# ZP Security Framework

*Marek Zelem and Milan Pikula*
Faculty of Electrical Engineering and Information Technology
Slovak University of Technology in Bratislava

## Keywords

security model, security policy, access matrix

## Abstract

In this paper, we introduce an access control framework with properties somewhat different from commonly used models. Our framework is defined in well-known terms of subjects, objects, and access rights, and is derived from the access matrix. After the definitions, we will show the scope of our framework and point out some of its applications.

## 1 Introduction

Access control models used today are mostly derived from the access control model introduced by Lampson [Lam71] and Graham and Denning [GD72]. Many of them share the basic idea of access control matrix on which they put some restrictions or rules. Each concept has some positives and some negatives. We introduce a framework which aims to be a link between most of them, with simplicity of implementation in mind. To achieve this goal, we define our substitution of the access matrix and some basic rules which apply on it (Virtual Spaces - VS model). This abstraction plays the role of some of known access control models in the static state. Then, we define the Security Decision Center which reflects the changes in system states.

# 2 ZP Security Framework

In this chapter, we define the ZP security framework, namely its two compoments, VS model and the Security Decision Center.

## 2.1 The VS Concept

The structure of the VS model is based on the access control model by Lampson [Lam71], and Graham and Denning [GD72]: The state of system is defined as a triple $(S, O, M)$ where $S$ is a set of subjects, $O$ is a set of objects, and $M$ is the access matrix which has one row for each subject and one column for each object. The cell $M[s, o]$ contains the access rights subject $s$ has for object $o$. These access rights are taken from a finite set of access rights $A$. To suit our purposes, we need to define this part more precisely. The set $A$ is a set of access **types** (for example read, write, etc.) and the presence of particular type $a \in A$ in the cell of table $M[s, o]$ is the **right** for subject $s$ to perform access of type $a$ on object $o$. Thus, we will refer to the set $A$ as to the set of access types.

The VS Model separates the objects and subjects into the finite number of domains, hereafter called **virtual spaces** (VS). The access matrix $M$ is decomposed into the set of properties of objects and set of properties of subjects.

Each object $o \in O$ in the system is assigned a membership in zero[1], one or more virtual spaces. This is defined by the function $OVS : O \to VS^*$.

Each subject $s \in S$ is assigned a set of abilities, one for each access type $a \in A$. Ability is set of zero, one or more virtual spaces. This is defined by the function $SVS_a : S \to VS^*$. Subjects often figure as objects and therefore the previous paragraph applies to subjects too.

Access of type $a \in A$ of the subject $s \in S$ on the object $o \in O$ is granted when the object and the subject share at least one common virtual space for the given access type, i.e.

$$SVS_a(s) \cap OVS(o) \neq \emptyset$$

[1]In fact, the object should belong to at least one virtual space to be visible for some subjects.

State of the system is changed by changing the abilities of subjects or membership of objects.

## 2.2 Properties of the VS Model

In this section, we describe the properties of the VS model.

Any state of the system $(S, O, M)$ can be easily translated to the VS model, provided that we have sufficient number of virtual spaces. We will mention techniques to do it later.

The output value of the functions $OVS$ and $SVS_a$ is a set of virtual spaces. The presence or absence of each virtual space $vs \in VS$ in the set can be represented as an array of boolean values, making it possible to describe some operations using boolean algebra. This results in an easy machine level implementation of the VS model.

The access matrix M is decomposed into two parts: first of them, related to the objects and the second one, related to the subjects. This can be easily implemented by adding some information to existing system subjects and objects.

## 2.3 Security Decision Center

The main role of the Security Decision Center (SDC) is to change the VS sets of objects and subjects in the system, i.e. $OVS(o)$ and $SVS_a(s)$. SDC catches the system events such as object or subject creation and accesses of subjects to objects. That are, in fact, the only points in which the change of security state of the system is reasonable. The SDC enables or disables given access and may change the state of the VS model.

The behaviour of the SDC, together with properly designed VS model, defines the particular security policy.

# 3 Using the ZP Security Framework

In this section, we focus on using the ZP security framework to implement various security models and policies.

