Database Security

So far we have discussed

- The elemental theory behind databases
- How to design them
- The theory and practice behind querying them

Now we are pivoting to

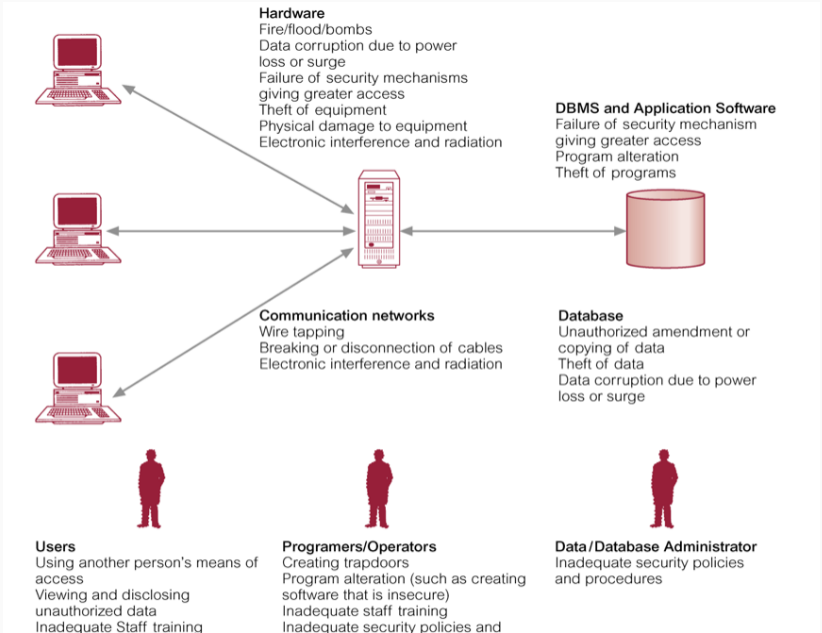
- Administratively, how do we keep them safe?

Database Security

Lots of reasons to care about security

- Data protection
- Laws
- Reputation
- Continuity
- Insider threats
- Competitive advantage
- Savings
- Network security
- Backup and Recover

Database Security



Hardware
Fire/flood/bombs
Data corruption due to power loss or surge
Failure of security mechanisms giving greater access
Theft of equipment
Physical damage to equipment
Electronic interference and radiation

DBMS and Application Software
Failure of security mechanism giving greater access
Program alteration
Theft of programs

Communication networks
Wire tapping
Breaking or disconnection of cables
Electronic interference and radiation

Database
Unauthorized amendment or copying of data
Theft of data
Data corruption due to power loss or surge

Users
Using another person's means of access
Viewing and disclosing unauthorized data
Inadequate Staff training

Programers/Operators
Creating trapdoors
Program alteration (such as creating software that is insecure)
Inadequate staff training
Inadequate security policies and

Data/Database Administrator
Inadequate security policies and procedures

Database Security

- Security is first enforce in application
- Secure systems can be compromised by badly written application code

Database Security

Our objectives today are

- Reviewing the role of the DBA in security
- Understanding common threats and how to counteract them

Threats

- SQL injection
- Data Breaches
- Phishing Attacks
- Malware
- Insider Threats
- Brute Force
- Zero-Day Exploit
- Cross-site Scripting (XSS)
- Data Tampering
- Social Engineering
- Inadequate Access Controls
- File Inclusion Vulnerabilities

SQL Injection

- Attackers get the application to execute a SQL query created by the attacker

Background: Accessing SQL From a Programming Language

- Not all queries can be expressed in SQL because its not a general-purpose language. We embed SQL in more powerful languages
- Printing results or sending them requires another language - need something for a nondeclarative action
- Dynamic SQL
  - SQL query is constructed as a char string at runtime
- Embdedded SQL
  - SQL statement is identified at compile time using a preprocessor

Prepared Statements

- We can a statement in which some values are replaced by ?
- ? is replaced later

```
PreparedStatement pStmt = conn.prepareStatement(
            ‘‘insert into instructor values(?, ?,?, ?)’’);
pStmt.setString(1, ’88877’);
pStmt.setString(2, ’Perry’);
pStmt.setString(3, ’Finance’);
pStmt.setString(4, 125000);
pStmt.executeUpdate();
pStmt.setString(1, ‘‘88878’’);
pStmt.edecuteUpdate();
```

pStmt.executeUpdate(); executes the update of the tuple (88878, Perry, Finance, 125000)

