International Journal of Information & Network Security (IJINS)
Vol.2, No.1, February 2013, pp. 78~90
ISSN: 2089-3299

Modeling and Verification of Access Rights in Take-Grant Protection Model Using Colored Petri Nets

Saeid Pashazadeh
Faculty of Electrical and Computer Engineering, University of Tabriz

Article Info

Article history:

Received Oct 08th, 2012
Accepted Oct 26th, 2012

Keyword:

Take-Grant protection
Modeling
Model checking
Verification
Proof generator

ABSTRACT

Take-Grant protection model (TGPM) is a powerful method for modeling access rights in a wide range of systems. It is graph based formal method that can be used for studying situations that rights may unintentionally be transferred as rights leakage. Deduction of new rights using rules of this model is difficult and time-consuming task especially for systems that have numerous parties and many rights between them. In this paper a novel model of TGPM using colored Petri net is presented. Using model checking of state space of the model we can prove that a party in the system can have specific right over other party or not? If a right leakage exists, model checking of state space of the model can generate automatic proof for it.

Copyright © 2013 Institute of Advanced Engineering and Science.
All rights reserved.

Corresponding Author:

Saeid Pashazadeh,
Departement of Information Technology,
Faculty of Electrical and Computer Engineering,
University of Tabriz,
29th Bahman Boulevard, Tabriz, East Azerbaijan Province, Iran
Email: pashazadeh@tabrizu.ac.ir

1. INTRODUCTION

Leakage of rights is one of the security threats in most computerized systems that a user or application may earn in the system. Right leakage is hidden in first glance and it is possible to be detected only by accurate analysis of the system. TGPM is a formal graph based method for modeling rights in wide range of systems. Labels of edges in TGPM graph represent rights of a system and vertexes of the graph represent subjects or objects of the system [1]. TGPM works based on the four basic rules. Hierarchical TGPM is introduced for modeling complex systems [2]. TGPM is another alternative to classical access control matrixes but has greater capabilities in comparison with them [3]. TGPM is a useful method for rights representation in wide range of systems especially in non-computerized applications. It permits us to study the security policies of the system and ensure that permissions of the system follows from constraints of the security policy [4]. TGPM is extended to support more complex and realistic real systems [5].

Hierarchical colored Petri net is powerful formal modeling language with great modeling capabilities. It permits formal modeling and verification of wide range of systems like proof generator of functional dependencies for normalization of databases, mechanism of concurrency controls in distributed database systems and communication protocols of different levels of computer networks [6, 7]. Colored Petri net can easily be applied in formal verification of security protocols and mechanism. TGPM is modeled using colored Petri net in this paper. Then a sample case study scenario is presented and state space graph of the system is generated. Finally, proof of a specific access right is explored in state space graph of the model using model checking.

2. COLOR SETS, VARIABLES AND INITIAL MARKINGS

In this part, brief descriptions of color sets, variables and initial markings that are used in modeling of the protocol are presented.

2.1. Color Sets

Color sets that are used in modeling of protocol are as follows:

```
colset VERTEXTYPE = with S | O | M;
colset VERTEXID = STRING;
colset RIGHT = with t | g | r | w | e | a;
colset RIGHTS = list RIGHT;
colset RULETYPE = with I | T | G | C | R;
colset PREDPERMLIST = list INT;
colset PERMGEN = product RULETYPE*PREDPERMLIST;
colset PERM = record N:INT * ST:VERTEXTYPE * SI:VERTEXID * R:RIGHTS *
DT:VERTEXTYPE * DI:VERTEXID * G:PERMGEN;
colset PERMS = list PERM;
```

Color set VERTEXTYPE represents vertex types in TGPM. S stands for subject, O for object and M for either subject or object. Color set VERTEXID is defined to represent the identifier of vertexes of the TGPM and is of type STRING. Color set RIGHT represents different rights (access types) in the graph. Color set RIGHTS is used to represent the list of rights that edges of TGPM can have. Color set RULETYPE represents first letter of a rule that is used in deduction of new edge of graph. Color set PREDPERMLIST is defined as list of integer values and represents list of rights (edges) that are used for deduction of a new edge by rules of the system. Color set PERMGEN represents type of rule and list of rights that are used for deduction of current right.

Colour set PERM is defined for fully introducing a right in TGPM. It is a record that contains seven fields. First field is denoted with title N and is of type integer and represents the number (index) of current right. Second field is denoted with title ST and is of type VERTEXTYPE and represents the type of source vertex of current right. Third field is denoted with the title SI and is of type VERTEXID and represents the identifier of source vertex of current right. Fourth field is denoted with title R and is of type RIGHTS and represents the list of rights of current edge. Fifth field is denoted with DT and is of type VERTEXTYPE and sixth field is denoted with DI and is of type VERTEXID and respectively represent the type and identifier of destination vertex of each right. Last field is denoted with title G and is of type PERMGEN and represents that which rights and rule are used for deducing current right. Color set PERMS is defined to represent all rights of the TGPM.

