School of

Engineering and Computer Science

Te Kura Mātai Pūkaha, Pūrorohiko

CYBR 171 T1 2023

Ngā whakapūtanga o Te Haumaru rorohiko
Cybersecurity Fundamentals

Web security – part 1

Test #1

- This evening
- Will start at 5:30 pm, BUT please arrive 10-15 minutes before then
- Rooms:
  - MCLT101 (82)
  - MCLT103 (88)
  - KKLT303 (90)
  - KKLT301 (50)
- MCQ answer sheet

Learning objectives

- PART I: Web Applications and Related Attacks
  - The basics of Web applications
  - A simple authentication scheme
  - Eavesdropping attack
  - Steal the cookie
- PART II: Databases and Related Attacks
  - A crash course on SQL (Structured Query Language)
  - SQL injection attacks
- PART III: More Attacks and Countermeasures

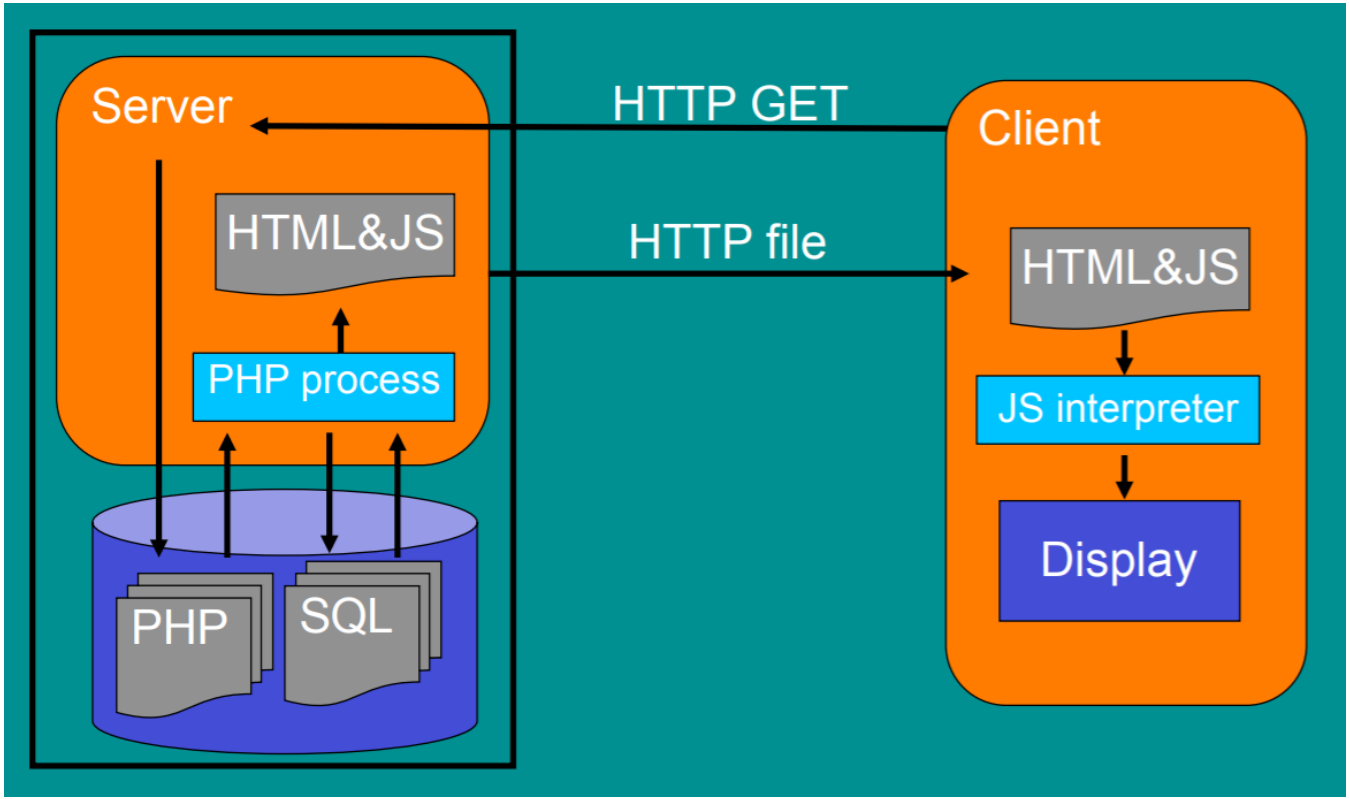
New Zealand Crimes Act

The New Zealand Crimes Act (available online at www.legislation.govt.nz) sections 248-254 document laws which criminalise certain acts involving computers. Some of the techniques shown could be used to break the law, it is your individual responsibility to ensure that you comply with the law.