Prepared Statements

Imagine we write

```
''select * from instructor where name = '"+name+"'"
```

And a user instead of a name argues

```
X' or 'Y' = 'Y
```

The statement becomes

```
''select * from instructor where name = '"+X' or 'Y' = 'Y+"'
```

Read as:

```
select * from instructor where name = 'X' or 'Y' = 'Y'
```

Prepared Statements

- The statement is always true by construction
- Now the where clause has the entire relation returned

SQL Injection: Another Example

Consider “name” as the string input by the user

```
String query = "select * from student where name like ’% + n
```

Attacker argues

```
’;<some SQL statement>;--
```

Servlet executes

```
select * from student where name like ’’;<some SQL statement
```

SQL Injection: Another Example

- The quote inserted closes the string, the semicolon terminates the query, and the next statement is interpreted as a new query
- Any SQL statement may occur, changing the database

SQL Injection: A Third Example

- Assume an employee is not satisfied with their salary
- Passwords for the system update as follows

```
UPDATE employee
set password = '$newpwd$
WHERE eid = '$eid$' and password = '$oldpwd$'
```

In the password box, an employee can use a SQL query

```
paswd456',salary=100000#
```

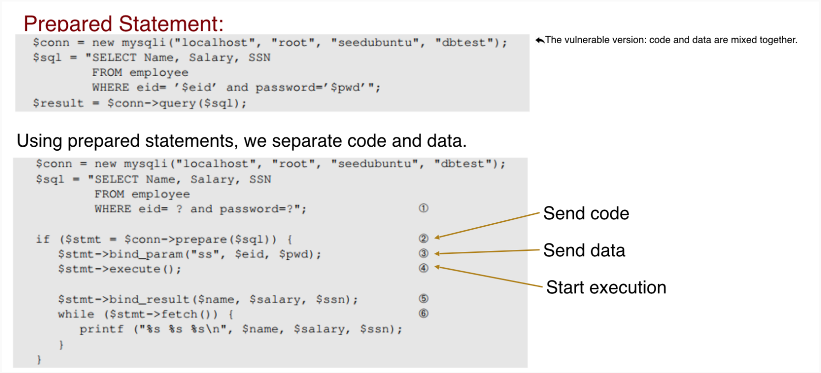
Prepared Statements

Solution 1: Prepared Statement

```
''select* from instructor where name = ‘X\’ or \‘Y\’ = \‘Y‘
```

Harmless statement

Prepared Statements



Prepared Statement:

```
$conn = new mysqli("localhost", "root", "seedubuntu", "dbtest");
$sql = "SELECT Name, Salary, SSN
        FROM employee
        WHERE eid= '$eid' and password='$pwd'";
$result = $conn->query($sql);
```

↖The vulnerable version: code and data are mixed together.

Using prepared statements, we separate code and data.

```
$conn = new mysqli("localhost", "root", "seedubuntu", "dbtest");
$sql = "SELECT Name, Salary, SSN
        FROM employee
        WHERE eid= ? and password=?";                 ①

if ($stmt = $conn->prepare($sql)) {                   ②
   $stmt->bind_param("ss", $eid, $pwd);               ③
   $stmt->execute();                                  ④

   $stmt->bind_result($name, $salary, $ssn);          ⑤
   while ($stmt->fetch()) {                           ⑥
      printf ("%s %s %s\n", $name, $salary, $ssn);
   }
}
```

Prepared Statements

- Trusted code is sent via a code channel
- Untrusted user-provided data is sent via data channel
- Database clearly knows the boundary between code and data
- Data received from the data channel is not parsed
- Attacker can hide code in data, but the code will never be treated as code, so it will never be attacked.

