МНОГОУРОВНЕВАЯ МОДЕЛЬ ПОЛИТИКИ БЕЗОПАСНОСТИ УПРАВЛЕНИЯ ДОСТУПОМ ОПЕРАЦИОННЫХ СИСТЕМ СЕМЕЙСТВА WINDOWS

Козачок В.И.[1], Козачок А.В.[2], Кочетков Е.В.[3]

***Цель исследования**: разработка более совершенного механизма разграничения доступа для защиты от скрытых каналов утечки информации по памяти в семействе операционных систем Windows.*

***Метод исследования:** анализ моделей мандатного управления доступом и контроля целостности в операционных системах семейства Windows, моделирование политики безопасности управления доступом для заданных свойств безопасности, автоматическая верификация моделей. Для описания модели и спецификации к ней использован язык темпоральной логики действий Лэмпорта (TLA+), позволяющий произвести автоматическую верификацию модели относительно заданных свойств безопасности.*

***Результаты исследования:** выявлены основные ограничения существующего обязательного контроля целостности операционных систем семейства Windows. Разработан набор структур многоуровневой модели, отражающий значимые для моделирования процесса доступа субъектов к объектам атрибуты. Промоделированы ключевые механизмы разграничения доступа в операционной системе: управление пользователями, группами, субъектами, объектами, ролями, правами, дискреционное и мандатное управление доступом, мандатный контроль целостности – многоуровневое управление доступом субъектов к объектам. В модели определен механизм контроля над созданием субъектов на основе исполняемых файлов для организации изолированной программной среды. Определены значения атрибутов переменных модели на этапе инициализации. Разработаны инварианты корректности переменных в процессе верификации и безопасного доступа субъектов к объектам. Модель задана с использованием языка моделирования TLA+ и верифицирована.*

***Ключевые слова:** информационная безопасность, несанкционированный доступ, модели безопасности, верификация.*

DOI: 10.21681/2311-3456-2021-1-41-56

Введение

Проблема отсутствия обязательного контроля целостности в операционных системах семейства Windows обострилась в связи с появлением так называемой «подрывной атаки» (Shatter attack) [1], суть которой заключается в выполнении произвольного кода приложением с высокими привилегиями после получения системного сообщения от приложения с низкими привилегиями. В связи с этим обязательный контроль обеспечения целостности был введен, начиная с версии Window Vista в 2007 году, и с тех пор существенно не изменился.

Работа механизма MIC заключается в том, что всем процессам, файлам, каталогам, а также другим именованным сущностям назначается определенный уровень целостности. При этом доступ субъектов к объектам осуществляется на основе модели безопасности Биба [2]. Целью данного механизма является использование политик управления целостностью и уровней целостности задействованных субъектов и объектов для ограничения доступа процессам, которые считаются потенциально менее надежными, по сравнению с доверенными процессами, работающими под той же учетной записью пользователя. С помощью задания различных уровней целостности MIC позволяет изолировать потенциально уязвимые приложения (например, приложения, ориентированные на работу в сети Интернет, офисные приложения), которые используются для открытия документов, полученных из недоверенных источников. Процессы с низким уровнем целостности имеют меньший доступ

1 Козачок Василий Иванович, доктор социологических наук, сотрудник Академия ФСО России, г. Орёл, Россия. E-mail: v.kozachok@academ.msk.rsnet.ru

2 Козачок Александр Васильевич, доктор технических наук, сотрудник Академия ФСО России, г. Орёл, Россия. E-mail: a.kozachok@academ.msk.rsnet.ru

3 Кочетков Евгений Викторович, сотрудник Академии ФСО России, г. Орёл, Россия. E-mail: e.kochetkov@academ.msk.rsnet.ru

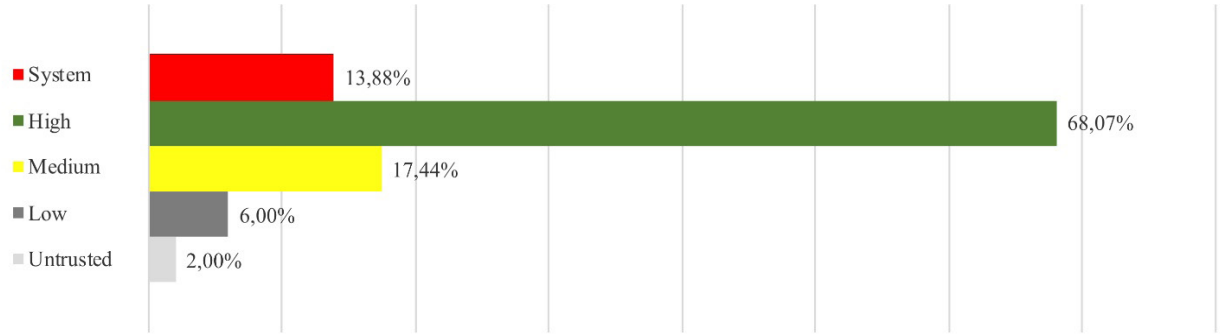

Рис. 1. Распределение вредоносных программ по уровням обеспечения целостности Windows в момент их обнаружения

(ограничены права на запись в системные области, чем процессы с более высокими уровнями целостности) [3].

При этом можно выделить следующие основные ограничения MIC [4]:

- по умолчанию большинство файлов и каталогов имеют средний уровень целостности (Middle integrity); таким образом, граница доверия между объектами становится размытой;
- процессу назначается низкий уровень целостности (Low integrity) только по инициативе самого процесса, чаще всего при работе в сети Интернет, однако большинство браузеров и других приложений этого не делает;
- при задании уровня целостности файлов не учитывается источник их получения, например, съемные носители, сетевые диски, а также файлы, полученные из сети, обладают явно более низким уровнем целостности, чем файлы, созданные самой ОС.

Несмотря на то, что дальнейшее распространение подрывных атак было прекращено введением обязательного контроля целостности и вынесением системного процесса в отдельную пользовательскую сессию, проблема недостаточного разделения процессов между уровнями целостности остается актуальной в настоящее время ввиду того, что нарушение целостности влечет за собой ослабление механизмов обеспечения конфиденциальности.

При этом результаты анализа уровней целостности, с которыми выполняются вредоносные программы в момент их детектирования поведенческими сигнатурами[4], показали, что большая часть вредоносных программ выполняются с высоким уровнем целостности. Распределение вредоносных программ, обнаруженных средствами антивирусной защиты, по уровням целостности Windows представлено на рисунке 1.

Анализ представленных на рисунке 1 сведений показывает, что при использовании штатного набора средств защиты ОС семейства Windows большинство вредоносных программ выполняется с высоким уровнем целостности.

Для усиления комплекса мер по контролю целостности в ОС семейства Widows предлагается разработка модели, компенсирующей ряд обозначенных выше недостатков MIC и расширяющей ее функциональные возможности за счет учета категорий. Важно отметить, что практическое решение данной задачи требует математического описания протекающих в операционной системе процессов доступа к информации, что является трудоемким и не всегда возможным в связи с большим разнообразием видов доступа, типов объектов, иерархии пользователей и другими особенностями. При этом проведение экспериментов по незначительному изменению атрибутов объектов или способов проверки доступа требует проведение верификации модели снова. В этом случае одним из решений является автоматическая формальная верификация модели политики безопасности управления доступом с использованием метода Model checking, который является математически обоснованным способом доказательства того, что модель соответствует заданной для нее спецификации [5, 6] с использованием инструментальных средств [7, 8].

Актуальность проблемы разработки формальной модели политики безопасности управления доступом также связана с процессом внедрения ФСТЭК России «Требований безопасности информации к операционным системам», в частности выполнение требований функциональной компоненты ADV_SPM.1 «Формальная модель политики безопасности» [9].

Для семейства операционных систем Linux решение подобной задачи рассмотрено в работах [10-12].

4 Русаков В. Вредоносный код и механизм целостности Windows [электронный ресурс] – Режим доступа: https://securelist.ru/malicious-code-and-the-windows-integrity-mechanism/29740 (дата обращения 22.05.2019).

Подход к верификации моделей методом Model checking с использованием TLA+

Под верификацией модели методом Model checking понимается автоматическое доказательство соответствия поведения системы заданной для нее спецификации. Спецификация отражает условия, при выполнении которых для каждого состояния системы подтверждено наличие у модели требуемых свойств жизнеспособности (liveness) и безопасности (safety) [13, с. 120].

