

# Segurança em Redes

14 de Novembro de 2020

## **TP2 - Grupo G03**

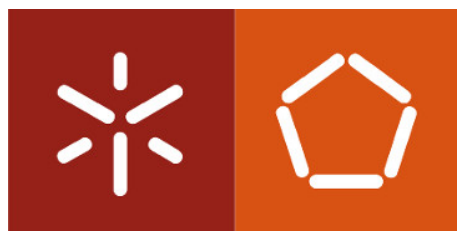
---

a83899	André Morais
a85367	Francisco Lopes
a83819	Miguel Oliveira
a84727	Nelson Faria
a85853	Pedro Fernandes
a84485	Tiago Magalhães

---

## Controlo de Acesso

---



Mestrado Integrado em Engenharia Informática  
Universidade do Minho

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>2</b>
<b>2</b>	<b>Lattice e a sua importância para modelos de controlo de acesso</b>	<b>3</b>
<b>3</b>	<b>Modelo Bell-LaPadula</b>	<b>5</b>
3.1	Conceção do modelo Bell-Lapadula para um sistema de informação usado num ambiente universitário . . . . .	5
<b>4</b>	<b>Implementação do modelo numa infraestrutura CIT</b>	<b>7</b>
<b>5</b>	<b>Conclusão</b>	<b>14</b>

# 1 Introdução

Este trabalho prático surge no âmbito da UC de Segurança de Redes, mais concretamente no que toca à aplicação de sistemas de controlo de acesso em ambientes ”**multi-user**”.

Estando perante um sistema que seja utilizado por múltiplos intervenientes, torna-se necessária a existência de algum tipo de controlo no que toca ao acesso de cada um desses utilizadores aos recursos existentes na infraestrutura, de modo a garantir a segurança e o bom funcionamento da mesma de acordo com a finalidade para a qual a mesma foi idealizada. Importa garantir, para cada um dos usuais utilizadores do sistema, a existência de confidencialidade, integridade e disponibilidade aquando do uso dos serviços do sistema qualquer que seja o seu papel (grupo) na infraestrutura.

No nosso contexto, em particular, é-nos pedido para idealizar a modulação de um sistema de controlo de acesso para um sistema de informação tipicamente usado num ambiente universitário, baseado no modelo BLP (Bell-LaPadula). De acordo com o que nos é solicitado no enunciado do problema em questão, existirão, neste contexto, 3 níveis de segurança (**P (public)**, **C (confidential)**, **SC (strictly confidential)**) e ainda 2 categorias (**AcS (academic services)** e **ScS (scientific services)**).

## 2 Lattice e a sua importância para modelos de controle de acesso

Em matemática, especialmente na teoria da ordem e em álgebra, um reticulado ou *lattice* é uma estrutura  $L = \langle L, \leq \rangle$  tal que  $L$  é parcialmente ordenado (os estados têm de ter as seguintes propriedades: reflexivo, transitivo e antissimétrico) por  $\leq$  e para cada dois elementos  $a, b$  de  $L$  existe supremo e ínfimo de  $a, b$ .

O modelo de fluxo de informação assente nos axiomas *Denning's* serve de base para muitos modelos de controle de acesso, por isso qualquer modelo que se baseia neste modelo segue uma estrutura de *lattice*.

### Information Flow Models: Denning's Axioms

**Denning's axioms:**

- Axiom 1:**  $SC$  is finite
- Axiom 2:** The may-flow relation  $\rightarrow$  is a partial order
- Axiom 3:**  $SC$  has a least element w.r.t.  $\rightarrow$
- Axiom 4:**  $\oplus$  is a least upper bound operator

**Proposition 1**  
Any information flow model that satisfies the Denning's axioms is a lattice.

Figura 1: *Denning's axioms*

A *lattice* em modelos de controle de acesso pode ser expressa da seguinte forma: cada objeto e sujeito tem um limite inferior e um limite superior de direitos de acesso, sendo os objetos tais como: recursos, aplicações e computadores e sujeitos tais como: indivíduos, grupos e organizações. Por exemplo, se dois sujeitos  $A$  e  $B$  precisam de acesso a um objeto, o nível de segurança é definido como o encontro dos níveis de  $A$  e  $B$ . Em outro exemplo, se dois objetos  $X$  e  $Y$  são combinados (*join*), eles formam outro objeto  $Z$ , ao qual é atribuído o nível de segurança formado pela junção dos níveis de  $X$  e  $Y$ . Podemos ver nestes axiomas semelhanças com o modelo de fluxo de informação. A **figura 1** dá o exemplo de uma *lattice* nestes contornos.

