Министерство образования и науки Российской Федерации

Санкт-Петербургский политехнический университет Петра Великого

—

Институт прикладной математики и механики

**Кафедра «Информационная безопасность компьютерных систем»**

**ЛАБОРАТОРНАЯ РАБОТА №6**

**Модель действий/сущностей**

по дисциплине «Модели безопасности компьютерных систем»

Выполнил

Студент гр.33636/1 Малинко А.В.

Проверил

Преподаватель Овасапян Т.Д.

Санкт-Петербург

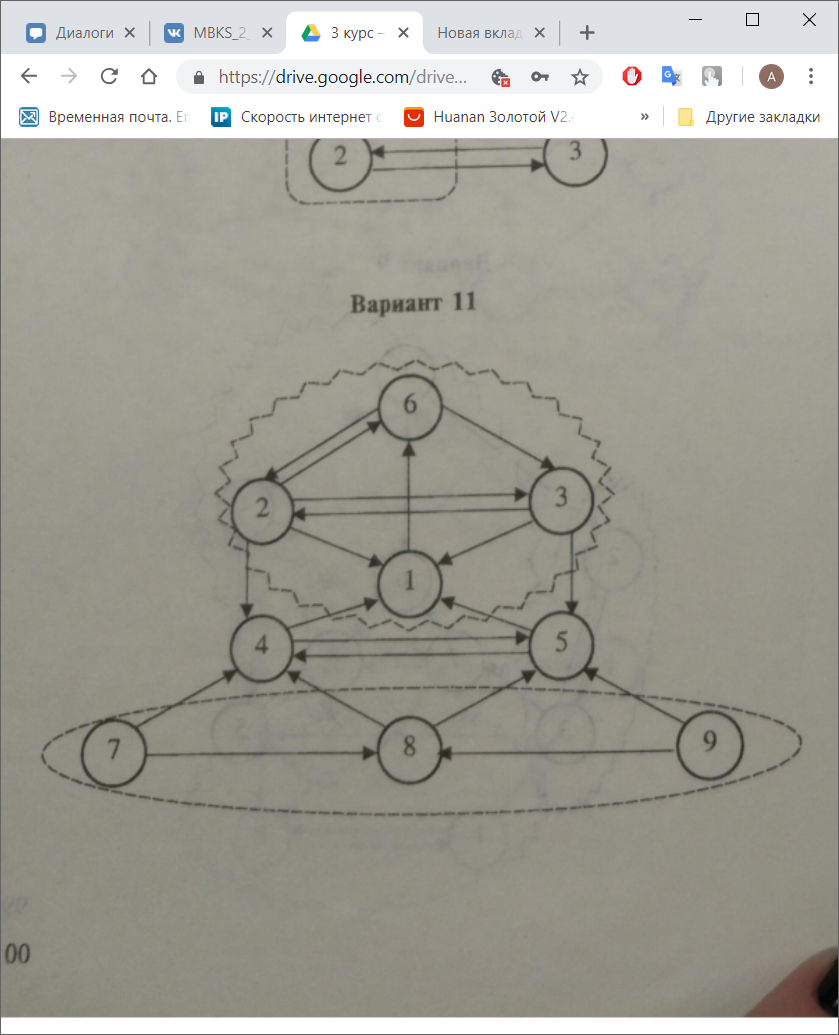
2018

**Цель работы**

Исследовать защитные характеристики и особенности дискреционной модели действий/сущностей.

**Задачи**

Написать программу на языке Prolog, описывающую модель действий/сущностей:



**Ход работы**

Была реализована программа на языке Prolog, описывающая данную модель. Программа поддерживает следующие команды:

**help** – выводит список всех команд.

**command** – запускает меню, в котором можно вызывать команды по номерам, не прописываю целиком.

**create\_entity** – создаёт новую сущность.

**create\_child\_entity** – создаёт дочернюю сущность.

**destroy\_entity** – удаляет сущность.

**change\_max\_active** – меняет активный потенциал.

**change\_max\_passive** – меняет пассивный потенциал.

**enter\_static\_right** – добавляет статическое право от root’а.

**check\_right** – проверяет статическое право.

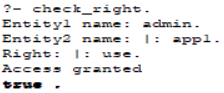
**grant** – передаёт статическое право.

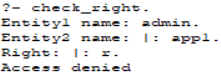
**revoke** –забирает статическое право.

