

2319368_I5_databases_and_application_dev_gaynor_php_report

by Matthew Evans

Submission date: 10-Jan-2025 09:20AM (UTC+0000)

Submission ID: 247903690

File name:

51109_Matthew_Evans_2319368_I5_databases_and_application_dev_gaynor_php_report_630034_1559737124.docx
(1.78M)

Word count: 617

Character count: 3519

ACCA5026 – L5
DATABASES &
APPLICATION
DEVELOPMENT REPORT
MATTHEW EVANS

Matthew Evans

STUDENT NUMBER: 2319368 EMAIL: 2319368@student.uwtsd.ac.uk

Table of Contents

PART B: SQL Injection	5
Summary.....	5
SQL injection example.....	5
Sony PlayStation Network SQL Injection Attack (2011)	6
Impact on Sony.....	6
SQL Injection and Prevention – Music Catalogue	7
Search page.....	7
Original code in search.php (problem)	7
 48 // Prepare and execute the SQL query using prepared statements	
49 \$query = "	
50 SELECT	
51 albums.album_id AS album_id,	
52 albums.title AS album_title,	
53 albums.release_date,	
54 albums.cover_image,	
55 artists.name AS artist_name,	
56 genres.name AS genre_name,	
57 producers.name AS producer_name,	
58 tracks.track_number,	
59 tracks.title AS track_title,	
60 tracks.duration	
61 FROM	
62 albums	
63 LEFT JOIN artists ON albums.artist_id = artists.artist_id	
64 LEFT JOIN genres ON albums.genre_id = genres.genre_id	
65 LEFT JOIN producers ON albums.producer_id = producers.producer_id	
66 LEFT JOIN tracks ON albums.album_id = tracks.album_id	
67 WHERE	
68 albums.title LIKE '%\$userSearch%'	
69 ORDER BY	
70 albums.title, tracks.track_number	
71 ";	7
.....	7
73 \$stmt = \$link->prepare(\$query);	
74 \$stmt->execute();	
75 \$result = \$stmt->get_result();	7
Search page - Attempted Attack (Pre solution)	8
Action	8

Music Catalogue

Your ultimate database for music albums, artists, and genres

[Home](#)[Albums](#)[Artists](#)[Admin](#)

Find Your Favorite Album

' OR 1=1 --

Search

Popular Albums



The Blackening
- Machine Head



Master of Puppets -
Metallica



Born in the USA
- Bruce Springsteen

© 2024 Music Catalogue. All rights reserved.

Result 8

You searched for: ' OR 1=1 -- '



1984

Artist: Van Halen
Genre: Rock
Producer(s): Ted Templeman
Release Date: 1984-01-09

Tracklist

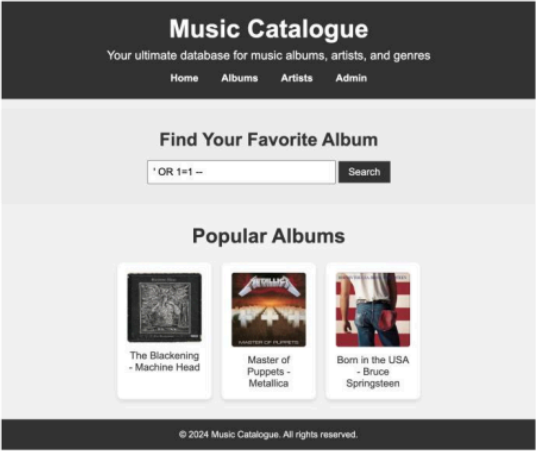
1	1984	00:01:07
2	Jump	00:04:02

..... 8

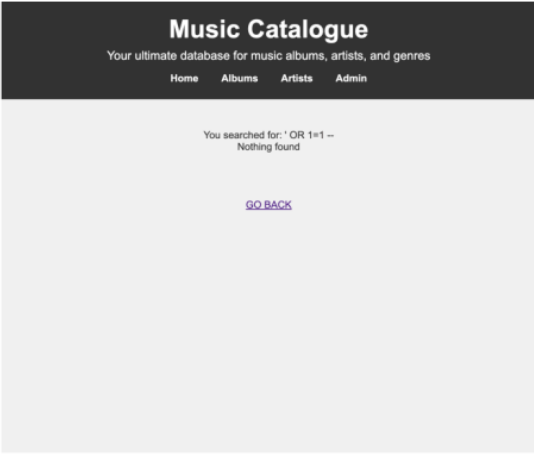
2

1	Pray for Rain	00:04:45
2	Hypermania	00:02:42
3	Masochist	00:04:08
4	Landmine	00:03:49
5	Vagabond	00:04:17
6	Creatures of Habit	00:04:20
7	Above My Head	00:04:42
8	Martyr (Waves)	00:03:50
9	All of This Is Fleeting	00:03:56
10	The Descent	00:05:18
1	Friends	00:03:27
2	The Extremist (Living On The Edge)	00:03:42
3	War	00:05:46
4	Cryin	00:05:42
5	Rubina's Blue Sky Happiness	00:06:10
6	Summer Song	00:04:52
7	Tears In The Rain	00:01:16
8	Why	00:04:45
9	Motorcycle Driver	00:04:56
10	New Blues	00:06:55

