



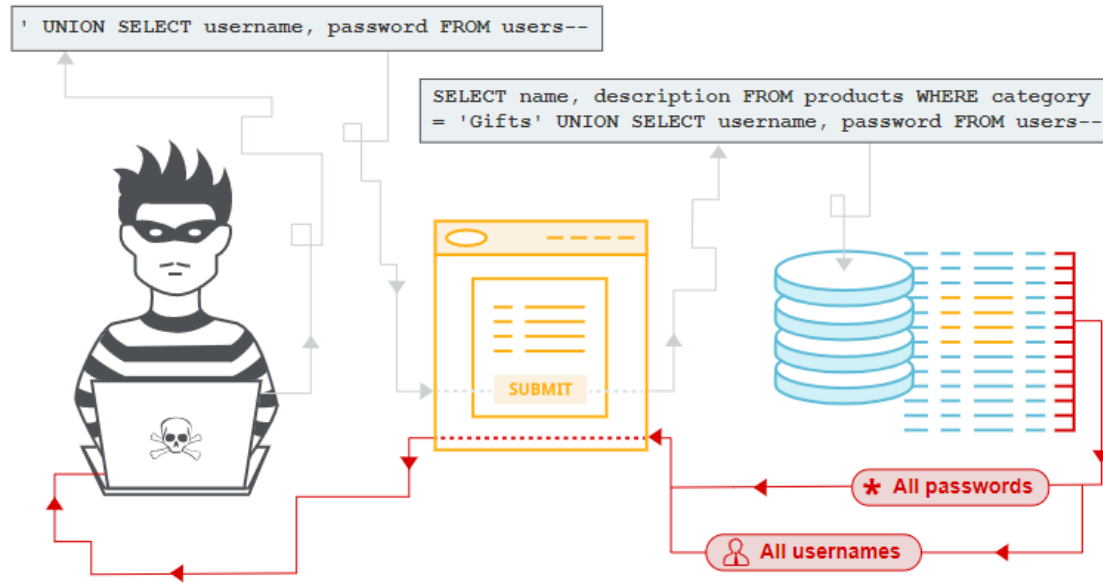
Module 2-7

Data Security

Objectives

- SQL Injection Attack
- Prepared statements
- Hashing
- Salt
- Encryption

SQL Injection attacks



SQL Injection attacks

- Makes it possible to execute malicious SQL statements
 - SQL statements control database server
 - Attackers can bypass authentication and authorization
 - Can add modify and delete records in a database

Types of SQL Injection

1. Query Modification

The attacker modifies the original query and then ignores the rest of the original by adding `--` at the end of their addition to comment it out.

2. Union Attack

The attacker creates a `UNION` with an existing query that returns results from their query mixed with results of a legitimate query.

3. Stacked Queries

The attacker ends the original query with a `;` and then appends their own query onto the original.

4. Second Order Attack

Parts of a SQL query are stored in related fields that execute together at a later time.

Preventing SQL Injection

1. **Parameterized Queries** - The single most effective thing you can do to prevent SQL injection is to use parameterized queries. *If this is done consistently, First Order SQL injection will not be possible*, however, second level attacks are still possible.
2. **Input Validation** - Limiting the data that can be input by a user can certainly be helpful in preventing SQL Injection, but is by no means an effective prevention by itself. *If done consistently then Second Order SQL Injection will be also prevented.*
3. **Limit Database User Privileges** - A web application should always use a database user to connect to the database that has as few permissions as necessary.

Preventing SQL Injection

- Parameterized Queries

```
String userId = {get data from end user};  
String sqlQuery = "select * from tbluser where userId = " + userId;
```

If this is executed with
a userId of 132, it will look
like this:
SELECT * FROM tbluser
WHERE userId=132;

A hacker can alter a user request to send SQL code
where the userId says 2 OR 1=1;

