CHAPTER - 5
SYSTEM ANALYSIS FOR
SECURITY MODEL FOR OBJECT ORIENTED DESIGN
(SMOOD)

Software security is a multifaceted and flexible concept. Every facet of this concept highlights a discrete external software characteristic in itself. It is obligatory to comprehend that no software unerringly demonstrates attributes of security while designing a software model. In its place, it manifests features of the product that add up to security traits or further traits (product flaws) that detract from the product’s security features. Majority of the software security models fail while trading with the problems occurring in case of the product features. They also fall short in undeniably linking security attributes and their equivalent product features. Profuse knowledge about software security is available in the literature. The greatest challenge in suggesting any model for software product security is to find a framework that can lodge this knowledge in a refine able, constructive, and intellectually manageable way. The foremost requirement of any such model is that it makes lucid and direct links between explicit product features at all levels and high level security attributes.

Generally, the external feature or external attributes were not straight forwardly related to any characteristic of the software product (Dromey R G, 1995). Nevertheless, to improvise the software product, we should be capable of affecting its internal traits. For this, it was essential to recognize some internal features, which effect the software security attributes, either in a direct or indirect fashion. As described before, there exist two types of security attributes: internal and external. Software attributes whose existence or non-existence may be realized by stakeholders like customers or users can be easily referred to by external attributes. Instances of such qualities comprise of confidentiality, integrity, availability, authorization and so forth. Internal attributes match up to the software features which were hidden and significant, specifically to software experts (e.g. developers, specifies, testers), who have the privilege to work products of software such as code, documentation, or specification. Cohesion, coupling and complexity were among the other instances of internal software attributes. At last, the main facts from the customer’s or the user’s outlook includes the external qualities for which they possess a lucid perspective.
Still, the external attributes can only be attained by intelligently employing schemes which are internal to software edifice assuring the internal features. To improvise a software product, it is required to influence its internal attributes. Pointless to mention, it is easy to collect internal metrics but hard to interpret them where as in the case of external metrics, it is vice-versa. Prediction models which were based, for instance, on Bayesian probability or regression analysis permit the mapping of hard to interpret internal measurement data into external measurement data which could simply comprehensible. The analysis of these models was done through empirical exploration. From the previous analysis made by different researchers, we could suggest some strategies that can be employed to study experimentally the association flanked by the external and internal notions of security measurement. 

In the Security Model for Object Oriented Design (SMOOD), different levels were described. In the first level, the security attributes and objects oriented design properties have been defined. After this it is necessary to deduce which security design principles were vital and which high-level security attributes they influence, for each component. Thus, it becomes easy for us to identify for every high level security attribute which security design principles entail that attribute and which product entities have those specific security-carrying properties. By employing this scheme, it could possible to define the scope of the task of building each high-level security attribute into software product.

5.1 Data Flow Diagram for SMOOD:

A data flow diagram is a graphical representation of information flow and transforms that are applied as data moves from input to output. The basic form of data flow diagram is also known as data flow graph or bubble chart. Data Flow Diagrams (DFD) serves following two purposes:

- To provide an indication of how data are transformed as they move through the system.
- To depict the functions that transforms the data flow.

The DFD provides additional information that is used during the analysis of the information domain and serves as a basis for the modelling of function. A description for each function presented in the DFD is contained in a process specification. As information moves through software, it is modified by a series of transformations.
The data flow diagram may be used to represent a system or software at any level of abstraction. In fact, DFDs may be partitioned into levels that represent increasing information flow and functional detail. Therefore, the DFD provides a mechanism for functional modelling as well as information flow modelling. The typical schematic representations of DFD at different level (0 to 2) were depicted in fig. 5.1 to 5.3.

**Fig No. 5.1: Data Flow Diagram (Level 0)**

**Fig No. 5.2: Data Flow Diagram (Level 1)**
5.2 Security attributes:

Confidentiality, availability, integrity are the attributes chosen as security attributes in the SMOOD model. This set of attributes then was investigated separately so as to decide their role in the design security and also to keep a check on whether the set was amply wide to consist of all aspects of security design. The contribution of these attributes was elucidated below:

5.2.1 Confidentiality:

Confidentiality guarantees that the information is inaccessible by persons who are unauthorized. It aims to limit the extent of information access and revelation to the users who are authorized, i.e., "the right people" and precluding admission or revelation to the ones who are not authorized, i.e., “the wrong people”. The confidentiality can be violated when data was not dealt with properly to shield the respective information. It is the method of obscuring the resources or information. Confidentiality is sustained by access control mechanisms. For instance,
Measuring the Impact of Software Design on Software Security

