны захвата контроля носит строго ограниченный характер. Ограничение заключается в том, что в этом случае в множестве захваченных компонентов всегда уже имеется сущность с уровнем целостности большим либо равным уровню целостности компонента, к которому реализован информационный поток (рис. 10).

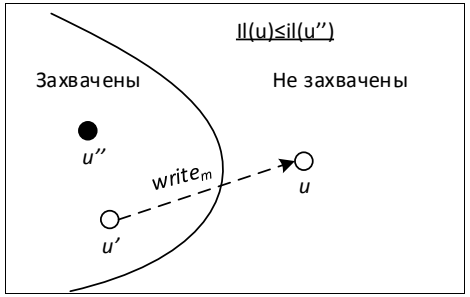

Рис. 10. Ограничение расширения зоны захвата контроля
Fig. 10. Restriction of the area of compromised components

Теорема 1. Для всех состояний $G = (S, O, OD, A, F, HO, (il, ilr), UP) \in States$ системы переходов $TS = (States, OP, \rightarrow, G_0)$, достижимых из начального состояния G_0 и принадлежащих вычислениям системы переходов TS без кооперации доверенных и недоверенных сущностей для реализации информационных потоков, выполняется:

Для всех сущностей и объектов $u, u' \in S \cup O$, если существует информационный поток от u' к u, то либо уровень целостности u меньше либо равен уровню целостности u', либо в множестве сущностей и объектов, находящихся под контролем, существует сущность u'', такая, что уровень целостности u меньше либо равен ее уровню целостности:

$$\forall G \in Reach(TS). \forall u, u' \in S \cup O. (u', u, write_m) \in F \Rightarrow il(u) \leq il(u') \vee \exists u'' \in CS. il(u) \leq il(u'').$$

Доказательство. Доказательство утверждения теоремы проведем методом математической индукции по длине пути системы переходов TS. Выберем произвольный путь $G_0 G_1 G_2 \ldots$ системы TS.

Базис индукции. Длина пути $n = 0$.

В соответствии с требованиями к начальному состоянию, $F_0 = \emptyset$. Следовательно, доказываемое утверждение верно.

Индуктивная гипотеза. Пусть для некоторого $n \geq 0$ верно:

$$\forall k \leq n. \forall u, u' \in S_k \cup O_k. (u', u, write_m) \in F_k \Rightarrow il_k(u) \leq il_k(u') \vee \exists u'' \in CS_k. il_k(u) \leq il_k(u'').$$

Шаг индукции.

Рассмотрим переход $G_n \rightarrow G_{n+1}$ системы переходов TS. Докажем, что

$$\forall u, u' \in S_{n+1} \cup O_{n+1}. (u', u, write_m) \in F_{n+1} \Rightarrow il_{n+1}(u) \leq il_{n+1}(u') \vee \exists u'' \in CS_{n+1}. il_{n+1}(u) \leq il_{n+1}(u'').$$

Рассмотрим произвольные $u, u' \in S_{n+1} \cup O_{n+1}$. Предположим, что $(u', u, write_m) \in F_{n+1}$ (гипотеза). Докажем, что

$$il_{n+1}(u) \leq il_{n+1}(u') \vee \exists u'' \in CS_{n+1}. il_{n+1}(u) \leq il_{n+1}(u'').$$

Рассмотрим все правила, порождающие переход $G_n \rightarrow G_{n+1}$, и для каждого правила – информационные потоки, которые могут возникнуть в результате применения этого правила.

Правило $read(x, d, y)$.

Случай 1. $d \notin CS_n$. Информационный поток $(u', u, write_m) \in F_{n+1}$ из гипотезы – это поток $(y, x, write_m)$. Если $x \notin CS_n$, то, согласно предусловию правила, $il_n(x) \leq il_n(y)$. Следовательно, $il_{n+1}(x) \leq il_{n+1}(y)$. Если $x \in CS_n$, то в качестве u'' можно выбрать x, поскольку правило $read$ не изменяет множество CS и $x \in CS_{n+1}$.

Случай 2. $d \in CS_n$. Информационный поток $(u', u, write_m) \in F_{n+1}$ из гипотезы – это один из следующих потоков.

$(y, x, write_m)$. Если $x \notin CS_n$, то $il_n(x) \leq il_n(d)$. Следовательно, $il_{n+1}(x) \leq il_{n+1}(d)$, и в качестве u'' можно выбрать d. Если $x \in CS_n$, то в качестве u'' можно выбрать x, поскольку правило $read$ не изменяет множество CS и $x \in CS_{n+1}$.