2.2. Variables and Initial Markings

A simple case study is modeled in this paper. Figure 1.a shows a sample TGPM and Figure 1.b shows a permission that we want to study it. We want to test that: Can permission of Figure 1.b be deducted using rules of TGPM from current rights of the system? How such leakage of right can occures?

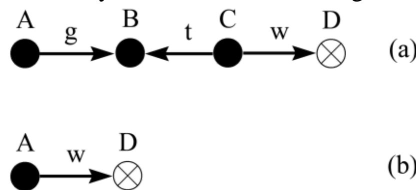

Figure 1.a) TGPM graph of a case study system. b) Specific right of under study

Initial parties and rights of the system are shown in Figure 1.a. These rights are represented as following initial marking of place Perms in the CPN model of system.

```
Val  InitialPerms=[{N=1,ST=S,SI="A",R=[g],DT=S,DI="B",G=(I,[])},
                   {N=2,ST=S,SI="C",R=[t],DT=S,DI="B",G=(I,[])},
                   {N=3,T=S,SI="C",R=[w],DT=M,DI="D",G=(I,[])}]
```

Variables of the model are as follows:

```
var L, L1, Lt, Lg : PERMS;       var u:UNIT;      var c, ct, cg :BOOL;
```

2.3. Model of TGPM System

Figure 2 shows the CPN model of TGPM system. Model is designed such that create and remove transitions fires only once. Multiple firing of create transition creates redundant rules.

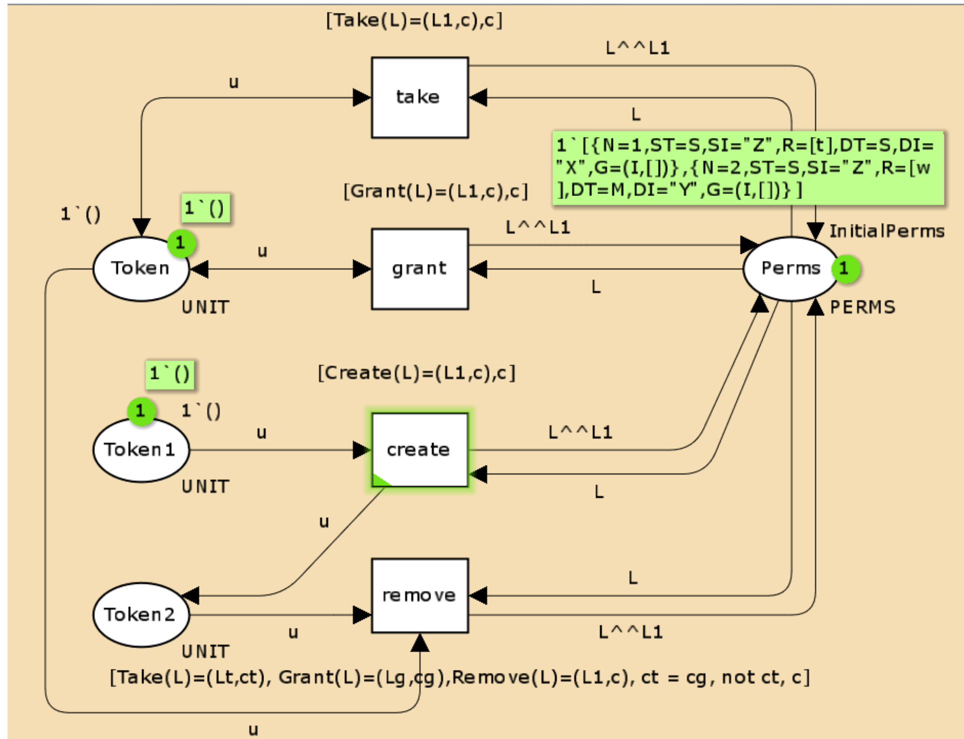

Figure 2. CPN model of TGPM system

Take and grant transitions can fire repeatedly. Transition remove fires after firing of transition create and when transitions take and grant cannot fires. Transitions that are related to rules of the TGPM follows a similar template. Guard condition of these transitions calls related function of these transitions. These functions get a list of current rights of the system as input and return a list of newly generated rights using related rule and a Boolean output value as results. If a newly generated right exists in the list of current existing rights or in the list of newly generated rights, it will not append to the list of newly generated rights for prohibiting existance of repeated rights. If a new deducted right is not repeated, then function adds this new right in the list of newly generated rights. If function cannot produce any new right using a rule, then it returns false and in otherwise it returns true. If transition of a rule cannot produces new rules, it will not become enabled.

