SSD

SSD CT Study Checklist

Week 1 - Module Intro Slide Deck –

o Secure Software Development Life Cycle (SSDLC) based on the following slides:

- Module Synopsis
- Aims of the module

Type of questions asked: Given a scenario suggest how SSDLC can help a company?

Ans: A company will be able to build secure software by gathering security requirement, implement secure software design, code, testing, and deployment, which will protect the company’s software from hacker and user failure.

Week 2 - Security Concepts

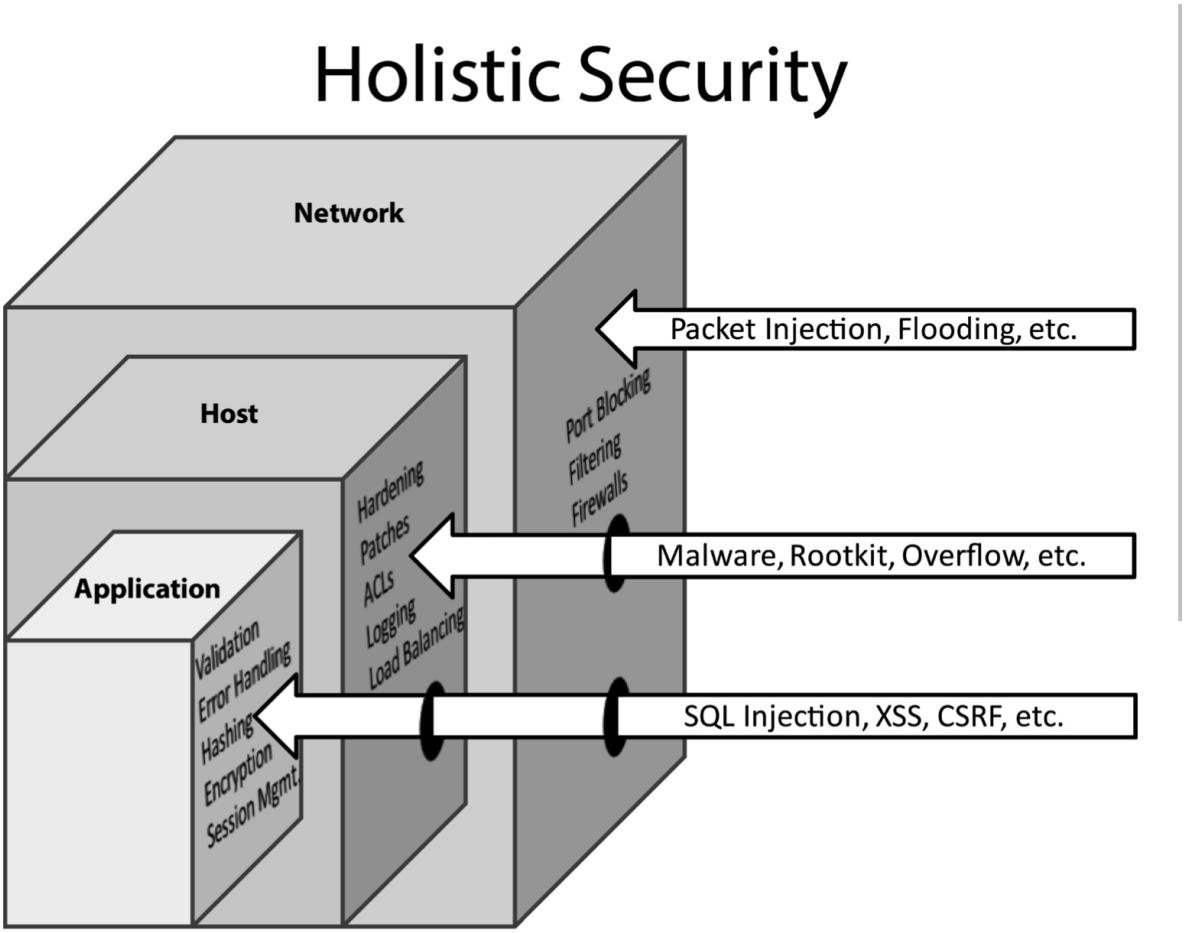


Figure 1.1 – Securing the Network, Hosts and Application Layer

o Holistic Security (secure applications running on secure hosts in secure networks.)

o Challenges

- **Iron Triangle** (scope, schedule, cost)
    - If the software development project's scope, schedule (time), and budget are very rigidly defined (as is often the case), it gives little to no room to incorporate even the basic, let alone additional security requirements into the software and unfortunately what is typically overlooked are elements of software security.
- **Security vs Usability**
    - Challenge to incorporate secure features in software is that the incorporation of secure features is viewed as rendering the software to become very complex, restrictive, and unusable.
- **Security as an afterthought**
    - Secure features are built into the software, instead of being added on at a later stage, since it has been proven that the cost to fix insecure software

earlier in the software development life cycle (SDLC) is insignificant when compared to having the same issue addressed at a later stage of the SDLC.

