О построении формальной модели управления доступом QP ОС

М.А. Алехина, (д.ф.-м.н., alekhina.marina19@yandex.ru), Васин А.В. (к.ф.-м.н., alvarvasin@mail.ru), Егоров В.Ю. (к.т.н., vec@cryptosoft.ru), Афонин А.Ю. (к.т.н., aau@cryptosoft.ru), НТП «Криптософт» (mails@cryptosoft.ru, г. Пенза, ул. Лермонтова, д. 3)

Ключевые слова: *Операционная система QP ОС, формальная модель механизма контроля доступа, дискреционный контроль доступа, мандатный контроль доступа, ролевое управление доступом, мандатный контроль доверия.*

Аннотация. Кратко описана формальная модель *механизма контроля доступа операционной системы QP ОС, которая включает в себя дискреционный контроль доступа, мандатный контроль доступа, ролевой контроль доступа, мандатный контроль доверия. Показано применение механизмов контроля доступа при выполнении операций над объектами системы.*

Введение

НТП «Криптософт» около 20 лет занимается разработкой защищённой операционной системы QP ОС[1]. QP ОС является многопользовательской операционной системой. В состав QP ОС входит подсистема безопасности, призванная обеспечить защищенность информации и ресурсов системы от действия объективных и субъективных, внешних и внутренних, случайных и преднамеренных угроз.

В соответствии с требованиями ФСТЭК одним из этапов сертификации является построение формальной модели управления доступом. Разработка формальной модели управления доступом позволяет повысить доверие к ОС, а также обеспечить надежное и корректное функционирование средств защиты информации.

В соответствии с реализованными в ОС средствами контроля доступа, разрабатываемая модель состоит из трех частей: дискреционный контроль доступа, мандатный контроль доступа и ролевое управление доступом.

Общие элементы модели и допущения

Все множества (субъектов, объектов, сессий, пользователей и т.д.), рассматриваемые в этой работе, считаются конечными. Множество субъектов системы обозначим через $Subjects$. В системе присутствует два типа субъектов: пользователи $Users \subset Subjects$ и группы пользователей $Groups \subset Subjects$. Группа содержит в себе некоторое подмножество пользователей, что задается отношением: $GroupUser: Users \leftrightarrow Groups$. Множество идентификаторов (дескрипторов безопасности) субъектов системы обозначим через $Sids$. Оно связано с множеством субъектов отношением $SubjectSid: Subjects \rightarrowtail Sids$. Каждому пользователю или каждой группе в системе однозначно сопоставлен идентификатор безопасности.

Группа администраторов безопасности выделена как отдельная группа, обладающая полномочиями администрирования системы безопасности:

$$Admins \in Groups, \qquad GroupUser \rhd \{Admins\} \neq \emptyset.$$

Множество объектов обозначим через $\mathrm{Objects}$. Множество объектов-контейнеров обозначим через $\mathrm{Containers}$. $\mathrm{Objects} \cap \mathrm{Containers} = \emptyset$. Данные множества представляют собой разбиение множества сущностей $\mathrm{Entities} = \mathrm{Objects} \cup \mathrm{Containers}$. Каждая сущность имеет свой тип. Множество типов обозначим через Types: $\mathrm{Type}: \mathrm{Entities} \longrightarrow \mathrm{Types}$. Множество видов/прав доступа обозначим через $\mathrm{AccessRights}$. Каждому типу сущности, соответствует некоторое подмножество прав доступа $\mathrm{Rd}: \mathrm{Types} \leftrightarrow \mathrm{AccessRights}$ применяемых к объекту. Владелец сущности задается отношением $\mathrm{EntityOwner}: \mathrm{Entities} \rightarrow \mathrm{Sids}$.

Модель дискреционного контроля доступа

После того, как пользователь $U \in Users$ выполнил процедуру входа в систему, для него создаются сессия $session \in Sessions$ и маркер доступа $token_{session} \in Tokens$.

Пользователь, сессия и маркер связаны отношениями $SessionUser: Users \to Sessions$ и $SessionToken: Sessions \rightarrowtail Tokens$.

Все процессы и нити в системе являются сущностями. Процесс и нить имеют однозначное соответствие с маркером: $GetToken: Entities \twoheadrightarrow Tokens$.

При создании процесса ему присваивается маркер сессии или маркер создающего процесса, при создании нити ей присваивается маркер процесса. Если нить олицетворяется, то ей может быть присвоен маркер, отличный от маркера процесса.