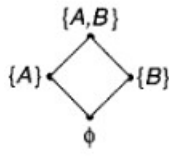


Figura 2: Exemplo de uma *lattice*

Na figura acima podemos ver que a *lattice* está limitada superiormente e inferiormente, bem como está parcialmente ordenado, uma vez que a criação do objeto  $\{A, B\}$  resulta de  $A$  e  $B$  serem incomparáveis e por isso o resultado de  $A \text{ join } B$  corresponder a  $\{A, B\}$ .

Foi com base nesta estrutura e no modelo e fluxo de informação, dado que estamos a trabalhar com o controlos de acesso a informação que o *Bell-Lapadula* foi formalmente criado.

### 3 Modelo Bell-LaPadula

O modelo *Bell-LaPadula* é baseado em *labels*, isto é, um par com um nível e categoria, por exemplo um documento pode ter *label* (Top Secret, crypto,nuclear) em que contém informação muito sensível em relação às categorias *crypto* e nuclear. Com base nisto um modelo pode ter  $(\#compartimentos) * 2^{\#niveis.de.seguranca}$  *labels*.

Este modelo apresenta as seguintes propriedades :

- ***The Simple Security Property*** : Um sujeito com uma determinada *label* atribuída, não pode ler para *labels* com um nível superior;
- ***The \*(star) Security Property*** : Um sujeito com uma determinada *label* atribuída, não pode escrever para *labels* com um nível inferior.

Para saber se uma *label* se apresenta a nível superior, basta ver a dominância através da seguinte propriedade, dadas duas *labels*  $L1 = (S1, C1)$  and  $L2 = (S2, C2)$ , dizemos que  $L1 \leq L2$  se :

- $S1 \leq S2$
- $C1 \subseteq C2$ .

Como  $\leq$  é de ordem parcial é possível que existam *labels* que não possam ser comparáveis.

#### 3.1 Conceção do modelo Bell-Lapadula para um sistema de informação usado num ambiente universitário

O modelo de Bell-LaPadula apresentado no trabalho, segue as seguintes categorias: **AS** – Academic Services e **ScS**- Scientific Service, contendo também as seguintes labels: **P** - public, **C** - confidential e **SC** - strictly confidential. Num contexto universitário, um aluno estará classificado como (C,AS), o que se traduz numa posição hierárquica inferior à do professor, classificado como (C,{AS,ScS}). De acordo com a equação apresentada anteriormente iremos ter 16 *labels*. O modelo contruído com base nas propriedades apresentadas anteriormente, foi o seguinte:

