Blind SQL Injection

Introduction to MSSQL/SQL Server

Introduction

`SQL` is a standardized language for interacting with `relational databases`. The five most common (as of Dec 15, 2022) are:

1. Oracle
2. MySQL
3. Microsoft SQL Server
4. PostgreSQL
5. IBM Db2

In this module, we will be focusing on `blind SQL injection` attacks using examples in `Microsoft SQL Server` (`MSSQL`). In addition to this, we will cover `MSSQL-specific` attacks. As SQL is standardized, the attacks taught in this module may be easily adapted to work against other relational databases.

Interacting with MSSQL

Although we will be dealing with injection vulnerabilities through websites for the rest of this module, it is helpful to understand how to interact with `MSSQL/SQLServer` directly, be it through a command line or GUI application.

Note: As this is an advanced SQL module, it is expected that you already understand the basics of SQL and are comfortable building queries yourself.

SQLCMD (Windows, Command Line)

SQLCMD is a `command-line` tool for `Windows` developed by `Microsoft` for interacting with `MSSQL`.

To connect to a `SQL Server` we can use the following syntax. In this case, we are connecting to the `bsqlintro` database on the server `SQL01` with the credentials `thomas:TopSecretPassword23!`. The last flag (`-W`) removes trailing spaces, which makes the output a bit easier to read.

```
PS C:\htb> sqlcmd -S 'SQL01' -U 'thomas' -P 'TopSecretPassword23!' -d
bsqlintro -W
1>
```

To run SQL queries, simply enter them and type `GO` (which is the default `batch` separator) at the end to run. In this example we select all `table information`, and then the `top 5` posts from the `users` table joined with the `posts` table.

```
PS C:\htb> sqlcmd -S 'SQL01' -U 'thomas' -P 'TopSecretPassword23!' -d
bsqlintro -W
1> SELECT *
2> FROM INFORMATION_SCHEMA.TABLES;
3> GO
TABLE_CATALOG TABLE_SCHEMA TABLE_NAME TABLE_TYPE
------------- ------------ ---------- ----------
bsqlintro dbo users BASE TABLE
bsqlintro dbo posts BASE TABLE

(2 rows affected)
1> SELECT TOP 5 users.firstName, users.lastName, posts.title
2> FROM users
3> JOIN posts
4> ON users.id=posts.authorId;
5> GO
firstName lastName title
--------- -------- -----
Edward Strong Voluptatem neque labore dolore velit ut.
David Ladieu Etincidunt etincidunt adipisci sed consectetur.
Natasha Ingham Aliquam quiquia velit non aliquam sed sit etincidunt.
Jessica Fitzpatrick Dolor porro quiquia labore numquam numquam sit.
Mary Evans Tempora sed velit consectetur labore consectetur.

(5 rows affected)
```

Impacket-MSSQLClient (Linux, Command Line)

MSSQLClient.py (or `impacket-mssqlclient`) is part of the Impacket toolset which comes preinstalled on many security-related linux distributions. We can use it to interact with remote `MSSQL` without having to use Windows.

The syntax to connect looks like this:

```
mayala@htb[/htb]$ impacket-mssqlclient thomas:'TopSecretPassword23!'@SQL01 -db bsqlintro
```