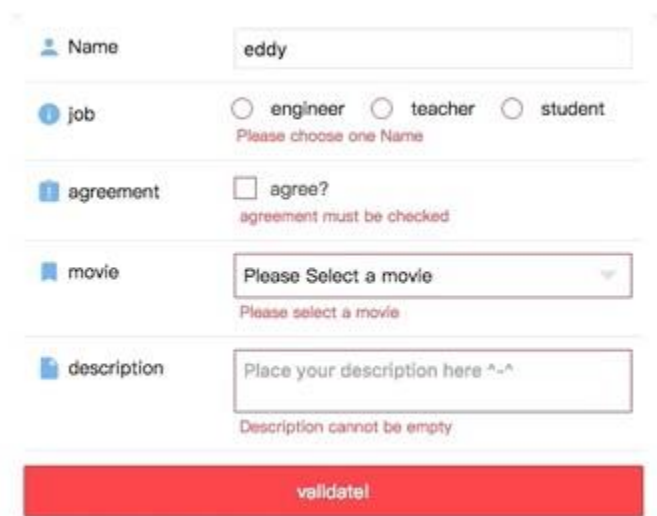
This will cause the sqlQuery to read:

SELECT * FROM tbluser WHERE userId=2 OR 1=1;

Because 1=1 is always true, it will return all data from the
table!

Preventing SQL Injection

- Input Validation



A screenshot of a web form with the following fields and validation messages:

- Name:** Input field containing "eddy".
- job:** Radio buttons for "engineer", "teacher", and "student". Below the buttons is the message "Please choose one Name".
- agreement:** A checkbox labeled "agree?". Below it is the message "agreement must be checked".
- movie:** A dropdown menu with the text "Please Select a movie". Below it is the message "Please select a movie".
- description:** A text area with the placeholder text "Place your description here ^-^". Below it is the message "Description cannot be empty".

At the bottom of the form is a red button labeled "validate!".



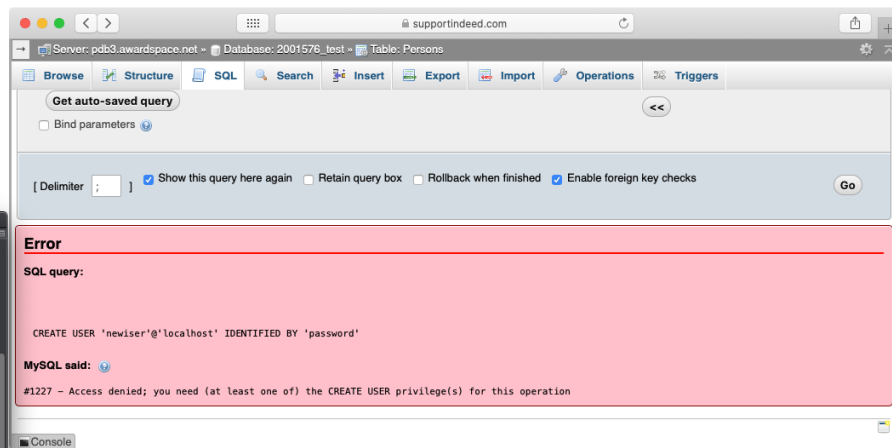
Preventing SQL Injection

- Limit Database User Privileges

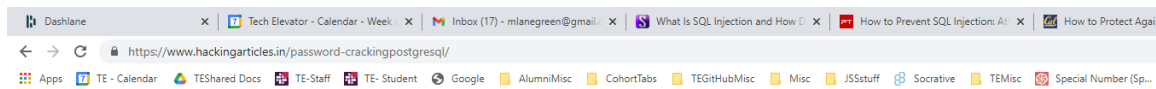
```
flaviocopes ~ — psql /Users/flaviocopes — psql postgres — 90x14
postgres=# CREATE ROLE employee WITH CREATEDB;
CREATE ROLE
postgres=# GRANT employee TO flavio;
GRANT ROLE
postgres=#
postgres=# \du
```

Role name	Attributes	Member of
employee	Create DB, Cannot login	{}
flavio	Cannot login	{employee}
flaviocopes	Superuser, Create role, Create DB, Replication, Bypass RLS	{}

```
postgres=#
```



Preventing SQL Injection



Medusa

Medusa is intended to be a speedy, massively parallel, modular, login brute-forcer. It supports many protocols: AFP, CVS, POSTGRES, HTTP, IMAP, rlogin, SSH, Subversion, and VNC to name a few