Prepared Statements

Solution 2: Encoding data

- Before mixing user-provided data with code, inspect the data. Filter out any character that may be interpreted as code.
- Special characters are commonly used in SQL Injection attacks. To get rid of them, encode them.
- Encoding a special character tells parser to treat the encoded character as data and not as code. This can be seen in the following example

```
aaa' OR 1=1#
aaa\' OR 1=1
```

SQL Injection

- The SQL injection example shows us a fundamental truth
- For each security threat, there is a countermeasure
- To make an adequate countermeasure, we need to understand the nature of the threat, vulnerability it exploits, the behavior that makes it possible

SQL Injection

- In SQL injection, the fundamental vulnerability is a marriage of code and data
- The solution is to divorce the two, so code cannot be disguised as data

Another Example: XSS

- Cross-site scripting
- Instead of entering a valid name for a query, code is embedded
- Browser enters the script, and sends cookie information back to the malicious user

XSS

- Disallow HTML tags in text input by users
- Identify a session with a cookie and an IP address, so someone using someone elses session information cannot login from a different machine

Passsword Leakage

- Even strong passwords stored in clear text could be exploited
- A user may gain access to scripts, allowing them to access passwords
- One solution is restricting access to a given set of Internet addresses, other attempts to connect are rejected

DBA's Security Responsibilities

- These are nice tips for discrete threats
- But how do they add up into a more holistic role for the DBA?

DBA’s role in data security

- Access Control: Manage user access permission and privileges
- Encryption: Implement encryption mechanisms to protect data at rest and during transmission
- Auditing: Track database activity to identify breaches
- Compliance Management: Adhere to special rules and laws pertaining to data
- Incident Response: Identifying breaches, implementing fixes, and reporting incidents

Auditing

Track database activity to identify breaches

Audit Trails

- A log of all changes to the application data along with who performed the change and when
- Can help
    - Identify who did what and how
    - Give you insight into when
- Ex: If a grade is wrong

Encryption

Implement encryption mechanisms to protect data at rest and during transmission

Encryption

- Transforming data into a form that is unreadable unless decryption is applied
- Makes it so sensitive information is only readable to intended receiver

Encryption

- Weak encryption can be cracked easily
- For instance, if we change “Perryridge” to “Qfsszsjehf” we have only shifted each letter by one position in the alphabet
- With a large enough number of codes, one can use statistics to crack something as weak as this

Encryption

Good encryption has the following properties

- Authorized users can encrypt and decrypt easily
- It relies on the secrecy of a parameter (encryption key)
- It is extremely difficult to learn the key even when the data is known

Advanced Encryption Standard (AES)

- **Overview:** AES is a symmetric key encryption algorithm widely used worldwide.
- **Development:** Developed to replace the older DES (Data Encryption Standard) and adopted by the U.S. government in 2001.
- **Key Sizes:** Supports key sizes of 128, 192, and 256 bits.
- **Operation:** Operates on 128-bit blocks of data, using several rounds of data scrambling.
- **Security:** Considered highly secure, AES is resistant to various attack strategies like linear and differential cryptanalysis.
- **Applications:** Used in numerous applications like securing wireless networks (WPA2), encrypting file systems, and securing web communications (SSL/TLS).
- **Performance:** Efficient in both software and hardware, relatively fast, and requires less memory, making it suitable for a wide range of devices.
- **Compliance:** Meets the requirements of many security protocols and standards, ensuring a high level of data protection.

[illegible]

Public Key Encryption

- **Overview:** Public key encryption, also known as asymmetric encryption, involves two distinct keys – a public key and a private key.
- **Key Mechanism:** The public key, which can be shared with everyone, is used for encrypting data, while the private key, kept secret, is used for decrypting data.
- **Security:** It relies on cryptographic algorithms based on mathematical problems to yield strong security, such as RSA, ECC (Elliptic Curve Cryptography), and Diffie-Hellman.
- **Applications:** Essential for secure communications over the Internet, used in protocols like HTTPS, for secure email (PGP), and digital signatures.
- **Key Exchange:** Facilitates secure key exchange over an insecure channel without the need for a shared secret key in advance.
- **Digital Signatures:** Enables authentication of digital documents, ensuring the integrity and origin of the data.

Public Key Encryption

- **Challenges:** Managing and distributing public keys securely is a complex task, often addressed through Public Key Infrastructure (PKI).
- **Performance:** Generally slower than symmetric key encryption due to the computational complexity of the algorithms.

