MAKERERE UNIVERSITY

COLLEGE OF COMPUTING AND INFORMATION SCIENCES

DEPARTMENT OF COMPUTER SCIENCE

COURSE UNIT: **BSE 3105** ADVANCED PROGRAMMING

LECTURER: Mr. LWOMWA JOSEPH

ASSIGNMENT: TEST 2

NAME	**REGISTRATION NUMBER**	**STUDENT NUMBER**
ABILA Raphael	16/U/2673/PS	216006923

QUESTION ONE

a)1. Modules of the system back-end and front-end had been incorrectly connected, validated and sanitized there by bringing a possibility of cross-site scripting attack.

Protection Against Cross-Site Scripting (XSS) Attacks –Any injections of active JavaScript content into your Webpages must not be allowed by programmers so as to ensure website security.

2. The time the system users take as they are trying to send and receive chat messages is too long. Because of the long-waited messages, a possibility of SQL injections may arise since there is real-time client-server exchange of messages.

Protection against SQL injection attacks – Parameterized queries must always be used by programmers and they should avoid standard Transact SQL as this would allow hackers to insert rogue code.

b) Attack for Cross-site scripting: Malicious JavaScript can be injected by hackers into the pages, and change the content, and when users access the web pages, their credentials and login cookie details would get stolen thus exploiting integrity.

Attack for SQL injection: Attacks which try to exploit an underlying SQL database can use faulty input sanitization to their advantage for example a malicious user can send ("?person = 1 or 1= --1") . It will cause a syntax error when the database interprets the SQL, if the backend does not sanitize this input, there are chances that the error output will be pushed to the front end and the hacker will now know that they can construct more sophisticated inputs to add clauses to the SQL query that may dump data from your database.

c) The product team will systematically identify assets, vulnerabilities, threats and rate the threats that are most likely to affect your system which is termed as asset management through threat modeling.

It will enable you Understand the application better: Team members specially the technical guys who make policies need to spend time analyzing the application in a structured manner and they end up inevitably with a deeper understanding of how the application works.

Threat modeling helps to find bugs: The additional scrutiny of the threat modeling process leads to the discovery of other, non-security related bugs.

It helps in Identifying complex design bugs: The analytical nature of the process can reveal complex multi-step security bugs where several small failures combine to cause a major failure.

This kind of vulnerability may be missed by unit testing performed by the developer and the majority of test plans

Threat modeling enables the product team to drive your security test plan design. Testers should test against the threat model, which will help them develop new test tools and procedures.

New members will be integrated: Threat modeling is a useful tool for new team members to become familiar with the architecture of the product.

Threat modeling allows you to systematically identify assets, vulnerabilities, threats and rate the threats that are most likely to affect your system.

Threat modeling also helps provide a strict guideline for patch and update management

It can further assist set up policies in regards to User access management

d)1: **Identifying assets**: This will involve both concrete and abstract, that could be targets of an attack which calls for building a list of all assets that require protection in the chat application, including: Confidential data, configuration files such as customer databases, Web pages, System availability, users, Host.

2: **Creating an architectural Overview:** At this stage the function of the application is documented, including architecture and physical deployment configuration. Identification of what the application does essentially involves understanding of the business rules of the application and the data flow to and from the various assets

a. Identifying what the application does: Document use cases to help the product team and others understand how the chat application is supposed to be used

b. Creating an application architecture diagram: This describes the composition and structure of your application and its subsystems as well as its physical deployment characteristics.

c. Identifying the technologies that would be used: This would help put focus on technology-specific threats later in the process.

3: **Decompose the application:** In this step in Microsoft's threat modeling process an application is broken down in order to create a security profile for the application that is based on traditional vulnerability areas. The steps for application decomposition include identification of trust boundaries, data flow, entry points, privileged code and security profile documentation.

4: **Identify threats:** At this point of threat modeling process we have identified that application assets, created an architectural overview and decomposed the application. It is now time to identify the threats of the entire chat application. For identifying those threats STRIDE or Attack trees would be used with the corresponding countermeasures that can be used to mitigate the risks.