3. FUNCTIONS OF MODEL

In this part a brief description of model's functions is presented. Figure 3 shows the structure chart of model's functions.

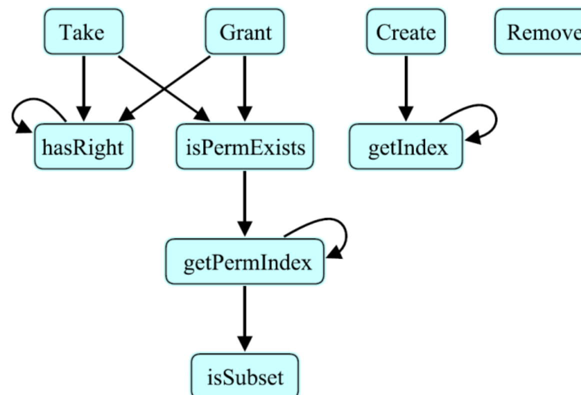

Figure 3. Structure chart of model's functions

Recursive function hasRight takes a right and a list of rights and searches this right in the list of rights. If the right exists in the list, function returns true and in otherwise returns false.

```
    fun hasRight(f:RIGHT,(h::L):RIGHTS):bool =
         if f = h then
              true
          else
              hasRight(f,L)
    | hasRight(_,[]) = false;
```

Function isSubset takes two lists of the rights and if rights of the first list is subset of rights of second list, it returns true and in otherwise returns false.

```
fun isSubset(L1:RIGHTS, L2:RIGHTS):bool =
 let    val   n1 = List.length(L1)
        val   n2 = List.length(L2)
        val   i = ref 0
        val   j = ref 0
        val   Found = ref true
  in  if n1 > n2 then
          false
      else(
         while !i < n1 andalso !Found do(
           let val F1 = List.nth(L1,!i)
           in  j := 0;
               Found := false;
               while !j < n2 andalso !Found = false do(
                  let  val F2 = List.nth(L2,!j)
                  in   if F1 = F2 then Found := true
                                else (  )
                  end;
                  j := !j+1)  (* while j *)
           end;
           i := !i+1);    (* while i *)
         !Found
         )
  end
| isSubset([],[]) = true
| isSubset(_,[]) = false
| isSubset([],_) = false;
```

Function getPermIndex takes a right as first parameter and list of rights as second parameter and returns index of first right in the list of rights of second parameter starting from index zero. If it can not find, returns -1.

```
fun getPermIndex(f:PERM, (h::L):PERMS):int =
 if #ST f = #ST h  andalso #SI f = #SI h andalso #DT f = #DT h andalso
        #DI f = #DI h andalso isSubset(#R f,#R h)
  then   0
   else
      let val res = getPermIndex(f,L)
      in   if (res <> ~1) then
                 res+1
           else   ~1
      end
| getPermIndex(_,[]) = ~1;
```

Function isPermExists takes a right as first parameter and list of rights as second parameter and if right of first parameter exists in the list of rights of the second parameter, then function returns true and in otherwise returns false.

```
fun isPermExists(f:PERM, L:PERMS):bool =
```

```
  let  val  n = getPermIndex(f,L)
  in   if n <> ~1 then
           true
      else
           false
  end
| isPermExists (_,[]) = false;
```

Function getIndex takes a value as first parameter and list of values as second parameter. It returns position of first parameter in the list of second parameter (index starts from zero) and in otherwise returns -1.

```
fun getIndex(f,(h::L)):int =
  if  f = h then 0
  else
     let  val res = getIndex(f,L)
     in   if  (res <> ~1) then
               res+1
          else   ~1
     end
| getIndex(_,[]) = ~1;
```

Function Remove takes list of current rights in the system as input and produces a list of new rights using remove rule of the TGPM.

```
fun Remove(L:PERMS):PERMS * bool =
  let   val   L2 =  ref []                    val   n = List.length(L)
        val   i = ref 0                       val   j = ref 0
        val   cs = ref 0                      val   Found = ref false
        val   Gen = ref false                 val   nr = ref 0
  in
    nr := n+1;
    while !i<n do(
      let  val F1 = List.nth(L,!i)
           val ta ={N=(!nr),ST=(#ST F1),SI=(#SI F1),R=[],DT=(#DT F1),
                          DI=(#DI F1), G=(T,[#N F1])}
      in
          if #ST F1 = S then(
                  L2 := !L2^^[ta];
                  nr := !nr+1;
                  Found := true)
             else ()
      end;
      i := !i+1);      (* while i *)
      (!L2,!Found)
  end
| Remove([]) = ([],false)
```