Свойство жизнеспособности модели заключается в адекватности поведения системы объекту реального мира. Для достижения этого свойства в модели должны быть заданы, а при проведении верификации достигнуты, такие состояния, в которых реализуется значимые для моделируемой системы функции. Примером достижения свойств жизнеспособности при моделировании работы лифта является наличие перехода в состояние перемещения между этажами и подтверждение при завершении верификации, что лифт перемещался.

Свойство безопасности модели отражает невозможность перехода системы во множество запрещенных состояний. При моделировании работы лифта одним из запрещенных состояний является перемещение между этажами с открытыми дверями. Примером запрещенного состояния при моделировании разграничения доступа к информации является факт несанкционированного доступа к ней [14-18].

В общем виде организация процесса верификации методом Model checking представляет собой последовательное выполнение следующих этапов [13, с. 11]:

1. Этап моделирования.
2. Этап автоматизированной верификации.
3. Этап анализа результатов.

Этап моделирования заключается в определении переменных модели, правил перехода между состояниями, начального состояния системы. Для определения требуемых свойств модели задается спецификация. Под состоянием системы понимается уникальным набор значений каждой из заданных переменных. При этом повторение набора значений переменных в разные моменты времени является переходом в одно и тоже состояние.

Следующим этапом является автоматизированная верификация, которая заключается в запуске специальной программы верификатора с учетом ограничений вычислительных ресурсов.

На этапе анализа результатов проверяется подтверждены ли заданные для модели свойства, о чем свидетельствует успешное завершение процесса верификации. В случае нарушения заданных свойств производится рассмотрение контрпримера, который представляет собой такой набор значений переменных модели, который не удовлетворяет заданной для модели спецификации. В этом случае производится уточнение модели или спецификации и повторение процесса верификации. Возможно исчерпание вычислительных ресурсов при проведении верификации модели, в этом случае производится уменьшение пространства состояний модели и возврат к этапу 2.

Существуют различные способы задания модели. В рамках настоящего исследования для определения модели применяется нотация языка TLA+ [7], близкая к математической, она обладает достаточной гибкостью и выразительными возможностями описания, а наличие верификатора TLC позволяет произвести ее автоматизированную верификацию.

Определение переменных системы производится с использованием выражения:

$$\text{VARIABLES A, B,}$$

где A, B – переменные, значение которых определяют состояние модели.

Переход системы в следующее состояние происходит в результате изменения значения одной или нескольких переменных. При этом в явном виде должны быть заданы новые значения всех переменных или указано, что переменные, для которых значения остались прежние, не изменились. Изменение значения переменных происходит при выполнении операторов, примером такого оператора является следующее выражение:

$$\begin{aligned} &swapA(a,b) \triangleq \\ &\quad \wedge\ 'A = (A \setminus a) \cup \{b\}\ , \\ &\quad \wedge\ UNCHANGED\langle B \rangle \end{aligned}$$

где $UNCHANGED$ – ключевой оператор языка TLA+, означающий, что переменные, перечисленные в кортеже, не изменились.

Результатом выполнения оператора $swapA$ является переход в новое состояние системы, при котором элемент a исключается из множества A, а элемент b включается. При этом значение переменной B не изменяется.

Задание начального состояния системы производится с использованием специального оператора $Init$. Пример определения начального состояния системы представлен следующим выражением:

$$Init \triangleq A = \{\} \wedge B = \{\}$$

Верификация динамической системы невозможна без учета временной зависимости между перехо-

DOI: 10.21681/2311-3456-2021-1-41-56

дами из одного состояния в другое. Для учета такой зависимости логика предикатов первого порядка расширяется темпоральными операторами логики Лемпорта, основными из которых являются:

□ – оператор «всегда в будущем»;

◊ – оператор «однажды в будущем»;

U – бинарный оператор «до тех пор пока»;

S – бинарный оператор «с тех пор как».

В общем виде спецификация модели языке TLA+ имеет следующий вид:

$$Spec @ Init \wedge \mathrm{W}[Next]_{vars},$$
$$Spec \triangleq Init \wedge \mathrm{W}[Next]_{vars},$$

где $Next$ – предикаты действий, изменяющие значения переменных модели, $vars$ – кортеж переменных модели.

При моделировании объектов реального мира использование переменных, хранящих единственное значение из заданного домена значений недостаточно, ввиду того такие объекты характеризуются множеством атрибутов. Для хранения множества атрибутов в одной переменной и доступа к ним в языке TLA+ используются структуры. Структура TLA+ представляет собой тип данных, включающий в себя набор полей, обращение к которым происходит по имени. Каждая структура модели задается набором пар имени поля и доменом значений. Доменом значений может быть структура.

В языке TLA+ определение типа структуры равнозначно формированию множества элементов данной структуры, включающего все возможные сочетания значений отдельных полей структуры. Пример определения структуры представлен следующим выражением:

$$\begin{aligned} &Sctructure \triangleq [\\ &\quad id: \quad \{1,2\}, \\ &\quad is\_main: \quad BOOLEAN, \\ &] \end{aligned}$$

где $BOOLEAN$ – встроенный тип данных, принимающий значения из множества $\{TRUE, FALSE\}$. Определение структуры эквивалентно определению следующего множества элементов:

$$\begin{aligned} &Structure \triangleq \{ \\ &\quad [id: \ 1, \quad is\_main: \ TRUE], \\ &\quad [id: \ 1, \quad is\_main: \ FALSE], \\ &\quad [id: \ 2, \quad is\_main: \ TRUE], \\ &\quad [id: \ 2, \quad is\_main: \ FALSE] \\ &\} \end{aligned}$$

Преобразование определения структуры во множество элементов $Structure$ дает возможность проверить является ли некоторое множество A подмножеством $Structure$:

$$A \subseteq Structure$$

Данная операция определяет соответствует ли каждый элемент множества A структуре $Structure$.

Обращение к элементам структуры производится с использованием разделителя «точка».

При определении операторов модели могут быть использованы следующие функции TLA+:

– $Len(x)$ возвращает длину кортежа x;

– $Cardinality(m)$ возвращает мощность множества m;

– $Range(y)$ возвращает множество элементов, полученное из кортежа y.

Определение структур субъектов и объектов модели

При определении структур субъектов и объектов с использованием TLA+ отсутствует строгая типизация (по умолчанию производится проверка только встроенных типов), однако, проверка инвариантов типов является неотъемлемой частью спецификации, поскольку верификация производится методом Model Checking.

Проведение верификации на множестве элементов близком к реальному приведет к существенному увеличению количества возможных состояний, что существенно замедлит процесс исследования. Ввиду этого, ряд значений, заданных в модели, являются модельными для снижения ресурсоемкости процесса верификации, но не влияют на общность и адекватность модели в целом [19].

Множества идентификаторов процессов, контейнеров, файлов и пользователей (значения модельные):

$$\begin{aligned} UserIDs &\triangleq 0..2 \\ GroupIDs &\triangleq 0..2, \\ SubjectIDs &\triangleq 0..1 \\ ObjectIDs &\triangleq 0..6 \end{aligned}$$

где $UserIDs$ – множество идентификаторов пользователей; $GroupIDs$ – множество идентификаторов групп; $SubjectIDs$ – множество идентификаторов субъектов; $ObjectIDs$ – множество идентификаторов объектов.

Для разграничения доступа с учетом многоуровневой модели доступа были определены следующие множества уровней и категорий конфиденциальности и целостности:

$$ConfLevs \triangleq 0..2 \qquad IntLevs \triangleq 0..2$$
$$ConfCats \triangleq \{ "C1", "C2" \}; \qquad IntCats \triangleq \{ "I1", "I2" \},$$

где $ConfLevs$ – множество уровней конфиденциальности; $IntLevs$ – множество уровней целостности; $ConfCats$ – множество категорий конфиденциальности; $IntCats$ – множество категорий целостности.

Множество разрешений на доступ к объекту заданы следующим выражением:

$$Permissions \triangleq \{$$
"read", "write", "append",
"create", "delete", "delete_subfolder",
"change_perm", "change_owner", "execute",
"read_attr", "write_attr"
$$\},$$

где "read" – чтение содержимого объекта; "write" – изменение содержимого объекта; "append" – запись в конец содержимого объекта без его чтения; "create" – создание объекта; "delete" – удаление объекта; "delete_subfolder" – удаление дочерних каталогов; "change_perm" – изменение разрешений на доступ к объекту;"change_owner" – изменение владельца объекта; "execute" – исполнение объекта; "read_attr" – чтение атрибутов объекта; "write_attr" – изменение атрибутов объекта.

