

Chapter 3

Mandatory Access Control

Mandatory access control (MAC) policies¹ are designed to support practices that institutions use to limit the damage that can be caused by individuals. Examples of such practices include using labels and clearances to control access, separation of duty, accounting controls such as double-entry bookkeeping, and the privileges implicit in role hierarchies found in larger organizations. By enforcing a MAC policy, we reduce the level of trust that must be placed in the individuals accessing an institution's computer systems and the software those systems execute. Enforcing a discretionary access control (DAC) policy would not suffice, because the authorization to access an object then would be controlled by the object's owner or by subjects whose authority can be traced back to that owner, but the goals of an institution might not align with the goals of those individuals.

3.1 MAC for Government

To prevent leaks of information, governments will authorize an individual to access only the information needed for the tasks that individual has been assigned. Such policies were first employed to protect military secrets, are now widely used throughout governments, but also have applications in commercial settings. This section² discusses two schemes for specifying and enforcing such policies for access by subjects to files, where a subject may be an individual or a program that individual has invoked. One scheme uses a partially ordered set of labels to specify authorizations; the other scheme replaces that partial order with tables specifying relations that might not be partial orders. Each scheme has been implemented in widely deployed systems.

¹The term *non-discretionary access control* is also sometimes used for these policies.

²Policies for information flow control also can be applicable. These are discussed in chapters 5 and 6 on information flow control.

3.1.1 Partially Ordered Labels

David Bell and Leonard LaPadula collaborated on this early and influential scheme to limit the propagation of file contents. The scheme—which we call BLP—uses a set Λ of labels and assumes (i) a fixed label $\lambda_F \in \Lambda$ has been assigned to each file F , (ii) a fixed label $\lambda_S \in \Lambda$ has been assigned to each subject S , and (iii) read and write accesses to files requested by subjects are the sole way that file contents propagates. Partial order \sqsubseteq on labels (where $\lambda \sqsubset \lambda'$ abbreviates $\lambda \sqsubseteq \lambda' \wedge \lambda \neq \lambda'$), specifies allowed and prohibited propagation of file contents to other files and to subjects, as follows.

BLP Policy. The contents of a file assigned label λ is authorized to propagate to a file or subject assigned label λ' only if $\lambda \sqsubseteq \lambda'$ holds. \square

For example, if Λ is $\{L, H\}$ with $L \sqsubset H$ then contents derived from files with label H (“high”) may not be copied to files or read by subjects assigned label L (“low”). So we prevent leaks from files containing secret information if those files are given label H and if files that can be made public are given label L . BLP Policy can protect file integrity, as well. To prohibit information in untrusted files from being copied into trusted files, the files storing untrusted information are given label H and the files storing trusted information are given label L .

BLP Policy is enforced when reads and writes to files comply with the following *BLP rules*.

Simple Security Condition. Subject S assigned label λ_S is authorized to read from file F assigned label λ_F only if $\lambda_F \sqsubseteq \lambda_S$ holds. \square

***-Property.** Subject S assigned label λ_S is authorized to write into file F assigned a label λ_F only if $\lambda_S \sqsubseteq \lambda_F$ holds. \square

The Simple Security Condition implies that λ_S is the upper bound on the labels for files that a subject S is authorized to read, and the *-Property implies that λ_S is the lower bound on the labels for files that S is authorized to write. So S is prohibited from “reading up” and from “writing down” relative to its label λ_S and partial order \sqsubseteq .

Notice, the *-Property prevents subjects from subverting the Simple Security Condition by copying from one file to another.³ Therefore, the BLP rules block Trojan horse⁴ attacks that circumvent access controls by causing a program to

³Suppose some subject S can read from file F and write into file F' . According to the Simple Security Condition and the *-Property, $\lambda_F \sqsubseteq \lambda_S \sqsubseteq \lambda_{F'}$ must hold so, by transitivity, $\lambda_F \sqsubseteq \lambda_{F'}$ holds. A subject S' could subvert the prohibitions on reading content from F if S' cannot read F but could read the copied content from F' . However, the existence of such a subject S' would lead to a contradiction, as follows, so we conclude that S' cannot exist. The Simple Security Condition and the *-Property imply $\lambda_{F'} \sqsubseteq \lambda_{S'} \sqsubseteq \lambda_F$, which implies $\lambda_{F'} \sqsubset \lambda_F$. We earlier concluded that $\lambda_F \sqsubseteq \lambda_{F'}$ holds, but that conclusion contradicts also having $\lambda_{F'} \sqsubset \lambda_F$ hold, since we are assuming that \sqsubseteq is a partial order.

⁴A *Trojan horse* attack is a program that an individual invokes because it appears useful, but the program also implements hidden and nefarious functionality. Greek mythology recounts how the 10-year Greek siege of Troy was ended by a clever subterfuge. The Greeks

copy contents from a file that an attacker cannot read to a file the attacker can read.

The $*$ -Property, however, does allow a subject S to update a file that S is not authorized to read. Such updates are called *blind writes*. Blind writes are best avoided, because a subject cannot subsequently perform a read in order to check whether a blind write actually was performed. We can prohibit blind writes by strengthening the $*$ -Property to require that $\lambda_S = \lambda_F$ hold (rather than requiring that $\lambda_S \sqsubseteq \lambda_F$ hold) in order for a subject S to update a file F .

3.1.1.1 Labels for Multilevel Security

Confidentiality. For access to files stored online, the U.S. Department of Defense (DoD) employs a MAC policy derived from their policy for controlling access to paper documents. The MAC policy authorizes a user's file access requests based on (i) the file's content, (ii) whether the access would facilitate the content being leaked, and (iii) the damage such a leak might cause. The policy is implemented by the BLP Policy in conjunction with a partial order on *multilevel security* (MLS) labels.

Each *MLS label* comprises a pair $\langle \mathcal{T}, \mathcal{C} \rangle$, with \mathcal{T} and \mathcal{C} having a different interpretation for file labels than for subject labels.

MLS Labels for Files. These MLS labels are assigned by *classification authorities* that are knowledgeable about the subjects covered in the file and that understand the broader context necessary for predicting possible damage from leaking the contents of the file.

- \mathcal{T} is a set of topic names. The information contained in the labeled file is limited to these topics. Topic names come from a catalog that has been adopted by the community using these labels.⁵
- \mathcal{C} is the labeled file's *sensitivity*. It gives an upper bound on the potential damage that could be caused if the file contents leaks. Figure 3.1 gives the terms that DoD uses, along with the meaning of each.

MLS Labels for Subjects. Subjects make requests on behalf of users, so a subject's MLS label is the MLS label assigned to the user instigating

built a huge wooden horse, hid a small force of warriors inside, placed the horse outside the gates of Troy, and then appeared to abandon the siege by sailing out of sight. With the Greek force gone, the Trojans opened the city gates and moved the horse—thought to be a tribute marking the end of the siege—inside. But once the sun had set, the Greek fleet turned around and headed back to Troy. At midnight, the Greek warriors inside the horse emerged, killed the Trojan guards, and opened the city gates. The Greek force, which by then had returned, entered the open gates and destroyed the city, thereby winning the war.

⁵For MLS labels used within DoD, topic names might be self-explanatory (e.g., *chem/bio*, *crypto*, or *nuclear*) or obscure (e.g., *Ultra* or *Umbra*). Obscure topic names are used so that people who see a label but do not have a need to know will remain ignorant about what the label describes. With DoD labels, names whose meanings are secret are called *codewords*. For example, the codeword *Ultra* was used during World War II to label information that the Allies obtained by decrypting intercepts of German communications, and *Umbra* is a more recent (but also now-retired) codeword for the most-sensitive kinds communications intercepts.

\mathcal{C}	file sensitivity: potential damage	user clearance: trustworthiness
TS (<u>T</u> op <u>S</u> ecret)	exceptionally grave	strong
S (<u>S</u> ecret)	serious	moderate
C (<u>C</u> onfidential)	some	somewhat
U (<u>U</u> nclassified)	none	unknown

Figure 3.1: Interpretations for \mathcal{C} in an MLS label $\langle \mathcal{T}, \mathcal{C} \rangle$, where $U < C < S < TS$ and $X \leq Y$ denotes $X = Y \vee X < Y$

the execution that is making the request. A user’s MLS label might be assigned by the user’s employer or by some external agency.

- \mathcal{T} is a set of topic names. These topics cover all content relevant to the user’s current position or task assignments.
- \mathcal{C} is the user’s *clearance*. The user is believed to be at least this trustworthy. Figure 3.1 gives the categories that DoD uses for user clearances and how each is interpreted. A user clearance is presumed to predict whether the individual will leak or corrupt the contents of files.⁶

To be using fixed labels, we must ensure that if the MLS labels initially assigned to files and users are accurate characterizations then these labels will remain so during any execution satisfying the Simple Security Condition and the *-Property. The BLP Policy allows propagation from a file with label λ to a subject or file with label λ' only if $\lambda \sqsubseteq \lambda'$ holds, so we define the partial order \sqsubseteq on MLS labels as follows to ensure that an MLS label λ' remains accurate after a read or write.

$$\langle \mathcal{T}, \mathcal{C} \rangle \sqsubseteq \langle \mathcal{T}', \mathcal{C}' \rangle: \mathcal{T} \subseteq \mathcal{T}' \wedge \mathcal{C} \leq \mathcal{C}' \quad (3.1)$$

To show that definition (3.1) of \sqsubseteq preserves the accuracy of MLS labels, we establish that propagation allowed by $\langle \mathcal{T}, \mathcal{C} \rangle \sqsubseteq \langle \mathcal{T}', \mathcal{C}' \rangle$ never transfers contents to any subject⁷ or file with MLS label $\langle \mathcal{T}', \mathcal{C}' \rangle$, where

- (i) the topics are not covered by \mathcal{T}' and
- (ii) the sensitivity of the contents exceeds \mathcal{C}' .

⁶Clearances are typically granted after an individual has submitted to a background investigation that seeks to identify character flaws or exploitable personal circumstances. This background investigation might range from a short interview with the individual to a set of interviews with the individual’s family and friends as well as a polygraph test.

⁷This analysis assumes that increasing the number of people who know a secret does not increase the risk of leaks, which probably is not a valid assumption about the general population. Benjamin Franklin is reported to have written: “Three can keep a secret if two of them are dead”. However, the assumption could be reasonably sound for a set of individuals who have been vetted by background investigations.

Assume $\langle \mathcal{T}_F, \mathcal{C}_F \rangle$ is the label on a file F and $\langle \mathcal{T}_S, \mathcal{C}_S \rangle$ is the label for a subject S . Propagation of file contents occurs because subjects read files and/or because subjects write files.

S reads F. $\langle \mathcal{T}_F, \mathcal{C}_F \rangle \sqsubseteq \langle \mathcal{T}_S, \mathcal{C}_S \rangle$ must hold for S to be authorized to read F . So we conclude that $\mathcal{T}_F \subseteq \mathcal{T}_S$ and $\mathcal{C}_F \leq \mathcal{C}_S$ hold due to definition (3.1) for \sqsubseteq . All content in F is covered by topic list \mathcal{T}_F . From $\mathcal{T}_F \subseteq \mathcal{T}_S$, we conclude that content also is covered by topic list \mathcal{T}_S , so requirement (i) is satisfied. From $\mathcal{C}_F \leq \mathcal{C}_S$, we conclude that the user associated with S is believed to be at least as trustworthy as the least trustworthy user that is authorized to read F . So requirement (ii) is satisfied.