Only hack something with *PERMISSION* or if *YOU owen it*!

PART I: WEB APPLICATIONS & RELATED ATTACKS

Web Application



Typical web setup

- HTML website

```
<form action="addResult.php" onsubmit="return validateForm(this);"
method="post" >
    <p>Username: <input type="text" name="user"></p>
    <p>Password: <input type="password" name="pass"></p>
    <p><input type="submit"></p>
</form>
```

Typical web setup (cont.)

- User's browser


```
POST addResult.php HTTP/1.1
Content-type: application/x-www-form-
urlencoded
Content-length: X

user=ian&pass=badpassword
```

Typical web setup

- PHP page reads and processes

```
$user = $_POST["user"];
$pass = $_POST["pass"];
$result = mysqli_query($con, "SELECT * FROM users
WHERE username='".$user."' && password='".$pass."';");
$row = mysqli_fetch_array($result);
if (empty($row)) {
    echo "Password and Username not found<br>\n";
  } else {
echo "Password correct!<br><br>";
}
```

Authenticating web users after login

- **IP address-based**
  - NAT or proxy may cause **several users to share the same IP**
  - DHCP (Dynamic Host Configuration Protocol) may cause the **same user to have different IPs**
- **Certificate-based (HTTPS/SSL)**
  - Who has a certificate and what is it, and who will sign it?
- **Cookie-based**
  - The most common

Simple authentication scheme

- Cookies are small data files that are stored on the computer.
- They are managed by the browser (demo)
- The Web Application:
  - **Verifies the credentials**, e.g., against database
  - **Generates a cookie** which is sent back to the user
  - **Set-Cookie**: **`auth=secret`**
- When the browser contacts the website again, it will include the session authenticator
  - Cookie: **`auth=secret`**

Eavesdropping attack

- If the connection is not encrypted, it is possible to eavesdrop, by
  - ISP,
  - anyone on the route
  - anyone on your local network, e.g. using the same wi-fi.
- Log-in should be done using SSL (https)
- This makes it impossible to eavesdrop on the password.
- After login, many websites used to **drop to http**
- Big security flaw

Steal the cookie

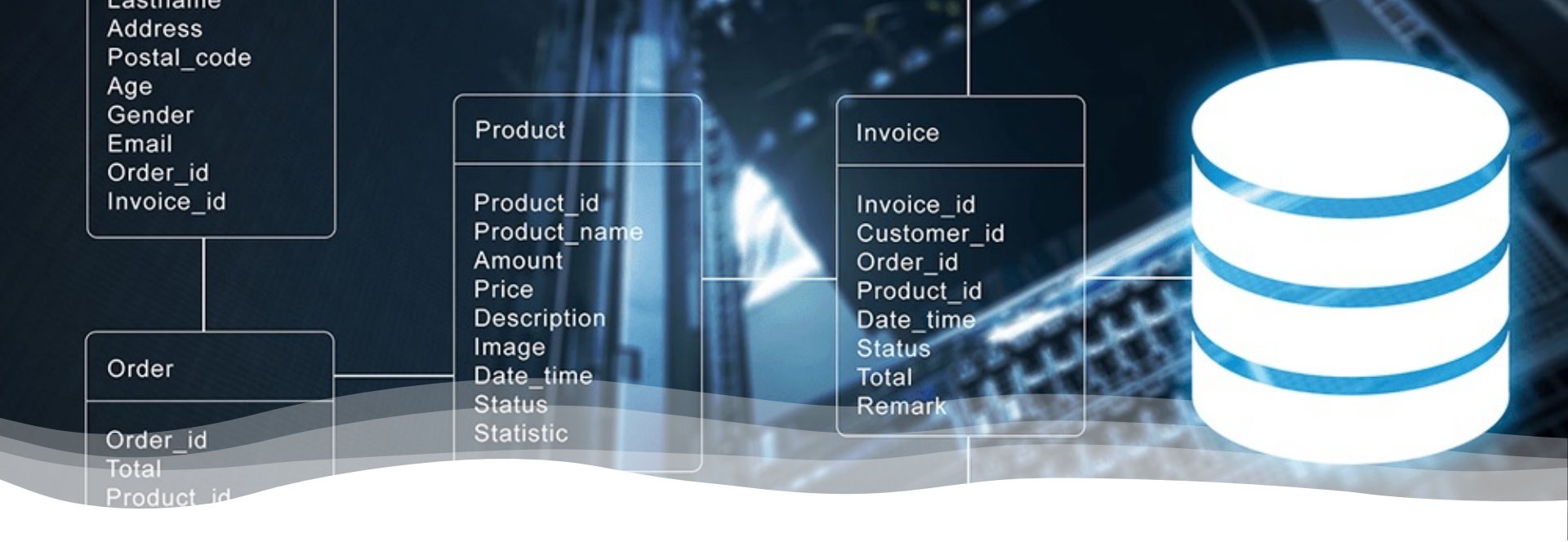
- So the attacker **doesn't** need the username and password
  - Just the **cookie**
- If the website uses **HTTPS** (SSL) it's safe
- But many websites used to drop back down to **HTTP** after a secure login.
  - This is a problem because ….
    - After login, the site asks for the **cookie** to see if still **authenticated**
    - The attacker **copies the cookie** and presents it to the site to pretend to be authenticated (as you!)