5.2.2 Integrity:
By Integrity, one can impart the reliability of resources or data, and it is typically articulated in terms of precluding unacceptable change. Making certain that information is not tainted by unauthorized persons in a way that is not detectable by authorized users. While taking into consideration the Information Security, the term integrity is recurrently used as it symbolizes one of the chief pointer of safety (or lack of it). Only appropriate Information has value. It includes the concept of "data integrity", namely, that data should not inappropriately change, either by accident or by some intentional detrimental activity. Integrity also comprises of "origin" or "source integrity" (here, the source of data, frequently called as authentication), that the data in reality appeared from the person you think it did, rather than a fraud. Integrity mechanisms are categorized into two classes: prevention mechanisms and detection mechanisms. Prevention mechanisms are meant to sustain the data’s integrity by clogging any attempts to modulate the data by unauthorized means. The latter occurs when a user authorized to make certain changes in the data tries to change the data in other ways. Sufficient verification and access controls will normally end the breach-in from the outside, but precluding the second sort of attempt necessitates extremely diverse controls. Detection mechanisms only tell us if the data’s integrity is trustworthy or not. Integrity takes account of both the trustworthiness and the correctness of the data.

5.2.3 Availability:
Availability is a promise of making systems, employed for processing, storing and delivering information, accessible whenever desirable. Availability certifies access to information to authorized parties whenever required. An information system that is not available when it’s required is almost equivalent to as having none at all. Information finds its worth only when the right people can access it at the appropriate
times. Also, the attacks desired against availability are called as denial of service (DoS) attacks. It has become common now-a-days to deny access to information. Almost every week we get to hear the news about high profile websites being taken down by DoS attacks. The chief goal of DoS assaults is to repudiate the access of the website’s resources to its users. Other factors comprise of mishaps like power outrages or natural disasters, for instance floods.

### 5.3 Object-oriented Design Properties:

In an object oriented design, the structure view of the software is described by the class diagram. Class definitions are assessed with other classes for external relationships. By doing evaluation of its internal methods, attributes, and components, we can deduce significant information that neutrally refers to the structural and functional features of class and its object. The internal and external structure, functionality, and relationship of design components, classes, methods, and attributes are concrete concepts that can be precisely evaluated by investigating design properties. The Design property like abstraction, encapsulation, design size, coupling, cohesion, and complexity are used for illustration of design security trait in both structural as well as object-oriented development. The design concepts which have been presented by object oriented model, i.e., messaging, inheritance, and polymorphism, signify new design concept and thus, are necessary to security of object oriented design. So, all the design properties like abstraction, coupling, cohesion, encapsulation, polymorphism, messaging, design size, hierarchies, complexity and inheritance are considered with respect to security.

Inheritance increases the reusability of code and design of the overall software; and thus, it also diminishes the pains and cost endured in software development. But on the other hand, inheritance is also blamed for dispersal of susceptibility from one class to another, due to its transitive nature. For a design that sustains inheritance, all the variables and methods defined for its parent class inevitably become accessible to all the subclasses, as well as for any subclass of the classes which are found down in the hierarchy, recursively.

Coupling is defined as the reliability of an entity on the other. Substantial research has been made to evaluate the influence of coupling on quality. It becomes necessary to
study the upshot of coupling on security since security is an attribute of quality. It has been proved that higher coupling has negative effect on quality as well as security. Cohesion recapitulates the extent of interdependence within a subsystem. A cohesive class, which has a petite and focused set of responsibilities and fanatically employs methods and attributes to execute those responsibilities. It is not easy to split a class into a separate one if it has high cohesion and performs one basic function. The classes those are highly cohesive and more comprehensible, adaptable and maintainable. Hence, it is advised to use a highly cohesive class for high quality and security object oriented design.

Complexity is habitually conjectured to be the foe of software security. Geer affirmed that “complexity provides both opportunity and hiding places for attackers” and “security failures come from it (complexity) as surely as dawn comes from the east” in a hearing at a subcommittee of the Committee on Homeland Security on 23rd, April, 2007.

Abstraction and encapsulation positively affect the security attributes by bounding sharing of methods which results in expanding the probability of breaking integrity, confidentiality, authentication, authorization and availability. It also avoids from heavy data loss which may happen due to inheritance, coupling and cohesion.

Polymorphism measures the capacity to replace object whose interfaces correspond for one another at the run time. It is used to check those services which are resolved at run time in an object, dynamically. But dynamic memory management is the main significant factor that may raise various susceptibilities like those of fragmentation, ambiguous references and starvation etc.

An attempt was made to find the substantial documents and evidences to relate as internal attributes the factors like coupling, cohesion, abstraction, polymorphism, inheritance, encapsulation and complexity to the security in object oriented design. Thus, we have incorporated them in our research work further. But for the attributes like design size, hierarchies, and messaging, no such documentation or research work has been found due to which they have not been considered further in this work.

