SQL Injection and Cookies

Question 1  Boogle

Boogle is a social networking website that’s looking into expanding into other domains. Namely, they recently started a map service to try their hand at fusing that with social media. The URL for the main website is https://www.boogle.com, and they want to host the map service at https://maps.boogle.com.

(a) For each of the following webpages, determine whether the webpage has the same origin as http://boogle.com/index.html.

i. https://boogle.com/index.html
ii. http://maps.boogle.com

(b) Describe how to make a cookie that will be sent to only Boogle’s map website and its subdomains.

(c) How can Boogle ensure that cookies are only transmitted encrypted so eavesdroppers on the network can’t trivially learn the contents of the cookies?

(d) Boogle wants to be able to host websites for users on their servers. They decide to host each user’s website at https://[username].boogle.com. Why might this not be a good idea?
(e) Propose an alternate scheme so that Boogle can still host other users websites with less risk, and explain why this scheme is better.

Note: It is okay if the user sites interfere with each other, as long as they cannot affect official Boogle websites.
Question 2  Second-order linear... err I mean SQL injection

Alice likes to use a startup, NotAmazon, to do her online shopping. Whenever she adds an item to her cart, a POST request containing the field item is made. On receiving such a request, NotAmazon executes the following statement:

```go
cart_add := fmt.Sprintf("INSERT INTO cart (session, item) " +
    "VALUES ('%s', '%s')", sessionToken, item)
db.Exec(cart_add)
```

Each item in the cart is stored as a separate row in the cart table.

(a) Alice is in desperate need of some pancake mix, but the website blocks her from adding more than 72 bags to her cart. Describe a POST request she can make to cause the cart_add statement to add 100 bags of pancake mix to her cart.

When a user visits their cart, NotAmazon populates the webpage with links to the items. If a user only has one item in their cart, NotAmazon optimizes the query (avoiding joins) by doing the following:

```go
cart_query := fmt.Sprintf("SELECT item FROM cart " +
    "WHERE session='%s' LIMIT 1", sessionToken)
item := db.Query(cart_query)
link_query = fmt.Sprintf("SELECT link FROM items WHERE item='%s'", item)
db.Query(link_query)
```

After part(a), Alice recognizes a great business opportunity and begins reselling all of NotAmazon's pancake mix at inflated prices. In a panic, NotAmazon fixes the vulnerability by parameterizing the cart_add statement.

(b) Alice claims that parameterizing the cart_add statement won't stop her pancake mix trafficking empire. Describe how she can still add 100 bags of pancake mix to her cart. Assume that NotAmazon checks that sessionToken is valid before executing any queries involving it.
Question 3  Session Fixation

A session cookie is used by most websites in order to manage user logins. When the user logs in, the server sends a randomly-generated session cookie to the user’s browser. The server also stores the cookie value in a database along with the corresponding username. The user’s browser sends the session cookie to the server whenever the user loads any page on the site. The server then looks the session cookie up in the database and retrieves the corresponding username. Using this, the server can know which user is logged in.

Some web application frameworks allow cookies to be set by the URL. For example, visiting the URL

\[\text{http://foobar.edu/page.html?sessionid=42}\]

will result in the server setting the sessionid cookie to the value “42”.

(a) Can you spot an attack on this scheme?

(b) Suppose the problem you spotted has been fixed as follows: foobar.edu now establishes new sessions with session IDs based on a hash of the tuple (username, time of connection). Is this secure? If not, what would be a better approach?