We can run queries as usual:

```
mayala@htb[/htb]$ impacket-mssqlclient thomas:'TopSecretPassword23!'@SQL01 -db bsqlintro Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation [*] Encryption required, switching to TLS [*] ENVCHANGE(DATABASE): Old Value: master, New Value: bsqlintro [*] ENVCHANGE(LANGUAGE): Old Value: , New Value: us_english [*] ENVCHANGE(PACKETSIZE): Old Value: 4096, New Value: 16192 [*] INFO(SQL01): Line 1: Changed database context to 'bsqlintro'. [*] INFO(SQL01): Line 1: Changed language setting to us_english. [*] ACK: Result: 1 - Microsoft SQL Server (150 7208) [!] Press help for extra shell commands SQL> SELECT * FROM INFORMATION_SCHEMA.TABLES; TABLE_CATALOG TABLE_SCHEMA TABLE_NAME TABLE_TYPE ------------------------------------------------------------------ ------------------------------------------------------------------ ------------------------------------------------------------------ ---------------------------------------------------------- bsqlintro dbo users b'BASE TABLE' bsqlintro dbo posts b'BASE TABLE' SQL> SELECT TOP 5 users.firstName, users.lastName, posts.title FROM users JOIN posts ON users.id=posts.authorId; firstName lastName title ------------------------------------------------------------ ------------------------------------------------------------ ------------------------------------------------------------ b'Edward' b'Strong' b'Voluptatem neque labore dolore velit ut.' b'David' b'Ladieu' b'Etincidunt etincidunt adipisci sed consectetur.' b'Natasha' b'Ingham' b'Aliquam quiquia velit non aliquam sed sit etincidunt.' b'Jessica' b'Fitzpatrick' b'Dolor porro quiquia labore numquam numquam sit.' b'Mary' b'Evans' b'Tempora sed velit consectetur labore consectetur.' SQL> exit
```

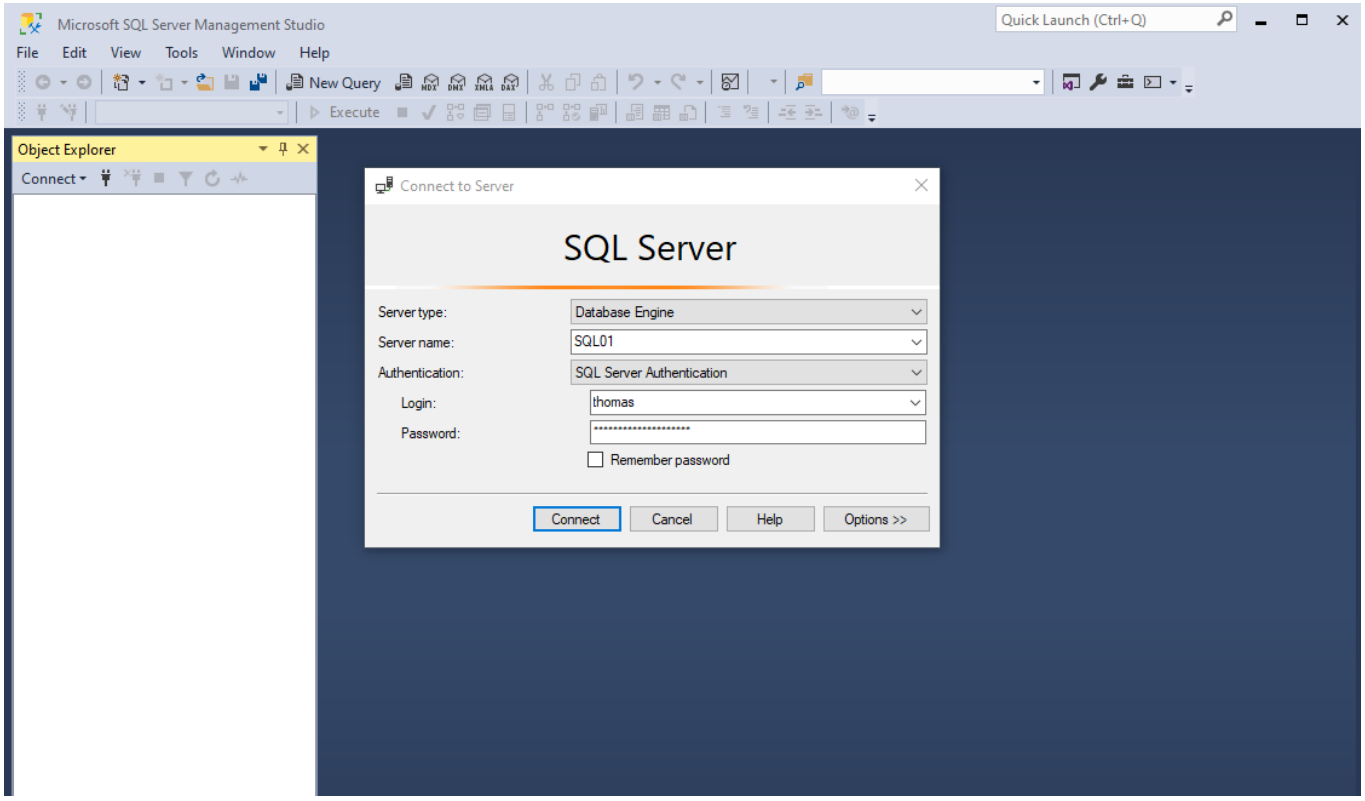
Since `MSSQLClient.py` is a pen-testing tool, it has a couple of features that help us when attacking `MSSQL` servers. For example, we can enable and use `xp_cmdshell` to run commands. We will cover this later on in the module.

```
mayala@htb[/htb]$ impacket-mssqlclient thomas:'TopSecretPassword23!'@SQL01 -db bsqlintro Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation [*] Encryption required, switching to TLS [*] ENVCHANGE(DATABASE): Old Value: master, New Value: bsqlintro [*] ENVCHANGE(LANGUAGE): Old Value: , New Value: us_english [*] ENVCHANGE(PACKETSIZE): Old Value: 4096, New Value: 16192 [*] INFO(SQL01): Line 1: Changed database context to 'bsqlintro'. [*] INFO(SQL01): Line 1: Changed language setting to us_english. [*] ACK: Result: 1 - Microsoft SQL Server (150 7208) [!] Press help for extra shell commands SQL> enable_xp_cmdshell [*] INFO(SQL01): Line 185: Configuration option 'show advanced options' changed from 1 to 1. Run the RECONFIGURE statement to install.
```

```
[*] INFO(SQL01): Line 185: Configuration option 'xp_cmdshell' changed from 1 to
1. Run the RECONFIGURE statement to install. SQL> xp_cmdshell whoami exitoutput
--------------------------------------------------------------------------------
NT SERVICE\mssqlserver NULL SQL> exit
```

