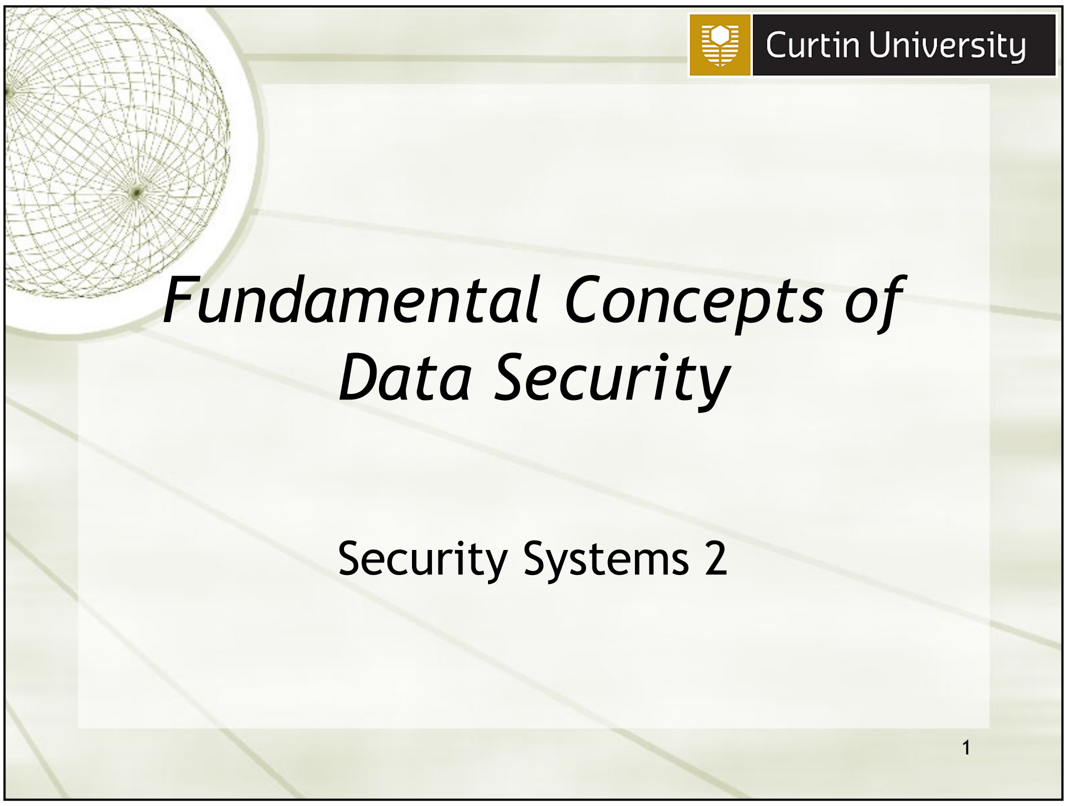
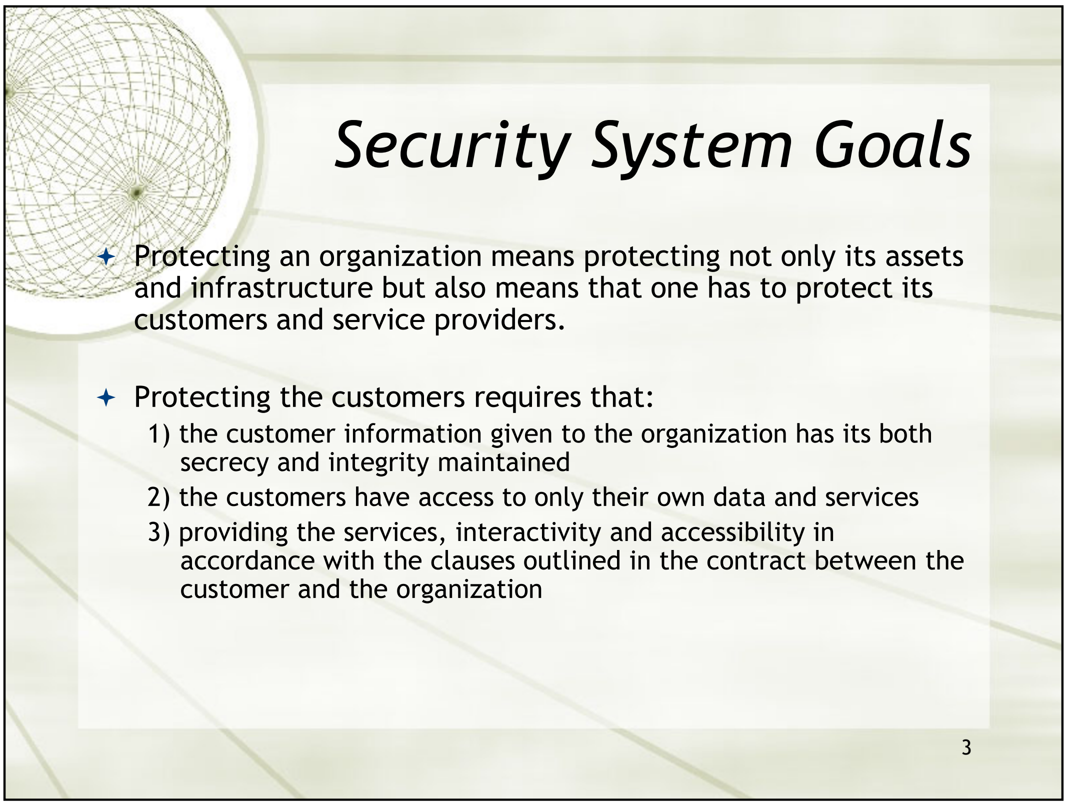


Curtin University

Fundamental Concepts of Data Security

Security Systems 2

Note: This is expected to be delivered over three lectures/workshops



Security System Goals

- Protecting an organization means protecting not only its assets and infrastructure but also means that one has to protect its customers and service providers.
- Protecting the customers requires that:
  1) the customer information given to the organization has its both secrecy and integrity maintained
  2) the customers have access to only their own data and services
  3) providing the services, interactivity and accessibility in accordance with the clauses outlined in the contract between the customer and the organization

3

SECURITY CONCEPTS AND GOALS

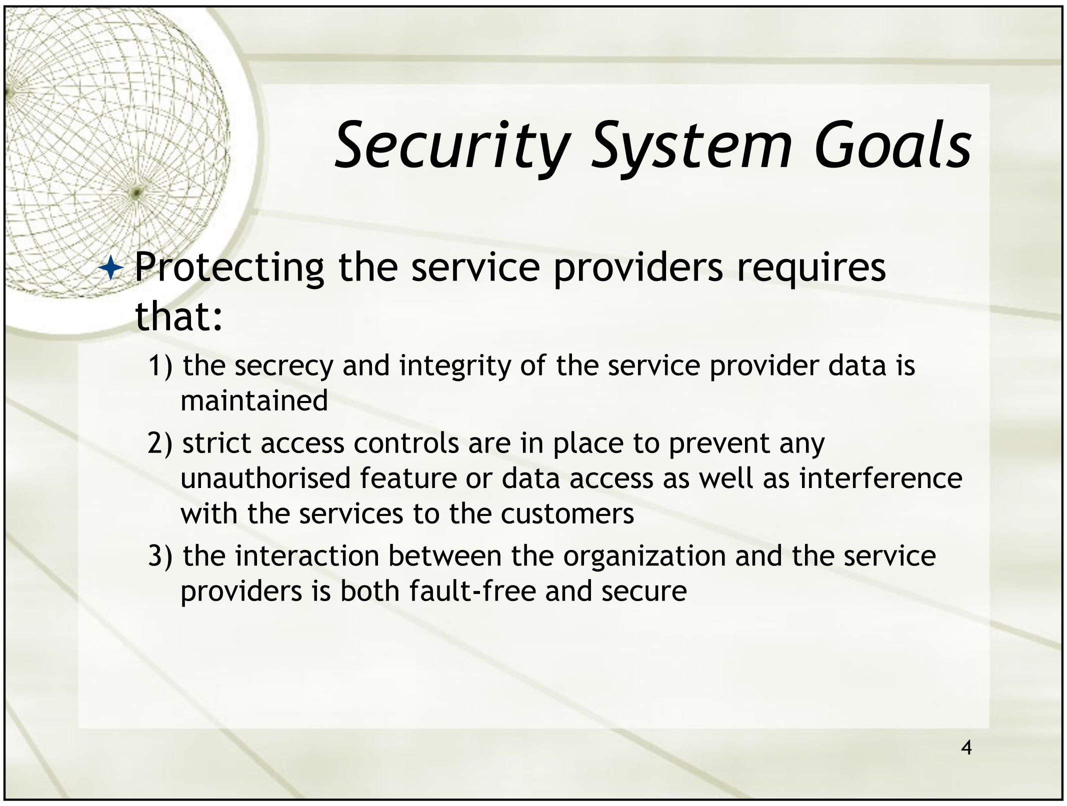
From a security objectives perspective, an organization is responsible for protecting customers, peering service providers, and the infrastructure. Customers should be protected in regard to:

. ensuring confidentiality and integrity of all customer information entrusted to the organization and any information the organization obtains as a result of customer usage of organization services;

. maintaining customer contracted for service availability in compliance with those service level agreements between the customer and the organization;

. enforcing customer access to only authorized features so that the actions of one customer cannot interfere with service availability to, or information about, other customers; and

. ensuring error-free and nonmalicious interaction between customers and the organization infrastructure.



Security System Goals

- Protecting the service providers requires that:
  1) the secrecy and integrity of the service provider data is maintained
  2) strict access controls are in place to prevent any unauthorised feature or data access as well as interference with the services to the customers
  3) the interaction between the organization and the service providers is both fault-free and secure

4

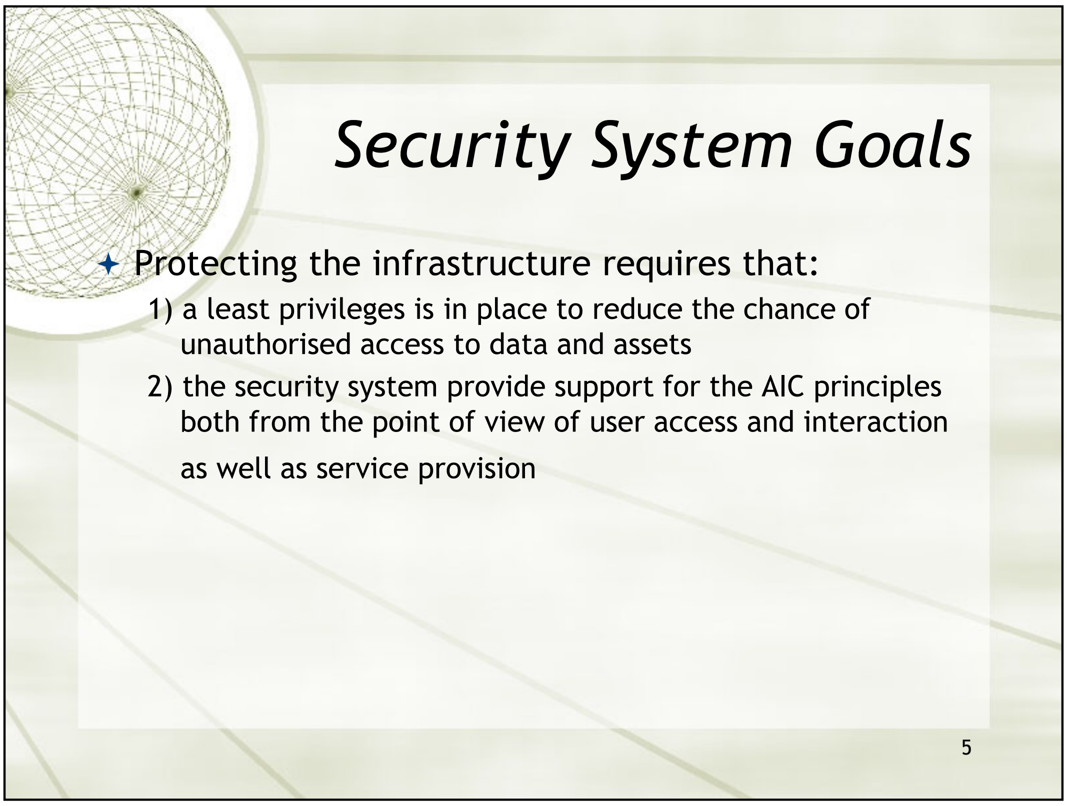
Peering service providers should be protected in regard to:

. ensuring confidentiality and integrity of all peering service provider information, entrusted to the organization, such as routing, subscriber, billing information, and video content;

. maintaining peering service provider contractually obligated availability in compliance with those service level agreements between the peering service provider and the organization;

. enforcing peering service provider access to only authorized features so that their actions cannot interfere with service availability to, or information about, organization customers; and

. ensuring error-free and nonmalicious interaction between peering service providers and the organization infrastructure.