$(d, x, write_m)$. Если $x \notin CS_n$, то $il_n(x) \leq il_n(d)$. Следовательно, $il_{n+1}(x) \leq il_{n+1}(d)$, и в качестве u'' можно выбрать d. Если $x \in CS_n$, то в качестве u'' можно выбрать x.

Правило $write(x, d, y)$.

Случай 1. $d \notin CS_n$. Информационный поток $(u', u, write_m) \in F_{n+1}$ из гипотезы – это поток $(x, y, write_m)$. Согласно предусловию правила, $il_n(y) \leq il_n(x)$. Следовательно, $il_{n+1}(y) \leq il_{n+1}(x)$.

Случай 2. $d \in CS_n$. Информационный поток $(u', u, write_m) \in F_{n+1}$ из гипотезы – это один из следующих потоков.

$(x, y, write_m)$. Согласно предусловию, $il_n(y) \leq il_n(d)$. Следовательно, $il_{n+1}(y) \leq il_{n+1}(d)$, и в качестве u'' можно выбрать d.

$(x, d, write_m)$. В качестве u'' можно выбрать саму сущность d.

Правило $execute(x, y, s, ils, ilsr)$. Информационный поток $(u', u, write_m) \in F_{n+1}$ из гипотезы – это поток $(y, s, write_m)$. Согласно предусловию правила, $ils \leq il_n(y)$. Следовательно, $il_{n+1}(s) \leq il_{n+1}(y)$.

Правила $create(x, y, z, d, ily)$, $create\_root(x, y, d, ily)$. Информационный поток $(u', u, write_m) \in F_{n+1}$ из гипотезы – это один из следующих потоков.

$(x, y, write_m)$. Согласно предусловию, $ily \leq il_n(x)$. Следовательно, $il_{n+1}(y) \leq il_{n+1}(x)$.

$(x, d, write_m)$. В качестве u'' можно выбрать саму сущность d.

Правило $move(x, y, d, from, to)$. Информационный поток $(u', u, write_m) \in F_{n+1}$ из гипотезы – это один из следующих потоков.

$(x, y, write_m)$. Так как при этом $d \in CS_n$, то, в качестве сущности u'' можно выбрать сущность d, поскольку в соответствии с предусловием $il_n(y) \leq il_n(d)$.

$(x, d, write_m)$. В качестве u'' можно выбрать саму сущность d.

Правило $delete(x, y, d, z)$.

Информационный поток из гипотезы – это поток $(x, d, write_m)$, возникающий, если $d \in CS_n$. В этом случае $d \in CS_{n+1}$, и эту сущность можно выбрать в качестве u''.

Правило $control$ не добавляет информационные потоки рассматриваемого вида, не изменяет уровни целостности и не удаляет сущности. Поэтому в соответствии с индуктивной гипотезой в этом случае утверждение шага индукции оказывается истинным.

Правило $call(x, y)$.

Случай 1. $y \notin CS_n$. Информационный поток из гипотезы – это поток $(y, x, write_m)$. Если $x \notin CS_n$, то $il_n(x) \leq il_n(y)$. Следовательно, $il_{n+1}(x) \leq il_{n+1}(y)$. Если $x \in CS_n$, то в качестве u'' можно выбрать x.

Случай 2. $y \in CS_n$. Информационный поток $(u', u, write_m) \in F_{n+1}$ из гипотезы – это один из следующих потоков.

$(y, x, write_m)$. Если $x \notin CS_n$, то $il_n(x) \leq il_n(y)$. Следовательно, $il_{n+1}(x) \leq il_{n+1}(y)$. Если $x \in CS_n$, то в качестве u'' можно выбрать x.

$(x, y, write_m)$. В качестве u'' можно выбрать саму сущность y.

Правило $invoke(x, y)$.

Случай 1. $x \notin CS_n$. Информационный поток из гипотезы – это поток $(x, y, write_m)$. Согласно предусловию, $il_n(y) \leq il_n(x)$. Следовательно, $il_{n+1}(y) \leq il_{n+1}(x)$.

Случай 2. $x \in CS_n$. Информационный поток $(u', u, write_m) \in F_{n+1}$ из гипотезы – это один из следующих потоков.

$(x, y, write_m)$. Поскольку $il_n(y) \leq il_n(x)$, то $il_{n+1}(y) \leq il_{n+1}(x)$.

$(y, x, write_m)$. В качестве u'' можно выбрать саму сущность x.

Правило $pass(x, z, y)$. Информационный поток $(u', u, write_m) \in F_{n+1}$ из гипотезы – это поток $(x, y, write_m)$.