    - Devs don’t see a point in investing in security as it is hard to show a one-to-one return on security investments

Type of question: Given a scenario, identify and explain what type of challenge(s) is being faced by the company.

Week 2 Research Point (in Week 3 Folder Week2ResearchPoint.pdf)

o Attacker vs Defender

- Attacker has upper hand as they only need to exploit one vulnerability, while defender needs to defend all possible vulnerabilities. Defenders must play by the rules, attackers do not. Defenders must guard 24/7

o Auditing & Accountability

- Accountability and auditing (who did what when and where) important to achieve non-repudiation

o Non-Repudiation

- Cannot deny an action

o Data Privacy / Anonymization Techniques (Focus)

- Replacement
- Suppression (Action, measure that reduces the level of, or inhibits the generation of, compromising emanations/result in an information system.)
- Generalization (**Transforming one value into a more imprecise one**. Generalisation techniques preserve the confidentiality of data by mapping sensitive features to more general values)
- Pertubation (Data perturbation is **a data security technique that adds 'noise' to databases allowing individual record confidentiality**. This technique allows users to ascertain key summary information about the data that is not distorted and does not lead to a security breach.)

Type of Question: Given a scenario, identify which of the above points are either applied or not applied.

Week 3 - Security Requirements

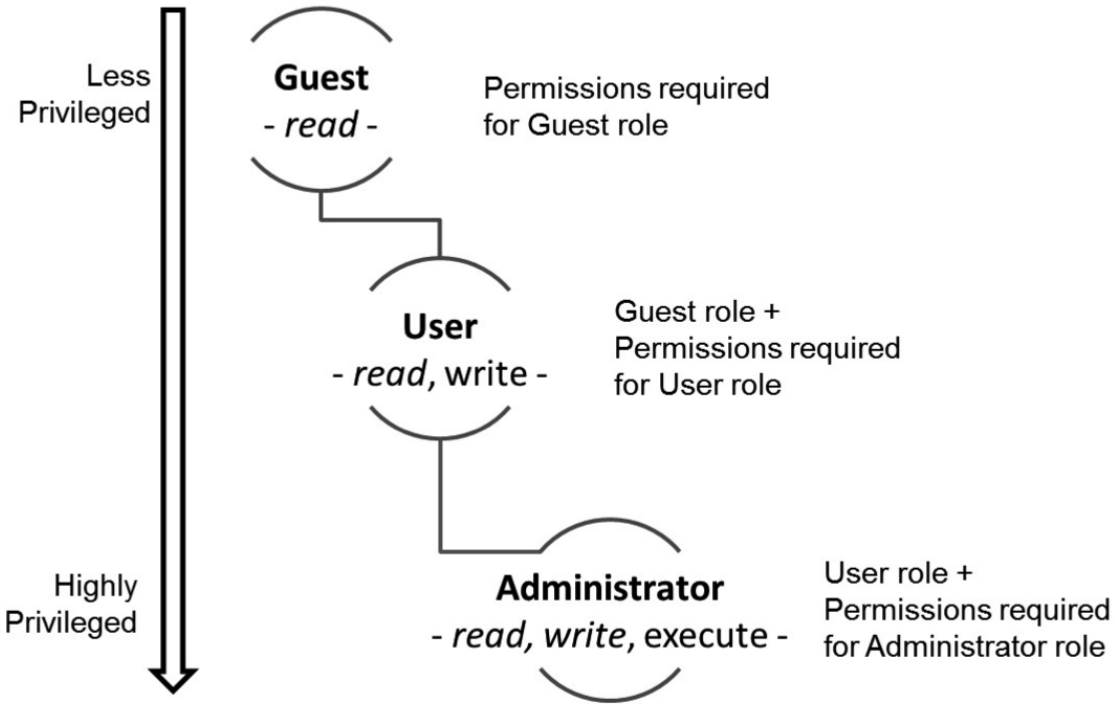


Figure 2.8 – **Role Hierarchy**

- Quality Attributes of Secure Software

- Reliability, software works as intended

- Resiliency, software can withstand attacks from threat agents be it intentional or accidental. Also does not violate any security policy

  - Recoverability (restore software operations), able to restore back to business operations by limiting, containing, and remediating threats that materialize

Type of question: Match a given scenario to the above quality attributes

- Confidentiality
  - Encryption and Decryption (Data in rest/stop), HTTPS (Data in transit), SSL to be specific
- Integrity
  - Hashing (**Salted hash** - adding random data to the input of a hash function to guarantee a unique output, the hash, even when the inputs are the same. **/ Unsalted hash**)

