Tree-based Access Control Modeling Language

Arne Øslebø
Department of Telematics, NTNU
O.S. Bragstads plass 2B
7491 Trondheim, Norway
arneos@uninett.no

7th September 2004

Abstract

In today’s multi-user systems, proper access control is an important aspect of a system’s overall security configuration. The Tree-based Access Control Modeling Language (TAML) is a graphical language for modeling mandatory access control for applications that store information in a tree-based structure. The advantage of such a modeling language is that one single method can be used for specifying access control in multiple applications like SNMP, LDAP or web pages in an HTTP server. The language is also very helpful in heterogeneous systems where systems from different vendors need to be configured.

The paper presents a general overview of the language, shows an example of how it can be used and describes a prototype implementation.

Keywords: access control, diagrams, modeling language, SNMP

1 Introduction

Pictures, diagrams and graphical notations have been used for a long time in computer science and telecommunication to help programmers and system operators to understand the complexities of modern computer and communication systems. Some examples of highly successful diagrammatic notations are the Entity-Relationship diagrams[3] which is used for modeling databases, Specification and Description Language (SDL)[4] which is used for specifying and designing telecommunication systems and the Unified Modeling Language (UML)[7] which is a collection of several diagram types and is used for object oriented software development. These graphical notations are just a few examples of successful graphical notations that prove the old proverb “A picture is worth a thousand words”.

1
In this paper the Tree-based Access Control Modeling Language (TAML) is presented. TAML is a graphical language that can be used for specifying and managing mandatory access control rules in applications that store information in a tree based structure. The language is a generic version of the MIB View Modeling Language (MVML)[1] which was developed for specifying access control in SNMPv3.

In today’s multi-user systems proper access control is a critical aspect of the overall security of a system. The problem is that as the number of users increases and if detailed access control is needed it can quickly become a daunting task to make sure that nobody is allowed any unauthorized access to resources. To make things even more complicated it is very common to have heterogeneous systems that contain equipment or software from multiple vendors, each of which often has to be configured in a different way to specify access control.

The TAML language has several advantages over conventional manual configuration of access control. First of all it is a graphical notation that makes it easier for administrators to see exactly who has access to which resources. It can also be used to configure products from different vendors and even completely different systems or applications. TAML can be used to configure mandatory access control rights in all applications or systems that store information in a tree-based structure. This can for example be SNMP, LDAP or even HTTP servers which stores documents on a tree-based file system.

2 Tree-based structures

Storing information in a tree structure is a method commonly used in computer science as it is an efficient method for organizing data in a hierarchical manner.

TAML can be used for specifying access control in rooted trees[8]. An example of a rooted tree is shown in Figure 1.

In this figure the circles with a number inside are called nodes and the lines between the nodes are edges. In a rooted tree one of the nodes are always distinguished from the others and forms the root of the tree.
The common mathematical notation for a tree $T$ is $T = (V, E)$ where $V$ is a finite set of nodes and $E$ is a binary relation on $V$ specifying edges. With this notation the tree shown in Figure 1 could be written as:

$T = \{(1, 2), (1, 3), (1, 4), (2, 5), (2, 6), (5, 9), (4, 7), (4, 8)\}$

When talking about a tree structure, it is often useful to refer to only parts of the tree. For example a subtree rooted at node $x$ is the tree rooted at $x$ containing the descendants of $x$. In a rooted tree $T$ with root $r$, any node $y$ on the direct path from $r$ to $x$ is an ancestor of $x$. If $y$ is an ancestor of $x$, then $x$ is a descendant of $y$. In Figure 1 the subtree rooted at node 2 will include node 2, 5, 6 and 9.

By definition all nodes are both an ancestor and a descendant of itself. If $x \neq y$ and $y$ is an ancestor of $x$, then $y$ is a proper ancestor of $x$ and $x$ is a proper descendant of $y$.

It can also be useful to refer to the children of a node. On the path from root $r$ of a tree $T$, to a node $x$, the last edge on the path is $(y, x)$. Here $y$ is the parent of $x$ and $x$ is a child of $y$. When two or more nodes have the same parent, they are siblings. The only node in $T$ that does not have any parent is the root node $r$.

## 2.1 Access control in tree structures

Access control in a tree structure means that access to only certain nodes or subtrees are allowed. The collection of access rules that specifies which nodes a user $U$ has access to in tree at can be written as $U = (T, A)$ where $T = (V, E)$ is the tree and $A$ is a set of tuples of the type $\{(N, I, S)\}$, where $N$ is a node, $I \in \{i, e\}$ specifies if a node is included or excluded and $S \in \{s, c, n\}$ specifies if access is granted to the entire subtree rooted at $N$, the children of $N$ or just the node $N$.