При помощи отношения $TokenUser$ каждый маркер связан с идентификатором пользователя $sid \in Sids$, от имени которого выполняется процесс: $TokenUser: Tokens \twoheadrightarrow Sids$.

Маркер безопасности содержит идентификаторы групп пользователей с разрешающими или запрещающими флагами, последнее в модели реализуется при помощи отношения $TokenGroups: Tokens \leftrightarrow Sids \times BOOL$.

Маркер может быть ограниченным и/или олицетворенным, что определяется отношениями: $TokenIsRestricted: Tokens \to BOOL$ и $TokenIsImpersonated: Tokens \to BOOL$.

Операционная система выполняет дискреционный контроль доступа всякий раз, когда необходимо выполнить операцию открытия объекта. Дискреционный контроль доступа производится на основе списков дискреционного контроля доступа $DACLs$. Сущности при создании ставится в соответствие список контроля доступа $dacl \in DACLs$, связанный с сущностью отношением $EntityDACL: Entities \twoheadrightarrow DACLs$. Каждый список доступа $dacl$ связывается с правилами доступа отношением $ACEs$:

$ACEs: DACLs \twoheadrightarrow (Sids \times (AccessRights \times Types) \twoheadrightarrow \mathbb{P}(RFS))$, где RFS – множество флагов наследования правил, а также разрешающих/запрещающих флагов.

Операционная система выполняет дискреционный контроль доступа всякий раз, когда необходимо выполнить операцию открытия сущности. В случае успешного контроля доступа с процессом связывается дескриптор безопасности. В модели дескрипторы безопасности представлены двумя отношениями. Первое отношение связывает нить и идентификатор дескриптора (представляющий собой натуральное число): $HandlesOfThread: Containers \leftrightarrow \mathbb{N}1$. Второе отношение связывает процесс, в рамках которого открыта сущность, дескриптор, сущность и полученные права: $Handle: Containers \twoheadrightarrow (\mathbb{N}1 \times Entities \leftrightarrow AccessRights)$.

В операционной системе контроль доступа осуществляет функция $access\_check\ (E, token, rd)$. К сожалению, система Rodin не позволяет выполнить реализацию этой функции, поэтому проверка прав осуществляется непосредственным вычислением в одноименном событии $access\_check$. Рассмотрим этот процесс подробнее.

Если пользователь при проверке доступа к сущности E имеет права владельца, то функция $access\_check$ предоставляет запрашиваемые права без просмотра списка доступа dacl. В противнос случае множество разрешенных пользователю прав относительно сущности E определяется следующим образом:

1. Пусть E – сущность, для которой осуществляется проверка прав доступа, $t = Type(E)$ - тип сущности, $r = Rd[t]$ – множество прав, которые могут быть применены к объекту данного типа.

2. По маркеру безопасности определяется множество идентификаторов $S_o \in Sids$, для которых будет проводиться следующая проверка прав: сначала определяется sid, от имени которого выполняется процесс: $sid = TokenUser(token)$, затем определяется множество идентификаторов G^0_{sid} разрешенных групп пользователя, от имени которого выполняется процесс: $G^0_{sid} = dom(ran(\{token\} \lhd TokenGroups) \rhd \{TRUE\})$. Тогда полагаем $S_o = \{sid\} \cup G^0_{sid}$.

3. По списку доступа $dacl = EntityDACL(E)$, определяется множество правил, применяемых для полученного множества идентификаторов S_o:

$$Rules_0 = S_o \times (r \times \{t\}) \lhd ACEs(dacl)\,.$$

4. Из разрешающих правил множества $Rules_0$ получается множество разрешенных прав доступа:

$$Allowed = dom\Big(ran\big(dom(Rules_0 \rhd FlagSetsWithTRUE)\big)\Big).$$

5. По маркеру $token$ определяется множество G^1_{sid} идентификаторов групп пользователя (всех, в том числе и исключенных из маркера): $G^1_{sid} = dom(ran(\{token\} \lhd TokenGroups))$.

6. По списку доступа $dacl = EntityDACL(E)$, определяется множество правил, применяемых для выбранного множества идентификаторов групп: $Rules_1 = G^1_{sid} \times (r \times \{t\}) \lhd ACEs(dacl)$.

