**COMP 3015, BCIT Computing**

**Final Exam Part 1**

Marked out of 50 points

**GOOD LUCK!**

**Monika Szucs**

1. (6 points) Describe, in detail, how a server side web application using sessions can determine which session (and associated data) belongs to which client. How does the server determine this on each request? Ensure you include details on any particular HTTP request/response headers that are commonly used in this process.

In a server-side web application utilizing sessions, the server ensures each client's session (and its associated data) is managed effectively through a defined process. Upon a user accessing the web application, the server generates a unique session identifier (session ID) for that client. This session ID, such as abcdef1234567890 in the example HTTP response header Set-Cookie: session\_id=abcdef1234567890; Path=/; Secure; HttpOnly; SameSite=Strict;, is stored within a cookie on the client's browser.

Subsequently, on every request, the client's browser automatically sends this session ID back to the server via the Cookie header in the HTTP request. Using this session ID, the server identifies and retrieves the corresponding session data stored in its session storage. This mechanism enables the server to associate each request with the correct client session, allowing personalized and secure interactions within the stateless HTTP protocol framework. By leveraging cookies and managing session IDs through HTTP headers, the server maintains continuity in user sessions, ensuring seamless and secure user experiences across the web application.

1. (5 points) Describe how passwords should be stored. What is the name of a cryptographic function that can provide this functionality, and what do the “cost” and “salt” do?

To securely store passwords, several best practices should be followed. Firstly, passwords should not be stored in plain text; instead, they should be hashed using a cryptographic hash function before being stored in the database. A cryptographic hash function takes an input (in this case, the password) and generates a fixed-length string of characters that is unique to that input. Importantly, this process is irreversible, meaning it should not be possible to determine the original password from its hash value.

One commonly used cryptographic function for this purpose is bcrypt. Bcrypt allows for the hashing of passwords with added security features such as a "cost" parameter and "salt". The "cost" parameter determines how computationally intensive the hashing process is, thereby increasing the difficulty for potential attackers attempting to reverse-engineer hashed passwords. A higher cost value means more computational effort is required, thus enhancing security against brute-force attacks.

Additionally, "salt" is a random value unique to each password instance. It is appended to the password before hashing, ensuring that even if two users have the same password, their hashed values will be different. This prevents attackers from efficiently using precomputed tables (rainbow tables) to crack passwords, as each hash must be cracked individually.

In summary, storing passwords securely involves hashing them using a cryptographic function like bcrypt, setting an appropriate cost parameter to control computational effort, and using a unique salt for each password to mitigate against common attack methods. These practices collectively enhance the security of stored passwords and protect user accounts from unauthorized access.

Password hash:

$password = 'user\_password';

$options = [

'cost' => 12, // Adjust this to make it harder to crack

];

$hashedPassword = password\_hash($password, PASSWORD\_BCRYPT, $options);

1. (4 points) Describe the cause of SQL injection vulnerabilities. Include a small pseudocode example (one or two lines) of what a vulnerability in code might look like.

SQL injection vulnerabilities happen when user inputs are directly added into SQL queries without proper checks. This lets attackers manipulate inputs to change the intended behavior of the SQL query. Example:

When a web application needs to check if a person's login credentials are valid. The application might construct a SQL query like this:

// Example vulnerable pseudocode

person = getInput(‘person’)

password = getInput('password')

// Vulnerable SQL query construction

query = "SELECT \* FROM users WHERE person = '" + person + "' AND password = '" + password + "'"

This is the vulnerable pseudocode:

SELECT \* FROM users WHERE person = 'admin' OR 1=1 --' AND password = '$password'

This input modifies the query to select all users where the person is 'admin' or where 1=1, which is always true. This allows the attacker to potentially access unauthorized data or bypass login checks altogether.

1. (5 points) How do parameterized prepared queries prevent SQL injection vulnerabilities?

Parameterized prepared queries prevent SQL injection vulnerabilities by utilizing placeholders (such as ? or named placeholders like :name) in the SQL query. Instead of directly inserting user input into the query string, which could be manipulated by attackers to alter query behavior, placeholders are used. These placeholders are then bound to the actual input values separately, ensuring that the database interprets them strictly as data and not as executable SQL code. This robust technique effectively mitigates the risk of SQL injection attacks, as even if an attacker attempts to inject malicious SQL code, it will be treated purely as data and not as executable commands.

$username = $\_POST['username'];

$password = $\_POST['password'];

$grab = "SELECT \* FROM clients WHERE username = ? AND password = ?";

$statement = $pdo->prepare($grab);

$statement->execute([$username, $password]);

1. (4 points) Describe the difference between an HTTP POST request and an HTTP GET request. Give an example of a feature where it would be bad to use an HTTP GET request, and explain specifically why it would be bad.

HTTP POST:

Sends data to the server to create/update a resource.

Transmits data in the body of the request.

Data is not visible in the URL.

Requests are not cached.

More secure because sensitive data is not exposed in the URL.

HTTP GET:

Requests data from a server.

Data is visible in the URL.

Requests can be cached by browsers and proxies.

Less secure because sensitive information may be exposed.

Example of when it would be bad to use an HTTP GET request:

It would be problematic to use an HTTP GET request in a signup form where users input personal information like their name, email, and password. This is because all the data submitted via GET requests appears in the URL, making it visible to anyone who can view the browser's history or network traffic. This exposure could lead to unauthorized access to sensitive information, compromising user privacy and security.