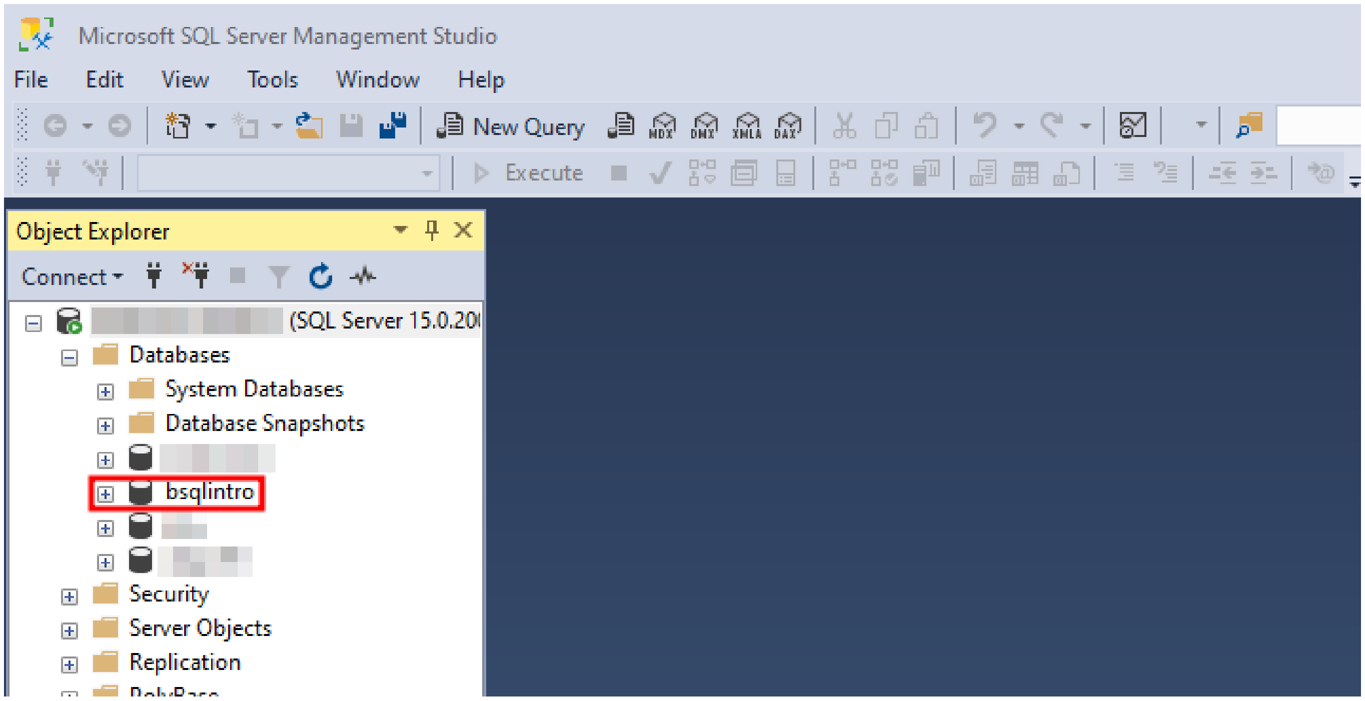
SQL Server Management Studio (Windows, GUI)

