Breaking Databases via SQLi attacks

Azqa Nadeem
PhD Student @ Cyber Security Group
About Cyber Security lecture series

• A hot topic, a buzz term
• Introducing the Cyber Security lecture series
  – Cyber security topics in existing courses
• Announcements
  – Assignment 3
  – Exam questions

https://www.tudelft.nl/cybersecurity/
Agenda for today

• Part I
  – Data breaches and their threat landscape
  – Information Security principles
  – Top threats for databases
  – Mitigating security threats

• Part II
  – SQL injection attacks
  – Injecting SQL queries  ← Hands-on!
  – Analysing SQLi attacks
  – Best practices to avoid SQLi
Go to https://b.socrative.com/login/student/

Room Name: IDMQ3
Why would anyone ever hack a database?
The role of databases

• A database is the heart of an organization.

• “Database servers are the most compromised asset in an organization.”
  – Verizon 2018
Education and Science Giant Elsevier Left Users’ Passwords Exposed Online

Due to a misconfigured server, a researcher found a constant stream of Elsevier users’ passwords.

https://motherboard.vice.com/en_us/article/vbw8b9/elsevier-user-passwords-exposed-online
In the news

Facebook Stored Hundreds of Millions of User Passwords in Plain Text for Years

Hundreds of millions of Facebook users had their account passwords stored in plain text and searchable by thousands of Facebook employees — in some cases going back to 2012, KrebsOnSecurity has learned. Facebook says an ongoing investigation has so far found no indication that employees have abused access to this data.

In the news

Marriott Breach Exposes Far More Than Just Data

David Volodzko | Contributor ▽
Manufacturing

I am an editor at the technology and information company Brightwire.

https://www.forbes.com/sites/davidvolodzko/2018/12/04/marriott-breach-exposes-far-more-than-just-data/#1f0d3c276297
In the news

How I hacked hundreds of Bitcoins!
AMA

 hacker0 (25) • in bitcoin • 2 years ago

It all begins 3 years and a 3 month ago.

My Wallet: Be your own bank.

0 BTC €0.00

Hash: 123456

August 17th 06:19 AM
Sent

August 17th 07:03 PM
Sent

August 17th 12:18 AM
Exchanges

August 17th 10:15 AM
Exchanges

Beginning 2013:

I was a hacker who focused on phishing victims bank details and selling them. I was working full time in a company and doing this black market.

https://steemit.com/bitcoin/@hacker0/how-i-hacked-hundreds-of-bitcoins-ama
Extra notes: MyHeritage does DNA sequencing

https://informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/
The CIA triad

Extra notes:
- Can unauth see it?
- Can unauth change it?
- Can legit user access it?
# Threats to DB Security

Worst Passwords of 2018

<table>
<thead>
<tr>
<th>Rank</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>123456</td>
</tr>
<tr>
<td>2</td>
<td>password</td>
</tr>
<tr>
<td>3</td>
<td>123456789</td>
</tr>
<tr>
<td>4</td>
<td>12345678</td>
</tr>
<tr>
<td>5</td>
<td>12345</td>
</tr>
<tr>
<td>6</td>
<td>111111</td>
</tr>
<tr>
<td>7</td>
<td>1234567</td>
</tr>
<tr>
<td>8</td>
<td>sunshine</td>
</tr>
<tr>
<td>9</td>
<td>qwerty</td>
</tr>
<tr>
<td>10</td>
<td>iloveyou</td>
</tr>
<tr>
<td>11</td>
<td>princess</td>
</tr>
<tr>
<td>12</td>
<td>admin</td>
</tr>
<tr>
<td>13</td>
<td>welcome</td>
</tr>
<tr>
<td>14</td>
<td>666666</td>
</tr>
<tr>
<td>15</td>
<td>abc123</td>
</tr>
<tr>
<td>16</td>
<td>football</td>
</tr>
<tr>
<td>17</td>
<td>123123</td>
</tr>
<tr>
<td>18</td>
<td>monkey</td>
</tr>
<tr>
<td>19</td>
<td>654321</td>
</tr>
<tr>
<td>20</td>
<td>!@$%^&amp;*</td>
</tr>
<tr>
<td>21</td>
<td>charlie</td>
</tr>
<tr>
<td>22</td>
<td>aa123456 w</td>
</tr>
<tr>
<td>23</td>
<td>donald</td>
</tr>
<tr>
<td>24</td>
<td>password1</td>
</tr>
<tr>
<td>25</td>
<td>qwerty123</td>
</tr>
</tbody>
</table>

Threats to DB Security

1. Weak authentication

Verizon 2017-2018 Data Breach Investigations Report
Threats to DB Security

1. Weak authentication

Extra notes:
- Default username/passwords
- Easy-to-guess passwords
- Passwords written on sticky notes