	9
Prevention	10
Search page - Attempted Attack (Post solution)	11
Action	11



	11
Result	11



.....11

References..... 12

PART B: SQL Injection

Summary

With the rise of cloud computing and Software as a Service (SaaS), web portals have become the primary way to access applications, often powered by backend SQL servers. [1]

Given how widespread SQL is attackers will seize any chance to exploit unprotected SQL-based applications to gain access to our data and systems. It's crucial that we safeguard our systems to prevent this.

SQL Injection (SQLi) is a cyberattack where malicious SQL code is injected into web application input fields, exploiting database communication weaknesses. This allows attackers to access, manipulate, or control data. Preventing SQLi requires security measures like input validation, parameterized queries, and machine learning-based detection systems like SQLR34P3R. [2]

SQL injection example

Fig.1

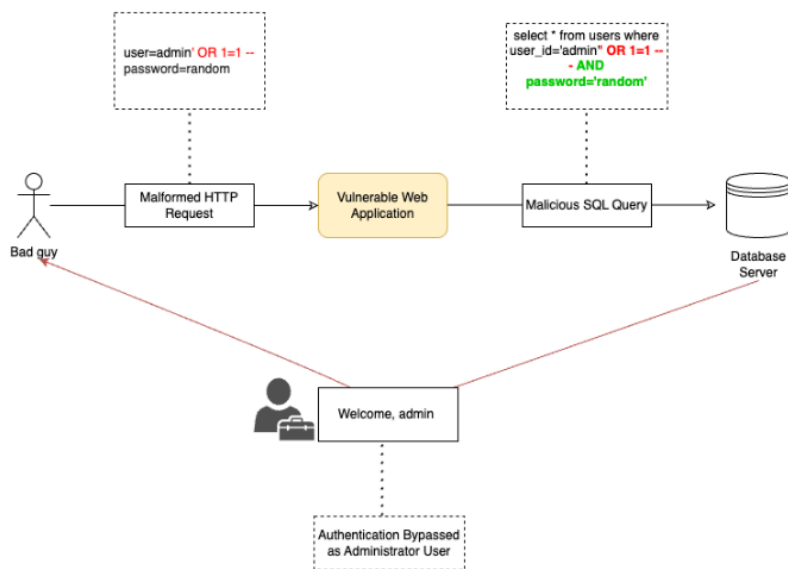


Fig. 1. An example of SQL injection.

An example of an SQLi attack used to bypass authentication is shown in **Fig. 1**. In this case, the attacker injects the SQLi payload "admin'OR 1=1 --" into the "user" field of a login page, using any arbitrary password. The malicious input, "admin'OR 1=1 --", always evaluates to true and is then directly appended to the original SQL query. The "--" part comments out the rest of the query, allowing the attacker to bypass password verification and successfully authenticate to the web application.

Sony PlayStation Network SQL Injection Attack (2011)

In 2011, the Sony PlayStation Network (PSN) was compromised due to a combination of SQL injection and other attack methods. Over 75 million homes were connected to the platform, making games social experiences. Hackers exploited vulnerabilities in Sony's network, using phishing and SQL injection to access servers and steal personal data from 77 million user accounts, including names, addresses, emails, and credit card details. The breach caused a major service outage, financial losses, and a loss of customer trust. Sony spent millions on security upgrades and compensating users. LulzSec later claimed responsibility for the attack, stating they accessed SonyPictures.com with a simple SQL injection and criticised the company for leaving itself open to such basic vulnerabilities. [4]

Impact on Sony

The cyberattack on Sony resulted in significant damage, affecting both its financials and reputation. The breach targeted Sony's PlayStation Network and Sony Online Entertainment services, leading to the theft of personal data from over 100 million accounts, including names, passwords, and addresses. The company estimated that the attack would cost \$170 million in operating profits. [5]