5: **Document the threats:** The next step in threat modeling process is to document all the threats that were identified. Documentation should have been performed throughout all of the previous stage of the process. In order to effectively document the identified threats, a template should be created that encapsulates various relevant threat attributes, such as target, risk, attack techniques and countermeasures

The following illustrates how the identified vulnerabilities can be documented

Threat Name

Threat Description	Injection of SQL Commands
Threat target	Data Access Component
Risk	
Attack techniques	Attacker appends SQL commands to user name, which is used to form a SQL query
Countermeasures	Use a regular expression to validate the user name, and use a stored procedure with parameters to access the database

6: **Rate the threats**

Rate all of the threats that were identified. That is done by evaluating potential impact of each of the threats on the system, the purpose of this exercise is to help priotize the threats. It may be unrealistic to expect that under the pressures of a typical software development schedule all of the threats will be mitigated. The formula for calculating the risk is
RISK=PROBABILITY*DAMAGE POTENTIAL using D (Damage potential) R(Reproducibility) E(Exploitability) A (Affected users) D(Discoverability) to rate the threats

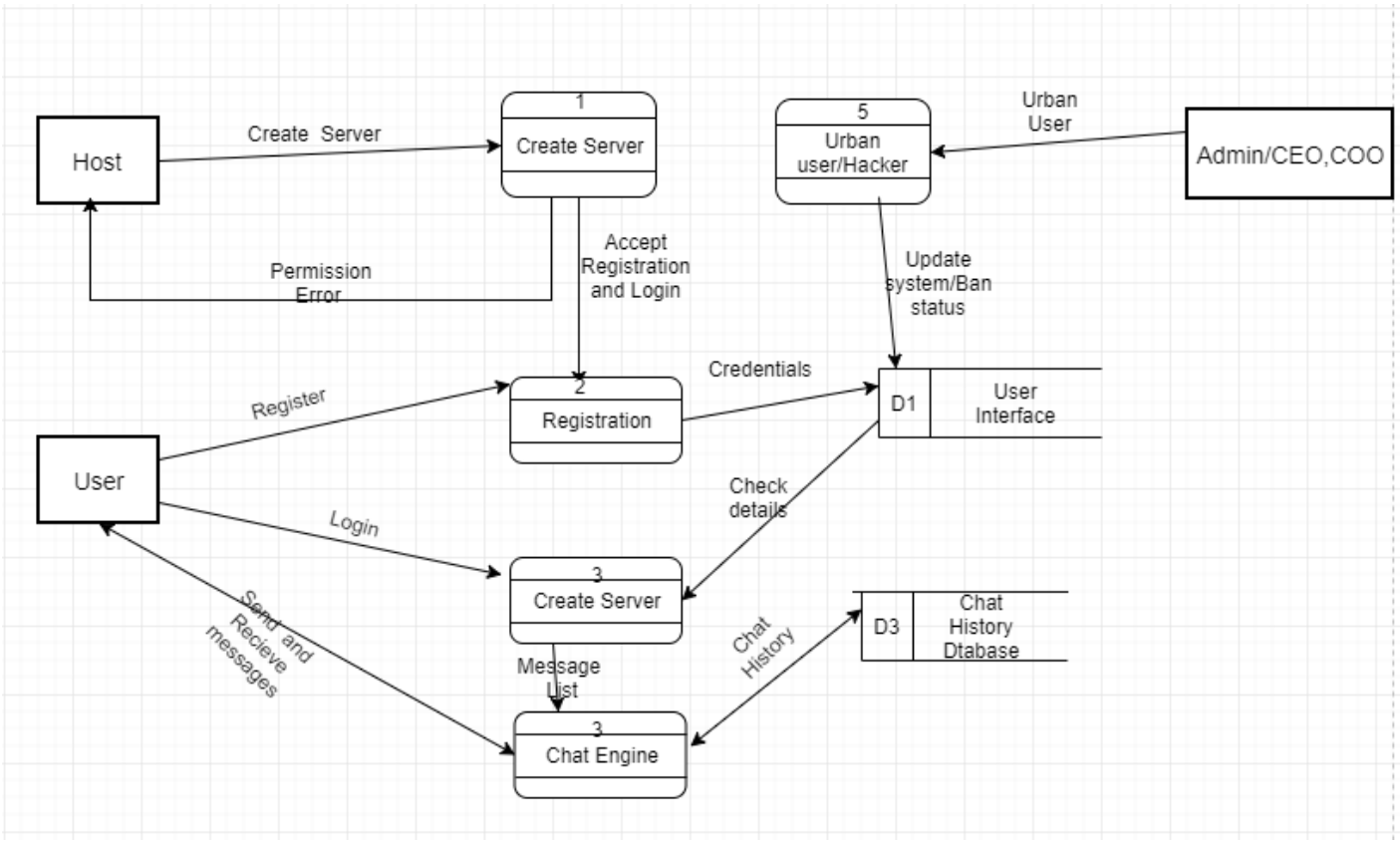
e) The approaches that can be used in step 3 include;