With this notation it is always assumed that all children of a node $N$ is included when $S \in \{s, c\}$. If only proper children are wanted, then two rules have to be specified. The first rule includes all children and the second removes the parent node so that only proper children are left.

Assume that user $U$ has access to node 2, 4, 5 and 6 in Figure 1. This can be written as $U = (T, A)$ where $T$ is as previously defined and $A = \{(2, i, c), (4, i, n)\}$.

Given a function $f(U)$ that returns all nodes that $U$ provides access to, $U' = (T', A')$ is equal to $U = (T, A)$ if and only if $f(U') = f(U)$. For Figure 1 it is also possible to write $U' = (T, A')$ where $A' = \{(2, i, s), (9, e, n), (4, i, n)\}$. In this example $U' = U$ because $f(U') = f(U) = \{2, 4, 5, 6\}$.

It is often advisable to optimize the number of entries in the set $A$. The set $A'$ is an optimization of $A$ if $f(A) = f(A')$ and $|A'| < |A|$. $A'$ is fully optimized if there do not exist an $A''$ where $f(A'') = f(A')$ and $|A''| < |A'|$.

## 2.2 Extended object identifiers

In the previous examples, the nodes in the tree from Figure 1 have been addressed directly by their names. In real systems nodes can often have the same name and must
therefore be addressed by a name that traverses the tree from the root node so that each node can be uniquely identified. To accomplish this extended object identifiers (EOIDs) are used. Normal OIDs are an ASN.1 data type that can be used as reference to data objects and are an ordered list of non-negative numbers. In Internet RFCs OIDs are usually written using a character string where the numbers are separated by a dot. For example the OID “1.2.5.9” points to node 9 of Figure 1.

Extended OIDs are a superset of normal OIDs as they are not limited to only simple non-negative numbers. An EOID is a string containing a list of string identifiers separated by some character token.

3 Tree-based Access Control Modeling Language

The Tree-based Access Control Modeling Language is a general purpose graphical notation that can be used for specifying and configuring access control in applications that store information in a tree-based structure. Figure 2 shows the various components that are needed for using TAML to configure the access control in a system. One goal when designing TAML was to make it simple to use and easy to implement support for new applications. So in this figure only the three boxes with grey background, Tree Generator, ACL Configurator and Application Attributes XML, represents code or XML schema that have to be specifically designed for the application that TAML is being used to specify access control rules for. All other boxes are generic code or XML formats that are common for all use of TAML.

Editors The editor used for drawing TAML diagrams can either be specifically designed for this task, or it can be a standard UML[7] editor. If a UML editor is used, diagrams will typically be stored in the XMI[5] format and then translated to the TAML XML format using XSLT[6].

TAML ACL Generator This module reads a TAML XML document and generates a list of access rules for each user and entity in the diagram. The access rules are stored in a simpler TAML ACL XML document that can be read by either the TAML ACL Optimizer or the ACL Configurator.

TAML ACL Optimizer The list of access rules generated by the TAML ACL generator will often not be optimized when it comes to having the minimum number of access rules. The goal of the TAML ACL optimizer is to take a TAML ACL XML file and for each user and entity find the fully optimized A in \( U_E = (T, A) \).

Tree Generator This module provides the description of the tree \( T \) that the TAML ACL Optimizer needs to do proper optimization. This generator must be specifically implemented for the application that TAML is used to specify access control rules for. The description of the tree is stored in an XML file format that can be read by the TAML ACL Optimizer.
Figure 2: TAML components
ACL Configurator  The ACL Configurator also needs to be implemented specifically for the application that TAML is used to model access control rules for. The ACL Configurator reads a TAML ACL XML file and uses this to do the appropriate configuration needed to implement the access control rules specified in the TAML diagram. The ACL Configurator can also take an application specific XML Schema as input and use it to verify application specific attributes.

3.1 Notation

TAML is a relatively simple graphical language with only two relations and eight symbols as shown in Figure 3.

3.1.1 Diagrams

In TAML two different types of diagrams are used. One is the top level diagrams that collect all symbols that define the access rights of users for a specific type of access like read-only, read-write etc. A top level diagram might contain one or more group symbols and each group symbol also have a group diagram attached to them where the content of the group is defined.