SQL Server Management Studio is a GUI tool developed by `Microsoft` for interacting with `MSSQL`. When launching the application we are prompted to connect to a server:

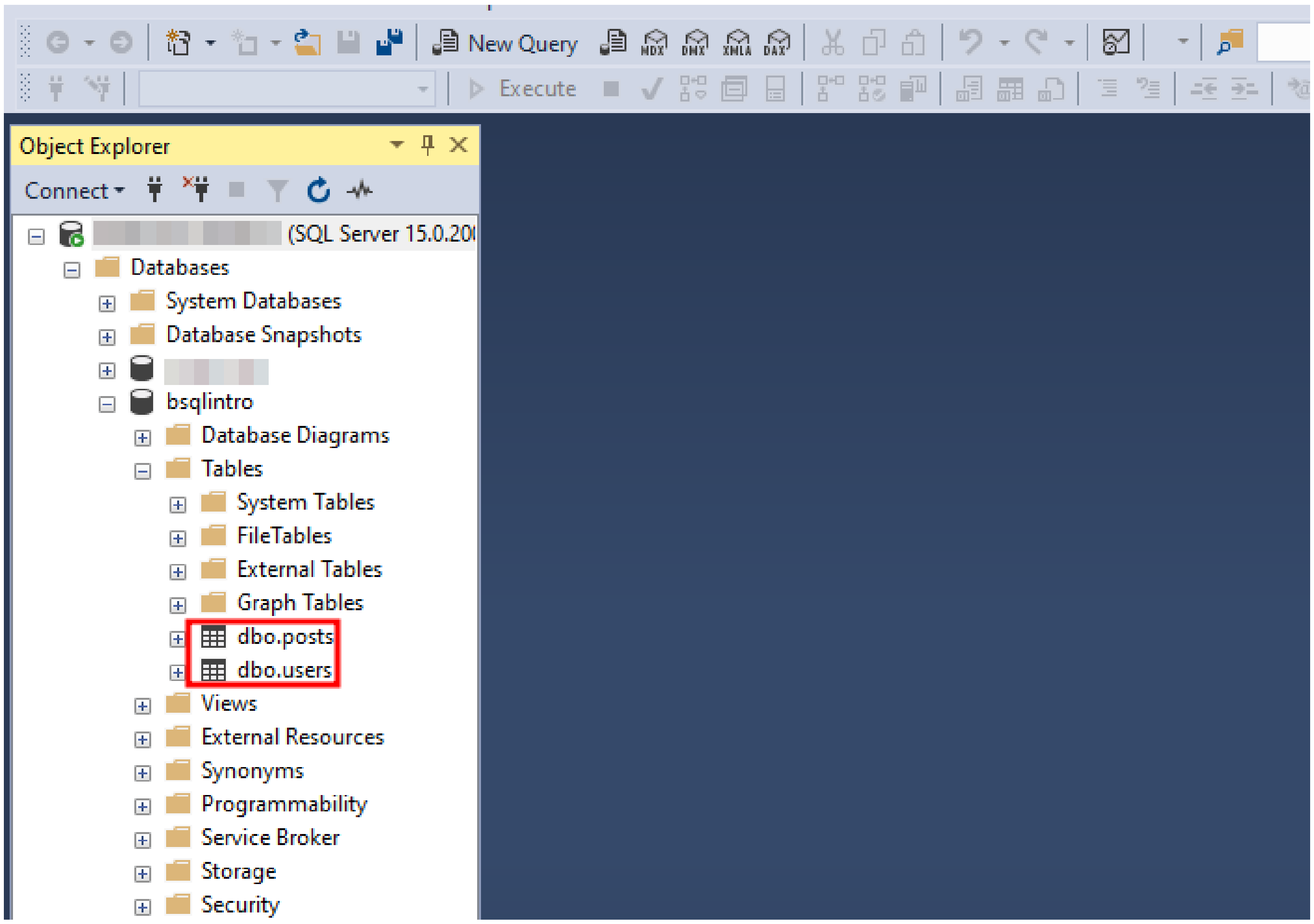


Server type: Database Engine
Server name: SQL01