S writes to F. $\langle \mathcal{T}_S, \mathcal{C}_S \rangle \sqsubseteq \langle \mathcal{T}_F, \mathcal{C}_F \rangle$ must hold for S to be authorized to write into F . So we conclude that $\mathcal{T}_S \subseteq \mathcal{T}_F$ and $\mathcal{C}_S \leq \mathcal{C}_F$ hold due to definition (3.1) for \sqsubseteq . From $\mathcal{T}_S \subseteq \mathcal{T}_F$ we have that any information S writes is covered by topic list \mathcal{T}_F because it is covered by \mathcal{T}_S , so requirement (i) is satisfied. To discharge (ii), observe that a subject S_R that can read F must have a clearance \mathcal{C}_R satisfying $\mathcal{C}_F \leq \mathcal{C}_R$. Thus, by transitivity with $\mathcal{C}_S \leq \mathcal{C}_F$ from $\langle \mathcal{T}_S, \mathcal{C}_S \rangle \sqsubseteq \langle \mathcal{T}_F, \mathcal{C}_F \rangle$, we conclude that $\mathcal{C}_S \leq \mathcal{C}_R$ must hold. So requirement (ii) is satisfied.

Integrity. *Multilevel integrity* (MLI) labels are used to ensure that content from low-integrity files and/or content from untrusted users is not written into files purported to store high-integrity content. Each MLI label comprises a pair $\langle \mathcal{T}, \mathcal{I} \rangle$ where \mathcal{T} and \mathcal{I} have different interpretations for file labels than for subject labels.

MLI Labels for Files.

- \mathcal{T} is a set of topic names. The information contained in the labeled file is limited to these topics.
- \mathcal{I} is the file's *criticality*. It gives an upper bound on the level of damage that could be caused if the file contents gets corrupted.⁸ Figure 3.2 suggests a list of possible categories, along with their definitions.

MLI Labels for Subjects. A subject label is the MLI label for the user initiating the execution making the request.

- \mathcal{T} is a set of topic names. These topics cover all content relevant to the user's current position or task assignments.
- \mathcal{I} is the user's *clearance*. A user's clearance is presumed to predict whether the individual will leak or corrupt the contents of files.

As with MLS labels, the definition of partial order \sqsubseteq on MLI labels is based on preserving the accuracy of these labels if execution causes file contents to propagate when $\langle \mathcal{T}, \mathcal{I} \rangle \sqsubseteq \langle \mathcal{T}', \mathcal{I}' \rangle$ holds. That is, propagation allowed by

⁸Be cautioned that higher values for \mathcal{I} indicate lower levels of integrity.

\mathcal{I}	criticality of content:	user clearance: trustworthiness
0	crucial	strong
1	very important	moderate
2	important	unknown

Figure 3.2: Interpretations for \mathcal{I} in an MLS integrity label $\langle \mathcal{T}, \mathcal{I} \rangle$

$\langle \mathcal{T}, \mathcal{I} \rangle \sqsubseteq \langle \mathcal{T}', \mathcal{I}' \rangle$ must never transfer contents to any user or file with an MLI label $\langle \mathcal{T}', \mathcal{I}' \rangle$, where

- (i) the topics are not covered by \mathcal{T}' and
- (ii) the integrity of the content does not satisfy \mathcal{I}' .

Requirement (i) is equivalent to $\mathcal{T} \subseteq \mathcal{T}'$, and requirement (ii) is equivalent to $\mathcal{I} \leq \mathcal{I}'$. So \sqsubseteq definition (3.1) works for MLI labels as well as working for MLS labels.

3.1.1.2 Trusted Subjects

Execution that complies with BLP Policy is constrained by the axioms that a partial order must satisfy. These constraints can be problematic. Consider, for example, a subject S that encrypts a confidential file txt by using the secret key in file key , writing the ciphertext output to file out . For S to read files key and txt , the Simple Security Condition requires that $\lambda_{key} \sqsubseteq \lambda_S$ and $\lambda_{txt} \sqsubseteq \lambda_S$ hold. For S to write into out , the *-Property requires that $\lambda_S \sqsubseteq \lambda_{out}$ hold. So, by transitivity, $\lambda_{key} \sqsubseteq \lambda_{out}$ and $\lambda_{txt} \sqsubseteq \lambda_{out}$ also hold.

However, a subject S' that the Simple Security Condition prohibits from reading files txt or key ought to be allowed to read ciphertext output file out , because ciphertext output reveals nothing about the plaintext input or the encryption key. That would mean $\lambda_{out} \sqsubseteq \lambda_{S'} \sqsubseteq \lambda_{key}$ and $\lambda_{out} \sqsubseteq \lambda_{S'} \sqsubseteq \lambda_{txt}$ hold and (by transitivity) $\lambda_{out} \sqsubseteq \lambda_{key}$ and $\lambda_{out} \sqsubseteq \lambda_{txt}$ would also hold. A label $\lambda_{S'}$ that satisfies those conditions cannot exist if \sqsubseteq is a partial order, given that we established previously that $\lambda_{key} \sqsubseteq \lambda_{out}$ and $\lambda_{txt} \sqsubseteq \lambda_{out}$ must hold, too. So we cannot have an instance of BLP Policy where ciphertext output file out can be read by subjects that cannot read the input files txt and key . There are also other examples where systems need to perform “write downs” that are safe because the content being written is carefully controlled.⁹

The BLP Policy prohibition of “write downs” is thus problematic. A solution is to introduce *trusted subjects*, which are subjects that are not required to comply with the BLP rules. Trusted subjects are thus allowed to perform “write downs”. In the encryption scenerio described above, if TS is designated as a

⁹One example is a server that receives requests involving classified information and then replies with an acknowledgment or it writes unclassified status information to a public log. Another example is an election system that reads secret votes and writes the majority to a public file.

trusted subject then TS , by fiat, would be allowed to read txt and key as well as to write file out .

Trusted subjects that misbehave can cause great harm, though. Therefore, a trusted subject should not be incorporated into a system without first considering whether restructuring that system might eliminate the need for that trusted subject. Also, a trusted subject should not be employed without first getting assurance about its behaviors. So it is better if the trusted subject is limited to executing a small and simple piece of code (because large programs are difficult to analyze) and if the system has only a few trusted subjects.

3.1.2 Type Enforcement

Type enforcement (TE) policies specify authorizations for subjects to perform operations on objects, including other subjects. Each subject is in a *domain*; each object is in a *type*. Privileges authorize the subjects in a given domain to perform operations on the objects in a type and on the subjects in a domain according to the following tables.

- *Domain definition table* $ddt[\cdot, \cdot]$ has a row for each domain and a column for each type. Entry $ddt[\mathcal{D}, \mathcal{T}]$ for a row \mathcal{D} and a column \mathcal{T} gives the privileges that subjects in domain \mathcal{D} are granted for operations on objects in type \mathcal{T} . Examples of privileges for types that comprise files might include r (for reading), w (for writing), and x (for executing).
- *Domain interaction table* $dit[\cdot, \cdot]$ has a row and a column for each domain. Entry $dit[\mathcal{D}, \mathcal{D}']$ for a row \mathcal{D} and a column \mathcal{D}' gives the privileges that subjects in domain \mathcal{D} are granted for operations on subjects in domain \mathcal{D}' . Examples of such privileges might include x (for transferring control to a subject), i (for interrupting execution of a subject), and d (for terminating the subject).

The domain definition table and the domain interaction table together specify an authorization relation \mathcal{A}_{TE} that relates a subject S , an object O , and a privilege p as follows.

$$\langle S, O, p \rangle \in \mathcal{A}_{TE} \quad \text{if and only if} \quad S \in \mathcal{D} \wedge O \in \mathcal{T} \wedge ddt[\mathcal{D}, \mathcal{T}] = p$$

$$\langle S, S', p \rangle \in \mathcal{A}_{TE} \quad \text{if and only if} \quad S \in \mathcal{D} \wedge S' \in \mathcal{D}' \wedge dit[\mathcal{D}, \mathcal{D}'] = p$$

Figure 3.3(a) is an example of a domain definition table that (among other things) gives subjects in domain \mathcal{D}_L privileges r and w for files in type \mathcal{F}_L and privilege w for files in type \mathcal{F}_H . Figure 3.3(b) is an example of a domain interaction table that (among other things) gives subjects in domain \mathcal{D}_H privilege x authorizing control transfers to domain \mathcal{D}_{Enc} and within \mathcal{D}_H .

Systems that support TE often provide a specialized language that a system administrator will use to populate the entries of the domain definition table and the domain interaction table. The language provides statements to define the system's domains by giving a name for each domain and by listing the entry

domain	type			
	$\mathcal{F}_{\text{keys}}$	\mathcal{F}_{H}	\mathcal{F}_{L}	...
\mathcal{D}_{L}		w	r, w	
\mathcal{D}_{H}		r, w	r	
\mathcal{D}_{Enc}	r	r, w	r, w	
⋮				

(a) *ddt* for Encryption Use

domain	domain			
	\mathcal{D}_{L}	\mathcal{D}_{H}	\mathcal{D}_{Enc}	...
\mathcal{D}_{L}	x			
\mathcal{D}_{H}		x	x	
\mathcal{D}_{Enc}		x	x	
⋮				

(b) *dit* for Encryption Use

Figure 3.3: Trust Enforcement Policy for Using Encryption

points for binaries that subjects in each given domain are authorized to execute. The language also provides statements to define the system's types by giving a name for each type and listing the files and/or other objects that the type includes. Finally, there would be a way to specify the privileges that are assigned to each domain for each type.

TE can be used to specify a broad range of MAC policies. The effects of the BLP rules (page 112) for a set Λ of labels with partial order \sqsubseteq can be achieved using TE by defining a domain \mathcal{D}_{λ} and a type \mathcal{T}_{λ} for each label $\lambda \in \Lambda$. A subject S with label λ_S would be in domain \mathcal{D}_{λ_S} , a file F with label λ_F would be in type \mathcal{T}_{λ_F} , and the domain definition table would satisfy:

$$ddt[\mathcal{D}_{\lambda}, \mathcal{T}_{\lambda'}] = \mathbf{r} \quad \text{if and only if } \lambda' \sqsubseteq \lambda \text{ holds.} \quad (3.2)$$

$$ddt[\mathcal{D}_{\lambda}, \mathcal{T}_{\lambda'}] = \mathbf{w} \quad \text{if and only if } \lambda \sqsubseteq \lambda' \text{ holds.} \quad (3.3)$$

The Simple Security Condition is satisfied because (3.2) asserts that a subject S having label λ_S is authorized to read a file F having label λ_F only if $\lambda_F \sqsubseteq \lambda_S$ holds. The *-Property is satisfied because (3.3) requires that $\lambda_S \sqsubseteq \lambda_F$ hold in order for subject S to write file F .

TE also can be used to specify MAC policies¹⁰ that do not comply with the BLP rules. Since authorization relation \mathcal{A}_{TE} is not constrained to represent a partial order, authorization is not required to be transitive. That means a subject S could be authorized to read file F , propagation of information from file F' to F allowed, but S not be authorized to read F' —a configuration that restricts S to accessing the content of F' through intermediary F . Moreover, a special category of trusted subjects is not required with TE, because any domain can be given any privileges. So the privileges assigned to each subject can be limited to those that are consistent with the Principle of Least Privilege.

As an illustration, Figure 3.3 gives a TE specification for the policy in §3.1.1.2 for the encryption of secrets that will be stored in a public file.

- For subjects in domains \mathcal{D}_{L} and \mathcal{D}_{H} , the Simple Security Condition and the *-Property are implied by the *ddt*: domain \mathcal{D}_{L} does not have an **r**

¹⁰Besides the examples we give here, the MAC policies for supporting commerce discussed in §3.2 are examples of such policies.

privilege for files in type \mathcal{F}_H , and domain \mathcal{D}_H does not have a w privilege for files in type \mathcal{F}_L .

- Subjects in domains \mathcal{D}_L and \mathcal{D}_H are not permitted to read or write files in type F_{keys} containing the cryptographic keys, but subjects in domain \mathcal{D}_{Enc} are permitted to read those files.
- Subjects in \mathcal{D}_{Enc} are permitted to violate the BLP rules by reading files in type \mathcal{F}_H and writing files in type \mathcal{F}_L . These subjects, however, are not permitted to write files in types F_{keys} .