Run the following command

```
1 | medusa -h 192.168.1.120 -U /root/Desktop/user.txt -P /root/Desktop/pass.txt -M postgres
```

Here

-U: denotes path for username list

-P: denotes path for the password list

As you can observe that we had successfully grabbed the Postgres **username** as **Postgres** and **password** as **postgres**.

```
root@kali:~# medusa -h 192.168.1.120 -U /root/Desktop/user.txt -P /root/Desktop/pass.txt -M postgres
Medusa v2.2 (http://www.fooofus.net) (C) JoMo-Kun / Fooofus Networks <jmk@fooofus.net>

ACCOUNT CHECK: [postgres] Host: 192.168.1.120 (1 of 1, 0 complete) User: root (1 of 5, 0 complete) Pa
ACCOUNT CHECK: [postgres] Host: 192.168.1.120 (1 of 1, 0 complete) User: root (1 of 5, 0 complete) Pa
ACCOUNT CHECK: [postgres] Host: 192.168.1.120 (1 of 1, 0 complete) User: root (1 of 5, 0 complete) Pa
ACCOUNT CHECK: [postgres] Host: 192.168.1.120 (1 of 1, 0 complete) User: root (1 of 5, 0 complete) Pa
ACCOUNT CHECK: [postgres] Host: 192.168.1.120 (1 of 1, 0 complete) User: raj (2 of 5, 1 complete) Pas
ACCOUNT CHECK: [postgres] Host: 192.168.1.120 (1 of 1, 0 complete) User: raj (2 of 5, 1 complete) Pas
ACCOUNT CHECK: [postgres] Host: 192.168.1.120 (1 of 1, 0 complete) User: raj (2 of 5, 1 complete) Pas
ACCOUNT CHECK: [postgres] Host: 192.168.1.120 (1 of 1, 0 complete) User: raj (2 of 5, 1 complete) Pas
ACCOUNT CHECK: [postgres] Host: 192.168.1.120 (1 of 1, 0 complete) User: toor (3 of 5, 2 complete) Pa
ACCOUNT CHECK: [postgres] Host: 192.168.1.120 (1 of 1, 0 complete) User: toor (3 of 5, 2 complete) Pa
ACCOUNT CHECK: [postgres] Host: 192.168.1.120 (1 of 1, 0 complete) User: toor (3 of 5, 2 complete) Pa
ACCOUNT CHECK: [postgres] Host: 192.168.1.120 (1 of 1, 0 complete) User: toor (3 of 5, 2 complete) Pa
ACCOUNT CHECK: [postgres] Host: 192.168.1.120 (1 of 1, 0 complete) User: postgres (4 of 5, 3 complete)
ACCOUNT CHECK: [postgres] Host: 192.168.1.120 (1 of 1, 0 complete) User: postgres (4 of 5, 3 complete)
ACCOUNT CHECK: [postgres] Host: 192.168.1.120 (1 of 1, 0 complete) User: postgres (4 of 5, 3 complete)
ACCOUNT FOUND: [postgres] Host: 192.168.1.120 User: postgres Password: postgres [SUCCESS]
ACCOUNT CHECK: [postgres] Host: 192.168.1.120 (1 of 1, 0 complete) User: pavan (5 of 5, 4 complete) P
ACCOUNT CHECK: [postgres] Host: 192.168.1.120 (1 of 1, 0 complete) User: pavan (5 of 5, 4 complete) P
ACCOUNT CHECK: [postgres] Host: 192.168.1.120 (1 of 1, 0 complete) User: pavan (5 of 5, 4 complete) P
```

Ncrack

Protecting sensitive data

- How many stories have we heard regarding data breaches divulging sensitive information??
<https://www.csoonline.com/article/2130877/the-biggest-data-breaches-of-the-21st-century.html>
- Data stored in a database hacked
- To stop this, we need to have data stored in a database in such a way that it is not readable by unauthorized parties
- Data can be protected by either hashing or encryption

Hashing

- Using an algorithm to map data of any size to a fixed length.
 - Called a hash code or hash value
 - Many different algorithms (MD2, MD4, MD5, SHA, SHA1, SHA2)
- Is a one-way function
 - Technically it is possible to reverse-hash, would require immense computing power therefore unfeasible
- Meant to verify that a file or piece of data has not been altered