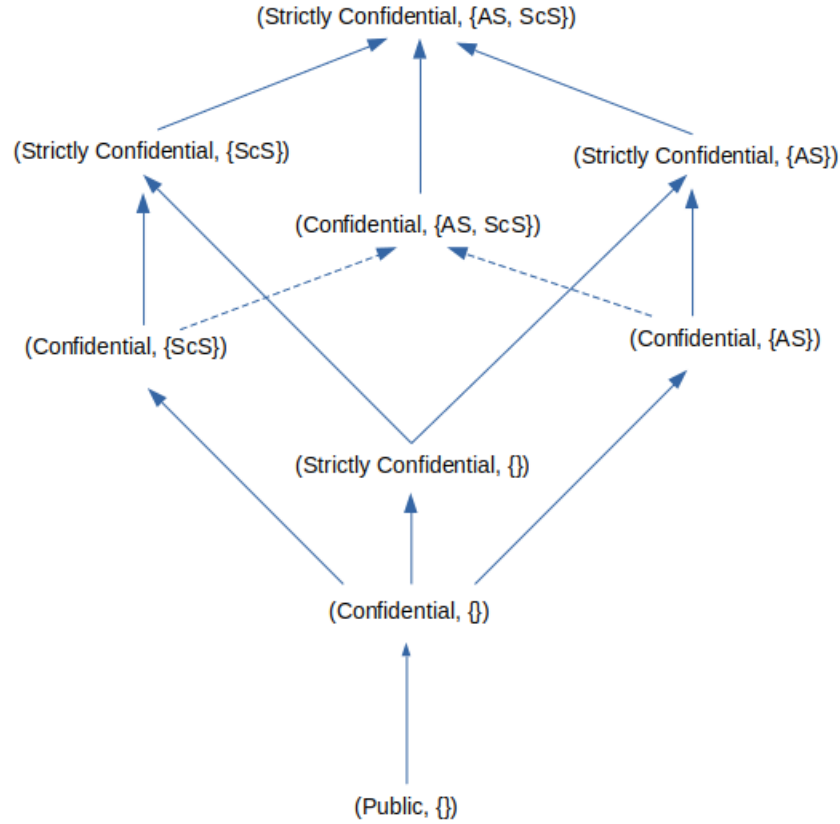


Figura 3: Modelação do sistema

Neste modelo construído (**Figura 3**) não estão presentes as combinações de *labels* com nível de segurança públicas, já que a um nível público não faz sentido existir restrições e também neste sistema o principal objetivo é garantir restrições entre alunos e professores, motivo esse que nos levou a retirar essas combinações do modelo.

Quanto à questão presente no enunciado se era possível um aluno enganar um professor, assumindo duas *labels* **Aluno** =  $(C, AS)$  e **Professor** =  $(C, \{AS, ScS\})$ , podemos afirmar que **Aluno**  $\leq$  **Professor**, através do seguinte  $C \leq C$  e  $AS \subseteq \{AS, ScS\}$ , assim, podemos esclarecer que, como o aluno está hierarquicamente num nível inferior ao professor, o mesmo não pode ter acesso aos ficheiros de nível superior (**{No read-up}**), no entanto pode escrever em níveis superiores. Deste modo o modelo confere confidencialidade, porém o aluno tem permissões para escrever numa hierarquia superior, constituindo assim um problema de integridade.

## 4 Implementação do modelo numa infraestrutura CIT

Para implementar automaticamente este sistema num ambiente típico poderíamos, por exemplo, usar um **script** para criar grupos e inserir os utilizadores no mesmo. É assumido que os utilizadores já estão criados na máquina em questão, que está a correr algum flavor de Linux.

Estes grupos serão o equivalente aos níveis de segurança e as categorias. Cada conjunto possível de níveis e categorias é um grupo único. Assim sendo, neste caso, ficamos com os seguintes grupos:

```
public;
C;
C_AS;
C_SCS;
C_AS_SCS;
SC;
SC_AS;
SC_SCS;
SC_AS_SCS.
```

Indicamos os grupos e as associações num ficheiro muito simples, que no nosso exemplo conceptual se chama `acl.txt`:

```
groups
public
C
C_AS
C_ScS
C_AS_SCS
SC
SC_AS
SC_ScS
SC_AS_SCS
users
teacher C_AS_SCS
student C_AS
end
```

A primeira linha indica que se seguem os grupos, os quais são enumerados linha a linha, até encontrarmos a palavra-chave “user”. Esta indica-nos que de seguida, em cada linha, é indicado em que grupo cada utilizador deve ser inserido. Por fim, uma linha com a palavra “end” indica o fim do ficheiro.



