CS4371 LAB 2

Dhruve Mistry

Due 2/16/23

Understanding AC lattice and how different levels connect.
Classifying different Linux based Access Controls.
Making an ACM and creating ACL & CL from the matrix.
Creating a C++ program to provide AC on a file.
Understanding the iptables and creating a firewall through Linux.

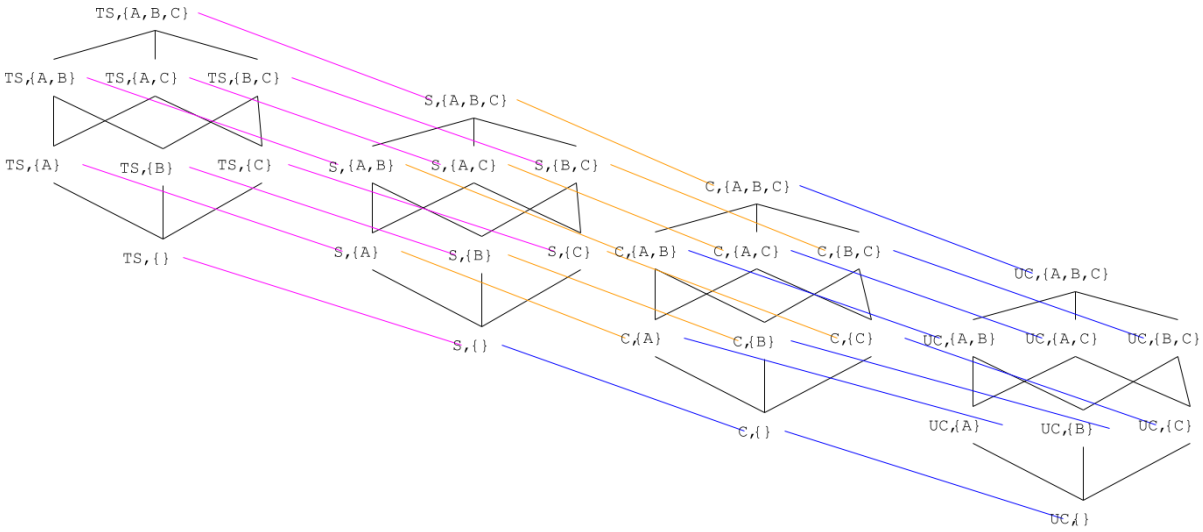
Mistry, Dhruve A
d_m704@txstate.edu

Task 1 – Lattice

We are instructed to make a lattice, sort of a matrix, given four security levels (ordered highest to lowest), which I've made acronyms for them noted on the level respectively,

- TOP SECRET (TS)
- SECRET (S)
- CONFIDENTIAL (C)
- UNCLASSIFIED (UC)

Given these four levels, all have access to objects A, B, and C on their own level.



We are to identify if in these hypothetical scenarios, can we allow this to happen. If so, what permissions they possess. We are using the BLP model for access control, which is a *read up and write down* structure.

Questions and answers regarding task 1 is on next page.

(A)Alice is(TS, {AC}), wants to access(S, {A,C}).

Alice can only read.

(B)Bob is (C, {C}), wants to access (UC, {A,B}).

Bob cannot read, write nor execute.

(C)Claire is (S, {C}), wants to access (UC, {C}).

Claire can only read.

(D)Harry is (TS, {A,C}), wants to access (C, {B}).

Harry cannot read, write, nor execute.

(E)Loma has no clearance, wants to access (C,{B}).

Loma can write only.

Task 2 – Classify Linux AC

(A) In a Linux system, a file's permission is set by the owner of the file.

- Policy: DAC
- Creator: User
- Owner: User
- System: Linux OS
- Admin: User
- Who decides permission: User who owns the file

(B) In a software repository, a file can be accessible to an agent based on the owner's choice.

- Policy: DAC
- Creator: Author of the file
- Owner: Author of the file
- System: Software repo
- Admin: Author of the file

(C) In a classified NSA database, only generals with top secret clearance can search in the database.