Security System Goals

- Protecting the infrastructure requires that:
  1) a least privileges is in place to reduce the chance of unauthorised access to data and assets
  2) the security system provide support for the AIC principles both from the point of view of user access and interaction as well as service provision

5

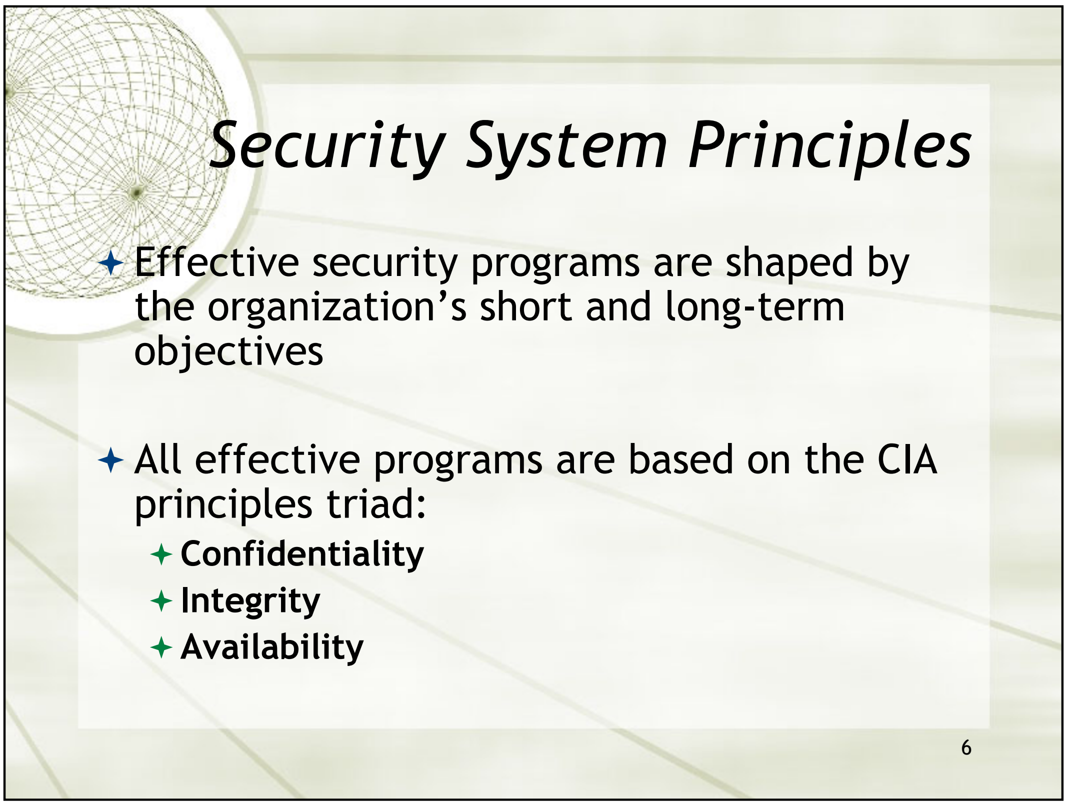
Peering service providers should be protected in regard to:

. ensuring confidentiality and integrity of all peering service provider information, entrusted to the organization, such as routing, subscriber, billing information, and video content;

. maintaining peering service provider contractually obligated availability in compliance with those service level agreements between the peering service provider and the organization;

. enforcing peering service provider access to only authorized features so that their actions cannot interfere with service availability to, or information about, organization customers; and

. ensuring error-free and nonmalicious interaction between peering service providers and the organization infrastructure.



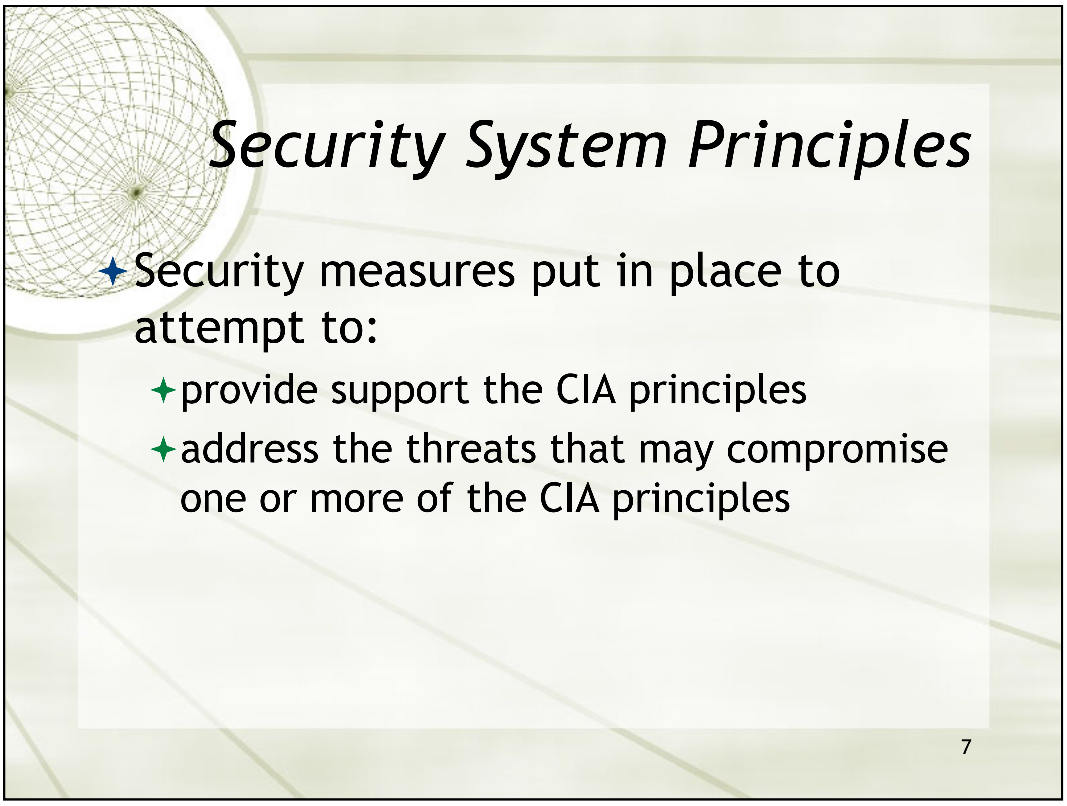
Security System Principles

- Effective security programs are shaped by the organization’s short and long-term objectives
- All effective programs are based on the CIA principles triad:
  - **Confidentiality**
  - **Integrity**
  - **Availability**

6

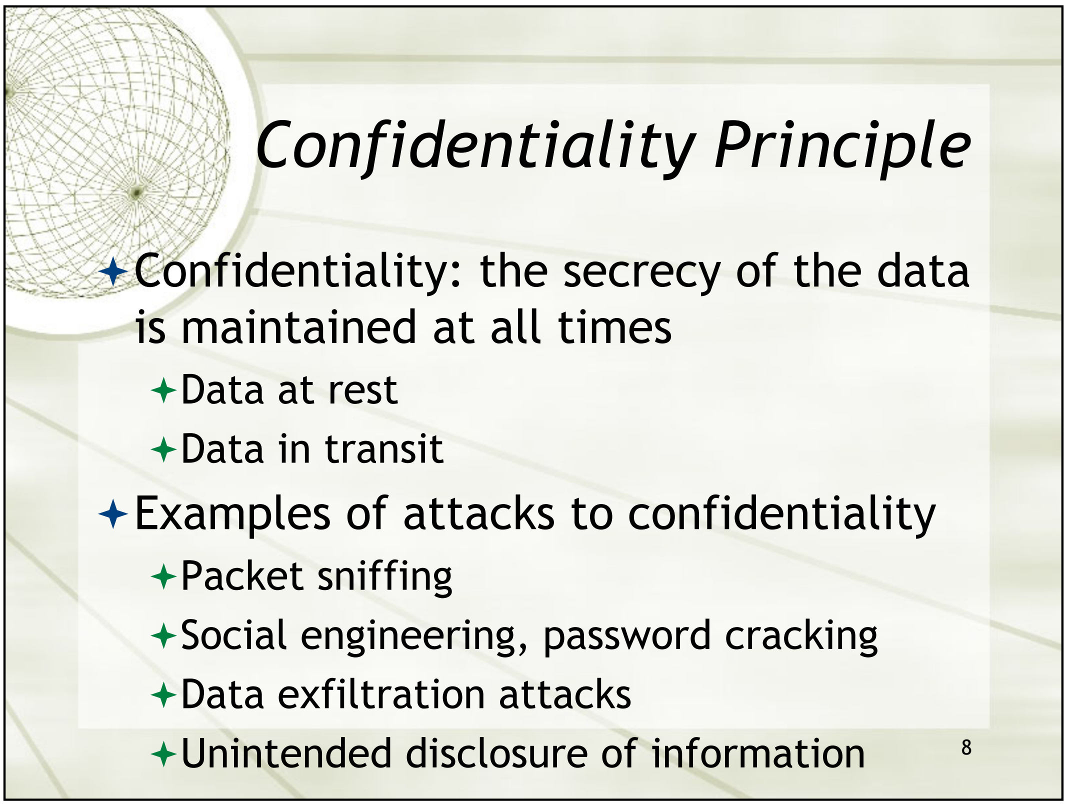
Fundamental Principles of Security

Security programs have several small and large objectives, but the three main principles in all programs are confidentiality, integrity, and availability. These are referred to as the CIA triad. The level of security required to accomplish these principles differs with each company, because each has its own unique combination of business and security goals and requirements. All security controls, mechanisms, and safeguards are implemented to provide one or more of these principles, and all risks, threats, and vulnerabilities are measured for their potential capability to compromise one or all of the AIC principles.



Security System Principles

- Security measures put in place to attempt to:
  - provide support the CIA principles
  - address the threats that may compromise one or more of the CIA principles



Confidentiality Principle

- Confidentiality: the secrecy of the data is maintained at all times
  - Data at rest
  - Data in transit
- Examples of attacks to confidentiality
  - Packet sniffing
  - Social engineering, password cracking
  - Data exfiltration attacks
  - Unintended disclosure of information

8

Confidentiality

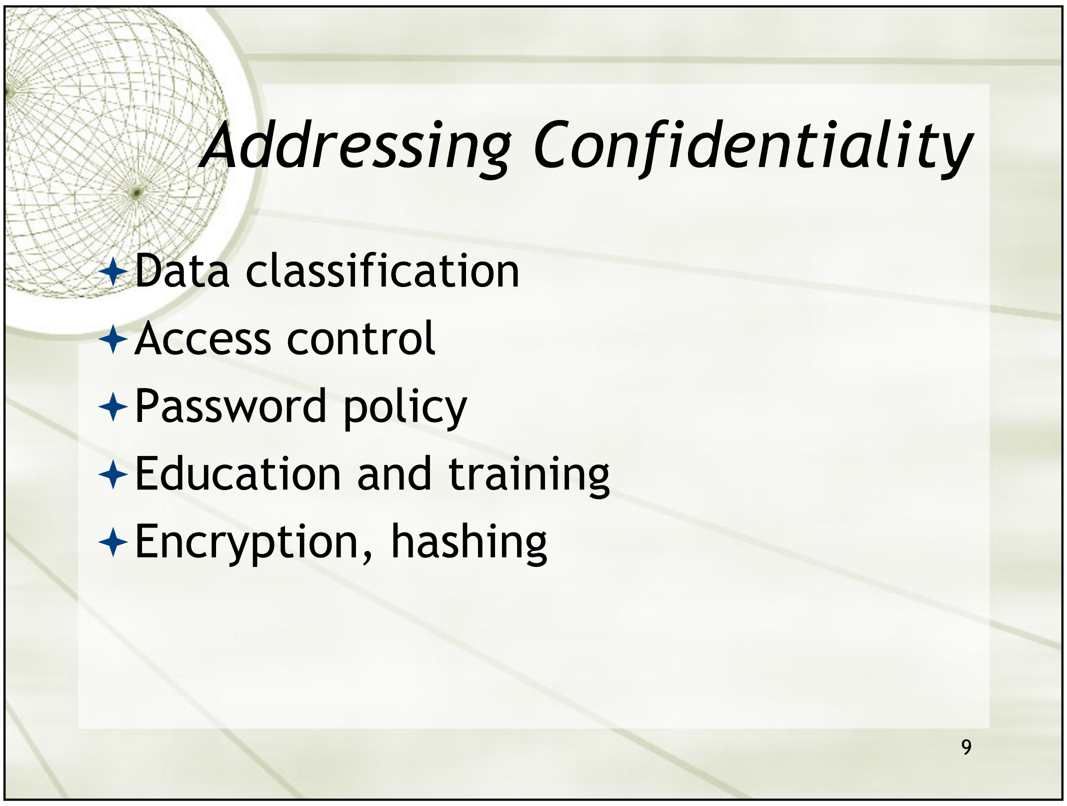
Confidentiality ensures that the necessary level of secrecy is enforced at each junction of data processing and prevents unauthorized disclosure. This level of confidentiality should prevail while data resides on systems and devices within the network, as it is transmitted, and once it reaches its destination.

Attackers can thwart confidentiality mechanisms by network monitoring, shoulder surfing, stealing password files, and social engineering. These topics will be addressed in more depth in later lectures, but briefly, shoulder surfing is when a person looks over another person's shoulder and watches their keystrokes or views data as it appears on a computer screen. Social engineering is when one person tricks another person into sharing confidential information, for example, by posing as someone authorized to have access to that information. Social engineering can take many other forms. Indeed, any one-to-one communication medium can be used to perform social engineering attacks.

Users can intentionally or accidentally disclose sensitive information by not encrypting it before sending it to another person, by falling prey to a social engineering attack, by sharing a company’s trade secrets, or by not using extra care to protect confidential information when processing it.

Confidentiality can be provided by encrypting data as it is stored and transmitted; by using network traffic padding, strict access control, and data classification; and by training personnel on the proper procedures.

