بسم الله الرحمن الرحيم

KHAWAJAGAN Technologies

SP 2025

Secure Software Design and Development

CYC386

LECTURE 002 – 21-FEB-2025

FAISAL SHAHZAD

SP 2025

Secure Design Principles

Secure Design Principles

- Secure software development is a key element in improving the cybersecurity landscape in the worldwide enterprise.
- Software is used by virtually every company, and user base relies upon secure coding efforts as part of their security posture.
- In this lecture will discuss
  - Classic system tenets of session, exception, and configuration management.
  - How the secure design tenets are embodied in software solutions.
  - The common models used to describe complex security issues
  - Things that can go wrong, such as attacks, and how they impact software.

Secure Design Principles

Secure Design Principles

- The creation of software systems involves the development of several foundational system elements within the overall system.
- Communication between components requires the management of a communication session, commonly called **session management**.
- When a program encounters an unexpected condition, an error can occur.
- Securely managing error conditions is referred to as **exception management**.
- Software systems require configuration in production, and **configuration management** is a key element in the creation of secure systems.

SDP – Session Management

- Software systems require communications between program elements or between users and program elements.
- The control of the communication session is essential to prevent the hijacking of an authorized communication channel by an unauthorized party.
- Session management refers to the design and implementation of controls to ensure that communication channels are secured from unauthorized access and disruption of a communication.

SDP – Exception Management

- A software system may encounters an unknown condition or is given input that results in an error.
- The process of handling these conditions is exception management.
- It is important for the system to respond in an appropriate fashion in case a remote resource may not respond, or there may be a communication error.
- Several criteria are necessary for secure exception management.
  - All exceptions must be detected and handled.
  - The system should be designed so as not fail to an insecure state.
  - All communications associated with the exception should not leak information.

SDP – Configuration Management

- Dependable software in production requires the managed configuration of the functional connectivity associated with today's complex, integrated systems.
- Initialization parameters, connection strings, paths, keys, and other associated variables are typical examples of configuration items.
- As these elements can have significant effects upon the operation of a system, they are part of the system and need to be properly controlled for the system to remain secure.
- The identification and management of these elements is part of the security process associated with a system.

SDP – Configuration Management

- Management has a responsibility to maintain production systems in a secure state.
- This requires protection of configurations from unauthorized changes.
- Configuration management, change control boards, and a host of workflow systems designed to control the configuration of a system.
- Separation of duties between production personnel and development/test personnel is frequently applied.
- This separation is one method to prevent the contamination of approved configurations in production.

SP 2025

Secure Design Tenets

Secure Design Tenets (SDT)

- Secure designs do not happen by accident.
- They require deliberate architecture and deliberate plans and are structured upon a foundation of secure design principles.
- These principles have borne the test of time and have repeatedly proven their worth in a wide variety of security situations.

SDT – Providing Enough Security

- Security is never an absolute, and there is no such thing as complete or absolute security.
- This is an important security principle, for it sets the stage for all of the security aspects associated with a system.
- There is a trade-off between security and other aspects of a system.
- Secure operation is a requirement for reliable operation, but under what conditions?
  - Every system has some appropriate level of required security, and it is important to determine this level early in the design process.
  - Just as one would not spend $10,000 on a safe to protect a $20 bill, a software designer should not use national security–grade encryption to secure publicly available information.

SDT – Least Privilege

- Least Privilege is the most fundamental approaches to security ensuring that a subject should have only the necessary rights and privileges to perform its current task with no additional rights and privileges.
- Limiting a subject's privileges limits the amount of harm that can be caused, limiting a system's exposure to damage.
- If a subject may require different levels of security for different tasks, it is better to switch security levels with specific tasks, rather than run all the time at a higher level of privilege.
- Security context in which an application runs is also important to consider.
- All programs, scripts, and batch files run under the security context of a specific user on an operating system.
- If the program was compromised after it had entered the root access state, the attacker could obtain a root-level shell.
- Programs should execute only in the security context that is needed for that program to perform its duties successfully.

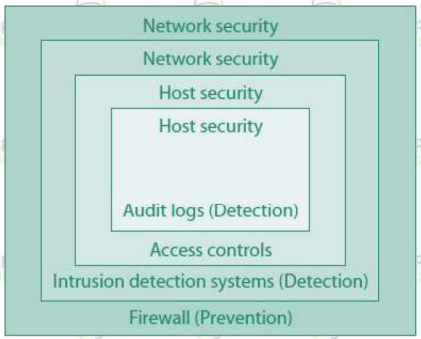
SDT – Separation Of Duties

- Separation of duties ensures that for any given task, more than one individual needs to be involved.
- The critical path of tasks is split into multiple items, which are then spread across more than a single party.
- By implementing a task in this manner, no single individual can abuse the system.
  - A simple example might be a system where one individual is required to place an order and a separate person is needed to authorize the purchase.
- This separation of duties must be designed into a system.
- Software components enforce separation of duties when they require multiple conditions to be met before a task is considered complete.
- These multiple conditions can then be managed separately to enforce the checks and balances required by the system.

SDT – Defense In Depth

- Defense in depth is oldest security principles also known as Layered Security based on the idea that "If one defense is good, multiple overlapping defenses are better".
  - A castle has a moat, thick walls, restricted access points, high points for defense, multiple chokepoints inside, etc., working together as a whole series of defenses all aligned toward a single objective.
- Software should utilize the same type of layered security architecture.
- There is no such thing as perfect security. No system is 100 percent secure, and there is nothing that is foolproof, so a single specific protection mechanism should never be solely relied upon for security.
- Every piece of software and every device can be compromised or bypassed in some way, including every encryption algorithm, given enough time and resources.
- The true goal of security is to make the cost of compromising a system greater in time and effort than it is worth to an adversary.

SDT – Defense In Depth



SDT – Fail-Safe Mechanism

- All systems will experience failures.
- The fail-safe design principle ensures that when a system experiences a failure, it should fail to a safe state.
- One implementation of fail-safe is to use the concept of explicit deny. Any function that is not specifically authorized is denied by default.
- When a system enters a failure state, the attributes associated with security, confidentiality, integrity, and availability need to be appropriately maintained.
- Availability is the attribute that tends to cause the greatest design difficulties. Ensuring that the design includes elements to degrade gracefully and return to normal operation through the shortest path assists in maintaining the resilience of the system.
- During design, it is important to consider the path associated with potential failure modes and how this path can be moderated to maintain system stability and control under failure conditions.