Согласно предусловию правила $pass$, $(z, x, read_a) \in A_n$. Анализ постусловий всех правил показывает, что доступ $(z, x, read_a) \in A_n$ мог быть получен только с помощью правила $read(z, OD(x), x)$, где $OD(x)$ – драйвер объекта x. Другими словами, одним из $op_i, i = 1, \ldots, n$ на вычислении $G_0 op_1 G_1 \ldots op_n G_n$ является $op_l = read(z, OD(x), x), 1 \leq l \leq n$. Поскольку случай, когда сущность z может безопасно получать доступ на чтение менее целостных данных в данном правиле не приводит к возникновению информационных потоков, то, согласно предусловию правила $read$ для случая $z \notin CS_l$ (поскольку драйвер

объекта остается неизменным на вычислениях системы переходов TS, для его обозначения будем просто писать OD, без индекса состояния):

1. если $OD(x) \notin CS_l$, то $il_l(z) \le il_l(x)$;
2. если $OD(x) \in CS_l$, то $il_l(z) \le il_l\big(OD(x)\big)$.

Поскольку уровни целостности и факты захвата контроля не меняются на вычислениях системы переходов TS, итого имеем

$$\big(OD(x) \notin CS_n \wedge il_n(z) \le il_n(x)\big) \vee \Big(OD(x) \in CS_n \wedge il_n(z) \le il_n\big(OD(x)\big)\Big) \vee z \in CS_n.$$

Согласно предусловию правила $pass$ также имеем $(z, y, write_m) \in F_n$. Согласно индуктивной гипотезе,

$$il_n(y) \le il_n(z) \vee \exists u'' \in CS_n.\, il_n(y) \le il_n(u'').$$

Если $\exists u'' \in CS_n.\, il_n(u) \le il_n(u'')$, то сразу заключаем $\exists u'' \in CS_{n+1}.\, il_{n+1}(y) \le il_{n+1}(u'')$.

Если $il_n(y) \le il_n(z)$, то имеем три случая:

1. $il_n(y) \le il_n(z) \wedge il_n(z) \le il_n(x)$, следовательно, $il_n(y) \le il_n(x)$, а значит $il_{n+1}(y) \le il_{n+1}(x)$;
2. $il_n(y) \le il_n(z) \wedge il_n(z) \le il_n\big(OD(x)\big)$, следовательно, $il_n(y) \le il_n\big(OD(x)\big)$, а значит $il_{n+1}(y) \le il_{n+1}\big(OD(x)\big)$. Поскольку в этом случае $OD(x) \in CS_{n+1}$, то в качестве искомого $u'' \in CS_{n+1}$ можно взять $OD(x)$;
3. $il_n(y) \le il_n(z) \wedge z \in CS_n$, следовательно, $z \in CS_{n+1} \wedge il_{n+1}(y) \le il_{n+1}(z)$, и в качестве искомого $u'' \in CS_{n+1}$ можно взять z.

Таким образом, проанализированы все возможные случаи, и можно заключить, что

$$il_{n+1}(y) \le il_{n+1}(x) \vee \exists u'' \in CS_{n+1}.\, il_{n+1}(y) \le il_{n+1}(u'').$$

Правило $post(x, z, y)$. Информационный поток $(u', u, write_m) \in F_{n+1}$ из гипотезы – это поток $(x, y, write_m)$.

Согласно предусловию правила $post$, $(y, z, read_a) \in A_n$. Следовательно,

$$\big(OD(z) \notin CS_n \wedge il_n(y) \le il_n(z)\big) \vee \Big(OD(z) \in CS_n \wedge il_n(y) \le il_n\big(OD(z)\big)\Big) \vee y \in CS_n.$$

Согласно предусловию правила $post$ также имеем $(x, z, write_m) \in F_n$. В соответствии с индуктивной гипотезой,

$$il_n(z) \le il_n(x) \vee \exists u'' \in CS_n.\, il_n(z) \le il_n(u'').$$

Если $\exists u'' \in CS_n.\, il_n(z) \le il_n(u'')$, то $\exists u'' \in CS_{n+1}.\, il_{n+1}(z) \le il_{n+1}(u'')$. Имеем три случая:

1. $\exists u'' \in CS_n.\, il_n(z) \le il_n(u'') \wedge il_n(y) \le il_n(z)$, следовательно, $\exists u'' \in CS_n.\, il_n(y) \le il_n(u'')$, а значит $\exists u'' \in CS_{n+1}.\, il_{n+1}(y) \le il_{n+1}(u'')$.
2. $\exists u'' \in CS_n.\, il_n(z) \le il_n(u'') \wedge il_n(y) \le il_n\big(OD(z)\big)$, следовательно, $il_{n+1}(y) \le il_{n+1}\big(OD(z)\big)$. Поскольку в этом случае $OD(z) \in CS_{n+1}$, то в качестве искомого $u'' \in CS_{n+1}$ можно взять $OD(z)$;
3. $\exists u'' \in CS_n.\, il_n(z) \le il_n(u'') \wedge y \in CS_n$, следовательно, $y \in CS_{n+1}$, и в качестве искомого $u'' \in CS_{n+1}$ можно взять саму сущность y.