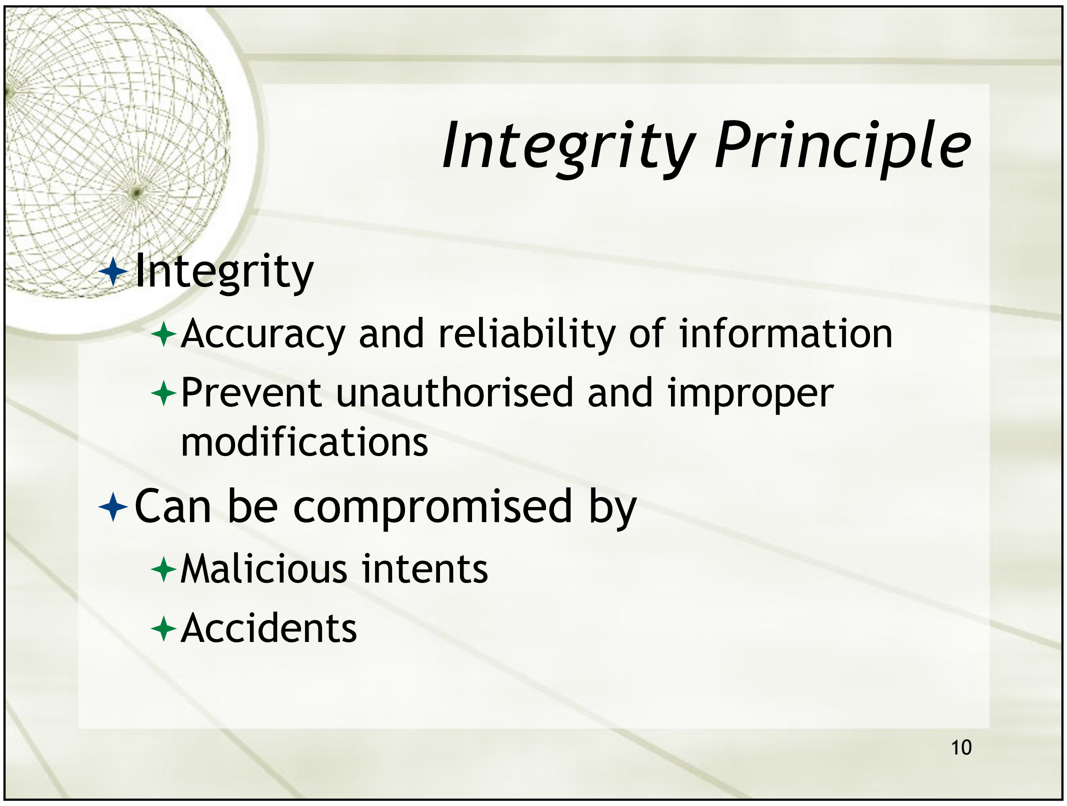
Availability, integrity, and confidentiality are critical principles of security. One needs to understand their meaning, how they are provided by different mechanisms, and how their absence can negatively affect an environment, all of which help you best identify problems and provide proper solutions.



Addressing Confidentiality

- Data classification
- Access control
- Password policy
- Education and training
- Encryption, hashing

9



Integrity Principle

- Integrity
  - Accuracy and reliability of information
  - Prevent unauthorised and improper modifications
- Can be compromised by
  - Malicious intents
  - Accidents

10

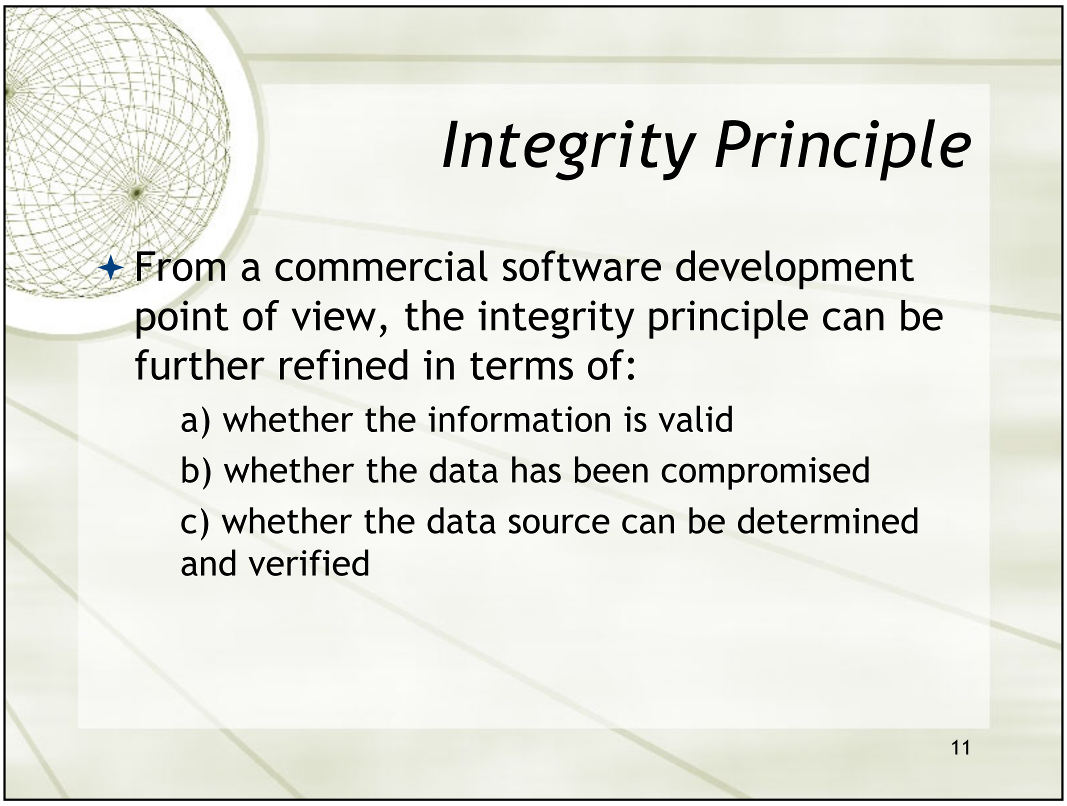
Integrity

Integrity is upheld when the assurance of the accuracy and reliability of the information and systems is provided, and any unauthorized modification is prevented. Hardware, software, and communication mechanisms must work in concert to maintain and process data correctly and to move data to intended destinations without unexpected alteration. The systems and network should be protected from outside interference and contamination.

Environments that enforce and provide this attribute of security ensure that attackers, or mistakes by users, do not compromise the integrity of systems or data. When an attacker inserts a virus, logic bomb, or back door into a system, the system's integrity is compromised. This can, in turn, harm the integrity of information held on the system by way of corruption, malicious modification, or the replacement of data with incorrect data. Strict access controls, intrusion detection, and hashing can combat these threats.

Users usually affect a system or its data's integrity by mistake (although internal users may also commit malicious deeds). For example, users with a full hard drive may unwittingly delete configuration files under the mistaken assumption that deleting a boot.ini file must be okay because they don't remember ever using it. Or, for example, a user may insert incorrect values into a data processing application that ends up charging a customer $3 million instead of $300. Incorrectly modifying data kept in databases is another common way users may accidentally corrupt data—a mistake that can have lasting effects.

Security should streamline users' capabilities and give them only certain choices and functionality, so errors become less common and less devastating. System-critical files should be restricted from viewing and access by users. Applications should provide mechanisms that check for valid and reasonable input values. Databases should let only authorized individuals modify data, and data in transit should be protected by encryption or other mechanisms.



Integrity Principle

- From a commercial software development point of view, the integrity principle can be further refined in terms of:
  - a) whether the information is valid
  - b) whether the data has been compromised
  - c) whether the data source can be determined and verified

11

One can talk about security goals and objectives in terms of confidentiality, integrity, and availability (CIA). However, one really needs to consider just what these words represent. So what is meant by confidentiality. Confidentiality rules consider: leakage of rights, and information flow.

The “commercial” requirements for confidentiality are not the same as those concerned with defense and national security. In the “military” environment, access to a security level brings the ability to access data classified to that level. In a commercial setting, roles are too informal for the assignment of security levels as specified by what are called clearances.

Integrity rules consider:

1. Information integrity—Is a given piece of information valid/correct?
2. Data integrity—Has a given piece of information been modified?
3. Origin integrity—Where did a piece of information come

from?

We could formally define integrity as: If X is a set of entities and I is some information, I has the property of integrity with respect to X if all x 2 X trust information I.

Unfortunately, this is not a very useful definition from an engineering perspective. Ideally we should have something a little more constructive, and in 1982 Steven Lipner articulated a useful set of requirements for rule 1 above (information integrity):

1. Users will not write their own programs and users will only use existing production programs and databases.
2. Programmers will develop and test programs on non-production system and, if access is needed to actual data, real data must be supplied via a special process that scrubs the data to be used on development systems.
3. Administrators will follow a special process to install a program into the production environment from the development system.
4. Managers and auditors will control and audit the special processing requirement.
5. Managers and auditors will have access to both the system state and the system logs that are generated on production systems.

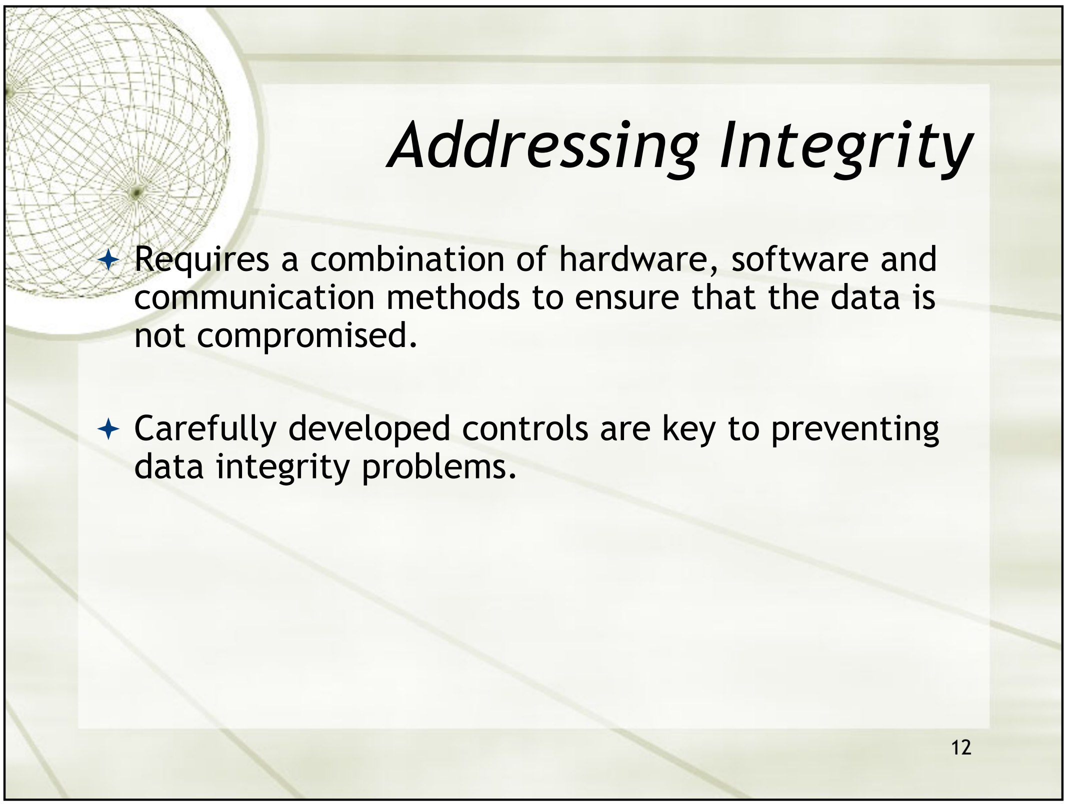
These information integrity requirements rely on three integrity policy principles of operation:

. Separation of duty—If two or more steps are required to perform a critical function then at least two different people should perform the steps.

. Separation of function—Developers do not develop new programs on production systems, and developers do not process production data on development systems.

. Auditing—Logging must be in place to enable one to determine what actions took place when and by whom.

