**CIA Triad**

CIA triad examples

To understand how the CIA triad works in practice, consider the example of a bank ATM, which can offer users access to bank balances and other information. An ATM has tools that cover all three principles of the triad:

* It provides **confidentiality** by requiring two-factor authentication (both a physical card and a PIN code) before allowing access to data
* The ATM and bank software enforce data **integrity** by ensuring that any transfers or withdrawals made via the machine are reflected in the accounting for the user's bank account
* The machine provides **availability** because it's in a public place and is accessible even when the bank branch is closed

But there's more to the three principles than just what's on the surface. Here are some examples of how they operate in everyday IT environments.

CIA triad confidentiality examples

Much of what laypeople think of as "cybersecurity" — essentially, anything that restricts access to data — falls under the rubric of confidentiality. This includes infosec's two big As:

* *Authentication,* which encompasses processes that allows systems to determine if a user is who they say they are. These include passwords and the panoply of techniques available for establishing identity: biometrics, security tokens, cryptographic keys, and the like.
* *Authorization,* which determines who has the right to access which data: Just because a system knows who you are, it doesn't necessarily open all its data for your perusal! One of the most important ways to enforce confidentiality is establishing need-to-know mechanisms for data access; that way, users whose accounts have been hacked or who have gone rogue can't compromise sensitive data. Most operating systems enforce confidentiality in this sense by having many files only accessible by their creators or an admin, for instance.

Public-key cryptography is a widespread infrastructure that enforces both As: by authenticating that you are who you say you are via cryptographic keys, you establish your right to participate in the encrypted conversation.

Confidentiality can also be enforced by non-technical means. For instance, keeping hardcopy data behind lock and key can keep it confidential; so can air-gapping computers and fighting against social engineering attempts.

A loss of confidentiality is defined as data being seen by someone who shouldn't have seen it. Big data breaches like the Marriott hack are prime, high-profile examples of loss of confidentiality.

Other examples include:

In the case of the payroll database of employees in an organization, only authorized employees to have access to the database. Additionally, within that group of authorized users, there could be more stringent limitations added on precise information that the group is allowed to access.

Another good example of confidentiality is the personal information of e-commerce customers. Sensitive information like credit card details, contact information, shipping details, or other personal information needs to be secured to prevent unauthorized access and exposure.

Violation of confidentiality can happen in many ways. It can occur through direct attacks, which are specifically designed to gain illegal access to systems, databases, applications, etc. For example, escalation of system privileges, network reconnaissance, electronic eavesdropping, man-in-the-middle attacks, etc. Human error can also be a reason for violation just as much as inadequate security measures.

Human errors include weak passwords; shared user accounts, shoulder surfing, no data encryption, poor, or absence of authentication systems, theft of physical equipment and storage devices. etc.

There are several countermeasures that can be taken to protect confidentiality. It includes data classification and labelling; strong authentication mechanisms, tight access controls, steganography, data encryption during a process, transit, and storage, remote wipe capabilities, and education and training on cybersecurity for all.

CIA triad integrity examples

The techniques for maintaining data integrity can span what many would consider disparate disciplines. For instance, many of the methods for protecting confidentiality also enforce data integrity: you can't maliciously alter data that you can't access, after all. We also mentioned the data access rules enforced by most operating systems: in some cases, files can be read by certain users but not edited, which can help maintain data integrity along with availability.

But there are other ways data integrity can be lost that go beyond malicious attackers attempting to delete or alter it. For instance, corruption seeps into data in ordinary RAM as a result of interactions with cosmic rays much more regularly than you'd think. That's at the exotic end of the spectrum, but any techniques designed to protect the physical integrity of storage media can also protect the virtual integrity of data.

Many of the ways that you would defend against breaches of integrity are meant to help you detect when data has changed, like data checksums, or restore it to a known good state, like conducting frequent and meticulous backups. Breaches of integrity are somewhat less common or obvious than violations of the other two principles, but could include, for instance, altering business data to affect decision-making, or hacking into a financial system to briefly inflate the value of a stock or bank account and then siphoning off the excess. A simpler — and more common — example of an attack on data integrity would be a defacement attack, in which hackers alter a website's HTML to vandalize it for fun or ideological reasons.