Verizon 2017-2018 Data Breach Investigations Report
Threats to DB Security

1. Weak authentication
2. ?
Threats to DB Security

1. Weak authentication
2. Excessive database privileges
3. (Inadvertent) Insider threats

Verizon 2017-2018 Data Breach Investigations Report
Threats to DB Security

1. Weak authentication
2. Excessive database privileges
3. (Inadvertent) Insider threats

Extra notes:
- Giving away privileges like they’re candy.
- Insider threats – Tricky business -- balance between convenience and security
- Phishing attacks

Verizon 2017-2018 Data Breach Investigations Report
Threats to DB Security

1. Weak authentication
2. Excessive database privileges
3. (Inadvertent) Insider threats
4. ?
Threats to DB Security

1. Weak authentication
2. Excessive database privileges
3. (Inadvertent) Insider threats
4. DB injection attacks
   - SQL injection attacks

Verizon 2017-2018 Data Breach Investigations Report
Threats to DB Security

1. Weak authentication
2. Excessive database privileges
3. (Inadvertent) Insider threats
4. DB injection attacks
   - SQL injection attacks
   - NoSQL injection (NoSQL does not mean you are safe!)

From the MongoDB documentation

- "One valid way to run the Mongo database is in a trusted environment, with no security and authentication"
- This "is the default option and is recommended"
Threats to DB Security

1. Weak authentication
2. Excessive database privileges
3. (Inadvertent) Insider threats
4. DB injection attacks
5. ?
Threats to DB Security

1. Weak authentication
2. Excessive database privileges
3. (Inadvertent) Insider threats
4. DB injection attacks
5. Unmanaged sensitive data
   - Storing sensitive data unprotected
Threats to DB Security

1. Weak authentication
2. Excessive database privileges
3. (Inadvertent) Insider threats
4. DB injection attacks
5. Unmanaged sensitive data
6. ?
Threats to DB Security

1. Weak authentication
2. Excessive database privileges
3. (Inadvertent) Insider threats
4. DB injection attacks
5. Unmanaged sensitive data
6. Vulnerable DBs
   - or unpatched Operating System
   - Causing DoS attack

Extra notes:
Equifax (credit risk assessment) had a major breach exposing personal information of about 143M people. The breach of was caused due to an unpatched apache web server.
The CIA triad

Is the information protected?

Is the information accurate?

Can I access the information?
Mitigating DB security threats

- Encrypting databases
  - Data-in-transit
  - Data-at-rest
- Never use default usernames/passwords
- Use 2nd Factor Authentication
- Least privilege – need-to-know basis
- Log everything!!
- Update everything regularly
- Maintaining regular backups in air gapped environment
- Disable public error reporting
- Messy architecture means difficult maintenance
- Employee awareness – humans are the weakest link

Extra notes:
Summary Part I

- Databases are the heart of an organization
- Information security – CIA triad
- Databases face a number of threats
  - Weak authentication and insider threats are the most common
- Awareness and simple security practices can mitigate those threats
Agenda for today

• Part I
  – Data breaches and their threat landscape
  – Information Security principles
  – Top threats for databases
  – Mitigating security threats

• Part II
  – SQL injection attacks
  – Injecting SQL queries ← Hands-on!
  – Analysing SQLi attacks
  – Best practices to avoid SQLi
SQL Injection

- SQL Injection (SQLi) refers to an injection attack wherein an attacker can execute malicious SQL statements that control a web application’s database server (also known as RDBMS).

- Look out if you have:
  - Web application
  - Data stored in databases
  - User-controlled parameters

**Extra notes:**
Can affect any website or web application that makes use of an SQL-based database, so this vulnerability is one of the oldest, most prevalent and most dangerous of web application vulnerabilities.

https://www.acunetix.com/websitesecurity/sql-injection/
How does a typical web app work?

Extra notes:
Who is to blame?
- Database developers?
- Oracle?
- Web developer?
- Schema designers?

```python
cur = db.cursor()
query = "SELECT * from items where itemName='" + qParam + "';"
cur.execute(query)
```
What can attackers do?

• Insert backdoor
  – `INSERT INTO users (username, password) VALUES ('attacker', 'youvebeenhashed')`

• Steal information
  – `SELECT * FROM users WHERE userType='admin'`

• Delete records/tables
  – `DELETE FROM users;`
  – `DROP SCHEMA webshop;`
Scenario

"SELECT itemPicture FROM inventory WHERE itemName LIKE "%" + "" + ";"
Scenario

"SELECT itemPicture FROM inventory WHERE itemName LIKE ' ' + car + ' ';"
Task1: How to list all items?

```
"SELECT itemPicture FROM inventory WHERE itemName LIKE ' ' + ?? + ";"
```

Inventory

<table>
<thead>
<tr>
<th>itemName</th>
<th>itemPicture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shirt</td>
<td>X</td>
</tr>
<tr>
<td>Pen</td>
<td>X</td>
</tr>
<tr>
<td>Car</td>
<td>X</td>
</tr>
</tbody>
</table>
Task1: How to list all items?

SELECT itemPicture FROM inventory WHERE itemName LIKE 'car' OR 1 # ,→ Tautology

inventory

<table>
<thead>
<tr>
<th>itemName</th>
<th>itemPicture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shirt</td>
<td>X</td>
</tr>
<tr>
<td>Pen</td>
<td>X</td>
</tr>
<tr>
<td>Car</td>
<td>X</td>
</tr>
</tbody>
</table>

Inventory
The login scenario...