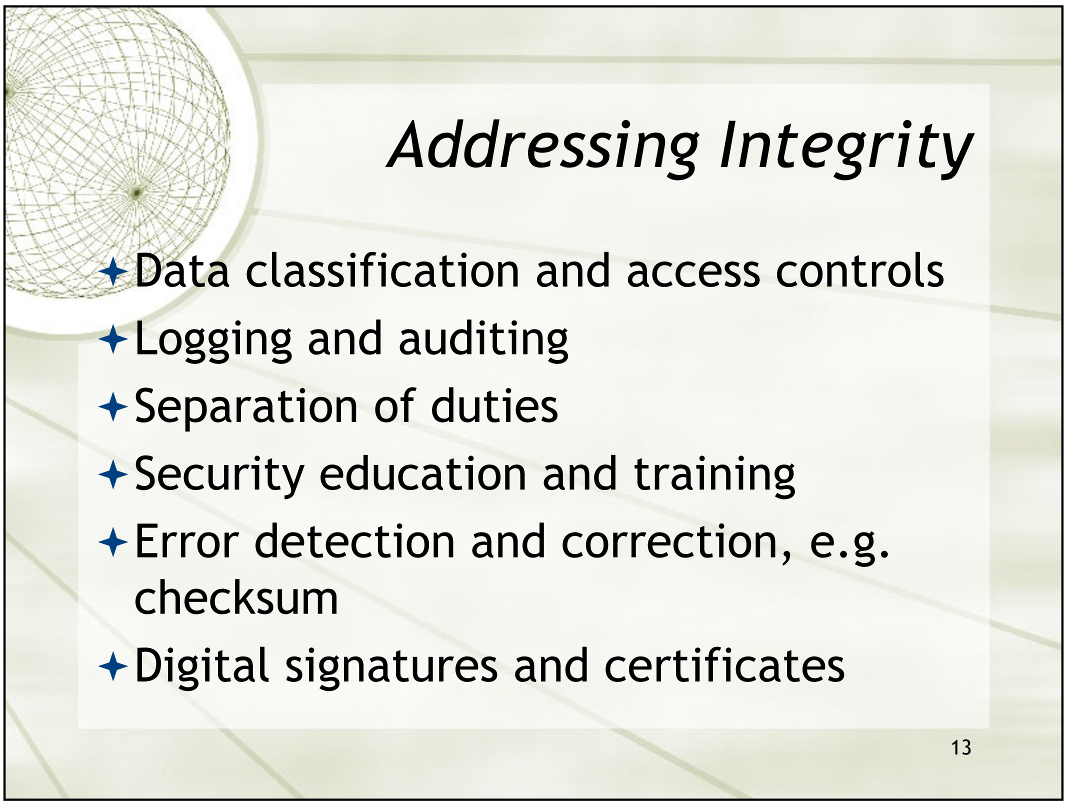
Data integrity and origin integrity we will discuss under the security services topics. These services identify the abstract forms of functionality necessary to achieve the organization’s technical security requirements and objectives. One should be able to identify the necessary security services based on the detailed security requirements that reflect the security rules/policy decomposed to atomic verifiable capabilities.



Addressing Integrity

- Requires a combination of hardware, software and communication methods to ensure that the data is not compromised.
- Carefully developed controls are key to preventing data integrity problems.

12



Addressing Integrity

- Data classification and access controls
- Logging and auditing
- Separation of duties
- Security education and training
- Error detection and correction, e.g. checksum
- Digital signatures and certificates

13

Examples:

LOGGING AND AUDITING.

The concept of logging, as part of information integrity, directly speaks to the last point made above. Specifically, all activities that alter information, or the processes that process information, need to generate log entries that allow for the auditing of production processes. This auditing is critical to understanding what is occurring within a system and providing a level of assurance that information is correct and valid. This capability relies on the presence of noncircumventable logging mechanisms.

SEPARATION OF DUTY

The concept of separation of duties, as part of information integrity, can also be seen in the example above. If:

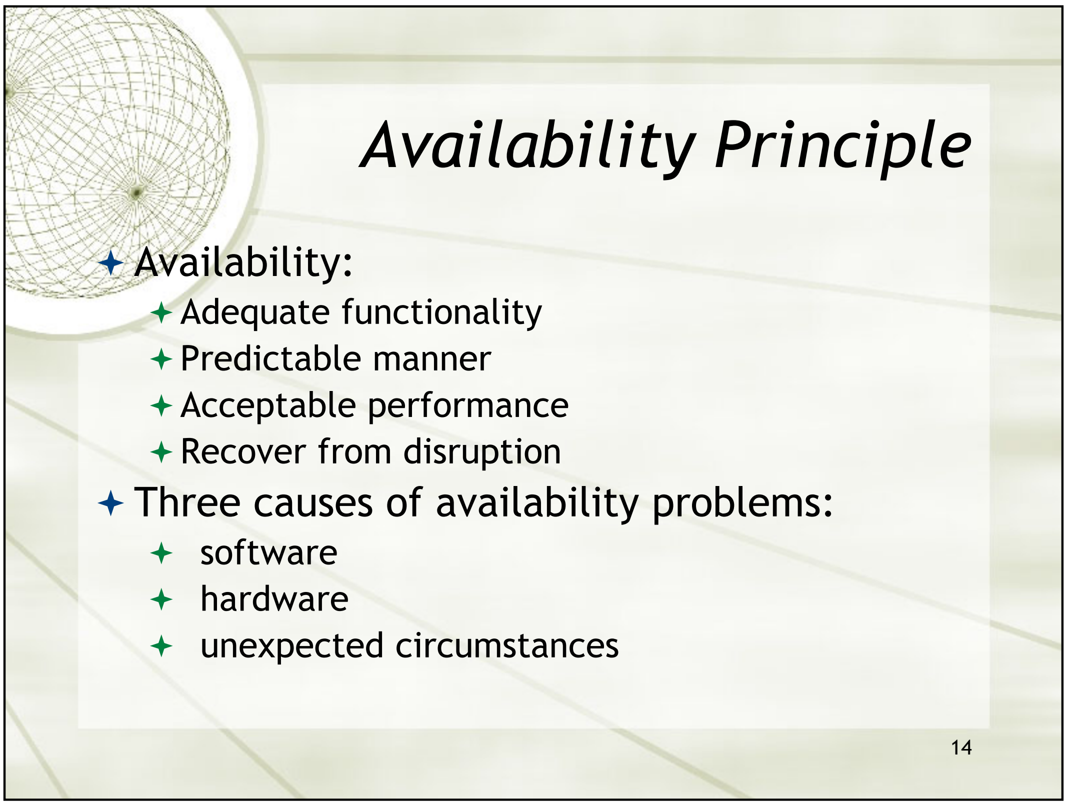
employee A is only authorized to issue purchase orders,

employee B is only authorized to receive shipments and shipping papers/receipts,

and

employee C is only authorized to issue checks to suppliers, . then any attempt by the supplier to get paid for an invalid invoice (products/services not ordered) would require the supplier to conspire with at least employees A and B.

A successful conspiracy among suppliers and employees, or only among employees, becomes more difficult as the number of conspiracy members increases. Separating duties among multiple employees reduces the probability that information is forged (not valid).



Availability Principle

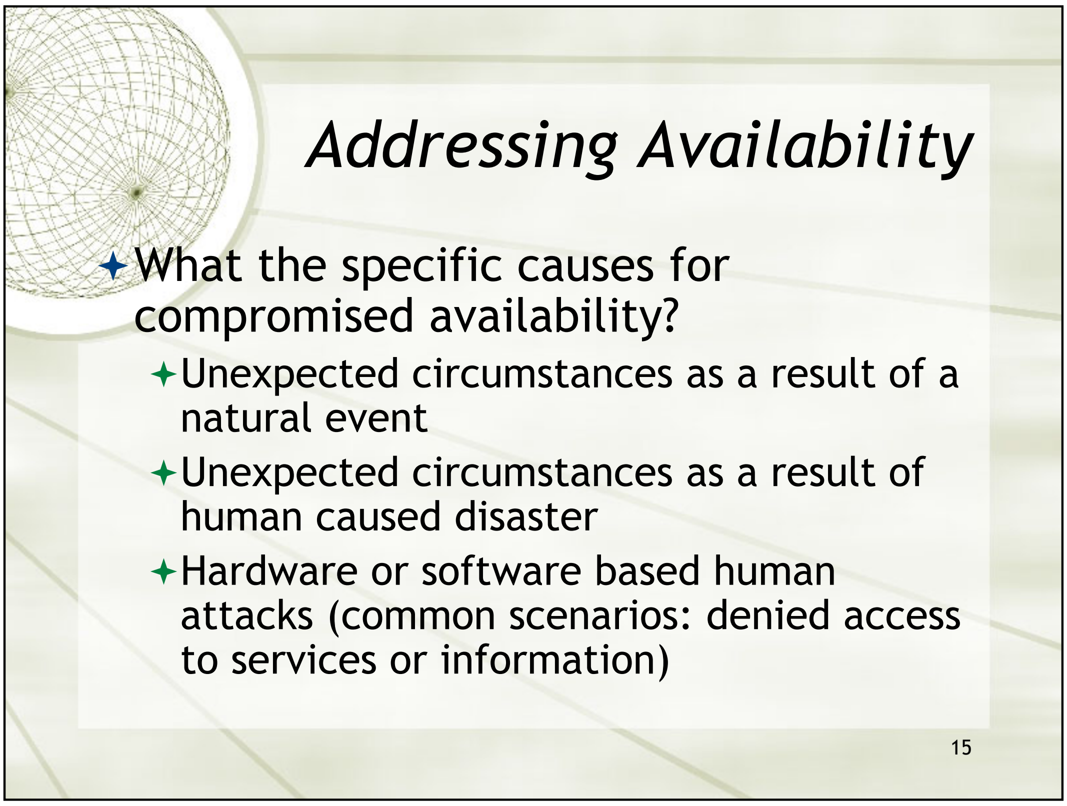
- Availability:
  - Adequate functionality
  - Predictable manner
  - Acceptable performance
  - Recover from disruption
- Three causes of availability problems:
  - software
  - hardware
  - unexpected circumstances

14

Availability

The systems and networks should provide adequate capacity to perform in a predictable manner with an acceptable level of performance. They should be able to recover from disruptions in a secure and quick manner so productivity is not negatively affected. Single points of failure should be avoided, backup measures should be taken, redundancy mechanisms should be in place when necessary, and the negative effects from environmental components should be prevented. Necessary protection mechanisms must be in place to protect against inside and outside threats that could affect the availability and productivity of the network, systems, and information. Availability ensures reliability and timely access to data and resources to authorized individuals.

System availability can be affected by device or software failure. Backup devices should be used and be available to quickly replace critical systems, and employees should be skilled and on hand to make the necessary adjustments to bring the system back online. Environmental issues like heat, cold, humidity, static electricity, and contaminants can also affect system availability. Systems should be protected from these elements, properly grounded electrically, and closely monitored.



Addressing Availability

- What the specific causes for compromised availability?
  - Unexpected circumstances as a result of a natural event
  - Unexpected circumstances as a result of human caused disaster
  - Hardware or software based human attacks (common scenarios: denied access to services or information)

15

Availability, as a security service, must consider business continuity from a number of perspectives, namely continuity of business operations during:

1. natural disasters, such as hurricanes, tornadoes, earthquakes, and floods;

2. human-caused disasters, such as, construction flaws and accidental fires; and

3. human-caused malicious actions such as attacks on physical or logical assets (object) and infrastructure components.

Let us look at these areas in more detail. Natural disaster business continuity (1) is not really a security subject, rather is a critical component of general business operational planning. Human-caused disaster business continuity (2) likewise is not really a security subject and not addressed further. However, human malicious activities (3) are a major security concern.

Malicious human-initiated activities that target the availability of legitimate access to service platforms, include:

. Preventing customer access to electronic commerce websites (online banking, stock trading, online stores, corporate information sites, etc.), and

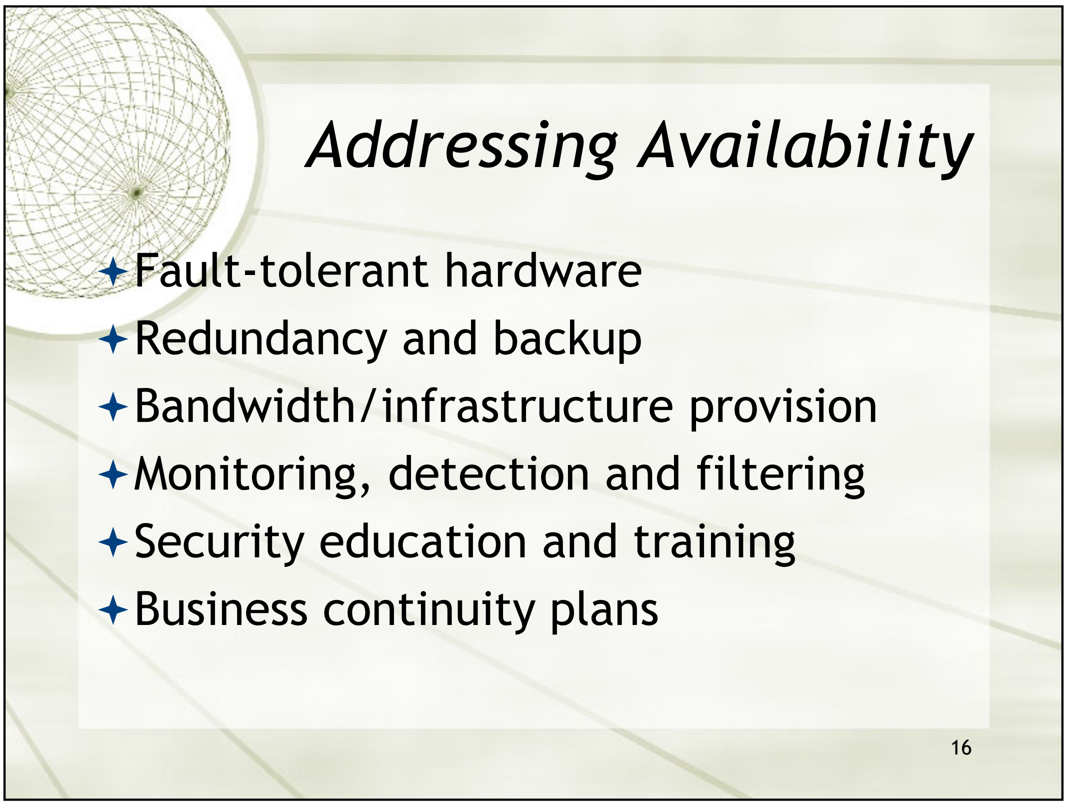
. Preventing organization employee access to internal services (DNS services, web and file servers, application and management servers, router and other network components, etc.)