DFD (Data Flow diagram), UML (Unified Modeling Language)

The following is the DFD Diagram showing decomposition of chat application's system.

Actors (User, Admin/CEO, COO, Hacker/Urban user)

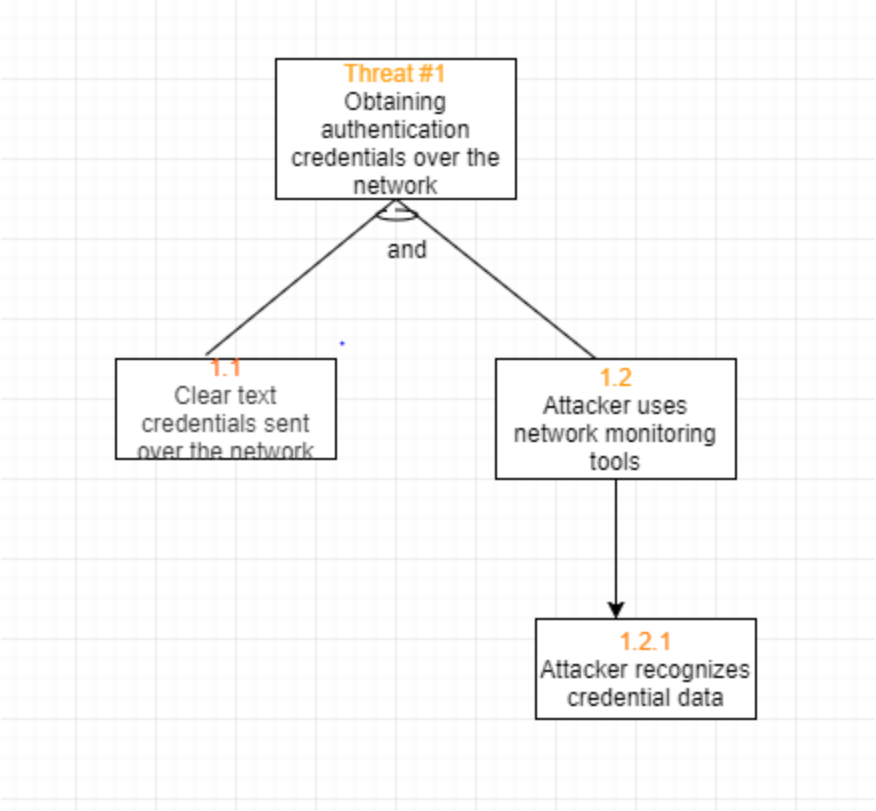
Level 1 DFD



For step 4 , The approaches that can be used for identifying the threats include;

Attack trees, STRIDE approach

According to the case scenario given, I would identify threats using an attack tree illustrated as follows;