To justify granting subjects in domain \mathcal{D}_{Enc} the privileges for reading from \mathcal{F}_H files and writing to \mathcal{F}_L files, though, the programs that these subjects execute should be scrutinized. We would want assurance that these programs cannot be subverted to reveal secret keys from files in F_{keys} or content from files in \mathcal{F}_H by writing into files in \mathcal{F}_L .

TE also can be used to mandate that information reaching a subject or an object must be first validated or transformed, resulting in an *assured pipeline*. We might write

$$S_1 \rightsquigarrow F_1 \rightsquigarrow S_2 \rightsquigarrow \cdots \rightsquigarrow F_{n-1} \rightsquigarrow S_n$$

to indicate that the outputs of each stage S_i are directed to file F_i and that the sole input to a stage S_i is the file F_{i-1} . If each stage S_i is a subject in a different domain \mathcal{D}_i and each file F_i is in a different type \mathcal{F}_i then these restrictions are specified if

$$ddt[\mathcal{D}_i, \mathcal{F}_i] = w \wedge ddt[\mathcal{D}_{i+1}, \mathcal{F}_i] = r \wedge dit[\mathcal{D}_i, \mathcal{D}_{i+1}] = x$$

holds for $1 \leq i < N$ and no other privileges are assigned to these domains.

One use for an assured pipeline would be to ensure that content being written to a disk is encrypted. The disk driver (presumed to be the only way to perform disk operations) would be the final stage of the assured pipeline, an encryption routine would be an earlier stage of that pipeline, and making an update to the disk would require transferring content to the assured pipeline's first stage. Another use of an assured pipeline arises in settings where each text page that is printed must include a banner identifying the authorized readers for that page—a useful feature for classified documents. In such an assured pipeline, the printer driver (presumed to be the only way to have something printed) would be the final stage of that assured pipeline. An earlier stage would be a program that inputs a file, divides that file into pages, adds the appropriate banner into each page, and forwards that sequence of pages to a successor stage.

3.2 MAC for Commerce

On-line records about a company's assets and liabilities are the basis for many business decisions. Dishonest individuals in this setting perpetrate fraud by

performing bogus updates to these records. Fraud, however, is as old as commerce itself, and the accounting profession long ago developed defenses, known as *financial controls*. They include:

- *Separation of Duty*. By requiring several individuals to participate, collusion becomes necessary in order to perform or subvert a transaction.
- *Prescribed Transformation Procedures*. Damage can be limited if specific programs offer the only way for an individual to update certain records.
- *Mandated Audit*. If each update is logged in a way that is immutable and identifies the accountable individual then attackers are deterred for fear of detection and because logs provide incriminating evidence for use in prosecution.

Notice that DAC's owner-control of authorization is incompatible with Separation of Duty. Also, defining a partial order on labels for individuals and for data items is not helpful for implementing any of these defenses. So the needs of commerce involve new kinds of MAC policies.

3.2.1 Separation of Duty

Fraud occurs when trusted individuals abuse their authority. Separation of duty defends against such abuses by limiting the authority granted to any one individual. With only limited authority, a dishonest individual working alone can cause only limited damage; a set of dishonest individuals must collaborate in order to perpetrate bigger frauds. The chances that one dishonest employee would be able to find others who would participate then can be much reduced by careful attention to hiring.

With a *static* separation of duty, the authority given to each individual is fixed and pre-determined. This approach is often adopted by companies where job titles are linked to sets of tasks. For example, individuals in the Billing Department who generate customer invoices would be authorized to update the accounts-receivable database but not authorized to update the database storing warehouse inventory, whereas the opposite would hold for warehouse workers. Static separation of duty has two shortcomings. First, attackers are able to identify the specific individuals to recruit for committing a fraud. Second, there is limited flexibility for reassigning work when a business must cope with unusual demand or employee absences.

In *dynamic* separation of duty, the authority assigned to an individual is neither pre-determined nor fixed. The assignment might be random, or it might depend on state or history. A software development group, for example, might follow a code check-in regime that requires every module any programmer changes to be audited by a different programmer—a separation of duty for updates versus audits. If the auditor is selected based on who is the least overworked, then we have state-based dynamic separation of duty; if a programmer who has (or has not) previously contributed to the updated module must be selected, then we have history-based dynamic separation of duty.

Randomness can be used to good advantage in dynamic separation of duty. The absence of advance knowledge about who will participate in a given transaction can frustrate attempts by attackers to recruit collaborators.¹¹ Banks and other financial institutions, for instance, require employees to be away from the office (at training or on vacation) annually, for an uninterrupted interval (typically one or two weeks). A randomly-selected peer is given the absent employee's workload for that period. The required absence is, by design, long enough that the temporary substitute would see some irregularities if fraud was being committed. And use of random selection makes it unlikely that the temporary substitute would already be in collusion with the absent employee.

Separation of duty might be tied to objects and/or tied to activities. In either case, enforcement could require that information about task assignments be available and, if history is involved, that it be preserved for future use.

- When separation of duty is tied to an object, then that object provides an obvious place for recording task assignments. For example, code check-in requires separation of duty for the programmer and the auditor of each module. The file storing the module's source code or that file's meta-data would be a natural place for recording the names of programmers and auditors involved in each a change.
- Activities typically are transient and, therefore, do not offer places for long-term storage of information about task assignments. New objects might have to be created for this purpose, and these objects might have to be stored indefinitely if separation of duty is based on history. Garbage collection of these objects now becomes an issue.

The enforcement of separation of duty for program execution initiated by a human user requires the system to authenticate that user. Authentication of users is useful only if different user identifiers actually do correspond to different individuals. We are thus making assumptions about whether the authentication protocol can be spoofed and about whether attributes are validated by that protocol when new individuals are enrolled as users.

Delegation of authority from one principal to another also brings complications, since distinct internal identifiers could now speak for the same individual. One option is to prohibit delegation from/to principals that are constrained by separation of duty. A second option is for each request to carry the identifier of the principal making the request as well as carrying the identifiers of all principals that delegated authority to that requester. And a third option is to monitor delegations and create a central database that records equivalences that delegations induce among the identifiers being used.

Finally, occasions where one user informally recruits another to share a workload can be tricky to handle while still enforcing separation of duty. Such delegations are a form of collusion if they are hidden from the enforcement mechanism. However, sharing a workload is quite natural and often encouraged.

¹¹A group is unlikely to collude unless the members already trust each other. It takes time to establish such trust. So attackers need long lead times to recruit collaborators.

One employee might lend a hand to another, or a manager might stand-in for a subordinate. A separation of duty policy can be formulated to support such delegations, provided they are made known to the enforcement mechanism and, therefore, appropriate authority assignments still can be made.

Chinese Wall Policies. A *conflict of interest* occurs for someone who is engaged in multiple activities if serving the best interests of one of those activities might not be in the best interests of another. For example, a consultant advising one company would have a conflict of interest if that consultant was also advising a competitor. But accepted¹² practice sometimes allows competitors to be advised by different employees from the same firm if the following *Chinese Wall*¹³ policy is being enforced: No employee of the firm is allowed to access information from two or more client companies that are competitors.

A Chinese Wall policy specifies a separation of duty. Initially, an employee would have accessed information from no client company and, therefore, is allowed to access information from any client company. Thereafter, authorization to access information from a client company will depend on the information the employee has already accessed. The separation of duty is thus dynamic and based on history.

Enforcement. Assume that a separate collection of files is maintained about each client company. Subjects correspond to executing programs that read and/or update these files, each subject associated with a different user.

- A *subject source set* I_S contains the names of the companies that provided the information to the collections of files that S has read. Initially $I_S = \emptyset$ holds.
- A *company source set* I_C contains the names of the companies that provided the information a subject could learn from reading the files in the collection associated with a company C . Initially $I_C = \{C\}$ holds.

The effect of a subject S reading from the files in the collection maintained about a client company C causes the contents of subject source set I_S to be increased, because we assume (conservatively) that S internalizes the contents of all the files:

$$I_S := I_S \cup I_C \tag{3.4}$$

¹²Acceptance is by no means universal. In the United States, investment banks do have clients that are competitors, but law firms do not.

¹³This name alludes to the Great Wall of China, which was built to protect the northern border of China. The Great Wall of China is roughly 5500 miles long, comprising segments that are above-ground wall (earth, stones, and wood), trenches, and geologic barriers (rivers and mountains). The earliest segments were built in the 7th century B.C.E. These were later connected in the 3rd century B.C.E.

C	files in collection
UA	u_1, u_2, u_3
DL	d_1, d_2, d_3
HILTON	h_1, h_2
OMNI	o_1, o_2, o_3, o_4
WESTIN	w_1, w_2, w_3, w_4

Figure 3.4: File Collections for Client Companies

The effects of a subject S updating a file associated with a client company C causes the contents of company source set I_C to be increased, because we assume (conservatively) that what S writes might reflect anything that S has internalized and, therefore, could have come from any file that S previously read:

$$I_C := I_C \cup I_S \quad (3.5)$$

Notice that (3.5) could cause $C' \in I_C$ to hold where $C' \neq C$.

Define a set of companies to be *cw-compliant* if that set does not contain companies that are competitors. So the Chinese Wall policy is enforced if, for each subject S , we ensure that I_S remains cw-compliant throughout execution.

Chinese Wall Enforcement (Compliance). Block a subject S from reading a file in the collection associated with a company C if $I_S \cup I_C$ is not cw-compliant. \square

To see this enforcement mechanism in action, we consider a sequence of accesses to the files listed in Figure 3.4 for client companies in two industry classes: airlines (UA and DL) and hotels (HILTON, OMNI, WESTIN). The first line in the sequence below gives the initial values for the subject and company source sets; each subsequent line gives an action and any updated subject and company source sets that result from executing that action. An access has a strike-thru if that access is rejected by the enforcement mechanism.

	access	I_S	$I_{S'}$	I_{UA}	I_{DL}	I_{HILTON}
	initial value	\emptyset	\emptyset	{UA}	{DL}	{HILTON}
1	S reads u_2	{UA}				
2	S reads h_2	{UA, HILTON}				
3	S reads d_3					
4	S' reads d_3		{DL}			
5	S writes h_2					{UA, HILTON}
6	S' reads h_1					
7	S writes d_1				{DL, UA, HILTON}	
8	S reads d_3					

The read at line 1 changes I_S to {UA} because subject S has read a file provided by UA. The Chinese Wall policy prohibits S from later reading files provided

by competitors to UA, and we see this enforced at line 3 for the attempt by S to read DL file d_3 . The enforcement mechanism rejects this read because $I_S \cup I_{DL}$ is not cw-compliant—it contains competitors $UA \in I_S$ and $DL \in I_{DL}$. The attempt at line 6 by S' to read h_1 is similarly rejected, since $I_{S'} \cup I_{HILTON}$ contains competitors $DL \in I_{S'}$ and $UA \in I_{HILTON}$ and, therefore, would not be cw-compliant.

At line 8, the attempt by S to read d_3 is rejected because $DL \in I_{DL}$ and $UA \in I_{DL}$, so $I_S \cup I_{DL}$ would not be cw-compliant. In fact, after the write by S at line 7, no subject can read any file from the collection associated with DL. Such a lock-out can be avoided by blocking any update that invalidates cw-compliance for a company source set.