Если $il_n(z) \le il_n(x)$, то имеем три случая:

1. $il_n(z) \le il_n(x) \wedge il_n(y) \le il_n(z)$, следовательно, $il_n(y) \le il_n(x)$, а значит $il_{n+1}(y) \le il_{n+1}(x)$;
2. $il_n(z) \le il_n(x) \wedge il_n(y) \le il_n\big(OD(z)\big)$, следовательно, $il_{n+1}(y) \le il_{n+1}\big(OD(z)\big)$. Поскольку в этом случае $OD(z) \in CS_{n+1}$, то в качестве искомого $u'' \in CS_{n+1}$ можно взять $OD(z)$;
3. $il_n(z) \le il_n(x) \wedge y \in CS_n$, следовательно, $y \in CS_{n+1}$, и в качестве искомого $u'' \in CS_{n+1}$ можно взять саму сущность y.

Таким образом, проанализированы все возможные случаи, и можно заключить, что

$$il_{n+1}(y) \le il_{n+1}(x) \vee \exists u'' \in CS_{n+1}.\, il_{n+1}(y) \le il_{n+1}(u'').$$

Правило $find(x, z, y)$. Информационный поток $(u', u, write_m) \in F_{n+1}$ из гипотезы – это поток $(x, y, write_m)$.

Согласно предусловию правила, $\{(x, z, write_m), (z, y, write_m)\} \subseteq F_n$. Следовательно, в соответствии с индуктивной гипотезой,

$$\big(il_n(z) \le il_n(x) \vee \exists u'' \in CS_n.\, il_n(z) \le il_n(u'')\big) \wedge \big(il_n(y) \le il_n(z) \vee \exists u'' \in CS_n.\, il_n(y) \le il_n(u'')\big).$$

Итого имеем четыре случая:

1. $il_n(z) \le il_n(x) \wedge il_n(y) \le il_n(z)$, следовательно, $il_{n+1}(y) \le il_{n+1}(x)$;
2. $il_n(z) \le il_n(x) \wedge \exists u'' \in CS_n.\, il_n(y) \le il_n(u'')$, следовательно, $\exists u'' \in CS_{n+1}.\, il_{n+1}(y) \le il_{n+1}(u'')$.
3. $il_n(y) \le il_n(z) \wedge \exists u'' \in CS_n.\, il_n(z) \le il_n(u'')$, следовательно, $\exists u'' \in CS_{n+1}.\, il_{n+1}(y) \le il_{n+1}(u'')$.
4. $\big(\exists u'' \in CS_n.\, il_n(z) \le il_n(u'')\big) \wedge \big(\exists u'' \in CS_n.\, il_n(y) \le il_n(u'')\big)$, следовательно, $\exists u'' \in CS_{n+1}.\, il_{n+1}(y) \le il_{n+1}(u'')$.

Таким образом, проанализированы все возможные случаи, и можно заключить, что

$$il_{n+1}(y) \le il_{n+1}(x) \vee \exists u'' \in CS_{n+1}.\, il_{n+1}(y) \le il_{n+1}(u'').$$

Теорема доказана.

5. Пример использования политики мандатного контроля целостности

Рассмотрим применение политик мандатного контроля целостности для обеспечения процесса безопасного обновления системы, работающей под управлением KasperskyOS. Под «безопасным» в данном случае понимается такое обновление, перед осуществлением которого была проверена цифровая подпись образа обновления. С точки зрения процесса обновления, компоненты системы могут получать и применять обновления. В этом процессе непосредственно принимают участие следующие приложения (сущности):

- `Downloader` загружает образ обновления с удаленного ресурса. Это приложение напрямую взаимодействует с недоверенной удаленной системой, и потенциально над ним может быть захвачен контроль. Поэтому данному приложению назначен низкий уровень целостности.
- `Verifier` проверяет цифровую подпись образа обновления. Это доверенное приложение, которое может считывать низкоцелостный образ обновления с целью проверки его цифровой подписи.
- `Updater` обновляет систему. Это доверенное приложение, которое должно получать данные только из высокоцелостных файлов.
- `FileSystem` реализует файловое хранилище (является драйвером для файлов).