Function Take gets a list of current rights of the system and produces new rights using take rule of TGPM. This function returns list of new deductable rights as first output. If this function can produce new rights, it returns true in second output parameter and in otherwise returns false.

```
fun Take(L:PERMS):PERMS* bool =
  let   val   L2 =  ref []                    val   n = List.length(L)
        val   i = ref 0                       val   j = ref 0
        val   cs = ref 0                      val   Found = ref false
        val   Gen = ref false                 val   nr = ref 0
  in
     nr := n+1;
     while !i < n   do (
        let val F1 = List.nth(L,!i)
        in  j := 0;
            while !j < n do (
               if  !i < !j  then
                 let   val F2 = List.nth(L,!j)
                       val ta = {N=(!nr), ST=(#ST F1), SI=(#SI F1), R=(#R F2),
```

```
                          DT=(#DT F2), DI=(#DI F2),G=(T,[#N F1, #N F2])}
                   val tb = {N=(!nr), ST=(#ST F2), SI=(#SI F2), R=(#R F1),
                          DT=(#DT F1), DI=(#DI F1),G=(T,[#N F2, #N F1])}
             in  if #ST F1 = S andalso (#DT F1 = S orelse #DT F1 = M)
                      andalso #DT F1=#ST F2 andalso #DI F1 = #SI F2 andalso
                      hasRight(t,#R F1) andalso #SI F1 <> #DI F2 then
                         (cs :=1; Gen := true)
                 else if #ST F2 = S andalso (#DT F2 = S orelse #DT F2 = M)
                      andalso #DT F2= #ST F1 andalso #DI F2 = #SI F1 andalso
                      hasRight(t,#R F2) andalso #SI F2 <> #DI F1 then
                         (cs := 2; Gen := true)
                      else ( cs := 0; Gen := false);
                 case (!cs) of
                     1 => if !Gen=true andalso not(isPermExists(ta,L))
                                  andalso not(isPermExists(ta,!L2)) then
                              (L2 := !L2^^[ta];
                               nr := !nr+1;
                               Found := true)
                          else ()
                     | 2 => if !Gen=true andalso not(isPermExists(tb,L))
                                  andalso not(isPermExists(tb,!L2)) then
                              (L2 := !L2^^[tb];
                               nr := !nr+1;
                               Found := true)
                            else ()
                     |0 => ()
                    end
              else ( );
             j := !j+1)  (* while j *)
     end;
     i := !i+1);    (* while i *)
    (!L2,!Found)
  end
|  Take([]) = ([],false)
```

Function Create gets a list of current rights of the system and produces new rights using create rule of TGPM. This function returns list of new deductable rights as first output. If this function can produce new rights, it returns true in second output parameter and in otherwise returns false.

```
fun Create(L:PERMS):PERMS * bool =
  let   val   L2 =  ref []                     val   n = List.length(L)
        val   i = ref 0                        val   index = ref 0
        val   cs = ref 0                       val   Found = ref false
        val   Gen = ref false                  val   nr = ref 0
        val   LI = ref []
  in
     nr := n+1;
     while !i< n do(
        let  val  F1 = List.nth(L,!i)
        in
             if #ST F1 = S then(
                   index := getIndex(#SI F1,!LI);
                   if (!index <> ~1) then ( )
                   else (LI := !LI^^[#SI F1];
                          let val ta={N=(!nr),ST=S,SI=(#SI F1),R=[t,g,r,w,e,a],
                                         DT=M,DI=(#SI F1^"1"),G=(C,[#N F1])}
                           in
                             L2 := !L2^^[ta];
                             nr := !nr+1;
                             Found := true
                           end )
              )  else ();
              if  #DT F1 = S then (
                   index := getIndex(#DI F1,!LI);
                   if (!index <> ~1) then (  )
                   else (LI := !LI^^[#DI F1];
```

```
                        let  val ta={N=(!nr),ST=S,SI=(#DI F1),R=[t,g,r,w,e,a],
                             DT=M, DI=(#DI F1^ "1"), G=(C,[#N F1])}
                            in
                            L2 := !L2^^[ta];
                            nr := !nr+1;
                            Found := true
                           end )
            )
           else ()
       end;
       i := !i+1 );  (* while i *)
      (!L2,!Found)
end
     |  Create([]) = ([],false)
```