Examples:

For example, in e-commerce, customers expect products, pricing, and other related details to be accurate and that it will not be altered once the order is placed. Similarly, in banking, a sense of trust regarding banking information and account balances has to be established by ensuring that these details are authentic and have not been tampered with.

Basically, ensuring integrity involves protecting the data at all times—in use, in transit (sending an email, uploading or downloading files, etc.), and when stored in a storage device, data centre, or cloud.

Like confidentiality, integrity can be compromised in different ways. It can happen directly through the intrusion of detection systems, modification of configuration files, change of system logs to avoid detection) or human errors.

Countermeasures like encryption, digital signatures, hashing, and digital certificates can help maintain data integrity. Aside from these, intrusion detection systems, strong authentication mechanisms, version control, auditing, and access controls can ensure integrity.

It is a given that integrity also closely ties in with the concept of non-repudiation, which means that one will not be able to deny certain actions as being not true. For example, if an email with a digital signature was sent or received, the integrity will be maintained for these kinds of online transactions that happen.

CIA triad availability examples

Maintaining availability often falls on the shoulders of departments not strongly associated with cybersecurity. The best way to ensure that your data is available is to keep all your systems up and running, and make sure that they're able to handle expected network loads. This entails keeping hardware up-to-date, monitoring bandwidth usage, and providing failover and disaster recovery capacity if systems go down.

Other techniques around this principle involve figuring out how to balance the availability against the other two concerns in the triad. Returning to the file permissions built into every operating system, the idea of files that can be read but not edited by certain users represent a way to balance competing needs: that data be available to many users, despite our need to protect its integrity.

The classic example of a loss of availability to a malicious actor is a denial-of-service attack. In some ways, this is the most brute force act of cyberaggression out there: you're not altering your victim's data or sneaking a peek at information you shouldn't have; you're just overwhelming them with traffic so they can't keep their website up. But DoS attacks are very damaging, and that illustrates why availability belongs in the triad.

Availability can be compromised if there is a hardware or software failure, natural disasters, power failure, or human error. DDoS attacks are one of the more common reasons for the violation of availability.

Availability can be ensured through network, server, application, and service redundancy. Hardware fault tolerance in servers and storage is another good countermeasure to avoid violation of availability. DoS protection solutions, system upgrades, regular software patching, comprehensive disaster recovery plans, backups, etc. are all ways to ensure availability.

**State Machine Model**

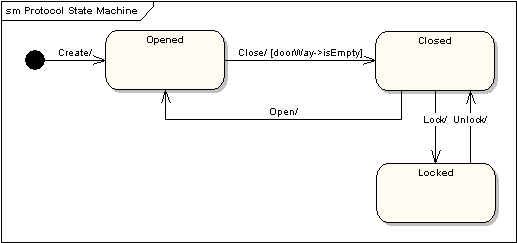
What is state machine modeling?

A state machine model is a mathematical model that groups all possible system occurrences, called states. Every possible state of a system is evaluated, showing all possible interactions between subjects and objects. If every state is proven to be secure, the system is proven to be secure.

State Machine Diagrams

A state machine diagram models the behaviour of a single object, specifying the sequence of events that an object goes through during its lifetime in response to events.

As an example, the following state machine diagram shows the states that a door goes through during its lifetime.

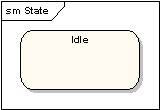


State Diagram

The door can be in one of three states: "Opened", "Closed" or "Locked". It can respond to the events Open, Close, Lock and Unlock. Notice that not all events are valid in all states; for example, if a door is opened, you cannot lock it until you close it. Also notice that a state transition can have a guard condition attached: if the door is Opened, it can only respond to the Close event if the condition doorWay->isEmpty is fulfilled. The syntax and conventions used in state machine diagrams will be discussed in full in the following sections.

States

A state is denoted by a round-cornered rectangle with the name of the state written inside it.