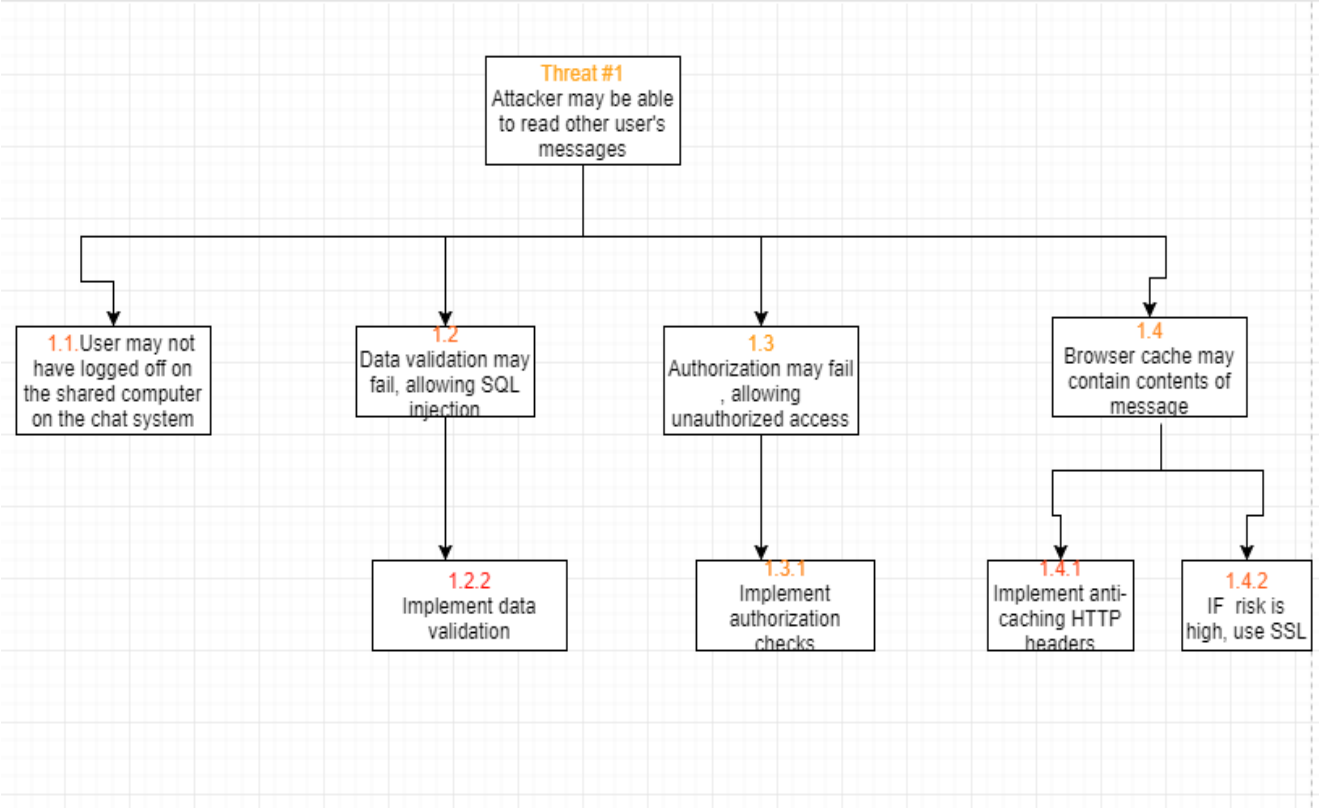


The outline representation of the attack tree 1

Threat#1 Attacker obtain authentication credentials by monitoring the network

1.1 Clear text credentials sent over the network
AND
1.2 Attacker uses network monitoring tools
1.2.1 Attacker recognizes credential data

Attack tree 2 using SQL injection



From step 6

The approach that can be used to rate the threats is DREAD where each threat that was identified is evaluated basing on its risk which is obtained from
Risk=Impact(R,E,D)*Probability(D,A)