7. Из запрещающих правил множества $Rules_1$ получается множество запрещенных прав доступа:

$$Denied = dom\Big(ran\big(dom(Rules_1 \rhd FlagSetsWithFALSE)\big)\Big),$$

где $FlagSetsWithTRUE \in \mathbb{P}(RFS)$ и $FlagSetsWithFALSE \in \mathbb{P}(RFS)$ – множества флагов с разрешающим и запрещающим признаком соответственно.

Множество прав доступа пользователя к объекту определяется как разность множеств разрешения доступа и запрета доступа: $Allowed \backslash Denied$.

Тогда имеем:

$$access_{check(E,token,rd)} = \begin{cases} true, \text{если } EntityOwner(E) = TokenUser(token), \\ true, rd \subseteq Allowed \backslash Denied, \\ false, \text{в остальных случаях.} \end{cases}$$

Модель мандатного контроля доступа

Мандатный (полномочный) контроль доступа основан на сравнении наборов мандатных категорий, приписанных субъекту и объекту, и содержит условия (критерии) получения доступа к объектам. Существуют два типа мандатных категорий – иерархические и неиерархические.

Под иерархической категорией будем понимать конечное, вполне упорядоченное множество. Например, рассмотрим множество степеней секретности A={несекретно, ДСП, секретно, совершенно секретно} и введем на нем отношение линейного порядка “не выше” (≪), считая, что “несекретно” ≪ ”ДСП” ≪ “секретно” ≪ “совершенно секретно”. Очевидно, легко установить изоморфизм между упорядоченными множествами (A, ≪) и ({0,1,2,3}, ≤), где знак “ ≤ “ означает “не больше” (в обычном смысле). Поэтому далее иерархической категорией будем называть вполне упорядоченное множество (B={0,1,…, l}, ≤), $l \leq 15$. Заметим, что число иерархических категорий не больше 8. Обозначим их через B_1, B_2,…, B_k ($k \leq 8$).

Неиерархической категорией будем называть множество, состоящее из одного элемента. Примером неиерархической категории может быть руководство компании, бухгалтерия компании или секретариат компании. Число неиерархических категорий не превосходит 16. Упорядочим эти категории и обозначим через C_1, C_2,…, C_m ($m \leq 16$). Поскольку каждая из них может либо присутствовать (1), либо отсутствовать (0), неиерархическую категорию будет представлять множество {0,1}.

Каждому объекту и каждому субъекту припишем набор длины k+m, который является элементом декартова произведения множеств $B_1 \times B_2 \times \ldots \times B_k \times \{0,1\}^m$. Обозначим П= $B_1 \times B_2 \times \ldots \times B_k \times \{0,1\}^m$.

Считаем, что наборы α и β из П равны, если равны их соответствующие компоненты.

Будем говорить, что набор α из П предшествует набору β из П, если каждая компонента набора α не больше соответствующей компоненты набора β (обозначение α ≤ β). Два набора из множества П сравнимы, если один из них предшествует другому. В противном случае будем называть наборы α и β несравнимыми. Если α ≤ β и α ≠ β, то будем говорить, что α меньше β и писать α < β, а также, что β больше α и писать β > α.

Набор α из множества П будем называть классификационным уровнем (КУ) субъекта (объекта защиты).

Определено множество прав доступа Rm с точки зрения мандатного контроля доступа с двумя режимами «чтение» и «запись»: $Rm = \{write, read\}$.

Правила доступа формулируются следующим образом:

- если КУ субъекта равен КУ объекта защиты, то субъект может и «читать» и «писать» данные этого объекта;
- субъект может «читать» данные объекта, если КУ субъекта больше КУ этого объекта;
- субъект может «писать» данные объекта, если КУ субъекта меньше КУ объекта;
- если КУ субъекта и КУ объекта защиты несравнимы, то доступ запрещен.

Замечание. Наборы из множеств $B_1 \times B_2 \times \ldots \times B_k$ и $\{0,1\}^m$ будем сравнивать также, как наборы из П. Набор из множества $B_1 \times B_2 \times \ldots \times B_k$ будем называть КУ субъекта (объекта защиты) относительно иерархических категорий, а набор из множества $\{0,1\}^m$ будем называть КУ субъекта (объекта защиты) относительно неиерархических категорий.

Операция изменения классификационного уровня объекта подчиняется следующим правилам:

изменение классификационного уровня существующего объекта допускается только в сторону повышения значений иерархических мандатных категория;

при установленном значении неиерархической мандатной категории, удалить его с классификационного уровня объекта невозможно.