Extra notes:
= missing after username

"SELECT status FROM users WHERE username='" + ?? + "' AND password='" + ?? + "';"
Another Tautology-based SQLi

```
SELECT status FROM users WHERE 
username= 'Blah' OR 1 # ' AND 
password= 'Blah';
```
Running multiple queries

• Useful keywords:
  – JOIN (Append horizontally)
  – UNION (Append vertically)

• `SELECT firstName, lastName FROM users;`
  
  Fluffy | Bunny

• `SELECT firstName, lastName FROM users`  
  
  `UNION`

• `SELECT 1,2 from dual;`
  
  Fluffy | Bunny
  1      2

Extra notes:
Dual is a one row, one column table in Oracle databases, called Dummy with value X.
Task 2: How to dump user data?

"SELECT itemPicture FROM inventory WHERE itemName LIKE '' + ?? + '';"
Task 2: How to dump user data?

Extra notes:
It's called: Union-based SQLi attack

```
SELECT itemPicture FROM inventory WHERE itemName LIKE 'car' UNION
SELECT password from users#
```
HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.

OH, DEAR—DID HE BREAK SOMETHING? IN A WAY—

DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?

OH, YES. LITTLE BOBBY TABLES, WE CALL HIM.

WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.

AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Piggy backed query

Why is it happening?

- Mixing of Code and data

```
SELECT profile FROM users WHERE
uname= 'Blah' AND pwd= 'Blah'
```

```
SELECT profile FROM users WHERE
uname= 'Blah' OR 1=1 # ' AND pwd= 'Blah'
```
Why is it happening?

• Mixing of Code and data

SELECT profile FROM users WHERE
uname= 'Blah' AND pwd= 'Blah'

SELECT profile FROM users WHERE
uname= 'Blah' OR 1=1 # ' AND pwd= 'Blah'
SQLi Avoidance

1. Input sanitization
   - Clean the input in order to use it
     ```python
     if re.match("^[a-zA-Z0-9 ]*$", inputField):
         replaced = re.sub("^[a-zA-Z0-9 ]*$", '', inputField)
     inputField = replaced
     ```
   - Problem:
     - Not all scenarios are known
SQLi Avoidance

1. Input sanitization
2. Escaping the input
   - To avoid *data* being mistaken as *code*
   - Input: ‘what is ‘www”
   - Processed as: 'what is 'www''
   - Must be processed as: 'what is \'www\''

- Problem:
  - Possibly a 2\textsuperscript{nd} Order SQLi attack
    - Effect not seen immediately
2\textsuperscript{nd} Order SQLi

<table>
<thead>
<tr>
<th>Username</th>
<th>Robert' Drop table users;#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Password</td>
<td>Blah</td>
</tr>
<tr>
<td>Register</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Username</th>
<th>Robert' Drop table users;#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Password</td>
<td>Blah</td>
</tr>
<tr>
<td>Login</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Welcome, Robert' Drop table users;#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Password</td>
</tr>
<tr>
<td>Confirm</td>
</tr>
<tr>
<td>Update password</td>
</tr>
</tbody>
</table>

**INSERT INTO users values**

('Robert\'; Drop table users;\'#', 'Blah');

**select profile from users where**

username='Robert\'; Drop table users;#' and password='Blah';

**update users set password='Blah2'**

where username='Robert'; Drop table users;#';
SQLi Avoidance

1. Input sanitization
2. Escaping the input
3. Prepared statements
   - Separation of concerns
   - Pre-compile legitimate query
   - Add placeholders for data

```java
statement = con.prepareStatement("SELECT profile FROM users where userName=?");
statement.setString(1, inputField);
statement.executeUpdate();
```

Extra notes:
Learn more about Prepared statements here: [https://youtu.be/jTasm64rz-c](https://youtu.be/jTasm64rz-c) and [https://stackoverflow.com/questions/23845383/what-does-it-mean-when-i-say-prepared-statement-is-pre-compiled](https://stackoverflow.com/questions/23845383/what-does-it-mean-when-i-say-prepared-statement-is-pre-compiled)
Summary Part II

• Executing SQL code on a database is called an SQL Injection attack
• SQLi is caused by mixing of code and data
• Prepared statements are the most useful in avoiding SQLi
• However, user input must always be sanitized

Extra notes:
Prepared statements can be used in all cases EXCEPT when using Dynamic Object Mappers (e.g. Hibernate, Jackson) because we don’t have variables to bind with beforehand. In such cases, escaping and sanitizing user input are the only options.
Additional material


• NoSQL injection attacks:


• Type of SQLi attacks: [https://pdfs.semanticscholar.org/81a5/02b52485e52713ccab6d260f15871c2acdc8b.pdf](https://pdfs.semanticscholar.org/81a5/02b52485e52713ccab6d260f15871c2acdc8b.pdf)

• Try it yourself:
Time for questions