SDT – Economy of Mechanism

- The terms security and complexity are often at odds with each other as more complex something is, the harder it is to understand, and hardest it is to secure.
- When something goes wrong with security mechanisms, a troubleshooting process is used to identify the actual issue.
- If the mechanism is overly complex, identifying the root of the problem can be overwhelming, if not nearly impossible.
- Security is a complex issue because there are so many
  - Variables involved
  - Types of attacks and vulnerabilities
  - Different types of resources to secure
  - Different ways of securing them
- Security processes and tools should be as simple and elegant as possible.
- They should be simple to troubleshoot, simple to use, and simple to administer.

SDT – Economy of Mechanism

- Another application of the principle of keeping things simple concerns the number of services that you allow your system to run.
- Default installations of OS comes with many services running.
- The keep-it-simple principle requires to eliminate unnecessary services.
- It results in fewer applications that can be exploited and fewer services that the administrator is responsible for securing.
- The general rule of thumb is to always eliminate all nonessential services and protocols.
- A stringent security approach is to assume that no service is necessary for initial blocking of all services and later activation of services and ports only as they are requested / required as per work requirements.

SDT – Complete Mediation

- The principle of complete mediation states that when a subject's authorization is verified with respect to an object and an action, this verification occurs every time the subject requests access to an object.
- The system must be designed so that the authorization system is never circumvented, even with multiple, repeated accesses.
- Same principle is used by security kernel in operating systems, functionality that cannot be bypassed, and allows security management of all threads being processed by the operating system (OS).
- Modern operating systems and IT systems with properly configured authentication systems are difficult to compromise directly, with most routes to compromise involving the bypassing of a critical system such as authentication.
- It is important during design to examine potential bypass situations and prevent them from becoming instantiated.

SDT – Open Design

- The idea of security through obscurity has not been effective in the actual protection of the object.
- Security through obscurity just make it a little harder to accomplish a security breach task, but it does not provide actual security.
- This approach has been used in software to hide objects, such as keys and passwords, buried in the source code.
- Reverse engineering and differential code analysis have proven effective at discovering these secrets, eliminating this form of "security."
- The concept of open design states that the security of a system must be independent of the form of the security design.
- The algorithm that is used will be open and accessible, and the security must not be dependent upon the design, but rather on an element such as a key.
- Modern cryptography has employed this principle effectively; security depends upon the secrecy of the key, not the secrecy of the algorithm being employed.

SDT – Least Common Mechanism

- The concept of least common mechanism refers to a design method designed to prevent the inadvertent sharing of information.
- Having multiple processes share common mechanisms leads to a potential information pathway between users or processes.
- Having a mechanism that services a wide range of users or processes places a more significant burden on the design of that process to keep all pathways separate.
- When presented with a choice between a single process that operates on a range of supervisory and subordinate-level objects and/ or a specialized process tailored to each, choosing the separate process is the better choice.
- The concepts of least common mechanism and leveraging existing components can place a designer at a conflicting crossroad. One concept advocates reuse and the other separation.
- The choice is a case of determining the correct balance associated with the risk from each.

SDT – Leverage Existing Components

- Component reuse has many business advantages, including increases in efficiency and security.
- As components are reused, fewer new components are introduced to the system; hence, the opportunity for additional vulnerabilities is reduced.
- This is a simplistic form of reducing the attack surface area of a system.
- The downside of massive reuse is associated with a monoculture environment, which is where a failure has a larger footprint because of all the places it is involved with.

SDT – Least Common Mechanism

- The concept of least common mechanism refers to a design method designed to prevent the inadvertent sharing of information.
- Having multiple processes share common mechanisms leads to a potential information pathway between users or processes.
- A mechanism that services a wide range of users or processes places a more significant burden on the design of that process to keep all pathways separate.
- When presented with a choice between a single process that operates on a range of supervisory and subordinate-level objects and/ or a specialized process tailored to each, choosing the separate process is the better choice.
- The concepts of least common mechanism and leveraging existing components can place a designer at a conflicting crossroad. One concept advocates reuse and the other separation. The choice is a case of determining the correct balance associated with the risk from each.

SDT – Psychological Acceptability

- Users are a key part of a system and its security. Including a user in the security of a system requires that the security aspects be designed so that they are psychologically acceptable to the user.
- When a user faced a security system that appears to obstruct the user, the result will be the user working around the security aspects of the system.
  - If a system prohibits the e-mailing of certain types of attachments, the user can encrypt the attachment, masking it from security, and perform the prohibited action anyway.
- Ease of use tends to trump many functional aspects.
- The design of security in software systems needs to be transparent to the user, just like air—invisible, yet always there, serving the need.
- This places a burden on the designers to design the system's security which is a critical functional element in a way that should impose no / minimal burden on the user.

SDT – Weakest Link

- The weakest link is the common point of failure for all systems and every system by definition has a "weakest" link.
- Adversaries do not seek out the strongest defense to attempt a breach.
- A system can be considered only as strong as its weakest link.
- Expending additional resources to add to the security of a system is most productive when it is applied to the weakest link.
- Throughout the software lifecycle, it is important to understand the multitude of weaknesses associated with a system, including the weakest link.
- Defense in depth strategy while designing security, is critical to hardening a system against exploitation.
- Managing the security of a system requires understanding the vulnerabilities and defenses employed, including the relative strengths of each one, so that they can be properly addressed.

SDT – Single Point of Failure

- Just as multiple defenses are a key to a secure system, so, too, is a system design that is not susceptible to a single point of failure.
- A single point of failure is any aspect of a system that, if it fails, means the entire system fails.
- It is imperative for a secure system to not have any single points of failure.
- The design of a software system should be in a way that all points of failure are analyzed and a single failure does not result in system failure.
- Single points of failure can exist for any attribute, confidentiality, integrity, availability, etc., and may well be different for each attribute.
- Examining designs and implementations for single points of failure is important to prevent this form of catastrophic failure from being released in a product or system.

SP 2025

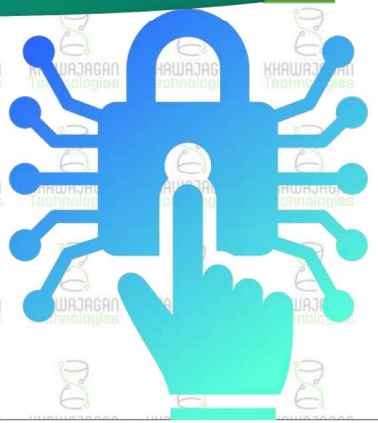
Security Models

Security Models

- Models are used to provide insight and explanation.
- Security models are used to understand the systems and processes developed to enforce security principles.
- Three key elements play a role in systems with respect to model implementation:
  - People
  - Processes
  - Technology
- Controls that rely on a single element, regardless of the element, are not as effective as controls that address two or all three elements.