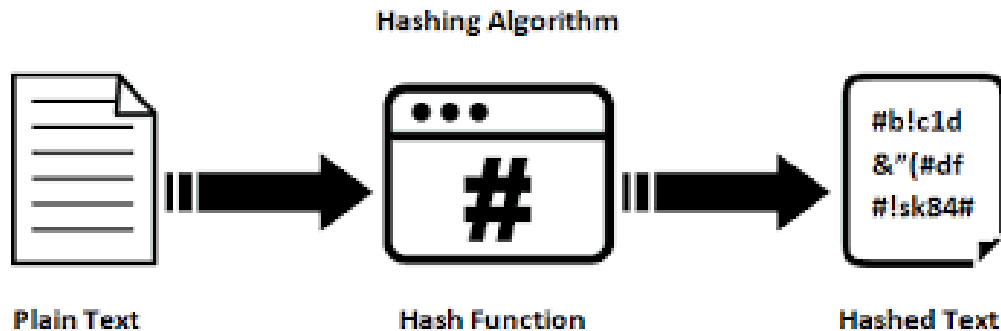
```
hash("password") = 2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e7
```

Hashing

A Hash Function is one that can map input data of arbitrary size to a fixed size output.

Hashing is 1-way, meaning that once data is hashed, the hash cannot be reversed back into the original data.

Commonly used to store passwords.



[MD5 Hash Generator](#)

Hashing

- Hashed output of the same string will be the same.
- Hashed data conforms to algorithm in terms of storage size
- The stronger hash function used, the more storage required, the slower the performance but minimal chance of having collision
- Humans are predictable, passwords tend to be memorable keywords, phrases, or numbers
- Hackers create a “rainbow” table of possible passwords and run this through while trying to hack in
 - Salt

SALT

- Unique value added to end of password to create a different value.
- Adds layer of security to hashing process
 - Helps protect against brute force
- Because salt is unique, produced hash of same password will not be the same.

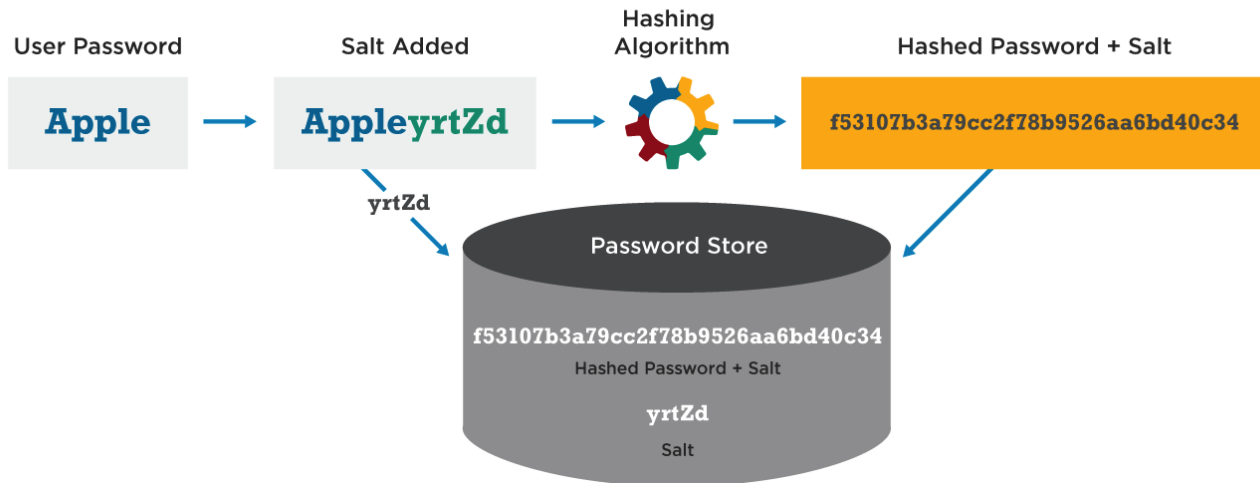
```
hash("hello") = 2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824  
hash("hello" + "QxLUF1bgIAdeQX") = 9e209040c863f84a31e719795b2577523954739fe5ed3b58a75cff2127075ed1  
hash("hello" + "bv5PehSMFV11Cd") = d1d3ec2e6f20fd420d50e2642992841d8338a314b8ea157c9e18477aaef226ab  
hash("hello" + "YYLmfY6IehjZMQ") = a49670c3c18b9e079b9cfaf51634f563dc8ae3070db2c4a8544305df1b60f007
```


Salting

A Salt is a fixed-length cryptographically-strong random value that is added to a password as input to a hash function.