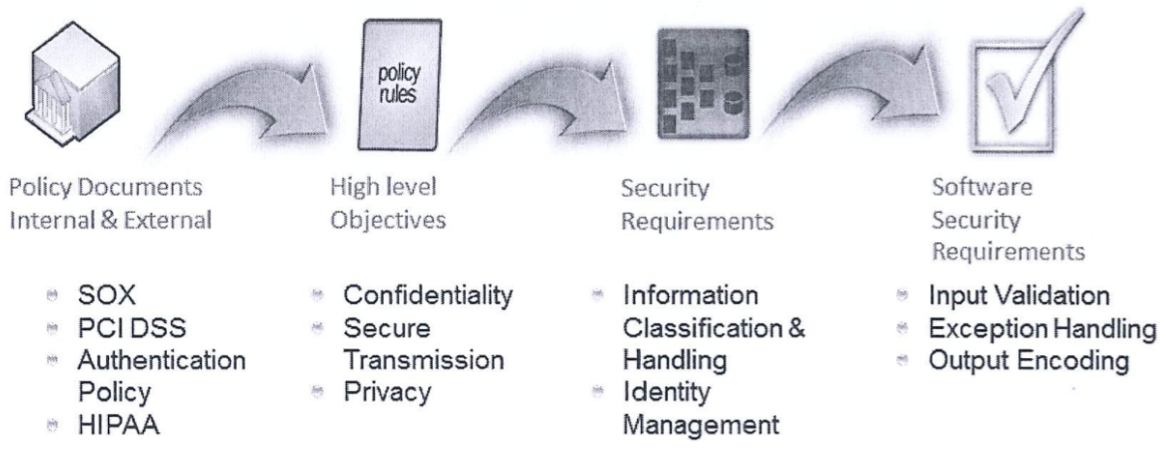
    - Benefits of Salted Hash (make it computationally infeasible to precompute possible password hashes, because the random salt screws up the pre-computation process.)
    - Example of Salted Hash (Passwords)
- Availability
    - Service Level Agreement 99.999% (5'9s), occurrence of breakdown of software in a year
    - Time frame given to recover the software from time of disruption
- Authentication
    - Forms (HTTPS), most popular
- Ways of authentication
    1. What you know (knowledge)
    2. What you are (biometrics)
    3. What you have (possession)
- Authorization
    - Role Based Access Control
    - Checks that subject has enough privileges to perform intended action on object
- Accountability and Auditing
    - Logging (**WHO** did **WHAT**, **WHERE** and **WHEN**)?
- Session Management – each user activity needs to be uniquely tracked
    - Session ID (Cookie)
- Errors and Exception Management
    - Non-Verbose error message (Not too detailed, min info revealed)
- Explicit vs High-Level Requirements
    - Explicit: usually not found within the software requirements specifications document.
    - How the instructions towards the software developer are detailed
    - Explicit = Direct instructions (Use SHA256 hash)

    - High level = How it operates (infer what is needed from the way the software operates)

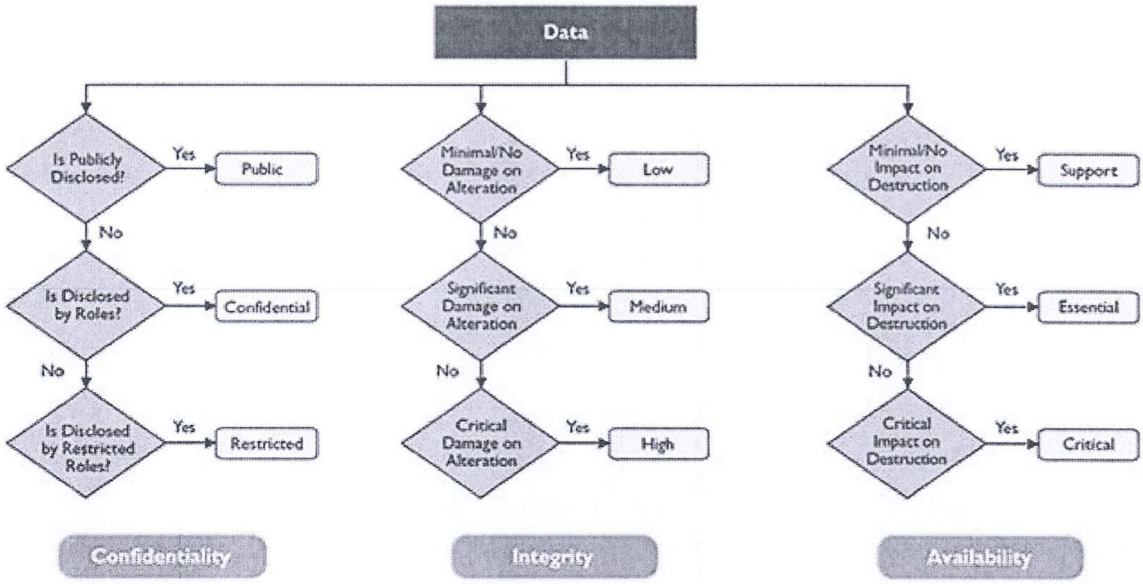
Type of question: Given a scenario identify and explain which of the above security requirements are applicable to the given scenario OR how the above security requirements can be applied to improve the given scenario.

