Security and Privacy

Database Security

12.03.2019

slide credits: N.Gailly, C. Basescu, E. Alp, Ph. Oechslin

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Outline

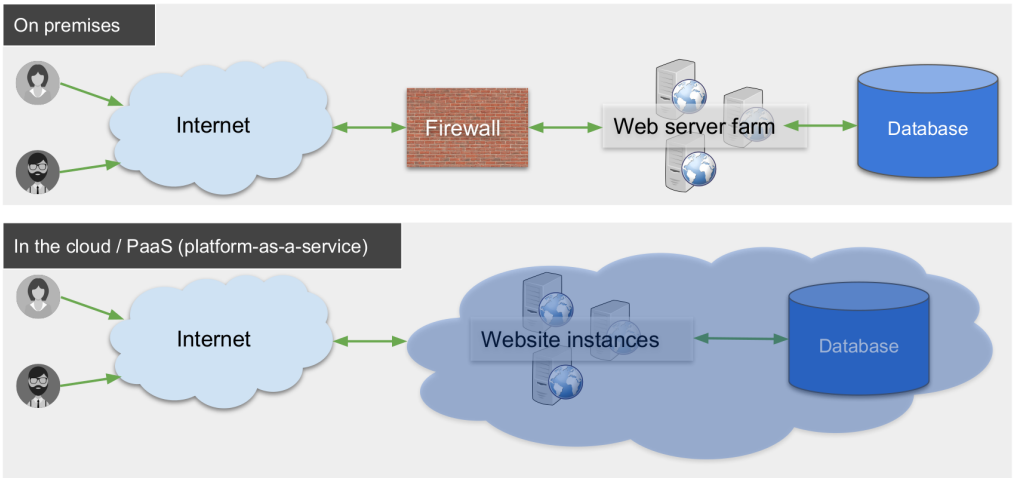
- Introduction
- Database access control
  - server, OS, DB, network, application
  - discretionary, role-based
- Securing the database
- Encrypting data
  - at rest, in transit, in the application
- Conclusions
  - Summary and questions

Introduction

Introduction

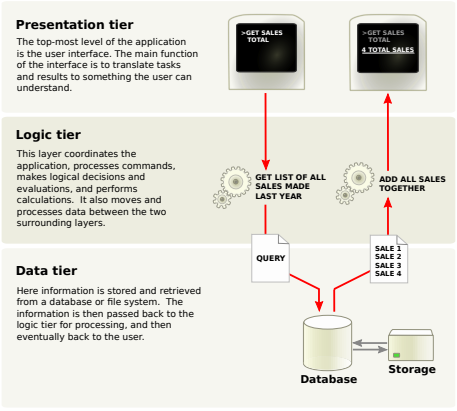
- Databases are used everywhere
  - Banking, industry, social networks, government, research, mobile apps, ...
  - Leads to new types of analysis (big data, machine learning, ...)
- Critical to think about the security of the data they contain
  - What are the security requirements of DB systems?
  - What are the main attack vectors of DB systems?
  - What are the main protections of DB systems?

Typical Setups



Multi-tier Architecture

- Presentation tier
  - user interface, web pages, static content
- Application (logic) tier
  - application logic
- Data tier
  - provides data persistence and APIs to access data



source: wikipedia

Database Threats

- Excessive privileges (users get more access than needed)
  - lack of segregation
- Weak passwords (anybody gets access)
- SQL injections (e.g. via web apps)
- Poor auditing records (you can't find out what happened)
- Storage media exposure (insider attacks, backups, ...)
- Denial of service

Examples of db attacks

- Taken from **SQLi Hall-of-Shame**:

Epic Games	2019-01	All Fortnite player accounts accessible, exposing credit card info etc.	**Bugs on Epic Games Site Allowed Hackers to Login to Any Fortnite Players Account**
Atlanta International Airport	2018-12	Data leak including over 700 passports	**Atlanta International Airport Hacked, 617.57 KB of Data Including +700 Passports Leaked Online**
Cisco	2018-12	vulnerability in Prime License Manager - now patched	**Cisco patches Prime License Manager SQL injection vulnerability**
Steam	2018-11	API vulnerability gave access to CD keys for any game. Now patched.	**Steam bug could have given you access to all the CD keys of any game**

Access Control

Layers of a database

- The layers of a database accesses:

Layer	Function	Threat
Hardware	The disk stores the data of the DB	A thief can take the disk or their backups
OS	The user running the DB accesses the files	The sysadmin can access the files
Database	DB administrators use privileged DB accounts for maintenance	They can access all data
Network	The application tier uses a TCP connection to talk to the DB	Hackers could connect to the DB remotely
Application	Uses a DB account to access the data of the application users	Application user can access data of other application users