Authentication: SQL Server Authentication
Login: thomas
Password: ********************
Remember password

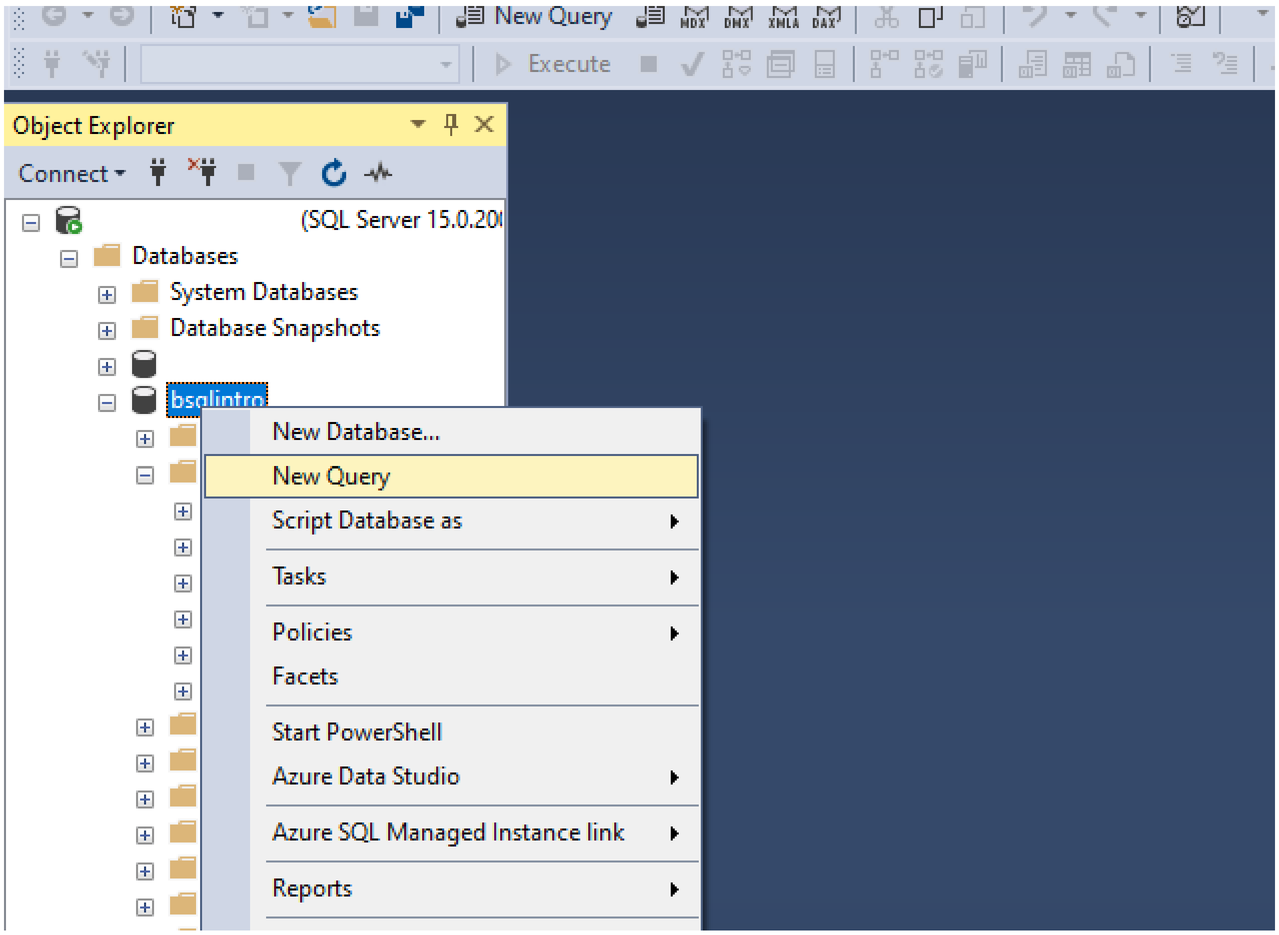
After connecting, we can view the databases in the server by opening the `Databases` folder.