## 3.1 Joining two VS Models - AND

Consider having two different VS configurations on the same object and subject space[2] (functions $OVS_1 : O \to VS_1^*$, $OVS_2 : O \to VS_2^*$ and $SVS_{1a} : S \to VS_1^*$, $SVS_{2a} : S \to VS_2^*$). If we want to use both of them at the same time, we must ensure, that the access is granted, only if both partial accesses are granted. This means, that

$$\Big(SVS_{1a}(s) \cap OVS_1(o) \neq \emptyset\Big) \wedge \Big(SVS_{2a}(s) \cap OVS_2(o) \neq \emptyset\Big)$$

Because the carthesian product $A \times B = \emptyset \Leftrightarrow A = \emptyset \vee B = \emptyset$, we can write

$$\Big(SVS_{1a}(s) \cap OVS_1(o)\Big) \times \Big(SVS_{2a}(s) \cap OVS_2(o)\Big) \neq \emptyset$$

which finally results in

$$\Big(SVS_{1a}(s) \times SVS_{2a}(s)\Big) \cap \Big(OVS_1(o) \times OVS_2(o)\Big) \neq \emptyset$$

Now we can easily make the new $SVS_a(s) : S \to VS^*$ and $OVS(o) : O \to VS^*$ functions:

$$VS = VS_1 \times VS_2$$
$$SVS_a(s) = SVS_{1a}(s) \times SVS_{2a}(s)$$
$$OVS(o) = OVS_1(o) \times OVS_2(o)$$

## 3.2 Joining two VS Models - OR

Consider having two different VS configurations on the same object and subject space (functions $OVS_1 : O \to VS_1^*$, $OVS_2 : O \to VS_2^*$ and $SVS_{1a} : S \to VS_1^*$, $SVS_{2a} : S \to VS_2^*$). Let the $VS_1 \cap VS_2 = \emptyset$. We want to grant access, when at least one partial access is granted. This means, that

$$\Big(SVS_{1a}(s) \cap OVS_1(o) \neq \emptyset\Big) \vee \Big(SVS_{2a}(s) \cap OVS_2(o) \neq \emptyset\Big)$$

Because $A \neq \emptyset \vee B \neq \emptyset \Leftrightarrow A \cup B \neq \emptyset$, we can write

$$\Big(SVS_{1a}(s) \cap OVS_1(o)\Big) \cup \Big(SVS_{2a}(s) \cap OVS_2(o)\Big) \neq \emptyset$$

[2] We refer to the object space as to the set of all objects (O); the same applies for the subject space, which is the set S.

Thus,

$$\Big(SVS_{1a}(s) \cup SVS_{2a}(s)\Big) \cap \Big(OVS_1(o) \cup OVS_2(o)\Big) \neq \emptyset$$

This leads to the new $SVS_a(s) : S \to VS^*$ and $OVS(o) : O \to VS^*$ functions as follows:

$$VS = VS_1 \cup VS_2$$
$$SVS_a(s) = SVS_{1a}(s) \cup SVS_{2a}(s)$$
$$OVS(o) = OVS_1(o) \cup OVS_2(o)$$

## 3.3 Optimizing the VS Model

In this section, we show the obvious facts about removing the unnecessary virtual spaces which are not used at any object, any subject or which are duplicate. The proof for all of the statements in this section is trivial and is based on fact that existence of such virtual spaces does not influence the process of granting or refusing the access.

If we have $x, y \in VS$ and $\forall o \in O : x \in OVS(o) \Leftrightarrow y \in OVS(o)$ then we can omit $x$ from the VS set[3], remove $x$ from the $OVS(o)$, and replace every occurence of $x$ in $SVS_a(s)$ with $y$.

If we have $x, y \in VS$ and $\forall s \in S, a \in A : x \in SVS_a(s) \Leftrightarrow y \in SVS_a(s)$ then we can omit $x$ from the VS set, remove $x$ from the $OVS(o)$, and replace every occurence of $x$ in $SVS_a(s)$ with $y$.

If we have $x \in VS$ and $\forall o \in O : x \notin OVS(o)$ then we can omit $x$ from the VS set[4] and remove each occurence of $x$ from $SVS_a(s)$.

If we have $x \in VS$ and $\forall s \in S, a \in A : x \notin SVS_a(s)$ then we can omit $x$ from the VS set and remove each occurence of $x$ from $OVS(o)$.

## 3.4 Transforming Access Control Lists to the VS Model

Access control list specifies, for given object, the list of subjects and their permitted access types to the object. We will first define the transformation of the access control lists to the VS model for one access type $a \in A$ and then we will show, how to extend this model for any number of access types.

