L6 - Input sanitization

There are two main vulnerabilities that can be exploited when sending data to a server:

- database queries
- file uploading

File uploading

To upload files to a server we can use simple HTML forms, or we can directly use HTTP clients like Postman. Either way, the files will be sent as encoded form parameters. If using HTML, the form must have an *multipart/form-data* encoding.

```
<form action="https://fmprj.ciroue.com" method="POST" enctype="multipart/form-data">
  <input type="file" name="filename">
  <input type="submit">
</form>
```

On the backend side, the request must be captured, then stored to a specific location. Alongside the binary content of the file, additional information is usually send by the HTTP client, like:

- name
- full path
- mime type
- size
- errors
- temporary location and name

As workflow, usually, the webserver will upload the file in a temporary location, then we can move the file to a designated final location on disk. With PHP we can access file information by using the *$_FILES* global variable, and can move the files from temp folders by using the *move_uploaded_file* function.

```
// $_FILES['filename'] - access filename input

//print file information
print_r($_FILES['filename']);

//move file to a designated location
move_uploaded_file($_FILES['filename']['tmp_name'], $uploadPath);
```

File upload can be exploited and can result in complete takeover of the server, if a malicious entity is able to upload a runnable script on the server (e.g. if using a php web server, that would be a php file). This could allow a third party to list, access and read all the contents of the files, and data on that environment (this usually includes database connection details, root passwords, logs, database data etc.)

In order to secure file uploading we can take two simple measures:

- allow only specific file types and extensions: you can extract the file extension from the full filename, and also check the mime type of the file

```
//extract extension
$ext = strtolower(end(explode('.',$_FILES['filename']['name'])));

//restrict extension
$allowedExtensions = ['jpeg', 'png', 'pdf'];
if(in_array($ext,$allowedExtensions)) {
    ... // upload file only if in allowed types
}
```

- randomize filenames in order to avoid automatic uploading and accessing by bots

```
//random filename, then append extracted extension
$filename = substr(str_shuffle(MD5(microtime())), 0, 10).'.'.$ext;
```

Database injection

SQL Injection is one the most known vulnerability. Still, there are systems that are yet not fully protected. SQL injections work by tricking the SQL processor into enabling additional query conditions or even additional queries and is usually related to un-sanitized input parameters. Let's consider the following query string, that is using a parameter to select user data based on an username or id:

```
SELECT * FROM users where id='$_REQUEST["id"]'
```

This will easily allow an attacker to trick the query processor into returning additional data or running additional queries. Let's take several examples:

- Using un-escaped quotes can alter the basic query and return all the entries in the *users* table

```
//1
$_REQUEST['id'] = 19' OR 1=1 OR ''='
```

- Using query delimiters or escape characters could allow running additional queries

```
//2
$_REQUEST['id'] = '; DELETE FROM users WHERE 1=1 OR ''='
```