- Each layer needs to implement proper access control

Access control: Least privilege

- Access control defines which actor can access which resource
- We should apply the principle of *least privilege*:
  - Each actor can only access the resources strictly needed for its function
- This must be applied at all levels of the DB

Access control: Hardware

- If it is a physical machine in a data center
  - physical protection : locks, cameras, alarms
- If it is a virtual machine in the cloud
  - cloning the machine is like stealing the hard disk
  - limit the number of people who have the right to clone
  - us strong authentication (e.g. 2 factor)

Access control: OS

- The DB runs as a process in the OS, owned by a certain OS user (e.g. mysql, oracle)

```
/com402$ ps -ef | grep mysql
mysql    29841     1  0 fév13 ?       00:04:02 /usr/sbin/mysqld
```

- This user is the only one allowed to access the files of the database:

```
/com402$ ls -l /var/lib/mysql/
total 110660
drwx------ 2 mysql mysql     4096 fév 13 23:13 com402
drwx------ 2 mysql root      4096 fév 13 22:49 mysql
drwx------ 2 mysql mysql     4096 fév 13 22:49 performance_schema
```

- The data of the com402 DB is stored in directory com402
- only user `mysql` is allowed to access this directory
  - or any user with root privileges...

Access control in the DB: DAC

- SQL databases use discretionary access control (DAC) to grant user access to objects through privileges,
- Typical objects are tables and views
- Typical privileges are INSERT, UPDATE, DELETE, CREATE, DROP
- By default, the `root` or `system` user has all privileges on all objects
- Allow Alice to read, modify or write the name, address and grades in the table called students of the database called com402:

```
grant SELECT,UPDATE,INSERT (name, address, grade) ON com402.students
  to alice@localhost;
```

Access control in the DB: Rows

- Granularity at the row level can be achieved by defining views:

```
CREATE VIEW Year_2019 AS SELECT * FROM com402.students
  WHERE academic_year=2019;
```

- We allow Bob to only read the lines with academic year 2019 in table students:

```
GRANT SELECT ON Year_2019 to bob@localhost;
```

- We can combine this to control access to each column and row of the database tables.

Access control in the DB: RBAC

- SQL databases also support role based access control
- Roles are created very much like users
  - privileges can be granted to roles
  - then, roles can be granted to users
  - profs can read most columns but only change grades:

```
CREATE ROLE prof;
GRANT SELECT (name, grade, academic_year) FROM students TO prof;
GRANT UPDATE (grade) ON com402.students TO prof;
```

  - Caetano is a prof:

```
GRANT prof to caetano@localhost;
```

- You can grant several roles to a same user (employee, lecturer, I&C)

Network access control

- Sometimes the application tier runs on the same machine as the DB
  - in that case the DB should be configured to only listen to connections coming from the local host

- In other cases, the application tier is on another machine.
  - We should accept connections only from the machines that are supposed to talk to the DB. Eg:
    - by installing a firewall in front of the DB, or
    - by using a local firewall on the DB server, or
    - by restricting users to IP addresses in the DB:

```
CREATE USER bob@10.2.2.33 IDENTIFIED BY 'horsebatterystapleOK'
```

Bob can only connect from 10.2.2.33

Application level Access control

- Applications usually have their own layer of users and privileges.
  - They use one or few DB accounts to interact with the DB
- Eg. an e-banking application may have 1000 customers and use a single DB user to manipulate the account balances.
  1. A table with all customers and their passwords
  2. A table with all accounts and their owners
  3. A table with all transactions of all accounts
- Access control is handled by the application
  - It uses the customer table for authentication
  - It uses the other two tables to find the accounts and give access to transactions
- Each customer only sees his own data, even if the DB user has access to all data.

QUIZ !

- **WCGW ?**
  (what could go wrong?)

SQL Injections (reminder)

- SQL injections in Python: DO NOT DO THIS:

```
param = "peter" # given by user
stmt = "SELECT name,grade  FROM students where name = '" + param +"'"
cursor.execute(stmt)
```

a hacker could can modify the request:

```
param ="'peter' union select name,password from students -- "
```

- DO THIS: Prepared statements in Python:

```
param = "peter" # given by user
stmt = "SELECT name,grade  FROM students where name = %s"
cursor.execute(stmt, (param,))
```

- Python replaces %s with the given parameter
  - The meaning of the statement can not change

Back to application level access control

- To limit the impact of SQL injections, use different DB users for different accesses:
  - One user with read access on the user table
    - for login in
  - One user with write access on the user table
    - for changing the password
  - One user with read access on the account owner table
    - for admins, customers can not change ownership of accounts
  - One user with read/write access to transactions
    - for the actual application

- The database access control is not exactly least privilege, but the impact of an injection is greatly reduced.
  - e.g. an SQL injection in the login form will not be able to read data from the transactions table