Каждому пользователю присваивается множество КУ субъекта, моделируемое посредством отношения: $UserMandatRule: Users \twoheadrightarrow \mathbb{P}1(П)$.

При создании маркера доступа для пользователя $U \in Users$ каждому маркеру ставится в соответствие КУ из множества $UserMandatRule(U)$, моделируемый отношением $TokenMandat: Users \twoheadrightarrow П$

Отношение $EntityXACLMandat: Entities \twoheadrightarrow П$ ставит в соответствие каждому объекту его КУ.

Проверка прав доступа по правилам мандатного контроля осуществляется согласно функции $maccess\_check(E, U, Rm)$ трёх переменных: объекта, пользователя и запрашиваемого права мандатного доступа, значения которой задаются следующим образом:

$$maccess\_check(E, U, \mathrm{Rm}):$$

$$maccess\_check(E, U, \{write\}) = TokenMandat(U) \leq EntityXACLMandat(E)$$

$$maccess\_check(E, U, \{read\}) = TokenMandat(U) \geq EntityXACLMandat(E)$$

$$maccess\_check(E, U, Rm) = maccess\_check(E, U, \{write\}) \& maccess\_check(E, U, \{read\})$$

Модель ролевого контроля доступа

На основе понятий «пользователь», «группа пользователей» и механизма привилегий в операционной системе QP ОС реализуется ролевое управление доступом.

Пусть $Priveleges = \{p_1, p_2, \ldots, p_n\}$ - множество привилегий $Priveleges$.

Каждому субъекту $s \in Subjects$ в операционной системе QP ОС задаётся множество привилегий отношением $SubjectPrivileges$:

$$SubjectPrivileges: Subjects \leftrightarrow Privileges$$

На множестве всех маркеров вводится отношение доступных привилегий:

$TokenPrivileges: Tokens \leftrightarrow Privileges \times BOOL.$

При создании маркеру ставится в соответствие множество доступных привилегий

$privileges = ran((\{user\} \cup ran(\{user\} \lhd GroupUser)) \lhd SubjectPrivileges)$

следующим образом:

$TokenPrivileges = TokenPrivileges \cup \{token\} \times (privileges \times \{FALSE\})$.

При этом для использования привилегии p требуется ее «включение», т.е. отношение $TokenPrivileges$ должно содержать тройку $token \mapsto (p \mapsto TRUE)$

Проверка привилегий субъекта в системе выполняется на основе функции $privilege\_check$, принимающей значение «истина» в случае, если требуемое множество привилегий принадлежит набору привилегий пользователя:

$$privilege_{check(token,p)} = \begin{cases} true, \text{при } \{token\} \times (p \times \{TRUE\}) \subseteq TokenPrivileges \\ false, \text{при } \{token\} \times (p \times \{TRUE\}) \nsubseteq TokenPrivileges \end{cases}$$

Проверка привилегий осуществляется каждый раз при необходимости выполнить в системе действие, требующее определенных прав.

Зададим множество системных вызовов $A = \{a_1, a_2, \dots, a_m\}$, требующих от пользователя наличия особых прав, а также отношение требования наличия привилегии при выполнении системного вызова $ActionPriveleges: A \leftrightarrow Privileges$.

Выполнение системного вызова $system\_call(a, token)$ разрешается при истинном значении функции $privilege\_check$:

$$system\_call(a, token): privilege\_check(token, ActionPriveleges[a]) \& action(a).$$

Мандатный контроль доверия

Мандатный контроль доверия в операционной системе QP ОС применим к операциям, требующим наличия у субъекта привилегий. С использованием этого механизма обеспечивается целостность системы.

Уровень мандатного доверия определяется как обратная величина к уровню целостности системы.

Изначально система находится в состоянии высокой целостности. Операции, приводящие к нарушению целостности системы, требуют повышения доверия к процессу до определенного уровня. Чем важнее результат операции с точки зрения контроля целостности системы, тем выше требуемый уровень доверия к процессу.

Пусть $MandatoryTrustLevels = \{normal, medium, high, full\}$ - множество уровней мандатного доверия.

Каждому пользователю присваивается некоторый базовый уровень мандатного доверия из множества $MandatoryTrustLevels$. Уровни доверия всех пользователей описываются отношением $UserMandatoryTrustLevel: Users \rightarrow MandatoryTrustLevels$. Требуемый уровень мандатного доверия для получения привилегии описывается отношением $PrivilegeMandatoryLevel: Privileges \rightarrow MandatoryTrustLevels$.