Для сравнения уровней конфиденциальности и целостности в модели приняты следующие соответствия уровней абсолютным значениям (таблица 1).

Состояние модели определяется набором значений каждого элемента из следующих переменных:

– множество доступов A (Actions);
– множество объектов доступа O (Objects);
– множество субъектов доступа S (Subjects);
– множество пользователей U (Users);
– множество групп пользователей G (Groups).

Множество доступов A содержит информацию об успешном совершении некоторого доступа субъектом из множества S к объекту из множества O в виде кортежей. Каждый кортеж включает в себя идентификатор субъекта, идентификатор объекта и тип доступа в виде строки. Эти сведения требуются для проверки выполнения инвариантов безопасности.

Множество объектов доступа O содержит множество элементов, каждый из которых соответствует следующему определению:

$$Objects \triangleq [$$
oid : ObjectIDs, cl : ConfLevs, ext_attr : SUBSET ExtAttr,
owner : UserIDs, conf_cats : ConfCats, exec_attr : ExecTypes,
poid : ObjPoids, il : IntLevs, aclgs : SUBSET ACLg, ,
type : ObjTypes, int_cats : IntCats, aclus : SUBSET ACLu,
$$]$$

где $type$ – тип объекта; ext_attr – расширенные атрибуты объекта; $exec\_attrs$ – атрибуты исполнения (только для исполняемых файлов) ; $aclgs$ – множество разрешений на доступ к объекту для групп; $aclus$ – множество разрешений на доступ для пользователей.

Атрибут тип объекта $type$ может принимать значения из множества, определенного следующим выражением:

$$ObjTypes \triangleq \{"root\_container", "container", "file", "executable"\},$$

где $root\_container$ – корневой каталог иерархии файловой системы (корень диска); $container$ – каталог для хранения файлов; $file$ – файл для хранения информации произвольного содержания; $executable$ – исполняемый файл.

Расширенные атрибуты объекта ext_attr является подмножеством из множества, определенного следующим выражением:

$$ExtAttr \triangleq \{"ccnr","check\_child\_perm","icnr"\},$$

где "ccnr" – флаг отсутствия необходимости проверки прав доступа по конфиденциальности при обращении к этому объекту (confidential control not required); "check_child_perm" – требуется проверка вложенных объектов по дискреционным правилам разграничения доступа; "icnr" – флаг отсутствия необходимости проверки прав доступа по целостности при обращении к этому объекту (integrity control not required).

Атрибут исполнения $exec\_attr$ применяется только для исполняемых файлов и может принимать значение из множества, определенного следующим выражением:

$$ExtAttr \triangleq \{"deny","app","install"\},$$

где "deny" – запуск запрещен; "app" – приложение; "install" – установщик программного обеспечения.

Множество субъектов доступа S определяется следующей структурой:

$$Subject \triangleq [$$
sid : SubjectIDs, uid : UserIDs,
cl : ConfLevs, il : IntLevs,
conf_cats : SUBSET ConfCats, int_cats : SUBSET IntCats,
$$]$$

Множество пользователей U определяется следующей структурой:

$$
\begin{array}{l}
Users \triangleq [\\
\quad uid: \ UserIDs, \quad is\_admin: \ BOOLEAN, \\
\quad cl: \ ConfLevs, \quad il: \ IntLevs, \\
\quad conf\_cats: \ SUBSET\ ConfCats, \quad int\_cats: \ SUBSET\ IntCats, \\
\quad groups: \ SUBSET\ GroupIDs, \quad white\_list\_execute\_objects: \ SUBSET\ ObjectIDs \\
]
\end{array}
$$

где $conf\_cats$, int_cats – доступные для пользователя категории конфиденциальности и целостности; $groups$ – множество идентификаторов группы пользователей, в которые входит данный пользователь; $white\_list\_execute\_objects$ – множество идентификаторов исполняемых объектов, разрешенных для запуска новых субъектов данным пользователем.

Множество групп пользователей G определяется следующей структурой:

$$
\begin{array}{l}
Groups \triangleq [\\
\quad gid: \ GroupIDs, \quad role: \ Roles, \\
]
\end{array}
$$

где $role$ – роль группы.

Таблица 1

Соответствие уровней конфиденциальности и целостности абсолютным значениям

Абсолютное значение	Уровень конфиденциальности	Уровень целостности
2	Высокий	Низкий
1	Средний	Средний
0	Низкий	Высокий

Реализация в модели дискреционного и многоуровневого разграничения доступа

При попытке доступа субъекта к объекту производится проверка возможности совершения такого доступа на основе дискреционного и многоуровневого контроля доступа. Доступ разрешается, только если проверка по обоим механизмам контроля доступа принимает истинное значение.

Дискреционное разграничение доступа основано на «запрещающей» политике. Все совершаемые доступы от субъектов к объектам должны быть разрешены явно. Разрешение на доступ к объекту хранит сам объект, и оно может быть задано для конкретного пользователя или группы.

Для проверки дискреционного доступа были заданы операторы с префиксом DAC_ (Discretionary access control). Возможность совершения доступа a субъектом s по отношению к объекту o проверяется предикатом DAC_may_do (выражение 1):

$$
\begin{array}{l}
DAC\_may\_do(s,o,a) \triangleq \\
\quad \vee o.owner = s.uid \\
\quad \vee IsUserAdmin(s) \\
\quad \vee \exists r \in o.aclus: \\
\qquad \wedge r[1] = s.uid \\
\qquad \wedge a \in r[2] \\
\quad \vee \exists g \in SelectUserGroups(s.uid): \\
\qquad \exists r \in o.aclgs: \\
\qquad\quad \wedge r[1] = SelectGroup(g).gid \\
\qquad\quad \wedge a \in r[2]
\end{array}
\qquad (1)
$$

Выражение 1 принимает истинное значение в следующих случаях:

– пользователь, создавший субъект s, является администратором; субъект s является владельцем объекта o;

– существует такое правило $r \in o.aclus$, в котором на первой позиции задан идентификатор пользователя, создавшего субъект, и доступ a входит во множество разрешений данного правила;

– существует такая группа $g \in SelectUserGroups(s.uid)$, в которую входит пользователь, и такое правило $r \in o.aclgs$, в котором на первой позиции задан идентификатор группы, и доступ a входит во множество разрешений данного правила.

Для проверки доступа к объекту o субъектом s, если для контейнера объекта задан расширенный атрибут "$check\_child\_perm$", то требуется проверка права доступа a для всех элементов из иерархии до этого объекта предикатом $DAC\_may\_access\_cont\_hierachy$ (Выражение 2).

$$
\begin{array}{l}
DAC\_may\_access\_cont\_hierachy(s,op,a) \triangleq \\
\quad \forall oi \in Range(op): \\
\qquad DAC\_may\_do(s, SelectObject(oi), a)
\end{array}
\qquad (2)
$$

Предикат в выражении 2 примет истинное значение только тогда, когда для всех элементов из иерархии объектов op разрешен доступ для субъекта s для разрешения a.

Многоуровневое управление доступом в модели строится на выполнении следующих принципов:

1. Субъекту с низким уровнем конфиденциальности запрещен доступ по чтению к объекту с высоким уровнем конфиденциальности «no read up».
2. Субъекту с высоким уровнем конфиденциальности запрещен доступ по изменению к объекту с низким уровнем конфиденциальности «no write down».
3. Субъекту с низким уровнем целостности запрещен доступ на изменение объектов с высоким уровнем целостности. В основе правила лежит принцип «no write up».

Принцип 1 и 2 сформулированы в модели Белла-Ла-Падулы [20], принцип 3 сформулирован в модели мандатной политики целостности информации Биба [1].

В настоящей модели приняты следующие свойства безопасности:
1. Уровень конфиденциальности вложенных объектов не может быть выше уровня конфиденциальности родительского объекта.
2. Уровень целостности вложенных объектов не может быть выше уровня целостности родительского объекта.
3. Проверка доступа к объектам по конфиденциальности и целостности не производится при наличии флагов "$ccnr$" и "$icnr$" соответственно.
4. Флаги "$ccnr$" и "$icnr$" могут быть сняты только при выполнении правил 1 и 2 соответственно.
5. Уровень конфиденциальности субъекта не может быть выше уровня конфиденциальности пользователя, создавшего его.
6. Уровень целостности субъекта не может быть выше уровня целостности пользователя, создавшего его.
7. Множество категорий всех дочерних объектов должно быть подмножеством множества категорий самого объекта.
8. Субъект может назначить категории объектам только из множества доступных ему категорий.