Service platform functionality can also be targeted by

. Attempting to cause service delivery platforms to fail completely (crashing routers, servers, and other infrastructure elements, etc.)

. Attempting to cause service delivery platforms to malfunction (DNS servers returning invalid host-address binding information, unauthorized modification of customer service profiles, etc.)

The availability security service focuses on preventing denial of service access, preventing service failures (as in high availability of services) and restoration or continuity of services when attacks occur.



Addressing Availability

- Fault-tolerant hardware
- Redundancy and backup
- Bandwidth/infrastructure provision
- Monitoring, detection and filtering
- Security education and training
- Business continuity plans

16

PREVENTION OF SERVICE ACCESS DENIAL.

Denial of service (DoS) attacks are all too common in our Internet connected world. These attacks can be launched from anywhere and can involve many attacking computers (in fact reports of hundreds of thousands to millions of computers involved in DoS attacks are becoming commonplace4). The need to deploy security mechanisms that can mitigate these DoS attacks is now fundamental to any organization that offers Internet access to services and products.

PREVENTION OF SERVICE FAILURE.

Complete failure of a service can cause significant financial loss and is most effectively addressed via what are called high-availability solutions. These solutions range from the use of “fault-tolerant” computer implementations, deployment of standby resources (one spare for each primary element, one (or more) spare devices for a number of primary elements), to “loadsharing” platform deployments. One consideration for the sparing and load-sharing approaches has to do with network access, namely the primary Internet internet working protocol (IP version 4) requires that an IP address can only be associated with a single network interface at any instant of time. Consequently sparing and load-sharing approaches need to address how the IP address used to access a service is transferred from one service platform to another platform. Unfortunately, the protocols used for network access to many services do not allow for platform IP address changes without service interruption. The fault-tolerant technology does not suffer from this issue.

RESTORATION/CONTINUITY OF SERVICES AND INFORMATION.

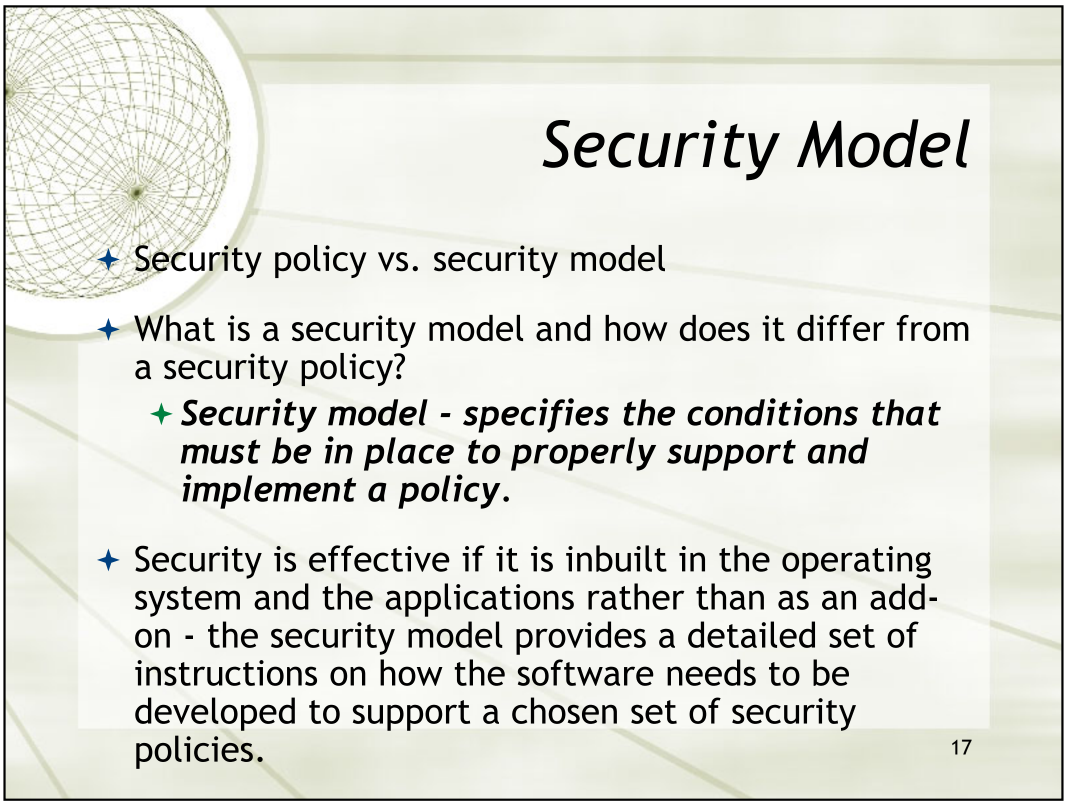
From a security perspective the restoration or continuity of services and information takes two forms:

. isolating systems that are, or have been, attacked to limit the spread of attack- caused damage (essentially the quarantine of components that are believed compromised by an attack), and

. restoring of information that was damaged (modified or deleted) by an attack.

Isolation of attacked system components depends on the availability of alternative components and the capability to redirect service access traffic away from the attacked components and toward the alternative components. This redirection capability has to be designed into a network infrastructure to allow the switching traffic to alternate network segments. The initiation of redirection is contingent on having the ability to recognize that an attack is in progress and be able to identify the specific infrastructure elements under attack.

Restoral of service-related information depends on possession of information that is known to be valid. This capability necessitates routinely making copies of essential information, verifying that the information copies are restorable, and storing the information copies in a manner that ensures that the stored copies cannot be modified, or otherwise damaged, by unauthorized individuals. This restoral capability is not just having valid copies of the information, well-planned procedures, trained personnel, and timely management involvement must exist so that the restoral activity can occur in a timely and effective manner, thereby ensuring that service access and functionality are minimally affected.



Security Model

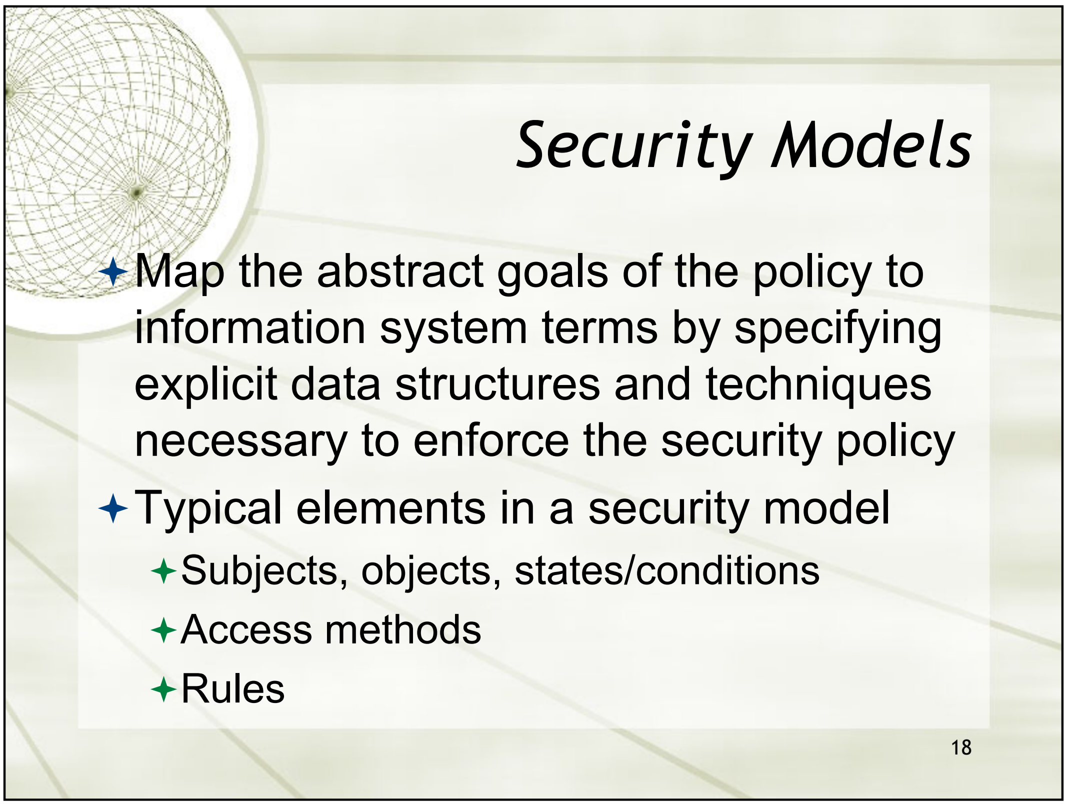
- Security policy vs. security model
- What is a security model and how does it differ from a security policy?
  - ***Security model - specifies the conditions that must be in place to properly support and implement a policy.***
- Security is effective if it is inbuilt in the operating system and the applications rather than as an add-on - the security model provides a detailed set of instructions on how the software needs to be developed to support a chosen set of security policies.

17

Computer and information security covers many areas within an enterprise. Each area has security vulnerabilities and, hopefully, some corresponding countermeasures that raise the security level and provide better protection. Not understanding the different areas and security levels of network devices, operating systems, hardware, protocols, and applications can cause security vulnerabilities that can affect the environment as a whole.

Two fundamental concepts in computer and information security are the security policy and security model. A security policy is a statement that outlines how entities access each other, what operations different entities can carry out, what level of protection is required for a system or software product, and what actions should be taken when these requirements are not met. The policy outlines the expectations that the hardware and software must meet to be considered in compliance. A security model outlines the requirements necessary to properly support and implement a certain security policy. If a security policy dictates that all users must be identified, authenticated, and authorized before accessing network resources, the security model might lay out an access control matrix that should be constructed so it fulfills the requirements of the security policy. If a security policy states that no one from a lower security level should be able to view or modify information at a higher security level, the supporting security model will outline the necessary logic and rules that need to be implemented to ensure that under no circumstances can a lower-level subject access a higher-level object in an unauthorized manner. A security model provides a deeper explanation of how a computer operating system should be developed to properly support a specific security policy.

Security is best if it is designed and built into the foundation of operating systems and applications and not added as an afterthought. Once security is integrated as an important part of the design, it has to be engineered, implemented, tested, audited, evaluated, certified, and accredited. The security that a product provides must be rated on the availability, integrity, and confidentiality it claims to provide. Consumers then use these ratings to determine if specific products provide the level of security they require. This is a long road, with many entities involved with different responsibilities.



Security Models

- Map the abstract goals of the policy to information system terms by specifying explicit data structures and techniques necessary to enforce the security policy
- Typical elements in a security model
  - Subjects, objects, states/conditions
  - Access methods
  - Rules

18

Security Models

An important concept in the design and analysis of secure systems is the security model, because it incorporates the security policy that should be enforced in the system. A model is a symbolic representation of a policy. It maps the desires of the policymakers into a set of rules that a computer system must follow.

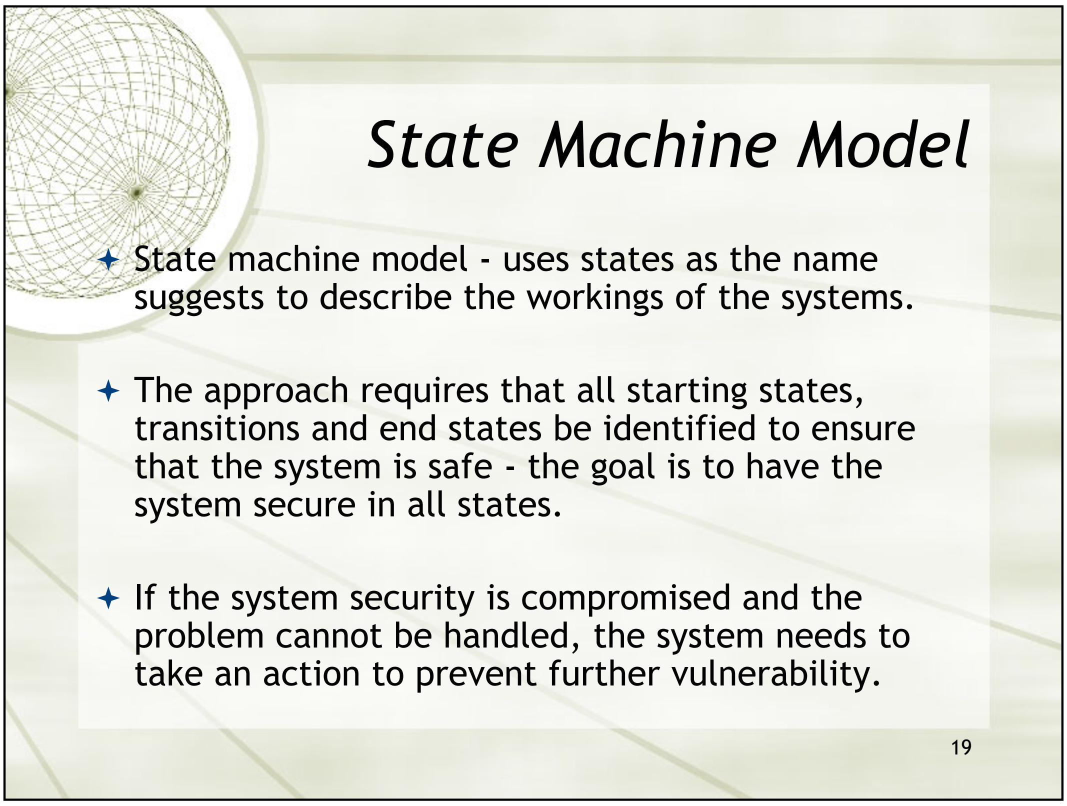
The reason this chapter has repeatedly mentioned the security policy and its importance is that it is an abstract term that represents the objectives and goals a system must meet and accomplish to be deemed secure and acceptable. How do we get from an abstract security policy to the point at which an administrator is able to uncheck a box on the GUI to disallow David from accessing configuration files on his system? There are many complex steps in between that take place during the system’s design and development.