Эти приложения реализуют следующий сценарий. `Downloader` загружает образ обновления и сохраняет его с использованием сервисов `FileSystem`. Затем `Verifier` проверяет цифровую подпись образа обновления. Если подпись является корректной, приложение дает инструкцию `FileSystem` на создание нового высокоцелостного файла, который является копией оригинального образа обновления. Наконец, `Updater` считывает новый файл и применяет обновление.

5.1 Определение интерфейсов

Перед написанием политики безопасности необходимо определить интерфейсы компонентов системы.

Следующий фрагмент определяет интерфейс операционной системы, который используется ядром для сообщения монитору безопасности о создании новых сущностей:

```
interface execute {
  exec (in Handle image);
}
```

Указанный в этом интерфейсе метод `exec` используется для создания новых сущностей. Метод принимает идентификатор образа, из которого необходимо создать новую сущность.

Следующий фрагмент определяет интерфейс, по которому сущность `FileSystem` передает необходимый контекст монитору безопасности:

```
interface FileSystem_Security {
  create (in Handle client,
          in Handle directory,
          in Handle object,
          in Label label);
  // …
}
```

Интерфейс определяет, как `FileSystem` взаимодействует с монитором безопасности для контроля доступа к файлам. В частности, когда `FileSystem` создает новые объекты, оно отправляет сообщение с использованием метода `create`. В ответ сущность получает решение о предоставлении доступа, в соответствии с которым она должна продолжить свою работу. Следовательно, драйвер ресурсов предоставляет контекст и действует согласно решениям о предоставлении доступа к его ресурсам, а монитор безопасности реализует политику безопасности.

Также определяется интерфейс самой сущности `FileSystem`:

```
interface FileSystem {
  // …
  write (in Handle object,
         in Data data,
         out DataLength count,
         out RetCode ret);
  read (in Handle object,
        in DataLength count,
        out Data data,
        out RetCode ret);
}
```

Другие приложения могут использовать этот интерфейс для работы с файлами.

5.2 Определение политики безопасности

Для определения политики безопасности системы необходимо разработать спецификацию на языке PSL. Мандатный контроль целостности представлен в языке *классом политик* `mandatory_integrity_control`, который определяет множество *правил*, соответствующих правилам модели. Спецификация политики безопасности определяет соответствие этих правил и взаимодействий в системе. При каждой попытке взаимодействия монитор безопасности исполняет правила для определения решения о допустимости взаимодействия.

Для использования класса политик необходимо создать на его основе *объект политики*, указав для него конфигурацию. Для объекта класса `mandatory_integrity_control` необходимо указать уровни целостности.

Политика безопасности определяет для каждого межпроцессного взаимодействия, какие правила класса политик будут исполнены монитором для разрешения или запрета взаимодействия. В число таких взаимодействий входят создание сущностей и объектов и другие взаимодействия.

В нашем примере сначала определяется объект политик `integrity` как экземпляр класса `mandatory_integrity_control`:

```
policy object integrity =
  mandatory_integrity_control {
    config : {
      levels : [“LOW”, “MEDIUM”, “HIGH”]
    }
}
```

В конфигурации этого объекта политик указаны три уровня целостности `LOW`, `MEDIUM` и `HIGH`. Если уровни целостности заданы в виде списка значений (как в данном примере), то задан полный порядок на множестве уровней целостности. Следовательно, `LOW < MEDIUM < HIGH`.

Далее задается политика создания сущностей:

```
execute method=exec, dst=Downloader {
  integrity.execute {
    target : dst,
    image  : message.image,
    level  : “LOW” }
}
execute method=exec, dst=Verifier {
  integrity.execute {
    target : dst,
    image  : message.image,
    level  : “HIGH”, levelR: “LOW” }
}
execute method=exec, dst=Updater {
  integrity.execute {
    target : dst,
    image  : message.image,
    level  : “HIGH” }
}
execute method=exec, dst=FileSystem {
  integrity.execute {
    target : dst,
    image  : message.image,
    level  : “HIGH” }
}
```

Язык спецификации политик предоставляет директиву `execute` для назначения правил, которые должны быть исполнены монитором безопасности при создании сущностей, и предоставления монитору необходимого контекста. В данном примере ядро предоставляет контекст посредством метода `exec` интерфейса `execute`. Доступ к этому контексту может быть получен посредством объекта `message` (например, `message.image`). Также неявно передаются два идентификатора: `src` – идентификатор сущности, обратившейся к ядру для создания новой сущности с идентификатором `dst`.

