An introduction to authorization: access control.

Alessandro Tomasi*[1]

[1]Security and Trust, Fondazione Bruno Kessler

October 22, 2018

Abstract

Role-based access control examples; introduction to databases using SQLite.

1 Introduction

In this workshop you will practice with authorization techniques, in particular role-based access control [Fer+01].

For convenience, I also included a summary of the important linux permission commands in Section 1.2. You can find a good introduction and sample exercise on wiki.debian.org/Permissions. Distinct from permissions are capabilities, which we shall not discuss here but you can read up on manpages.org/capabilities. You may also be interested to note that the Bell-LaPadula model is a policy model enforceable in SELinux.

1.1 SQLite and python

SQLite SQLite is a very popular SQL database, with most common SQL commands available. In particular, refer to the documentation for the following topics.

Tables: CREATE or DROP TABLE

Rows: INSERT, SELECT, UPDATE, DELETE

Optionally, you may find it useful to look up the following constraints:

Primary: Every table is created with a default unique integer PRIMARY KEY for each row, called `rowid`. You *may* specify other primary keys if you wish, but this internally points to the default `rowid` anyway.

*altomasi@fbk.eu

Foreign: FOREIGN KEY links columns of tables together so that it is not possible to create invalid entries or change entries on which others depend.

There is also a list of commands specific to SQLite called pragma, but you will probably not need most of them.

Python sqlite3 is a library to execute SQLite commands via python scripts. The basic commands are:

```
import sqlite3
conn = sqlite3.connect('MY_PRECIOUS.DB')
c = conn.cursor()
c.execute('''SQL COMMANDS''')
c.fetchone()[0]
c.fethmany(int)
c.fetchall()
```

It is also good practice to make sure changes are committed, so that other connections can see them (`conn.commit()`); and to close the connection when you no longer need it (`conn.close()`).

You may also wish to have a look at this tutorial for future reference.

If you are wondering why one might want to do this through python, aside from fitting well with the rest of this class, bear in mind that the main purpose of databases is to efficiently store and retrieve data of basic types that is well-organized. You *can* write complex logic and data types into a db structure, but you should ask yourself why you would want to. In particular, there is a strong trade-off between efficiency on the one hand, and the ease with which you can read and maintain the logic flow on the other.

Notation: capital letters are not mandatory, but they are common practice and make it easier to distinguish the parameters from the commands. The triple-double quotation marks are also not mandatory, but in a python script they allow you to write on multiple lines and to use single and double quotation marks without breaking the string.

Example: you can find a worked example of how to create tables and add users in Appendix A. It is based on the 'small university' access control example you saw during the lectures, which you can find on slide 50 of lecture Access Control I (4-AuthZ-2p), available on the course website. You can also find the source code on github.

1.2 Linux permissions

The linux command `ls -l` lists all contents of a directory, with further information. In particular, you will see a group of characters such as in Table 1, which are drawn from the set {`d,r,w,x`}, standing for `d`irectory, `r`ead, `w`rite, and execute. The first character tells you whether the item is a directory; the

-	rwx	rw-	---
directory?	user	group	other

Table 1: Linux permissions example.

other nine characters are in groups of three, one each for user, group, and other.

You can use `chmod i+p` to change the permissions of a file; for instance `o+x` enables *o*ther users to execute.

You can also write these permissions in octal: for instance, `chmod 741` corresponds to `chmod a+rwx,g-wx,o-rw`. Can you see why?

You can create new users with `adduser username` and delete them with `deluser --remove-home username`. Switch to their account with `su - username`.You can add them to a group with `usermod -a -G examplegroup exampleusernames`.

You can see which groups the current user belongs to with `groups` or `id`.

2 Exercises

2.1 Access control with SQLite

Registration You should now be able to modify the web app you created last week to store hashed and salted user passwords in an sqlite database. Create a table called `users` with all the necessary information, and modify yur flask script accordingly.

Tracking and tracing Your task is to design an access control structure for a parcel tracking system designed to maximise the privacy of user data, the accountability for custody of an item, and the traceability of parcels.

Use case scenario A user creates a digital envelope for a parcel and uploads it to the cloud, then takes the parcel to the post office to have it shipped. The user assigns the parcel to a courier. The tracking system calculates a shipping route, but the courier is only told the next shipping destination; the courier does not need to know anything about the sender or the receiver.

When the parcel changes hands, two steps have to take place: the current custodian of the parcel has to assign it to another person, and that person has to accept custody.

Along the way, an inspector should be allowed to check the contents of the digital envelope and terminate hte chain of custody if they determine that something is wrong.