Для проверки многоуровневого контроля доступа были заданы следующие операторы с префиксом MAC_ (mandatory access control). Базовыми предикатами проверки по конфиденциальности и целостности являются $MAC\_may\_access\_conf$ и $MAC\_may\_access\_int$ соответственно.

$$MAC\_may\_access\_conf(s,o) \triangleq$$
$$\vee\ "ccnr" \in o.ext\_attr$$
$$\vee\ \wedge\ o.cl \le s.cl$$
$$\wedge\ o.conf\_cats \subseteq s.conf\_cats$$

$$MAC\_may\_access\_int(s,o) \triangleq$$
$$\vee\ "icnr" \in o.ext\_attr$$
$$\vee\ \wedge\ o.il \ge s.il$$
$$\wedge\ o.int\_cats \subseteq s.int\_cats$$

Предикат $MAC\_may\_access\_conf$ принимает истинное значение, когда расширенные атрибуты объекта содержат флаг "$ccnr$" или уровень конфиденциальности объекта меньше или равен уровню конфиденциальности субъекта, и множество категорий конфиденциальности объекта является подмножеством множества категорий, доступных субъекту. Характеристика предиката $MAC\_may\_access\_int$ аналогичная.

Для операций чтения и записи субъектами объектов определены предикаты MAC_may_read и MAC_may_write (выражение 3).

$$MAC\_may\_read(s,o) \triangleq$$
$$MAC\_may\_access\_conf(s,o) \qquad (3)$$

$$MAC\_may\_write(s,o) \triangleq$$
$$\wedge\ MAC\_may\_access\_conf(s,o)$$
$$\wedge\ MAC\_may\_access\_int(s,o)$$

При этом важно отметить, что проверка уровня целостности происходит только для операций записи, при чтении ограничений по уровню целостности нет. Это связано с тем, что системным процессам, работающим на высоком уровне целостности, для корректной работы всей системы необходимо получать информацию от объектов низкого уровня целостности.

Дополнительными операциями, требующими многоуровневого контроля доступа, помимо чтения и записи объектов, являются операции: изменения уровня конфиденциальности и целостности (выражение 4), удаление флагов "$ccnr$" и "$icnr$" (выражение 5).

Предикаты проверки возможности изменения уровня конфиденциальности и целостности объектов представлены в выражении 4.

$$MAC\_may\_change\_cl(o,cl) \triangleq$$
$$\wedge\ \vee\ cl \le ParentCont(o).cl$$
$$\vee\ "ccnr" \in ParentCont(o).ext\_attr$$
$$\wedge\ \vee\ \wedge\ o.type \in Containers$$
$$\wedge\ \vee\ \forall ch \in SelectAllChilds(o):$$
$$ch.cl \le cl$$
$$\vee\ "ccnr" \in o.ext\_attr$$
$$\vee\ o.type \notin Containers \qquad (4)$$

$$MAC\_may\_change\_il(o,il) \triangleq$$
$$\wedge\ \vee\ il \ge ParentCont(o).il$$
$$\vee\ "icnr" \in ParentCont(o).ext\_attr$$
$$\wedge\ \vee\ \wedge\ o.type \in Containers$$
$$\wedge\ \vee\ \forall ch \in SelectAllChilds(o):$$
$$ch.il \ge il$$
$$\vee\ "icnr" \in o.ext\_attr$$
$$\vee\ o.type \notin Containers$$

Уровень конфиденциальности объекта может быть изменен, если он не превышает уровень конфиден-

DOI: 10.21681/2311-3456-2021-1-41-56

циальности родительского контейнера, или для него установлен флаг "*ccnr*". Если объект сам является контейнером, то для всех его вложенных объектов их уровень конфиденциальности не должен превышать назначаемый уровень конфиденциальности, или объект должен иметь расширенный атрибут "*ccnr*", иначе назначение не возможно.

$$
\begin{aligned}
&MAC\_may\_remove\_ccnr(o) \triangleq \\
&\vee \ \wedge \ o.type \in Containers \\
&\quad \wedge \ \forall ch \in SelectAllChilds(o): \\
&\quad\quad \wedge \ ch.cl \le o.cl \\
&\quad\quad \wedge \ ch.conf\_cats \subseteq o.conf\_cats \\
&\vee \neg o.type \in Containers
\end{aligned}
\tag{5}
$$

$$
\begin{aligned}
&MAC\_may\_remove\_icnr(o) \triangleq \\
&\vee \ \wedge \ o.type \in Containers \\
&\quad \wedge \ \forall ch \in SelectAllChilds(o): \\
&\quad\quad \wedge \ ch.cl \ge o.cl \\
&\quad\quad \wedge \ ch.int\_cats \subseteq o.int\_cats \\
&\vee \neg o.type \in Containers
\end{aligned}
$$

Флаг отсутствия проверки уровней конфиденциальности при проверке доступа к объектам "*ccnr*" может быть снят, для объектов типом контейнер при этом уровень конфиденциальности всех дочерних объектов должен быть меньше или равен уровню конфиденциальности объекта, и множество категорий конфиденциальности дочерних объектов должно быть подмножеством множества категорий конфиденциальности самого объекта. Условия снятия флага "*icnr*" аналогичны. Важно отметить, что установка этих флагов, с учетом выполнения дискреционных правил разграничения доступа, дополнительных проверок не требует.

Реализация в модели изолированной программной среды

Одним из способов снижения вероятности совершения несанкционированного доступа субъектов к объектам с использованием вредоносных программ является организация изолированной программной среды, которая заключается в ограничении создания только таких субъектов, функционирование которых разрешено явно, а их целостность подтверждена. В рамках настоящей модели изолированная программной среда реализуется на основе следующих правил:

1. Создание исполняемых файлов может осуществлять только администратор системы.
2. Модификация исполняемых файлов запрещена.
3. Добавление идентификаторов исполняемых объектов в список разрешенных осуществляет только администратор.
4. Для создания субъекта пользователем необходимо, чтобы идентификатор исполняемого файла входил во множество разрешенных для запуска объектов этого пользователя.

Ограничение на создание исполняемых объектов заложено при вычислении предусловий оператора *CreateObject*, и представлено следующим выражением:

$$
\begin{aligned}
&\ldots \\
&\wedge \ \vee \ IsUserAdmin(s) \\
&\quad \vee \ \wedge \ \sim IsUserAdmin(s) \\
&\quad\quad \wedge \ type \# \text{"}executable\text{"} \\
&\ldots
\end{aligned}
$$

Согласно данному выражению, исполняемый объект может быть создан только администратором системы. Если субъект s создается не от имени администратора системы, то тип создаваемого объекта не может быть "*executable*" (исполняемый объект).

Для ограничения записи в исполняемые объекты в предусловиях оператора *WriteD* добавлено условие записи только в объекты с типом "*file*".

Добавление идентификатора объекта o в список разрешенных для создания на основе его субъекта пользователем u возможно только от имени Администратора системы и представлено следующим выражением:

$$
\begin{aligned}
&AddAppToWhiteListD \triangleq \\
&\exists s \in S: \\
&\exists u \in U: \\
&\exists o \in SelectExecuteObjects: \\
&\quad \wedge \ IsUserAdmin(s) \\
&\quad \wedge \ AddAppToWhiteList(s,u,o)
\end{aligned}
$$

где, *SelectExecuteObjects* – множество исполняемых объектов системы.

После выполнения предусловия оператора *AddAppToWhiteListD* происходит непосредственное добавление идентификатора объекта во множество идентификаторов исполняемых объектов данного пользователя, данная процедура представлена следующим выражением:

$$
\begin{aligned}
&AddAppToWhiteList(s,u,oid) \triangleq \\
&\quad \wedge \ U' = (U \setminus \{u\}) \cup \{[u \ EXCEPT \ !["white\_list\_execute\_objects"] = \\
&\qquad u.white\_list\_execute\_objects \cup \{oid\}]\} \\
&\quad \wedge \ A' = A \cup \{\langle s.sid, u.uid, \text{"}add\_app\_to\_white\_list\text{"}\rangle\} \\
&\quad \wedge \ UNCHANGED \ \langle S, O, G\rangle
\end{aligned}
$$

Контроль над созданием субъектов только из разрешенного множества объектов для выполнения 4 правила производится в операторе создания субъекта $CreateSubjectD$:

$$\begin{aligned}&\ldots\\&\wedge\ \vee\ IsUserAdmin(s)\\&\quad\vee\ \exists oid \in SelectUser(s.uid).white\_list\_execute\_objects :\\&\qquad uid.id = o.id\\&\ldots\end{aligned}$$

Предусловием создания субъекта на основе исполняемого объекта является то, что он создается от имени администратора или существует такой идентификатор oid во множестве $white\_list\_execute\_objects$ данного пользователя, который равен идентификатору исполняемого объекта o.