**delegate** – передаёт динамическое право.

**abrogate** – забирает динамическое право.

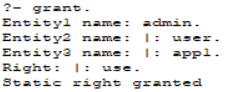
Проверим модель на безопасность:

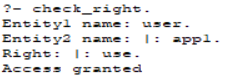




Доступ по существующим правам был предоставлен, по несуществующим нет.

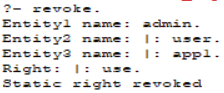
Проверим работу функции grant:

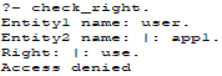




Доступ был передан.

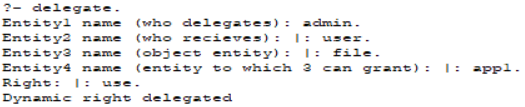
Проверим функцию revoke:

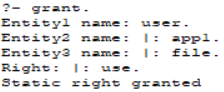


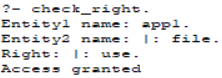


В доступе было отказано.

Функцию delegate:

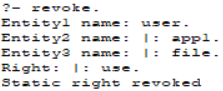


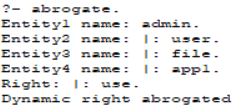


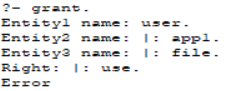


Право доступа было передано и доступ был получен.

Проверим последнюю функцию abrogate:







Право не было передано, так как право grant было забрано.

В данной модели невозможно реализовать атаку троянского коня, так как каждая сущность контролирует предоставление прав. Если нужно передать какое-то право другой сущности, выставляется право grant к одному праву, а не ко всей сущности, а значит права, к которым нельзя предоставлять доступ останутся секретными.

По той же причине невозможна атака передачи всех прав. Поскольку, сущность сама решает какие права она может передавать, а какие нет, а также кому.

Проблема отсутствия контроля распространения прав также решена. Теперь, если сущности нужно предоставить другой сущности право, без возможности передачи его дальше, она использует функцию grant и право не ибудет передано дальше

**Контрольные вопросы**

**В чем отличие модели TG от модели действий/сущностей?**

В МДС, в отличие от модели TG, были введены следующие дополнения:

предложены дополнительные правила контроля доступа;

предложены дополнительные административные права;

выделены отдельно административные права;

усилен контроль администрирования прав;

определены два графа: для представления доступа и для распространения прав.

**Какие проблемы модели TG решены в модели действий/сущностей?**

Проблема отсутствия значимости прав, проблем отсутствия контроля распространения прав.

**С какой целью введено правило преобразования динамического графа?**

Для контроля косвенной авторизации и распространения прав.

**Что называют пассивным потенциалом сущности?**

Пассивный потенциал сущности – статическое действие высшего приоритета.

**С какой целью в описании МДС введен динамический граф?**

Для решения проблемы значимости административных прав.

**Вывод**

В результате работы была изучена и реализована модель действий/сущностей. Исследованы уязвимости этой модели и основываясь на них были сделаны выводы о невозможности реализации атак, доступных в матрица получения и передачи прав.

**Приложение**

:-dynamic entity/3.

:-dynamic d\_access/2.

:-dynamic s\_access/3.

:-dynamic owner/2.

:-dynamic grant/4.

entity(root, cd, no).

entity(admin, cd, no).

entity(user, cd, use).

entity(app1, upd, upd).

entity(app2, r, cd).

entity(file, no, cd).

entity(printer, no, use).

owner(root, admin).

owner(root, user).

owner(admin, app1).

owner(admin, printer).

owner(user, app2).

owner(user, file).

d\_access(admin, user).

d\_access(admin, file).

s\_access(admin, app1, use).

grant(admin, app1, use, user).

s\_right(use, 1).

s\_right(r, 2).

s\_right(upd, 3).

s\_right(cd, 4).

help:-

write('create\_entity'), nl,

write('create\_child\_entity'), nl,

write('destroy\_entity'), nl,

write('change\_max\_active'), nl,

write('change\_max\_passive'), nl,

write('enter\_static\_right'), nl,

write('check\_right'), nl,

write('grant'), nl,

write('revoke'), nl,

write('delegate'), nl,

write('abrogate'), nl.

command:-

write('0 - create entity'), nl,

write('1 - create child entity'), nl,

write('2 - destroy entity'), nl,

write('3 - change max active right'), nl,

write('4 - change max passive right'), nl,

write('5 - enter static right'), nl,

write('6 - check right'), nl,

write('7 - grant right'), nl,

write('8 - revoke right'), nl,

write('9 - delegate right'), nl,

write('10 - abrogate right'), nl,

write('14 - display the tip again'), nl,

write('15 - quit'), nl,

read(N), redirect(N).