User Wants to send a parcel to a recipient. Prepares a digital document containing all the information about a shipment, such as sender, receiver, contents etc.

Courier Is responsible for the custody of a parcel. Can initiate the transfer to another courier, and can accept custody of transfers initiated to them. Can only see the information strictly relevant to their part of the delivery: only the 'last mile' courier knows the address of the recipient, and no couriers know the address of the sender.

Inspector Can see all the information uploaded to the cloud. Can take charge of parcels and terminate the chain of custody.

There are two separate items to keep track of: the digital envelope saved in the cloud, and the physical parcel and its associated events.

First, identify the roles in this system and the assets. Then, write the permissions associated with each role.

2.2 Security issues

Encryption An SQLite db contains unencrypted human-readable text. Check this using your favorite text editor. This makes perfect sense for a light-weight solution.

Enforcement SQLite does not enforce foreign key constraints by default; you have to do this manually (although this may depend on the default settings of the distribution you downloaded).

Check that you can in fact create a table of users and one of roles, with foreign keys properly set, and then create a user without a role.

Then, turn on foreign key enforcement, and check that this is no longer possible.

```
PRAGMA foreign_keys = ON
```

Authentication SQLite does not authenticate users by default. Other databases include user management tools. If commands are executed without checks, anyone execurint sqlite can do anything as if they were the administrator. In and of itself, this is an issue of internal security - not everyone in a company should be allowed to read and write to the user db.

Injection However, a much bigger issue is that even people who have no direct access to your server may be able to execute arbitrary commands in your database.

There is a syntax to sanitize commands in the python sqlite3 library, but you have to use it rather than formatting strings in standard fashion.

Finally, no mention of SQL injection would be complete without reference to xkcd.

2.3 Linux permissions

1. Open a terminal, t_1. Create a new users, u_2. Note that a new group has been created with their name.
2. In t_1, log in as u_1. Open a new terminal, t_2, and log in as u_2. Create a new `txt` file in u_2's home directory. Check the permissions on that file; they should be `-rw-rw-r--`.
3. In t_1, try to access u_2's home directory; you should be able to do this since others have read access.
4. Try writing to this file; this should not be possible. Depending on the text editor you are using, you may be able to immediately see that the file has been opened in read-only mode.
5. In t_2, add u_1 to u_2's group.
6. In t_1, try once again to write to the new file. This should now be possible.

A sqlite3 example

```
import os
import sqlite3


do_create_tables = True
do_create_content = True

connection = sqlite3.connect('UNI.DB')
cursor = connection.cursor()


if do_create_tables:

    create_roles = """
       CREATE TABLE IF NOT EXISTS roles (
       name text NOT NULL UNIQUE,
       permission text NOT NULL)"""

    cursor.execute(create_roles)

    create_users = """
       CREATE TABLE IF NOT EXISTS users (
       name text NOT NULL UNIQUE,
       role text NOT NULL,
       FOREIGN KEY (role) REFERENCES roles (name))"""

    cursor.execute(create_users)
```

```
    # sanity check

    for row in cursor.execute("PRAGMA TABLE_INFO('USERS')").fetchall():
        print row

    for row in cursor.execute("PRAGMA TABLE_INFO('ROLES')").fetchall():
        print row

    # connection.commit()


if do_create_content:

    RA = [ ['"PCMember"', '"GrantTenure"'],
           ['"Faculty"', '"AssignGrades"'],
           ['"TA"', '"AssignHWScores"'],
           ['"UEmployee"','"ReceiveBenefits"'],
           ['"Student"', '"UseGym"'],
           ['"UMember"','"UseGym"'] ]

    UA = [ ['"Alice"', '"PCMember"'],
           ['"Bob"', '"Faculty"'],
           ['"Charlie"', '"Faculty"']]

    for entry in RA:

        cursor.execute("""insert into roles (name, permission) values ({0}, {1})
                """.format(entry[0], entry[1]))

    select_roles = "SELECT * FROM ROLES"
    cursor.execute(select_roles)
    print 'All roles: {0}'.format( cursor.fetchall() )

    for entry in UA:

        cursor.execute("""insert into users (name, role) values ({0}, {1})
                    """.format(entry[0], entry[1]))



    select_users = "SELECT * FROM USERS"
    cursor.execute(select_users)
    print 'All users: {0}'.format(cursor.fetchall())

# connection.close()
```

References

[Fer+01] D. F. Ferraiolo et al. “Proposed NIST standard for role-based access control”. In: *ACM Transactions on Information and System Security (TISSEC)* 4.3 (Aug. 2001), pp. 224–274. DOI: 10.1145/501978.501980.