



О построении формальной модели управления доступом QR ОС

НТП «Криптософт», г. Пенза

М.А. Алехина,

Васин А.В.,

Афонин А.Ю.,

Егоров В.Ю.

Основные элементы модели



Множество субъектов системы: $Subjects$

Множество пользователей: $Users \subset Subjects$

Множество групп пользователей: $Groups \subset Subjects$

Отношение членства в группах: $GroupUser: Users \leftrightarrow Groups$

Множество дескрипторов (идентификаторов) безопасности: $Sids$

Отношение принадлежности дескриптора субъекту $SubjectSid: Subjects \rightarrow Sids$

Среди множества predetermined групп выделим группу администраторов безопасности: $Admins \in Groups, GroupUser \triangleright \{Admins\} \neq \emptyset$

Множество объектов: $Objects$

Множество объектов-контейнеров: $Containers$

$$Entities = Objects \cup Containers$$

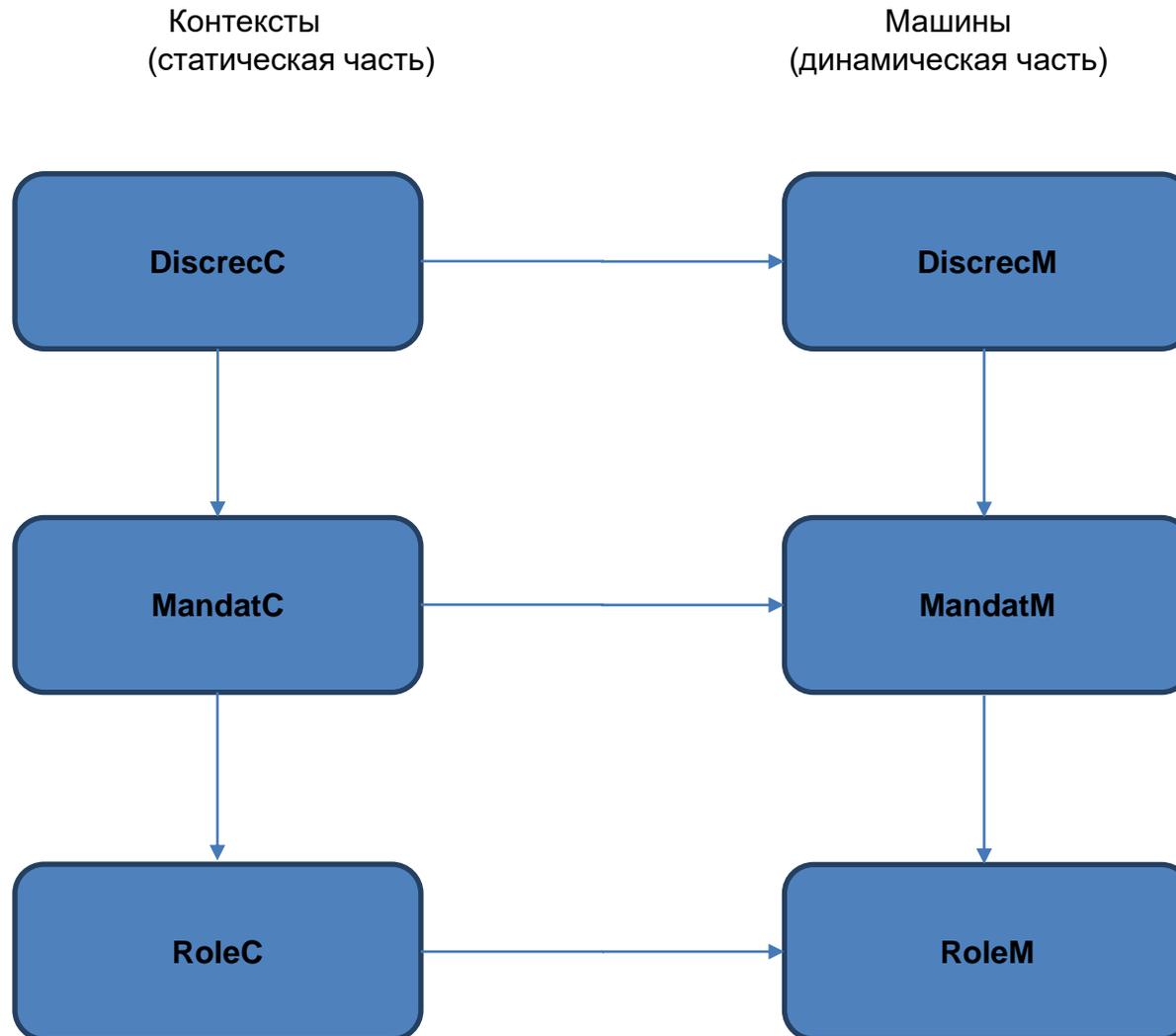
Множество типов: $Types$, каждый объект имеет тип: $Type: Entities \rightarrow Types$