Инициализация начальных значений модели

Начальное состояние системы определено из минимально необходимого набора значений на основе экспертного подхода. Множество текущих действий чтения и записи A на этапе инициализации является пустым. Для ускорения процесса верификации вводятся начальные взаимосвязанные между собой элементы множеств переменных. Для пользователей, субъектов и объектов в табличном виде для более компактного представления (таблица 2, 3, 4), а для групп с использованием TLA+ (выражение 6):

Начальные значения элементов множества групп определены следующими выражениями:

$$\begin{aligned}&g_0 \triangleq [&& g_1 \triangleq [\\&\quad gid \ \rightarrow\ 0, && \quad gid \ \rightarrow\ 1,\\&\quad role \ \rightarrow\ FullRole && \quad role \ \rightarrow\ ReadRole\\&] \ , &&] \ .\end{aligned} \qquad (6)$$

Изначально в системе существуют две группы: g_0 и g_1. Группа g_0 является административной, для нее определен идентификатор $0 \in GroupIDs$ и роль $FullRole$. Субъекты, порожденные пользователями данной группы, имеют все разрешения на доступ к объектам. Субъекты, порожденные пользователями, входящими в группу g_1, имеют все разрешения на доступ к объектам только для чтения.

В системе существуют два пользователя: u_0 и u_1. Пользователь u_0 является администратором системы, для него определен идентификатор $0 \in UserIDs$ и входит в группу с идентификатором 0. Максимальный уровень конфиденциальности cl и целостности il порождаемых субъектов равен 2. То есть уровень конфиденциальности и целостности субъектов этого пользователя не может быть больше 2. При создании новых субъектов пользователь определяет доступные категории конфиденциальности и целостности субъектам (множества $\{"C1","C2"\}$ и $\{"I1","I2"\}$ соответственно) для назначения объектам. Множество идентификаторов исполняемых объектов $white\_list\_execute\_objects$, на основе которых могут быть созданы субъекты, пусто для обоих пользователей. Характеристика пользователя u_1 аналогична u_0 с учетом значения атрибутов.

Изначально в системе заданы два субъекта: s_0 и s_1. Субъект s_0 порожден пользователем с идентификатором 0. Для него заданы уровень конфиденциальности cl равный 0 и целостности 1 соответственно. Владельцем субъекта является пользователь с идентификатором 0. Характеристика субъекта s_1 аналогична характеристике субъекта s_0 с учетом значения атрибутов. Значение атрибута $content\_readed$ для обоих субъектов равно пустому множеству. Что означает отсутствие операций чтения объектов этими субъектами.

Изначально в системе задано 4 объекта, образующих иерархию файловой системы. Уровень вложенности объектов определяется значением атрибута $poid \in ObjPoids$ для каждого из них. Тип объекта определяется значением атрибута $type$: o_0 – корень файловой системы; o_1 – вложенный в o_0 каталог; o_2 – вложенный в o_1 файл данных; o_3 – вложенный в o_1 исполняемый файл. Владельцем для объектов o_0, o_1, o_3 является пользователь с идентификатором 0, для объекта o_2 – пользователь с идентификатором 1. Для объектов o_0 и o_3 в атрибуте $aclgs$ заданы дискреционные правила разграничения доступа для групп: группа с идентификатором 0 имеет полные права на доступ к объекту, группа с идентификатором 1 только на чтение. На уровне пользователей права доступа атрибутом $\{"I1","I2"\}$ разграничены только для объекта o_2: для пользователя с идентификатором 0 разрешено создание субъекта из объекта o_2.

Определение начального состояния системы с использованием TLA+ производится следующим выражением:

$$\begin{aligned}&Init \triangleq\\&\quad\wedge\ G = \{g_0, g_1\}\\&\quad\wedge\ U = \{u_0, u_1\}\\&\quad\wedge\ S = \{s_0, s_1\}\\&\quad\wedge\ O = \{o_0, o_1, o_2, o_3\}\\&\quad\wedge\ A = \{\}\end{aligned}$$

Предикаты действий

Последовательное выполнение предикатов действий в процессе верификации имитирует реальное поведение моделируемой системы. Каждый преди-

DOI: 10.21681/2311-3456-2021-1-41-56

Таблица 2

Атрибуты элементов множества пользователей при инициализации

Атрибут	u_0	u_1
uid	0	1
groups	{0}	{1}
is_admin	*TRUE*	*FALSE*
cl	2	1
conf_cats	{"C1","C2"}	{"C1"}
il	2	1
int_cats	{"I1","I2"}	{"I1"}
white_list_execute_objects	{}	{}

Таблица 3

Атрибуты элементов множества субъектов при инициализации

Атрибут	s_0	s_1
sid	0	1
uid	0	1
cl	0	1
conf_cats	{"C1","C2"}	{"C1"}
il	1	1
int_cats	{"I1","I2"}	{"I1"}
content_readed	{}	{}

Таблица 4

Атрибуты элементов множества объектов при инициализации

Атрибут	o_0	o_1	o_2	o_3
oid	0	1	2	3
type	"root_container"	"container"	"executable"	"file"
owner	0	0	1	0
poid	$\langle 0 \rangle$	$\langle 0,1 \rangle$	$\langle 0,1 \rangle$	$\langle 0,1 \rangle$
cl	0	0	0	1
conf_cats	{}	{}	{}	{}
il	0	0	0	0
int_cats	1	{}	{}	{}
ext_attr	{"check_child_perm", "ccnr"}	{"check_child_perm"}	{}	{}
exec_attr	"deny"	"deny"	"app"	"deny"
aclgs	$\{\langle 0, FullRole \rangle, \langle 1, ReadRole \rangle\}$	{}	{}	$\{\langle 0, FullRole \rangle, \langle 1, ReadRole \rangle\}$
aclus	{}	{}	$\{\langle 0, ExecuteRole \rangle\}$	{}

кат действия состоит из пред- и постусловий. Предусловиями являются предикаты, выполнение которых необходимо для того, чтобы система всегда переходила в одно из множества разрешенных состояний.

Предусловия можно разделить на три группы: взаимодействующие сущности соответствуют выполняемому действию, проверка возможности заданного действия по дискреционным правилам разграничения доступа, проверка возможности заданного действия по многоуровневым правилам разграничения доступа.

Постусловия изменяют значения переменных системы, тем самым она переходит в новое состояние. В настоящей модели определены следующие предикаты действий:

$$
\begin{aligned}
Next \triangleq\ & \\
& \vee ChangeOwnerD \quad \vee ReadD \quad \vee WriteD \\
& \vee InstallD \quad \vee ReadAttrD \quad \vee WriteAttrD \\
& \vee ChangeExtAttrsD \quad \vee ListFilesInFolderD \quad \vee SearchD \\
& \vee CreateUserD \quad \vee DeleteUserD \quad \vee AddSpecACLuD \\
& \vee DelSpecACLuD \quad \vee AddUserPermD \quad \vee DelUserPermD \\
& \vee ChangeUserPermD \quad \vee CreateGroupD \quad \vee DeleteGroupD \\
& \vee AddUserGroupD \quad \vee DeleteUserGroupD \quad \vee AddSpecACLgD \\
& \vee DelSpecACLgD \quad \vee AddGroupPermD \quad \vee DelGroupPermD \\
& \vee ChangeGroupPermD \quad \vee ChangeMACILevelD \quad \vee AppendmD \\
& \vee ChangeMACConfCatsD \quad \vee ChangeMAC!\ LevelD \quad \vee ChangeMACIntCatsD \\
& \vee ChangeExecTypeD \quad \vee CreateObjectD \quad \vee CreateSubjectD
\end{aligned}
$$