Chinese Wall Enforcement (No Lock-Out). Block a subject S from updating files or adding files to the collection associated with a company C if $I_S \cup I_C$ is not cw-compliant. \square

One last form of potentially problematic behavior remains possible. The enforcement mechanisms presented thus far allow one subject's writes to block another subject from repeating a previously successful read. Here is an example:

	action	I_S	$I_{S'}$	I_{UA}	I_{DL}	I_{HILTON}
	initial value	\emptyset	\emptyset	{UA}	{DL}	{HILTON}
1	S reads h_2	{HILTON}				
2	S reads d_3	{DL, HILTON}				
3	S' reads u_1		{UA}			
4	S' writes h_2					{UA, HILTON}
5	S reads h_2					

Subject S reads HILTON file h_2 (at line 1) but is later (at line 5) blocked from rereading that file. The blocking occurs because S reads DL file d_3 (at line 2), so when line 5 is reached $I_S \cup I_{HILTON}$ is no longer cw-compliant— $UA \notin I_{HILTON}$ holds at line 1, but $UA \in I_{HILTON}$ holds at line 5. In blocking the read at line 5, the enforcement mechanism is being conservative. It is preventing file h_2 from serving as a channel that gives S indirect access to information about client company UA. That information could have been written into h_2 at line 4 by S' . Since S has been accessing client company DL, getting information about competitor UA would violate the Chinese Wall policy.

The way to prevent files from serving as illicit channels is to further restrict which companies each subject can access. To do this, we group sets of companies that are not competitors into *conglomerates*, with each conglomerate considered a competitor to all of the other conglomerates. A Chinese Wall policy enforced for access to the conglomerates does not block attempts to re-read files.¹⁴

¹⁴To see why this works, consider a partition Π_i . If every subject S that writes to files for companies in Π_i is restricted to reading files for companies in Π_i then throughout execution we have $I_S \subseteq \Pi_i$ and $I_C \subseteq \Pi_i$ for every $C \in \Pi_i$. That implies $(I_S \cup I_C) \subseteq \Pi_i$ holds throughout execution. By construction, Π_i does not contain competitors, so it and its subsets are cw-compliant. Therefore, subset $I_S \cup I_C$ is cw-compliant. So the enforcement mechanisms described above will not block a read or write operation.

Chinese Wall Enforcement (Allow Rereading). Partition the client companies into disjoint sets where no partition contains companies that are competitors. Restrict each subject to reading and/or writing only the companies in a single partition. \square

For example, the companies in Figure 3.4 might be partitioned into three conglomerates: $\{\text{UA, HILTON}\}$, $\{\text{DL, OMNI}\}$, and $\{\text{WESTIN}\}$. A subject that reads files from UA then also would be authorized to read files from HILTON but not from DL, OMNI, or WESTIN. And a subject that reads files from UA and/or HILTON will not be subsequently blocked from rereading files from those companies.

3.2.2 Transformation Procedures and Consistency

An *external consistency constraint* is an assertion that links the information a computer system stores to the actual values of those quantities in the physical world. For example, Widgets-R-Us might define the following external consistency constraint for the number of widgets it has available for sale.

$$nw = w_S + \sum_{1 \leq i \leq N} w_i \quad (3.6)$$

If w_S is the number of widgets stored in the Widgets-R-Us warehouse and w_i is the number of widgets in stock at distributor i of N then (3.6) asserts that the Widgets-R-Us computer system's variable nw is an accurate count of the actual number of widgets that Widgets-R-Us has available to sell.

A company's computing system can manipulate only the variables that it stores in its memory and files. However, it is reasonable to expect that some company employee would observe or participate in any event that changes the values of any other variables mentioned in an external consistency constraint. In (3.6), for example, variable nw would be changed by the computer system, but the values of variables w_S and w_i change in response to events like transferring some widgets from the warehouse to a distributor (which changes w_S and a w_i). By mandating *business processes* that employees must follow, we can ensure that a computer system's variables do get updated as necessary to keep external consistency constraints satisfied.

External Consistency Preservation. To maintain the validity of external consistency constraints, follow business processes that ensure:

- Every event in the physical world that could invalidate an external consistency constraint will be observed by some employee.
- The business process requires that employee to run an appropriate program when such an event is observed.
- That program performs appropriate updates to the computing system's variables. \square

Attackers don't necessarily follow mandated business processes, though. So a company's business processes ought to include periodic audits that validate each of the external consistency constraints. During an audit, an independent third party first gathers evidence about the state of the physical world. In this phase, the auditor conducts local inspections, such as counting inventory in the warehouse. This phase also has the auditor contact customers, creditors, and banks, to learn about outstanding orders, pending deliveries, and bank balances. The auditor then uses this information to identify differences with the values being stored by the computer system. Finally, the auditor determines the reasons for any differences, and the information in the computer system is adjusted if necessary.

A business process defines restrictions on the sequences of steps that employees perform for their jobs. One of those steps might involve invoking a program. In the computer security literature, the term *transformation procedure* is used for such a program. Execution of a transformation procedure TP will read and/or write some objects—typically files—and we require that it also record details about its invocation (who invoked TP , when, and what updates were made) in a write-only tamper-proof log file. The log file enables implementing the Mandated Audit financial control discussed at the start of §3.2.

Trustworthy employees perform their jobs properly and invoke transformation procedures as appropriate. Untrustworthy employees, however, might attempt to perpetrate a fraud by invoking a transformation procedure when it is not appropriate. To help defend against such fraud, we restrict which employees are allowed to invoke each transformation procedure, and we restrict which objects and operations can be accessed during such an invocation. These restrictions can be specified as an authorization relation \mathcal{A}_{com} having elements of the form¹⁵

$$\langle \text{EmplSet}, TP, \text{Obj}_1\{op_1^1, \dots, op_1^{m_1}\}, \dots, \text{Obj}_N\{op_N^1, \dots, op_N^{m_p}\} \rangle$$

to indicate that only employees in EmplSet are authorized to execute transformation procedure TP , and this execution of TP is allowed access only to objects $\text{Obj}_1, \dots, \text{Obj}_N$ by performing operations $op_i^1, \dots, op_i^{m_i}$ on each object Obj_i . Enforcement of the authorization policy specified by \mathcal{A}_{com} presumes a computer system having

- a means to authenticate each employee that attempts to invoke a transformation procedure, and
- a reference monitor¹⁶ that uses this authenticated identity to restrict execution of transformation procedures according to \mathcal{A}_{com} .

¹⁵This relation could be represented using type enforcement's domain definition table. Domains would be pairs $\langle emp, TP \rangle$, where emp is an employee and TP is a transformation procedure; types would be objects. Thus, $op \in ddt[\langle E, TP \rangle, Obj]$ would authorize an employee E to invoke a transformation procedure TP that performs an operation op on object Obj .

¹⁶Chapter 12 gives a detailed treatment of reference monitors.

Notice, static separation of privilege can be supported by having *EmplSet* contain disjoint sets of employees.

A reference monitor is, by definition, tamperproof. Assume this integrity guarantee extends to whatever representation is being used for \mathcal{A}_{com} . To ensure compliance with business processes, though, we must also protect the integrity of the programs implementing the transformation procedures. A standard approach is to use separation of duty.

Independent Certification. No employee who is authorized to execute transformation procedures is authorized to certify or modify the code that implements a transformation procedure. No employee who is authorized to write or modify a transformation procedure is authorized to certify that transformation procedure. \square

Collusion now becomes necessary to create a bogus transformation procedure and cause it to be executed.

Double-Entry Bookkeeping. The financial state of a business can be described using a set of *accounts*. Every account stores an initial balance and a sequence of *postings*. A posting $\langle amt, id \rangle$ is a pair that gives an amount *amt* to change the account's balance and gives an identifier *id* that is associated with the employee who made the posting. With *double-entry bookkeeping*, the balances of a company's accounts always satisfy an invariant¹⁷

$$\sum_{i \in Q} i = \sum_{i \in A \cup I} i - \sum_{i \in L \cup E} i \quad (3.7)$$

where Figure 3.5 defines categories *A*, *E*, *I*, *L*, and *Q* that accountants traditionally use for grouping accounts.¹⁸ Notice, adding a single (non-zero) posting to only one account will falsify (3.7). The need to add two or more postings (usually to different accounts) is what leads to the name double-entry bookkeeping.

When double-entry bookkeeping is used, allowable sequencings for the transformation procedures in a business process can be mandated by having each transformation procedure store information about which transformation procedures are next eligible to execute. To enforce the sequence $tp_1 tp_2 \dots tp_n$, each transformation procedure tp_i would be designed to block or terminate with an error if some prespecified Boolean condition $pre(tp_i)$ is *false* when tp_i is invoked. Condition $pre(tp_i)$ would achieve this effect by checking the states of certain specified set of accounts; execution of tp_i then adds postings to accounts in a way that falsifies $pre(tp_i)$ and makes $pre(tp_{i+1})$ *true*, so only tp_{i+1} can next be executed. Postings in accounts are thus serving as a token that is required

¹⁷An accountant would know (3.7) as the *accounting equation*.

¹⁸Assets and liabilities refer to all aspects of a company's current financial position. This includes inventories of supplies, unpaid bills, accounts payable (including outstanding purchase orders and worker time-sheets), accounts receivable (including unpaid customer invoices), the cash disbursements journal (i.e., list of checks issued), the cash receipts journal (i.e., list of deposits), and so on.

Set	Class	Informal Description
A	assets	things that add value to the business
E	expenses	expenditures over some specified period
I	income	income over that same period
L	liabilities	things that reduce the value of the business
Q	equity	overall value of the business

Figure 3.5: Classes of Accounts in Double-entry Bookkeeping

for execution and that gets passed from one transformation procedure tp_i in a sequence to its successor tp_{i+1} in that sequence.

To make this concrete, we give transformation procedures to implement a business process for acquiring inventory. The code for these transformation procedures uses two new statement types: **post** and **check**.

- A **post** statement

$$post \langle +cost, id \rangle \text{ to } L\text{-}AcntPay$$

causes the *posting* $\langle +cost, id \rangle$ to be appended to the sequence of postings associated with account $L\text{-}AcntPay$. Identifier id enables this posting to be linked with other postings and to the employee who is responsible for the posting. We adopt the convention that account names include a prefix to indicate a class from Figure 3.5, so it is easy to see that no transformation procedure falsifies invariant (3.7).

- A **check** statement either executes a **then** clause or an **else** clause, depending on whether some specified Boolean condition evaluates to *true*. That Boolean condition can introduce new identifiers, which then can be referenced within the scope of the **then** and will have the values of the most recent posting that satisfied the Boolean condition. For example, in

$$\mathbf{check} \langle \hat{c}, id \rangle \in A\text{-}ShpExp \wedge \langle \hat{c}, id \rangle \notin A\text{-}Invtry \mathbf{then} \dots$$

variables \hat{c} and id when execution of the **then** starts have initial values that correspond to some prior posting $\langle \hat{c}, id \rangle$ to $A\text{-}ShpExp$ that is not also a posting to $A\text{-}Invtry$.

The steps in the business process for inventory acquisition are: place an order, receive delivery, and pay the invoice for goods received. For the initial step, an employee invokes a transformation procedure $genPO$, specifying the *item* to purchase, a source *supplier*, and an amount *cost* that will be paid.

```

genPO: transproc(item, supplier, cost, id)
      post  $\langle +cost, id \rangle$  to  $L\text{-}AcntPay$ 
      post  $\langle +cost, id \rangle$  to  $A\text{-}ShpExp$ 
      send order number id for item to supplier
end genPO

```

Execution of *genPO* creates a liability, because the goods will ultimately have to be paid for—this liability is reflected by the posting to *L-AcntPay*. However, the expectation for ultimate delivery is an asset, reflected in the posting to *A-ShpExp*. Identifier *id* is thereafter associated with this purchase order and expected to appear in the paperwork that accompanies the delivery and in the supplier’s invoice.

Eventually the item is delivered, accompanied by a packing slip that references *id*. The employee on the loading dock who accepts this delivery would invoke transformation procedure *orderRcvd*.

```

orderRcvd: transproc(id)
    check  $\langle \hat{c}, id \rangle \in A\text{-ShpExp} \wedge \langle \hat{c}, id \rangle \notin A\text{-Invtry}$  then
        post  $\langle -\hat{c}, id \rangle$  to A-ShpExp
        post  $\langle +\hat{c}, id \rangle$  to A-Invtry
    else exception(“Unsolicited or duplicate delivery”)
end orderRcvd

```

Execution of *orderRcvd* checks *A-ShpExp* to ensure that the delivery was expected and checks *A-Invtry* to make sure that the delivery is not a duplicate. A decrement is then posted to *A-ShpExp* because the shipment is no longer expected, and an increment is posted to *A-Invtry* because inventory has been increased.