5.4 Software Security Principles:

Least Privilege: The principle of least privilege specifies, “Do not give more than absolutely necessary privileges to do the required job.” This principle is relevant to
other non-information systems privileges of an organization’s staff in addition to privileges of users and applications on a computer system. The principle of least privilege acts as a defensive control by lessening the number of privileges that may be maltreated abused and consequently restricts the possible damage. The principle of least privilege, like all other good principles, is fitting in all information systems environments. Some instances of application of this principle involve the following:

- Enabling the read access for users so as to allow them to share files if that’s what they are in need of, and ascertaining that the write access is disabled.
- If all that the help-desk staff may have to do is to reset a password, then they should not be allowed to create or delete user accounts.
- By restricting the movement of software from development servers to production servers by software developers.

Integrity protection mechanisms have been grouped into two broad types: preventive mechanisms, such as access controls, and detective mechanisms. Controls that guard integrity comprise of principles of least privilege, separation, and rotation of duties. The relation between the least privilege principle and cohesion has been used to define metrics for cohesion. The chief purpose of this principle is to diminish the communications among privileged programs. To stick to this principle, systems must confine the privileges of their users to the least possible (Bandar Alshammari, 2009). Yanguo Michael Liu studied the principle of least privilege, and came up with the conclusion that security decreases with the increase in service complexity and service coupling and security decreases as excess privilege increases and security increases as mechanism strength increases (Yanguo Michael Liu 2007).

J. Viega, G. McGraw and P. Mandhata, has studied cohesion with respect to the principle of least privilege and concluded that providing privilege to access vulnerable attributes means objects of the other classes have also privilege for the vulnerable attributes through the methods. Hence, cohesion also goes against the principle of providing least privilege. So, it will ultimately decrease the security. (A Agrawal, 2014). So least privilege is related to security external attributes integrity and security internal attributes cohesion, complexity, coupling and inheritance.

**Minimization:** The minimization principle somehow related to the least privilege principle and customarily applies to the process of system configuration. The
minimization principle states “do not run any software, applications, or services that are not strictly required to do the entrusted job.” To demonstrate, in case of a computer whose one and only utility is to serve as an e-mail server ought to have only e-mail server software installed and enabled on it and all other services and protocols should either be disabled or not installed at all to eradicate any leeway of negotiation or squander. Observance to the minimization principle boosts security as well as saves storage space, typically advances performance, and in general forms a good system administration practice.

The minimization principle is the cousin of the least privilege principle and frequently relates to system configuration. The minimization principle says “do not run any software, applications, or services that are not strictly required to do the entrusted job. So minimization was related to security external attributes integrity and security internal attributes cohesion, complexity, coupling and inheritance.

**Compartmentalization:** The principle of compartmentalization or the use of compartments (also known as zones, jails, sandboxes, and virtual areas), limits the harm and guards other compartments when software in one compartment is not working or has negotiated. It can be matched to the compartments on ships and submarines, where a disaster in one compartment does not mean that the entire ship or submarine is lost. Compartmentalization in context to the information security refers that applications running in different compartments are segregated from each other. In a system like this, the negotiation of web server software, for instance, does not note or influence e-mail server software running in a separate compartment on the same system. Zones in Solaris 10 accomplish the compartmentalization principle and are treated as powerful security mechanisms.

J. Viega, G. McGraw and P. Mandhata have studied the impact of cohesion on the principle of compartmentalization and based on their studies, A. Agrawal has stated that compartmentalization in the design is necessary for increasing reusability, simplicity and understandability as well as for enhancing security. And since cohesion is used to increase modularity, it can also be used to increase compartmentalization (A Agrawal 2014). So compartmentalization is related to security internal attributes cohesion and encapsulation.
Kepp Things Simple: Complexity is the most horrible enemy of security. Complex systems are intrinsically extra at risk because they are difficult to design, implement, test, and secure. As the complexity of a system increases, there is less guarantee of the fact that it will function as per expectations. Even though, it is obvious that there will be an increase in the complexity of information systems and processes as our expectations of functionality keep on increasing, we should still be extremely wary to pencil in a line between preventable and inevitable complexity and not relinquish security for bells and whistles, only to be repentant about it later. So, it is always advisable to choose the simple one when you have to pick between a complex system that does much and a simple system that does a bit less but enough.

The extra complex a system, the fewer guarantees this may have function as expected J. Viega, G. McGraw and P. Mandhata have studied the impact of cohesion on the principle of compartmentalization and based on their studies, A. Agrawal has also put forth that the studies on improvement of quality of an object oriented design have proved that high cohesion for classes reduces complexity of the design. Hence, cohesion helps for making an object oriented design simple (A Agrawal, 2014). So keeping thing simple is related to security internal attributes cohesion and complexity.

