Database Security

Tyler Bletsch
Duke University
Relational Databases

- Table of data consisting of rows and columns
  - Each column holds a particular type of data
  - Each row contains a specific value for each column
  - Ideally has one column where all values are unique, forming an identifier/key for that row

- Enables the creation of multiple tables linked together by a unique identifier that is present in all tables

- Use a relational query language to access the database
  - Allows the user to request data that fit a given set of criteria
Figure 5.2 Example Relational Database Model. A relational database uses multiple tables related to one another by a designated key; in this case the key is the PhoneNumber field.
Relational Database Elements

- **Primary key**
  - Uniquely identifies a row
  - Consists of one or more column names

- **Foreign key**
  - Links one table to attributes in another

- **View/virtual table**
  - Result of a query that returns selected rows and columns from one or more tables

- Relation/table/file
- Tuple/row/record
- Attribute/column/field
(a) Two tables in a relational database

(b) A view derived from the database

---

**Figure 5.4  Relational Database Example**
Structured Query Language (SQL)

- Standardized language to define schema, manipulate, and query data in a relational database
- Several similar versions of ANSI/ISO standard
- All follow the same basic syntax and semantics

SQL statements can be used to:

- Create tables
- Insert and delete data in tables
- Create views
- Retrieve data with query statements
SQL Injection Attacks (SQLi)

- One of the most prevalent and dangerous network-based security threats
- Designed to exploit the nature of Web application pages
- Sends malicious SQL commands to the database server

- Most common attack goal is bulk extraction of data
- Depending on the environment SQL injection can also be exploited to:
  - Modify or delete data
  - Execute arbitrary operating system commands
  - Launch denial-of-service (DoS) attacks
Figure 5.5  Typical SQL Injection Attack

Legend:
- Data exchanged between hacker and servers
- Two-way traffic between hacker and Web server
- Credit card data is retrieved from database
The SQLi attack typically works by prematurely terminating a text string and appending a new command. Because the inserted command may have additional strings appended to it before it is executed, the attacker terminates the injected string with a comment mark “--”.

Subsequent text is ignored at execution time.
Inband Attacks

- Uses the same communication channel for injecting SQL code and retrieving results
- The retrieved data are presented directly in application Web page
- Include:

  **Tautology**
  
  This form of attack injects code in one or more conditional statements so that they always evaluate to true

  **End-of-line comment**
  
  After injecting code into a particular field, legitimate code that follows are nullified through usage of end of line comments

  **Piggybacked queries**
  
  The attacker adds additional queries beyond the intended query, piggy-backing the attack on top of a legitimate request
Out-of-Band Attack

- Data are retrieved using a different channel
- This can be used when there are limitations on information retrieval, but outbound connectivity from the database server is lax
SQLi Countermeasures

- Three types:
  - Manual defensive coding practices
  - Parameterized query insertion
  - SQL DOM

**Defensive coding**
- Manual defensive coding practices
- Parameterized query insertion
- SQL DOM

**Detection**
- Signature based
- Anomaly based
- Code analysis

**Run-time prevention**
- Check queries at runtime to see if they conform to a model of expected queries
SQL injection examples

See here:

http://www.w3schools.com/sql/sql_injection.asp
Proper database coding practices

• Escaping special characters  ← Better than nothing…
  
  $query = \\
  \texttt{sprintf("SELECT * FROM users WHERE user='\%s'", \\
  mysql\_real\_escape\_string($user));}

• Parameterized queries  ← Decent, if you have to…
  
  $stmt = $pdo->\texttt{prepare}('SELECT * FROM employees WHERE \\
  name = :name'); \\
  $stmt->\texttt{execute}(array('name' => $name));

• FRAMEWORKS: NOT DOING SQL YOURSELF!  ← That’s where it’s at.
  
  new\_guy = User\_create(
    username = 'foo',
    email = 'foo@bar.com',
    age = 25,
    lang = ['en', 'fr']
  )
  new\_guy\_commit();

Object Relational Mapper (ORM)
  • The most common form of database framework.
  • Programmer writes class definitions, framework creates whole database automatically
  • Classes integrate with database with no extra code.
  • Less work, no SQL injection
Inferential Attack

- There is no actual transfer of data, but the attacker is able to reconstruct the information by sending particular requests and observing the resulting behavior of the Website/database server.