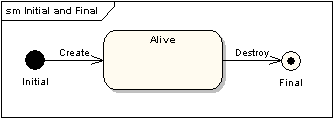


States

Initial and Final States

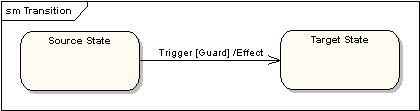
The initial state is denoted by a filled black circle and may be labeled with a name. The final state is denoted by a circle with a dot inside and may also be labeled with a name.

Initial and Final States



Transitions

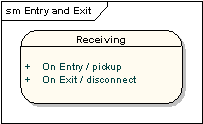
Transitions from one state to the next are denoted by lines with arrowheads. A transition may have a trigger, a guard and an effect, as below.



Transitions

"Trigger" is the cause of the transition, which could be a signal, an event, a change in some condition, or the passage of time. "Guard" is a condition which must be true in order for the trigger to cause the transition. "Effect" is an action which will be invoked directly on the object that owns the state machine as a result of the transition.

State Actions



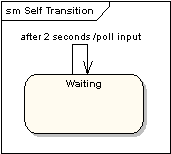
In the transition example above, an effect was associated with the transition. If the target state had many transitions arriving at it, and each transition had the same effect associated with it, it would be better to associate the effect with the target state rather than the transitions. This can be done by defining an entry action for the state. The diagram below shows a state with an entry action and an exit action.

State Actions

It is also possible to define actions that occur on events, or actions that always occur. It is possible to define any number of actions of each type.

Self-Transitions

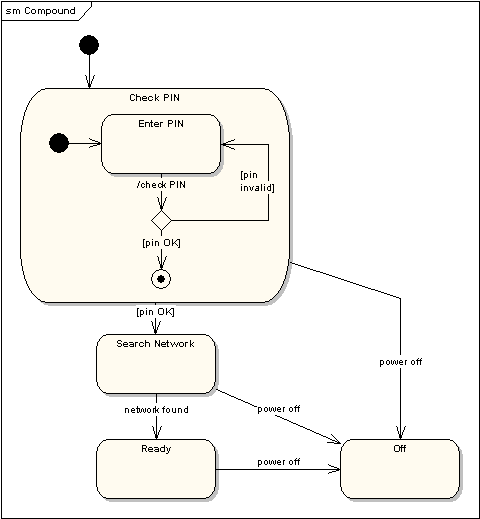
A state can have a transition that returns to itself, as in the following diagram. This is most useful when an effect is associated with the transition.



Self-Transitions

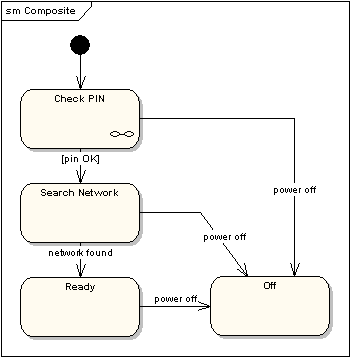
Compound States

A state machine diagram may include sub-machine diagrams, as in the example below.



Compound States

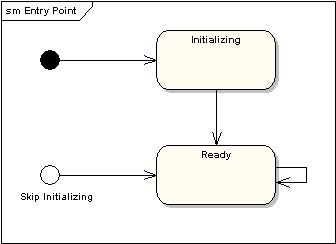
The alternative way to show the same information is as follows.



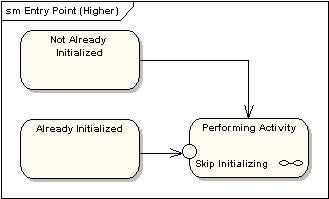
The notation in the above version indicates that the details of the Check PIN sub-machine are shown in a separate diagram.

Entry Point

Sometimes you won’t want to enter a sub-machine at the normal initial state. For example, in the following sub-machine it would be normal to begin in the "Initializing" state, but if for some reason it wasn’t necessary to perform the initialization, it would be possible to begin in the "Ready" state by transitioning to the named entry point.