A security model maps the abstract goals of the policy to information system terms by specifying explicit data structures and techniques necessary to enforce the security policy. A security model is usually represented in mathematics and analytical ideas, which are mapped to system specifications and then developed by programmers through programming code. So we have a policy that encompasses security goals such as “each subject must be authorized to access each object.” The security model takes this requirement and provides the necessary mathematical formulas, relationships, and structure to be followed to accomplish this goal. From there, specifications are developed per operating system type (Unix, Windows, Macintosh, and so on), and individual vendors can decide how they are going to implement mechanisms that meet these necessary specifications.

So in a very general and simplistic example, if a security policy states that subjects need to be authorized to access objects, the security model would provide the mathematical relationships and formulas explaining how x can access y only through the outlined specific methods. Specifications are then developed to provide a bridge to what this means in a computing environment and how it maps to components and mechanisms that need to be coded and developed. The developers then write the program code to produce the mechanisms that provide a way for a system to use ACLs and give administrators some degree of control. This mechanism presents the network administrator with a GUI that enables the administrator to choose (via check boxes, for example) which subjects can access what objects and to be able to set this configuration within the operating system. This is a rudimentary example, because security models can be very complex, but it is used to demonstrate the relationship between the security policy and the security model.

Some security models, such as the Bell-LaPadula model, enforce rules to provide confidentiality protection. Other models, such as the Biba model, enforce rules to provide integrity protection. Formal security models, such as Bell-LaPadula and Biba, are used to provide high assurance in security. Informal models, such as Clark-Wilson, are used more as a framework to describe how security policies should be expressed and executed.

A security policy outlines goals without regard to how they will be accomplished. A model is a framework that gives the policy form and solves security access problems for particular situations. Several security models have been developed to enforce security policies. The following sections provide overviews of each model.



State Machine Model

- State machine model - uses states as the name suggests to describe the workings of the systems.
- The approach requires that all starting states, transitions and end states be identified to ensure that the system is safe - the goal is to have the system secure in all states.
- If the system security is compromised and the problem cannot be handled, the system needs to take an action to prevent further vulnerability.

19

State Machine Models

In state machine models, to verify the security of a system, the state is used, which means that all current permissions and all current instances of subjects accessing objects must be captured. Maintaining the state of a system deals with each subject’s association with objects. If the subjects can access objects only by means that are concurrent with the security policy, the system is secure. State machines have provided a basis for important security models. A state of a system is a snapshot of a system at one moment of time. Many activities can alter this state, which are referred to as state transitions. The developers of an operating system that will implement the state machine model need to look at all the different state transitions that are possible and assess whether a system that starts up in a secure state can be put into an insecure state by any of these events. If all of the activities that are allowed to happen in the system do not compromise the system and put it into an insecure state, then the system executes a secure state machine model.

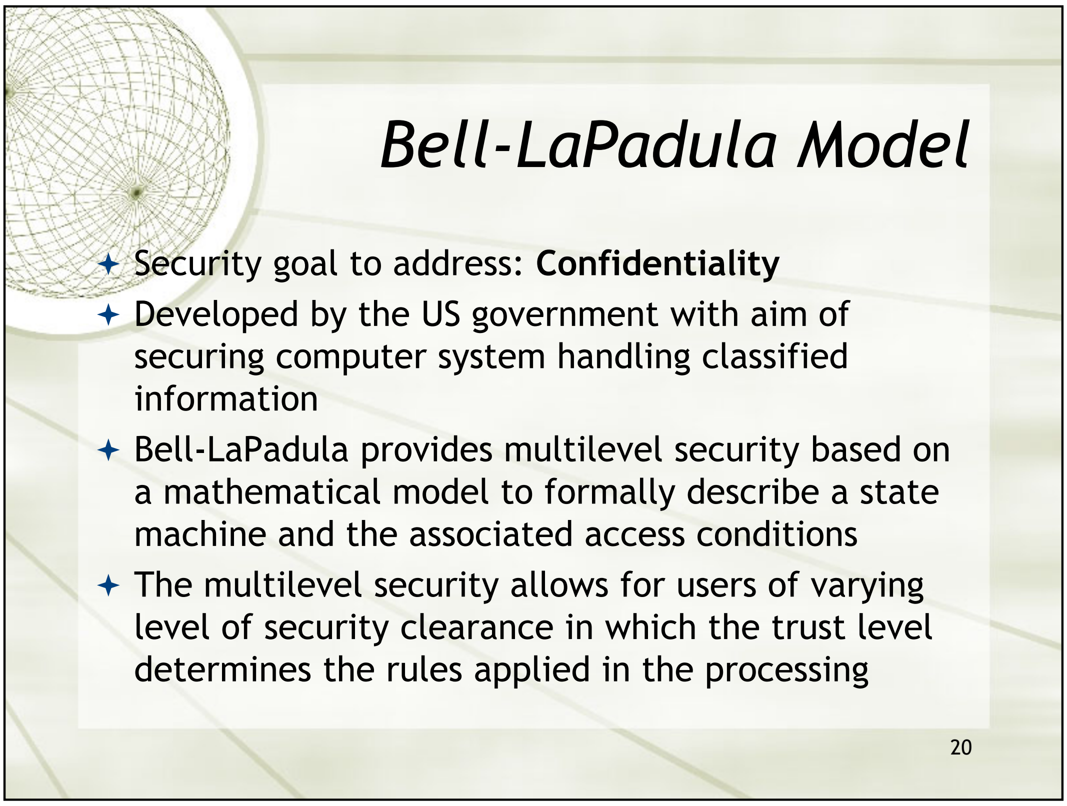
The state machine model is used to describe the behavior of a system to different inputs. It provides mathematical constructs that represent sets (subjects and objects) and sequences. When an object accepts an input, this modifies a state variable. A simplistic example of a state variable is (Name, Value). This variable is part of the operating system's instruction set. When this variable is called upon to be used, it can be populated with (Color, Red) from the input of a user or program. Let's say the user enters a different value, so now the variable is (Color, Blue). This is a simplistic example of a state transition. Some state transitions are this simple, but complexity comes in when the system must decide if this transition should be allowed. To allow this transition, the object's security attributes and the access rights of the subject must be reviewed and allowed by the operating system.

Developers who implement the state machine model must identify all the initial states (default variable values) and outline how these values can be changed (inputs that will be accepted) so the various number of final states (resulting values) still ensure that the system is safe. The outline of how these values can be changed is often implemented through condition statements: "if condition then update."

A system that has employed a state machine model will be in a secure state in each and every instance of its existence. It will boot up into a secure state, execute commands and transactions securely, allow subjects to access resources only in secure states, and shut down and fail in a secure state. Failing in a secure state is extremely important. It is imperative that if anything unsafe takes place, the system must be able to "save itself" and not make itself vulnerable. When an operating system displays an error message to the user or reboots or freezes, it is executing a safety measure. The operating system has experienced something that is deemed illegal and it cannot take care of the situation itself, so to make sure it does not stay in this insecure state, it reacts in one of these fashions. Thus, if an application or system freezes on you, know that it is simply the system trying to protect itself and your data.

Several points should be considered when developing a product that uses a state machine model. Initially, the developer must define what and where the state variables are. In a computer environment, all data variables could independently be considered state variables, and an inappropriate change to one could conceivably change or corrupt the system or another process's activities. Next, the developer must define a secure state for each state variable. The next step is to define and identify the allowable state transition functions. These functions will describe the allowable changes that can be made to the state variables.

After the state transition functions are defined, they must be tested to verify that the overall machine state will not be compromised and that these transition functions will keep the integrity of the system (computer, data, program, or process) intact at all times.



Bell-LaPadula Model

- Security goal to address: **Confidentiality**
- Developed by the US government with aim of securing computer system handling classified information
- Bell-LaPadula provides multilevel security based on a mathematical model to formally describe a state machine and the associated access conditions
- The multilevel security allows for users of varying level of security clearance in which the trust level determines the rules applied in the processing

20

The Bell-LaPadula Model

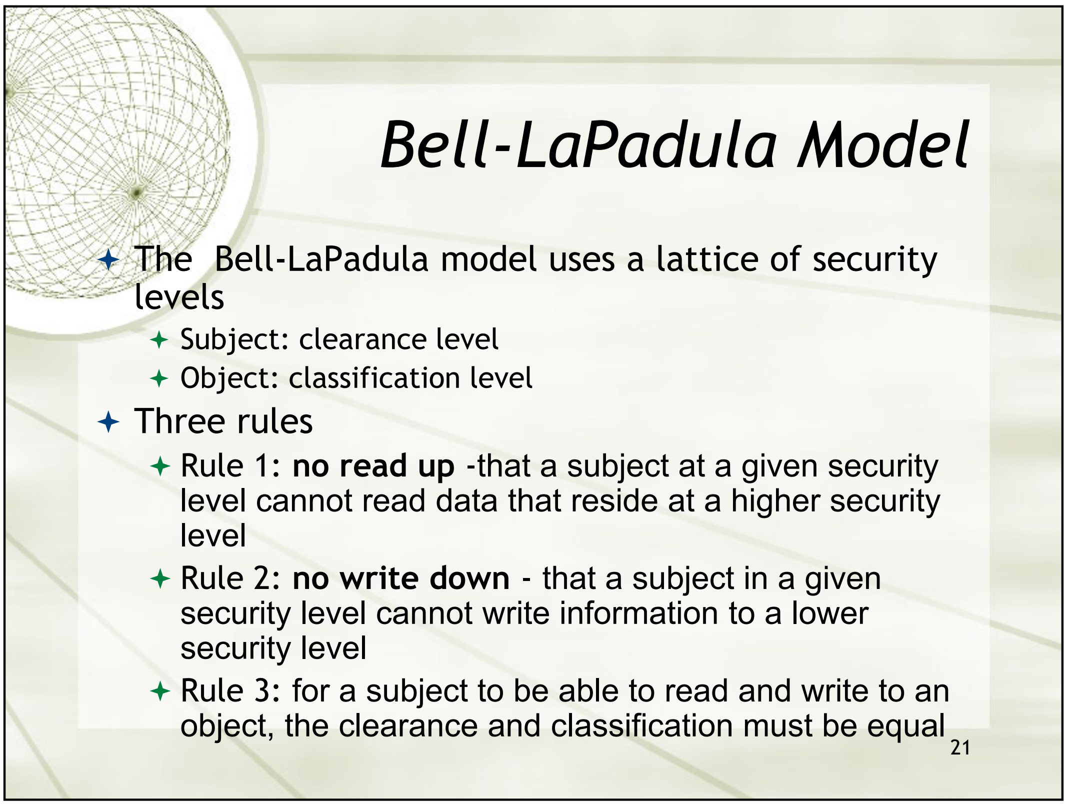
In the 1970s, the U.S. military used time-sharing mainframe systems and was concerned about the security of these systems and leakage of classified information. The Bell-LaPadula model was developed to address these concerns. It was the first mathematical model of a multilevel security policy used to define the concept of a secure state machine and modes of access and outlined rules of access. Its development was funded by the U.S. government to provide a framework for computer systems that would be used to store and process sensitive information. The model's main goal was to prevent secret information from being accessed in an unauthorized manner.

A system that employs the Bell-LaPadula model is called a multilevel security system because users with different clearances use the system, and the system processes data with different classifications. The level at which information is classified determines the handling procedures that should be used. The Bell-LaPadula model is a state machine model that enforces the confidentiality aspects of access control. A matrix and security levels are used to determine if subjects can access different objects. The subject's clearance is compared to the object's classification and then specific rules are applied to control how subject-to-object interactions can take place.

This model uses subjects, objects, access operations (read, write, and read/write), and security levels. Subjects and objects can reside at different security levels and will have relationships and rules dictating the acceptable activities between them. This model, when properly implemented and enforced, has been mathematically proven to provide a very secure and effective operating system. It is an information-flow security model also, which means that information does not flow in an insecure manner.

The Bell-LaPadula model is a subject-to-object model. An example would be how you (subject) could read a data element (object) from a specific database and write data into that database. The Bell-LaPadula model focuses on ensuring that subjects are properly authenticated—by having the necessary security clearance, need to know, and formal access approval—before accessing an object.

Three main rules are used and enforced in the Bell-LaPadula model: the simple security rule, the *-property (star property) rule, and the strong star property rule.



Bell-LaPadula Model

- The Bell-LaPadula model uses a lattice of security levels
  - Subject: clearance level
  - Object: classification level
- Three rules
  - Rule 1: **no read up** -that a subject at a given security level cannot read data that reside at a higher security level
  - Rule 2: **no write down** - that a subject in a given security level cannot write information to a lower security level
  - Rule 3: for a subject to be able to read and write to an object, the clearance and classification must be equal

21

The simple security rule states that a subject at a given security level cannot read data that reside at a higher security level. For example, if Bob is given the security clearance of secret, this rule states he cannot read data classified as top secret. If the organization wanted Bob to be able to read top-secret data, it would have given him that clearance in the first place.