- Include:
  - Illegal/logically incorrect queries
    - This attack lets an attacker gather important information about the type and structure of the backend database of a Web application.
    - The attack is considered a preliminary, information-gathering step for other attacks.
  - Blind SQL injection
    - Allows attackers to infer the data present in a database system even when the system is sufficiently secure to not display any erroneous information back to the attacker.
Figure 5.7 Indirect Information Access Via Inference Channel
### (a) Employee table

<table>
<thead>
<tr>
<th>Name</th>
<th>Position</th>
<th>Salary ($)</th>
<th>Department</th>
<th>Dept. Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andy</td>
<td>senior</td>
<td>43,000</td>
<td>strip</td>
<td>Cathy</td>
</tr>
<tr>
<td>Calvin</td>
<td>junior</td>
<td>35,000</td>
<td>strip</td>
<td>Cathy</td>
</tr>
<tr>
<td>Cathy</td>
<td>senior</td>
<td>48,000</td>
<td>strip</td>
<td>Cathy</td>
</tr>
<tr>
<td>Dennis</td>
<td>junior</td>
<td>38,000</td>
<td>panel</td>
<td>Herman</td>
</tr>
<tr>
<td>Herman</td>
<td>senior</td>
<td>55,000</td>
<td>panel</td>
<td>Herman</td>
</tr>
<tr>
<td>Ziggy</td>
<td>senior</td>
<td>67,000</td>
<td>panel</td>
<td>Herman</td>
</tr>
</tbody>
</table>

### (b) Two views

<table>
<thead>
<tr>
<th>Position</th>
<th>Salary ($)</th>
<th>Name</th>
<th>Department</th>
</tr>
</thead>
<tbody>
<tr>
<td>senior</td>
<td>43,000</td>
<td>Andy</td>
<td>strip</td>
</tr>
<tr>
<td>junior</td>
<td>35,000</td>
<td>Calvin</td>
<td>strip</td>
</tr>
<tr>
<td>senior</td>
<td>48,000</td>
<td>Cathy</td>
<td>strip</td>
</tr>
</tbody>
</table>

### (c) Table derived from combining query answers

<table>
<thead>
<tr>
<th>Name</th>
<th>Position</th>
<th>Salary ($)</th>
<th>Department</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andy</td>
<td>senior</td>
<td>43,000</td>
<td>strip</td>
</tr>
<tr>
<td>Calvin</td>
<td>junior</td>
<td>35,000</td>
<td>strip</td>
</tr>
<tr>
<td>Cathy</td>
<td>senior</td>
<td>48,000</td>
<td>strip</td>
</tr>
</tbody>
</table>

**Figure 5.8 Inference Example**
Access control system determines what access rights the user has (create, insert, delete, update, read, write)

Two commands for managing access rights:
- Grant
  - Used to grant one or more access rights or can be used to assign a user to a role
- Revoke
  - Revokes the access rights

Typical access rights are:
- Select
- Insert
- Update
- Delete
- References
The database is typically the most valuable information resource for any organization. Protected by multiple layers of security:

- Firewalls, authentication, general access control systems, DB access control systems, database encryption
- Encryption becomes the last line of defense in database security

Can be applied to the entire database, at the record level, the attribute level, or level of the individual field.

Disadvantages to encryption:

- Key management
  - Authorized users must have access to the decryption key for the data for which they have access
- Inflexibility
  - When part or all of the database is encrypted it becomes more difficult to perform record searching
Database security summary

- Don’t do dumb coding practices that allow SQL injection
  - Object Relational Mapper (ORM) = good

- Think carefully about different views of data and what they could reveal if combined

- Apply principle of least privilege to database permissions

- Keep your database credentials secret!
  - Don’t put them into git

- Database encryption may be applicable, if you can deal with key management (don’t put key next to data!)