Function Grant gets a list of current rights of the system and produces new rights using grant rule of TGPM. This function returns a list of new deductable rights as first output. If this function can produce new rights, it returns true in second output parameter and in otherwise returns false.

```
fun Grant(L: PERMS) : PERMS* bool =
  let   val   L2 =  ref []             val   n = List.length(L)
        val   i = ref 0                val   j = ref 0
        val   cs = ref 0               val   Found = ref false
        val   Gen = ref false          val   nr = ref 0
  in
     nr := n+1;
     while !i < n do (
        let val  F1 = List.nth(L,!i)
        in  j := 0;
            while !j < n do (
               if  !i < !j  then
                 let  val F2 = List.nth(L,!j)
                      val ta ={N=(!nr),ST=(#DT F1),SI=(#DI F1),R=(#R F2),
                              DT=(#DT F2),DI=(#DI F2),G=(G,[#N F1,#N F2])}
                      val tb ={N=(!nr),ST=(#DT F2),SI=(#DI F2),R=(#R F1),
                              DT=(#DT F1),DI=(#DI F1),G=(G,[#N F2,#N F1])}
                      val tc={N=(!nr+1),ST=(#DT F2),SI=(#DI F2),R=(#R F1),
                              DT=(#DT F1),DI=(#DI F1),G=(G,[#N F2,#N F1])}
                  in if #ST F1 = S andalso(#DT F1 = S orelse #DT F1 = M)
                          andalso #ST F1=#ST F2 andalso #SI F1=#SI F2
                          andalso hasRight(g,#R F1)andalso #DI F1<>#DI F2
                      then (  cs :=1;   Gen := true )
                      else ( cs:= 0; Gen := false );
                     if #ST F2=S andalso(#DT F2 = S orelse #DT F2 = M)
                          andalso  #ST F2 = #ST F1 andalso #SI F2 = #SI F1
                          andalso hasRight(g,#R F2)andalso #DI F2<> #DI F1
                      then
                          ( if !cs = 0 then cs:=2 else cs:=3; Gen:=true )
                      else ( );
                     case (!cs) of
                          1=>if not(isPermExists(ta,L))andalso
                                not(isPermExists(ta,!L2))then
                                 ( L2 := !L2^^[ta];
                                   nr := !nr+1;
                                   Found := true )
                             else ()
                         |2=>if not(isPermExists(tb,L))andalso
                                not(isPermExists(tb,!L2))then
                                  ( L2 := !L2^^[tb];
                                    nr := !nr+1;
                                    Found := true )
                              else ()
                         |3=>(if not(isPermExists(ta,L))andalso
                                 not(isPermExists(ta,!L2))then
                                   ( L2 := !L2^^[ta];
                                     nr := !nr+1;
```

```
                                                Found := true )
                                         else ();
                                         if not(isPermExists(tc,L))andalso
                                            not(isPermExists(tc,!L2))then
                                              ( L2 := !L2 ^^ [tc];
                                                nr := !nr +1;
                                                 Found := true )
                                               else ())
                                    |0 => ()
                                  end
                            else (  );
                           j := !j+1)   (* while j *)
                   end;
                   i := !i+1);        (* while i *)
                  (!L2,!Found)
            end
     |   Grant([]) = ([],false)
```

4. STATE SPACE GRAPH OF CASE STUDY MODEL

State space report of TGPM model using case study scenario that is shown in Figure 2 is as follows:

```
 State Space                       Liveness Properties
     Nodes:   42                          Dead Markings
     Arcs:    41                          [...,42,41,40,38,37]   8
     Secs:    0                           Dead Transition Instances
     Status: Full                         None
  Scc Graph                        Live Transition Instances
     Nodes:   42                          None
     Arcs:    41
  Secs:    0
```

Figure 4 shows the complete state space graph of the model intruced case study scenario in part 2.2. Extracting proof from state space requires model checking of the state space.

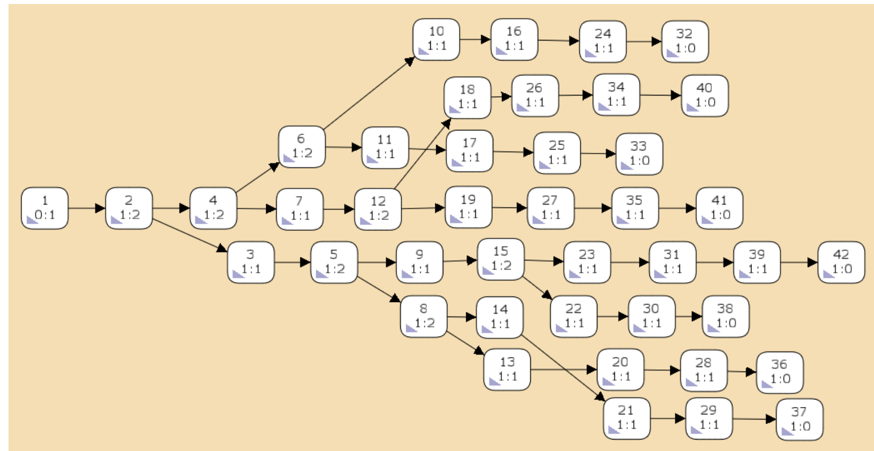

Figure 4. State space graph of model with case study scenario

Although state space of the model only has 42 nodes, but each node contains a list of rights that have many deducted rights using rules of TGPM. In designing color sets of the model, for each of the rights of the system a field that represents index of prerequisite rights and the rule that is used in deduction of current right is considered as explained in part 2.1.