Базовый уровень доверия может быть повышен в результате интерактивного запроса пользователя на повышение мандатного уровня доверия, моделируемого событием $rise\_confidence$. Для упрощения будем считать, что

$$rise\_confidence(user, privilege) =$$
$$\begin{cases} true, UserMandatoryTrustLevel(user) = \max(PrivilegeMandatoryLevel[privilege]), \\ \text{при положительном решении пользователя;} \\ false, \text{при отрицательном решении пользователя} \end{cases}$$

Функция проверки мандатного уровня доверия описывается следующим образом:

$$confidence\_check(user, privileges) =$$
$$\begin{cases} rise\_confidence(user, privilege), \\ \quad \text{при } \max(PrivilegeMandatoryLevel[privilege]) > UserMandatoryTrustLevel(user) \\ true, \\ \quad \text{при } \max(PrivilegeMandatoryLevel[privilege]) \leq UserMandatoryTrustLevel(user) \end{cases}$$

При включенном механизме контроля мандатного уровня доверия выполнение системного вызова описывается следующей функцией:

$$system\_call(a, token): confidence\_check(user, ActionPriveleges[a]) \&$$
$$privilege\_check(token, ActionPriveleges[a]) \,\&\, action(a)$$

О модели на формальном языке Event-B

Верификация разработанной модели проводится в инструментальной среде Rodin[2] с представлением модели на формальном языке Event-B. Данная среда позволяет строить модели

сложных систем, последовательно расширяя и уточняя их. Наша модель, в настоящее время состоит из трех частей, реализующих соответственно модели дискреционного, мандатного и ролевого контроля доступа. На рисунке 1 представлен состав компонент модели и их взаимосвязь:

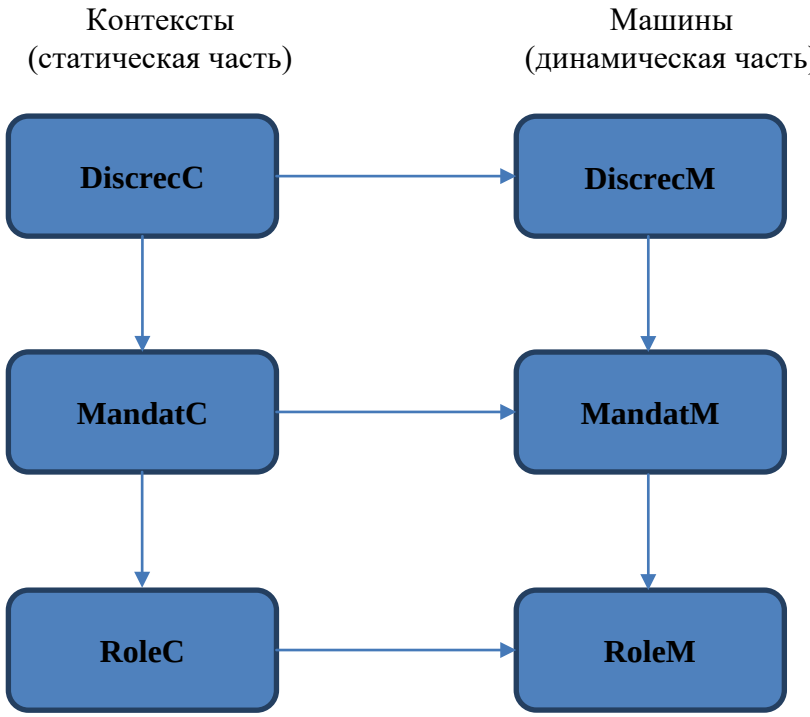

Рис.1 Диаграмма компонент системы в среде Rodin

Модели, описываемые в среде Rodin, состоят из контекстов и машин. Контексты содержат типы данных, константы и аксиомы, описывающие взаимосвязи между ними. Машина в среде Rodin представляет собой динамическую часть модели, в которой описываются состояния системы и события, реализующие функции перехода между состояниями, а также условия, на основе которых проводится верификация модели.

События, обрабатываемые моделью

На текущем этапе модель обрабатывает 36 событий, включающих в себя создание и завершение сессий пользователей, управление пользователями, управление группами пользователей, управление привилегиями и мандатным доступом пользователей, управление объектами, создаваемыми в системе, а также события изменения и проверки прав, привилегий и классификационных уровней. Привести описания всех событий в рамках данной работы не представляется возможным. Приведем примеры описания некоторых событий.