SQL Injection and Prevention – Music Catalogue

Search page

Original code in search.php (problem)

```
48 // Prepare and execute the SQL query using prepared statements
49 $query = "
50 SELECT
51     albums.album_id AS album_id,
52     albums.title AS album_title,
53     albums.release_date,
54     albums.cover_image,
55     artists.name AS artist_name,
56     genres.name AS genre_name,
57     producers.name AS producer_name,
58     tracks.track_number,
59     tracks.title AS track_title,
60     tracks.duration
61 FROM
62     albums
63 LEFT JOIN artists ON albums.artist_id = artists.artist_id
64 LEFT JOIN genres ON albums.genre_id = genres.genre_id
65 LEFT JOIN producers ON albums.producer_id = producers.producer_id
66 LEFT JOIN tracks ON albums.album_id = tracks.album_id
67 WHERE
68     albums.title LIKE '%$userSearch%'
69 ORDER BY
70     albums.title, tracks.track_number
71 ";
```

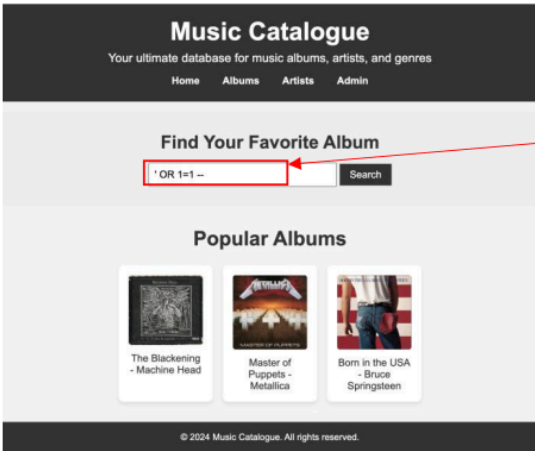
Directly embedding user input into the SQL statement, leaving it vulnerable to SQL injection

```
73 $stmt = $link->prepare($query);
74 $stmt->execute();
75 $result = $stmt->get_result();
```

No statement binding occurs here, meaning the search query will be used in execution as source type.

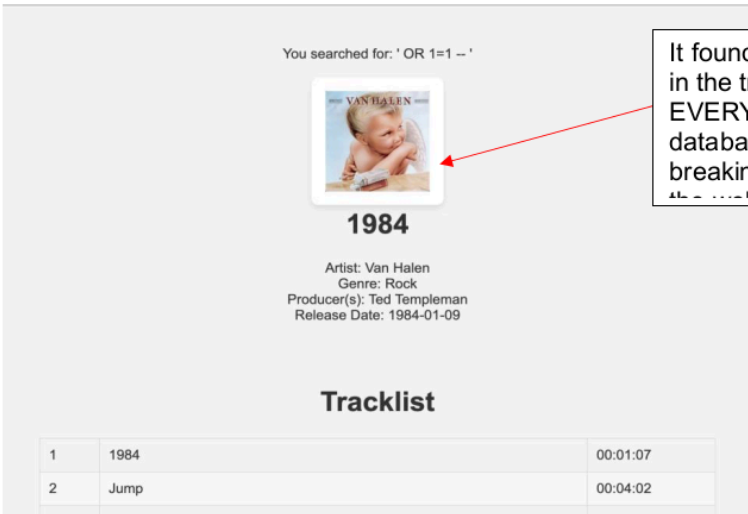
Search page - Attempted Attack (Pre solution)

Action



This SQL will evaluate to true and return everything if the code hasn't been treated against SQL injection

Result



It found an album however, in the track list below, EVERY track stored in the database was outputted, breaking the behaviour of the website

1	Pray for Rain	00:04:45
2	Hypermania	00:02:42
3	Masochist	00:04:08
4	Landmine	00:03:49
5	Vagabond	00:04:17
6	Creatures of Habit	00:04:20
7	Above My Head	00:04:42
8	Martyr (Waves)	00:03:50
9	All of This Is Fleeting	00:03:56
10	The Descent	00:05:18
1	Friends	00:03:27
2	The Extremist (Living On The Edge)	00:03:42
3	War	00:05:46
4	Cryin	00:05:42
5	Rubina's Blue Sky Happiness	00:06:10
6	Summer Song	00:04:52
7	Tears In The Rain	00:01:16
8	Why	00:04:45
9	Motorcycle Driver	00:04:56
10	New Blues	00:06:55

If a website with 10 million tracks repeatedly loops through the entire track list, it could crash the server and disrupt users. Additionally, running such a large query would incur significant costs in a serverless framework.

Prevention

The SELECT Statement

```
48 // Prepare and execute the SQL query using prepared statements
49 $query = "
50 SELECT
51     albums.album_id AS album_id,
52     albums.title AS album_title,
53     albums.release_date,
54     albums.cover_image,
55     artists.name AS artist_name,
56     genres.name AS genre_name,
57     producers.name AS producer_name,
58     tracks.track_number,
59     tracks.title AS track_title,
60     tracks.duration
61 FROM
62     albums
63 LEFT JOIN artists ON albums.artist_id = artists.artist_id
64 LEFT JOIN genres ON albums.genre_id = genres.genre_id
65 LEFT JOIN producers ON albums.producer_id = producers.producer_id
66 LEFT JOIN tracks ON albums.album_id = tracks.album_id
67 WHERE
68     albums.title LIKE ?
69 ORDER BY
70     albums.title, tracks.track_number
71 ";
```

The query uses placeholders (?) to separate SQL logic from data, preventing malicious input from altering the query.