Receipt of the invoice for the goods with identifier *id*, causes an employee in the payments department to invoke transformation procedure *invRcvd* to pay the supplier.

```

invRcvd: transproc(supplier, cost, id, chkNo)
    check  $\langle +cost, id \rangle \in L\text{-AcntPay}$ 
     $\wedge \langle +cost, id \rangle \in A\text{-Invtry}$ 
     $\wedge \langle -cost, id \rangle \notin L\text{-AcntPay}$ 
    then
        post  $\langle -cost, id \rangle$  to L-AcntPay
        post  $\langle -cost, chkNo \rangle$  to A-ChkAcnt
        send check chkNo for cost to supplier re invoice id
    else exception(“Not delivered, not ordered, or paid once”)
end GenCheck

```

This transformation procedure validates that a purchase order was issued (by checking $\langle +cost, id \rangle \in L\text{-AcntPay}$), the delivery has been received (by checking $\langle +cost, id \rangle \in A\text{-Invtry}$), and the invoice has not yet been paid (by checking $\langle -cost, id \rangle \notin L\text{-AcntPay}$). A posting to *L-AcntPay* then cancels the liability of an unpaid invoice; a posting to *A-ChkAcnt* records an equivalent expense.

Notice that *orderRcvd* is enabled due to the **check** for a posting in *A-ShpExp* not also appearing in *A-Invtry*. And *invRcvd* is enabled due to the **check** for postings to *L-AcntPay* and *A-Invtry* but not also appearing in *L-AcntPay*. So a sequence—*genPO*, *orderRcvd*, *invRcvd*—is imposed on the execution order for transformation procedures that all concern the same value of *id*.

Also note (at least) two separate transformation procedures are always involved in any update to an account. If no employee is authorized to execute more than one of the transformation procedures $genPO$, $orderRcvd$, and $invRcvd$ then two or more employees would have to collude in order to embezzle funds or inventory. So we posit that sets $Emps_{Gen}$, $Emps_{Deliv}$, and $Emps_{Pay}$ of employees be disjoint with the following authorizations.

$$\begin{aligned} &\langle Emps_{Gen}, genPO, L-AcntPay\{\mathbf{post}\}, A-ShpExp\{\mathbf{post}\} \rangle \\ &\langle Emps_{Deliv}, orderRcvd, A-ShpExp\{\mathbf{post}, \mathbf{check}\}, A-Invtry\{\mathbf{post}, \mathbf{check}\} \rangle \\ &\langle Emps_{Pay}, invRcvd, A-Invtry\{\mathbf{check}\}, L-AcntPay\{\mathbf{post}, \mathbf{check}\} \rangle \end{aligned}$$

3.3 Role-based Access Control

Responsibilities and authority in enterprises and institutions are often associated with *roles*. A role might be equated with being assigned to a particular job title, project, client, or some combination. The role grants privileges. These privileges authorize only those actions expected from a role's occupants. This set of actions is presumed to be relatively fixed for a given role, so a fixed set of privileges can be associated with each role. Authorization schemes we have discussed thus far, which decide access based on user identity, obviously could be used to implement such a policy. But the administration of privilege-assignments to users based on identity is cumbersome when privileges are in support of roles and change only when a user's role changes. So identity-based access-control schemes are not well suited for enterprise and institutional settings; roles are a better basis for authorization.

With *role-based access control* (RBAC), all access requests are made during a *session*. Each session S speaks for a set $\rho(S)$ of *roles* occupied by the user $\mu(S)$ who instigated that session. A system that implements RBAC will typically provide commands to begin or end a session and, from within a session, to enter or exit a specified role. For example, a graduate student in Computer Science, after being authenticated as user EK, might begin a session S (say), enter role `studentCS6110` and then also enter role `graderCS5430`, so $\mu(S) = \text{EK}$ and $\rho(S) = \{\text{studentCS6110}, \text{graderCS5430}\}$ would hold.

The privileges associated with a session S are constrained by two sets.

- $UserRoles(U)$ is the set of roles that a user U is authorized to occupy.
- $RolePrivs(R)$ is the set of privileges granted to occupants of role R .

One constraint is that $UserRoles(\cdot)$ restricts $\rho(S)$ based on $\mu(S)$:

$$\rho(S) \subseteq UserRoles(\mu(S))$$

A second constraint is that during a session S , user $\mu(S)$ is granted privileges in $RolePrivs(R)$ for every $R \in \rho(S)$. For example, $RolePrivs(\text{studentCS6110})$ might include privileges granting read access to lecture notes (but not homework solution sets) for course CS6110, whereas $RolePrivs(\text{graderCS5430})$ might

include privileges that grant write access to lecture notes and read access to solutions sets for CS5430.

RBAC privileges might authorize low-level operations, such as reads or writes to particular files. Or the privileges might authorize high-level operations that bundle access by some given program to specific objects, as required by transformation procedures (§3.2.2).

Role Hierarchy and Privilege Inheritance. An organization's structure might imply that the occupants of one role are also entitled to the privileges granted by some other role. For example, an individual occupying an **officer** role in many clubs is entitled to all of the privileges that the **member** role grants. RBAC provides the reflexive and transitive *role inheritance* relation¹⁹ \sqsubseteq to specify when privileges associated with one role are also granted to occupants of another role: $R \sqsubseteq R'$ specifies that occupants of role R' receive all privileges granted by role R —whether or not the occupant is even authorized to occupy role R . The following defines a set $\rho^*(S)$ containing all of the roles that grant privileges to the user $\mu(S)$ operating within a session S .

$$\rho^*(S) = \rho(S) \cup \{R' \mid R \in \rho(S) \wedge R' \sqsubseteq R\}$$

The set $\text{privs}(S)$ of privileges that a session S grants to user $\mu(S)$ is then:

$$\text{privs}(S) = \bigcup_{R \in \rho^*(S)} \text{RolePrivs}(R) \quad (3.8)$$

For example, an RBAC scheme implemented for Cornell University might specify role inheritance relations

$$\text{cornellian} \sqsubseteq \text{CUstudent} \quad \text{and} \quad \text{cornellian} \sqsubseteq \text{CUstaff}$$

to specify that occupants of the **CUstaff** and **CUstudent** roles also receive privileges (e.g., access to the library, parking, and cafeterias) granted to occupants of the **cornellian** role (a label for all current members of the university community), making it unnecessary for $\text{RolePrivs}(\text{CUstaff})$ and $\text{RolePrivs}(\text{CUstudent})$ to explicitly list any of the privileges in $\text{RolePrivs}(\text{cornellian})$.

Not only might multiple roles inherit privileges from a single role, but a single role might inherit privileges from multiple roles. Managers are often authorized to do anything that their immediate subordinates are authorized to do. If **emp₁**, ..., **emp_n** are the roles occupied by individuals supervised by the occupant of role **mngr**, then specifying role inheritance relations

$$\text{emp}_1 \sqsubseteq \text{mngr}, \dots, \text{emp}_n \sqsubseteq \text{mngr}$$

would ensure that **mngr** occupants are granted the needed privileges.

There are situations, though, where a manager should inherit some but not all of the privileges granted to subordinates. For example, the programmers in

¹⁹So RBAC gives a different meaning to symbol \sqsubseteq used in §3.1.1 as a partial order on BLP labels.

Term	Informal Description
$\rho(S)$	set of roles entered in session S
$\rho^*(S)$	set of roles effectively occupied in session S
$\mu(S)$	user who instigated session S
Act	set of all sessions currently active
$privs(S)$	set of privileges granted in session S
$R \sqsubseteq R'$	role R' inherits privileges granted by role R
$Roles$	set of all roles
$RolePrivs(R)$	the set of privileges granted by role R
$Users$	set of all users
$UserRoles(U)$	set of roles user U is authorized to occupy

Figure 3.6: Terms for Formulating RBAC Constraints

a group should have write privileges for the code repository, but whether their manager also should have that privilege depends on whether that manager is a capable programmer. To specify that a role R' inherits some—but not all—of privileges associated with another role R we could employ a *selective role inheritance* relation $R \sqsubseteq_P R'$ that adds only the privileges in $RolePrivs(R) \cap P$ to the privileges granted to the occupants of role R .

Role inheritance relation \sqsubseteq is not strictly necessary. We could instead manually copy the privileges from $RolePrivs(R)$ to $RolePrivs(R')$ for each role R satisfying $R \sqsubseteq R'$. The implicit granting of privileges through role inheritance is invaluable for system administration, though. When $R \sqsubseteq R'$ holds, a change to $RolePrivs(R)$ not only changes what privileges are granted to occupants of role R but automatically changes what privileges are granted to occupants of all roles R' satisfying $R \sqsubseteq R'$. This effect is impossible to achieve by analyzing and updating the $RolePrivs(R')$ sets. Yes, an administrator could check whether R' satisfies $RolePrivs(R) \subseteq RolePrivs(R')$, but $RolePrivs(R) \subseteq RolePrivs(R')$ might hold either because $R \sqsubseteq R'$ holds or because roles R and R' coincidentally authorize overlapping privileges. Without knowing whether a role inheritance relation $R \sqsubseteq R'$ is intended, a system administrator would not know whether to change $RolePrivs(R')$ when changes to $RolePrivs(R)$ are made.

Constraints. In RBAC, separation of duty and other policies that restrict mutual occupancy of roles are specified by giving *RBAC constraints*. An RBAC constraint can be any Boolean expression formulated using the terms given in Figure 3.6; an action is prohibited from executing if that execution would cause an RBAC constraint to be falsified.

The RBAC constraint to specify the separation of duty policy that any user authorized to occupy role R is not authorized to occupy R' and *vice versa* is:

$$(\forall U \in Users: R \notin UserRoles(U) \vee R' \notin UserRoles(U)) \quad (3.9)$$

A weaker separation of duty policy is defined by RBAC constraint

$$(\forall S \in Act: R \notin \rho(S) \vee R' \notin \rho(S)) \quad (3.10)$$

since it excludes users from occupying roles R and R' within a single session but does not prevent a user from occupying roles R and R' in separate, concurrent sessions; the following RBAC constraint does prevent that.

$$(\forall S, S' \in Act: \mu(S) = \mu(S') \Rightarrow (R \notin \rho(S) \vee R' \notin \rho(S'))) \quad (3.11)$$

Notice that (3.11) achieves the desired effect only if $\mu(S) \neq \mu(S')$ implies that sessions S and S' speak for different individuals. The validity of that assumption depends on how individuals are being authenticated and on the accuracy of the enrollment protocol employed to register new users.

Role inheritance creates a complication for separation of duty policies. If there is role inheritance then separation of duty policies formulated in terms of role occupancy do not necessarily impose separation of duty for privileges. Separation of duty for privileges can be achieved by adding RBAC constraints, though. For example, we specify that a privilege p is granted to only a single user at any time (without identifying what user or restricting which privileges various roles grant) with the following RBAC constraint.

$$\neg(\exists S, S' \in Act, R, R' \in Roles: \mu(S) \neq \mu(S') \wedge R \in \rho^*(S) \wedge R' \in \rho^*(S') \wedge p \in RolePrivs(R) \wedge p \in RolePrivs(R')) \quad (3.12)$$

Besides specifying separation of duty policies, RBAC constraints offer a way to incorporate user attributes or system state into authorization. The following RBAC constraint restricts occupancy in a role R to normal working hours

$$(\forall S \in Act: R \in \rho(S) \Rightarrow 0900 \leq time() \leq 1700) \quad (3.13)$$

if function $time()$ evaluates to the current time. And given a function $locate(U)$ that evaluates to the current location of a user U , the following RBAC constraint restricts occupancy in R to those users working at the office (versus, say, at home or at an Internet cafe).

$$(\forall S \in Act: R \in \rho(S) \Rightarrow locate(\mu(S)) = office) \quad (3.14)$$

Just because a constraint is easy to specify does not mean it would be easy to check. To check a constraint might require detecting some set of relevant events and informing a runtime whenever these events occur. Certain events are straightforward to detect because they cause the operating system to be invoked. These events include:

- system administrator commands to change the sets of users or roles, the roles each user is authorized to occupy, the privileges associated with each role, and the role inheritance relation;

- system calls that allow a user to begin/end a session or to enter/exit a specified role within a session.