If the access rules for a user are different for different access types, there will be multiple main diagrams with one diagram for each access type. If the access rules are the same for multiple access types, only one main diagram is needed.
3.1.2 Relations

There are only two relations defined in TAML, include and exclude. The include relation is used to include nodes in the access rights while the exclude relation is used for excluding them.

3.1.3 Symbols

Each symbol has a set of attributes that can be used to control the exact semantic of the symbol. For example the user symbol has attributes that define the login name and password of the user while the entity symbol contains attributes to identify and locate the entity so that it can be configured. Each symbol also has the possibility of adding application specific attributes that can be used by the ACL Configurator when configuring the access control rules.

Cfg
The Config symbol is used for providing configuration information to the TAML ACL generator, TAML ACL optimizer and ACL Configurator. One of the most important roles of the Config symbol is that it configures what type of access should be specified by the diagram that contains the symbol. Type of access can for example be read-only, read-write etc.

Children
The children symbol includes or excludes the children of a node in the access rights. An EOID is used to identify the parent node and there is also an attribute specifying if the symbol should include all children or only proper children.

Entity
The entity symbol specifies which entity or entities a user has access to. An entity identifies where the access control rules should be configured. It can be a PC, a router or any other type of equipment where the access control need to be configured. The entity symbol can also represent software. For example if a server is running two HTTP servers, the entity symbol will uniquely identify which server should be configured.

Group
The group symbol is used to group together related symbols to make diagrams easier to read and to be able to reuse parts of a TAML diagram. A group symbol has its own diagram attached to it where the content of the group is drawn.

Node
The node symbol is used to include or exclude a single node in the access rights. An EOID is used to identify the exact node in the tree that the node symbol represents.

Subtree
The subtree includes or excludes a subtree in the access rights. An EOID is used to identify the root node of the subtree.
Table row  The table row symbol represents a row in a virtual table. Tables are a technique commonly used by SNMP to help organize the data. The EOID for a cell in a generic table is written as \( T.C.I \) where \( T \) is the EOID for the table, \( C \) is the column and \( I \) is the index of the row. To give access to a specific row in a table, \( T \) and \( I \) will be constant and \( C \) will be a wildcard so that all columns of the row are included.

It is also possible to use some predefined functions when specifying the index attribute of the tree row symbol. The predefined functions returns all or part of the EOID used as index for the row. The exact functions available will depend on the implementation of TAML and which type of application access control is being configured for. One example is a function that returns the user ID of a user in a system. This makes it possible to use the same table row symbol for multiple users and each of them will get access to a different row in the table based on the user ID they have.

User  The user symbol represents one or more users that are allowed access to an entity. If the user symbol represents multiple users, all the users will have the same access rights. A user can not belong to more than one user symbol.

3.2 Example

The following example shows some of the aspects of the TAML language. In this example there is a company with two departments, \( A \) and \( B \). There are two users, \( A1 \) and \( B1 \) which belong to department \( A \) and \( B \). Department \( A \) have two entities, \( E_{A1} \) and \( E_{A2} \), which all the users have some access to. Figure 1 represents the tree structure of resources that are available in both entities. Table 1 shows which nodes each user have access to and using this table the following access rules apply to this example:

- \( A1_{E1} = \{(1.2, i, c), (1.4, i, n)\} \)
- \( A1_{E2} = \{(1.2, i, c), (1.4, i, n), (1.4.7, i, n)\} \)
- \( B1_{E1} = \{(1.2, i, c), (1.4, i, n), (1.2.6, c, n)\} \)

From the access rights we can see that the set of nodes \( \{2, 4, 5, 6\} \) are common for many of the access rights. To be able to reuse access specification between the users and entities it is therefore useful to draw the TAML diagram giving access to this set of nodes inside a TAML group symbol. The contents of this group is shown in Figure 4 and what it does is to use the children symbol to include nodes \( \{2, 5, 6\} \), and a node symbol to include node 4.

Figure 5 shows the TAML diagram that specifies the access rights for user \( U1 \). In this diagram user \( U1 \) includes the two entities \( E1 \) and \( E2 \). Entity \( E1 \) in turn includes
Table 1: Access rights

<table>
<thead>
<tr>
<th>Node</th>
<th>A1(E1)</th>
<th>A1(E2)</th>
<th>B1(E1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>5</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The group symbol $G$. This means that user $U_1$ is granted access to the nodes specified in group $G$ for entity $E_1$. For entity $E_2$, both the group $G$ and the node 7 is included and we get $f(U_1E_2) = \{2, 5, 6\} + \{7\} = \{2, 5, 6, 7\}$.