The threat rating table would be demonstrated as follows Using DREAD

Rating table

	Rating	High (3)	Medium (2)	Low (1)
D	Damage Potential	The attacker can subvert the security system; get full trust authorization; upload content	Leaking sensitive information	Leaking trivial information
R	Reproducibility	The attack can be reproduced every time and does not	The attack can be reproduced, but only with a timing window and a	The attack is very difficult to reproduce, even with knowledge of the security hole

		require a timing window	particular race situation	
E	Exploitability	A novice programmer could make the attack in short time	A skilled programmer could make the attack, then repeat the steps	The attack requires an extremely skilled person and in-depth knowledge every time to exploit
A	Affected Users	All users, default configuration	Some users, non-default configuration	Very small percentage of users, obscure feature; affects anonymous users
D	Discoverability	Published information explains the attack. The vulnerability is found in the most commonly used feature and is very noticeable	The vulnerability is in a seldom-used part of the product, and only a few users should come across it. It would take some thinking to see malicious use	The bug is obscure, and it is unlikely that uses will work out damage potential

Using values (1-3) should be counted for each threat. The result can fall in the range (5-15) then values (12-15) are treated as high risk, 8-11 as medium and 5-7 as low.

Using the vulnerabilities used in step 5;

Threat	D	R	E	A	D	Total	Rating
Attacker obtains authentication credentials by monitoring the network	3	3	2	2	2	12	High
SQL commands injected into application	3	3	3	3	2	14	High

Applying DREAD again to complete the template documenting from the previous step

Threat Description	Attacker obtains authentication credentials by monitoring the network
Threat target	Chat application user authentication process
Risk rating	High
Attack techniques	Use of network monitoring software
Countermeasures	Use SSL to provide encrypted channel

f) Application decomposition procedure in threat modeling has been used to identify the different components of the system thus enabling us to implement security on independent components (Securing the weakest link).

During request authentication and data validation, independent fail secure mechanisms, like exception handling, Regular expressions have been used to ensure proper authenticity of the API's and data inputs. Assuming that all the secrets (user passwords) are not safe, data sanitization mechanisms can be used like data masking, substitution, nulling off.

Privacy in the system should also be priotized so as to keep the user's credentials safe. This can be achieved by authenticating each login, user profile requests and also clearing the session cookies and the cached data such that it is not visible to malicious people.

For proper identification of the threats on each and every component, the architecture of the system should be kept simple for easy management.

Access Control Lists (ACL) have been used to ensure proper authentication and also make sure that the users access the proper data only available for their category. control lists have been used to identify the minimum privileges that can be granted to users of minimal authority. Granting minimum privileges helps in the way that when code is limited in the system-wide actions it may perform, vulnerabilities in one application cannot be used to exploit the rest of the system. This can also achieve by usage of just enough security measures that are not too complex so as not to give users reasons to find the security of the system hectic and devise means of circumventing around them or avoid them.

g) Cross-site Scripting (XSS) refers to client-side code injection attack wherein an attacker can execute malicious scripts (also commonly referred to as a malicious payload) into a legitimate website or web application. XSS is amongst the most rampant of web application vulnerabilities and occurs when a web application makes use of unvalidated or uencoded user input within the output it generates.

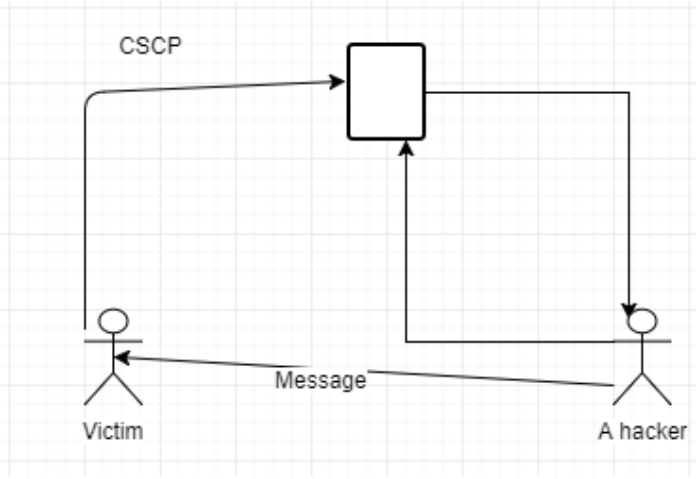
Through leveraging XSS, instead of an attacker targeting a victim directly, he exploits a vulnerability within a website or web application that the victim would visit, essentially using the vulnerable website as a vehicle to deliver a malicious script to the victim's browser.