Множество прав доступа: $AccessRights$

Каждому типу прав применяемых к объекту, $Rd: Types \leftrightarrow AccessRights$

Владелец объекта определен отношением $EntityOwner: Entities \rightarrow Sids$

Схема модели в системе Rodin



Модель дискреционного контроля доступа



Сессии пользователей, выполнивших процедуру входа: *Sessions*

Множество маркеров доступа: *Tokens*

Пользователь, сессия и маркер связаны отношениями

SessionUser: Users \rightarrow *Sessions* и *SessionToken: Sessions* \rightarrow *Tokens*

Каждому процессу/нити соответствует маркер: *GetToken: Entities* \rightarrow *Tokens*.

Каждый маркер связан с дескриптором безопасности:

TokenUser: TokenUser: Tokens \rightarrow *Sids*

Маркер безопасности содержит идентификаторы разрешенных/запрещенных групп:

TokenRules: TokenRules: Tokens \leftrightarrow *Sids* \times *BOOL*

Маркер ограниченным и олицетворенным:

TokenIsRestricted: Tokens \rightarrow *BOOL* и *TokenIsImpersonated: Tokens* \rightarrow *BOOL*.

Множество списков контроля доступа: *DACLs*

Отношение сущностей и списков доступа объектов: *EntityDAcl: Entities* \rightarrow *DACLs*

Отношение правил доступа *ACEs*:

$ACEs: DAcls \rightarrow (Sids \times (AccessRights \times Types)) \rightarrow \mathbb{P}(RFS)$

Описатели нити: *HandlesOfThread: Containers* \leftrightarrow $\mathbb{N}1$

Права, полученные нитью для объекта:

$Handle: Containers \rightarrow (\mathbb{N}1 \times Entities \leftrightarrow AccessRights)$.

Модель дискреционного контроля доступа



Пусть $E \in Entities, token \in Tokens, rd \subseteq AccessRights$)

В операционной системе контроль доступа осуществляет функция $access_check(E, token, rd)$:

Определение разрешенных прав:

1. $t = Type(E), r = Rd[t]$
2. $sid = TokenUser(token)$ и $G_{sid}^0 = dom(ran(\{token\} \triangleleft TokenRules) \triangleright \{TRUE\})$.
3. $S_o = \{sid\} \cup G_{sid}^0$
4. $dacl = EntityDACL(E)$ и $Rules_0 = S_o \times (r \times \{t\}) \triangleleft ACEs(dacl)$
5. $Allowed = dom(ran(dom(Rules \triangleright FlagSetsWithTRUE)))$
6. $G_{sid}^1 = dom(ran(\{token\} \triangleleft TokenRules))$.
7. $Rules_1 = G_{sid}^1 \times (r \times \{t\}) \triangleleft ACEs(dacl)$.
8. $Denied = dom(ran(dom(Rules \triangleright FlagSetsWithFALSE)))$

Множество прав доступа субъекта к объекту: $Allowed \setminus Denied$.

Тогда имеем:

$$access_check(E, token, rd) = \begin{cases} true, \text{ при } EntityOwner(E) = TokenUser(token) \\ true, \text{ при } rd \subseteq Allowed \setminus Denied \\ false, \text{ в остальных случаях} \end{cases}$$

Модель мандатного контроля доступа



Пусть $(B = \{0,1, \dots, l\}, \leq)$, $l \leq 15$ - вполне упорядоченное множество

Иерархические категории: $B_k \subseteq B$, $k \leq 8$

Неиерархические категории: $C_m \subseteq \{0,1\}$, $m \leq 16$

Пусть $\Pi = B_1 \times B_2 \times \dots \times B_k \times \{0,1\}^m$, $\alpha, \beta \in \Pi$ - классификационные уровни (КУ).

Каждому маркеру и каждому объекту приписан КУ.