Countermeasures

- Use https (SSL) all the time.
  - https://www.eff.org/https-everywhere

- Set the **secure flag**, cookie is sent **only** over secure connections:

```
Cookie secureCookie = new Cookie("credential",c);
secureCookie.setSecure(true);
```



PART II: DATABASES AND RELATED ATTACKS

SQL

- **SQL** = **S**tructured **Q**uery **L**anguage
- Used to interact with the **database** behind the web application.
- Data is organised into databases that contain **tables**.
- Use a language to interact with it.
- We are using **mysqldb**.

```
mysql> use webdemo
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+------------------+
| Tables_in_webdemo |
+------------------+
| users            |
+------------------+
1 row in set (0.00 sec)

mysql> describe users;
+----------+------------+------+-----+---------+-------+
| Field    | Type       | Null | Key | Default | Extra |
+----------+------------+------+-----+---------+-------+
| username | varchar(20) | YES  |     | NULL    |       |
| password | varchar(20) | YES  |     | NULL    |       |
| count    | int(11)    | YES  |     | NULL    |       |
+----------+------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

SQL SELECT

SELECT *column_name* **FROM** *table_name* **WHERE** *column_name operator_value*;

```
mysql> select * from users;
+----------+--------------+-------+
| username | password     | count |
+----------+--------------+-------+
| ian      | badpassword  |    17 |
| harith   | goodpassword |     5 |
+----------+--------------+-------+
2 rows in set (0.00 sec)
```

SQL SELECT (cont.)

SQL statement to check a **username** and **password** are correct or not. Returns a row if it matches, otherwise **no rows**.

```
mysql> select * from users where username='ian' and password='badpassword';
+----------+-------------+-------+
| username | password    | count |
+----------+-------------+-------+
| ian      | badpassword |    17 |
+----------+-------------+-------+
1 row in set (0.00 sec)
```

```
mysql> select * from users where username='ian' and password='wrong';
Empty set (0.00 sec)
```

SQL Create, Destroy & COMMENTS

- Creates a new table

```
CREATE TABLE table_name {
      column1 datatype;
      column2 datatype;
      …
}
```

- Describe the table structure

```
DESCRIBE table_name;
```

- Deletes a table with all its data

```
DROP TABLE table_name;
```

```
mysql> create table persons (name varchar(20), address varchar(255));
Query OK, 0 rows affected (0.07 sec)

mysql> describe persons;
+---------+--------------+------+-----+---------+-------+
| Field   | Type         | Null | Key | Default | Extra |
+---------+--------------+------+-----+---------+-------+
| name    | varchar(20)  | YES  |     | NULL    |       |
| address | varchar(255) | YES  |     | NULL    |       |
+---------+--------------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql> drop table persons;
Query OK, 0 rows affected (0.02 sec)

mysql> describe persons;
ERROR 1146 (42S02): Table 'webdemo.persons' doesn't exist
mysql>
```

- Two minus signs (**--**) mean that **everything** afterwards is a **comment** and ***not executed***.

SQL Injection Attacks

http://www.shop.com/page?do=buy&product=17453

Web server might look up "17453" in a SQL database using:

...

SELECT * **FROM** *products* **WHERE** (*code='17453'*)

...

INSERT INTO *sales* **VALUES** (*id, customer, 17453*)

...

SQL Injection Attacks (cont.)