Moreover, we might reasonably expect that these events would be the sole means for causing any of the terms in Figure 3.6 to change value.

A reference monitor incorporated into the operating system could be used to support RBAC constraints (3.9)–(3.12). Enforcement becomes expensive, though, if an RBAC constraint depends on functions (e.g., *time()* and *locate()*) that change value undetectably or too frequently for the reference monitor to be invoked. Although (3.13) could be enforced by scheduling timer-interrupts that invoke the reference monitor at times 0900 and 1700, other RBAC constraints involving the passage of time might require more frequent checking than would be practical. And enforcement of RBAC constraints involving location might well be completely infeasible.

Some Practical Considerations. RBAC is best suited for settings in which

- there are many more users than roles, and
- the role hierarchy and privileges assigned to each role change slowly, even if the role assignments for users change relatively rapidly,

because then assigning privileges to roles involves less administrative effort than directly assigning privileges to individual users. Therefore, RBAC is well suited for use in mature industries and institutions, because these tend to have a stable set of roles, where each role has fixed responsibilities and authority. But RBAC is not well suited for settings where the number of users is comparable to the number of roles (i.e., small organizations), many users should be assigned the same set of privileges (i.e., flat organizations), or privilege assignments to roles change frequently in unanticipated ways (i.e., highly dynamic organizations).

RBAC also does not work well in settings where a user's authority does not derive entirely from that user's role(s) in the institution. The role of doctors in a hospital illustrates. We might be tempted to define a single **doctor** role for authorizing access to patient records. However, each doctor should be authorized to access only the records for that doctor's patients. So a separate role could be needed for each doctor, which diminishes the value of using roles for authorization. A bank with multiple branches is another example. Every branch manager has the same responsibilities, but the branch manager for each branch should have access only to the records associated with that branch. Defining a single role for all branch managers then does not provide sufficient selectivity.

As just illustrated, a roles explosion makes RBAC expensive to administer when authorization depends, in part, on user attributes. With *attribute-based access control* (ABAC), an authorization decision is made based on a set of attributes that is associated with each user. A role that a user occupies, however, can be seen as an attribute of that user. So the authorization that RBAC specifies can be enforced using ABAC. In fact, RBAC and ABAC are theoretically equivalent: the privilege to occupy a role can be seen as defining an attribute,

and each possible subset of attributes can be seen as defining a role. There are an exponential number of subsets for a given set of attributes. So having a smaller number of attributes significantly reduces the burden on system administrators. Since a role can replace a subset of the attributes, RBAC and roles offer a way to reduce that burden.

Notes and Reading for Chapter 3

Security policies imposed by an institution—the *sine qua non* for a mandatory access control policy—have a long history. Long before the advent of computers, armies were concerned with protecting secrets in order to preserve technological superiority and/or to surprise an adversary. The Byzantine Empire kept secret the recipe for making Greek Fire, a napalm-like burning mixture that their ships in combat would spray or catapult to wreak havoc on an adversary’s wooden-hulled ships. And Sun Tzu’s *The Art of War* [59], written in the fifth century B.C.E., advocates a secrecy policy when it opines that “the formation and procedure used by the military should not be divulged beforehand”. Outside of the military, medieval guilds imposed secrecy policies that prevented nonmembers from learning the skills needed to enter certain occupations. We also find businesses throughout history maintaining trade secrets, either to protect an existing competitive advantage or to secure a first-mover advantage.

The Orange Book [16, §3.1.1.3] popularized the term “mandatory access control” and equates it with a class of authorization policies where access to objects by subjects is based on labels that form a lattice. Those labels were required to be “a combination of hierarchical classification levels and non-hierarchical categories” [16, §3.1.1.4], which was describing the document classification scheme being used by the U.S. government. Quist [51, chapter 2], drawing heavily from an unpublished manuscript by Patterson [50], chronicles the precursors and evolution of U.S. government document-classification schemes. The current scheme is detailed in Executive Order 13526 [47], signed by President Barack Obama in December 2009. It is the most recent in a series of Executive Orders, starting with Executive Order 8381 [52], signed by President Roosevelt in March 1940 [51, chapter 3].

U.S. document-classification schemes were derived from a British scheme circa 1917. But Britain’s document-classification schemes date back to the late 19th century. Prior to the Crimean War (1853–1856), the British War Office had been marking documents that should be kept confidential, and by 1894 British Army regulations were distinguishing between markings “Secret” and “Confidential” that each imposed specific rules for handling and disclosure. An early version of “need to know” appears in an 1868 publication of British Army regulations:

Access to official records is only permitted to those who are entrusted with the duties of the office or department to which they belong ...

Peacetime classification of military secrets in Britain commenced with an 1866

report on mines and torpedoes—new technologies that would be less effective if the details became known by an adversary.

The Orange Book was produced by a process that began when the U.S. Department of Defense (DoD) began to contemplate storing classified information on time-sharing systems.²⁰ An early and widely-cited discussion of the technical issues to be addressed appears in the 1970 report [61] from a committee chaired by Ware and convened in Fall 1967 under the auspices of the Defense Science Board. Computers back then were expensive and, thus, had to be shared. So for storing classified documents, DoD required

- a system that could support multiple, concurrent users having different clearances and accessing objects that had different classifications, and
- an assurance argument to establish that the system’s access controls could not be circumvented either by *bona fide* users or by outsiders.

The Adept-50 time-sharing system [62] was an early and notable attempt to address these DoD needs. That system, which was operational by 1969, was developed at System Development Corporation (SDC) under an ARPA²¹ contract. The authorization policy that Adept-50 enforced was based on a *high-water mark* security label that was maintained for each process by the system. Adept-50 also supported a **change** command for reducing the security label associated with an object. Unfortunately, a program could leak secret information by executing **change**—read the secret information from one file, write it to another file (causing the file’s label to be set appropriately), and then invoke **change** to decrease the label on that output file so the secret information could be read by any process.

Frustrated by a lack of progress in creating secure time-sharing systems, USAF Major Roger R. Schell commissioned a study. Chaired by Edward Glaser of Case Western Reserve University, this study assembled the established experts in order to reach a consensus on a research and development plan. The final report [2], published in 1972, is today known as the Anderson Report, named after the committee member who managed the study and did much of the writing. The report advocates (among other things) that a small security kernel be the only software involved in enforcing a system’s authorization policies—an architecture that Schell had long been advocating. This recommendation was a rejection of the prevailing view at the time, which was that extant time-sharing systems could be made sufficiently secure simply by adding code to perform further checking. Schell had contended that only with a small security kernel could a thorough analysis and testing be feasible, thereby providing assurance about the enforcement mechanism. An assurance argument for the entire time-sharing system then would be obtained by combining the

²⁰See Mackenzie and Pottinger [43] for the context and history of this effort.

²¹The Advanced Research Projects Agency (ARPA) was created in 1958 to fund research in support of DoD to help avoid technological surprises like the Soviet Union’s October 1957 launch of Sputnik 1, the first artificial Earth satellite. The name Defense Advanced Research Projects Agency (DARPA) for this organization has been in use since March 1996.

assurance for security kernel with a proof that the authorization policy implies the intended security properties. The Anderson Report also brought the term Trojan horse to cybersecurity; it is discussed in appendix I “Security threats and penetration techniques” where it is attributed to Dan Edwards, one of two NSA representatives to committee.

Two research groups were subsequently funded to develop formal models, define security policies, and prove that information labeled as classified could not be read by users lacking suitable clearances.²² Bell and LaPadula, working at MITRE, published their proposal in 1973 [6, 7], though Shell and the members of the Anderson Report committee are likely to have seen write-ups of this work in earlier unpublished MITRE reports. This Bell and LaPadula proposal became the basis for much of DoD’s computer security work over the next decade; the BLP rules in §3.1.1 are from there. The other research group, working independently at Case Western University, developed essentially the same restrictions (couched in terms of repositories and agents rather than files and processes) and published their work [60] a few months later.

Trusted subjects were proposed in Bell and LaPadula [8] when their original scheme [6, 7] was found to be deficient for enforcing security in Multics [49].²³ Bell and LaPadula [8, section IV] also identified some limitations with the definition of security given in their original scheme [6, 7], which had ignored various covert channels, information corruption, and denial of service.

Integrity is ignored in Bell and LaPadula [6, 7]. That omission was problematic once DoD grew concerned about preventing unauthorized changes to targeting data that was going to be loaded onto U.S. missiles [56]. An extension to handle integrity was proposed by Biba [9]. This scheme introduced MLI labels, an additional set of partially ordered labels forming a lattice that was described as being the dual of the lattice used for confidentiality in Bell and LaPadula [6, 7]. A close look, however, reveals (as discussed on page 112 in connection with the BLP policy) that integrity is being enforced by Biba [9] using the same partial order that Bell and LaPadula [6, 7] uses to enforce confidentiality—it prohibits the propagation of content from containers with high labels to containers with lower labels.

Biba [9] also describes a scheme that adapts the Adept-50 [62] high-water mark policy in order to relax the requirement that an MLI label for a subject or file must be fixed throughout execution. Using Biba’s low-water mark integrity scheme, the label for a subject or file during an execution reflects the lowest integrity content previously transferred there. Twenty-five years later, the low-water mark scheme was found to be ideally suited for protecting system integrity against malware inadvertently downloaded from the internet, and the scheme was implemented in LOMAC [27], a UNIX variant. In LOMAC, every file from the initial operating system installation is given a label signifying highest integrity,

²²Landwehr [37] surveys the various formal models for computer security from that time period.

²³Walter et al. [60] at Case Western University is credited with developing the access policy in [8] for Multics tree-structured directories. Directories are files having specific semantics and thus warranted special treatment.

objects a user creates get labels indicating mid-level integrity, and content from hardware devices that provide external access (e.g, keyboard, mouse, and network interfaces) has a label for the lowest level of integrity. The BLP rules then ensure that content downloaded from the Internet cannot become part of the operating system.

Besides a suitable security policy, the vision of the Anderson Report required that a small security kernel be used for enforcement. To establish the feasibility of that, DoD by the late 1970's had funded four prototyping efforts.

- Multics developers added the Access Isolation Mechanism (AIM) [18], implemented other security functionality [63], and ran a vulnerability analysis [33].
- KVM/370 [29, 30] was developed by System Development Corporation (SDC) as a retrofit to IBM's virtual machine operating system VM/370.
- Kernelized Secure Operating System (KSOS) [44] was implemented at Ford Aerospace as a security kernel that executed a UNIX emulator.
- The Secure Communications Processor (Scomp) and its operating system STOP (Secure Trusted Operating System) [25] were built at Honeywell to serve as a secure front-end processor for Multics.

It was understood that ultimately a range of available commercial off-the-shelf (COTS) computer systems would be needed for use by DoD. The National Computer Security Center (NCSC) within NSA was created to help achieve that goal.

The NCSC then drafted the Orange Book to specify the requirements that commercially developed systems would have to satisfy for hosting classified content, and it also created a process for rating how well a system complied with those requirements.²⁴ The lowest rating was D and the highest was A1, with the rating based on security functionality, system structure, and the assurance argument that was being provided. Discretionary access control was required for C and above; authorization that enforced multilevel security (i.e., labels and the BLP rules) was required for B and above; and assurance arguments were required for an A1 rating. Multics was submitted for evaluation and received a B2 rating; Scomp received an A1 rating.