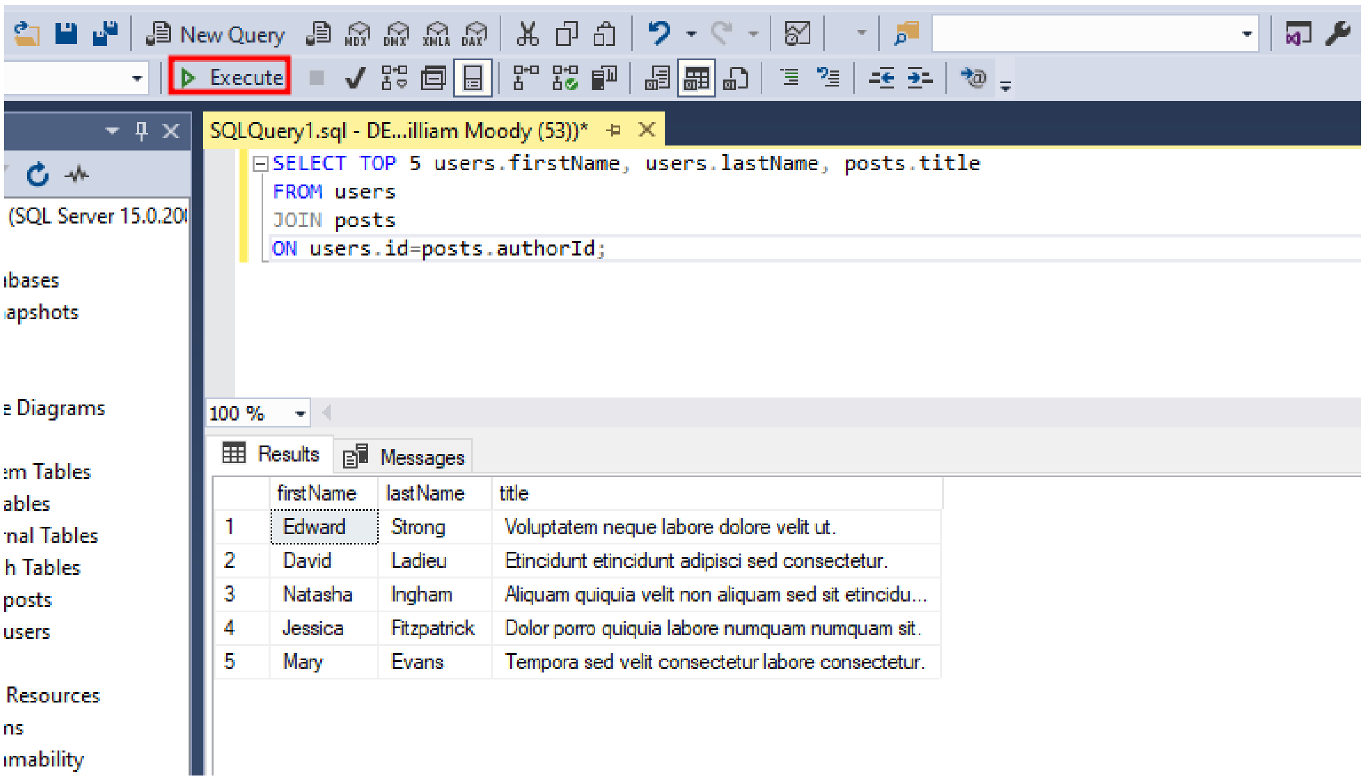
We can list the tables by opening the specific database, and then the `Tables` folder.