get\_new:-

read(N), redirect(N).

redirect(N):-

N = 0, create\_entity, nl, get\_new;

N = 1, create\_child\_entity, nl, get\_new;

N = 2, destroy\_entity, nl, get\_new;

N = 3, change\_max\_active, nl, get\_new;

N = 4, change\_max\_passive, nl, get\_new;

N = 5, enter\_static\_right, nl, get\_new;

N = 6, check\_right, nl, get\_new;

N = 7, grant, nl, get\_new;

N = 8, revoke, nl, get\_new;

N = 9, delegate, nl, get\_new;

N = 10, abrogate, nl, get\_new;

N = 11, nl, command;

N = 12;

write('Error'), nl.

create\_entity:-

write('Name: '), read(Ent\_name),

write('Active right: '), read(Act\_right),

write('Passive right: '), read(Pas\_right),

(

entity(Ent\_name, Act\_right, Pas\_right), write('Error'), nl;

assert(entity(Ent\_name, Act\_right, Pas\_right)), write('Entity created'), nl

).

create\_child\_entity:-

write('Parent name: '), read(Par\_name),

write('Child name: '), read(Chil\_name),

(

not(entity(Par\_name, \_, \_)), write('Error'), nl;

entity(Chil\_name, \_, \_), write('Error'), nl;

entity(Par\_name, Act\_right, Pas\_right),

assert(entity(Chil\_name, Act\_right, Pas\_right)),

assert(owner(Par\_name, Chil\_name)),

assert(d\_access(Par\_name, Chil\_name)), write('Child created'), nl

).

destroy\_entity:-

write('Name: '), read(Ent\_name),

(

not(entity(Ent\_name, \_, \_)), write('Error'), nl;

retract(entity(Ent\_name, \_, \_)),

retract(owner(\_, Ent\_name)),

retractall(s\_access(Ent\_name, \_, \_)),

retractall(s\_access(\_, Ent\_name, \_)),

retractall(d\_access(Ent\_name, \_)),

retractall(d\_access(\_, Ent\_name)),

retractall(grant(Ent\_name, \_, \_, \_)),

retractall(grant(\_, Ent\_name, \_, \_)),

retractall(grant(\_, \_, \_, Ent\_name)), write('Entity destroyed')

).

change\_max\_active:-

write('Entity name: '), read(Ent\_name),

write('Right: '), read(Right),

(

not(entity(Ent\_name, \_, \_)), write('Error'), nl;

not(s\_right(Right, \_)), write('Error'), nl;

entity(Ent\_name, Act\_right, Pas\_right),

retract(entity(Ent\_name, Act\_right, Pas\_right)),

assert(entity(Ent\_name, Right, Pas\_right)), write('Active right entered'), nl

).

change\_max\_passive:-

write('Entity name: '), read(Ent\_name),

write('Right: '), read(Right),

(

not(entity(Ent\_name, \_, \_)), write('Error'), nl;

not(s\_right(Right, \_)), write('Error'), nl;

entity(Ent\_name, Act\_right, Pas\_right),

retract(entity(Ent\_name, Act\_right, Pas\_right)),

assert(entity(Ent\_name, Act\_right, Right)), write('Passive right entered'), nl

).

enter\_static\_right:-

write('Entity1 name: '), read(Ent1\_name),

write('Entity2 name: '), read(Ent2\_name),

write('Owner name: '), read(Own\_name),

write('Right: '), read(Right),

(

not(entity(Ent1\_name, \_, \_)), write('Error'), nl;

not(entity(Ent2\_name, \_, \_)), write('Error'), nl;

not(Own\_name == root), write('Error'), nl;

not(s\_right(Right, \_)), write('Error'), nl;

s\_right(Right, Right\_prior),

entity(Ent1\_name, Act\_pot, \_),

entity(Ent2\_name, \_, Pas\_pot),

s\_right(Act\_pot, Act\_prior),

s\_right(Pas\_pot, Pas\_prior),

(

Right\_prior < Act\_prior;

Right\_prior < Pas\_prior

), write('Error'), nl;

s\_access(Ent1\_name, Ent2\_name, Right), write('Error'), nl;

assert(s\_access(Ent1\_name, Ent2\_name, Right)), write('Static right entered'), nl