- Policy: RBAC
- Creator: Author of the file
- Owner: Author of the file
- System: NSA database
- Admin: General

Task 3 – ACM -> ACL & CL

There are three users on a computer system, Alice, Bob, and Cyndy. Alice owns file a, and Bob and Cyndy cam read that file. Bob owns file b, and Cyndy can read and write to the file b, but Alice can only read that file. Cyndy owns file c, but neither Alice nor Bob can read nor write to file c. If a user owns their respective file, they can also execute said file.

(A) First, we are asked to create the Access Control Matrix (ACM) for the system.

$$\begin{bmatrix} & A & B & C \\ Alice & rwx & r & - \\ Bob & r & rwx & - \\ Cyndy & r & rw & rwx \end{bmatrix}$$

(B)

Part a) Now that we have the ACM, we can start to create our ACL,

Alice -> [(A, rwx), (B,r)]

Bob -> [(A,r), (B,rwx)]

Cyndy -> [(A,r), (B,rw), (C, rwx)]

Part b) We can also create the CL from the ACM,

A -> [(Alice, rwx), (Bob, r), (Cyndy, r)]

B -> [(Alice, r), (Bob, rwx), (Cyndy, rw)]

C -> [(Cyndy, rwx)]

(C) Cyndy allows Alice to read file c, and Alice removes Bob's ability to read file a. Show the new ACM.

$$\begin{bmatrix} & A & B & C \\ Alice & rwx & r & r \\ Bob & - & rwx & - \\ Cyndy & r & rw & rwx \end{bmatrix}$$

Task 4 – C++ Function AC

The example code given is designed for a permission list of a file in Linux.

Note; u=7 means rwx, u=5 means r-x. All permissions are in octal (base 8).

```
typedef struct {
     unsigned int uid; // owner id
     unsigned int gid; // group id
     unsigned char u;  // owner's permission
     unsigned char g;  // group's permission
     unsigned char o;  // other's permission
} Permission;
```

Permission check procedure:

1) A user requests an operation p on a file f.
2) If user is owner of the file, operation will be checked against the owner's permission of the file. The result will either grant or deny.
3) Otherwise, if the user is not the owner but in the group of the file, the operation will be checked against the group's permission of the file The result is either grant or deny.
4) Otherwise, if the user is neither the owner nor the in the group of the file, the operation will be checked against the other's permission of the file. The result is either grant or deny.

Our goal – Write a C/C++ function "int accesscheck(unsigned int uid, unsigned int gid, unsigned int p, int f){}" to enforce access control in Linux.

Args:

- uid and gid are the user id and group id of who requests access.
- f is the file id.
- p is the requested operation. E.g., p=7 means three operations rwx, p=6 means two operations rw-, p=1 means one operation --x.

Function returns 1 if access is granted or 0 if denied.

Assume: "Permission getPermission(int f)" can return the permission of file f.

The code I have produced is submitted in an extra file. It is not here. There is a .txt and .c version submitted. Alternatively, I have only included the snippet with regard to the accesscheck(~) function.

```
int accesscheck(unsigned int uid, unsigned int gid, unsigned int p, int f)
{
    // Is the file even accessible?
    if (p <= 0) return 0; // deny
    // Creates a local copy of permission value
    Permission permission = getPermission(f);
    // First check if the user is owner of the file
    if (permission.uid == uid) {
        if ((permission.u & p) == p) { // Check to use if the user has the permission
            return 1; // grant
        } else {
            return 0; // deny
        }
    }
    // If the user is not the owner, check if user is part of the group that can access.
    else if (permission.gid == gid) {
        if ((permission.g & p) == p) { // Check to use if the user has the permission
            return 1; // grant
```

```
        } else {
            return 0; // deny
        }
    }
    // The user is not owner nor in the group, but could be with other's
permission.
    else {
        if ((permission.o & p) == p) {
            return 1; // grant
        } else {
            return 0; // deny
        }
    }
}
```

Task 5 – iptables

We have a Linux server but without any firewalls.

OS Version: Ubuntu 22.04 LTS (Works on any

Our goal – Show the commands of iptables that add the Linux firewall rule to enforce the protection.