Fail Securely: Failing securely con notes that the system is not solidified to an insecure state, if a security measure or control has failed, for whatever reason. For instance, when a firewall crashes, it should be set to default to “deny all” rule, and not to “permit all.” Nevertheless, fail securely doesn’t refer to “close everything” in all situations; If we are discussing about a computer-controlled access control system in a building, for illustration, in case of a fire, the system should be by default set to “open doors”, if humans are trapped in the building. In this case, human life takes precedence over the risk of unauthorized access.

Yanguo Michael Liu studied the principle of fail securely and concluded that security decreases as service complexity increases, security decreases as service coupling increases, security decreases as excess privilege increases and security increases as mechanism strength increases (Yanguo Michael Liu, 2007). So fail securely is related to security internal attributes coupling and complexity.
Secure the Weakest Link: Numerous information security principles and techniques are like a bit in excess of common sense. Still, it does not help us much as we still fall short to work with rationality. One such case is the principle of securing the weakest link, where we probably find that resources are spent on strengthening already ample defences instead of safe-guarding the weakest link, whatsoever it may be. Yanguo Michael Liu studied the principle of secure the weakest link, and concluded that security decreases as service complexity increases, security decreases as service coupling increases, security decreases as excess privilege increases and security increases as mechanism strength increases (Yanguo Michael Liu, 2007). So secure weakest link is related to security internal attributes coupling and complexity.

Segregation of Duties: The aim of the segregation (or separation) of duties is to evade the likelihood of a single individual being accountable for several occupations within an organization, explain information security fundamentals and define good security architectures which when blended may result in a security defilement that may go ignored. Segregation of duties can preclude or deter security breaches and should be accomplished when possible. Though the real job labels and organizational pyramids may fluctuate critically, the notion behind and the standard of separation of duties stays the same: no single being ought to be capable to interrupt security and escape with it. Rotation of duties is like control that is envisioned to perceive abuse of privileges or fraud and is an exercise to aid your organization from becoming exaggeratedly reliant on a single member of the staff. By rotating staff, the organization has more probabilities of defining violations or fraud. Integrity protection mechanisms have been grouped into two broad types: preventive mechanisms, such as access controls, and detective mechanisms. Controls that protect integrity include principles of least privilege, separation, and rotation of duties. So segregation of duties is related to security external attributes integrity.

Reduce attack surface: The attack surface of a software environment is measured as a sum of the different points (the "attack vector") where a prohibited or an unauthorized user (the "attacker") can attempt to enter data to or extract data from an environment. The critical strategies of attack surface reduction include reducing entry points available to distrusted users, reducing the amount of code running and
eradicate services demanded by comparatively few users. One way of improving information security is to trim down the attack surface of a system or software. Also, by turning off needless functionalities, there are fewer chances of security risks. If less code is accessible to unauthorized users, there will tend to be fewer collapses. Even if attack surface reduction helps avert security failures, it does not minimize the extent of damage an attacker could impose once susceptibility is discovered.

Kurmus has calculated the attack surface metric and concluded that attackers could target full control with arbitrary code execution in kernel mode, to breach confidentiality, and denial of service by crashing the kernel to reduce the system’s availability (Anil Kurmus and Reinhard Tartler, 2014).

Due to increased inherited methods and classes design complexity will increase, with more faults and attack surface (R A Khan, 2010). Attack surface principle is used to develop security metrics for cohesion and encapsulation. This incurs that a class design ought to have the least probable available methods each with the smallest number of constraints which can mark confidential attributes required for indispensable tasks. A class which has smaller amount of approachability to confidential methods that intermingle with as little confidential attributes as probable would placate the necessities of the security principle of plummeting the attack surface (Bandar Alshammari, 2009).

J. Viega, G. McGraw and P. Mandhata have studied the impact of cohesion on security. A. Agrawal has conclude that when less number of methods will have access to vulnerable attributes, less attack surface will be exposed. And, the less exposed the attack surface, the less propagation of vulnerability throughout the design. Thus, reduced attack surface decreases the propagation of vulnerabilities through cohesion (A Agrawal, 2014). So segregation of duties is related to security external attributes confidentiality, availability and security internal attributes inheritance, cohesion and encapsulation.

5.5 Discussion:

Security can be correctly built inside software by integrating it all through the entire software development life cycle. For integrating the security it is very necessary to analyze the various aspects related to it. For instance, calculating the impact of design properties on security attributes: confidentiality, integrity and availability, it is
necessary to evaluate the security principals. A security principal has been studied in connection with security attributes and design properties. Hence, we have collaborated various different data related to security principals. This focuses on gathering information about security properties and design properties impact on security. The goal of this study is to come across with all the facets affecting software security and which can be useful in providing proper conclusions. After gathering all the different reviews we were able to get the collection of data which when analyzed further used in development of security model for object oriented design (SMOOD).