где *ChangeOwnerD* – смена владельца объекта; *ReadD* – чтение объекта; *WriteD* – изменение объекта, *InstallD* – установка программного обеспечения; *ReadAttrD* – чтение атрибутов объекта; *WriteAttrD* – изменение атрибутов объекта; *ChangeExtAttrsD* – изменение расширенных атрибутов объекта; *ListFilesInFolderD* – получение списка файлов в каталоге (чтение контейнера); *SearchD* – поиск объекта; *CreateUserD* – добавление пользователя; *DeleteUserD* – удаление пользователя; *AddSpecACLuD* – добавление прав доступа на объект для пользователя; *DelSpecACLuD* – удаление прав доступа на объект для пользователя; *AddUserPermD* – добавление разрешений для пользователя к объекту; *DelUserPermD* – удаление разрешений для пользователя к объекту; *ChangeUserPermD* – изменение разрешений для пользователя к объекту; *CreateGroupD* – добавление группы; *DeleteGroupD* – удаление группы; *AddUserGroupD* – добавление пользователя в группу; *DeleteUserGroupD* – удаление пользователя из группы; *AddSpecACLgD* – добавление прав доступа к объекту для группы; *DelSpecACLgD* – удаление прав доступа к объекту для группы; *AddGroupPermD* – добавление разрешений для группы к объекту; *DelGroupPermD* – удаление разрешений для группы к объекту; *ChangeGroupPermD* – изменения разрешений для группы к объекту; *AppendmD* – дозапись в объект; *ChangeMACConfCatsD* – изменение категорий конфиденциальности объекта; *ChangeMACILevelD* – изменение уровня целостности объекта; *ChangeMACIntCatsD* – изменение категорий целостности объекта; *ChangeExecTypeD* – изменения типа запуска исполняемого объекта; *ChangeMAC! LevelD* – изменение уровня конфиденциальности объекта; *CreateSubjectD* – создание субъекта; *KillSubjectD* – завершение работы субъекта; *CreateObjectD* – создание объекта; *cl* – удаление объекта; *AddAppToWhiteListD* – добавление приложения в список разрешенных для запуска пользователю; *RemoveAppFromWhiteListD* – удаление приложения из списка разрешенных для запуска пользователю.

Предикат действия *WriteD* моделирует изменение некоторым субъектом s в объект $conf\_cats$ и представлен в выражении 7.

$$
\begin{aligned}
& WriteD \triangleq \\
& \quad \exists s \in S : \\
& \quad \exists o \in O : \\
& \qquad \wedge\ o.type = "file" \\
& \qquad \wedge\ DAC\_may\_do(s, o, "write") \\
& \qquad \wedge\ MAC\_may\_write(s, o) \\
& \qquad \wedge\ Write(s, o)
\end{aligned}
\tag{7}
$$

Предусловия предиката действия *WriteD* определяют, что изменение субъектом s в объект o может быть совершено только для объектов типа «файл». В остальных случаях действие не будет совершено. Далее проверяется возможность совершения доступа согласно дискреционному и многоуровневому разграничению доступа. Если оба предиката возвращают истинное значение, то выполняется оператор $Write(s, o)$.

Постусловия для предиката действия *WriteD* представлены в выражении 8.

$$
\begin{aligned}
& Write \triangleq \\
& \quad \wedge\ A' = A \cup \{\langle s.sid, o.oid, "write" \rangle\} \\
& \quad \wedge\ UNCHANGED \langle U, G, S \rangle
\end{aligned}
\tag{8}
$$

Выполнения постусловия *WriteD* добавляет во множество совершенных доступов A кортеж, содержащий идентификатор субъекта, идентификатор объекта и имя операции. Переменные U, G, S остаются неизменными.

DOI: 10.21681/2311-3456-2021-1-41-56

Инварианты модели

Модель является верифицированной, если для каждого из возможных состояний системы выполняются все инварианты – глобальные свойства безопасности. На каждом шаге верификации выполнение всех инвариантов подтверждает, что все переменные находятся в допустимых доменах значений, данное состояние не является запрещенным. Инвариант представляет собой логическое выражение, включающее в себя темпоральные операторы. Инварианты модели можно разделить на следующие группы.

Инварианты корректности: $TypeInv$, $NoCyclesInContainers$, $UserACLCardinality$, $GroupACLCardinality$.

Инварианты жизнеспособности: $OneAdminExists$, $MACSafety$.

Инварианты безопасности: $MACAccessSafety$, $DACAccessSafety$.

При выполнении верификации важно, чтобы все переменные находились в заданных для них доменах значений. Для достижения этого требования введен инвариант $TypeInv$, представленный следующим выражением:

$$\begin{aligned} &TypeInv \triangleq \\ &\quad \wedge\ S \subseteq Subjects \\ &\quad \wedge\ O \subseteq Objects \\ &\quad \wedge\ U \subseteq Users \\ &\quad \wedge\ G \subseteq Groups \end{aligned}$$

Инвариант $TypeInv$ не проверяет вхождение переменных из множества A в заданный домен значений, так как элементы кортежей из множества совершенных доступов имеют различные домены значений. Например, для действий, совершенных по отношению к субъектам, вторым элементом кортежа будет идентификатор $sid \in SubjectIDs$.

Для подтверждения отсутствия образования циклов в иерархии объектов определен инвариант $NoCyclesInContainers$, представленный следующим выражением:

$$\begin{aligned} &NoCyclesInContainers \triangleq \\ &\quad \forall o \in SelectContainers: \\ &\qquad Len(o.poid) = Cardinality(Range(o.poid)) \end{aligned},$$

где $SelectContainers$ – множество объектов, состоящее из объектов, являющихся контейнерами. Для отсутствия циклов в иерархии объектов достаточно, чтобы длина кортежа вложенности объектов $o.poid$ была равна мощности множества, полученного из него, то есть отсутствовали дублирующие элементы.

Инварианты $UserACLCardinality$ и $GroupACLCardinality$ проверяют, что для каждого пользователя и группы соответственно существует не более одного ACL для одного объекта (маска доступа). Данное требование необходимо для ускорения процесса верификации и исключения ситуации с дублированием разрешений. Инвариант $UserACLCardinality$ аналогичен $GroupACLCardinality$ и представлен следующим выражением:

$$\begin{aligned} &UserACLCardinality \triangleq \\ &\quad \forall u \in U: \\ &\quad \forall o \in O: \\ &\qquad Cardinality(SelectAllAclByUser(u,o)) \le 1 \end{aligned}$$

Для реализации системой своих функций и исключения состояния отсутствия возможных переходов (deadlock) в ней должен присутствовать режим администрирования, для этого должен существовать как минимум один администратор, что подтверждается выполнением следующего инварианта:

$$OneAdminExists \triangleq \exists u \in U: u.is\_admin$$

В соответствии с правилами реализации многоуровневого разграничения доступа для любого состояния системы должен выполняться инвариант $MACSafety$:

$$\begin{aligned} &MACSafety \triangleq \\ &\quad \wedge\ \forall o \in O: \\ &\qquad \vee\ \wedge\ o.type \in Containers \\ &\qquad\quad \wedge\ \forall ch \in SelectAllChilds(o): \\ &\qquad\qquad \wedge\ \vee\ "ccnr" \in o.ext\_attr \\ &\qquad\qquad\quad \vee\ \wedge\ ch.cl \le o.cl \\ &\qquad\qquad\qquad \wedge\ \vee\ o.conf\_cats = \{\} \\ &\qquad\qquad\qquad\quad \vee\ ch.conf\_cats \subseteq o.conf\_cats \\ &\qquad\qquad \wedge\ \vee\ "icnr" \in o.ext\_attr \\ &\qquad\qquad\quad \vee\ \wedge\ ch.cl \ge o.cl \\ &\qquad\qquad\qquad \wedge\ \vee\ o.int\_cats = \{\} \\ &\qquad\qquad\qquad\quad \vee\ ch.int\_cats \subseteq o.int\_cats \\ &\qquad \vee\ o.type \notin Containers \\ &\quad \wedge\ \forall s \in S: \\ &\qquad \wedge\ s.cl \le SelectUser(s.uid).cl \\ &\qquad \wedge\ s.conf\_cats \subseteq SelectUser(s.uid).conf\_cats \\ &\qquad \wedge\ s.il \le SelectUser(s.uid).il \\ &\qquad \wedge\ s.int\_cats \subseteq SelectUser(s.uid).int\_cats \end{aligned}$$

В данном инварианте происходит проверка, что для каждого объекта, если он является контейнером, должен содержаться флаг "$ccnr$" или "$icnr$" в его расширенных атрибутах, или уровни конфиденциальности и целостности вложенных объектов должны быть ниже уровней самого объекта соответственно. При отсутствии флагов "$ccnr$" или "$icnr$" категории вложенных объектов контейнера должны быть подмноже-

ствами категорий самого контейнера. Выполнение этих правил касается также и субъектов, созданных пользователями.