First,

* Only computers from address 172.90.0.0/16 but excluding 172.90.255.0/24 can browse the website that is running on port 8888 in the server.

```
iptables -A INPUT -s 172.90.225.0/24 -j DROP

iptables -A INPUT -s 172.90.0.0/16 -p tcp --dport 8888 -j ACCEPT

iptables -L -n
```

```
vboxuser@DhruveM:~$ sudo iptables -A INPUT -s 172.90.225.0/24 -j DROP
vboxuser@DhruveM:~$ sudo iptables -A INPUT -s 172.90.0.0/16 -p tcp --dport 8888 -j ACCEPT
vboxuser@DhruveM:~$ sudo iptables -l -n
iptables v1.8.7 (nf_tables): unknown option "-l"
Try `iptables -h' or 'iptables --help' for more information.
vboxuser@DhruveM:~$ sudo iptables -L -n
Chain INPUT (policy DROP)
target     prot opt source               destination
DROP       all  --  172.90.225.0/24      0.0.0.0/0
ACCEPT     tcp  --  172.90.0.0/16        0.0.0.0/0            tcp dpt:8888

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
vboxuser@DhruveM:~$
```

---

* We will ping the server ip address. 172.90.10.20

```
iptables -A INPUT -p tcp --dport 22 -s 172.90.10.20 -j ACCEPT

iptables -A INPUT -s 172.90.10.20 -p icmp --icmp-type 8 -j ACCEPT

iptables -A INPUT -p tcp --dport 22 -j DROP

iptables -A INPUT -p icmp --icmp-type 8 -j DROP
```

```
vboxuser@DhruveM:~$ sudo iptables -L -n
Chain INPUT (policy DROP)
target     prot opt source               destination
DROP       all  --  172.90.225.0/24      0.0.0.0/0
ACCEPT     tcp  --  172.90.0.0/16        0.0.0.0/0            tcp dpt:8888
ACCEPT     tcp  --  172.90.10.20         0.0.0.0/0            tcp dpt:22
DROP       tcp  --  0.0.0.0/0            0.0.0.0/0            tcp dpt:22
ACCEPT     icmp --  172.90.10.20         0.0.0.0/0            icmptype 8
DROP       icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 8
```

---

* The server cannot initiate anything to send out

---

iptables –P OUTPUT DROP

iptables -A OUTPUT -m state --state ESTABLISHED,RELATED, -j ACCEPT

```
vboxuser@DhruveM:~$ sudo iptables -L -n
Chain INPUT (policy DROP)
target     prot opt source               destination
DROP       all  --  172.90.225.0/24      0.0.0.0/0
ACCEPT     tcp  --  172.90.0.0/16        0.0.0.0/0            tcp dpt:8888
ACCEPT     tcp  --  172.90.10.20         0.0.0.0/0            tcp dpt:22
DROP       tcp  --  0.0.0.0/0            0.0.0.0/0            tcp dpt:22
ACCEPT     icmp --  172.90.10.20         0.0.0.0/0            icmptype 8
DROP       icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 8

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0            state RELATED,ESTABLISHED
```

---

* The default policy for all chains is ACCEPT

---

iptables -P INPUT ACCEPT

iptables –P FORWARD ACCEPT

iptables -P OUTPUT ACCEPT

Picture on next page

```
vboxuser@DhruveM:~$ sudo iptables -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       all  --  172.90.225.0/24      0.0.0.0/0
ACCEPT     tcp  --  172.90.0.0/16        0.0.0.0/0           tcp dpt:8888
ACCEPT     tcp  --  172.90.10.20         0.0.0.0/0           tcp dpt:22
DROP       tcp  --  0.0.0.0/0            0.0.0.0/0           tcp dpt:22
ACCEPT     icmp --  172.90.10.20         0.0.0.0/0           icmptype 8
DROP       icmp --  0.0.0.0/0            0.0.0.0/0           icmptype 8

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0           state RELATED,ESTABLISHED
```

This concludes task 1-5.

Reminder: task 4 code is located in the .cpp and .txt file submitted.