1. (4 points) Describe how autoloading works in the context of PHP application development.

* What are the benefits of autoloading over explicitly requiring PHP class files?
* When does the callback to **spl\_autoload\_register** get invoked?

Autoloading in PHP Application Development:

How Autoloading Works:

- Autoloading acts like a personal assistant for your PHP class files.

- Instead of manually including each file, autoloading automatically loads the necessary class files when they are referenced in your code.

Benefits of Autoloading:

Less Hassle:

No need to clutter your code with include or require statements everywhere.

Files are included automatically as needed, keeping your code cleaner.

Improved Performance:

Autoloading loads classes only when they are required, which can enhance PHP performance by reducing unnecessary file loads.

Simplified Maintenance:

Easier updates when adding or removing classes.

Organizing classes using namespaces becomes straightforward, as autoloading manages file locations seamlessly.

Invocation of spl\_autoload\_register Callback:

When It's Invoked:

spl\_autoload\_register is called early in your script's execution.

It registers a function (or multiple functions) to autoload classes, ensuring PHP knows where to find the necessary class files without manual intervention.

In essence, autoloading streamlines PHP development by automating the process of class file inclusion, leading to cleaner code, improved performance, and easier maintenance of your application.

1. (4 points) What data is encrypted in transit when using HTTPS? Are cookies and query/route parameters encrypted in transit?

**Data Encrypted by HTTPS:**

1. **Request and Response Body**:
   * HTTPS encrypts the entire request and response body which contains any form data, JSON payloads. or other data between the client and server.
2. **URL Path and Query Parameters**:
   * The path of the URL and any query parameters are also encrypted. For example, if a URL is https://example.com/search?q=keyword, both /search and q=keyword are encrypted.
3. **Cookies**:
   * Cookies sent over HTTPS are encrypted. Such as session cookies and other cookies for user sessions or storing information.
4. **Headers**:
   * HTTP headers are encrypted when transmitted over HTTPS.

In summary, HTTPS ensures that all sensitive data exchanged between the client and server—whether it's the request and response content, URL components, cookies, or headers—is encrypted to protect against interception and tampering.

Both cookies and query/route parameters are encrypted in transit when using HTTPS. This encryption ensures that sensitive information contained in cookies and parameters is protected from unauthorized access during transmission over the network.

1. (5 points) Describe the steps needed to use key based authentication with a service such as GitHub. What do you need to generate, and what do you need to upload to the service?

For service such as github we need to:

* 1. Generate SSH key by using ssh-keygen

ssh-keygen -t ed25519 -C “[email@test.com](mailto:email@test.com)”

* 1. We need to upload the public key to Github

cat /root/.ssh/id\_ed25519.pub

* 1. We need to go to our Github account settings and go to SSH keys when we create a new SSH year we past the key in the key field and give it a title.

then we can do

we can clone the repository on our desktop using git clone <repo link>

git add .

git commit -m “message”

git push

1. (5 points) Why shouldn’t the TLS protocol be implemented to send public keys across an unsecured communications network (as a first step to establishing encrypted communications)?

What is the solution to this problem?

Implementing the TLS protocol to send public keys over an unsecured communication network as a first step to establishing encrypted communications poses risks. One significant concern is the potential for man-in-the-middle attacks, where an attacker intercepts the transmission and substitutes the legitimate public key with their own. This substitution enables the attacker to decrypt messages intended for the legitimate recipient.

Solution to the Problem:

* + - 1. Use of Digital Certificates:
* Digital certificates include not only the public key but also additional information, providing a means of authentication and ensuring the integrity of transmitted data.
* Clients can verify the authenticity of certificates using trusted root certificates stored on their systems.
  + - 1. Involvement of Certificate Authorities (CAs):
* Trusted third-party Certificate Authorities issue digital certificates that bind public keys to specific entities, such as websites.
* These certificates are digitally signed by the CA, establishing their authenticity and ensuring that clients can trust the public keys associated with them.

By leveraging digital certificates and Certificate Authorities, organizations can establish secure communication channels over unsecured networks. This approach mitigates the risks of man-in-the-middle attacks and safeguards the integrity and confidentiality of transmitted data.

1. (8 points) Describe some issues with the following code. Note that the name of the file below is **register.php**. There are multiple security vulnerabilities and various bad practices.

This question assumes that a database exists named “db\_name\_here” with no password, and that a users table with “username” and “password” columns are on the table.

register.php:



Describe the issues below in as much detail as you can.

* Changed method="GET" to method="POST" in the HTML form to securely send the sensitive data of information. We probably don’t want to pass over the username in the url just to protect the users information.
* Updated form inputs to use type="password" for the password field to obscure user input.
* Encouraged proper validation and hashing of passwords using password\_hash() which consists of the password, PASSWORD\_BCRYPT, and the cost options. We should not use md5 we should be using bycrypt or another modern algorithm
* Emphasized the need for using prepared statements prevent SQL injection attacks. User inputs ($\_GET['username'] and $\_GET['password']) have no sanitization or validation.
* There is improper redirection. After inserting the user into the database, the code redirects using header("Location: index.php?name=$username"). This redirection can be changed by an attacker (if they control $\_GET['username']) to perform a redirect attack
* Issue with connecting to the database. The database connection credentials ("localhost", "root", "", "db\_name\_here", 3306) are hard-coded in the PHP file. This exposes sensitive database information and can lead to unauthorized access if the file is accessed directly by anyone.