To run queries on a database we can right-click and select `New Query`.



We can enter queries into the new tab, and run by clicking `Execute`.



```sql
SELECT TOP 5 users.firstName, users.lastName, posts.title
FROM users
JOIN posts
ON users.id=posts.authorId;
```

	firstName	lastName	title
1	Edward	Strong	Voluptatem neque labore dolore velit ut.
2	David	Ladieu	Etincidunt etincidunt adipisci sed consectetur.
3	Natasha	Ingham	Aliquam quiquia velit non aliquam sed sit etincidu...
4	Jessica	Fitzpatrick	Dolor porro quiquia labore numquam numquam sit.
5	Mary	Evans	Tempora sed velit consectetur labore consectetur.

Introduction to Blind SQL Injection

Introduction

`Non-Blind SQL injection` is the typical "easy-to-exploit" SQL injection that you are likely familiar with. An example could be a vulnerable search feature that returns matching posts that you could exploit by injecting `UNION SELECT table_name,table_schema FROM information_schema.tables;--` to list all the tables in the database.

`Blind SQL injection` is a type of SQL injection where the attacker isn't returned the results of the relevant SQL query, and they must rely on differences in the page to infer the query results. An example of this could be a login form that does use our input in a database query but does not return the output to us.

The two categories of `Blind SQL Injection` are:

- `Boolean-based` a.k.a. `Content-based`, which is when the attacker looks for differences in the response (e.g. Response Length) to tell if the injected query returned `True` or `False`.
- `Time-based`, which is when the attacker injects `sleep` commands into the query with different durations, and then checks the response time to indicate if a query is evaluated as `True` or `False`.

`Blind SQLi` can occur when developers don't properly sanitize user input before including it in a query, just like any other SQL injection. One thing worth noting is that all `time-based` techniques can be used in `boolean-based` SQL injections, however, the opposite is not possible.

Example of Boolean-based SQLi

Here's an example of some `PHP` code that is vulnerable to a `boolean-based` SQL injection via the `email` POST parameter. Although the results of the SQL query are not returned, the server responds with either `Email found` or `Email not found` depending on if the query returned any rows or not. An attacker could abuse this to run arbitrary queries and check the response content to figure out if the query returned rows (`true`) or not (`false`).

Code: php

```php
<?php
...
$connectionInfo = Array("UID" => "db_user", "PWD" => "db_P@55w0rd#",
"Database" => "prod");
$conn = sqlsrv_connect("SQL05", $connectionInfo);
```

```
$sql = "SELECT * FROM accounts WHERE email = '" . $_POST['email'] . "'";
$stmt = sqlsrv_query($conn, $sql);
$row = sqlsrv_fetch_array($stmt, SQLSRV_FETCH_ASSOC);
if ($row === null) {
    echo "Email found";
} else {
    echo "Email not found";
}
...
?>
```

Conclusion

Up to this point, we've introduced `MSSQL` and the two types of `Blind SQL injection`. The best way to learn is to practice, so in the next two chapters we will cover custom examples of `boolean-based` and `time-based` SQL injections, and how to exploit them by writing custom scripts.