SP 2025

Access Control Models



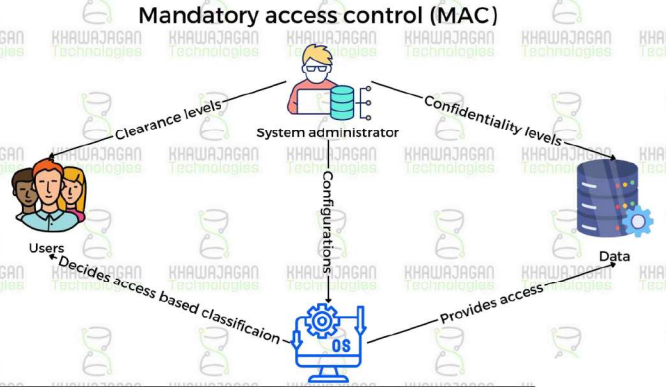
Access Control Models (ACM)

- Access controls define what actions a subject can perform on specific objects to ensure protection.
- Access controls assume that the identity of the user has been verified through an authentication process.
- There are different access control models that emphasize different aspects of a protection scheme.
- One of the most common mechanisms used is an access control list (ACL) which is a list that contains the subjects that have access rights to a particular object.
- An ACL identify subject, and specific access that subject has for the object. Typical types of accesses include read, write, and execute.
- Most common ACL models are discretionary access control (DAC), mandatory access control (MAC), role-based access control (RBAC), and rule-based access control (RBAC).

ACM – MAC Model

- MAC has its roots in military control systems, and referring to the Orange Book, a Department of Defense document, a standard for describing what constituted a trusted computing system,
- Mandatory access controls, is "a means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization (i.e., clearance) of subjects to access information of such sensitivity."
- In MAC systems, the owner or subject can't determine whether access is to be granted to another subject; it is the job of the security controlling system to decide.
- In MAC, the security mechanism controls access to all objects, and individual subjects cannot change that access, which places the onus of determining security access upon the designers of a system, requiring that all object and subject relationships be defined before use in a system.
- SELinux, a specially hardened form of Linux based on MAC, was developed by the National Security Agency (NSA) to demonstrate the usefulness of this access model.

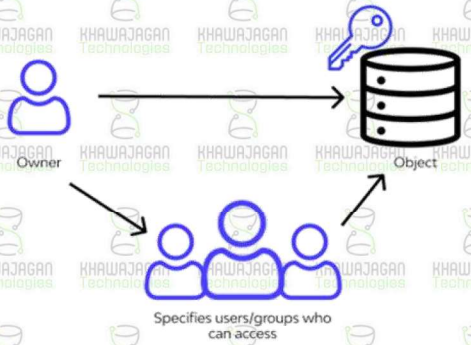
ACM – MAC Model



ACM – DAC Model

- Discretionary access control (DAC) is the most commonly employed model for access control.
- As defined by the Orange Book, discretionary access controls are "a means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject."
- In systems that employ DAC, the owner of an object can decide which other subjects may have access to the object and what specific access they may have. i.e. the owner of a file can specify what permissions are granted to which users.
- Access control lists are the most common mechanism used to implement DAC.
- The strength of DAC is its simplicity. The weakness is that it is discretionary, or, optional. Also DAC requires the object owner to define access for all objects under its control, a task that can end up with tens of thousands of access control decisions.
- To facilitate the scaling of DAC, group-based models can be employed.

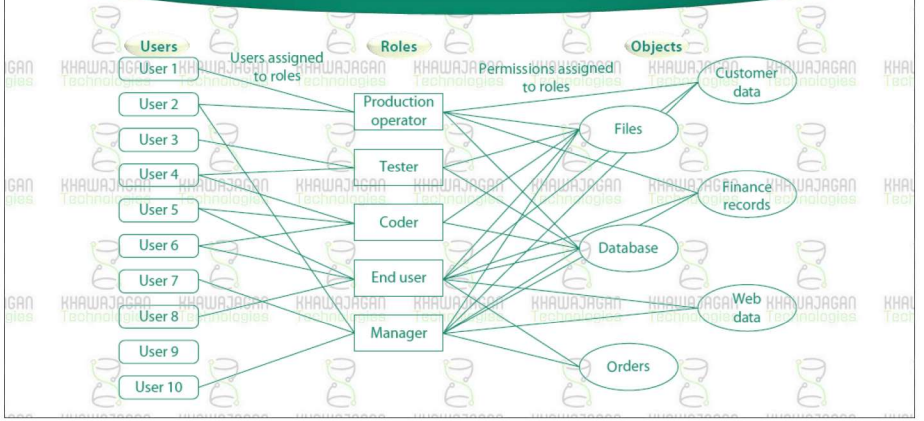
ACM – DAC Model



ACM – RBAC Model

- Access control lists can become long, cumbersome, and take time to administer properly in case of MAC and DAC.
- Role-based access control (RBAC) is the access control mechanism that addresses the length and cost of ACLs.
- In RBAC, instead of assigning specific access permissions for the objects associated with the computer system or network, users are assigned to a set of roles that they may perform.
- Roles would be developer, tester, production, manager, and executive. E.g. a user could be a developer and be in a single role or could be a manager over testers and be in two roles.
- The roles are assigned the access permissions necessary to perform the tasks associated with them. Users will made part of the roles (role members) to get the required permission indirectly.
- An auditor, for instance, can be assigned read access only—allowing audits but preventing change.
- When the number of users grows and permissions must be applied to thousands of objects, roles reduce the number of assignments required. In a firm with hundreds of employees and hundreds of thousands of objects, the number of assignments can approach the number of users times the number of objects, or more. Roles can reduce this by many orders of magnitude.

ACM – RBAC Model



ACM – RuBAC Model

- Rule-based access control systems (RuBAC) are much less common than role-based access control, but they serve a niche.
- In rule-based access control, we utilize elements such as access control lists to help determine whether access should be granted. In this case, a series of rules is contained in the access control list, and the determination of whether to grant access will be made based on these rules.
- An example of such a rule might be a rule that states that nonmanagement employees may not have access to the payroll file after hours or on weekends.
- Rule-based access control can actually be used in addition to, or as a method of, implementing other access control methods.
- For example, role-based access control may be used to limit access to files based on job assignment, and rule-based controls may be added to control time-of-day or network restrictions.