Application level access control

- Using different DB users is an application of the defense in depth principle
- The application implements fine grained access control
- The database implements coarse grained control
- If the access control at the application level fails, the access control of the database reduces the impact

Securing the database

Securing the database

- Some configurations can be dangerous
- Here are a few examples:
  - error messages
  - file acesss
  - default users

Securing the database (errors)

- Never show the error messages to the users
  - DBs try to not include data in error message
  - If you make an error complex enough, you can still succeed:

```
select
  count(*),
  concat((select password from students where name = 'peter' ),
         ' ',floor(rand(0)*2)) x
  from students group by x;

ERROR 1062 (23000): Duplicate entry 'password1234 1' for key
'group_key'
```

  tl;dr complex errors can reveal data

- Error messages should be sent to a log file rather than being displayed

Securing the database (files)

- SQL commands can read and write files.
- This could be used to extract data or to install a backdoor:
  - reading:

```
select LOAD_FILE('/etc/passwd');

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
...
```

  - writing:

```
SELECT "<? system($_REQUEST[‘cmd’]) ?>" INTO OUTFILE
'/var/www/website/backdoor.php';
```

- Access to files is a privilege that can be revoked

Securing the database (default users)

- Some databases come with default users and passwords
  - MySql: root / empty password (v 5.6 and older)
  - Oracle: SYSTEM / manager (for older versions)
  - MongoDB: by default, there is no authentication

Encrypting data

Encrypting data

- We have the same layers as with access control

Layer	Function	Protect against
Hardware / OS	Data is encrypted when read/write to disk	Stealing of disk/cloning virtual machines
Database	DB encrypts when read/write to file	Access by OS users/admins
Network	DB encrypts when read/write to network (e.g. TLS)	Hackers cannot sniff data in transit
Application	Application encrypts when read/write to the DB	Access by DB admins, memory dumps by OS admins

Encrypting data at rest

- Data stored on the server is data at rest
- Encrypting the hard disk at hardware or OS level only protects against theft/copy of the disks
  - A user that can log into the machine and access the files of the database will see clear text data
- Databases can be configured to encrypt data before writing it to files
  - file access does not yield clear text anymore
  - the keys may be stored in local files or obtained from a key server
    - e.g from the Amazon key management service for machines in the Amazon cloud
- Most DBs call this type of encryption Transparent Data Encryption (TDE)

Encrypting data in motion

- Data is exchanged between the DB and the application (web server or logic tier)
- If not encrypted, it could be eavesdropped
- Most DBs support TLS encryption for DB connections
  - Certificate pinning: the client can be given a copy of the server certificate or its CA
  - It will refuse to connect if the server presents a different certificate.
    - There is no use for 150 CAs as in web browsers

Encrypting data in the application

- With encryption at rest and in motion, data is still in clear in the memory of the DB!
  - an admin of the DB server can dump the memory and see the data
- The solution is to encrypt data in the application before storing it into the DB
- The key stays is in the application tier,
  - there is no way to decrypt the data on the DB server

QUIZ !

- WCGW if the database can not decrypt the data ?

Encrypted database

- If the data is encrypted in the database, then the DB can not
  - search for given values (e.g. where name='Pete%')
  - sort, compare or aggregate data
- This makes the DB pretty useless
- Useful for certain information
  - e.g. credit card numbers
  - pin codes
  - passwords ?

Encrypt with hardware

- Encryption/decryption can be delegated to a Hardware Security Module (HSM)
  - Neither your DB nor your application server know the key

source: **Securosys**

- HSMs safely store encryption keys (symmetric and asymmetric)
  - They can encrypt, decrypt or sign data
  - Keys can be generated inside the HSM and never leave
- You can even have 'cloud' HSMs in the Amazon, Google or Azure cloud

Conclusions

Conclusions

- Data stored in DBs is often the main asset of a company
- There are both privacy and security risks
- Apply access control at all layers
  - physical, OS, DB, network, application
- DBs can do both discretionary and role based access control
- Don't forget to correctly configure the database for security
  - e.g. errors, file access, default accounts
- Encryption is an efficient way to protect data
  - Transparent data encryption protects at OS and physical levels
  - Network traffic encryption is a must
  - Some data can be encrypted by the application (or an HSM)
    - you lose functionalities like search, sort, compare, ...

Questions

Questions

- How can database views be used to implement access control ?
- Why is it important to use different database users for different parts of an application ?
- Why is transparent data encryption (the DB encrypts before writing to files) better than en encrypted file system (the OS encrypts the content of the files) ?
- What is wrong with this:

```
stmt = "SELECT name,grade  FROM students where name = '" + param +"'"
cursor.execute(stmt)
```