Правило `integrity.execute` имеет следующие параметры: `target` определяет создаваемую сущность, `image` определяет объект, из которого должна быть создана

сущность, `level` и `levelR` задают уровни целостности (ils и $ilsr$ в правиле $execute$) новой сущности. Если `levelR` опущен, то `levelR=level`.

Секция `request` соответствует всем IPC-запросам (первым частям синхронного межпроцессного взаимодействия) и подразумевает два неявным параметра: `src` для инициатора запроса и `dst` для получателя. Следующий фрагмент определяет, что для каждого IPC-запроса, `integrity.call` применяет правило *call* с аргументами `source` (`src`) и `target` (`dst`).

```
request {
  integrity.call {
    source : src,
    target : dst }
}
```

Секция `security` соответствует прямым обращениям сущностей к монитору безопасности. В ней есть неявный параметр `src` – идентификатор этой сущности.

При создании файлов `FileSystem`, как драйвер файловых ресурсов, предоставляет контекст монитору безопасности путем вызова метода `create` интерфейса `FileSystem_Security`:

```
security src=FileSystem {
  match method=create {
    integrity.create {
      initiator : message.client,
      target    : message.object,
      container : message.directory,
      driver    : src,
      level     : message.label }
  }
}
```

Сообщения, которые `FileSystem` отправляет монитору безопасности, предоставляют следующий контекст:

- идентификатор сущности, которая инициирует запрос на создание объекта (`message.client`);
- идентификатор объекта, который должен быть создан (`message.object`). Этот идентификатор создается сущностью `FileSystem`;
- идентификатор контейнера, в котором объект должен быть создан (`message.directory`). Контейнер определяется сущностью `FileSystem` и не может иметь уровень целостности, больший, чем уровень целостности сущности `FileSystem`;
- идентификатор драйвера файловых ресурсов (`src`);
- метка, определяющая уровень целостности, который должен быть назначен создаваемому файлу (`message.label`).

Если `container` не указан, то данная операция соответствует правилу $create\_root$, и `initiator` должен быть равен `driver`. Только драйверы могут создавать корневые контейнеры для ресурсов, которые они предоставляют. Следовательно, уровень целостности драйвера никогда не может быть ниже уровня целостности его ресурсов.

Правила, соответствующие запросам на чтение и запись к `FileSystem`, определяются следующим образом:

```
request dst=FileSystem {
  match method=read {
    integrity.read {
      reader : src,
      object : message.object }
```



```
    }
  match method=write {
    integrity.write {
      writer : src,
      object : message.object }
  }
}
```

Методы `read` и `write` сущности `FileSystem` соответствуют правилам модели $read(reader, FileSystem, object)$ и $write(writer, FileSystem, object)$ соответственно, то есть происходит автоматическое указание драйвера объекта равным параметру методов `dst`.

5.3 Применение политики

Рассмотрим, как разработанная политика позволяет обеспечивать целостность системы (рис. 11). В соответствии с данной политикой, загруженный образ обновления имеет уровень целостности `LOW`. `Verifier` может считать данный файл и проверить его цифровую подпись. Если верификация успешна, `Verifier` создает новый файл с содержимым старого файла и уровнем целостности `HIGH`. Для применения обновления сущность `Updater` должна получить доступ на чтение к образу обновления посредством метода `read` сущности `FileSystem`. Определение того, будет ли данный доступ предоставлен или нет, происходит в соответствии с правилом $read$. Это означает, что монитор безопасности выполняет следующие проверки:

$$\big(il(Updater) \le il(f)\big) \vee \Big(\neg\big(il(Updater) \le il(f)\big) \wedge \big(ilr(Updater) \le il(f)\big)\Big),$$
$$il(f) \le il(FileSystem),$$

где f – идентификатор образа обновления. Если файл был верифицирован, то $il(f) = HIGH$, и условие $il(Updater) \le il(f)$ истинно. Следовательно, монитор безопасности разрешит предоставление рассматриваемого доступа, и `Updater` сможет применить обновление.

Если файл не был верифицирован, то $il(f) = LOW$, и условие $il(Updater) \le il(f)$ ложно. Вторая часть дизъюнкции также ложна, поскольку хоть и $\neg\big(il(Updater) \le il(f)\big)$ истинно, но $ilr(Updater) \le il(f)$ ложно. Следовательно, монитор безопасности запретит выдачу такого доступа на чтение, и `Updater` не сможет применить обновление. Таким образом, монитор безопасности предотвратит попытки обновления системы с использованием образов обновления, уровень целостности которых не равен `HIGH`.