Определение: α из Π предшествует набору β , если каждая компонента α не больше соответствующей компоненты β (обозначение $\alpha \leq \beta$).

Определение: Два набора из множества Π сравнимы, если один из них предшествует другому. В противном случае наборы α и β несравнимы.

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.



Модель мандатного контроля доступа

Пусть $Rm = \{write, read\}$.

Пользователю присваивается множество КУ субъекта:

$UserMandatRule: Users \rightarrow \mathbb{P}1(\Pi)$.

При создании маркера для $U \in Users$ маркеру ставится в соответствие КУ из множества $UserMandatRule(U)$: $TokenMandat: Users \rightarrow \Pi$

Каждому объекту присваивается КУ: $EntityXACLMandat: Entities \rightarrow \Pi$.

Проверка прав доступа $maccess_check(E, U, Rm)$ - функция трёх переменных (объекта, пользователя и запрашиваемого права мандатного доступа), значения которой задаются следующим образом:

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

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

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

Модель ролевого контроля доступа



Ролевое управление доступом реализуется на основе понятий «пользователь», «группа пользователей» и механизма привилегий.

Набор привилегии - конечное определённое множество

$$Privileges = \{p_1, p_2, \dots, p_n\}.$$

Для каждого субъекта $s \in Subjects$ задаётся множество привилегий при помощи отношения $SubjectPrivileges: Subjects \leftrightarrow Privileges$

Отношение доступных привилегий маркера:

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

При создании маркера $token$ отношение $TokenPrivileges$ меняется следующим образом:

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

$$\text{где } privileges = ran(\{user\} \cup ran(\{user\} \triangleleft GroupUser))$$

$$\triangleleft SubjectPrivileges)$$

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

Модель ролевого контроля доступа



Проверка привилегий субъекта в системе выполняется на основе функции *privilege_check*, принимающей значение «истина» в случае, если требуемое множество привилегий принадлежит набору привилегий пользователя:

$$\begin{aligned} & \textit{privilege_check}(\textit{token}, p) = \\ & = \begin{cases} \textit{true}, & \text{при } \{\textit{token}\} \times (p \times \{\textit{TRUE}\}) \subseteq \textit{TokenPrivileges} \\ \textit{false}, & \text{при } \{\textit{token}\} \times (p \times \{\textit{TRUE}\}) \not\subseteq \textit{TokenPrivileges} \end{cases} \end{aligned}$$

Пусть $A \in \{a_1, a_2, \dots, a_n\}$ - множество системных вызовов, требующих от пользователя наличия особых прав.

Тогда отношение требования наличия привилегии:

ActionPrivileges: $A \leftrightarrow \textit{Privileges}$.

Выполнение системного *system_call(a, token)* вызова разрешается при истинном значении функции *privilege_check*:

system_call(a, token): *privilege_check(token, ActionPrivileges[a])* & *action(a)*.

Мандатный контроль доверия



Уровни мандатного доверия:

MandatoryTrustLevels = {normal, medium, high, full}

Мандатный уровень доверия пользователя:

UserMandatoryTrustLevel: Users → MandatoryTrustLevels

Уровни мандатного доверия привилегий:

PrivilegeMandatoryLevel: Privileges → MandatoryTrustLevels

Мандатный контроль доверия



Базовый уровень доверия может быть повышен в результате интерактивного запроса:

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

Функция проверки мандатного уровня доверия описывается следующим образом:

$$\begin{cases} \text{confidence_check}(user, privileges) = \\ \text{rise_confidence}(user, privilege), \\ \text{при } \max(PrivilegeMandatoryLevel[privilege]) > UserMandatoryTrustLevel(user) \\ \text{true,} \\ \text{при } \max(PrivilegeMandatoryLevel[privilege]) \leq UserMandatoryTrustLevel(user) \end{cases}$$

СИСТЕМНЫЙ ВЫЗОВ:

```
system_call(a, token): confidence_check(user, ActionPrivileges[a]) &  
privilege_check(token, ActionPrivileges[a]) & action(a)
```