Week 4 - Gathering Secure Software Requirements

- Brainstorming
    - Issues
    - Finding Available time to come together and brainstorming
    - People may be introverted and not contribute
    - Takes a long time (not all details can be captured)
    - Go off topic on irrelevant matters
    - Used for preliminary security requirements
- Surveys and Questionnaires
    - Ask questions related to the security requirement in a nontechnical manner (descriptive manner) for ease of understanding
    - Quality depends on questions asked (want open ended to yield more security requirements that might be missed and need to be addressed)
    - Good communication between both parties
    - Scribe to record interviewee responses
- Policy Decomposition
    - Extracting security requirements from existing policies can be external / internal or both
    - From policy to high level objectives to .. to ..



- Data Classification
  - To identify which data to protect or make publicly available (refer to the labelling Week 4 Slide 9)



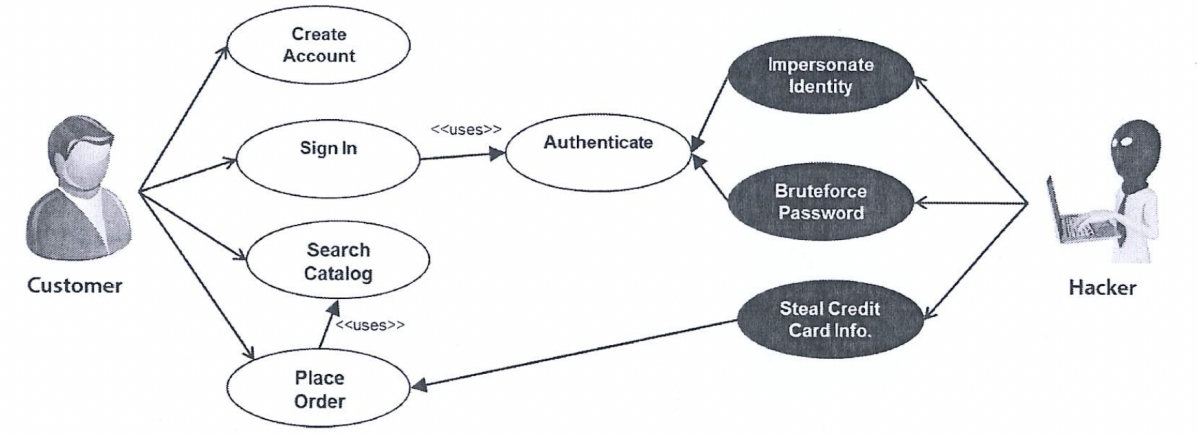
Type of question: Match a given scenario to the above PNE / Secure Software Requirements Gathering Techniques

- Subject-Object Matrix
  - 2-dimensional matrix that identifies the subjects and objects to perform CRUD (Create, Read, Update, Delete)
  - Subjects (roles) across the columns, objects (components) down the rows

		User Roles		
		Administrator	**Customer**	**Sales Agent**
Data	**Customer Data**	C, R, U, D	C, R, U, D	C, R, U
	Product Data	C, R, U, D	R	R, U
	Order Data		C, R, U, D	C, R, U, D
	Credit Card Data		C, R, U, D	R, U

Figure C.7 – **Data Access Control Matrix**

- Use Case and Misuse Case Modelling
  - Use case - Actions that can be performed by users
  - Misuse case - Actions that unauthorised users can perform
  - Misuses cases can be intentional or accidental
  - Help identify security requirements by modelling negative scenarios



Type of question: Given a scenario come up with Subject-Object Matrix and Use Case and Misuse Case diagrams

Week 5 / Week 6 - Secure Software Design

Benefits of Secure Software Design

- Minimizes the time needed to fix identified security issues

- Assures stakeholders trust because software is:
  - Reliable (functioning as expected and less prone to errors)
  - Resilient (less likely to be successfully exploited)
  - Recoverable (more prone to be able to restore itself to normal business operations upon error or exploitation)
- Requires minimal redesign when the software is designed with future threats in mind
- Designing software with security in mind, business logic flaws and other architectural design issues can be uncovered.

- Encryption
  - Symmetric Key Cryptosystem
    - No Complex Mathematical Formulas
    - Fast
    - Transposition, Bitwise XOR function, Substitution
  - Asymmetric Key Cryptosystem
    - Exponentiation (computers hate it !!!!! SLOW!!@JOJO)
    - Combining Symmetric Key Cryptosystem and Asymmetric Key Cryptosystem (HTTPS / SSL)
- Integrity
  - Hashing
    - Salted Hash
      - Frustrate Hackers (they need to crack the hash again even if it’s the same password)
      - Defends against Dictionary Attack and Rainbow Attacks
    - Unsalted Hash
      - Same passwords will produce the same hash and the Hackers does not require to crack it again
    - Characteristics of hashing