Инварианты безопасности проверяют, что в процессе верификации система не перешла в одно из запрещенных состояний. В данном случае запрещенным состоянием является совершение несанкционированного доступа. При корректном определении предусловий совершения операций доступа перехода системы в такое состояние быть не должно. Проверка наличия доступов, нарушающих многоуровневое разграничение доступа, производится в инвариантах $MACAccessSafety$, а дискреционного в $DACAccessSafety$.

$$
\begin{aligned}
&MACAccessSafety \triangleq \\
&\quad \neg\exists a \in A: \\
&\quad\quad \wedge \ \vee \ a[3] = "read" \\
&\quad\quad\quad\ \vee \ a[3] = "write" \\
&\quad\quad\quad\ \vee \ a[3] = "install" \\
&\quad\quad\quad\ \vee \ a[3] = "read\_attr" \\
&\quad\quad\quad\ \vee \ a[3] = "write\_attr" \\
&\quad\quad\quad\ \vee \ a[3] = "append" \\
&\quad\quad\quad\ \vee \ a[3] = "list\_files" \\
&\quad\quad\quad\ \vee \ a[3] = "search" \\
&\quad\quad \wedge \ \neg IsUserAdmin(SelectUser(SelectSubject(a[1]).uid)) \\
&\quad\quad \wedge \ \vee \neg SelectObject(a[2]).conf\_cats \subseteq SelectSubject(a[1]).conf\_cats \\
&\quad\quad\quad\ \vee \neg SelectObject(a[2]).cl > SelectSubject(a[1]).cl \\
&\quad\quad\quad\ \vee \neg SelectObject(a[2]).int\_cats \subseteq SelectSubject(a[1]).int\_cats \\
&\quad\quad\quad\ \vee \neg SelectObject(a[2]).il < SelectSubject(a[1]).il
\end{aligned}
$$

Определение инварианта $MACAccessSafety$ интерпретируется следующим образом: «не должно существовать такого кортежа $a \in A$, который добавлен при операции чтения, записи, установки, записи атрибутов, дозаписи, чтения каталога, поиска, и субъект, выполнивший эту операцию, не является администратором и при этом множество категорий объекта не является подмножеством категорий субъекта, или уровень конфиденциальности/целостности объекта больше уровня конфиденциальности/целостности субъекта.

Определение инварианта $DACAccessSafety$ интерпретируется следующим образом: «не должно существовать такого кортежа $a \in A$, который добавлен при операции чтения, записи, установки, записи атрибутов, дозаписи, чтение каталога, поиска и субъект, выполнивший эту операцию, не является администратором, и при этом операция доступа должна быть разрешена дискреционными правилами доступа к объекту.

Выполнение инвариантов безопасности для каждого из возможных состояний модели подтверждает,

$$
\begin{aligned}
&DACAccessSafety \triangleq \\
&\quad \neg\exists a \in A: \\
&\quad\quad \wedge \ \vee \ a[3] = "read" \\
&\quad\quad\quad\ \vee \ a[3] = "write" \\
&\quad\quad\quad\ \vee \ a[3] = "install" \\
&\quad\quad\quad\ \vee \ a[3] = "read\_attr" \\
&\quad\quad\quad\ \vee \ a[3] = "write\_attr" \\
&\quad\quad\quad\ \vee \ a[3] = "append" \\
&\quad\quad\quad\ \vee \ a[3] = "list\_files" \\
&\quad\quad\quad\ \vee \ a[3] = "search" \\
&\quad\quad \wedge \ \neg IsUserAdmin(SelectUser(SelectSubject(a[1]).uid)) \\
&\quad\quad \wedge \ \vee \neg DAC\_may\_access\_cont\_hierach(SelectSubject(a[1]), SelectObject(a[2]), a[3]) \\
&\quad\quad\quad\ \vee \neg DAC\_may\_do(SelectSubject(a[1]), SelectObject(a[2]), a[3])
\end{aligned}
$$

DOI: 10.21681/2311-3456-2021-1-41-56

выполнение заданной для нее спецификации, это означает что система не перешла в запрещенное состояние.

Выводы

Реализация предложенного в настоящей модели многоуровневого управления доступом на основе уровней и категорий целостности и конфиденциальности в существующих операционных системах семейства Windows позволит ужесточить контроль над образованием информационных потоков при доступе субъектов к объектам, что повысит общий уровень защищенности информации, обрабатываемой в ОС. Применяемая для описания модели нотация языка TLA+ позволяет произвести полностью автоматическую верификацию модели, что исключает влияние человеческого фактора на результат верификации, но не исключает возможность совершения ошибки при формировании спецификации модели. Направлением дальнейших исследований является анализ модели политики безопасности управления доступом на наличие скрытых каналов утечки информации по времени, а также ее верификация.

Литература

1. Leuenberger, Adrian. “Shatter Attack Privilege Escalation on Win32Systems.” CSNC Security Event 2003, Compass Security.
2. S. Zune, Mandatory access control by Biba model. M.C.Sc Dissertation. University of Computer Studies, Yangon, 2018.
3. Gartaganis, Charalampos. “Comparative analysis of the Windows security XP, Vista, 7, 8 and 10.” Master’s thesis, Πανεπιστήμιο Πειραιώς, 2017.
4. Yile, Fan. “Research on the Security Problem in Windows 7 Operating System.» 2016 Eighth International Conference on Measuring Technology and Mechatronics Automation (ICMTMA). IEEE, 2016. DOI: 10.1109/ICMTMA.2016.139
5. Havelund, Klaus, and Doron Peled. “Runtime verification: From propositional to first-order temporal logic.” In International Conference on Runtime Verification, pp. 90-112. Springer, Cham, 2018. DOI: 10.1007/978-3-030-03769-7_7
6. Kozachok A., Bochkov M., Lai Minh T., Kochetkov E. First order logic for program code functional requirements description // Вопросы кибербезопасности. 2017. № 3 (21). С. 2-7. DOI: 10.21681/2311-3456-2017-3-2-7
7. Markus Alexander Kuppe, Leslie Lamport and Daniel Ricketts. The TLA+ toolbox. In F-IDE@FM 2019, pages 50–62, 2019. DOI: 10.4204/EPTCS.310.6
8. McMillan K. L. Eager Abstraction for Symbolic Model Checking // Computer Aided Verification / ed. by G. Chockler Hana and Weissenbacher. Cham : Springer International Publishing, 2018. P. 191-208. DOI: 10.1007/978-3-319-96145-3_11.
9. Девянин П.Н. О проблеме представления формальной модели политики безопасности операционных систем. Труды ИСП РАН, том 29, вып. 3, 2017. С. 7-16. DOI: 10.15514/ISPRAS-2017-29(3)-1
10. Devyanin, P.N., Khoroshilov, A.V., Kuliamin, V.V., Petrenko, A.K. and Shchepetkov, I.V., 2020. Integrating RBAC, MIC, and MLS in verified hierarchical security model for operating system. Programming and Computer Software, 46(7), pp.443-453. DOI: 10.1134/S0361768820070026
11. Efremov, D. and Shchepetkov, I., 2019, October. Runtime Verification of Linux Kernel Security Module. In International Symposium on Formal Methods (pp. 185-199). Springer, Cham. DOI: 10.1007/978-3-030-54997-8_12
12. Georget, L., Jaume, M., Tronel, F., Piolle, G., Tong, V.V.T.: Verifying the reliability of operating system-level information flow control systems in Linux. In: 2017 IEEE/ACM 5th International FME Workshop on Formal Methods in Software Engineering (FormaliSE), pp. 10–16 DOI: 10.1109/FormaliSE.2017.1
13. Baier C., Katoen J. P. Principles of model checking. MIT press, 2008. ISBN: 978-0-262-02649-9
14. Zheng, B., Li, W., Deng, P., Gérard, L., Zhu, Q. and Shankar, N., 2015, June. Design and verification for transportation system security. In Proceedings of the 52nd annual design automation conference (pp. 1-6). DOI: 10.1145/2744769.2747920
15. Стародубцев Ю.И., Бегаев А.Н., Козачок А.В. Способ управления доступом к информационным ресурсам мультисервисных сетей различных уровней конфиденциальности // Вопросы кибербезопасности. 2016. № 3 (16). С. 13-17. DOI:10.21681/2311-3456-2016-3-13-17
16. Козачок А.В. Спецификация модели управления доступом к разнокатегорийным ресурсам компьютерных систем // Вопросы кибербезопасности. 2018. № 4 (28). С. 2-8. DOI: 10.21681/2311-3456-2018-4-2-8
17. Козачок А. В., Кочетков Е. В. Обоснование возможности применения верификации программ для обнаружения вредоносного кода // Вопросы кибербезопасности. 2016. Т. 16, № 3. С. 25—32. DOI:10.21681/2311-3456-2016-3-25-32
18. Козачок А.В., Туан Л.М. Комплекс алгоритмов контролируемого разграничения доступа к данным, обеспечивающий защиту от несанкционированного доступа // Системы управления и информационные технологии. 2015. № 3 (61). С. 58-61.
19. Kozachok, A.: TLA+ based access control model specification. In: Proceedings of the Institute for System Programming of the RAS, vol. 30, pp. 147–162, January 2018. DOI: 10.15514/ISPRAS-2018-30(5)-9
20. Bell, D.E., LaPadula, L.: Secure Computer Systems: Mathematical Model. ESD-TR 73-278, The MITRE Corporation, McLean (1973)