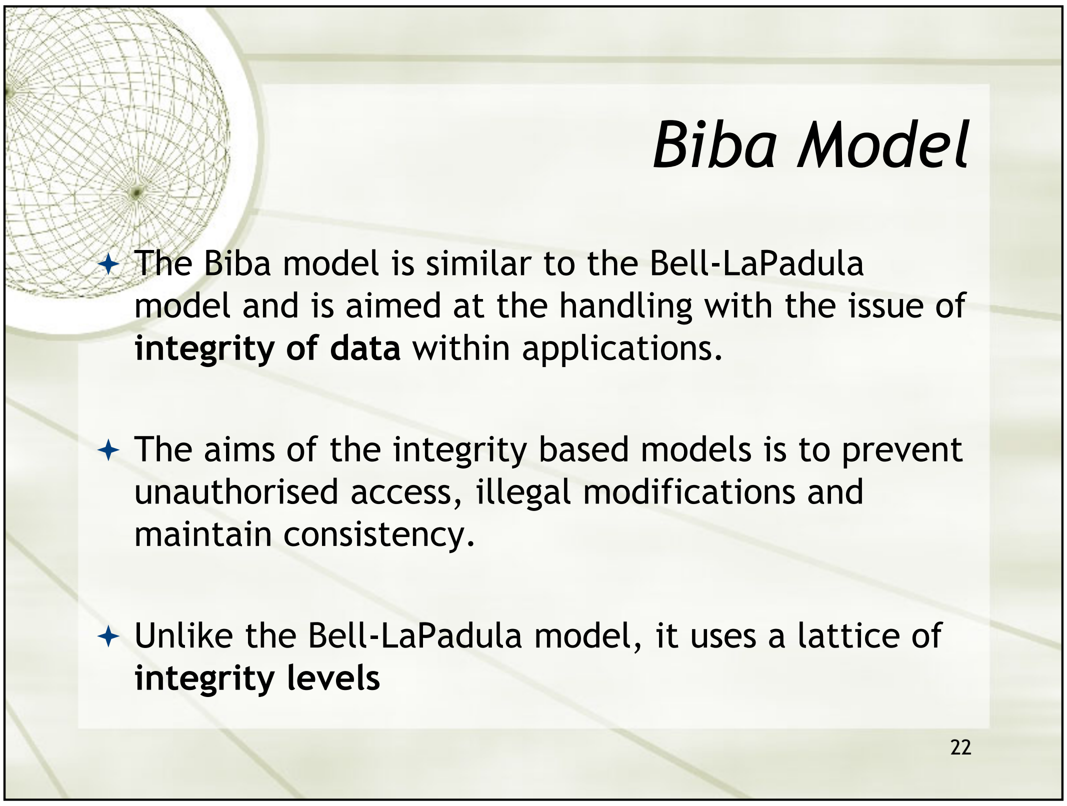
The *-property rule (star property rule) states that a subject in a given security level cannot write information to a lower security level. The simple security rule is referred to as the “no read up” rule, and the *-property rule is referred to as the “no write down” rule.

The third rule, the strong star property rule, states that a subject that has read and write capabilities can only perform those functions at the same security level, nothing higher and nothing lower. So, for a subject to be able to read and write to an object, the clearance and classification must be equal.

These three rules indicate what states the system can go into. Remember that a state is the values of the variables in the software at a snapshot in time. If a subject has performed a read operation on an object at a lower security level, the subject now has a variable that is populated with the data that was read, or copied into its variable. If a subject has written to an object at a higher security level, the subject has modified a variable within that object's domain.

All Mandatory Access Control systems are based on the Bell-LaPadula model, because it allows for multilevel security to be integrated into the code. Subjects and objects are assigned labels. The subject's label contains its clearance label (top secret, secret, or confidential) and the object's label contains its classification label (top secret, secret, or confidential). When a subject attempts to access an object, the system compares the subject's clearance label and the object's classification label and looks at a matrix to see if this is a legal and secure activity. In our scenario, it is a perfectly fine activity, and the subject is given access to the object. Now, if the subject's clearance label is top secret and the object's classification label is secret, the subject cannot write to this object, because of the *-property rule, which makes sure that subjects cannot accidentally or intentionally share confidential information by writing to an object at a lower security level. As an example, suppose that a busy and clumsy general (who has top-secret clearance) in the army opens up a briefing letter (which has a secret classification) that will go to all clerks at all bases around the world. He attempts to write that the United States is attacking Cuba. The Bell-LaPadula model will come into action and not permit this general to write this information to this type of file because his clearance is higher than that of the memo.

Likewise, if a nosey military staff clerk tried to read a memo that was available only to generals and above, the Bell-LaPadula model would stop this activity. The clerk's clearance is lower than that of the object (the memo), and this violates the simple security rule of the model. It is all about keeping secrets secret.



Biba Model

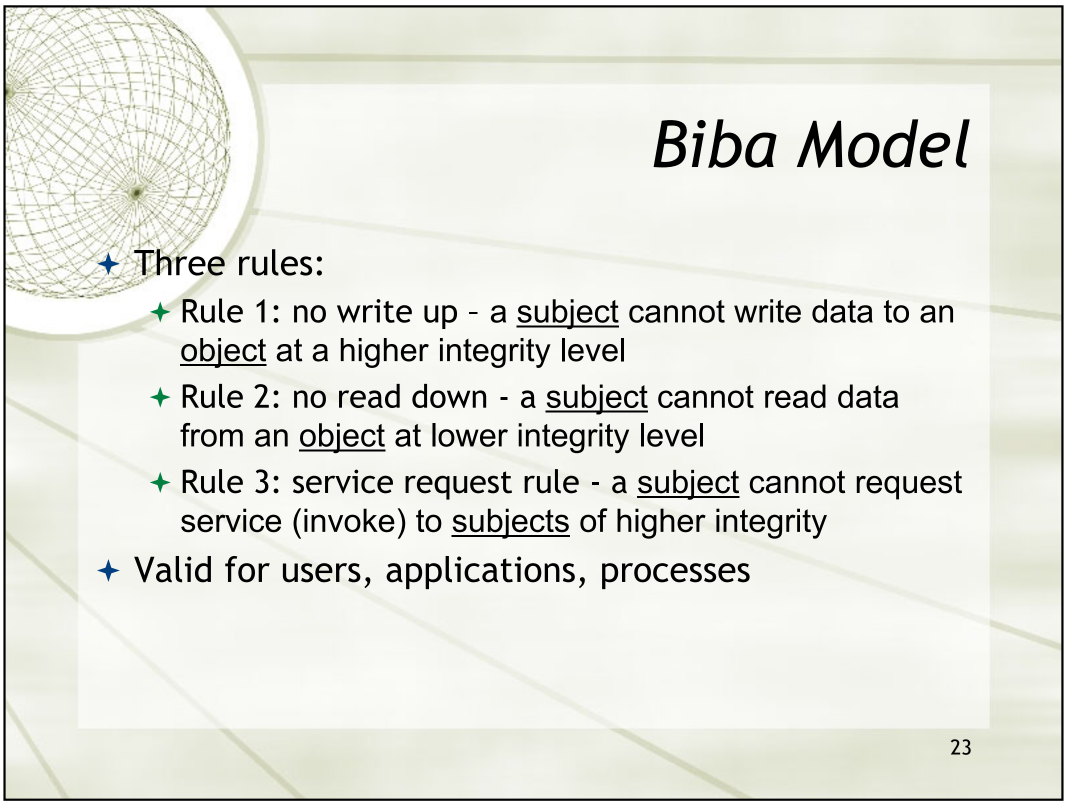
- The Biba model is similar to the Bell-LaPadula model and is aimed at the handling with the issue of **integrity of data** within applications.
- The aims of the integrity based models is to prevent unauthorised access, illegal modifications and maintain consistency.
- Unlike the Bell-LaPadula model, it uses a lattice of **integrity levels**

22

The Biba model was developed after the Bell-LaPadula model. It is a state machine model and is very similar to the Bell-LaPadula model. Biba addresses the integrity of data within applications. The Bell-LaPadula model uses a lattice of security levels (top secret, secret, sensitive, and so on). These security levels were developed mainly to ensure that sensitive data were only available to authorized individuals. The Biba model is not concerned with security levels and confidentiality, so it does not base access decisions upon this type of lattice. The Biba model uses a lattice of integrity levels.

Note: this is about integrity levels – it is not the same as security levels in the Bell-Lapadulla model

If implemented and enforced properly, the Biba model prevents data from any integrity level from flowing to a higher integrity level.


Biba Model

- Three rules:
  - Rule 1: no write up - a subject cannot write data to an object at a higher integrity level
  - Rule 2: no read down - a subject cannot read data from an object at lower integrity level
  - Rule 3: service request rule - a subject cannot request service (invoke) to subjects of higher integrity
- Valid for users, applications, processes

23

Biba has three main rules to provide this type of protection:

- *-integrity axiom A subject cannot write data to an object at a higher integrity level (referred to as “no write up”).
- Simple integrity axiom A subject cannot read data from a lower integrity level (referred to as “no read down”).
- Invocation property A subject cannot request service (invoke) to subjects of higher integrity.

The name “simple integrity axiom” might sound a little goofy, but this rule protects the subject and data at a higher integrity level from being corrupted by data at a lower integrity level. This is all about trusting the source of the information. Another way to look at it is that trusted data are “clean” data and untrusted data (from a lower integrity level) are “dirty” data. Dirty data should not be mixed with clean data, because that could ruin the integrity of the clean data.

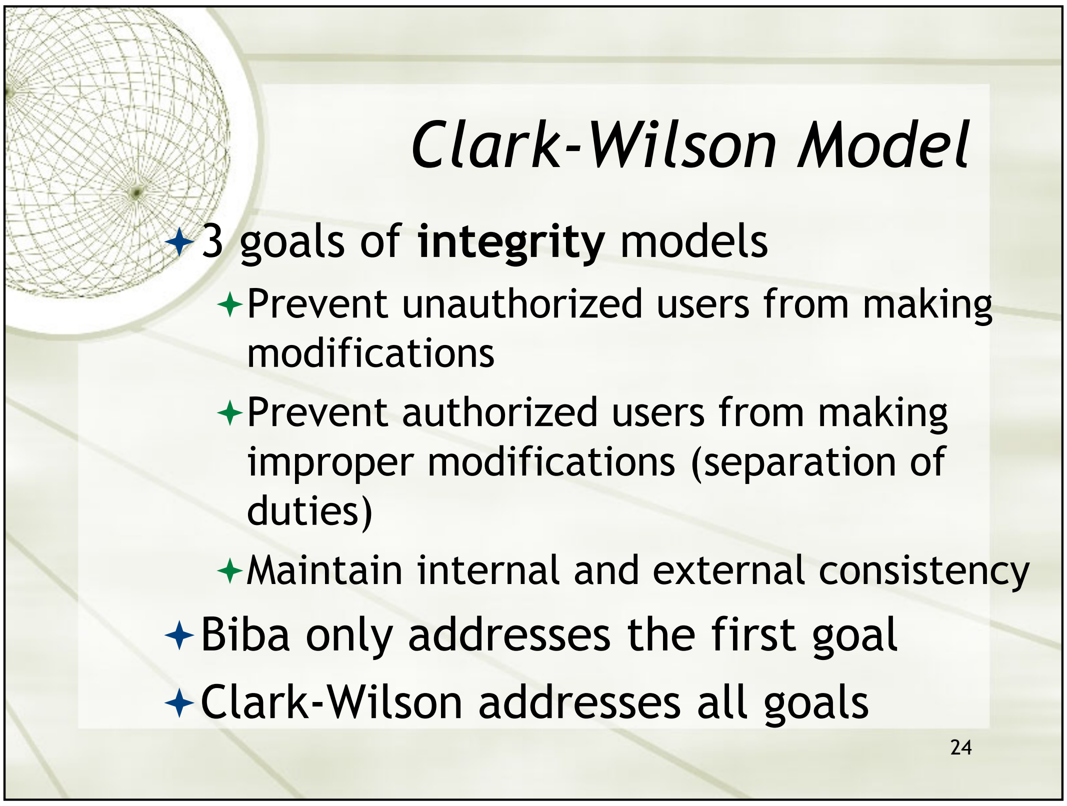
The simple integrity axiom applies not only to subjects creating the data, but also to processes. A process of lower integrity should not be writing to trusted data of a higher integrity level. The areas of the different integrity levels are compartmentalized within the application that is based on the Biba model.

An analogy would be if you were writing an article for The New York Times about the security trends over the last year, the amount of money businesses lost, and the cost/ benefit ratio of implementing firewalls, IDSs, and vulnerability scanners. You do not want to get your data and numbers from any old web site without knowing how those figures were calculated and the sources of the information. Your article (data at a higher integrity level) can be compromised if mixed with unfounded information from a bad source (data at a lower integrity level).

When you are first learning about the Bell-LaPadula and Biba models, they may seem very similar, and the reasons for their differences may be somewhat confusing. The Bell-LaPadula model was written for the U.S. government, and the government is very paranoid about leakage of its secret information. In its model, a user cannot write to a lower level because that user might let out some secrets. Similarly, a user at a lower level cannot read anything at a higher level because that user might learn some secrets. However, not everyone is so worried about confidentiality and has such big important secrets to protect. The commercial industry is more concerned about the integrity of its data. An accounting firm is more worried about keeping its numbers straight and making sure decimal points are not dropped or extra zeroes are not added in a process carried out by an application. The accounting firm is more concerned about the integrity of these data and is usually under little threat of someone trying to steal these numbers, so the firm would use software that employs the Biba model. Of course, the accounting firm does not look for the name Biba on the back of a product or make sure it is in the design of its application. Which model to use is something that was decided upon and implemented when the application was being designed. The assurance ratings are what consumers use to determine if a system is right for them. So, even if the accountants are using an application that employs the Biba model, they would not necessarily know (and we're not going to tell them).