How Cross-site Scripting works

In order to run malicious JavaScript code in a victim's browser, an attacker must first find a way to inject a payload into a web page that the victim visits. Of course, an attacker could use social engineering techniques to convince a user to visit a vulnerable page with an injected JavaScript payload.

In order for an XSS attack to take place the vulnerable website needs to directly include user input in its pages. An attacker can then insert a string that will be used within the web page and treated as code by the victim's browser.

Cross –site scripting can be illustrated using the diagram below



Message

User id : Raph

Pw:password

Sorry Raph

Invalid login<script>

2. **SQL Injection (SQLi**) refers to an injection attack wherein an attacker can execute malicious SQL statements (also commonly referred to as a malicious payload) that control a web application's database server. Since an SQL Injection vulnerability could possibly affect any website or web application that makes use of an SQL-based database, the vulnerability is one of the oldest, most prevalent and most dangerous of web application vulnerabilities.

By leveraging an SQL Injection vulnerability, given the right circumstances, an attacker can use it to bypass a web application's authentication and authorization mechanisms and retrieve the contents of an entire database. SQL Injection can also be used to add, modify and delete records in a database, affecting data integrity.

To such an extent, SQL Injection can provide an attacker with unauthorized access to sensitive data including, customer data, personally identifiable information (PII), trade secrets, intellectual property and other sensitive information.

How SQL Injection works

In order to run malicious SQL queries against a database server, an attacker must first find an input within the web application that is included inside of an SQL query.

In order for an SQL Injection attack to take place, the vulnerable website needs to directly include user input within an SQL statement. An attacker can then insert a payload that will be included as part of the SQL query and run against the database server.

SQL injection can be illustrated as below using the same fields

User-id: ‘OR 1=1;/* (when someone inputs that given id in the field)

Password: */----

The resultant execution would be;

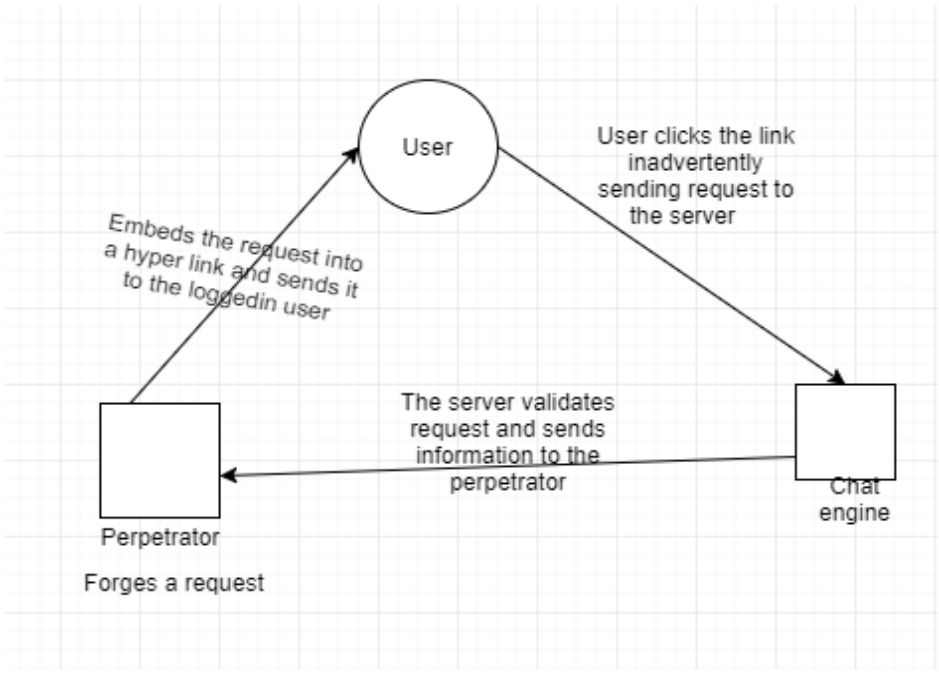
Select*from users where user id=’ ‘OR 1=1; /*’ and password =’*/---‘

This is so malicious resulting into unexpected execution.