```
http://www.eshop.co.ukaction=buy&product=X
SELECT * FROM products WHERE (code='X');
```

- Case 1: What else could we use for X?

```
'); DROP TABLE products; --
```

```
SELECT * FROM products WHERE (code='');
DROP TABLE products; -- ');
```

- Case 2: Secret Item

```
'OR '1'='1'); --
```

```
SELECT * FROM products WHERE (code='' OR
'1'='1'); --');
```

  - **1 does equal 1!** Therefore, I will return all item's details

SQL Injection Attacks (cont.)

- Example 1

```
CREATE TABLE `users` (
`id` INT NOT NULL AUTO_INCREMENT,
`email` VARCHAR(45) NULL,
`password` VARCHAR(45) NULL,
PRIMARY KEY (`id`)
 );
INSERT INTO users (email, password) VALUES ('m@m.com',md5('abc'));
SELECT * FROM users WHERE email = 'admin@admin.sys' AND password =
 md5('1234');
SELECT * FROM users;
SELECT * FROM users WHERE email = 'xxx@xxx.xxx' OR 1=1 --;
```

- Example 2

```
xxx@xxx.xxx
xxx') OR 1=1 --]
```

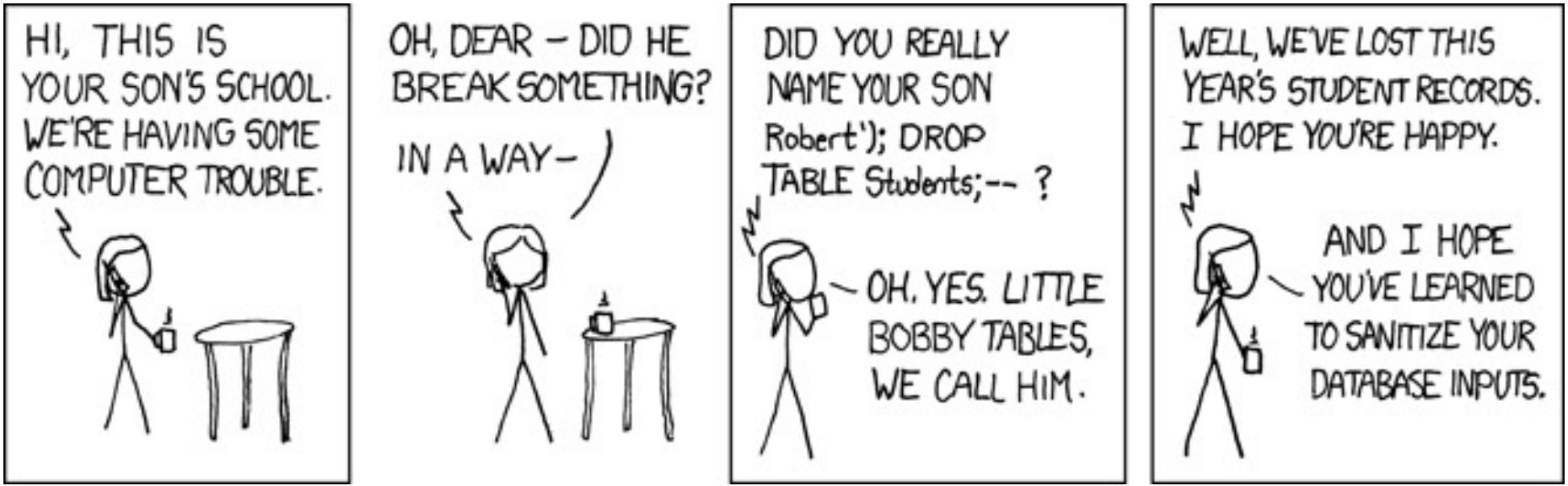
Source: https://www.guru99.com/learn-sql-injection-with-practical-example.html

Stopping SQL Injection Attacks

- You can stop SQL attacks by **checking the input**.
  - In PHP **`mysqli_real_escape_string`**!
  - Example: `'OR '1'='1' --` maps to `\'OR \'1\'=\'1\'--`

- All user-controlled data must be "**sanitised**" (**this can be hard**).
  - **`EVIL STRING`**
  - **`EBC<backspace><backspace>VIL STRING`**

Exploits of a Mom



https://xkcd.com/327/

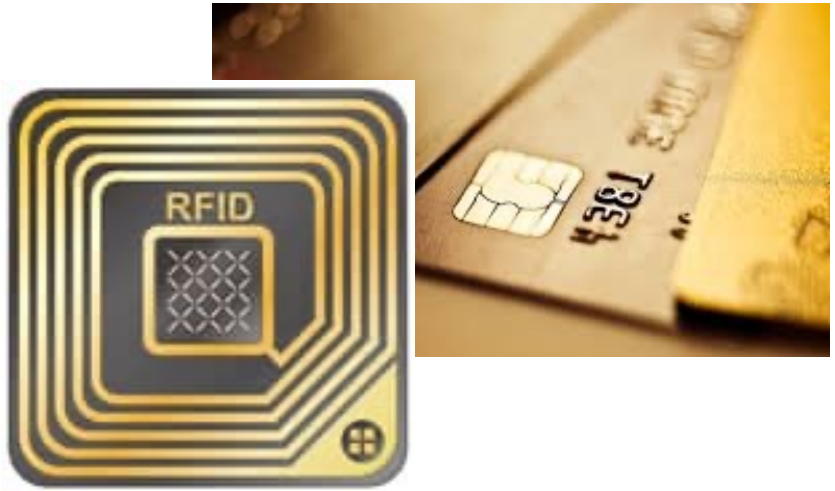
Not just websites

QR Code Generator: https://www.the-qrcode-generator.com/

"; DROP TABLE ITEM; --

Any source of data can be used for an SQL attack

- Barcodes
- RFID cards
- Election voting papers
- Number plates



Example 1

Did Little Bobby Tables migrate to Sweden?

Posted by Jonas Elfström on 24/09/2010

As you may have heard, we've had a very close election here in Sweden. Today the Swedish Election Authority published the hand written votes. While scanning through them I happened to notice