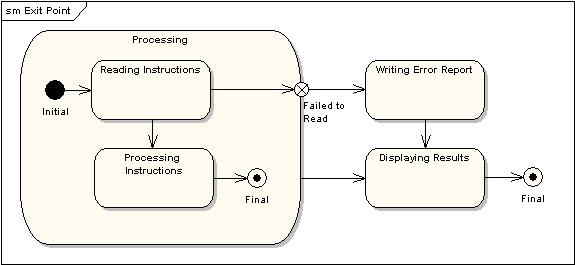


The following diagram shows the state machine one level up.



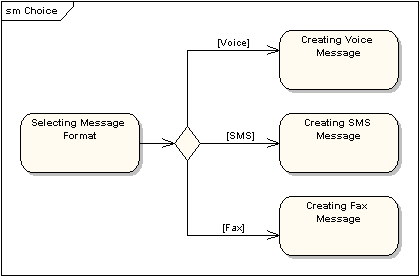
Exit Point

In a similar manner to entry points, it is possible to have named alternative exit points. The following diagram gives an example where the state executed after the main processing state depends on which route is used to transition out of the state.



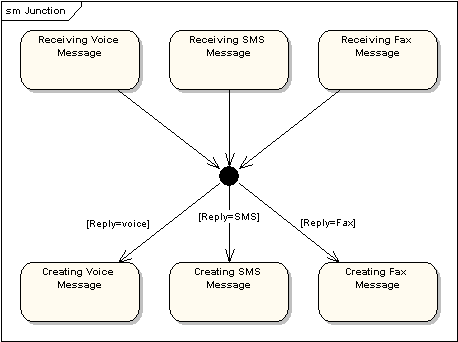
Choice Pseudo-State

A choice pseudo-state is shown as a diamond with one transition arriving and two or more transitions leaving. The following diagram shows that whichever state is arrived at, after the choice pseudo-state, is dependent on the message format selected during execution of the previous state.



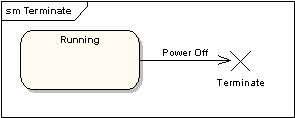
Junction Pseudo-State

Junction pseudo-states are used to chain together multiple transitions. A single junction can have one or more incoming, and one or more outgoing, transitions; a guard can be applied to each transition. Junctions are semantic-free. A junction which splits an incoming transition into multiple outgoing transitions realizes a static conditional branch, as opposed to a choice pseudo-state which realizes a dynamic conditional branch.



Terminate Pseudo-State

Entering a terminate pseudo-state indicates that the lifeline of the state machine has ended. A terminate pseudo-state is notated as a cross.

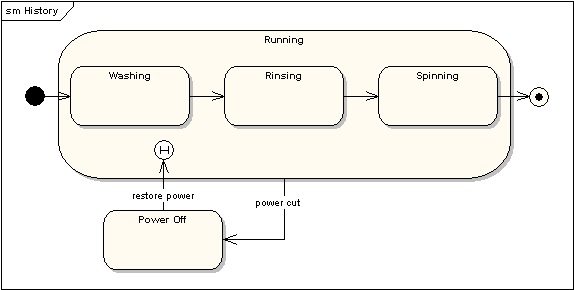


History States

A history state is used to remember the previous state of a state machine when it was interrupted. The following diagram illustrates the use of history states. The example is a state machine belonging to a washing machine.

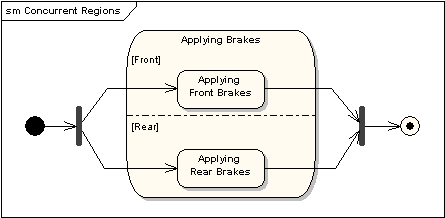
History State

In this state machine, when a washing machine is running, it will progress from "Washing" through "Rinsing" to "Spinning". If there is a power cut, the washing machine will stop running and will go to the "Power Off" state. Then when the power is restored, the Running state is entered at the "History State" symbol meaning that it should resume where it last left-off.



Concurrent Regions

A state may be divided into regions containing sub-states that exist and execute concurrently. The example below shows that within the state "Applying Brakes", the front and rear brakes will be operating simultaneously and independently. Notice the use of fork and join pseudo-states, rather than choice and merge pseudo-states. These symbols are used to synchronize the concurrent threads.