        - Unique, Original, Fixed length, Irreversible
    - Referential Integrity
        - Ensure that data is not left in an orphaned state.
        - If customer orders are tied to a customer and the customer is deleted/updated from the database, the order records must be deleted/updated as well (cascading deletes/updates) . Failure to have referential integrity would lead to order (child) records existing in the database without being tied to a customer (parent)
    - Resource Locking
        - Two updates on the same resource are stopped
        - Poor design results in a Deadlock
- Availability techniques
    - Failover (more for applications)
        - Automatic switching from active (primary) system to standby system
        - Potential Single Points of failure are addressed during the designing of solution
    - Replication (more for DB)
        - Active / Active is where Master (Primary Node) and Slaves (Secondary Node) get updated at the same time
        - Active / Passive is where Master (Primary Node) gets updated first, Slaves (Secondary Node) later
    - Scalability
        - Able to handle an increase traffic load
        - Vertical Scaling (scaling UP)
            - Increasing memory and storage resources
        - Horizontal Scaling (scaling OUT)
            - Installing additional copies of software
- Authentication
    - Multi-Factor Authentication

    - Example: OTP, Authenticator Code
    - What you have, What you are and What you know? 3 factors
    - Single Sign On (SSO)
        - Only needs to verify once
        - Sign into multiple services
        - Example: Microsoft Office, Google
        - Poor design can result in major security breaches as it can lead to total compromise since all services are somewhat connected.
    - Federation
        - Does not require registration of another service
        - Example: LinkedIn can be logged into by using a Google Account

Type of question: Given a scenario identify and explain which of the above security design techniques are applicable to the given scenario or how the above security requirements can be applied to improve the given scenario.

Key Secure Design Principles

- Least Privilege
    - Attack surface area reduced when minimum privilege is given to users to perform their tasks
    - Example: Modular Programming
    - Execute Smaller code to perform actions specific to role of user
- Separation of Duties
    - Reduces the extent of damage done by one person or resource
    - With auditing, discourage insider fraud as it required collusion with multiple parties
    - Example: Programmer should not be allowed to review their own code
- Defence-in-Depth
    - Having multiple layers of defence (mitigate Single point of failure, weakest link threats)

    - Layering Security Controls
    - Breach of a single vulnerability does not result in total compromise
    - Incorporating in-depth software is use as a deterrent for the curious and non-determined attackers when they are confronted with one defensive measure over another
    - Input Validation to prevent against malicious code/dynamic query constructions, Security Zones (separate different levels of access)
    - **E.g.** Use of security zones, which separates the different levels of access according to the zone that the software or person is authorised to access.
- Fail Secure
    - Ensures that the software reliably functions when attacked and is rapidly recoverable into a normal business and secure state in the event of design or implementation failure
    - Error Management
    - Account lockout mechanism
    - Example: Too many failed password entry, General Error Messages
- Economy of Mechanism
    - Unnecessary Functionalities should be avoided
    - It states that the more complex a software is the more likely there is of vulnerabilities
    - Strive for simplicity in design and operational ease of use as attack surface will be lower and fewer weak links
    - Devs adding extra functionalities is contrary to Economy of mechanisms
    - E.g. Modular programming & SSO (Single Sign On)
- Complete Mediation
    - Access requests need to be mediated each time, every time, so that authority is not circumvented in subsequent requests.
    - Protects against authentication/confidentiality threats, useful in addressing the integrity aspects of software
    - Parameter values when changed leads to an appropriate action

    - Example: changing username in the parameter results in the site going to the login page
- Open Design
    - Implementation of security should be independent of the design. Inverse is security through obscurity which is bad because just the understanding of the inner working of the mechanism is all it takes to defeat the protection mechanisms (obscurity)
    - Security of software is not dependent on the secrecy of the design
    - E.g. of Open Design is AES
- Least Common Mechanisms
    - Principle by which mechanisms available to more than 1 user should not be shared as shared mechanisms especially those with shared variables represent a potential information path
    - Design should isolate the code by user roles as it increases security by limiting the exposure
    - E.g. Instead of having 1 function that is shared between members with supervisor and non-supervisor roles, have 2 distinct functions, each serving its respective roles
- Psychological Acceptability
    - Security mechanisms should be designed to maximize usage, adoption, and automatic application
    - Easy to use, don’t affect usability, transparent to users
    - E.g. designing the software to notify the user through explicit error messages and callouts
- Weakest Link
    - Resiliency of your software against hacker attempts will depend heavily on the protection of its weakest components (code, service, interface). A breakdown in the weakest link will result in a security breach and can lead to compromise in other areas.
    - Related concept: (Should not have) Single point of failure
    - Usually in software security, weakest link is several single points of failures

- Leveraging Existing Components
  - Promotes reusability of existing components
  - Use security management that are well reviewed instead of creating your own as it is often the weakest link (your own)
  - Also by reusing code that has been tested and proven gives benefits such as ,
    - Attack surface is not increased
    - No new vulnerabilities are introduced
    - Increase Productivity by reducing time needed to develop the existing components

Balancing Secure Design Principles

There is a trade off between principles and you cannot have all of them. Hence you have to decide which ones to enforce more. E.g. Psychological Acceptability and Complete mediation contradict each other and you can’t fully have both and must determine to what extent do you need to implement these principles.