Query handling

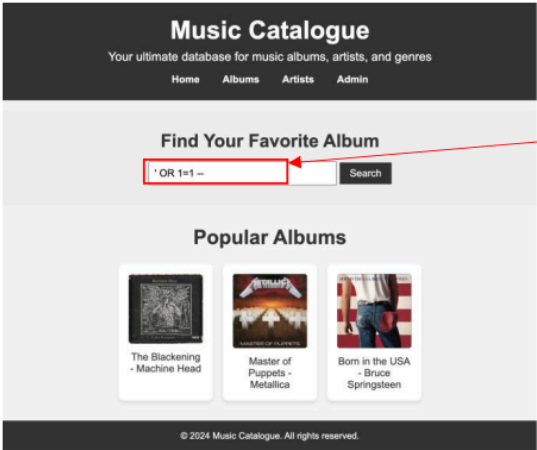
Bind param treating input as a value not as part of the query.

```
$stmt = $link->prepare($query);
$searchParam = $userSearch . '%';
$stmt->bind_param("s", $searchParam);
$stmt->execute();
$result = $stmt->get_result();
```

The bind_param method binds the user input (\$userSearch) as a string parameter, ensuring it's treated as data, not part of the query.

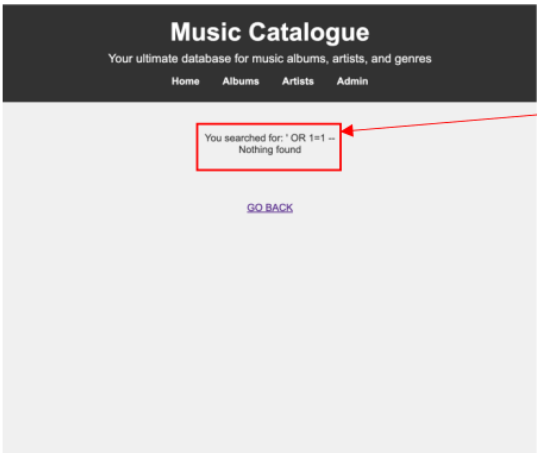
Search page - Attempted Attack (Post solution)

Action



This SQL will evaluate to true and return everything if the code hasn't been treated against SQL injection

Result



It treated the SQL line of code as a string literal and as a result, found NO matches

References

[1]

M. Shore, linkedin.com, and linkedin.com, Carpenteria, CA. *Ethical Hacking: SQL Injection.*, (2021).

[2]

A. Paul, V. Sharma, and O. Olukoya, "SQL injection attack: Detection, prioritization & prevention," *Journal of information security and applications*, vol. 85, pp. 103871-, 2024, doi: 10.1016/j.jisa.2024.103871.

[3]

S. Team, "Sony Data Breach: What Happened and How to Prevent It," *Strongdm.com*, Jul. 18, 2024. <https://www.strongdm.com/what-is/sony-data-breach> (accessed Dec. 03, 2024).

[4]

B. News, "Sony investigating another hack," *BBC News*, Jun. 03, 2011. Accessed: Dec. 03, 2024. [Online]. Available: <https://www.bbc.co.uk/news/business-13636704>

[5]

B. News, "Sony confirms \$3.1bn annual loss," *BBC News*, May 26, 2011. <https://bbc.co.uk/news/business-13557431> (accessed Dec. 03, 2024).

FINAL GRADE

73/100

GENERAL COMMENTS

Database Structure and Design - Not sure why you a repeating genre in one of the tables when it's already normalised into its own table 8/10

Range of back-end coding - Good to see you've used index.php so that the front page on the site loads automatically.

Search is OK, but it would be nice to be able to search on multiple fields such as band name

Add is OK with validation - genre really should be a pulldown menu. I appreciate that you've validated it, but a pulldown menu would be much easier (this is linked to the lack of normalisation mentioned above)

Good range of table joins shown.

I'm not sure what's going on with Edit, but it doesn't seem to be quite working, though it does have validation. Delete is OK

Login is well-managed with URL attacks accounted for.

37/50

Overall integration of website - A good layout with a navigation bar. Perhaps a little more colour contrast, it's very subtle grey above while.

9/10

SQL Injection overview - Some good background research, could have had a few more examples
10.5/15

SQL Code implementation/proposal - Well shown, but could also have implemented on other queries such as Add 9/15

Overall mark for PHP assignment: 72.5%

Overall mark for module (including Divya's assignment) - 67%

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13