Операции над объектами. Создание объекта



$$Entities' = Entities \cup \{object\}$$

$$Objects' = Objects \cup \{object\}$$

$$Type' = Type \cup object \mapsto object_type$$

$$DACLS' = DACLS \cup \{dacl\}$$

$$ACEs'(dacl) = (Sids \times (ran(\{object_type\} \triangleleft Rd) \times \{object_type\})) \triangleleft ACEs(TokenDACL(token))$$

$$EntityDACL'(object) = dacl$$

$$EntityOwner'(object) = TokenUser(token)$$

$$Handle'(Parent(thread)) = Handle(Parent(thread)) \cup (\{n \mapsto object\} \times rights)$$

$$HandlesOfThread' := HandlesOfThread \cup \{thread \mapsto n\}$$

$$gettedRights' := gettedRights \cup (\{token \mapsto object\} \times rights)$$

Открытие объекта

$$Handle'(Parent(thread)) := Handle(Parent(thread)) \cup (\{n \mapsto object\} \times needed_rights)$$

$$HandlesOfThread' := HandlesOfThread \cup \{thread \mapsto n\}$$

$$gettedRights' := gettedRights \cup (\{token \mapsto object\} \times needed_rights)$$

Выполнение операции над объектом



Выполнение функции `check_access` при выполнении действий реализуется одноименным событием со следующим набором охранных условий:

$thread \in Entities$

$entity \in Entities$

$type = Type(entity)$

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

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

Заккрытие описателя

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

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

Удаление объекта

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

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

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

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

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

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

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

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

$Handle'(Parent(thread)) = (dom(Handle(Parent(thread))) \triangleright \{entity\}) \triangleleft Handle(Parent(thread))$

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

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



Модель в системе Rodin



```
MandatC  RoleC  DiscrecC  MandatM  DiscrecM  RoleM X
111         user_communio→SE_CREATE_TOKEN,
112         all_communio→SE_CREATE_TOKEN} >TODO:Уточнить список привилегий для Субъектов
113     ◦ act44: UserMandatoryTrustLevel={local_system→4,admin→3} >
114     ◦ act45: TokenPrivileges=∅ >
115     END
116
117 ◦ create_user_session: extended ordinary >
118   REFINES
119   ◦ create_user_session
120   ANY
121   ◦ new_session >
122   ◦ user >
123   ◦ sid >
124   ◦ token >
125   ◦ session >
126   ◦ rules >
127   ◦ session_mandat >
128   ◦ privileges >
129   WHERE
130   ◦ grd1: new_session∈Curr_Union not theorem >Новая сессия не используется в системе
131   ◦ grd2: session∈Sessions not theorem >активная сессия, которая создает новую сессию (пока не используется)
132   ◦ grd3: user∈Users ∧ user∈dom(GroupUser) ∧ user∈dom(SubjectSid) not theorem >Определяем пользоват
133   ◦ grd10: sid=SubjectSid(user) ∧ sid∈dom(SubjectDACL) not theorem >Определяем сид этого пользователя
134   ◦ grd4: token∈Tokens\Curr_Tokens not theorem >выбираем маркер, не используемый в системе
135   ◦ grd7: rules={token}↔Sids×BOOL not theorem >Локальный тип, чтобы определить правила нового токена (тип должен
136   ◦ grd6: ∀g.g∈Groups ∧ g∈dom(SubjectSid) ∧ user→g∈GroupUser ⇒ token→(SubjectSid(g)→TRUE)∈rules
137   ◦ grd11: user∈dom(UserMandatRule) not theorem >
138   ◦ grd12: session_mandat∈UserMandatRule(user) not theorem >
139   ◦ grd13: privileges=ran({{user}uran({user}←GroupUser)}←SubjectPrivileges) not theorem >
140   THEN
141   ◦ act1: Sessions=Sessions∪{new_session} >добавляем новую сессию
142   ◦ act2: Curr_Union=Curr_Union∪{new_session} >Расширяем множество использованных элементов
143   ◦ act3: SessionUser(new_session)=user >Связываем сессию с пользователем
144   ◦ act4: SessionOwner(new_session) = user >Сохраняем владельца сесии
145   ◦ act5: SessionParent(new_session) = system_session >Сохраняем родительскую сессию
146   ◦ act6: Curr_Tokens=Curr_Tokens∪{token} >Добавляем новый токен
147   ◦ act7: SessionToken(new_session)=token >Связываем токен с сессией
148   ◦ act8: TokenUser=TokenUser∪{token→SubjectSid(user)} >Привязываем к токenu сид пользователя
149   ◦ act9: TokenGroups=TokenGroups ∪ rules >добавляем сиды всех групп пользователя к токenu
150   ◦ act14: TokenDACL(token)=SubjectDACL(sid) >указываем DACL по умолчанию
151   ◦ act15: TokenIsRestricted(token)=FALSE >здесь всегда неограниченный маркер
152   ◦ act16: TokenIsImpersonated(token)=FALSE >здесь всегда неолицетворенный маркер
153   ◦ act17: TokenMandat(token)=session_mandat >
154   ◦ act18: TokenPrivileges=TokenPrivileges∪({token}×(privileges×{FALSE})) >
155   END
156
157 ◦ delete_session: extended ordinary >
```