COTS software developers had good reasons not to invest in building systems that would receive a B or higher Orange Book rating, though. First, was the market size. The U.S. government and defense industry did not form a large segment of the market for COTS software, only a small part of that segment actually needed systems rated B or higher, and export restrictions on certain technologies at that time made it difficult to sell such systems abroad. Second, the rating process increased time-to-market. Third, customers in the private

²⁴Lipner [41] gives a history, writing with the perspective of a participant in the discussions leading to publication of the Orange Book, a contributor to the debate that followed, and a developer of a system that would comply with its directives.

sector did not perceive a need for multilevel security or for most of the other security functionality required by a B or higher rating.

With hopes of entering the markets for computers that handle classified information, in 1981 Digital Equipment Corporation (DEC) nevertheless began developing VAX/SVS (Secure Virtual System) [34, 35, 39], with plans to obtain an A1-rating. VAX/SVS was implemented as a virtual machine manager and, therefore, could run the VMS and Ultrix-32 operating systems already in use on DEC's VAX computers, easing the migration path for a large base of customers. By 1989, a version of VAX/SVS was being field-tested at government and defense industry sites. The system exhibited acceptable performance and allowed multilevel secure tasks to be performed. But the limited prospects for domestic sales and difficulties in arranging sales abroad (among other things), prompted DEC to cancel the project a year later.

Besides the dearth of COTS systems rated B or above, questions were also being raised about the utility of using multilevel security for DoD applications. A widely reported attempt by Landwehr et al. [38] at the Naval Research Laboratory to implement security for military-message systems reported various ways in which multilevel security was not well suited for this and other applications. The difficulties included (i) a need for trusted subjects because certain write-downs had to be performed but there was no way to impose application-specific constraints on those trusted subjects and (ii) a lack of support for objects having components with different security levels. Further concerns were subsequently raised in McLean [45, 46] with the infamous system *Z*, which automatically declassified objects to make all reads and writes appear legal. The same "Basic Security Theorem" proved in Bell and LaPadula [8] for Multics could be proved for system *Z*. That troubling observation led to much debate [5] about whether proving that a system satisfies the "Basic Security Theorem" actually provides assurance.

The potential vulnerabilities introduced by trusted subjects led Boebert and Kain to devise *type enforcement* [10, 11] (TE) for supporting MAC policies in LOCK [55, 48] (Logical Coprocessing Kernel).²⁵ TE was a radical (and politically perilous, given the beliefs of the intended customer for the effort) break with the Orange Book's dogma, because TE did not use labels or the BLP rules. To avoid provocation, early publications about TE and LOCK focus on enforcing integrity policies and implementing assured pipelines; there is little discussion that TE does not require introducing trusted subjects or that TE enforces a broader class of MAC policies than are described in the Orange Book.

LOCK was commercialized as Sidewinder, a successful firewall and VPN gateway that was developed and sold by Secure Computing Corporation (SCC), a 1989 spinoff from Honeywell's Secure Computing Technology Center (the developer of LOCK). TE also has served as a starting point for the authorization

²⁵If LOCK was the sole path to reach a network then a time-sharing system or a personal computer no longer had to be trusted to encrypt the messages it sends. Supply chain attacks that compromise personal computer hardware or software were among the concerns that LOCK was intended to address.

implemented by other operating systems that support MAC policies. One example is DTE Unix [3], which implements *domain and type enforcement* (DTE), an extension of TE where the types for files and certain other objects are derived automatically from the hierarchies defined by UNIX directories. A Linux implementation of DTE is described in Hallyn and Karns [31]. Extensions to support a dynamically configurable variant of DTE—*dynamic domain and type enforcement* (DDTE)—are proposed by Tidswell and Potter [58].

In addition to all of the concerns raised about using Orange Book systems for DoD systems, there were questions about whether the needs of commercial institutions would be well served. Lipner [40] in 1982 claims that a security lattice model may be applicable in these settings, based on discussions with a senior EDP auditor. But a 1987 paper [13] by David Clark (a computer scientist) and David Wilson (an accountant) forcefully argues the opposite. The Orange Book’s MAC policies are shown to be ill well suited for commercial institutions because outside of the military the primary concern is with different forms of malfeasance (e.g., fraud and error, rather than disclosure).

The defenses that Clark and Wilson [13] describes—audit and the use of well-formed transactions—derive from classic accounting controls, which date back to the beginnings of commerce and taxation. The Egyptians and Babylonians employed audits to keep track of warehouse contents, so they could reconcile inventory with deposits and withdrawals. By the 1400’s, Venice was an important commercial center, and double-entry bookkeeping was proving an effective method for recording business transactions. The ledger for 1299–1300 of the Florentine merchant Amantino Manucci is the earliest documented use of the approach [17]. The first full description of double-entry bookkeeping was apparently printed in a volume by Luca Pacioli that was published in November 1494.²⁶

So Clark and Wilson [13] establish that that multilevel security is just one kind of MAC policy, and a new (or significantly revised) Orange Book would be needed to support commerce. Further corroboration comes with the publication of Brewer and Nash [12], which introduces computer security researchers to yet another useful class of MAC policies for commercial institutions: Chinese Wall policies. These policies are not only orthogonal to the MAC policies in the Orange book but they differ from the commercial policies of Clark and Wilson [13]. Chinese Wall²⁷ policies originated in U.S. investment-industry regulators following the 1929 stock market crash that began the Great Depression. A so-called Chinese Wall was intended to give the public assurance that a brokerage was

²⁶Pacioli’s description of so-called Venetian bookkeeping appears in volume I, chapter 9, part 11 of his mathematical encyclopedia *Summa de arithmetica, geometria, proportioni et proportionalità*, which also was the first printed book to describe Hindu-Arabic arithmetic and algebra. The publication a decade later of the pamphlet *La scuola perfetta dei mercanti* excerpting the material about double-entry bookkeeping doubtless was important for facilitating widespread adoption of the approach. This history and the later impacts of double-entry bookkeeping are given in Gleeson-White [28].

²⁷The term “Chinese Wall” is attributed to U.S. President Franklin D. Roosevelt who, shortly after he was elected in 1933, used the phrase “Chinese wall of silence” to describe the isolation sought for different principals within a single financial institution. [32, page 81].

being prevented from profiting at the expense of its customers.

Separation of duty had enjoyed a long history in governance, where it is called “separation of powers”. The U.S. Constitution (drafted in 1789), for example, stipulates a tripartite structure comprising a legislative branch (to make laws), an executive branch (to enforce laws), and a judiciary branch (to interpret laws). Tripartite governing structures, which is attributed to the Age of Enlightenment’s political philosopher Baron de Montesquieu [15], had earlier been adopted by the Dutch and the English. Still further back, we see governments of the Roman Republic and the Greek city-states in antiquity employing separation of powers. Among the earliest discussions of separation of duty in connection with computer security is the “separation of privilege” principle discussed in Saltzer and Schroeder [53], which credits a 1973 conversation with Roger Needham.

Early operating systems did not support RBAC or roles per se. However, early operating systems did support groups that (like individuals) could be assigned privileges. Some operating systems only supported a predefined set of groups; other operating systems allowed groups to be created and populated as needed, either by system administrators or by users. By the mid 1980’s, Landwehr et al. [38] for their military-message systems security policy had proposed something that foreshadows RBAC, saying:

Role—the job a user is performing, such as downgrader, releaser, distributor, and so on. A user is always associated with at least one role at any instant, and the user can change roles during a session. To act in a given role, the user must be authorized for it. Some roles may be assumed by only one user at a time (e.g., distributor). With each role comes the ability to perform certain operations.

Ferraiolo and Kuhn were the first to argue in favor of using roles in general as the basis for mandatory access control policies. Concerns being voiced about the policies discussed in the Orange Book had led Ferraiolo et al. to conduct a survey²⁸ [20] of the security needed for civilian government agencies and commercial enterprises. The survey responses indicated that these non-military institutions required a means to associate privileges with roles (rather than associating privileges with individuals), and doing that association was not easily implemented using existing systems. A form of role-based access control thus seemed like it would better serve those surveyed than the Orange book’s MAC policies or than Clark and Wilson’s transformation procedures.

The original proposal by Ferraiolo and Kuhn [21] focused on roles and inheritance [21]. It generalized *named protection domains*, which had been described in Baldwin [4] as a basis for authorization in ANSI SQL databases. Constraints were added after roles proved inadequate for formalizing certain separation of duty policies [19]. The presentation in §3.3 is derived from Sandhu et al. [54], an early and influential effort to structure role-based access control in terms of the

²⁸The study covered 28 organizations, including 17 federal agencies, 10 corporations, and 1 state agency.

simpler models: $RBAC_0$ supports roles, sessions, and privileges; $RBAC_1$ adds role inheritance to $RBAC_0$; $RBAC_2$ adds constraints to $RBAC_0$; and $RBAC_3$ combines $RBAC_1$ and $RBAC_2$. ANSI standard INCITS 359-2004 (approved February 2004) for role based access control is based on $RBAC_3$ (see [22]). The standard is implemented in operating systems (e.g., Microsoft's active directory) and in database management systems (Microsoft SQL server, Oracle DBMS, SAP R/3). See Franqueira and Wieringa [26] for a discussions of settings well suited for RBAC. A proposal to combine ABAC and RBAC is made in Kuhn et al. [36].

But given this diverse set of credible MAC policies, an operating system that supported only a single MAC policy would not have broad appeal and, therefore, commercial developers of operating systems did not provide support for MAC policies. Within DoD, interest nevertheless remained high in having operating systems that could meet the government's needs. So DoD funded research in this area. One thrust—part of the Synergy research program at NSA—was to develop an architecture that allowed various different MAC policies to be enforced within the same operating system, since a large user community could then be served.

The idea of a *security server* in a microkernel offered a promising solution. In a microkernel, the interface to all abstractions is through message-passing to ports. By having a separate security server for each port, a policy could be enforced on all requests to the associated abstraction, and different policies could be enforced on different abstractions. A cache in the microkernel then lowered the overhead by allowing repeated invocations of the security server to be eliminated for repeated requests. SCC was supported to create a first implementation of this approach. The result was DTMach [23], a microkernel that extended Mach [1]. The approach was then explored further with a project at SCC that developed the DTOS microkernel architecture and also explored the attendant assurance questions [14]. The DTOS security architecture, with the help of University of Utah researchers, was then migrated to the Fluke microkernel environment [24], resulting in the Flask architecture [57].

The next step was an implementation of the Flask architecture in a mainstream operating system. By the late 1990's, the adoption of Linux, an open source operating system, was growing. That made Linux an ideal target for Flask, and SELinux [42] adds to the Linux kernel a security server that supported multi-level security (BLP), type enforcement (TE), identity-based access control, and dynamic role-based access control (RBAC). MAC policies were finally being supported in a mainstream operating system that had a growing set of applications being curated by a large community. Subsequent SELinux releases provided even more flexibility by allowing loadable kernel modules to serve as Flask security servers.