ACM – ACM Model

- The access control matrix model ACMM is a simplified form of access control notation where the allowed actions a subject is permitted with an object are listed in a matrix format.
- This is a general-purpose model, with no constraints on its formulation.
- The strength in this model is its simplicity in design, but this also leads to its major weakness: difficulty in implementation.
- Because it has no constraints, it can be difficult to implement in practice and does not scale well.
- As the number of subjects and objects increases, the intersections increase as the product of the two enumerations, leading to large numbersof ACL entries.

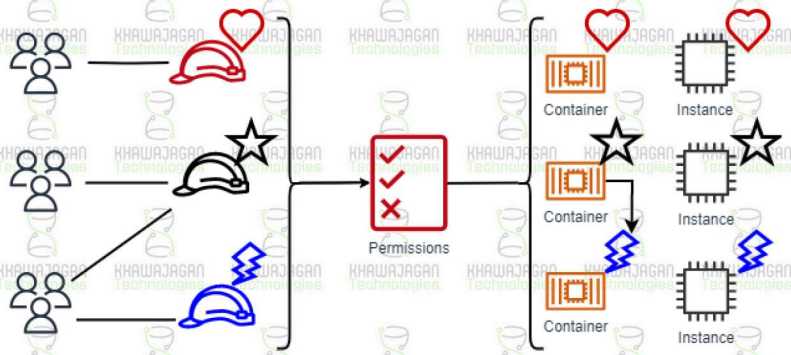
ACM – ACM Model

	File 1	File 2	File 3	File 4
User 1	Read	Write	Own	—
User 2	Write	Own	—	—
User 3	Own	—	—	Read
User 4	Read	Read	Read	Own

ACM – ABAC Model

- Attribute-based access control (ABAC) is a new access control schema based on the use of attributes associated with an identity.
- These can use any type of attributes (user attributes, resource attributes, environment attributes, and so on), such as location, time, activity being requested, and user credentials.
- An example would be a doctor getting one set of access for a specific patient versus a different patient. ABAC can be represented via the eXtensible Access Control Markup Language (XACML), a standard that implements attribute- and policy-based access control schemes.

ACM – ABAC Model



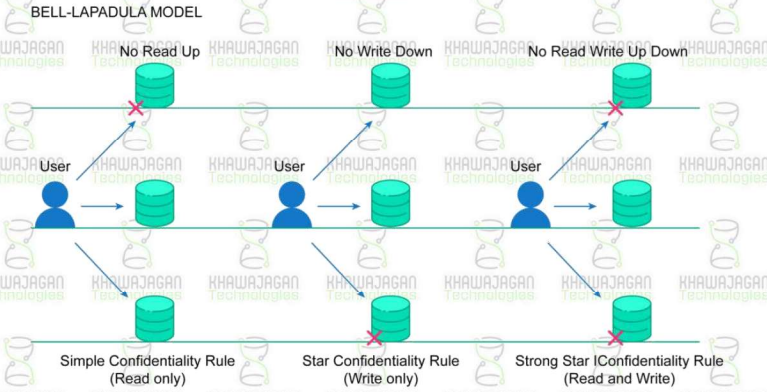
ACM – BLC Model

- The Bell-LaPadula Confidentiality model is a confidentiality preserving model.
- The Bell-LaPadula security model employs both mandatory and discretionary access control mechanisms when implementing its two basic security principles.
- The first of these principles is called the Simple Security Rule, which states that no subject can read information from an object with a security classification higher than that possessed by the subject itself.
- This rule is also referred to as the no-read-up rule.
- This means that the system must have its access levels arranged in hierarchal form, with defined higher and lower levels of access.
- Because the Bell-LaPadula model was designed to preserve confidentiality, it is focused on read and write access. Reading material higher than a subject's level is a form of unauthorized access.

ACM – BLC Model

- The second security principle enforced by the Bell-LaPadula security model is known as the *-property (pronounced "star property").
- This principle states that a subject can write to an object only if its security classification is less than or equal to the object's security classification.
- This is also known as the no-write-down principle.
- This prevents the dissemination of information to users who do not have the appropriate level of access.
- This can be used to prevent data leakage, such as the publishing of bank balances, presumably protected information, to a public webpage.

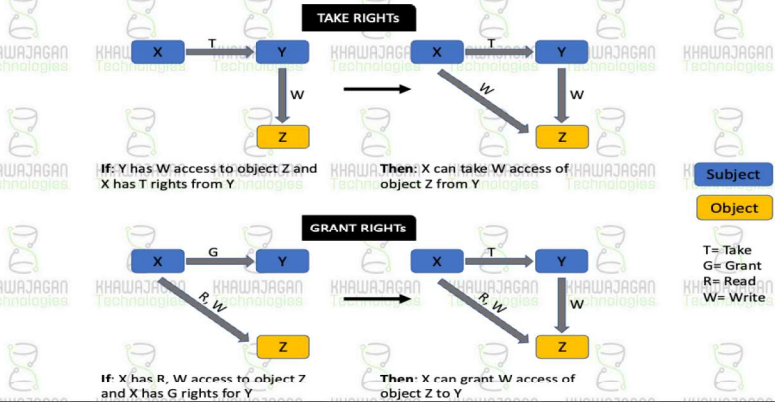
ACM – BLC Model



ACM – Take-Grant Model

- The take-grant model for access control is built upon graph theory. This can be used to definitively determine rights.
- This model is a theoretical model based on mathematical representation of the controls in the form of a directed graph, with the vertices being the subjects and objects. The edges between them represent the rights between the subject and objects.
- There are two unique rights to this model: take and grant.
- The representation of the rights takes the form of {t, g, r, w}, where t is the take right, g is the grant right, r is the read right, and w is the write right.
- A set of four rules, one each for take, grant, create, and remove, forms part of the algebra associated with this mathematical model.
- The take-grant model is not typically used in the implementation of a particular access control system.
- Its value lies in its ability to analyze an implementation and answer questions concerning whether a specific implementation is complete or might be capable of leaking information.