Модель в системе Rodin



```
MandatC  RoleC  DiscrecC  MandatM  DiscrecM  RoleM x
1202  END
1203
1204  open_entity:  extended ordinary >
1205  REFINES
1206  - open_entity
1207  ANY
1208  - thread >
1209  - entity >
1210  - owner >
1211  - type >
1212  - dacl >
1213  - token >
1214  - rights > >запрашиваемые права
1215  - all_rights >
1216  - allowed >
1217  - needed_rights >
1218  - n >
1219  WHERE
1220  - grd1:  thread=Entities ^
1221           threadedom(Type) ^ Type(thread)=T_THREAD ^
1222           threadedom(GetToken) ^ GetToken(thread)edom(TokenUser) ^
1223           threadedom(Parent) ^ Parent(thread)edom(Handle) not theorem >
1224  - grd2:  entity=Entities ^ entityedom(Type) ^ entityedom(EntityDACL) ^ entityedom(EntityOwner) not theorem >
1225  - grd14: n=N1 not theorem >
1226  - grd3:  type=Type(entity) not theorem >
1227  - grd4:  owner=EntityOwner(entity) not theorem >
1228  - grd5:  token=GetToken(thread) ^ tokenedom(TokenUser) not theorem >
1229  - grd6:  dacl=EntityDACL(entity) ^ dacledom(ACEs) not theorem >
1230  - grd7:  all_rights=Rd[{{type}}] not theorem >
1231  - grd8:  rights=all_rights not theorem >
1232  - grd15: ( n#ran({thread}#HandlesOfThread)^n#entityedom( Handle(Parent(thread)) ) ) v
1233           ( n#ran(HandlesOfThread) not theorem >
1234  - grd9:  needed_rights= rights#ran( {n#entity}#Handle(Parent(thread)) ) not theorem >
1235  - grd12: allowed= ran(dom((( {{TokenUser(token)}#edom(ran({token}#TokenGroups)#TRUE})) x(all_rights#{{type}}) # ACes(dacl)) # FlagSet
1236           ran(dom((( dom(ran({token}#TokenGroups)) x(all_rights#{{type}}) # ACes(dacl)) # FlagSetsWithFALSE)) not theorem >
1237  - OwnerCanReadDescriptor: owner=TokenUser(token) v (needed_rights#{{type}}#allowed ^ {{2}}#needed_rights ) not theorem >
1238  - grd18: tokenedom(TokenMandat) not theorem >
1239  - grd19: entityedom(EntityXACL_Mandat) not theorem >
1240  - grd16: ran({{type}}#GENERIC_R) n needed_rights # 0 => TokenMandat(token)#EntityXACL_Mandat(entity)#MORE not theorem >
1241  - grd17: ran({{type}}#GENERIC_W) n needed_rights # 0 => TokenMandat(token)#EntityXACL_Mandat(entity)#LESS not theorem >
1242  THEN
1243  - act1:  Handle(Parent(thread))=Handle(Parent(thread)) u ({{n#entity}}#needed_rights) >
1244  - act2:  HandlesOfThread=HandlesOfThreadu{thread#n} >
1245  - gr:  gettedRights=gettedRightsu({token#entity}#needed_rights) >
1246  END
1247
1248  close_entity:  extended ordinary >
```