The TAML diagram for user $B_1$ is very similar and is shown in Figure 6. In this diagram the contents of the group $G$ is first included and then node 6 is excluded from the access rights to entity $E_1$. This provides access to the following nodes: $f(B_1E_1) = \{2, 4, 5, 6\} - \{6\} = \{2, 4, 5\}$.

3.3 Hierarchy

The example diagrams that have been shown clearly demonstrate the hierarchical nature of TAML. TAML itself follows a tree-based structure to specify access control. In this hierarchy it is easy to encounter situations where access control rules at different layers in the hierarchy have conflicts. Rules at one level may provide access to some resources while rules at another level can deny access to the same resources. The general rule in TAML is that access control rules should be calculated using a top-down approach where the access rules for each symbol in the main diagram is fully calculated before moving on to the next symbol. Rules at higher levels supersede rules at lower levels and if there are any discrepancies at the same level, the default is that rules excluding access rights have priority over include rules. This can however be changed.
Figure 5: User A1

Figure 6: User B1
by setting a priority attribute in the Config symbol

4 Implementation

A prototype of the TAML ACL Generator and a ACL Configurator for SNMPv3 have been implemented in JAVA. A commercial UML editor was used for creating the TAML diagrams. The diagrams were stored using the XMI format which was then converted to TAML XML format using XSLT.

The TAML ACL generator parses the TAML XML document using a SAX parser and represents the document using an internal object model as shown in Figure 7.

Most of the classes in the object model represents a TAML symbol in the TAML diagram. All of these objects are based on the object tamlObject which contain common code. As the XML SAX parser encounters the various XML tags that indicates a new TAML symbol or relation, it creates a new object for representing this symbol. As the parser continues it will find the XML tags describing the attributes of the symbols and these are then read and applied to the last object that was created. All objects that are created belong to the last diagram that was parsed. This can either be the main diagram or diagrams belonging to a group symbol.

When the parser reaches the end of a main diagram, it will start to generate a TAML ACL XML document for this diagram. It starts by finding all users and entities defined in the TAML diagram. For each entity it processes all users that have access to this entity and generates the TAML ACL XML document.

4.1 SNMPv3 ACL Configurator

The SNMPv3 ACL Configurator(SAC) reads TAML ACL XML documents and configures the access control in SNMP entities based on the information in these doc-
To control who has access to SNMP information, SNMPv3 uses the User-based Security Model (USM) [2] to authorize users and the View-based Access Control Model (VACM) [9] to provide proper access control. Both models can be configured through a series of SNMP set commands which create entries in various MIB tables.

The design of SAC is very similar to the TAML ACL generator and uses a SAX XML parser to create its own object model of the XML document. When the parser is finished reading all the access rights and access rules for users for a specific entity, this entity is configured before the next entity is parsed. To configure the SNMP access control in the entity, SAC first loops through all users and creates a series of SNMP set commands that creates the necessary entries in the USM MIB.

For each user SNMP set commands are then generated to create the necessary entries in the VACM MIB tables. This is a simple prototype implemented as proof of concept and no optimization is implemented. So even if two users have the same access control rights, two different groups will be created in the VACM MIB.

5 Conclusions and future work

The main goal of TAML is to provide an easy to use tool for specifying mandatory access control rules for applications that store information in tree-based structures. This is achieved by using a simple graphical notation to express the access control rules. The entire language only has eight symbols and two relations. This makes it an easy language to learn and to use. Most of the work of translating the diagrams into access control rules that can be used for configuring entities are done by a generic module so that adding support for new systems that can be configured through TAML is relatively simple.

One area that still needs more work is the optimization of the number of access control rules that are generated. Some systems that can be configured by TAML might have limited resources to implement large sets of access control rules. In the above example the contents of group $G$ used two rules to give access to the four nodes \{2, 4, 5, 6\}. This could also have been done by four rules giving access to each node. If someone used the latter technique, the TAML ACL Optimizer should automatically optimize this into only two rules. The problem is that to find the fully optimized $A$ the tree structure must be fully known in advanced. This is often not possible as the tree structure in some application can change dynamically based on the state of the system. Work has started on creating an algorithm that can find the optimized number of access control rules based on a limited knowledge of the full tree structure.

A policy version of the language is also under development. This version will be more complex to learn and use, but will offer role based access control and the possibility of distributed specification of access control across multiple administrative domains.
References