MULTI-LEVEL POLICY MODEL ACCESS CONTROL SECURITY OPERATING SYSTEMS OF THE WINDOWS FAMILY

Kozachok V.I.[5], Kozachok A.V.[6], Kochetkov E.V.[7]

*The **purpose** of research – development of a more advanced Windows NT family access control mechanism to protect against information leakage from memory by hidden channels.*

*The **method** of research – analysis of Windows NT family models of mandatory access control and integrity control, modeling of access control security policy for specified security properties, automatic verification of models. The Lamport Temporal Logic of Actions (TLA +) used to describe the model and its specification is used. TLA+ allows automatic verification of the model with the specified security properties.*

*The **result** of research – revealed the main limitations of the existing mandatory integrity control of operating systems of the Windows NT family. A set of structures of a multilevel model has been developed, reflecting the attributes that are significant for modeling the process of access of subjects to objects. The key mechanisms of access control in the operating system are modeled: management of users, groups, subjects, objects, roles, rights, discretionary and mandatory access control, mandatory integrity control - multilevel control of subjects' access to objects. The model defines a mechanism for controlling the creation of subjects based on executable files to organize an isolated software environment. The values of the attributes of the model variables for the initialization stage are determined. The invariants of variables correctness in the process of verification and subjects to objects safe access are developed. The model was specified using the TLA + modeling language and verified.*

Keywords: *information security, unauthorized access, access control model, model verification.*

References

1. Leuenberger, Adrian. "Shatter Attack Privilege Escalation on Win32Systems." CSNC Security Event 2003, Compass Security.
2. S. Zune, Mandatory access control by Biba model. M.C.Sc Dissertation. University of Computer Studies, Yangon, 2018
3. Gartaganis, Charalampos. "Comparative analysis of the Windows security XP, Vista, 7, 8 and 10." Master's thesis, Πανεπιστήμιο Πειραιώς, 2017
4. Yile, Fan. "Research on the Security Problem in Windows 7 Operating System.» 2016 Eighth International Conference on Measuring Technology and Mechatronics Automation (ICMTMA). IEEE, 2016. DOI: 10.1109/ICMTMA.2016.139
5. Havelund, Klaus, and Doron Peled. "Runtime verification: From propositional to first-order temporal logic." In International Conference on Runtime Verification, pp. 90-112. Springer, Cham, 2018. DOI: 10.1007/978-3-030-03769-7_7
6. Kozachok A.V., Bochkov M.V., Lai Minh T., Kochetkov E.V. First order logic for program code functional requirements description // Voprosy kiberbezopasnosti. 2017. № 3 (21). С. 2-7. DOI: 10.21681/2311-3456-2017-3-2-7
7. Markus Alexander Kuppe, Leslie Lamport, and Daniel Ricketts. The TLA+ toolbox. In F-IDE@FM 2019, pages 50–62, 2019. DOI: 10.4204/EPTCS.310.6
8. McMillan K. L. Eager Abstraction for Symbolic Model Checking // Computer Aided Verification / ed. by G. Chockler Hana and Weissenbacher. Cham : Springer International Publishing, 2018. P. 191-208. DOI: 10.1007/978-3-319-96145-3_11.
9. Devyanin, P.N., On the problem of representation of the formal model of security policy for operating systems. Proceedings of the Institute for System Programming of the RAS, 2017, 29(3), pp.7-16 DOI: 10.15514/ISPRAS-2017-29(3)-1
10. Devyanin, P.N., Khoroshilov, A.V., Kuliamin, V.V., Petrenko, A.K. and Shchepetkov, I.V., 2020. Integrating RBAC, MIC, and MLS in verified hierarchical security model for operating system. Programming and Computer Software, 46(7), pp.443-453. DOI: 10.1134/S0361768820070026
11. Efremov, D. and Shchepetkov, I., 2019, October. Runtime Verification of Linux Kernel Security Module. In International Symposium on Formal Methods (pp. 185-199). Springer, Cham. DOI: 10.1007/978-3-030-54997-8_12

5 Vasilii Kozachok, Dr. Sc., Academy of Federal Guard Service, Oryol, Russia. E-mail: v.kozachok@academ.msk.rsnet.ru
6 Alexander Kozachok, Dr. Sc., Academy of Federal Guard Service, Oryol, Russia. E-mail: a.kozachok@academ.msk.rsnet.ru
7 Evgenii Kochetkov, Academy of Federal Guard Service, Oryol, Russia. E-mail: e.kochetkov@academ.msk.rsnet.ru

DOI: 10.21681/2311-3456-2021-1-41-56

12. Georget, L., Jaume, M., Tronel, F., Piolle, G., Tong, V.V.T.: Verifying the reliability of operating system-level information flow control systems in Linux. In: 2017 IEEE/ACM 5th International FME Workshop on Formal Methods in Software Engineering (FormaliSE), pp. 10–16 DOI: 10.1109/FormaliSE.2017.1
13. Baier C., Katoen J. P. Principles of model checking. – MIT press, 2008.
14. Zheng, B., Li, W., Deng, P., Gérard, L., Zhu, Q. and Shankar, N., 2015, June. Design and verification for transportation system security. In Proceedings of the 52nd annual design automation conference (pp. 1-6). DOI: 10.1145/2744769.2747920
15. Starodubcev YU.I., Begaev A.N., Kozachok A.V. Sposob upravleniya dostupom k informacionnym resursam mul'tiservisnyh setej razlichnyh urovnej konfidencial'nosti // Voprosy kiberbezopasnosti. 2016. № 3 (16). pp. 13-17. DOI:10.21681/2311-3456-2016-3-13-17
16. Kozachok A.V. Specifikaciya modeli upravleniya dostupom k raznokategorijnym resursam komp'yuternyh sistem // Voprosy kiberbezopasnosti. 2018. № 4 (28). pp. 2-8. DOI: 10.21681/2311-3456-2018-4-2-8
17. Kozachok A. V., Kochetkov E. V. Obosnovanie vozmozhnosti primeneniya verifikacii programm dlya obnaruzheniya vredonosnogo koda // Voprosy kiberbezopasnosti. – 2016. – T. 16, № 3. – pp. 25–32. DOI:10.21681/2311-3456-2016-3-25-32
18. Kozachok A.V., Tuan L.M. Kompleks algoritmov kontroliruemogo razgranicheniya dostupa k dannym, obespechivayushchij zashchitu ot nesankcionirovannogo dostupa // Sistemy upravleniya i informacionnye tekhnologii. 2015. № 3 (61). pp. 58-61.
19. Kozachok, A.: TLA+ based access control model specification. In: Proceedings of the Institute for System Programming of the RAS, vol. 30, pp. 147–162, January 2018. DOI: 10.15514/ISPRAS-2018-30(5)-9
20. Bell, D.E., LaPadula, L.: Secure Computer Systems: Mathematical Model. ESD-TR 73-278, The MITRE Corporation, McLean (1973)