Sendo assim, precisamos de um script que leia este ficheiro e aplique os comandos necessário. Claro, é necessário correr o script com privilégios elevados.

```
#!/usr/bin/env bash

$(sudo apt-get install acl)

read groups

read group
while [ $group != "users" ]
do
    $(sudo addgroup $group --force-badname)
    read group
done

read users

read user
while [ "$user" != "end" ]
do
    sa=($user)
    $(sudo adduser ${sa[0]} ${sa[1]})
    read user
done
```

Para os ficheiros em si, podemos usar o comando **sudo setfacl -m g:grupo:permissões path** para definir as permissões para cada grupo. Por exemplo, para um ficheiro público chamado “**public.txt**”, podemos definir que o grupo **C** tem permissão para leitura.

```
sudo setfacl -m g:C:r public.txt
```

Com esta lógica, definimos 3 ficheiros de teste: “**public.txt**”, “**student.txt**” e “**teacher.txt**” que pertencem aos grupos **public**, **C\_AS** e **C\_AS\_SCS**, respetivamente. Usando o comando **sudo getfacl path** podemos verificar as permissões de cada ficheiro:

```
pedro@pedro-N751JK:~/Documents/SR$ sudo getfacl public.txt
# file: public.txt
# owner: root
# group: public
user::rw-
group::rw-
group:C:r--
group:C_AS:r--
group:C_ScS:r--
group:C_AS_ScS:r--
group:SC_AS:r--
group:SC_ScS:r--
group:SC_AS_ScS:r--
group:SC:r--
mask::rw-
other::---
```

Figura 4

```
pedro@pedro-N751JK:~/Documents/SR$ sudo getfacl student.txt
# file: student.txt
# owner: student
# group: C_AS
user::rw-
group::rw-
group:public:-w-
group:C:-w-
group:C_ScS:-w-
group:C_AS_ScS:r--
group:SC_AS:r--
group:SC_ScS:-w-
group:SC_AS_ScS:r--
group:SC:-w-
mask::rw-
other::---
```

Figura 5

```
pedro@pedro-N751JK:~/Documents/SR$ sudo getfacl teacher.txt
# file: teacher.txt
# owner: teacher
# group: C_AS_SCS
user::rw-
group::rw-
group:public:-w-
group:C:-w-
group:C_AS:-w-
group:C_ScS:-w-
group:SC_AS:-w-
group:SC_ScS:-w-
group:SC_AS_ScS:r--
group:SC:-w-
mask::rw-
other::---
```

Figura 6

Podemos testar a nossa implementação, por exemplo, como “**student**” tentar ler o ficheiro “**teacher.txt**”, algo que não deve ser possível.

```
student@pedro-N751JK:/home/pedro/Documents/SR$ less teacher.txt
teacher.txt: Permission denied
```

Figura 7

Podemos também tentar ler o ficheiro “**student.txt**” como “**teacher**”, que deve ser permitido.

```
teacher@pedro-N751JK:/home/pedro/Documents/SR$ cat student.txt
ola ola
tudo bem
e quê
Que fazes da vida
teste teste
```

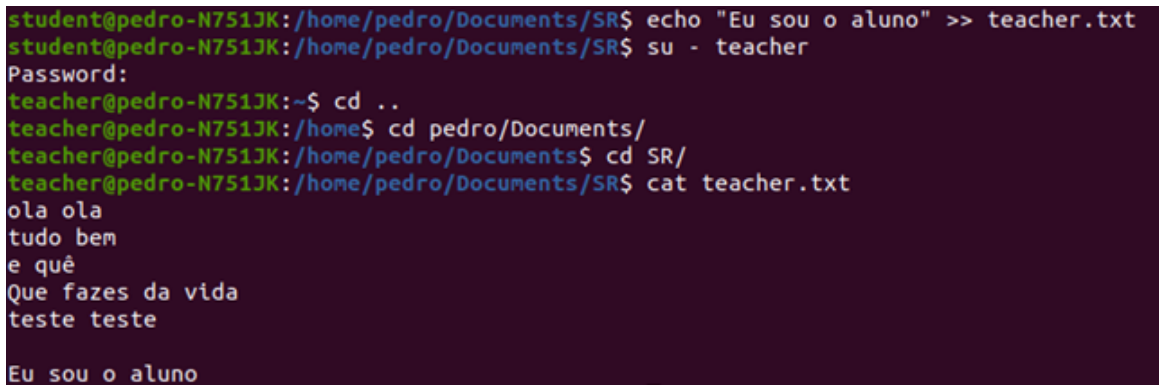
Figura 8

No entanto, para a escrita, os papéis são invertidos. O professor não deverá conseguir escrever no ficheiro “**student.txt**”.

```
teacher@pedro-N751JK:/home/pedro/Documents/SR$ cat student.txt
ola ola
tudo bem
e quê
Que fazes da vida
teste teste
teacher@pedro-N751JK:/home/pedro/Documents/SR$ echo "Eu sou o professor" >> student.txt
-bash: student.txt: Permission denied
```