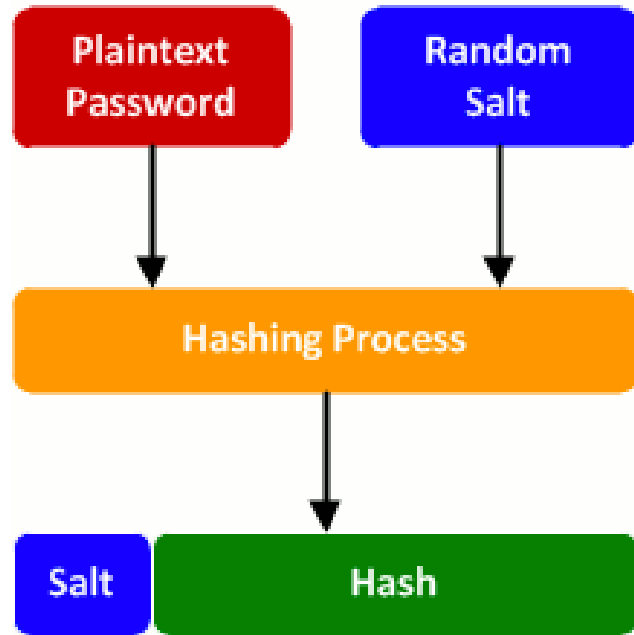
Dictionary attacks make passwords hashed with common algorithms vulnerable, salting reduces the effectiveness of dictionary attacks by making all input values for passwords unique.

Password Hash Salting

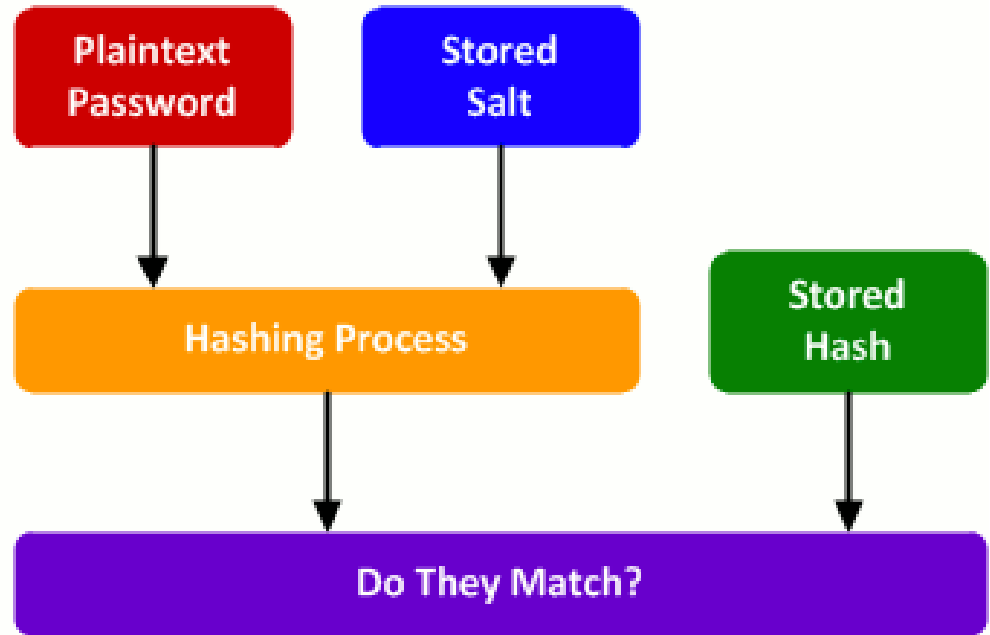


Password Salting

Password Creation



Password Verification



Encryption

- Most effective way to achieve data security
- Practice of scrambling information
 - Needs a key to unscramble
- Two-way function