Access Control

Manage user access permission and privileges

Access Control and Permissions

- Access control is the overarching framework for controlling resource access. Permissions, on the other hand, are the specific rules and settings that determine what actions can be performed on those resources by authorized users.
- Key aspects include authentication and authorization

Authorization

- Authorization determines what authenticated users are allowed to do

Authorization

Authorizations

- Read
- Insert
- Update
- Delete

These are privileges

Privileges in SQL

Types of privileges:

- `select`: Allows read access to relation or the ability to query using the view.
- `insert`: The ability to insert tuples.
- `update`: The ability to update using the SQL update statement.
- `delete`: The ability to delete tuples.
- `all privileges`: Used as a short form for all the allowable privileges.

Granting Privileges

- The grant statement is used to give authorization

```
grant <privledge list>
on <relation name OR view name>
to <user/role list>'
```

Give Amit and Santoshi select authorization on the department relation

```
grant select on department to Amit, Santoshi;
```

Grant update authorization on the budget attribute

```
grant update (budget) on department to Amit, Santoshi;
```

user name “public” is all current and future users of the system

Revoking Authorization in SQL

The `revoke` statement is used to revoke authorization.

- `revoke <privilege list> on <relation or view> from <user list>`
- Example: `revoke select on student from U1, U2, U3`
- If the same privilege was granted twice to the same user by different grantees, the user may retain the privilege after the revocation.

Roles

A role distinguishes among various users regarding what they can access/update in the database.

- To create a role: `create role <name>`
- Example: `create role instructor`
- Once a role is created, users can be assigned to the role using: `grant <role> to <users>`

Roles Example

Creating and granting roles:

- Create a role: `create role instructor`
- Grant a role to a user: `grant instructor to Amit`
- Privileges can be granted to roles: `grant select on takes to instructor`
- Roles can be granted to users, as well as to other roles.
- Chain of roles: Create a role `dean` and grant it to `Satoshi`.

Application-Level Authorization

Say we wanted students to be able to view their grades and no others. This is hard

- We lack end-user information (syscontext.user_id())
- Lack of fine-grained authorization
    - We need tuple-level view authorization
    - We could use

```
create view studentTakes
as select *
from takes
where takes.ID=syscontext.user_id()
```

- Oracle Virtual Private Database (VPD)

Application Authentication

Authentication confirms the identity of users

- Username and Password: A user provides a username and a corresponding password. The system compares the provided password with the stored, hashed password to determine authenticity.
- Multi-Factor Authentication (MFA): MFA requires users to provide multiple forms of verification, such as something they know (password), something they have (a token or mobile device), or something they are (biometric data like a fingerprint).
- Public Key Infrastructure (PKI): PKI uses digital certificates and cryptographic keys to establish identity and secure communication between parties.
- Biometrics: Biometric authentication uses unique physical or behavioral characteristics like fingerprints, facial recognition, or iris scans to verify identity.

Application Authentication

- Man in the Middle Attacks

Backup and Recovery

Backup and Recovery

- Security threats don’t only come from malicious users
- They also come from nature

Importance of Backup and Recovery

- **Data Loss Prevention:** Backups safeguard against data loss due to hardware failures, software errors, human mistakes, or cyberattacks.
- **Business Continuity:** In the event of disasters or breaches, backups ensure minimal disruption to operations.
- **Data Integrity:** Restoration from backups maintains accurate and reliable data in case of corruption.
- **Protection Against Ransomware:** Regular backups enable recovery without paying ransoms, thwarting attackers' intentions.
- **Compliance Requirements:** Adherence to industry regulations mandating data retention and backup practices.
- **Disaster Recovery Planning:** Backups are crucial for systematic recovery during crises.
- **Cost Savings:** Prevention of data loss and associated recovery expenses.
- **Data Migration and Testing:** Backups aid in data migration and software testing without risking production data.

Different Types of Database Backups

- **Full Backup:** Captures a complete copy of the entire database, including all data and schema objects.
- **Differential Backup:** Captures only the changes made since the last full backup, reducing backup size and time.
- **Incremental Backup:** Captures only the changes made since the last backup (full or incremental).
- **Transaction Log Backup:** Captures changes in the database's transaction log, crucial for point-in-time recovery.
- **Copy-Only Backup:** Does not affect the regular backup sequence, useful for ad-hoc backups.
- **Filegroup Backup:** Captures specific filegroups or files within a database, allowing for granular backup and recovery.
- **Partial Backup:** Captures only the primary filegroup and optionally specified secondary filegroups.