Предположим, что существует еще одна файловая система с уровнем целостности `MEDIUM`. В этом случае с использованием ее сервисов сущность `Downloader` смогла бы записать образ обновления, сущность `Verifier` – проверить его, но сущность `Updater` не смогла бы получить доступ на чтение к этому файлу. Это связано с тем, что правило по получению доступа на чтение к объекту проверяет, что уровень целостности этого объекта не выше уровня целостности драйвера, посредством которого осуществляется доступ. Другими словами, для доступа к файлу f посредством драйвера $driver$ условие

$$il(Updater) = HIGH \wedge il(Updater) \le il(f) \wedge il(f) \le il(driver)$$

может быть удовлетворено только если $HIGH \le il(driver)$, что не выполняется, если $il(driver) = MEDIUM$. Таким образом, если система содержит драйверы с различными уровнями целостности, то политика мандатного контроля целостности автоматически принимает это во внимание при контроле доступа к ресурсам.

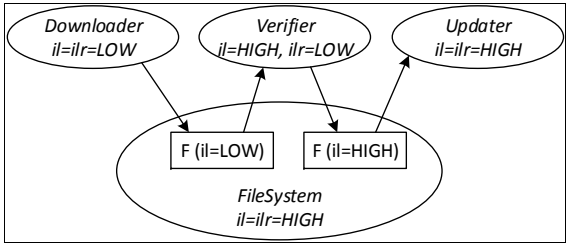

Рис. 11. Схема обработки и применения образа обновления. F – файл-образ обновления
Fig. 11. Update image processing and application scheme. F is the image file

6. Заключение

В работе представлена модель мандатного контроля целостности для микроядерной операционной системы KasperskyOS. В отличие от других моделей, данная модель учитывает наличие драйверов ресурсов при доступах к ресурсам, что позволяет избавиться от предположения доверенности драйверов (присущего ранее существовавшим моделям), которое не позволяло моделировать микроядерные системы с драйверами разного уровня доверия. Тем не менее, даже недоверенные драйверы должны выполнять определенные присущие им функции. В связи с этим сформулированы требования, которым должны удовлетворять драйверы ресурсов и другие сущности, при выполнении которых не возникает угрозы целостности системы (не возникают информационные потоки от менее целостных компонент системы к более целостным либо компонентам с несравнимым уровнем целостности). Проведен анализ модели в случае, когда эти требования не выполняются (например, когда захвачен контроль над некоторыми компонентами системы).

Сформулирована и доказана теорема о допущении моделью либо только разрешенных информационных потоков, либо об ограничении зоны захвата контроля в случае наличия захваченных компонентов. Разработан язык спецификации политик мандатного контроля целостности. Транслятор этого языка создает исполнимый монитор безопасности, выполняющий проверки мандатного контроля целостности при доступе к ресурсам приложений, работающих под управлением операционной системы KasperskyOS. Продемонстрирован пример использования языка для разработки политики безопасности системы с безопасным обновлением. Показано, как мандатный контроль целостности запрещает использование образов обновления с непроверенной цифровой подписью.

Список литературы / References

[1]. Jaeger T. Operating System Security. Morgan and Claypool Publishers, 2008, 220 p.
[2]. Landwehr C. Formal Models for Computer Security. ACM Computing Surveys, vol. 13, issue 3, 1981, pp. 247-278.
[3]. Федеральная служба по техническому и экспортному контролю. Информационное сообщение о требованиях по безопасности информации, устанавливающих уровни доверия к средствам технической защиты информации и средствам обеспечения безопасности информационных технологий от 29 марта 2019 г. / FSTEC Russia. Information message on information security requirements establishing levels of confidence in information security tools and information technology security tools dated March 29, 2019. Available at: https://fstec.ru/normotvorcheskaya/informatsionnye-i-analiticheskie-materialy/1812-informatsionnoe-soobshchenie-fstek-rossii-ot-29-marta-2019-g-n-240-24-1525 (in Russian), accessed 14.01.2020.
[4]. Young M. et al. The duality of memory and communication in the implementation of a multiprocessor operating system. In Proc. of the 11th Symposium on Operating Systems Principles, 1987, pp. 63-76.