5. MODEL CHECKING AND PROOF EXTRACTION

5.1. Functions of Model Checking

Text-based notation is used for representing steps of proof in this paper. A subject node X in TGPM is represented by notation ((X)), object Y is represented by [Y] and subject or object Z with (Z). Permission of Figure 1.b will be represented in textual form as ((A))-t->(B).

Figure 5 shows the structure chart of functions that are used in extracting proof of permission by analyzing nodes of state space graph of the model.

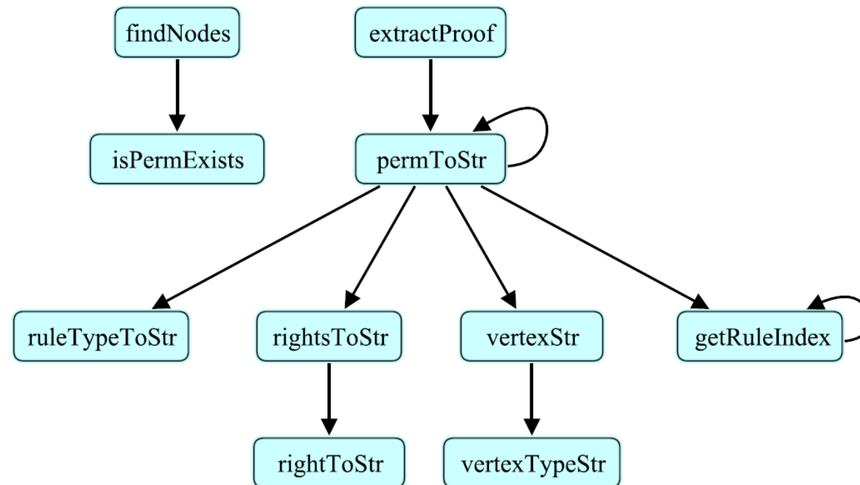

Figure 5. Structure chart of functions that are used in extracting proof of permission

Function vertexTypeStr gets two parameters that first parameter represents the vertex type of a node in TGPM. If Second parameter be 1, function returns a string that contains characters that must appear before title of a vertex node in text-based notation of permission rules and if it be 2, function returns a string that contains characters that must appear after title of a vertex node.

```
fun vertexTypeStr( vt : VERTEXTYPE ,  n : int ) : string =
    if n =1 then
           case vt of
               S => "(("
             | O => "["
             | M => "("
     else
         case vt of
               S => "))"
             | O => "]"
             | M => ")";
```

Function rightToStr takes a parameter of colorset RIGHT that represents one of the constituting rights of an arc in TGPM and converts it to equivalent text-based notation.

```
fun rightToStr(rg : RIGHT):string =
  case rg of
     t => "t" | g => "g" | r =>  "r" | w => "w" | e => "e" | a => "a";
```

Function vertexStr gets a parameter of colorset VERTEXID and a parameter of type VERTEXTYPE that represent identifier and type of a vertex node of TGPM and returns equivalent text-based notation of this node. It calls function VertexTypeStr.

```
fun vertexStr(VI :VERTEXID ,VT: VERTEXTYPE):string =
   vertexTypeStr(VT,1)^VI^vertexTypeStr(VT,2);
```

Function rightsToStr takes a parameter of colorset RIGHTS that represents a list of rights in a single arc of TGPM. This function converts all rights of single edge of graph to equilavent string format using function rightToStr without space between names of rights.

```
fun rightsToStr(rg: RIGHTS):string =
  let   val  n = List.length(rg)
        val  i = ref 0
        val  s = ref ""
  in  while !i < n do (
          let  val t1 = List.nth(rg, !i)
          in   s := rightToStr(t1) ^ !s
          end;
          i := !i+1);
      !s
  end
| rightsToStr([]) = "";
```

Function ruleTypeToStr takes a parameter of colorset RULETYPE and returns its text based representation.

```
fun ruleTypeToStr(rt: RULETYPE ):string =
  case rt of
    I => "Initial Access"
  | T => "Take"
  | G => "Grant"
  | C => "Ceate"
  | R => "Remove";
```