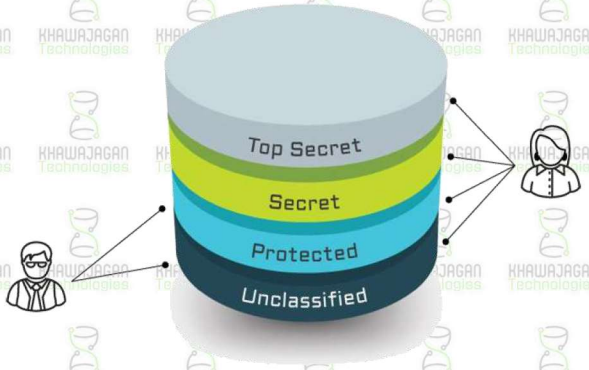
ACM – Take-Grant Model



ACM – Multilevel Security Model

- The multilevel security model is a descriptive model of security where separate groups are given labels to act as containers, keeping information and processes separated based on the labels.
- These can be hierarchical, in which some containers can be considered to be superior to or include lower containers.
- Example : Military classification scheme: Top Secret, Secret, and Confidential. A document can contain any set, but the "container" assumes the label of the highest item contained.
- If a document contains any Top Secret information, then the entire document assumes the Top Secret level of protection.
- Individual items in the document are marked with their applicable level, so that if information is "taken" from the document, the correct level can be chosen.
- Additional levels can be added to the system, e.g. NoForn, to restricts distribution to foreigners.
- There is also the use of keywords associated with Top Secret so that specific materials can be separated and not stored together.

ACM – Multilevel Security Model



SP 2025

Integrity Models

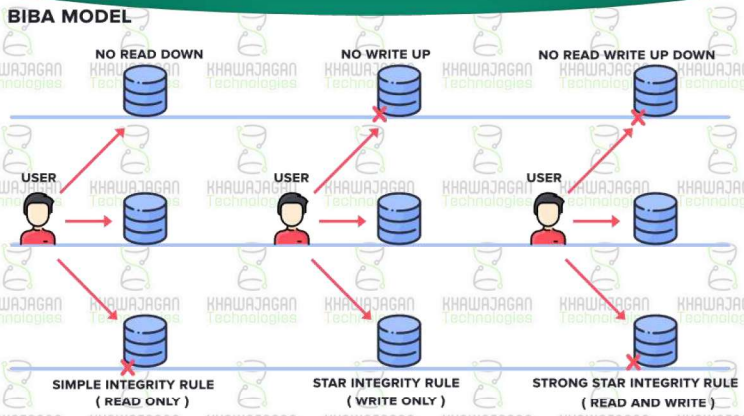
Integrity Models - InM

- Integrity-based models are designed to protect the integrity of the information.
- For some types of information, integrity can be as important as, or even more important than, confidentiality.
- Public information, such as stock prices, is available to all, but the correctness of their value is crucial, leading to the need to ensure integrity.

InM - Biba Integrity Model

- In the Biba model, integrity levels are used to separate permissions.
- The principle behind integrity levels is that data with a higher integrity level is believed to be more accurate or reliable than data with a lower integrity level.
- Integrity levels indicate the level of "trust" that can be placed in the accuracy of information based on the level specified.
- The Biba model employs two rules to manage integrity efforts.
  - The first rule is referred to as the low-water-mark policy, or no-write-up rule. This policy in many ways is the opposite of the *-property from the Bell-LaPadula model, in that it prevents subjects from writing to objects of a higher integrity level.
  - The Biba model's second rule states that the integrity level of a subject will be lowered if it acts on an object of a lower integrity level. The reason for this is that if the subject then uses data from that object, the highest the integrity level can be for a new object created from it is the same level of integrity of the original object.
- In other words, the level of trust you can place in data formed from data at a specific integrity level cannot be higher than the level of trust you have in the subject creating the new data object, and the level of trust you have in the subject can be only as high as the level of trust you had in the original data.

InM - Biba Integrity Model



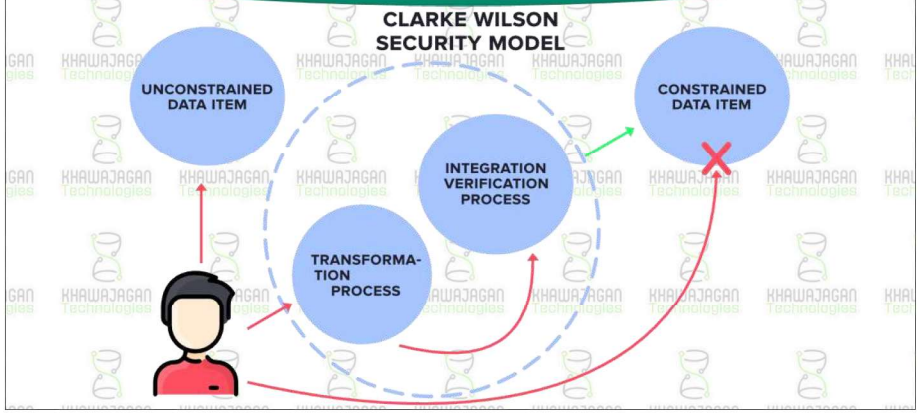
InM - Clark-Wilson Model

- The Clark-Wilson security model use transactions as the basis for its rules.
- It has two levels of integrity:
  - constrained data items (CDIs)
  - Unconstrained data items (UDIs).
  - CDI data is subject to integrity controls, while UDI data is not.
- The model defines two types of processes:
  - Integrity verification processes (IVPs)
  - Transformation processes (TPs)

InM - Clark-Wilson Model

- IVPs ensure that CDI data meets integrity constraints (to ensure the system is in a valid state).
- TPs are processes that change the state of data from one valid state to another.
- Data in this model cannot be modified directly by a user; it can only be changed by trusted TPs.
- In banking example, an object with a need for integrity would be an account balance.
  - In the Clark-Wilson model, the account balance would be a CDI because its integrity is a critical function for the bank.
  - Due to its extreme importance, changes to a person's balance must be accomplished through the use of a TP.
  - Ensuring that the balance is correct would be the duty of an IVP. Only certain employees of the bank should have the ability to modify an individual's account, which can be controlled by limiting the number of individuals who have the authority to execute TPs that result in account modification.