Source: <https://sparxsystems.com/resources/tutorials/uml2/state-diagram.html>

How State Machine Models work

In state-based models of testing, testers construct state machine models which attempt to model an application in terms of its runtime states. Test case generation, then, attempts to satisfy various coverage criteria defined on the state machine. Though referred to by various names in the literature, coverage of transitions or edges in the model is a commonly used criteria [12–14]. Note that coverage of transitions requires the consideration of a single-length history of prior state. The event-driven domain of protocol testing was in many respects a catalyst for state-based testing approaches in the early 1990s [17, 18], though protocols considered at that time were much simpler domains than the event-driven protocols of today's applications.

In practice, automated construction of state machine models of nontrivial applications is often infeasible, despite a number of enhancements which attempt to limit the number of states and transitions that must be captured by the model. Cheng and Krishnakumar describe the Extended Finite State Machine (EFSM), which limits states by adding additional counters and runtime components [19]. Similarly, Shehady and Siewiorek describe an extension of FSMs to use variables, which limits the number of states to be explored [20], and the work of White et al. suggests subdividing models into usage-specific categories [21, 22].

In practice, these techniques provide some utility in well-defined or highly critical domains, but would likely be ineffective in the more complex domains of larger event-driven systems. In addition to the high cost of model construction at the often hidden and complex level of application states, the task of mapping back to a concrete, executable input domain remains an error-prone manual exercise.

In contrast to control-flow models, data-flow models of software track context by following traces of variables in a program. A model known as a control-flow graph details the flow of a program through its executable statements. The primary indicators of context are the uses of variables. Variable reads and writes and their use in interesting combinations are analyzed. Rapps and Weyuker first introduced a set of adequacy criteria for data-flow testing [15], focusing primarily on coverage of their Def/Use program graph. Frankl and Weyuker later expanded on these original criteria, claiming that many programs could not satisfy the central criteria of the original work [16], for reasons as common as a simple for loop. In practice, once again, reliable construction of program graphs and generation of tests which satisfy these criteria do not scale to the complexities of modern applications.

Source: <https://www.sciencedirect.com/topics/computer-science/state-machine-model>

**Biba Model**

Simply put in common words

if a general wants to change a mission that's in progress, then he has the right to do that. That data has a high level of integrity. Therefore, everyone below him has permission to trust that information.

if a private sends a message about a change in the mission, then that data is not to be trusted. It has a low level of integrity. Therefore, no one above him has permission to trust that information.

Implementations

* In FreeBSD, the Biba model is implemented by the mac\_biba MAC (Mandatory Access Control) policy
* In Linux, the Biba model is implemented in the General Dynamics Mission Systems PitBull product.
* In XTS-400, the Biba model is implemented in the BAE Systems's XTS-400 operating system.

The Biba Integrity security model has been implementedin British Aerospace (BAE) Systems (XTS-400 operating systems), General Dynamics MissionSystems (Linux), and MacBook (FreeBSD)

FreeBSD

According to the FreeBSD handbook,

DESCRIPTION

In Biba environments, an “integrity” label is set on each subject or object. These labels are made up of hierarchal grades, and non-hierarchal components. As an object's or subject's grade ascends, so does its integrity.

Supported labels are biba/low, biba/equal, and biba/high; as explained below:

The biba/low label is considered the lowest integrity an object or subject may have. Setting this on objects or subjects will block their write access to objects or subjects marked high. They still have read access though.

The biba/equal label should only be placed on objects considered to be exempt from the policy.

The biba/high label will permit writing to objects set at a lower label, but not permit reading that object. It is recommended that this label be placed on objects that affect the integrity of the entire system.

Biba provides for:

* Hierarchical integrity level with a set of non hierarchical integrity categories;
* Fixed rules: no write up, no read down (opposite of MLS). A subject can have write access to objects on its own level or below, but not above. Similarly, a subject can have read access to objects on its own level or above, but not below;
* Integrity (preventing inappropriate modification of data);
* Integrity levels (instead of MLS sensitivity levels).

The following sysctl tunables can be used to manipulate the Biba policy.

security.mac.biba.enabled may be used to enable/disable enforcement of the Biba policy on the target machine.