Если опустить охранные условия, то создание некоторого объекта в модели можно описать следующими действиями:

$$Entities' = Entities \cup \{object\}$$
$$Objects' = Objects \cup \{object\}$$
$$Type' = Type \cup object \mapsto object\_type$$
$$DACLs' = DACLs \cup \{dacl\}$$
$$ACEs'(dacl) = (Sids \times (ran(\{object\_type\} \lhd Rd) \times \{object\_type\})) \lhd ACEs(TokenDACL(token))$$
$$EntityDACL'(object) = dacl$$
$$EntityOwner'(object) = TokenUser(token)$$
$$Handle'(Parent(thread)) = Handle(Parent(thread)) \cup (\{n \mapsto object\} \times rights)$$
$$HandlesOfThread' := HandlesOfThread \cup \{thread \mapsto n\}$$
$$gettedRights' := gettedRights \cup (\{token \mapsto object\} \times rights)$$

При открытии объекта просто создается описатель для данного объекта:

$Handle'(Parent(thread)) \coloneqq Handle(Parent(thread)) \cup (\{n \mapsto object\} \times needed\_rights)$

$HandlesOfThread' \coloneqq HandlesOfThread \cup \{thread \mapsto n\}$

$gettedRights' \coloneqq gettedRights \cup (\{token \mapsto object\} \times needed\_rights)$

Выполнение операций над объектом моделируется событием access_check, охранные условия которого проверяют наличие необходимых прав:

$thread \in Entities$

$entity \in Entities$

$type = Type(entity)$

$rights \subseteq Rd[\{type\}]$

$\exists n \cdot n \in ran(\{thread\} \lhd HandlesOfThread) \land (\{n \mapsto entity\} \times rights) \subseteq Handle(Parent(thread))$

По завершению работы с объектом его можно закрыть, при этом удаляется его описатель:

$Handle'(Parent(thread)) = Handle(Parent(thread)) \setminus (\{n \mapsto entity\} \times Rd[\{type\}])$

$HandlesOfThread' = HandlesOfThread \setminus (\{thread\} \times dom(dom(Handle(Parent(thread))) \rhd \{entity\}))$

Если объект более не нужен, то он может быть удален:

$Entities' = Entities \setminus \{entity\}$

$Objects' = Objects \setminus \{entity\}$

$DACLs' = DACLs \setminus \{dacl\}$

$ACEs' = \{dacl\} \lhd ACEs$

$EntityDACL' = \{entity\} \lhd EntityDACL$

$Type' = \{entity\} \lhd Type$

$GetToken' = \{entity\} \lhd GetToken$

$EntityOwner' = \{entity\} \lhd EntityOwner$

$Handle'(Parent(thread)) = (dom(Handle(Parent(thread))) \rhd \{entity\}) \lhd Handle(Parent(thread))$

$HandlesOfThread' = HandlesOfThread \setminus (\{thread\} \times dom(dom(Handle(Parent(thread))) \rhd \{entity\}))$

$gettedRights' = (Tokens \times \{entity\}) \lhd gettedRights$

Заключение.

Предложенная формальная модель контроля доступа операционной системы QP ОС позволяет проводить исследование безопасности ОС с целью подтверждения корректности применяемых подходов к разграничению доступа. В настоящее время проводится верификация построенной модели.

Литература

1.Операционная система QP ОС. URL: https://cryptosoft.ru/qpos.html

2. Rodin Handbook. http://handbook.event-b.org

3. Афонин А.Ю. Механизмы безопасности QP ОС. // Сборник докладов 8-го межведомственного научно-практического семинара «Системы и средства защиты информации». г. Пенза, № 619-дсп от 2017 г. – с.176-180.

4. Егоров В.Ю. Мандатное управление доверием в QP ОС. // Сборник докладов 9-го межведомственного научно-практического семинара «Системы и средства защиты информации». г. Пенза, № 400-17-дсп от 2017 г. – с. 294-296.

5. Девянин П.Н. Модели безопасности компьютерных систем. Управление доступом и информационными потоками. Учебное пособие для вузов. 2-е изд., испр. и доп. М.: Горячая линия – Телеком, 2013. 338 с.

6. А.Ю. Афонин, Формальная модель механизма контроля доступа QP ОС, ООО НТП «Криптософт», г. Пенза, №555-19-ДСП от 12.12.2019г.