Recursive function getRuleIndex takes a permission number n as first parameter and a list of permissions as second parameter and returns position of permission n in the list of permissions (starting from index 0). If no permission with number n exists in the list of permissions in the second parameter, function returns -1 as the result.

```
fun getRuleIndex (n: INT ,(rl::L):PERMS):int=
  if n=#N rl  then  0
  else
      let val res = getRuleIndex(n,L)
      in  if  (res <> ~1) then
                res+1
          else  ~1
      end
| getRuleIndex(_,[]) = ~1;
```

Function permToStr takes a list of permissions pr and a permission p as input parameters and converts it to text-based representation. This recursive function calls functions vertexStr, rightsToStr, getRuleIndex and ruleTypeToStr.

```
fun permToStr(pr: PERMS, p:PERM):string =
  let   val  st = ref ""
        val  i1 = ref 0
        val  iL = ref 0
        val  s1 = ref " "
  in
    if  #1(#G  p) = I then
       st:=vertexStr(#SI p,#ST p)^"-"^rightsToStr(#R p)^"->"^vertexStr(#DI p,#DT p)
    else (
       let val Len= List.length ( #2( #G p) )
       in
         while !iL < Len do (
              i1 := getRuleIndex( List.nth( (#2(#G p)), !iL) ,pr);
              s1 := !s1^ permToStr( pr, List.nth(pr, !i1));
              iL := !iL+1;
              if ( !iL < Len ) then  s1 :=  !s1^","
              else ()
              );
```

```
          st:="{"^!s1^"="^ruleTypeToStr(#1(#G p))^"=>"^vertexStr(#SI p,
             #ST p)^"-"^rightsToStr(#R p )^"->"^vertexStr(#DI p,#DT p)^"}\n"
         end );
      !st
    end;
```

Function extractProof takes a list of permissions as first input parameter and index number of a specific permission as second parameter, then returns the proof in the form of general list and saves it in the file "Proof.txt" via calling function permToStr.

```
fun extractProof(p:PERMS, n:INT):string =
  let   val  ff = List.nth(p, n)
        val  s = ref ""
        val  f = TextIO.openOut "Proof.txt"
  in  s := permToStr(p, ff);
      TextIO.output(f, !s);
      TextIO.closeOut f;
      !s
   end
| extractProof([],_)="";
```

5.2. ML Codes of State Space Analysis

Constant finalPerm defines specific permission that its leakage is under study. From now, I name it target permission and use the model to test that, can subject A earn write permission on object D or not?

```
val finalPerm ={N=1,ST=S,SI="A",R=[w],DT=M,DI="D",G=(I,[])}:PERM;
```

Function findNodes gets a node n of state space as input and if target permission (finalPerm) appears in any nodes of the state space of the model, returns true and in otherwise returns false. Function ms_to_col is a build-in function of CPNTool and converts a multi set of a state space node to a list.

```
fun findNodes n=(isPermExists(finalPerm,ms_to_col(Mark.Model'Perms 1 n))
                   = true)
```

Signiture of function findNodes is as follows:

```
val findNode = fn: Node -> bool
```

Following ML code returns the list of state space nodes that contains target permission.

```
PredAllNodes   findNodes;
```

It is possible that target permission appears in more than one nodes of the state space. Output of executing this ML code on state space graph of case study model is as follows:

```
Val it=[42,41,40,39,38,37,36,35,34,33,32,31,30,29,28,27,25,24,22,21,17]:Node list
```

For simplicity, first node of the following list is used for extracting proof of target permission. Following ML code extracts index of first node of state space graph that contains target permission:

```
List.hd (PredAllNodes  findNodes);
```

Output of this ML code is as follows:

```
val it = 42 : Node
```

Following ML code returns list of permissions that exists in state space node with number 42.

```
ms_to_col (Mark.Model'Perms 1 (List.hd (PredAllNodes   findNodes))
```

Following ML code returns index of target permission in list of permissions of the first selected node (42) of state space.

```
getPermIndex(finalPerm,ms_to_col(Mark.Model'Perms 1
                                (List.hd(PredAllNodes   findNodes))));
```

Output of this ML code is as follows:

```
val it = 27 : int
```

It represents that our target permission is appeared in 27th permission of node 42 of state space. Following ML code returns number of permissions in the list of permissions of node 42 of state space.

```
List.length(ms_to_col(Mark.Model'Perms 1
                                (List.hd(PredAllNodes findNodes))));
```

Output of this ML code is as follows:

```
Val it = 47 : int
```

Following ML code extract the proof of target permission (finalPerm) from the first node of the state space that this permission appeared in it. Proof of the way that target permission occurs can be extracted by backward tracing of prerequisite rights of target right.

```
extractProof(ms_to_col(Mark.Model'Perms 1 (List.hd(PredAllNodes
findNodes))), getPermIndex(finalPerm,ms_to_col(Mark.Model'Perms 1
(List.hd (PredAllNodes findNodes)))));
```