Figura 9

Mas o aluno deve conseguir escrever, às “cegas”, no ficheiro “**teacher.txt**”.



```
student@pedro-N751JK:/home/pedro/Documents/SR$ echo "Eu sou o aluno" >> teacher.txt
student@pedro-N751JK:/home/pedro/Documents/SR$ su - teacher
Password:
teacher@pedro-N751JK:~$ cd ..
teacher@pedro-N751JK:/home$ cd pedro/Documents/
teacher@pedro-N751JK:/home/pedro/Documents$ cd SR/
teacher@pedro-N751JK:/home/pedro/Documents/SR$ cat teacher.txt
ola ola
tudo bem
e quê
Que fazes da vida
teste teste

Eu sou o aluno
```

Figura 10

Claro, cada um pode tanto ler como escrever no seu ficheiro respetivo. Esta solução, apesar de funcional, deixa-nos com um grande problema. Cada vez que um ficheiro é criado, é preciso definir corretamente todas as permissões, para cada grupo.

Seria possível criar um *script* para definir as permissões, mas se o número de grupo for relativamente grande, é muito propenso a erros. Era também necessário alguém com permissões elevadas para mudar estas mesmas do ficheiro, sempre disponível. Para além disso, se for necessário criar um novo grupo, todos os ficheiros existentes no sistema, que podem ser centenas, milhares ou muitos mais, precisam de ser atualizados, manualmente ou com um *script*.

Tudo isto significa que este método é apenas exequível num cenário em que os utilizadores criam os ficheiros numa pasta protegida apenas para eles e em intervalos regulares o **sysadmin** copiaria os ficheiros para o local adequado, com as permissões corretas.

Estes problemas poderiam ser mitigados com o uso de **SELinux**. Este é uma camada de segurança para sistemas Linux desenvolvida inicialmente pela **NSA**. É uma implementação da arquitetura **MAC**, que mais ou menos tentamos emular com o sistema de permissões default do Linux.

Não criamos uma implementação deste sistema pois a configuração inicial é bastante complexa, visto que está diretamente ligada ao **kernel** e é direcionada a ambientes mais hostis, pelo que user friendliness não estava no topo da lista dos interesses dos desenvolvedores. No entanto, fizemos alguma pesquisa para tentar perceber de forma razoável como seria implementado este sistema.

A instalação dos pacotes necessário vai depender da distro instalada na máquina em questão. Pode variar entre pré-instalado, a alguns **apt-get**, até à instalação de uma nova imagem de kernel.

Em **SELinux**, cada objeto, que pode ser um ficheiro, uma diretoria, um processo ou até uma porta, entre outros, tem uma label associada. Esta label costuma ser representada como “**user:role:type:level**”, sendo que o “**level**” é opcional.

No modelo **BLP**, “**level**” corresponde ao nível de segurança e categorias necessários para

aceder a esse objeto. Depois de instalado e ativado o **SELinux**, o utilizador root terá que se incluir no role “**sysadm\_r**”, que como o nome indica, é o sysadmin da máquina.

Isto poderá ser feito usando o comando “**newrole -r sysadm\_r**”. De seguida deve ser corrido o comando “**make -C etc/selinux relabel**”, para evitar problemas que nos deixariam completamente bloqueados depois de fazer um reboot. Neste momento, estamos prontos para configurar o sistema. Neste ponto, poderíamos fazer algo parecido com o primeiro exemplo.

Precisaríamos de um **script** que, primeiramente, correria o comando “**sepolgen /path/to/binary**”. Este gera ficheiros exemplo para criar novos tipos. Estes tipos são associados aos ficheiros, na sua label. Um dos ficheiros criados, **app.te** precisa de ser editado e adicionar linhas que representam a informação necessária. Por exemplo:

```
type public_t;
files_type(public_t);
```

Podemos também opcionalmente acrescentar mais informação que restringe as operações possíveis sobre um dado tipo de ficheiros. É uma prática comum acrescentar “**\_t**” no fim do nome dos tipos.