Different Types of Database Backups

- **Snapshot Backup:** Creates a point-in-time, read-only copy of a database, filegroup, or volume.
- **Remote Backup:** Involves copying data to a remote location or cloud storage for offsite protection.
- **Offline Backup:** Taken while the database is offline, ensuring consistency but may involve downtime.

Disaster Recovery Planning

- **Risk Assessment and Business Impact Analysis (BIA):** Evaluating potential risks and their impact on business operations.
- **Recovery Objectives:** Setting clear goals for recovery time and recovery point objectives.
- **Cross-functional Disaster Recovery Team:** Establishing a team responsible for implementing the disaster recovery plan.
- **Data Backup and Storage:** Ensuring regular backups and secure storage of critical data.
- **Recovery Procedures:** Developing detailed procedures for restoring systems and data after a disaster.
- **Testing and Validation:** Regularly testing the recovery plan to ensure its effectiveness.
- **Redundancy and Failover:** Implementing redundant systems and failover mechanisms for critical components.
- **Communication Plan:** Establishing clear communication channels for use during a disaster.
- **Documentation and Version Control:**

Disaster Recovery Planning

- **Vendor and Supplier Assessments:** Evaluating the disaster recovery capabilities of vendors and suppliers.
- **Employee Training:** Training staff in disaster recovery procedures and their roles during a disaster.
- **Compliance and Regulations:** Ensuring the disaster recovery plan meets regulatory requirements.
- **Continuous Improvement:** Regularly updating the plan based on new risks, technologies, and business changes.
- **Recovery Environment:** Setting up an alternate environment for business continuity during recovery.
- **External Support:** Arranging for external support services as part of the disaster recovery strategy.

Case Study: Code Spaces (2014)

- **Incident:** Cyberattack through unauthorized access and deletion of data.
- **Impact:** Critical data and backups were destroyed, leading to significant data loss.
- **Recovery Effort:** The company was unable to recover its data due to the extent of damage.
- **Outcome:** Code Spaces ceased operations, demonstrating the devastating impact of not having a robust, off-site backup strategy.

Case Study: Delta Airlines (2016)

- **Incident:** A power outage caused a system failure, leading to significant operational disruptions.
- **Impact:** Resulted in widespread flight cancellations and data disruptions.
- **Recovery Effort:** Utilized a comprehensive disaster recovery plan with redundant systems.
- **Outcome:** Delta successfully restored operations within hours, minimizing the impact on its services and customers.

Case Study: GitLab (2017)

- **Incident:** Accidental deletion of a production database due to human error.
- **Impact:** Loss of user data and disruption to GitLab services.
- **Recovery Effort:** Implemented backup strategies with community support.
- **Outcome:** Successful data recovery, but the incident emphasized the critical importance of regular and validated backups.

Case Study: Puerto Rico's Healthcare System (2017)

- **Incident:** Hurricanes Maria and Irma caused extensive damage, disrupting operations and data access.
- **Impact:** Widespread disruption in healthcare services and data access.
- **Recovery Effort:** Leveraged effective disaster recovery planning and geographic redundancy.
- **Outcome:** Successful restoration of healthcare services, highlighting the importance of comprehensive disaster recovery strategies.

Case Study: Norsk Hydro (2019)

- **Incident:** Targeted by a cyberattack that led to substantial operational disruptions.
- **Impact:** Affected IT systems across multiple business areas, causing significant operational challenges.
- **Recovery Effort:** Managed to maintain operations and initiate recovery using well-planned backup and security strategies.
- **Outcome:** Demonstrated the effectiveness of proactive security measures and robust backup systems in mitigating the impact of cyberattacks.

Case Study: Salesforce (2021)

- **Incident:** A database script error led to granting users broader data access than intended.
- **Impact:** Thousands of customers potentially had access to all their company's data, regardless of their permission levels.
- **Recovery Effort:** Salesforce immediately blocked access to the affected databases and began an internal investigation to rectify the issue.
- **Outcome:** The company quickly restored correct permissions and ensured no customer data was compromised, showcasing effective incident response and communication with affected parties.
- **Lessons Learned:** Emphasized the importance of rigorous testing of database scripts and robust access control mechanisms.