* security.mac.biba.ptys\_equal may be used to disable the Biba policy on pty(4) devices.
* security.mac.biba.revocation\_enabled will force the revocation of access to objects if the label is changed to dominate the subject.

To access the Biba policy setting on system objects, use the setfmac and getfmac commands:

# setfmac biba/low test

# getfmac test

test: biba/low

Observations: a lower integrity subject is unable to write to a higher integrity subject; a higher integrity subject cannot observe or read a lower integrity object.

Drawbacks

However, its drawbacks include; it is hard to choose the appropriate procedures to be implemented since it allows for various policies to beused across different situations. It does not enforce information confidentiality. Lastly, it does notsupport authorization revocation. Biba Integrity model can be used for access control and enforcing data integrity.

**Bell-LaPadula Model**

Explain using simple example

Everybody gets a shirt. You give your absolute best friends shirts with a number 3, you give the people you like a shirt with a 2 and everyone else gets a 1.

When anyone passes a note in class they wrap it in an envelope with the same numbers, 1,2,3. There are two central rules,

anyone can open and read the contents of an envelope that has the same number or lower than their shirt

anyone can put a note in an envelope with a number higher than their shirt

This makes it so that people who have a 3 (your best friends) can read anybody's note but can't pass note down to 2 or 1s. This prevents them from possibly betraying your secret notes by rewriting one and passing it down. People with a 2 (people you like) can send you a note, but can't send a 1 a note. People with a 1 on their shirt (everyone else) can send notes to anyone but only read replies from other 1s.

Uses

As a high-assurance, MLS (Multi-level security) system, XTS-400 can be used in cross-domain solutions, which typically need a piece of privileged software to be developed which can temporarily circumvent one or more security features in a controlled manner. Such pieces are outside the CC evaluation of the XTS-400, but they can be accredited.

The XTS-400 can be used as a desktop, server, or network gateway. The interactive environment, typical Unix command line tools, and a GUI are present in support of a desktop solution. Since the XTS-400 supports multiple, concurrent network connections at different sensitivity levels, it can be used to replace several single-level desktops connected to several different networks.

In support of server functionality, the XTS-400 can be implemented in a rackmount configuration, accepts an uninterruptible power supply (UPS), allows multiple network connections, accommodates many hard disks on a SCSI subsystem (also saving disk blocks using a sparse file implementation in the file system), and provides a trusted backup/save tool. Server software, such as an Internet daemon, can be ported to run on the XTS-400.

A popular application for high-assurance systems like the XTS-400 is to guard information flow between two networks of differing security characteristics. Several customer guard solutions are available based on XTS systems.

Limitations

* Only addresses confidentiality, control of writing (one form of integrity), \*-property and discretionary access control
* Covert channels are mentioned but are not addressed comprehensively
* The tranquility principle limits its applicability to systems where security levels do not change dynamically. It allows controlled copying from high to low via trusted subjects. [Ed. Not many systems using BLP include dynamic changes to object security levels.]

**Clark-Wilson Model**

Basic Principles

According to Stewart and Chapple's CISSP Study Guide Sixth Edition, the Clark–Wilson model uses a multi-faceted approach in order to enforce data integrity. Instead of defining a formal state machine, the model defines each data item and allows modifications through only a small set of programs. The model uses a three-part relationship of subject/program/object (where program is interchangeable with transaction) known as a triple or an access control triple. Within this relationship, subjects do not have direct access to objects. Objects can only be accessed through programs. Look here to see how this differs from other access control models.

The model's enforcement and certification rules define data items and processes that provide the basis for an integrity policy. The core of the model is based on the notion of a transaction.

A well-formed transaction is a series of operations that transition a system from one consistent state to another consistent state.

In this model the integrity policy addresses the integrity of the transactions.

The principle of separation of duty requires that the certifier of a transaction and the implementer be different entities.