InM - Clark-Wilson Model



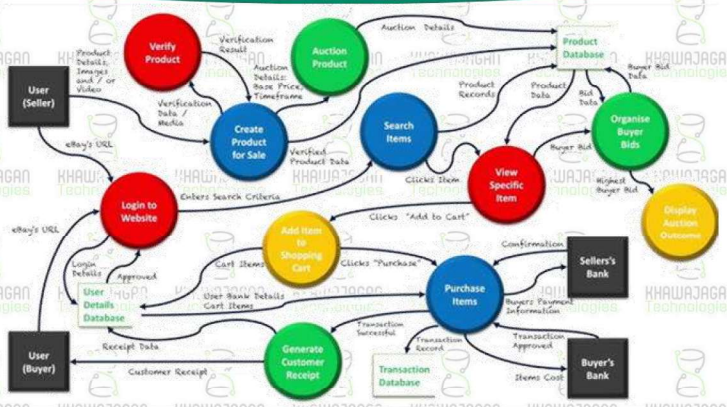
SP 2025

Information Flow Models

Information Flow Models - IFM

- Another methodology in modeling security is built around the notion of information flows.
- Information in a system must be protected when at rest, in transit, and in use.
- Understanding how information flows through a system, the components that act upon it, and how it enters and leaves a system provides critical data on the necessary protection mechanisms.
- A series of models that explores aspects of data or information flow in a system assist in the understanding of the application of appropriate protection mechanisms.

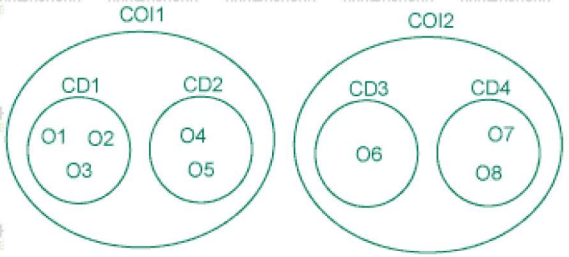
Information Flow Models - IFM



IFM - Brewer-Nash (Chinese Wall)

- The Brewer-Nash model is designed to enforce confidentiality in commercial enterprise operations.
- In a commercial enterprise, different aspects of a business may have access to elements of information that cannot be shared with the other aspects. E.g. In a financial consulting firm, personnel in the research arm may become privy to information that would be considered "insider information." This information cannot, by law or ethically, be shared with other customers.
- Security is characterized by elements involving technology, people, and processes. The Brewer-Nash model is a model where elements associated with all three can be easily understood.
- Technology can be employed to prevent access to data by conflicting groups.
- People can be trained not to compromise the separation of information.
- Policies can be put in place to ensure that the technology and the actions of personnel are properly engaged to prevent compromise.
- Employing actions in all three domains provides a comprehensive implementation of a security model.

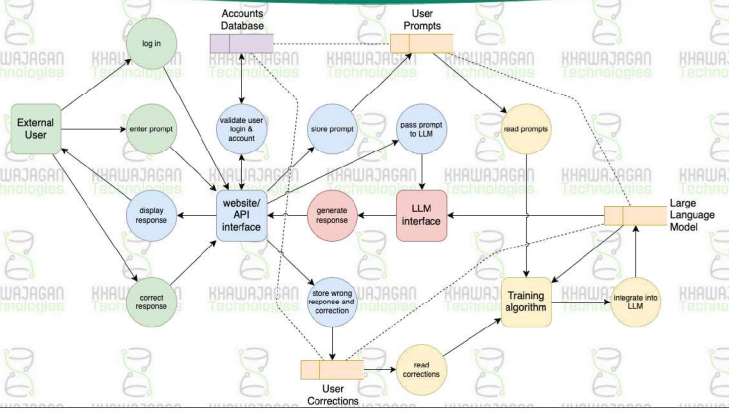
IFM - Brewer-Nash (Chinese Wall)



IFM - Data Flow Diagrams

- The primary issue in security is the protection of information when stored, while in transit, and while being processed.
- Understanding how data moves through a system is essential in designing and implementing the security measures to ensure appropriate security functionality.
- Data flow diagrams (DFDs) are specifically designed to document the storage, movement, and processing of data in a system.
- Data flow diagrams are graphical in nature and are constructed on a series of levels.
  - The highest level, level 0, is a high-level contextual view of the data flow through the system.
  - The next level, level 1, is created by expanding elements of the level 0 diagram.
  - This level can be exploded further to a level 2 diagram, or lowest-level diagram of a system.

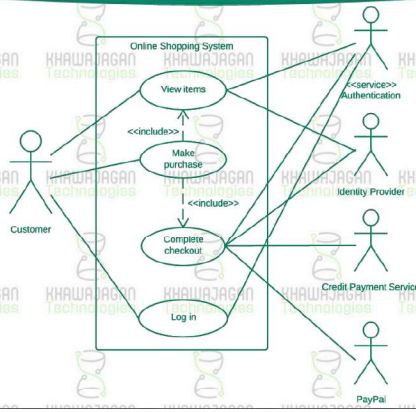
IFM - Data Flow Diagrams



IFM – Use Case Model

- While a DFD examines a system from the information flow perspective, the use case model examines the system from a functional perspective model.
- Requirements from the behavioral perspective provide a description of how the system utilizes data.
- Use cases are constructed to demonstrate how the system processes data for each of its defined functions.
- Use cases can be constructed for both normal and abnormal (misuse cases) to facilitate the full description of how the system operates.
- Use case modeling is a well-defined and mainstream method for system description and analysis.
- Combined with DFDs, use cases provide a comprehensive overview of how a system uses and manipulates data, facilitating a complete understanding of the security aspects of a system.

IFM – Use Case Model



SP 2025

Assurance Models

Assurance Model

- The Committee on National Security Systems has defined software assurance as the "**level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at any time during its lifecycle, and that the software functions in the intended manner.**"
- The current software development methodology employed by many teams is focused on speed to market and functionality as more functions and new versions can drive sales.
- This focus gives lesser position for security, with patching methodology for cybersecurity issues.
- This has proven less satisfactory for many types of critical programs, and the government has led the effort to push for an assurance-based model of development.
- Assurance cases, including misuse and abuse cases, are designed and used to construct a structured set of arguments and a corresponding body of evidence to satisfactorily demonstrate specific claims with respect to its security properties.
- An assurance case is structured like a legal case. An overall objective is defined.
- Specific elements of evidence are presented that demonstrate conclusively a boundary of outcomes that eliminates undesirable ones and preserves the desired ones. When sufficient evidence is presented to eliminate all undesired states or outcomes, the system is considered to be assured of the claim.

Operational Model of Security

- There are three primary actionable methods of managing security in production:
  - Prevention
  - Detection
  - Response
- The operational security model encompasses these three elements in a simple form for management efforts.
- Most effort is typically put on prevention efforts, because incidents that are prevented are eliminated from further concern.
- For the issues that are not prevented, the next step is detection.
- Some issues may escape prevention efforts, and if they escape detection efforts, then they can occur without any intervention on the part of security functions.
- Elements that are detected still need response efforts.