3 Cross-site request forgery (CSRF). CSRF is an attack that tricks the victim into submitting a malicious request. It inherits the identity and privileges of the victim to perform an undesired function on the victim's behalf. For most sites, browser requests automatically include any credentials associated with the site, such as the user's session cookie, IP address, Windows domain credentials, and so forth. Therefore, if the user is currently authenticated to the site, the site will have no way to distinguish between the forged request sent by the victim and a legitimate request sent by the victim.

CSRF attacks target functionality that causes a state change on the server, such as changing the victim's email address or password, or purchasing something. Forcing the victim to retrieve data doesn't benefit an attacker because the attacker doesn't receive the response, the victim does. As such, CSRF attacks target state-changing requests.

Cross-site request forgery can be demonstrated as below



h) Cross-Site Scripting: By use of regular expressions in order not to allow active JavaScript code to be submitted into the system and also make sure that the system doesn't use data it has not self generated .

SQL Injection: – The system must always use parameterized queries and avoid standard Transact SQL as this would allow hackers to insert rogue code.

Cross-site request forgery: Transition from cookies that perform session tracking to session tokens that are dynamically generated. This makes it more difficult for attacker to get hold of a session.

g) Exception/error handling is a practice where the program continues to execute in an intended way when it comes across an error or an exception. A programmer implements a handling mechanism as a practice for secure programming to achieve the fail secure principle of programming. This can be used to handle errors that occur during data submissions and database queries

Regular expressions refer to the set of rules or combinations that denotes the structure of data usually strings. For example, a regular expression that can be used to allow emails.

(/ ^([a-z]+)([a-z A-Z0-9_\-.]+) @ ([a-zA-Z0-9_\-.]+)\.[a-zA-Z]{2,5}$/). Can be used to make sure that only strings of the above format can be accepted past the mechanism.

QUESTION TWO

a) Input sanitization refers to the way of describing, cleansing and scrubbing user input to prevent it from jumping the fence and exploiting security holes through modifying the input to ensure that it is valid for example changing all single quotation marks in a string to double quotation marks (sanitize).

b) SQL injection: These are Attacks which try to exploit an underlying SQL database for the chat engine and can use faulty input sanitization to their advantage. One of the first probes an attacker will make is to test whether the given code sends SQL queries without any data sanitization. This is often done with a single quote. If the backend does not sanitize input, it will cause a syntax error when the database interprets the SQL. Chances are that the error output will be pushed to the Web page and the hacker will now know that they can construct more sophisticated inputs to add clauses to the SQL query that may dump data from your database

Cross-Site request forgery (CSRF): CSRF is an attack that tricks the victim into submitting a malicious request. It inherits the identity and privileges of the victim to perform an undesired function on the victim's behalf, according to the above code, the CSRF token is not handled leaving the form vulnerable to cross-site request forgery done by a malicious user

c) $safe_data=filter_input (INPUT_GET, 'comment', FILTER_SANITIZE_SPECIAL_CHARS);

According to the above line of code, it is taking the comment field and then sanitizes it by removing the tags and replaces it with ampersand signs such that the code is not rendered by the browser

d) **Masking**: This is where a certain column input fields is replaced with a mask character such as an X. This effectively disguises the data content while preserving the formatting, the masking characters effectively remove the sensitive content from the record.

Shuffling records: In this technique, the data in the column is randomly moved between rows until there is no longer any reasonable correlation with the remaining information in the row.

e) Implementing authentication: This can be done by ensuring that the username and password, biometrics, Digital Certificates is correctly validated.

Ensuring Authorization this is also another way of practicing secure programming by Using Access Control Lists (ACLs)

f) Using prepared SQL statements in the base code doesn't guarantee security over SQL injections because the SQL statements can be taken advantage of to execute other malicious SQL statements so, this can be overcame/ mitigated by using Regular expressions on top of other data validation techniques