Example Cipher

A = D

A B C D E...
x3

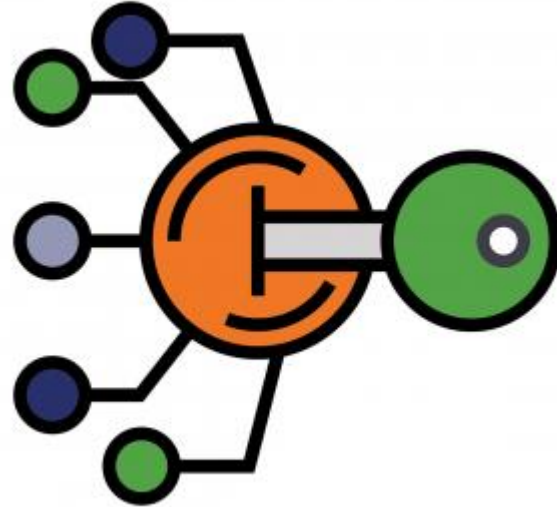
Plaintext: Don't be a jerk

Becomes:

Ciphertext: Grqwehdmhun

Encryption algorithms

- Shift ciphers
- Substitution ciphers
- Transposition ciphers
- Polyalphabetic ciphers
- Nomenclature ciphers



Modern encryption algorithms

- Asymmetric Encryption

- Public key example – 1 key encrypts, 1 key decrypts
- Used in SSL/TLS transfer of data

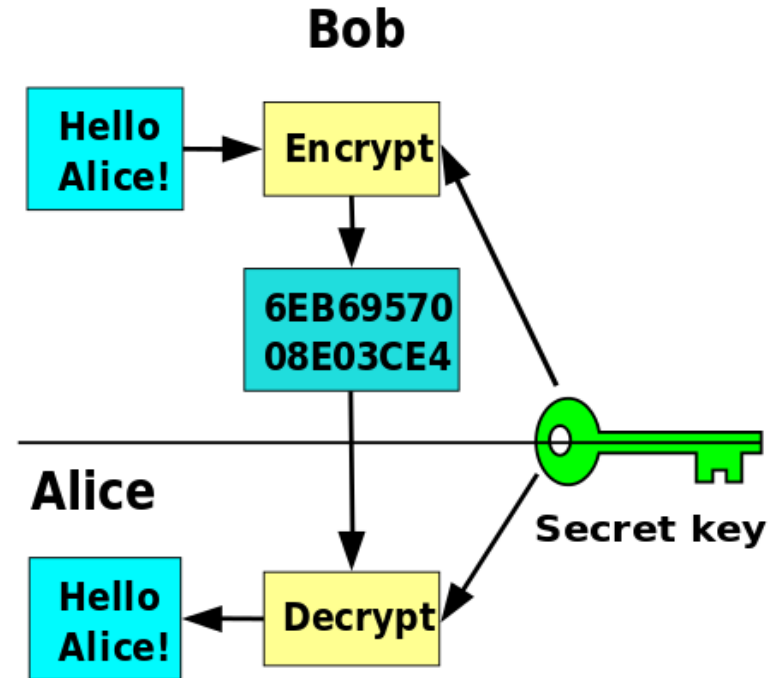
- Symmetric Encryption

- Closer to form of private key encryption
- Each party has a key that encrypts and decrypts
- After asymmetric encryption in SSL handshake, browser and server communicate with symmetric key that is passed along

Encrypting Data at Rest

1. Data at rest can use a form of encryption called **symmetric key encryption**
2. Requires both parties to use the key to encrypt and decrypt data.
3. Any party possessing the key can read the data.
4. Has difficulties securing the symmetric key amongst multiple parties.