The model contains a number of basic constructs that represent both data items and processes that operate on those data items. The key data type in the Clark–Wilson model is a Constrained Data Item (CDI). An Integrity Verification Procedure (IVP) ensures that all CDIs in the system are valid at a certain state. Transactions that enforce the integrity policy are represented by Transformation Procedures (TPs). A TP takes as input a CDI or Unconstrained Data Item (UDI) and produces a CDI. A TP must transition the system from one valid state to another valid state. UDIs represent system input (such as that provided by a user or adversary). A TP must guarantee (via certification) that it transforms all possible values of a UDI to a “safe” CDI.

Clark-Wilson Model

Users.

It all starts with users, otherwise known as the subjects. The subjects which will access the objects. Users are the ones that need the information. I think the books call users the “active agents”.

Transformation Procedures (TPs)

Then we have Transformation Procedures. Think of them as operations the subject is trying to perform.

Is the subject trying to read a file?

Write to a file?

Or modify a file?

Transformation Procedures are simply operations which can be performed.

Constrained Data Items (CDIs)

There’s Constrained Data Items and Unconstrained Data Items.

Objects which belong in the subset of Constrained Data Items are at a higher level of protection.

In a Clark-Wilson Model, there are two types of protections given to data items, constrained and unconstrained.

In order to read an object located in the Constrained Data Items subset, we have to go through a transformation procedure.

Constrained Data Items can only be manipulated by a Transformation Procedure.

Objects within a Constrained Data Items are so valuable, that in a Clark-Wilson model, a subject has to go through an intermediary to even just access it.

Unconstrained Data Items (UDIs)

Now objects in an Unconstrained Data Item subset, well they probably aren’t that important.

These can be accessed by the subject directly, it doesn’t need to go through an intermediary like a Transformation Procedure. The subject can perform their own read and write operations without going through a Transformation Procedure.

Subjects access objects in a UDI like how they would normally access an object in a networked environment, as if they weren't using a Clark-Wilson Model. Like how we access files or objects in Windows operating systems.

Integrity Verification Procedures (IVPs)

What is an Integrity Verification Procedure?

Are they anything like Transformation Procedure?

Remember how we talked about internal and external consistency?

That’s what Integrity Verification Procedures do, they check the internal and external consistency.

IVPs are a way to audit the transformation procedure.

EXAM CONCEPTS

The Clark-Wilson Model enforces the concept of separation of duties.

This is done through the Integrity Verification Procedures.

When the IVP audits the well-formed transactions of the Transformation Procedures, the IVP is independently performing a duty separate from that of the Transformation Procedure.

Separation of duties is performed when the IVP audits the Transformation Procedure.

Makes sense right? It prevents the Transformation Procedure from auditing its own work, and instead has some other entity making sure true integrity is upheld.

So remember this one: Clark-Wilson implements a form of separation of duties in its access control methodology.

And actually, you can also see the Separation of Duties principle when the Transformation Procedure accesses the Constrained Data Item on behalf of the subject. Duties are separated because it isn’t the job of the subject to access the CDI, but rather belongs to the Transformation Procedure.

The Clark-Wilson model is trying to separate a subject completely from an object in a CDI through the use of an intermediary. This separates duties so the subject cannot access the object directly, as it is not part of the subject's duty.

Another exam concept as we’ve said, Clark-Wilson upholds integrity through the use of well-formed transactions in the process of using Transformation Procedures.

Unlike the Biba model which uses integrity levels based on classifications.

I also wanted to mention that the Clark-Wilson follows a trifecta for subject to object access.

Meaning it involves a subject, a program, and an object.

Or a subject, a middleman, and an object.

Shon Harris calls this the Access Triple, where you have the user, which is the subject, a program in the middle which is the Transformation Procedures, and finally the object, which is the Constrained Data Item.

And another exam concept is that a constrained data item cannot be manipulated without first going through the program. Clark-Wilson uses Integrity Verification Procedures to verify and audit all transactions are done in a consistent manner and as they are supposed to be done.

The Clark-Wilson model is almost an improvement over the Biba Model I think because in order to make sure authorized subjects don’t make unintentional changes, there is always a middle-man in between the object and subject

There is always a intermediary which makes sure that the transaction is solid, in other words, a well-formed transaction.