Operational Model of Security

Protection = Prevention + (Detection + Response)

Prevention	Detection	Response
Access controls	Audit logs	Backups
Firewalls	Intrusion detection systems	Incident response teams
Encryption	Honeypots	Computer forensics

The NIST CSF Model

- In 2024, the National Institute of Standards and Technology (NIST) released the latest version of its Cybersecurity Framework (CSF).
- This is a voluntary framework, consisting of standards, guidelines, and best practices, and is designed to manage cybersecurity-related risk in an organization.
- The CSF uses five main functions to categorize activities:
  - Identify, Protect, Detect, Respond, Recover.
- The CSF is being adopted as a framework for information security activities at many firms and thus is an important foundational element in understanding how security is actually implemented.
- Understanding these concepts and terms can assist in cases where the project under development will be used in a CSF environment.

The NIST CSF Model

- In 2024, the National Institute of Standards and Technology (NIST) released the latest version of its Cybersecurity Framework (CSF).
- This is a voluntary framework, consisting of standards, guidelines, and best practices, and is designed to manage cybersecurity-related risk in an organization.
- The CSF uses five main functions to categorize activities:
  - Identify, Protect, Detect, Respond, Recover.
- The CSF is being adopted as a framework for information security activities at many firms and thus is an important foundational element in understanding how security is actually implemented.
- Understanding these concepts and terms can assist in cases where the project under development will be used in a CSF environment.

SP 2025

Adversaries

Adversaries

- Security is the protection of elements from an adversary.
- Adversaries come in many shapes and sizes, with widely varying motives and capabilities.
- The destructive capability of an adversary depends upon many factors, including efforts to protect an element from damage.
- One of the most damaging adversaries is nature.
- Natural disasters range from narrow damage associated with storms, such as tornados, to large-scale issues from hurricanes and ice storms and the resulting outages in the form of power and network outages.
- The "saving grace" with natural disasters is the lack of motive or specific intent and hence the nonadaptability of an attack.
- Other classifications of adversaries are built around capabilities, specifically in the form of their adaptability and capability to achieve their objective despite security controls in place.

Adversary Types

- Adversaries can be categorized by their skill level and assigned following types
  - Script Kiddies
  - Hackers
  - Elite Hackers
- This classification is useful when examining the levels of defenses needed and to what degree they can be effective.
- As the skill level of adversaries increases, the numbers in each category decline.
- While little to no specific training is required to practice at the level of a script kiddie, years of dedicated study are required to obtain, and even more to retain, the rank of an elite hacker.

Adversary Groups

- Adversaries can be analyzed from the perspective of adversary groups, which, based on skill and capability, provides a structure to measure the effectiveness of defenses.
  - Unstructured Threat
  - Structured Threat
  - Highly Structured Thread
  - Nation – State Threat
  - Insider / Outsider Threat
- The least capable form is an unstructured threat, a single actor from the outside.
- A highly structured threat, or nation-state attack, has significantly greater capability.
- The delineation of an attacker as either an insider or an outsider can also play a role in determining capability.
- An insider has an advantage in that they already have some form of legitimate access that can be exploited. This, coupled with their internal knowledge of systems and the value of data and its location, places insiders ahead of the pack on essential knowledge in the prosecution of an attack.

SP 2025

Threat Landscape Shift

Threat Landscape Shift

- The threat landscape in past, been defined by the type of actor in the attack.
- From individual hackers whose motivation is to explore how a system works to the hacktivist groups that have some message to proclaim to nation-states whose motivation is espionage based, the group defined the purpose and the activity to a great degree.
- Around 2005, a shift occurred in the attack landscape: the criminalization of cyberspace.
- This results in criminal groups developing methods of monetizing their efforts which led to an explosion in attack methods, and a market for exploits associated with vulnerabilities emerges.
- This shift has driven actual research into the art of the attack, with criminal organizations funding attacks for the purpose of financial gain.
- The botnets being used today are sophisticated decentralized programs, using cryptographic and antivirus detection methods, and are designed to avoid detection.
- These botnets are used to gather credentials associated with online financial systems, building profiles of users including bank access details, stock account details, and other sensitive information.

Threat Landscape Shift

- This shift has a practical effect on all software—everything is now a target.
- In the past, unless one was a bank, large financial institution, or government agency, security was considered not as important as "who would attack me?"
- This dynamic has changed, as criminals are now targeting smaller businesses, and even individuals, as the crimes are nearly impossible to investigate and prosecute, and just as people used to think they could hide in all the bits on the Internet, the criminals are doing just that.
- And with millions of PCs under botnet control, their nets are wide, and even small collections from large numbers of users are profitable.
- The latest shift in direction is toward targeting individuals, making specific detection even more difficult.

KHAWAJAGAN Technologies

SP 2025

Define Software Security Requirements

Software Security Requirements

- Requirements are the blueprint by which software is designed, built, and tested.
- As one of the important foundational elements, it is critical to manage this portion of the software development lifecycle (SDLC) process properly.
- Requirements set the expectations for what is being built and how it is expected to operate.
- Developing and understanding the requirements early in the SDLC process are important because if one has to go back and add new requirements later in the process, it can cause significant issues, including rework, delays, and increased cost.

Functional Requirements

- Functional requirements describe how the software is expected to function.
- They begin as business requirements and can come from several different places.
- The line of business that is going to use the software has some business functionality it wants to achieve with the new software. These business requirements are translated into functional requirements.
- The IT operations group may have standard requirements, such as deployment platform requirements, database requirements, disaster recovery/business continuity planning (DR/BCP) requirements, infrastructure requirements, and more.
- The organization may have its own coding requirements in terms of good programming and maintainability standards.
- Security may have its own set of requirements.
- In the end, all of these business requirements must be translated into functional requirements that can be followed by designers, coders, testers, and more to ensure they are met as part of the SDLC process.

Role and User Definitions

- Role and user definitions are the statements of who will be using what functionality of the software.
- At a high level, these will be in generic form, such as which groups of users are allowed to use the system.
- Subsequent refinements will detail specifics, such as which users are allowed which functionality as part of their job.
- The detailed listing of what users are involved in a system form part of the use-case definition.
- In computer science terms, users are referred to as subjects.
- This term is important to understand the subject-object-activity matrix.