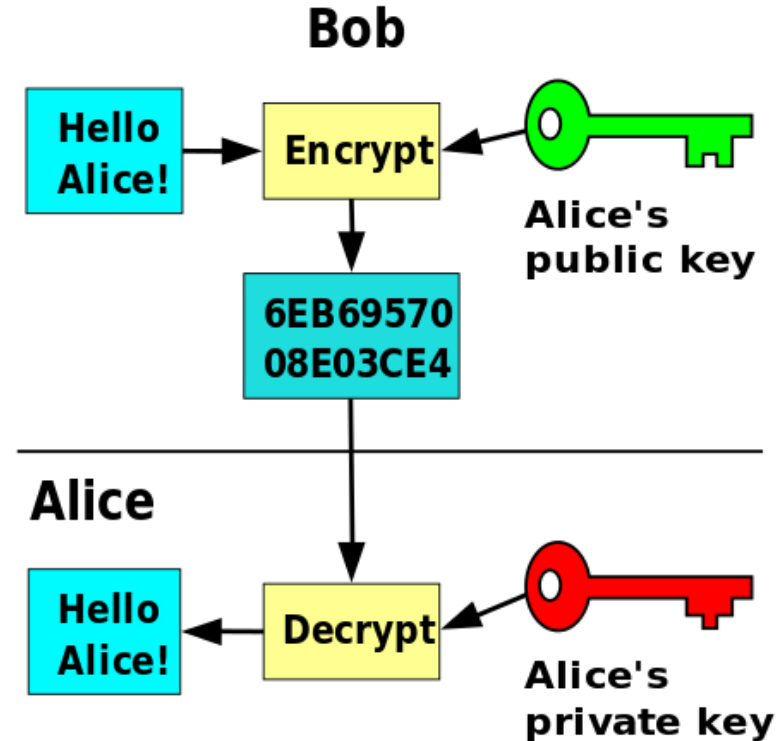
Symmetric cryptography involves the parties sharing a common secret passphrase or key. Data is encrypted and decrypted using the same key.



Securing Data in Transit

1. Data in transit can use a form of encryption called **asymmetric key encryption**
2. Uses a Private and Public Key
3. Any party can be sent the public key. It can encrypt the data, but not decrypt it.
4. Only the key owner has the private key. It can decrypt data encrypted by the public key.
5. Has difficulties securing the symmetric key amongst multiple parties.

Asymmetric cryptography uses two keys: one to encrypt the data and the other to decrypt.



Digital certificate

- Public key certificate
- Used for encryption and authentication
- Certificate authority (CA) is trusted third-party that provide certificate
 - Prevents attacker from impersonating a server

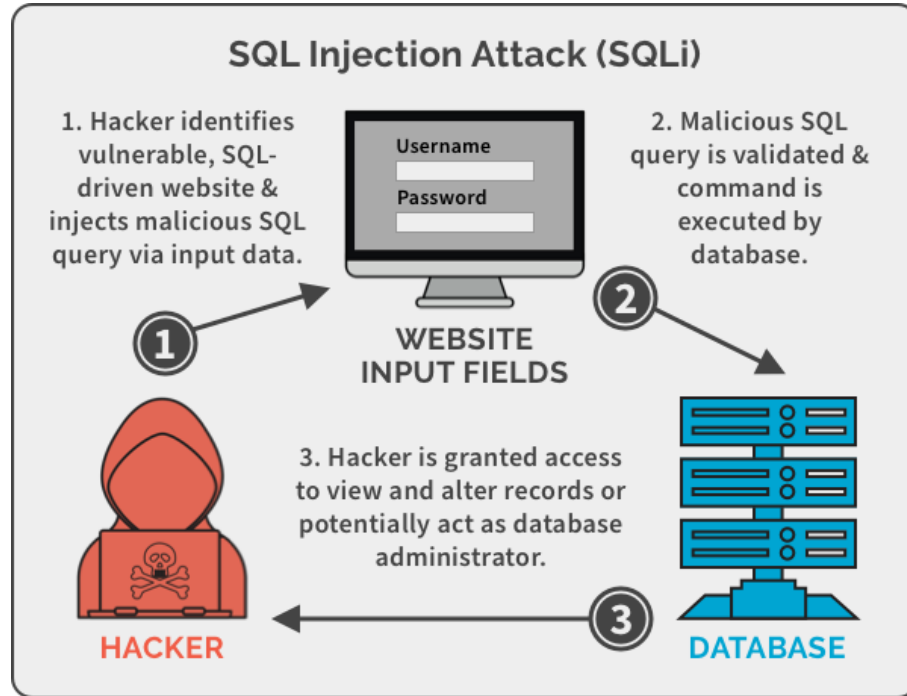
Man in the Middle Attack

- Attacker intercepts communications between two parties
 - Either to eavesdrop
 - Modify traffic
- Oldest form of cyber attacks
- Not as common as ransomware or phishing, still threat
- Encryption protocols (SSL/TLS) are best way to help protect against