Bell-LaPadula vs. Biba

The Bell-LaPadula model is used to provide confidentiality. The Biba model is used to provide integrity. The Bell-LaPadula and Biba models are informational flow models because they are most concerned about data flowing from one level to another. Bell-LaPadula uses security levels, and Biba uses integrity levels. It is important to know the rules of Biba and Bell-LaPadula. Their rules sound very similar: simple and * rules—one writing one way and one reading another way.



Clark-Wilson Model

- 3 goals of **integrity** models
  - Prevent unauthorized users from making modifications
  - Prevent authorized users from making improper modifications (separation of duties)
  - Maintain internal and external consistency
- Biba only addresses the first goal
- Clark-Wilson addresses all goals

24

Goals of Integrity Models

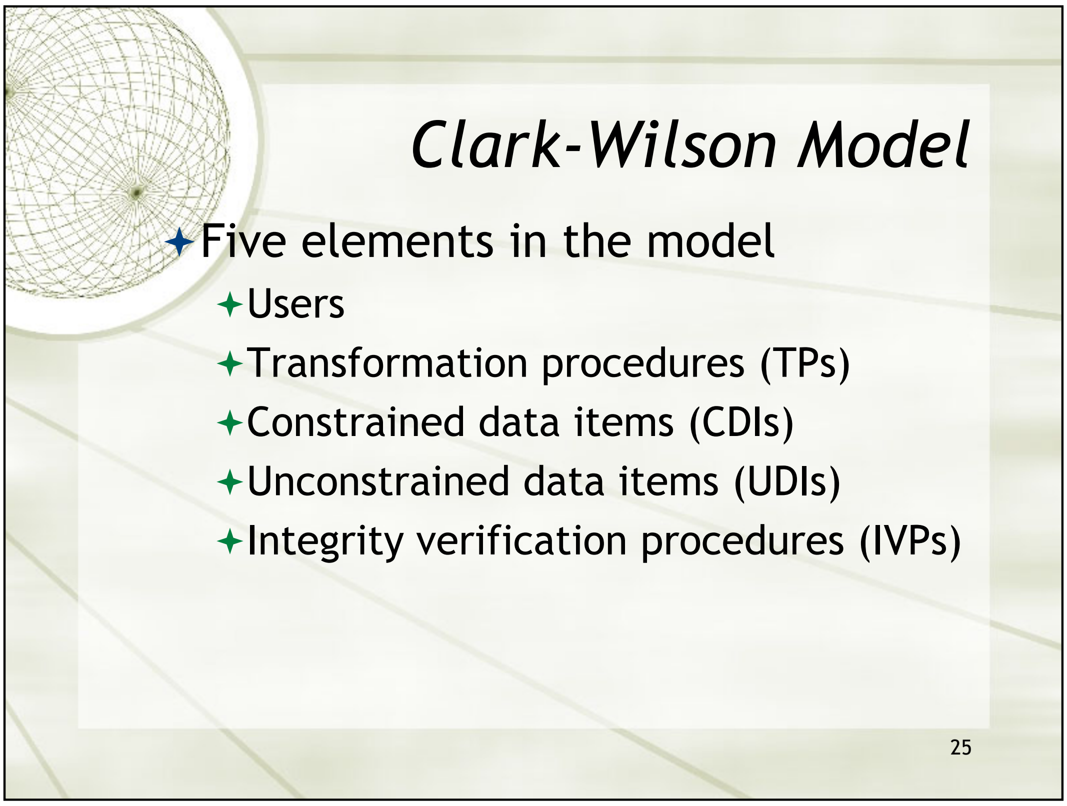
The following are the three main goals of integrity models:

1) Prevent unauthorized users from making modifications

2) Prevent authorized users from making improper modifications (separation of duties)

3) Maintain internal and external consistency (well-formed transaction)

Clark-Wilson addresses each of these goals in its model. Biba only addresses the first goal.



Clark-Wilson Model

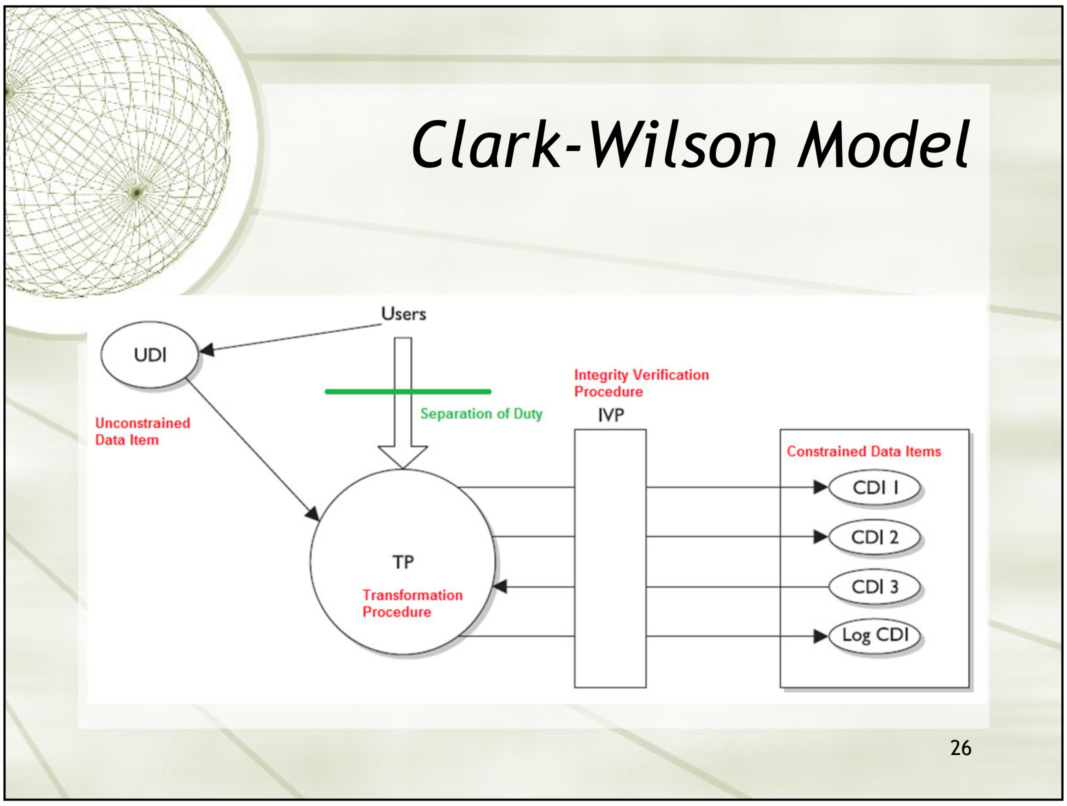
- Five elements in the model
  - Users
  - Transformation procedures (TPs)
  - Constrained data items (CDIs)
  - Unconstrained data items (UDIs)
  - Integrity verification procedures (IVPs)

25

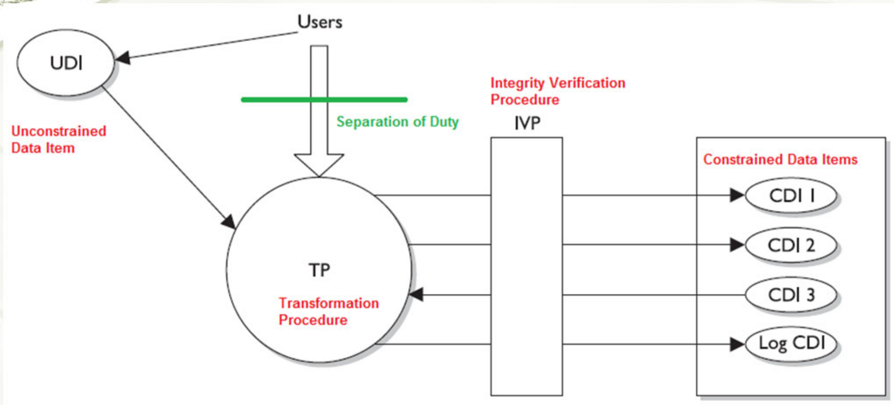
The Clark-Wilson Model

The Clark-Wilson model was developed after Biba and takes some different approaches to protecting the integrity of information. This model uses the following elements:

- Users Active agents
- Transformation procedures (TPs) Programmed abstract operations, such as read, write, and modify
- Constrained data items (CDIs) Can be manipulated only by TPs
- Unconstrained data items (UDIs) Can be manipulated by users via primitive read and write operations
- Integrity verification procedures (IVPs) Check the consistency of CDIs with external reality



Clark-Wilson Model



26

The Clark-Wilson Model

The Clark-Wilson model was developed after Biba and takes some different approaches to protecting the integrity of information. This model uses the following elements:

- Users Active agents
- Transformation procedures (TPs) Programmed abstract operations, such as read, write, and modify
- Constrained data items (CDIs) Can be manipulated only by TPs
- Unconstrained data items (UDIs) Can be manipulated by users via primitive read and write operations
- Integrity verification procedures (IVPs) Check the consistency of CDIs with external reality

Although this list may look overwhelming, it is really quite straightforward. When an application uses the Clark-Wilson model, it separates data into one subset that needs to be highly protected, which is referred to as a constrained data item (CDI), and another subset that does not require a high level of protection, which is called an unconstrained data item (UDI). Users cannot modify critical data (CDI) directly. Instead, the subject (user) must be authenticated to a piece of software, and the software procedures (TPs) will carry out the operations on behalf of the user. For example, when Kathy needs to update information held within her company's database, she will not be allowed to do so without a piece of software controlling these activities. First, Kathy must authenticate to a program, which is acting as a front end for the database, and then the program will control what Kathy can and cannot do to the information in the database. This is referred to as access triple: subject (user), program (TP), and object (CDI). A user cannot modify CDI without using a TP.

So, Kathy is going to input data, which is supposed to overwrite some original data in the database. The software (TP) has to make sure this type of activity is secure and will carry out the write procedures for Kathy. Kathy (and any type of subject) is not trusted enough to manipulate objects directly.

The CDI must have its integrity protected by the TPs. The UDI does not require such a high level of protection. For example, if Kathy did her banking online, the data on her bank's servers and databases would be split into UDI and CDI categories. The CDI category would contain her banking account information, which needs to be highly protected. The UDI data could be her customer profile, which she can update as needed. TPs would not be required when Kathy needed to update her UDI information.

In some cases, a system may need to change UDI data into CDI data. For example, when Kathy updates her customer profile via the web site to show her new correct address, this information will need to be moved into the banking software that is responsible for mailing out bank account information. The bank would not want Kathy to interact directly with that banking software, so a piece of software (TP) is responsible for copying that data and updating this customer's mailing address. At this stage, the TP is changing the state of the UDI data to CDI.

A model is made up of constructs, mathematical formulas, and other PhD kinds of stuff. The model provides the framework that can be used to build a certain characteristic into software (confidentiality, integrity, and so on). So the model does not stipulate what integrity rules the IVP must enforce; it just provides the framework, and the vendor chooses the integrity rules. The vendor implements integrity rules that its customer base needs the most. So if a vendor is developing an application for a financial institution, the UDI could be customer profiles that they are allowed to update and the CDI could be the bank account information, usually held on a mainframe. The UDI data do not need to be as highly protected and can be located on the same system or another system. A user can have access to UDI data without the use of a TP, but when the user needs to access CDI, they must use TP. So the vendor who develops the product will determine what type of data is considered UDI and what type of data is CDI and develop the TPs to control and orchestrate how the software enforces the integrity of the CDI values.

In a banking application, the IVP would ensure that the CDI represents the correct value. For example, if Kathy has $2,000 in her account and then deposits $50, the CDI for her account should now have a value of $2,050. The IVP ensures the consistency of the data. So after Kathy carries out this transaction and the IVP validates the integrity of the CDI (new bank account value is correct), then the CDI is considered to be in a consistent state. TPs are the only components allowed to modify the state of the CDIs. In our example, TPs would be software procedures that carry out deposit, withdrawal, and transfer functionalities. Using TPs to modify CDIs is referred to as a well-formed transaction.

A well-formed transaction is a series of operations that are carried out to transfer the data from one consistent state to the other. If Kathy transfers money from her checking account to her savings account, this transaction is made up of two operations: subtract money from one account and add it to a different account. By making sure the new values in her checking and savings accounts are accurate and their integrity is intact, the IVP maintains internal and external consistency. The Clark-Wilson model also outlines how to incorporate separation of duties into the architecture of an application. If we follow our same example of banking software, if a customer needs to withdraw over $10,000, the application may require a supervisor to log in and authenticate this transaction. This is a countermeasure to potential fraudulent activities. The model provides the rules that the developers must follow to properly implement and enforce separation of duties through software procedures.

Internal and external consistency is provided by the IVP, which ensures that what is stored in the system as CDI properly maps to the input value that modified its state. So if Kathy has $2,500 in her account and she withdraws $2,000, the resulting value in the CDI is $500.

To summarize, the Clark-Wilson model enforces the three goals of integrity by using access triple (subject, software [TP], object), separation of duties, and auditing. This model enforces integrity by using well-formed transactions (through access triple) and separation of user duties.