O **script** poderia então ler de um ficheiros todos os tipos que são necessários criar, e depois, de formar semelhante, criar os roles necessários. Estes podem ser criados com o comando, com o exemplo de “**teacher**”.

```
sepolcity generate --teacher -n NAME
```

Este comando, à semelhança do anterior, vai gerar 5 ficheiros. O **script** tem então que fazer “**make**” na pasta destino, e “**semodule -i**”, o que vai acrescentar os novos roles.

Depois de um reboot, outro script teria de criar os User **SELinux** necessários, usando o comando “**useradd**”. Aqui podemos definir o seu nível de segurança assim como as categorias a que tem acesso.

Por fim, o script tem de editar o ficheiro “**etc/selinux/users/**”, e no final do mesmo, acrescentar linhas do com o formato:

```
user user\name roles { roles };
```

Isto indica-nos que o user “**user\_name**” pode ser incluído nos roles entre chavetas. Para aplicar, o script faz “**make -C etc/selinux load**”. Por últimos, o **script** teria de correr o comando roles

```
role role_id types type_id
```

o que permite definir que roles têm direito a aceder que tipo de ficheiros. No nosso caso específico, podemos ter apenas um tipo de ficheiro e associar-lhe todos os roles, visto que estamos mais focados no aspeto de níveis e segurança e categorias.

Tudo o que resta fazer é dar labels aos ficheiros, sendo que este sistema tem uma diferença fundamental para o primeiro que analisamos. Quando um ficheiro não está explicitamente acessível por um dado role, o mesmo fica completamente fechado. Enquanto um

ficheiro que acaba de ser criado normalmente é acessível a toda a gente, com o **SELinux**, um ficheiro que acabou de ser criado não é acessível a ninguém.

Assim, sempre que um ficheiro é criado, basta dizer qual é o seu nível de segurança e categoria, simplificando bastante a gestão dos ficheiros no futuro.

No entanto, o setup é bastante mais complicado, e acreditamos até que faltem alguns passos da instalação. A documentação sobre este sistema é francamente muito fraca. Não conseguimos encontrar um único guia que explicasse de maneira relativamente coerente como instalar e configurar o **SELinux** do início ao fim. Tivemos que amalgamar bastante informação de vários sites diferentes, o que pode ter resultado em algumas falhas. No entanto, este será o procedimento geral, e como percebemos, poderia ser relativamente automatizado, que era o objetivo da questão.

## 5 Conclusão

Tendo em conta o principal objetivo deste trabalho, acreditamos ter conseguido idealizar um possível modelo de segurança para um Sistema de informação em contexto universitário, modelo esse elaborado com os princípios inerentes ao BLP, que já fomos referindo ao longo deste mesmo relatório. De um modo geral, não foram sentidas grandes dificuldades em entender o modelo BLP em si, assim como na própria "construção do lattice" para este caso particular.

No que toca à última parte do presente relatório, em que era pedido para refletirmos sobre a implementação do modelo numa típica infraestrutura CIT, a princípio tivemos alguma dificuldade em entender o que era pedido, contudo pensamos ter conseguido chegar a bom porto, tendo alcançado um exemplo bastante interessante sobre o funcionamento do modelo na prática.

Nota para o facto de que foi levada a cabo uma pesquisa sobre o SELinux que servirá para o caso de querermos usar esta ferramenta Linux para implementar o modelo numa infraestrutura de forma automatizada, o que permitiria ainda mitigar alguns problemas que foram descritos nessa secção (secção 4) do presente relatório.

## Referências

- [1] <http://www.cs.cornell.edu/courses/cs5430/2011sp/NL.accessControl.html>
- [2] <http://www.cs.cornell.edu/courses/cs5430/2011sp/NL.accessControl.html>
- [3] <http://www.cs.unc.edu/~dewan/242/f96/notes/prot/node1.html>
- [4] Sandhu, Ravi S. "Lattice-based access control models." *Computer* 26.11 (1993) 9-19.