[5]. Hohmuth M., Peter M., Härtig H., Shapiro J. Reducing TCB Size by Using Untrusted Components: Small Kernels versus Virtual-Machine Monitors. In Proc. of the 11th ACM SIGOPS European Workshop, 2004..
[6]. Biggs S., Lee D., Heiser G. The Jury Is. In Proc. of the 9th Asia-Pacific Workshop on Systems. 2018, Article No. 16.
[7]. Herder J. N., Bos H., Gras B., Homburg P., Tanenbaum A. S. Construction of a Highly Dependable Operating System. In Proc. of the Sixth European Dependable Computing Conference, 2006, pp. 3-12.
[8]. Девянин П. Н. Модели безопасности компьютерных систем. Управление доступом и информационными потоками. Горячая линия-Телеком, 2013, 338 с. / Devyanin P.N. Security models of computer systems. Control for access and information flows. Hotline-Telecom, 2013, 338 p. (in Russian).
[9]. Biba K. Integrity Considerations for Secure Computer Systems. Technical report MTR-3153, The MITRE Corporation, 1977.
[10]. Lipner S. Non-Discretionary Controls for Commercial Applications. In Proc. of the 1982 IEEE Symposium on Security and Privacy, 1982. pp. 2-10.
[11]. Clark D., Wilson D. A Comparison of Commercial and Military Computer Security Policies. In Proceedings of the 1987 IEEE Symposium on Security and Privacy, 1987, pp. 184-195.
[12]. Lee, T. Using Mandatory Integrity to Enforce “Commercial” Security. In Proc. of the 1988 IEEE Symposium on Security and Privacy, 1988, pp. 140-146.
[13]. Brewer D., Nash M. The Chinese Wall Security Policy. In Proc. of the 1989 IEEE Symposium on Security and Privacy, 1989, pp. 206-214.
[14]. Liu Z., Wang T., Li W. An Integrity Control Model for Operating System. In Proc. of the 2009 International Conference on Management and Service Science, 2009, pp. 1-4.
[15]. Bell D., LaPadula L. Secure Computer System: Unified Exposition and Multics Interpretation. Technical Report MTR-2997 Rev. 1, The MITRE Corporation, 1976.
[16]. Li N., Mao Z., Chen H. Usable Mandatory Integrity Protection for Operating Systems. In Proc. of the 2007 IEEE Symposium on Security and Privacy, 2007, pp. 164-178.
[17]. Zhai E. et al. SecGuard: Secure and Practical Integrity Protection Model for Operating Systems. In Proc.of the 13th Asia-Pacific Web Conference, 2011, pp. 370-375.
[18]. Devyanin P. et al. Using Refinement in Formal Development of OS Security Model. Lecture Notes in Computer Science, vol. 9609. 2015, pp. 107-115.
[19]. Devyanin P. et al. Formal Verification of OS Security Model with Alloy and Event-B. Lecture Notes in Computer Science, vol. 8477, 2014, pp. 309-313.
[20]. П.Н. Девянин и др. Моделирование и верификация политик безопасности управления доступом в операционных системах. Горячая линия–Телеком, 2019б 214 стр. / P.N. Devyanin, D.V. Efremov, V.V. Kulyamin, A.K. Petrenko, A.V. Khoroshilov, I.V. Shchepetkov. Modeling and verification of access control security policies in operating systems. Hotline – Telecom, 2019, 214 p.
[21]. Yosifovich P., Russinovich M., Solomon D., Ionescu A. Windows Internals, Part 1: System architecture, processes, threads, memory management, and more (7th Edition). Microsoft Press. 2017. 800 p.
[22]. Rushby J. Design and Verification of Secure Systems. In Proc. of the 8th ACM Symposium on Operating Systems Principles, 1981, pp. 12-21.
[23]. Alves-Foss J., Oman P., Taylor C. The MILS Architecture for High-Assurance Embedded Systems. International Journal of Embedded Systems, vol. 2, no. 3/4, 2006, pp. 239-247.
[24]. Spencer R., Smalley S., Loscocco P., Hibler M., Andersen D., Lepreau J. The Flask Security Architecture: System Support for Diverse Security Policies. In Proc. of the 8th USENIX Security Symposium, 1999.
[25]. Baier C., Katoen J.-P. Principles of Model Checking. The MIT Press. 2008. 975 pp.
[26]. Bishop M., Snyder L. The Transfer of Information and Authority in a Protection System. In Proc. of the 7th ACM symposium on Operating systems principles, 1979, pp. 45-54.
[27]. Bishop M. Conspiracy and Information Flow in the Take-Grant Protection Model. Journal of Computer Security, vol. 4, no. 4, 1996, pp. 331-359.

Информация об авторах / Information about authors

Владимир Сергеевич БУРЕНКОВ – кандидат технических наук, разработчик-исследователь. Сфера научных интересов: модели программно-аппаратных систем, формальные методы верификации.

Vladimir Sergeevich BURENKOV – PhD, research developer. Research interests: models of computer systems, formal verification methods.

Дмитрий Александрович КУЛАГИН – кандидат технических наук, ведущий разработчик. Сфера научных интересов: информационная безопасность, компиляторы, операционные системы.

Dmitry Aleksandrovich KULAGIN – PhD, development team lead. Research interests: information security, compilers, operating systems.