[3]because $x$ and $y$ are duplicates

[4]because $x$ is never used

Assume the function $f : S \to VS$ assigns unique virtual space to each subject. Next, let the value of $OVS(o)$ will be a set of virtual spaces corresponding to subjects which have the right to perform the operation on object $o$. Let the value of $SVS_a(s) = \{f(s)\}$ (i.e. $SVS_a(s)$ contains exactly $f(s)$-th virtual space). Now we have the ACL for one type of operation on object.

Extending this model to support any number of access types is straightforward. First, we make a separate model for each access type. Then, we join them using OR - if any of partial accesses is granted, access is granted. We should be aware that this kind of joining requires the VS sets to be completely distinct, as stated in the definition of this kind of joining.

If we want to include a group $G \in S^*$ of subjects into the ACL, we can add new virtual space $x \in VS$ to each subject of the group, i.e. $\forall s \in G : SVS_{1a}(s) = SVS_a(s) \cup x$. Hierarchical ordering of ACLs is only matter of proper design of SDC, if there is a need for it.

## 3.5 Transforming Capabilities to the VS Model

Capabilities specify the access rights of subjects. Each subject is assigned a set of capabilities, each of them representing one object or group of objects, to which the subject has the access right. First, we will show the non-hierarchical capabilities, then we will show, how to represent hierarchy in VS.

Capabilities are very similar to abilities, which are present in each subject. Assume the function $f : O \to VS$, which assigns unique virtual space to each object. Let the value of $OVS(o) = f(o)$. If we want to grant access of type $a \in A$ to object $o$ for subject $s$, $SVS_a(s)$ must contain $f(o)$. The presence of the virtual space $f(o)$ in $SVS_a(s)$ represents capability to access the object $o$.

Hierarchy in capabilities means the ability to access a group $G \in O^*$ of objects using one capability. This can be achieved by adding new virtual space $x \in VS$ for each such group of objects, i.e. $\forall o \in G : OVS_1(o) = OVS(o) \cup x$ for each object in group. If we want to grant access of type $a \in A$ to group $G$ of objects for subject $s$, $SVS_a(s)$ must contain $x$, i.e. the access is granted when $x \in SVS_a(s)$

## 3.6 Transforming Security Levels to the VS Model

Now, we discuss the possibility to implement security levels [Den76] using virtual spaces. Each subject $s \in S$ is assigned a security level $sl : S \rightarrow L$. Each object $o \in O$ is assigned a security level $ol : O \rightarrow L$. Access is granted when, for example, $sl(s) \geq ol(o)$. We will show two equivalent approaches. It is on the reader to choose the one which better fits his purposes.

Assume the function $f : L \rightarrow VS$ assigns unique virtual space to each level.

Approach 1: Let the value of $OVS(o)$ be $f(ol(o))$. Now we must assign this virtual space to all subjects which have $sl(s) \geq ol(o)$,thus

$$SVS_a(s) = \bigcup_{l=min(L)}^{sl(s)} f(l)$$

.

Approach 2: Let the value of $SVS_a(s)$ be $f(sl(s))$. Now we must assign this virtual space to all objects, which have $sl(s) \geq ol(o)$, thus

$$OVS(o) = \bigcup_{l=ol(o)}^{max(L)} f(l)$$

.

The transformation for the another case $sl(s) \leq ol(o)$ is straightforward.

## 3.7 Transforming Compartments to the VS Model

Compartments [BLP75] are the access control mechanism which separates the subjects by their competencies in the system. Let the $C$ is a set of all compartments in the system. Access of subject $s \in S$ to object $o \in O$ is granted, if the set of compartgments of accessed object $C_o \in C^*$ is a subset of the set of compartments of subject $C_s \in C^*$, i.e. $C_o \subset C_s$.

To achieve this behaviour of VS model, we must assign an unique virtual space to each subset of the set of compartments. Assume the function $f : C^* \rightarrow VS$, which assigns unique virtual space to each subset of $C$. Let the value of $OVS(o)$ be $f(C_o)$ for each object in the system. Finally, treat the subjects as objects, i.e. let the $SVS_a(s)$ be $f(C_s)$.