Output of the function ExtractProof in the form of general list is as follows:

```
{
  { ((A))-g->((B)) =Ceate=> ((A))-aewrgt->(A1)},
  {
    {
     ((C))-t->((B)),
       {
       ((A))-g->((B)),{((A))-g->((B)) =Ceate=> ((A))-aewrgt->(A1)} =Grant=>
                                                            ((B))-aewrgt->(A1)
       } =Take=> ((C))-aewrgt->(A1)
    },
   ((C))-w->(D) =Grant=> (A1)-w->(D)
  }  =Take=> ((A))-w->(D)
}
```

For more clarity, I drew automatically generated proof in simple graph-based format as is shown in Figure 6. If we want to test whether current system can have special right leakage, it is sufficient that we search list of rights in all nodes of state space graph. If we find that right, then right leakage can happen. Proof of the way that this leakage occurs can be extracted using function extractProof.

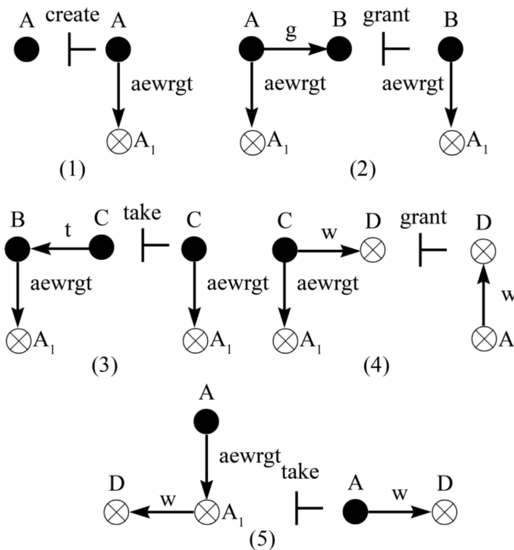

Figure 6. Steps 1 to 5 shows steps of automatically generated proof of write permission of subject "A" on object "D" in graph representation

6. CONCLUSION

Colored Petri net is powerful formal method with great modeling capabilities and facilities for model checking. In this paper, TGPM along with a simple case study scenario is modeled using colored Petri net. State space of this model is analyzed using model checking and leakage of a sample right is studied. Results show that state space of the model is generated in short time and automatic proof of right leakage can be generated easily. Manual testing right leakage in big systems is very tedious work that is not feasible without using automatic proof generator tools.

REFERENCES

[1] A. K. Jones, *et al.*, "A Linear Time Algorithm For Deciding Security," *17th Annual Symposium on Foundations of Computer Science*, pp. 33-41, 1976.

[2] M. Bishop, "Hierarchical Take-Grant Protection Systems," *8th ACM symposium on Operating systems principles*, Pacific Grove, California, United States, 1981.

[3] M. A. Harrison, "Theoretical Issues Concerning Protection in Operating Systems," *EECS Department, University of California, Berkeley UCB/CSD-84-170*, 1984.

[4] M. Bishop, "Applying The Take-Grant Protection Model," *Technical Report PCS-TR90-151, Dept. of Computer Science, Dartmouth College, Hanover, NH 03755* (1990)

[5] M. Bishop, Computer Security: Art and Science: Addison-Wesley, 2003.

[6] S. Pashazadeh and M. Pashazadeh, "Modelling An Automatic Proof Generator For Functional Dependency Rules Using Colored Petri Net," *International Journal in Foundations of Computer Science & Technology (IJFCST)*, vol. 2, no. 5, pp. 31-47, September 2012.

[7] S. Pashazadeh, "Modeling and Verification of Deadlock Potentials of a Concurrency Control Mechanism in Distributed Databases Using Hierarchical Colored Petri Net," *International Journal of Information and Education Technology (IJIET)*, vol. 2, no. 2, pp. 77-82, April 2012.

BIOGRAPHY OF AUTHOR

Saeid Pashazadeh is Assistant Professor of Software Engineering and chair of Information Technology Department at Faculty of Electrical and Computer Engineering in University of Tabriz in Iran. He received his B.Sc. in Computer Engineering from Sharif Technical University of Iran in 1995. He obtained M.Sc. and Ph.D. in Computer Engineering from Iran University of Science and Technology in 1998 and 2010 respectively. He was Lecturer in Faculty of Electrical Engineering in Sahand University of Technology in Iran from 1999 until 2004. His main interests are modeling and formal verification of distributed systems, computer security and wireless sensor/actor networks. He is member of IEEE and senior member of IACSIT and member of editorial board of journal of electrical engineering at University of Tabriz in Iran.