Objects

- Objects are items that users (subjects) interact with in the operation of a system.
- An object can be a file, a database record, a system, or a program element.
- Anything that can be accessed is an object.
- One method of controlling access is through the use of access control lists assigned to objects.
- As with subjects, objects form an important part of the subject-object-activity matrix.
- Specifically defining the objects and their function in a system is an important part of the SDLC.
- This ensures all members of the development team can properly use a common set of objects and control the interactions appropriately.

Activities / Actions

- Activities or actions are the permitted events that a subject can perform on an associated object.
- The specific set of activities is defined by the object.
  - A database record can be created, read, updated, or deleted.
  - A file can be accessed, modified, deleted, etc.
- For each object in the system, all possible activities/actions should be defined and documented.
- Undocumented functionality has been the downfall of many a system when a user found an activity that was not considered during design and construction, but still occurred, allowing functionality outside of the design parameters.

Subject-Object-Activity Matrix

- Subjects represent the who, objects represent the what, and activities or actions represent the how of the subject-object-activity relationship.
- Understanding the activities that are permitted or denied in each subject-object combination is an important requirements exercise.
- To assist designers and developers in correctly defining these relationships, a matrix referred to as the subject-object-activity matrix is employed.
- For each subject, all of the objects are listed, along with the activities for each object.
- For each combination, the security requirement of the state is then defined.
- This results in a master list of allowable actions and another master list of denied actions.
- These lists are useful in creating appropriate use and misuse cases, respectively.
- The subject-object-activity matrix is a tool that permits concise communication about allowed system interactions.

Subject-Object-Activity Matrix

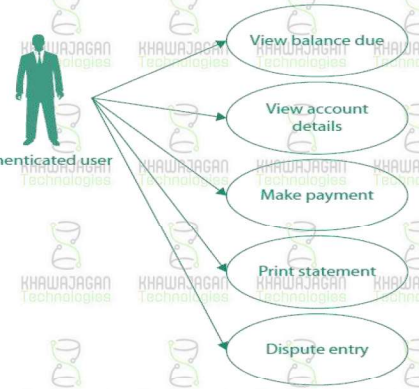
ACCESS MATRIX

Object / Domain	File 1	File 2	File 3	File 4
D1	*read write*	*read*	*read*	*read*
D2	*read*	*write*	-	*exceute*
D3	*write*	-	*write*	*exceute*

Use Cases

- Use cases are a powerful technique for determining functional requirements in developer friendly terms.
- A use case is a specific example of an intended behavior of the system.
- Defining use cases allows a mechanism by which the intended behavior (functional requirement) of a system can be defined for both developers and testers.
- Use cases are not intended for all subject-object interactions, as the documentation requirement would exceed the utility.
- Use cases are not a substitute for documenting the specific requirements.
- Where use cases are helpful is in the description of complex or confusing or ambiguous situations associated with user interactions with the system.
- This facilitates the correct design of both the software and the test apparatus to cover what would otherwise be incomplete due to poorly articulated requirements.

Use Cases – Example



Sequencing and Timing

- In today's multithreaded, concurrent operating model, it is possible for different systems to attempt to interact with the same object at the same time.
- It is also possible for events to occur out of sequence based on timing differences between different threads of a program.
- Sequence and timing issues such as race conditions and infinite loops influence both design and implementation of data activities.
- Understanding how and where these conditions can occur is important to members of the development team.
- In technical terms, what develops is known as a race condition, or from the attack point of view, a system is vulnerable to a time of check/time of use (TOC/TOU) attack.

Race Conditions

- Race conditions are software flaws that arise from different threads or processes with a dependence on an object or resource that affects another thread or process.
- These conditions can be difficult to predict and find.
- Example : one thread depends on a value (A) from another function that is actively being changed by a separate process. The first process cannot complete its work until the second process changes the value of A. If the second function is waiting for the first function to finish, a lock is created by the two processes and their interdependence.
- Multiple unsynchronized threads, sometimes across multiple systems, create complex logic loops for seemingly simple atomic functions.
- Understanding and managing record locks is an essential element in a modern, diverse object programming environment
- Race conditions are defined by race windows, a period of opportunity when concurrent threads can compete in attempting to alter the same object. The first step to avoid race conditions is to identify the race windows.
- Once the windows are identified, the system can be designed so that they are not called concurrently, a process known as mutual exclusion.

Infinite Loop

- Another timing issue is the infinite loop.
- When program logic becomes complex—for instance, date processing for leap years—care should be taken to ensure that all conditions are covered and that error and other loop-breaking mechanisms do not allow the program to enter a state where the loop controls will fail.
- Failure to manage this exact property resulted in Microsoft Zune devices failing if they were turned on across the New Year following a leap year.
- The control logic entered a sequence where a loop would not be satisfied, resulting in the device crashing by entering an infinite loop and becoming nonresponsive.

SP 2025

Secure Coding Standards

Secure Coding Standards

- Secure coding standards are language-specific rules and best practices for secure programming.
- Application programming can be considered a form of manufacturing which turns requirements into value-added product at the end of a series of business processes.
- Controlling these processes and making them repeatable is one of the objectives of a secure development lifecycle through adoption of an enterprise-specific set of secure coding standards.
- Because secure coding guidelines have been published for most common languages, adoption of these practices is an important part of secure coding standards in an enterprise.

Secure Coding Standards

- Adapting and adopting industry best practices are also important elements in the secure development lifecycle.
- One common problem in many programs results from poor error trapping and handling. This is a problem that can benefit from an enterprise rule where all exceptions and errors are trapped by the generating function and then handled in such a manner so as not to divulge internal information to external users.
- Logging is another area that can benefit from secure coding standards.
- Standards can be deployed specifying what, where, and when issues should be logged which ensures appropriate levels of logging, simple management of the logging infrastructure.

Operational and Deployment Reqs.

- Enterprises have standards for technology deployment, specifying platforms, operating systems (Linux, Microsoft Windows), specific types and versions of database servers, web servers, and other infrastructure components.
- These technology choices are part of the enterprise overall strategy, and the overall collection is optimal for the enterprise working.
- Software in the enterprise works by connecting to other pieces of software in the technology landscape.
- A new system may provide new functionality, but would do so touching existing systems, such as connections to users, parts databases, customer records, etc.
- One set of operational requirements is built around the idea that a new or expanded system must interact with the existing systems over existing channels and protocols.
- Ensuring that systems are secure by design is commonly seen as the focus of an SDLC, but it is also important to ensure systems are secure when deployed.
- This includes elements such as secure by default and secure when deployed.