Type of question: Match a given scenario the above Secure Software Design Principles

Week 7 - Secure Software Design

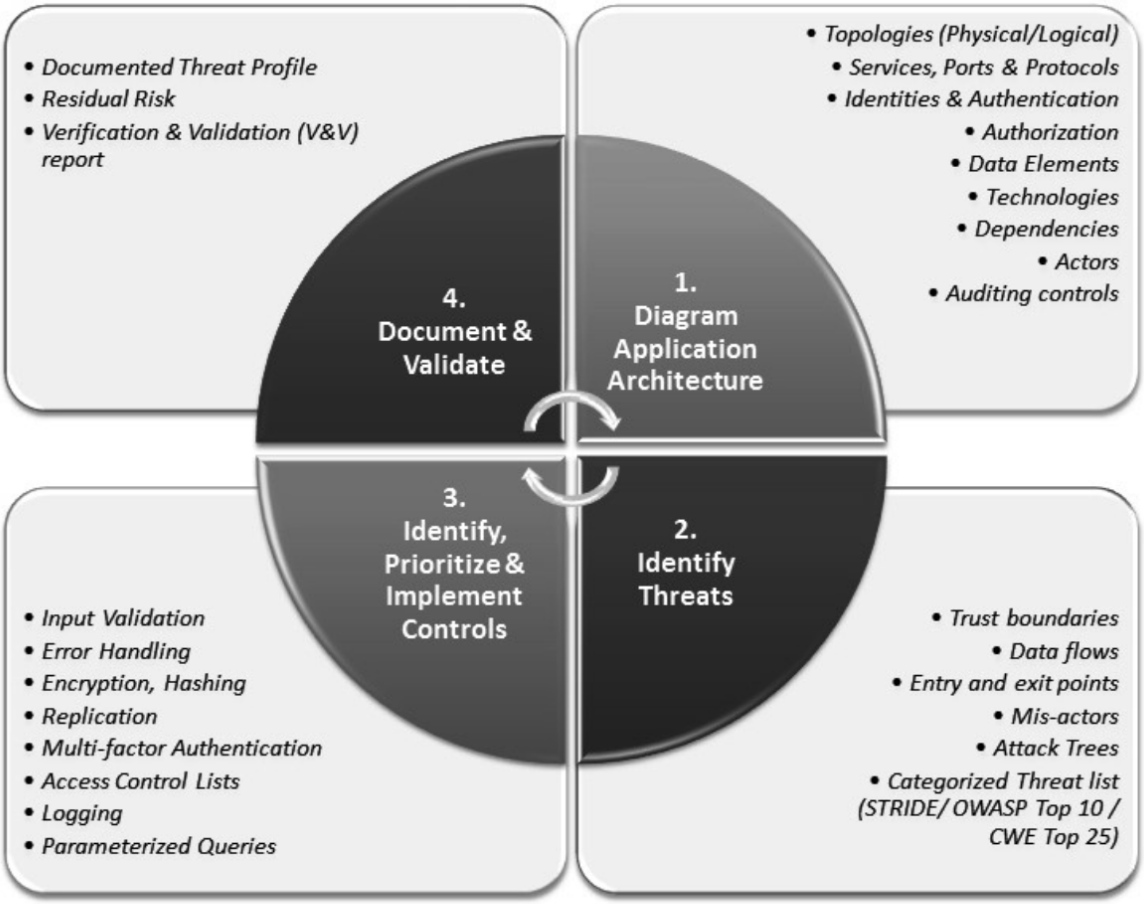


Figure C.1 – **Threat Modeling Process (Phases and Activities)**

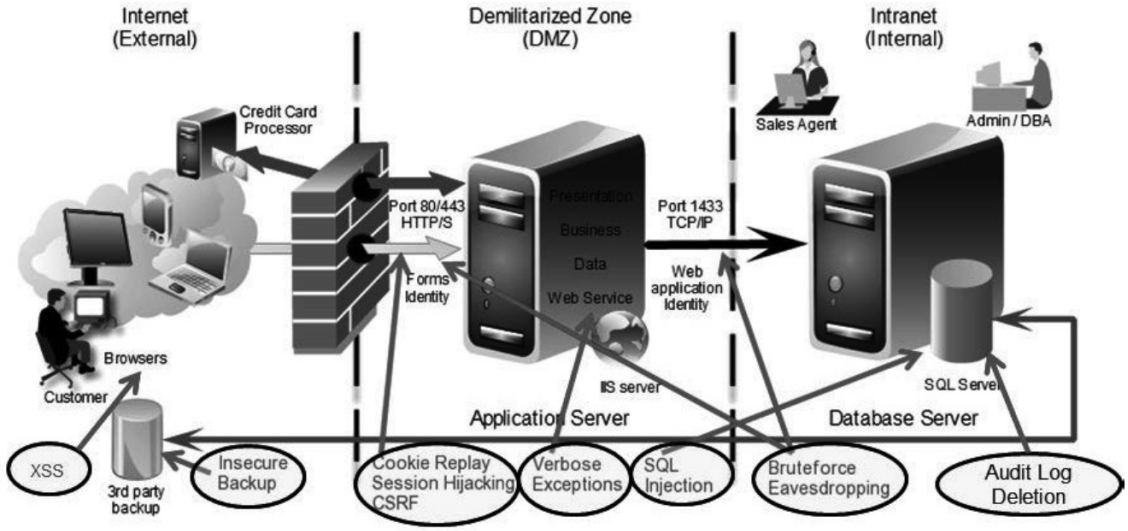


Figure C.10 – **Diagrammatically documents threats**

Threat Description	Injection of SQL commands
Threat targets	- Data access component - Backend database.
Attack techniques	- Attacker appends SQL commands to user name, which is used to form an SQL query.
Security Impact	- Information Disclosure. - Alteration. - Destruction (Drop table, procedures, delete data etc.). - Authentication bypass.
Safeguard controls to implement	- Use a regular expression to validate the user name. - Disallow dynamic construction of queries using user supplied input without validation. - Use parameterized queries.
Risk	- High

Table C.5 – **Textual documentation of a SQL Injection threat**

- Threat Modelling
  - Extremely crucial for developing hack-resilient software
  - Should address common threats
  - Identify entry and exit point
  - Threat Modelling Process
- **Model Application Architecture**
- Deployment structure (Servers)
- Ports / Protocol (HTTPS - 443)
- Identities and Authentication
- Actors (End users and staff)
- Subject object matrix
- External Dependencies (3 party services, web-server)

Threat Modeling: Document and Validate

- Document the threat modelling process.
- Validate the control measures applied are effective.

1. Project Title
2. Project Vision
3. Project Description
4. Project Features (with brief description)
5. List members and the features they worked on
6. Feature 1 (e.g., Registration):
    - 1.1 Secure Software Requirements
        1. Surveys (Questionnaires and Interviews)
        2. Data Classification
        3. Data Labeling
        4. Data Ownership
        5. Use Case and Misuse Case Modeling
        6. Subject/Object Matrix
        7. List relevant Confidentiality, Integrity, Availability, Authentication, Authorization, Accountability, Session Management, and Error/Exceptions Management.
    - 2.1 Secure Software Design
        1. Secure Software Design Principles
        2. Attack Surface Evaluation
        3. Threat Modeling