## 3.8 Transforming the Bell - LaPadula Mandatory Access Control to the VS Model

Using the hints from the previous sections, we are now able to transform the Bell - LaPadula MAC model [BLP75] into the VS model by joining the particular VS models. To assure the data flow constraints of the BLP MAC model are satisfied, we must carefully design the security level model. This section is not meant as an exhaustive guide and it assumes the reader is familiar with the [BLP75].

### 3.8.1 Access Types

access types are: **e** (empty), **r** (read), **a** (append), **w** (write). $A = \{e, r, a, w\}$

### 3.8.2 Dominancy

Each object in the system have security level (security pair), which consists of **security level** AND **compartment**. Security pair $(S_1, C_1)$ dominates the pair $(S_2, C_2)$ when $S_1 \geq S_2$ AND $C_1 \supset C_2$.

Because the VS model defines separate SVS functions for each access type, we must have the access type $a \in A$ in mind, when transforming the particular dominancy test.

We will separately made the VS model for security levels $(VS_1, OVS_1, SVS_{1a})$ and the VS model for compartments $(VS_2, OVS_2, SVS_{2a})$. The values of function $SVS_{1b} : S \to VS^*; SVS_{2b} : S \to VS^*$ must be $\emptyset$ for all $b \in A; b \neq a$.
Then we will join the VS models 1 and 2 using AND. This will simulate the behaviour of given dominancy test for given access type.

### 3.8.3 ss-Property

The ss-property can be transformed to the VS model using OR on the VS models for dominancy and access types read and write.

### 3.8.4 *-Property

Trusted subjects can be transformed to the VS model, for example, as the extreme variant of the access control list. The group of trusted subjects has one virtual space $x \in VS$ in the $SVS_a$ function. All objects have the same virtual space in the $OVS(o)$.

We will join this model using OR with the VS models for dominancy and access types read, write, and append.

### 3.8.5 ds-Property

This property can be transformed to the VS model using the instructions for transforming the ACLs or capabilities.

### 3.8.6 Joining the Models Together

Now we must join the particular models (for each property) to the final VS model. We join them using AND.

# 4 Implementation

In this section, we describe the implementation of the ZP security framework, named Medusa DS9 [ZPO2000]. The implementation is based on the earlier work [ZP99]. We assuem in this section, that the reader is familiar with UNIX[5].

## 4.1 Features of the Implementation

The ZP security framework is implemented on operating systems based on Linux 2.2 or newer. The resources of the operating system are covered by the abstractions of the ZP security framework as follows.

The set of subjects ($S$) contains all processes and threads. Objects ($O$) are all files, processes, and IPC (inter-process communication) objects. The architecture supports easy addition of new object types. There are three defined access types of subjects to objects:

$$A = \{read, write, see\}$$

read: this access type represents the data flow from an object to a subject. For files, this means the opening of a file for reading. For processes, this access type is unused. Because UNIX treats the directories as files, the directory lookup accesses can be viewed as the read accesses to the filesystem object, which represents the given directory.

write: this access type represents the data flow from a subject to an object. For files, this means the opening of file for writing. For processes, this means the sending of signals. Because UNIX treats the directories as files, the directory modify access can be viewed as the write access to the filesystem object which represents the given directory.

see: this access type represents the access of a subject to an object which is neither read nor write. If the subject cannot do access of type 'see' to some object, this object is hidden for the subject. For files, this means the ability to see the files in the directory, for processes this means the ability to see the processes in the process table.

The implementation supports the finite set of virtual spaces. Let the number of supported virtual spaces is $N$. The membership of an object in particular virtual spaces is represented by bit array[6] with $N$ elements, stored

[5]UNIX is the registered trademark of X/Open Company Limited

[6]array of binary digits with elements TRUE or FALSE

at the memory along with the operating system data for given object. The set of abilities of subject is represented by set of bit array with $N$ elements, stored at the memort along with the operating system data for given subject. The presence of TRUE at $n$-th position in the bit array represents the membership or ability to access the $n$-th virtual space; $n \in \{1, ..., N\}$.

## 4.2 Security Decision Center

The SDC implemented in Medusa DS9 is an user-space daemon process named Constable. It communicates with the kernel using character device. Messages from the kernel are sequentially read from the device and the decisions and VS modifications are sent to the device. User space implementation allows kernel changes to be simpler and smaller and thus easier to port to new versions of the Linux kernel and to be more flexible. Thus, improvements to the authorization server should not require changes of the kernel.