).

check\_right:-

write('Entity1 name: '), read(Ent1\_name),

write('Entity2 name: '), read(Ent2\_name),

write('Right: '), read(Right),

(

s\_access(Ent1\_name, Ent2\_name, Right), write('Access granted'), nl;

write('Access denied'), nl

).

grant:-

write('Entity1 name: '), read(Ent1\_name),

write('Entity2\_name: '), read(Ent2\_name),

write('Entity3 name: '), read(Ent3\_name),

write('Right: '), read(Right),

(

not(entity(Ent1\_name, \_, \_)), write('Error'), nl;

not(entity(Ent2\_name, \_, \_)), write('Error'), nl;

not(entity(Ent3\_name, \_, \_)), write('Error'), nl;

not(s\_right(Right, \_)), write('Error'), nl;

not(s\_access(Ent1\_name, Ent3\_name, Right)), write('Error'), nl;

not(grant(Ent1\_name, Ent3\_name, Right, Ent2\_name)), write('Error'), nl;

s\_access(Ent2\_name, Ent3\_name, Right), write('Error'), nl;

s\_right(Right, Right\_prior),

entity(Ent2\_name, Act\_pot, \_),

s\_right(Act\_pot, Act\_prior),

Right\_prior > Act\_prior, write('Error'), nl;

assert(s\_access(Ent2\_name, Ent3\_name, Right)), write('Static right granted'), nl

).

revoke:-

write('Entity1 name: '), read(Ent1\_name),

write('Entity2 name: '), read(Ent2\_name),

write('Entity3 name: '), read(Ent3\_name),

write('Right: '), read(Right),

(

not(entity(Ent1\_name, \_, \_)), write('Error'), nl;

not(entity(Ent2\_name, \_, \_)), write('Error'), nl;

not(entity(Ent3\_name, \_, \_)), write('Error'), nl;

not(s\_right(Right, \_)), write('Error'), nl;

not(grant(Ent1\_name, Ent3\_name, Right, Ent2\_name)), write('Error'), nl;

not(s\_access(Ent2\_name, Ent3\_name, Right)), write('Error'), nl;

retract(s\_access(Ent2\_name, Ent3\_name, Right)), write('Static right revoked'), nl

).

delegate:-

write('Entity1 name (who delegates): '), read(Ent1\_name),

write('Entity2 name (who recieves): '), read(Ent2\_name),

write('Entity3 name (object entity): '), read(Ent3\_name),

write('Entity4 name (entity to which 3 can grant): '), read(Ent4\_name),

write('Right: '), read(Right),

(

not(entity(Ent1\_name, \_, \_)), write('Error'), nl;

not(entity(Ent2\_name, \_, \_)), write('Error'), nl;

not(entity(Ent3\_name, \_, \_)), write('Error'), nl;

not(entity(Ent4\_name, \_, \_)), write('Error'), nl;

not(s\_right(Right, \_)), write('Error'), nl;

not(d\_access(Ent1\_name, Ent2\_name)), write('Error'), nl;

not(grant(Ent1\_name, Ent3\_name, Right, Ent2\_name)), write('Error'), nl;

grant(Ent2\_name, Ent3\_name, Right, Ent4\_name), write('Error'), nl;

assert(grant(Ent1\_name, Ent3\_name, Right, Ent4\_name)), write('Dynamic right delegated'), nl

).

abrogate:-

write('Entity1 name: '), read(Ent1\_name),

write('Entity2 name: '), read(Ent2\_name),

write('Entity3 name: '), read(Ent3\_name),

write('Entity4 name: '), read(Ent4\_name),

write('Right: '), read(Right),

(

not(entity(Ent1\_name, \_, \_)), write('Error'), nl;

not(entity(Ent2\_name, \_, \_)), write('Error'), nl;

not(entity(Ent3\_name, \_, \_)), write('Error'), nl;

not(entity(Ent4\_name, \_, \_)), write('Error'), nl;

not(s\_right(Right, \_)), write('Error'), nl;

not(d\_access(Ent1\_name, Ent2\_name)), write('Error'), nl;

not(grant(Ent1\_name, Ent3\_name, Right, Ent2\_name)), write('Error'), nl;

not(grant(Ent1\_name, Ent3\_name, Right, Ent4\_name)), write('Error'), nl;

retract(grant(Ent1\_name, Ent3\_name, Right, Ent4\_name)), write('Dynamic right abrogated'), nl

).