```
R;13;Hallands län;80;Halmstad;01;Halmstads västra
valkrets;0904;Söndrum 4;pwn DROP TABLE VALJ;1
```

The second to last field[1] is the actual text on the ballot[2]. Could it be that Little Bobby Tables is all grown up and has migrated to Sweden? Well, it's probably just a joke but even so it brings questions since an SQL-injection on election data would be very serious.

https://alicebobandmallory.com/articles/2010/09/23/did-little-bobby-tables-migrate-to-sweden

Example 2

https://hackaday.com/2014/04/04/sql-injection-fools-speed-traps-and-clears-your-record/

PART III: MORE ATTACKS AND COUNTERMEASURES

URL Hacking

- The user can type **anything** they want into the URL bar, or even form the request by hand.
  `http://`**nameOfHost**`/`**filePath**

- An attacker can try to **guess** filenames,
  - Guessable directory names will be found.

URL Hacking (cont.)

- Web owners often include **`robots.txt`**

  User-agent: [user-agent name]
  Disallow: [URL string not to be crawled]

  For example:

  User-agent: *
  Disallow: /images

URL Hacking (cont.)

- Bad idea, use this protect web files that contain passwords or sensitive content

  For example:

  User-agent: *
  Disallow: /assets/

Hmmmm let’s look at http://site.com/assets/

Path Traversal

- The user can type anything they want into the URL bar, or even form the request by hand.

  `http://nameOfHost/../../../etc/shadow`

- If the webserver is running **with root permission**, this will give me the password file.

Path Traversal: Fix

- Use **access control** settings to stop Path Traversal.
- The best practice, make a **specific user account** for the webserver.
- Only give that account access to **public files**.

Missing Function Level: Access Control

- Query strings are used to tell dynamic webpages what to do

```
http://myWebShop.com/index.php?account=tpc&action=add
http://myWebShop.com/index.php?account=tpc&action=show
```

- What if the attacker tries:

```
http://myWebShop.com/index.php?account=admin&action=delete
```

Fix

- **No security through obscurity**
- Never rely on just the URL request for authentication.
- E.g. use **cookies** to control access.

Security Through Obscurity: The Good, The Bad, The Ugly

Summary

- Anatomy of a dynamic web site
- Eavesdropping
- Stealing a cookie
- SQL injection
- File traversal