Each message consists of message ID, request or reply type, answer from the SDC (allow, deny, silently skip the access or let the kernel decide) and additional data which are interpreted according to the request/reply type and usually carry the description of the subject and object of the access.

The decisions of Constable are defined by the event handling program which is loaded at its startup. The programming language used to describe the given security policy is somewhat similar to C language. Each occurence of access is treated as an event which causes the given subroutine to be started. The subroutine may form the response, modify the VS sets of subject and object, log the occurence of event (access) and any number of additional data and change the internal variables used, for example, to implement a state machine in the program.

The initial VS definition for subjects is done at the time of their creation which causes the special event "fork" in the configuration program of Constable to be executed. The initial definition of VS for objects is done with the pseudo-event "set".

Other supported events [ZPO2000] are "access", "create", "link", "unlink", "symlink", "mkdir", "rmdir", "mknod", "rename", "truncate", "permission", "exec", "init", "fork", "sexec", "setuid", "kill", "ptrace", "capable" and "syscall". Their exhaustive description can be found in the documentation of the Constable [ZPO2000].

### 4.3 An Example of Configuration File

```
/* set the filesystem VS */

recursive for set "/"        vs=0b001;
recursive for set "/bin"     vs=0b010;
recursive for set "/usr/bin" vs=0b010;
        recursive for set "/home"    vs=0b100;

/* set the process VS sets */
for exec "/bin/login" {
log "login executed with pid " pid;
vs  = 0b100; /* it is ‘user’ process */
vsr = 0b111; /* can read everything */
vsw = 0b100; /* can write only to /home */
}
/* thus all processes, started from "/bin/login"
 * will be unable to modify system parts;
 * the exec operation will be catched by the SDC,
 * further access operations are catched by the VS model.
 */

/* and this operation is catched by the SDC */
for unlink "/etc/passwd"
answer = NO;
```

## 5 Conclusions and Further Works

The ZP security framework and its VS model, unlike the common security models, is easy to implement. The implementation requires only small changes in the target computer system, and the permission checks can be easily implemented by logical operations which are usually the fastest operations on today processors. Despite of these properties, the ZP security framework can mimic the behaviour of almost any access control model. Because of its simplicity, it was implemented and tested in practice and proven to be useful.

The Medusa DS9 is being used for a long time worldwide. Besides other applications, it is in use by several Internet Service Providers and universities.

There are some possible extensions both to the ZP security framework and to the Medusa DS9. The framework can be extended by adding rewriting rules for more existing security models and by enhancing the VS formalism and optimisations. The objects, covered by the Medusa DS9, can include more system objects in the future (e.g. routing tables). The internal design is being cleaned at the present to achieve a better internal structure. This step will be followed by the ports of Medusa DS9 to another operating systems (beginning with OpenBSD and NetBSD) and other platforms (including Sparc and Alpha, possibly MIPS). The main problem reported by users is the lack of the configuration tools. GUI based configuration tool will come with the new generation of Constable. This Constable will also contain support for RBAC [SCFY96] which is the final step in our effort for painless administration of Medusa.

## Acknowledgements

We would like to thank Jiří Šafařík and Branislav Steinmüller for their helpful comments. We would also like to thank Martin Očkaják for his untiring work on Medusa.

# References

[BLP75] Bell, D.E. and LaPadula, L.J. "Secure Computer Systems: Unified Exposition and Multics Interpretation." Technical report MTR-2997, MITRE (1975).

[Den76] Denning, D.E. "A Lattice Model of Secure Information Flow." *Communications of the ACM* (1976).

[GD72] Graham, G. and Denning, P. "Protection - principles and practice" *Spring Joint Computer Conference* AFIPS Press (1972).

[Lam71] Lampson, B.W. "Protection." *5th Princeton Symposium on Information Science and Systems* (1971).

[SCFY96] Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E. "Role Based Access Control Models" *IEEE Computer, Volume 29, pages 38-47* (1996).

[ZP99] Zelem, M. and Pikula, M. "Increasing the Security of OS Linux" *Bachelor Thesis, Faculty of Electrical Engineering and Information Technology, Slovak University of Technology in Bratislava* (1999) (in Slovak).

[ZPO2000] Zelem, M., Pikula, M., Očkaják, M. "Medusa DS9 Homepage" *http://medusa.fornax.sk/* (2000).