Objectives

- SQL Injection Attack



Objectives

- SQL Injection Attack
- Prepared statements

```
*/  
@Override  
public User saveUser(String userName, String password) {  
    byte[] salt = passwordHasher.generateRandomSalt();  
    String hashedPassword = passwordHasher.computeHash(password, salt);  
    String saltString = new String(Base64.encode(salt));  
    long newId = jdbcTemplate.queryForObject(  
        "INSERT INTO users(username, password, salt) VALUES (?, ?, ?) RETURNING id", Long.class, userName,  
        hashedPassword, saltString);  
  
    User newUser = new User();  
    newUser.setId(newId);  
    newUser.setUsername(userName);  
  
    return newUser;  
}
```

Objectives

- SQL Injection Attack
- Prepared statements
- Hashing

```
/**
 * Given a clear text password and a salt, hash the password and return
 * the computed hash.
 *
 * @param clearTextPassword the password as given by the user
 * @param salt a salt to add to the password during hashing
 * @return the hashed password
 */
public String computeHash(String clearTextPassword, byte[] salt) {
    Key key = createKey(clearTextPassword, salt);
    byte[] digest = key.getEncoded();
    return new String(Base64.encode(digest));
}
```

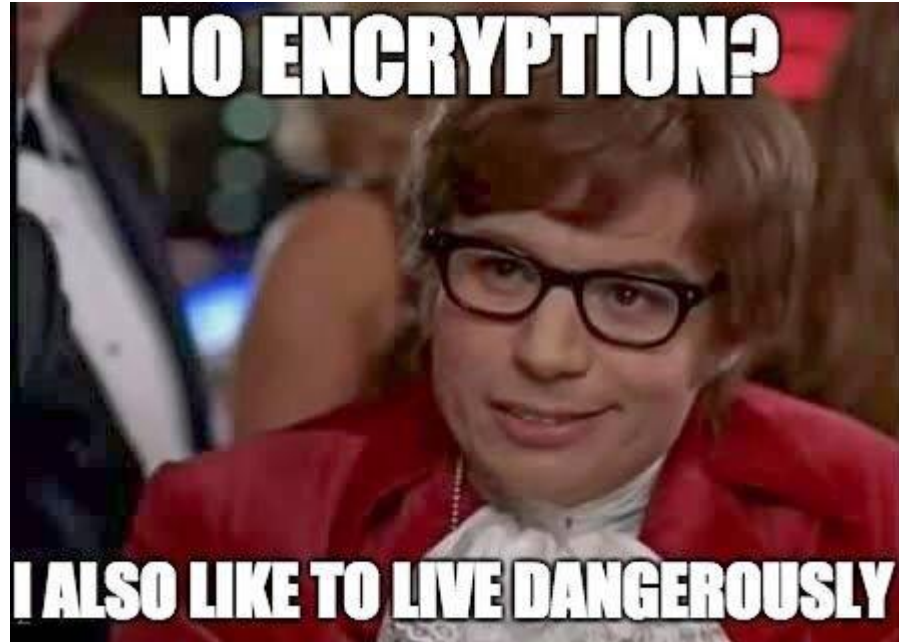
Objectives

- SQL Injection Attack
- Prepared statements
- Hashing
- Salt

```
/**  
 * Generate a new random salt.  
 *  
 * @return a new random array of bytes to be used as a salt  
 */  
public byte[] generateRandomSalt() {  
    return random.generateSeed(128);  
}
```

Objectives

- SQL Injection Attack
- Prepared statements
- Hashing
- Salt
- Encryption



Let's Code!