    - 3.1 Secure Software Coding

Identify Threats

- Entry Points (items that take in user input)
    - Any Page that requires updating by user
    - Preferences that are selected by user stored to the database
- Exit Points (items that display info from within the system)
    - Any Page that shows the result which calls on the database
    - Preferences of the page is stored on the database
    - Any page that changes based on the user's action
    - Can be the source of information leakage
    - Include processes that take data out of the system
- Apply **STRIDE**
    - Spoofing
    - Tampering
    - Repudiation
    - Information Disclosure
    - Denial of Service

- Elevation of Privilege

	Goal	Description
S	*Spoofing*	Can an attacker impersonate another user or identity?
T	*Tampering*	Can the data be tampered with while it is in transit or in storage or archives?
R	*Repudiation*	Can the attacker (user or process) deny the attack?
I	*Information Disclosure*	Can information be disclosed to unauthorized users?
D	*Denial of Service*	Is denial of service a possibility?
E	*Elevation of Privilege*	Can the attacker bypass least privilege implementation and execute the software at elevated or administrative privileges?

STRIDE List	*Identified Threats*
Spoofing	- Cookie Replay - Session Hijacking - CSRF
Tampering	- Cross Site Scripting (XSS) - SQL Injection
Repudiation	- Audit Log Deletion - Insecure Backup
Information Disclosure	- Eavesdropping Verbose Exception - Output Caching
Denial of Service	- Website Defacement
Elevation of Privilege	- Logic Flaw

Identify, Prioritize, and implement controls

- **DREAD**
  - Damage Potential (D)
  - Reproducibility (R)
  - Exploitability (E)
  - Affected Users (A)
  - Discoverability (Di)
- Risk = Probability (R+E+Di) X Impact (D+A)

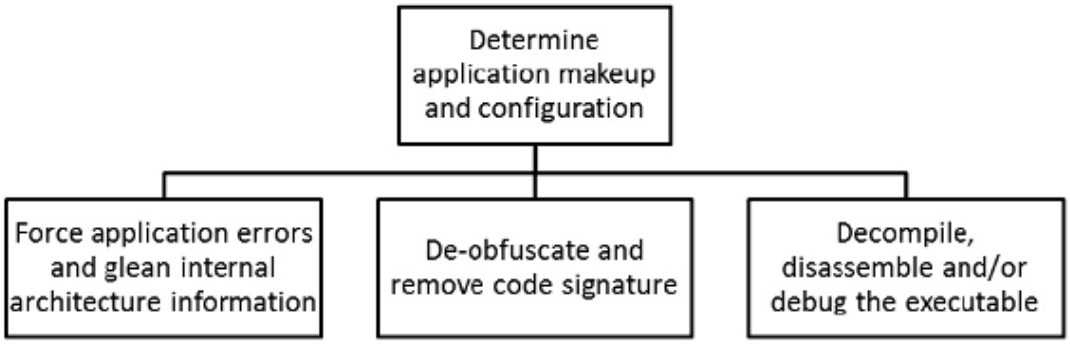


Figure 3.19 - ***Attack Tree:*** **Attacker's objective in the root node**

Figure 3.20 depicts an attack tree with an attack vector at its root node. When the root node is an attack vector, the child node from the root nodes is the unmitigated or vulnerability condition. The next level node (child node of an unmitigated condition) is usually the mitigated condition or a safeguard control to be implemented.