Модель в системе Rodin

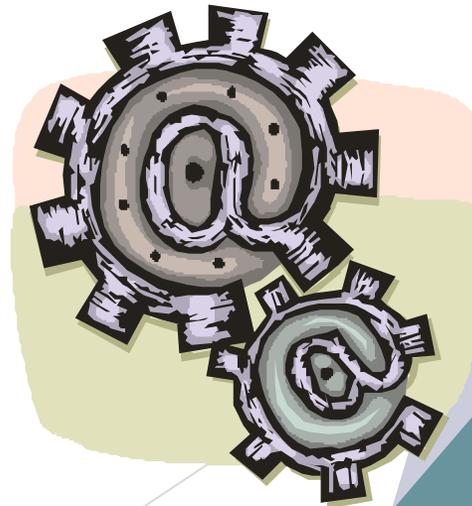


```
MandatC  RoleC  DiscrecC  MandatM  DiscrecM  RoleM x
1362     ◦  ChangedRules:   $\forall p, b1, p \in \text{Privileges} \wedge b1 \in \text{BOOL} \wedge \text{token} \Rightarrow (\exists b2 \cdot \text{token} \Rightarrow (p \Rightarrow b2) \in \wedge$ 
1363     ◦  SingleRuleCondition:   $\forall p, b1, b2 \cdot p \in \text{Privileges} \wedge b1 \in \text{BOOL} \wedge b2 \in \text{BOOL} \wedge \text{token} \Rightarrow (p \Rightarrow b1) \in \text{rules} \wedge \text{token} \Rightarrow (p \Rightarrow b2) \in \wedge$ 
1364     THEN
1365     ◦  act1:  TokenPrivileges = {{token}  $\Leftarrow$  TokenPrivileges} urules >
1366     END
1367
1368     ◦  change_owner:  extended ordinary >
1369     REFINES
1370     ◦  change_owner
1371     ANY
1372     ◦  entity >
1373     ◦  new_owner >
1374     ◦  thread >
1375     ◦  user >
1376     ◦  token >
1377     ◦  owner >
1378     WHERE
1379     ◦  NewOwner:  new_owner  $\in$  Users  $\wedge$  new_owner  $\in$  dom(SubjectSid) not theorem >
1380     ◦  Thread:  thread  $\in$  Entities  $\wedge$  thread  $\in$  dom(Type)  $\wedge$  Type(thread) = T_THREAD  $\wedge$  thread  $\in$  dom(GetToken)  $\wedge$ 
1381     ◦  User:  user  $\Rightarrow$  TokenUser(GetToken(thread))  $\in$  SubjectSid not theorem >
1382     ◦  grd5:  token = GetToken(thread)  $\wedge$  token  $\in$  dom(TokenUser) not theorem >
1383     ◦  grd16:  entity  $\in$  Entities  $\wedge$  entity  $\in$  dom(EntityOwner) not theorem >
1384     ◦  grd17:  owner  $\neq$  SubjectSid(new_owner) not theorem >
1385     ◦  OwnerOrCanWriteDescriptor:  owner = TokenUser(token)  $\vee$  (  $\exists n \cdot n \in \text{ran}(\{\text{thread}\} \Leftarrow \text{HandlesOfThread}) \wedge n$ 
1386     ◦  grd4:  owner = EntityOwner(entity) not theorem >
1387     THEN
1388     ◦  act12:  EntityOwner(entity) = SubjectSid(new_owner) >
1389     END
1390
1391     ◦  take_ownership:  extended ordinary >
1392     REFINES
1393     ◦  take_ownership
1394     ANY
1395     ◦  entity >
1396     ◦  thread >
1397     ◦  user >
1398     ◦  token >
1399     ◦  owner >
1400     WHERE
1401     ◦  Thread:  thread  $\in$  Entities  $\wedge$  thread  $\in$  dom(Type)  $\wedge$  Type(thread) = T_THREAD  $\wedge$  thread  $\in$  dom(GetToken)  $\wedge$ 
1402     ◦  grd5:  token = GetToken(thread)  $\wedge$  token  $\in$  dom(TokenUser) not theorem >
1403     ◦  User:  user = TokenUser(token) not theorem >
1404     ◦  grd16:  entity  $\in$  Entities  $\wedge$  entity  $\in$  dom(EntityOwner) not theorem >
1405     ◦  grd4:  owner  $\in$  ran(EntityOwner)  $\wedge$  owner = EntityOwner(entity) not theorem >
1406     ◦  grd17:  owner  $\neq$  user not theorem >
1407     ◦  BecomeOwnerPrivilege:  token  $\Rightarrow$  (SE_TAKE_OWNERSHIP  $\Rightarrow$  TRUE)  $\in$  TokenPrivileges not theorem >
1408     THEN
```

Дальнейшее развитие



- ▶ Завершение верификации формальной модели в системе Rodin;
- ▶ уточнение модели;
- ▶ покрытие кода формальной моделью.





Спасибо за внимание!

qpos@cryptosoft.ru