Bibliography

- [1] Michael J. Accetta, Robert V. Baron, William J. Bolosky, David B. Golub, Richard F. Rashid, Avadis Tevanian, and Michael Young. Mach: A new kernel foundation for UNIX development. In *Proceedings of the USENIX Summer Conference*, pages 93–113. USENIX Association, June 1986.
- [2] James P. Anderson. Computer security technology planning study. Technical Report ESD-TR-73-51, Electronic Systems Division (AFSC), Hanscom Field, Bedford, MA, October 1972.
- [3] Lee Badger, Daniel F. Sterne, David L. Sherman, Kenneth M. Walker, and Sheila A. Haghghat. Practical domain and type enforcement for UNIX. In *Proceedings of 1995 IEEE Symposium on Security and Privacy*, pages 66–77. IEEE Computer Society Press, 1995.
- [4] Robert W. Baldwin. Naming and grouping privileges to simplify security management in large databases. In *Proceedings of 1990 IEEE Symposium on Security and Privacy*, pages 116–132. IEEE Computer Society Press, May 1990.
- [5] D. Elliott Bell. Concerning modeling of computer security. In *Proceedings of 1988 IEEE Symposium on Security and Privacy*, pages 8–13. IEEE Computer Society Press, 1988.
- [6] D. Elliott Bell and Leonard J. LaPadula. Secure computer systems: Mathematical foundations. Technical Report ESD-TR-73-278, Volume I, Electronic Systems Division (AFSC), Hanscom Field, Bedford, MA, November 1973.
- [7] D. Elliott Bell and Leonard J. LaPadula. Secure computer systems: A mathematical model. Technical Report ESD-TR-73-278, Volume II, Electronic Systems Division (AFSC), Hanscom Field, Bedford, MA, November 1973.
- [8] D. Elliott Bell and Leonard J. LaPadula. Secure computer systems: Unified exposition and MULTICS interpretation. Technical Report EDS-TR-75-306, Electronic Systems Division (AFSC), March 1976.
- [9] K. J. Biba. Integrity consideration for secure computer systems. Technical Report MTR-3153, MITRE Corporation, Bedford, MA, June 1975.
- [10] W. E. Boebert and R. Y. Kain. A practical alternative to hierarchical integrity policies. In *Proceedings of the 8th National Computer Security Conference*, pages 18–27. U.S. Government Printing Office, October 1985.
- [11] William E. Boebert and Richard Y. Kain. A further note on the confinement problem. In *1996 International Carnahan Conference on Security Technology*, pages 198–202, 1996.

- [12] David F. C. Brewer and Michael J. Nash. The Chinese Wall security policy. In *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, pages 206–214. IEEE Computer Society Press, May 1989.
- [13] David D. Clark and David R. Wilson. A comparison of commercial and military computer security policies. In *Proceedings of 1987 IEEE Symposium on Security and Privacy*, pages 184–194. IEEE Computer Society Press, 1987.
- [14] Secure Computing Corporation. DTOS lessons learned, December 1993. Contract No. MDA904-93-C-4209, CRDL Sequence No. A008, Part number 87-0902025A006.
- [15] Baron de Montesquieu. De l’esprit des lois, 1748. Republished as *Montesquieu: The Spirit of Laws*, Cambridge Texts in the History of Political Thought, Cambridge University Press, 1989.
- [16] Department of Defense. *Department of Defense Trusted Computer System Evaluation Criteria*. DoD 5200.28-STD, Supercedes CSC-STD-001-83 dated 15 August 1984, Library Number S225,711.
- [17] J. Richard Edwards. A history of double-entry bookkeeping. *The Accounting Historians Journal*, 16(1):59–91, 1989.
- [18] B2 Security Evaluation. <https://multicians.org/b2.html>.
- [19] David F. Ferraiolo, Janet A. Cugini, and D. Richard Kuhn. Role-based access control (RBAC): Features and motivations. In *Proceedings of 11th Annual Computer Security Applications Conference*, pages 241–248. IEEE Computer Society Press, December 1995.
- [20] David F. Ferraiolo, Dennis M. Gilbert, and Nickilyn Lynch. Assessing federal and commercial information security needs. Technical Report NISTIR 4976, National Institute of Standards and Technology, Computer Systems Laboratory, Gaithersburg, Maryland, November 1992.
- [21] David F. Ferraiolo and D. Richard Kuhn. Role-based access controls. In *Proceedings of 15th National Computer Security Conference*, pages 554–593. National Institute of Standards and Technology, National Computer Security Center, October 1992.
- [22] David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information System Security*, 4(3):224–274, August 2001.
- [23] Todd Fine and Spencer E. Minear. Assuring distributed trusted Mach. In *Proceedings of the 1993 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 206–217. IEEE Computer Society, May 1993.

- [24] Bryan Ford, Godmar Back, Greg Benson, Jay Lepreau, Albert Lin, and Olin Shivers. The Flux OSKit: A substrate for kernel and language research. In Michel Banâtre, Henry M. Levy, and William M. Waite, editors, *Proceedings of the Sixteenth ACM Symposium on Operating System Principles*, SOSP '97, pages 38–51. ACM, October 1997.
- [25] L. J. Fraim. Scomp: A solution to the multilevel security problem. *IEEE Computer*, 16(7):26–34, 1983.
- [26] Virginia N. L. Franqueira and Roel J. Wieringa. Role-based access control in retrospect. *Computer*, 45(6):81–88, 2012.
- [27] Timothy Fraser. LOMAC: Low water-mark integrity protection for COTS environments. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 230–245. IEEE Computer Society, May 2000.
- [28] Jane Gleeson-White. *Double Entry*. W. W. Norton & Company, October 2013.
- [29] Barry D. Gold, Richard R. Linde, and P. F. Cudney. KVM/370 in retrospect. In *Proceedings of the 1984 IEEE Symposium on Security and Privacy*, pages 13–23. IEEE Computer Society, May 1984.
- [30] Barry D. Gold, Richard R. Linde, Marv Schaefer, and John F. Scheid. VM/370 security retrofit program. In James S. Ketchel, Harvey Z. Kriloff, H. Blair Burner, Patricia E. Crockett, Robert G. Herriot, George B. Houston, and Cathy S. Kitto, editors, *Proceedings of the 1977 Annual Conference, ACM '77*, pages 411–418. ACM, October 1977.
- [31] Serge E. Hallyn and Phil Kearns. Domain and type enforcement for Linux. In *4th Annual Linux Showcase & Conference 2000*. USENIX Association, October 2000.
- [32] Anthony Hilton. *City within a State: A Portrait of Britain's Financial World*. I. B. Tauris & Company Limited, 1987.
- [33] Paul A. Karger and Roger R. Schell. Multics security evaluation: Vulnerability analysis. Technical Report ESD-TR-74-193, Vol. II, Electronic Systems Division (AFSC), Hanscom AFB, MA, June 1974.
- [34] Paul A. Karger, Mary Ellen Zurko, Douglas W. Bonin, Andrew H. Mason, and Clifford E. Kahn. A VMM security kernel for the VAX architecture. In *Proceedings of the 1990 IEEE Symposium on Security and Privacy*, pages 2–19. IEEE Computer Society Press, May 1990.
- [35] Paul A. Karger, Mary Ellen Zurko, Douglas W. Bonin, Andrew H. Mason, and Clifford E. Kahn. A retrospective on the VAX VMM security kernel. *IEEE Transactions on Software Engineering*, 17(11):1147–1165, 1991.

- [36] D. Richard Kuhn, Edward J. Coyne, and Timothy R. Weil. Adding attributes to role-based access control. *Computer*, 43(6):79–81, 2010.
- [37] Carl E. Landwehr. Formal models for computer security. *ACM Computing Surveys*, 13(3):247–278, September 1981.
- [38] Carl E. Landwehr, Constance L. Heitmeyer, and John D. Mclean. A security model for military message systems. *ACM Transactions on Computer Systems*, 2(3):198–222, August 1984.
- [39] Steve Lipner, Trent Jaeger, and Mary Ellen Zurko. Lessons from VAX/SVS for high-assurance VM systems. *IEEE Security and Privacy*, 10(6):26–35, 2012.
- [40] Steven B. Lipner. Non-discretionary controls for commercial applications. In *Proceedings of 1982 IEEE Symposium on Security and Privacy*, pages 2–10. IEEE Computer Society Press, 1982.
- [41] Steven B. Lipner. The birth and death of the Orange Book. *IEEE Annals of the History of Computing*, 37(2):19–31, April–June 2015.
- [42] Peter A. Loscocco and Stephen Smalley. Integrating flexible support for security policies into the Linux operating system. In Clem Cole, editor, *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference*, pages 29–42. USENIX, June 2001.
- [43] Donald Mackenzie and Garrel Pottinger. Mathematics, technology, and trust: Formal verification, computer security, and the U.S. military. *IEEE Annals of the History of Computing*, 19(3):41–59, July 1997.
- [44] E. J. McCauley and P. J. Drongowski. KSOS—The design of a secure operating system. In *Proceedings of the National Computer Conference*, NCC, pages 345–353. IEEE, 1979.
- [45] John McLean. A comment on the ‘Basic Security Theorem’ of Bell and LaPadula. *Information Processing Letters*, 20(2):67–70, February 1985.
- [46] John McLean. Reasoning about security models. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, pages 123–133. IEEE Computer Society Press, May 1987.
- [47] Barack Obama. Classified national security information. Executive Order EO 13526, The White House, December 2009. <https://www.fas.org/irp/offdocs/eo/eo-13526.htm>.
- [48] Richard O’Brian and Clyde Rogers. Developing applications in LOCK. In *Proceedings of 14th National Computer Security Conference*, pages 147–156. National Institute of Standards and Technology, National Computer Security Center, October 1991.

- [49] Elliott I. Organick. *The Multics System: An Examination of its Structure*. MIT Press, 1972.
- [50] Andrew Patterson Jr. “CONFIDENTIAL” – The beginning of defense-information marking. Unpublished manuscript. Sterling Chemistry Laboratory, Yale University, April 1980.
- [51] Arvin S. Quist. Security Classification of Information, Volume 1. Introduction, History, and Adverse Impacts. Technical Report ORCA-12, Oak Ridge Classification Associates, LLC, September 2002. <http://www.fas.org/sgp/library/quist/index.html>.
- [52] Franklin D. Roosevelt. Defining certain vital military and naval installations and equipment. Executive Order EO 8381, The White House, March 1940. <https://www.fas.org/irp/offdocs/eo/eo-8381.htm>.
- [53] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, March 1975.
- [54] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E Youman. Role-based access control models. *Computer*, 29(2):38–47, Feb 1996.
- [55] O. Sami Saydjari, J. M. Beckman, and J. R. Leaman. LOCK Trak: Navigating uncharted space. In *Proceedings of the 1989 IEEE Symposium on Security and Privacy*,, pages 167–175. IEEE Computer Society, May 1989.
- [56] Roger R. Schell. Oral history interview with Roger R. Schell. Charles Babbage Institute. Retrieved from the University of Minnesota Digital Conservancy, May 2012. <https://hdl.handle.net/11299/133439>.
- [57] Ray Spencer, Stephen Smalley, Peter A. Loscocco, Mike Hibler, Dave G. Andersen, and Jay Lepreau. The Flask security architecture: System support for diverse security policies. In G. Winfield Treese, editor, *Proceedings of the 8th USENIX Security Symposium*. USENIX Association, August 1999.
- [58] Jonathon Tidswell and John Potter. An approach to dynamic domain and type enforcement. In Vijay Varadharajan, Josef Pieprzyk, and Yi Mu, editors, *Information Security and Privacy, Second Australasian Conference, ACISP’97*, volume 1270 of *Lecture Notes in Computer Science*, pages 26–37. Springer, July 1997.
- [59] Sun Tzu. *The Art of War*. Courier Dover Publications, 2013.
- [60] K. G. Walter, W. F. Ogden, W. C. Rounds, F. T. Bradshaw, S. R. Ames, and D. G. Shumway. Primitive models for computer security. Interim Technical Report ESD-TR-4-117, Case Western Reserve University, 1974. NTIS AD-778 467.

- [61] Willis H. Ware. Security control for computer systems: Defense Science Board Task Force on Computer Security. Technical Report R-609-1, Rand Corporation, Santa Monica, CA, February 1970.
- [62] C. Weissman. Security controls in the ADEPT-50 time-sharing system. In *Proceedings of the 1969 Fall Joint Computer Conference*, AFIPS Conference Proceedings, pages 119–133. AFIPS Press, 1969.
- [63] J. Whitmore, A. Bensoussan, P. Green, D. Hunt, A. Kobziar, and J. Stern. Design for Multics security enhancements. Technical Report ESD-TR-74-176, Honeywell Information Systems, Cambridge, MA, December 1973.