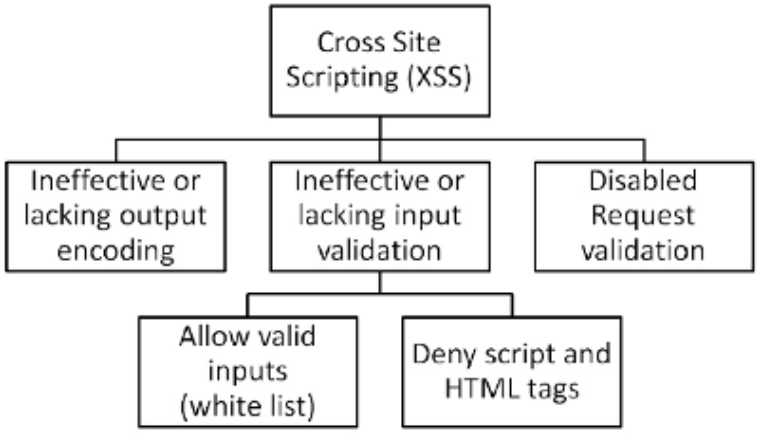


Figure 3.20 - ***Attack Tree:*** **Attack vector in the root node**

Type of question: Given a scenario identify the Entry and Exit points, and apply STRIDE and Calculate DREAD risk based on the given values and propose security controls to the top threats

---

Confidentiality controls assures protection against unauthorized disclosure.

Integrity controls assures protection unauthorized modifications or alterations.

Availability controls assures protection against downtime/denial of service

and destruction of information.

Authentication is the mechanism to validate the claims/credentials of an entity.

Authorization has to do with rights and privileges that a subject has upon

requested objects.

Anonymization is the process of removing private information from the data. Anonymization techniques such as replacement, suppression, generalization

and pertubation are useful to assure data privacy.

Sanitization has to do with inputs and outputs as a defensive control and includes techniques such as escaping and encoding.

Degaussing and Formatting are information and media sanitization

techniques and they are not selective of what they remove/dispose.

Example Questions:

PRIMARY reason for incorporating security into the software

development life cycle is to protect corporate brand and reputation.

MAIN reason as to why the availability aspects of software must

be part of the organization’s software security initiatives is software issues can cause downtime to the business.

Threat Agent Type	Description
Ignorant User	The ignorant user is the one that is often the cause of unintentional and plain user error. Plain user error is also referred to sometimes as plain error or simply user error. To combat this threat source is simple but requires investment in time and effort. User education is the best defense against this type of threat agent. Mere documentation and help guides are insufficient measures, if they are not used appropriately.
Accidental Discoverer	An ordinary user who stumbles upon a functional mistake in the software and who is able to gain privilege access to information or functionality. This user never sought to circumvent security protection mechanisms in the first place.
Curious Attacker	An ordinary user who notices some anomaly in the functioning of the software and decides to pursue it further. Often an accidental discoverer graduates into being a curious attacker.
Script Kiddies	This type of threat agents are to be dealt with seriously, merely because of their prevalence in the industry. They are those ordinary users who execute hacker scripts against corporate assets without understanding the impact and consequences of their actions. Most elite hackers today were one day script kiddies. A litmus test to the identification of a script kiddy's work is that they do not often hide or know how to hide their footprint on the software or systems they have attacked.
Insider	One of the most powerful attackers in this day and age is the insider or the enemy inside the firewall. These are potentially disgruntled employees or staff member within the company that has access to insider knowledge. The database administrator with unfettered and unaudited access to sensitive information directly is a potential threat source that should not be ignored. Proper identification, authentication, authorization and auditing of user actions are important and necessary controls that need to be implemented to deter insider attacks. Auditing can also be used as a detective control in the cases where insider fraud and attack is speculated.
Organized Cybercriminals	These are highly skilled malefactors that are paid professionally for using their skills to thwart security protection of software and systems, seeking high financial gain. They not only have a deep understanding of software development, but also of reverse engineering and network and host security controls. They can be used for attacks against corporate assets as well as are a threat to national security as cyber terrorists. Malware developers and Advance Persistent Threat (APT) hackers usually fall into this category as well.
Third Party/Supplier	When software is developed outside the purview of one's company control, then malicious logic and malicious code (malcode) such as logic bombs and Trojan horses can be unintentionally or intentional embedded in the software code, as the software moves through the supply chain. When outsourcing software development, the Foreign Ownership Control and Influence (FOCI) of the third party or supplier must be determined and *code inspection (review) prior to acceptance must be performed by the acquirer.*

Table 3.4 – **Human Threat Source/Agent**